**Ana Sofia Gomes**

Mestre em Engenharia Informática

# Transactions in Dynamic Reactive Environments

Dissertação para obtenção do Grau de Doutora em Informática

Orientador : José Júlio Alferes, Prof. Catedrático, Universidade Nova de Lisboa

Júri:

| | |
|---|---|
| Presidente: | Prof. Doutor Nuno Correia |
| Arguentes: | Prof. Doutor Wolfgang May |
| | Prof. Doutor Werner Nutt |
| Vogais: | Prof. Doutor João Pavão Martins |
| | Prof. Doutor José Júlio Alferes |
| | Prof. Doutora Carla Ferreira |

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

**Janeiro, 2015**

**Transactions in Dynamic Reactive Environments**

*To my parents*

# Acknowledgements

First and foremost, I need to sincerely thank my supervisor, José Júlio Alferes, for his incredible support and expertise. Even as Head of the department or as faculty Vice-Dean, he always managed to be available to help and advise me, or just to calm me down whenever I reached (one of the too many) dead ends. I really appreciated all our discussions either scientific, political, or just on random subjects. Thank you for giving me confidence to grow, for teaching me how to be a researcher, and for magically providing for all my work needs and conditions. It was a pleasure to share this journey with you.

I am grateful to the members of my Thesis Advisory Committee, Wolfgang May and Carla Ferreira, for their useful feedback and comments throughout my studies.

I had a very fruitful stay in Stony Brook University, for which I thank Michael Kifer, Paul Fodor, Martín Rezk, Reza Basseda, and the rest of the Stony Brook workgroup, for their hospitality and the valuable discussions and comments on a preliminary version of this work. A special thanks goes to Polina Kuznetsova, Ritwik Bose, Jiemin Zeng, Reza Basseda, Paul Fodor and Martín Rezk, for making my stay so enjoyable.

It was a pleasure to collaborate with João Leite, Marco Alberti, Martin Slota, Matthias Knorr, Ricardo Gonçalves, and Terrance Swift. I also really enjoyed the seminars and discussion of the KRR Research Group at CENTRIA, and thank all its members, especially to Alfredo Gabaldon, Carlos Damásio, João Leite, José Alferes, for sharing their experience, knowledge and criticisms with us.

It was a pleasure to share the office throughout the years with the fellow and former students: Alexandre Pinto, Matthias Knorr, Marco Correia, Martin Slota, Martin Aleksandrov, Peter Skocovsky, João Moura, João Martins, Ricardo Silva, Sinan Elgimez, Tobias Kaminski, and many others. Also from the "other" office, and steady companions of our regular Open Houses: Ari Saptawijaya, Emmanuelle Dietz, Han the Ahn, Hanna Thingamajig and Sergejs Pugacs.

I really enjoyed sharing the tales and troubles of a PhD student during lunch with Bernardo Toninho, Claúdio Gomes, Filipa Peleja, João Martins, Jorge Costa, Luísa Lourenço, Mário Pires, Miguel Domingues, Miguel Lourenço, and Ricardo Silva.

It was also a pleasure to be a part of the IEEE family, and learning so much from this

experience. A special thanks goes to Jorge Soares, Diogo Mónica, Rui Costa, Femia Ar, José Pedro Silva, Miguel Marques, Rui Cruz, Pedro Afonso, Ana Madureira, and Nuno Borges de Carvalho.

I also have to thank my dearest friends (whom I don't need to enumerate) for not letting me take myself too serious, reminding me that life exists outside academia (that happiness is better experienced in Arrábida or in Andorra), and helping me not to procrastinate alone.

Obviously, I am greatly indebted to my parents and my family in general, for their continuous support and encouragement throughout my whole my life. It would be impossible to put in words everything they have done for me. Finally, but most important, I thank Bruno for his love, patience, understanding, and being able to put up with me all of these years.

This work was supported by the by the FCT grant SFRH/BD/64038/2009, and conceived within project ERRO (PTDC/EIA-CCO/121823/2010).

# Abstract

Most of today's systems, especially when related to the Web or to multi-agent systems, are not standalone or independent, but are part of a greater ecosystem, where they need to interact with other entities, react to complex changes in the environment, and act both over its own knowledge base and on the external environment itself. Moreover, these systems are clearly not static, but are constantly evolving due to the execution of self updates or external actions. Whenever actions and updates are possible, the need to ensure properties regarding the outcome of performing such actions emerges. Originally purposed in the context of databases, transactions solve this problem by guaranteeing atomicity, consistency, isolation and durability of a special set of actions. However, current transaction solutions fail to guarantee such properties in dynamic environments, since they cannot combine transaction execution with reactive features, or with the execution of actions over domains that the system does not completely control (thus making rolling back a non-viable proposition). In this thesis, we investigate what and how transaction properties can be ensured over these dynamic environments. To achieve this goal, we provide logic-based solutions, based on *Transaction Logic*, to precisely model and execute transactions in such environments, and where knowledge bases can be defined by arbitrary logic theories.

**Keywords:** Transactions, Knowledge Representation, Transaction Logic, External Actions, Compensations, Reverse Actions, Reactivity, Complex Events

# Resumo

Actualmente, a maioria dos sistemas de informação, especialmente quando relacionados com a Internet ou com sistemas multi-agentes, não pode ser vista como *standalone* ou independente, ignorando que faz parte de todo um ecossistema, no qual têm que interagir com outras entidades (ou agentes), reagir e detectar mudanças complexas no ambiente, e agir tanto sobre a a sua base de conhecimento interna, como no ambiente externo onde estão inseridos. Por tudo isto, sistemas com estas características não podem ser considerados estáticos, mas em constante evolução, graças à actualização contínua que estes fazem sobre o seu próprio conhecimento, mas também devido à execução de acções externas. A partir do momento em que é possível fazer acções e actualizações num determinado sistema, surge também a necessidade de assegurar propriedades relativas ao resultado de executar estas acções. Originalmente propostas no contexto das bases de dados, as transacções resolvem este problema garantindo que um conjunto de acções é executado atomicamente, isoladamente, consistentemente e de forma durável.

No entanto, a maioria das soluções existentes que assegura a execução transaccional das acções, não é capaz de garantir estas propriedades nos sistemas altamente dinâmicos referidos anteriormente. Em particular, estas falham em garantir propriedades transaccionais sobre acções executadas em domínios externos, já que nestes não é possível controlar totalmente as acções efectuadas (e onde portanto o acto de *rollback* é impossível); mas também quando é necessário combinar transacções com reactividade. Nesta dissertação investigamos como, e que propriedades transaccionais são compatíveis com estes sistemas dinâmicos. Com este intuito, propomos três soluções baseadas na lógica *Transaction Logic* para modelar e executar transacções neste tipo de sistemas, e onde as bases de conhecimento podem ser definidas por várias teorias de acções baseadas em lógica.

**Palavras-chave:** Transacções, Representação do Conhecimento, Transaction Logic, Acções Externas, Compensações, Reactividade, Eventos Complexos

# Contents

# Part I

# Dynamic Environments and Transactional Properties

# 1

# Introduction

Artificial Intelligence (AI) aims to provide machines and software with the ability to solve tasks in an intelligent and autonomous way. A fundamental problem inherent to this goal is how to represent knowledge, either about the decisions concerning the task itself, or to describe the information about the domain involving the task.

Knowledge Representation and Reasoning (KRR) is a subfield of AI that studies how to express knowledge in a machine-interpretable form, and to draw conclusions over this knowledge in a automated way. With strong roots in mathematics, computer science, philosophy and economics, KRR solutions are typically based on logic constructors, and on inference procedures, the former to achieve a precise formalization of knowledge, and the latter to enable computer systems to reason about the formalized knowledge statements, and to act upon them accordingly.

In the realm of KRR, knowledge is precisely formalized using a *semantics*. In other words, a semantics establishes how knowledge statements are interpreted, what is the relation between statements, and what conclusions should be inferred in case of some knowledge conflict. Subsequently, a semantics intrinsically defines the representation language and the possible constructs to express the knowledge, but also the operators and techniques to interpret these knowledge statements. As a result, several different semantics exist to model different problems in AI, but also to model different facets of the same problem. For instance, for the same context problem, we may consider the knowledge to be static, but also to be dynamic. To address the latter, semantics must be able to represent and reason about *actions*, which can change the state of the world we are modeling. As such, these semantics are normally labelled as *state change* semantics, and focus mostly on formalizing what actions can be performed at each moment, and the possible effects that result from executing a set of actions in a given context.

While over the last 50 years, a multitude of semantics have been abundantly proposed in the literature of KRR, this sheer amount of semantics proposals also show that a "perfect" all-around semantics does not exist. In fact, the existing semantics in the literature can differ considerably on how knowledge statements can be expressed, and how they are interpreted. These choices determine the complexity of the procedures to reason about the knowledge, but also the domain of applicability of the semantics. Intuitively, a highly expressive semantics allows the construction of rich knowledge statements and eases the task of encoding the details and exceptions of a given representation problem. However, expressibility affects strongly the complexity class of algorithms that reason over such knowledge statements. The more expressive knowledge statements are, the higher is the complexity class of their algorithms.

A semantics is normally designed for a specific domain of interest, which determines how knowledge should be expressed and interpreted. For instance, describing knowledge over a medical domain demands highly expressive constructors (at the expense of a higher complexity), while describing knowledge in an alarm detection context requires instant computation at the cost of a lower expressivity.

Nonetheless, differences between semantics surpass the choice of the knowledge constructs available, as even with a fixed grammar, semantics can diverge on how knowledge statements are interpreted. For example, reasoning over an internal domain over which one has a complete control of the knowledge (as e.g. a traditional relational database) requires a different interpretation of negated information than when reasoning about open and incomplete knowledge like the Web [APP98; DAAW06]. These correspond to the concepts of closed world and open world assumption where, in closed world assumption, everything that is not possible to be derived by our knowledge statements is considered to be false, whereas in open world assumption, we assume that our knowledge description is probably incomplete, and thus lack of knowledge never implies the conclusion of negation.

However, even when the domain is pre-defined, picking only one right semantics can be a problem. This is e.g., the case of the Semantic Web, a collaborative movement to explicitly define the semantics of web contents and links, with the goal to allow manipulation and reasoning of web content by automated processes. In this context, the complexity of the web environment has triggered the appearance of several web standards, each providing a different semantics, like RDF [KCM04] or OWL [MPSPBFHHHRS+09]. The latter standard, based on a family of languages known as Description Logics [BCMNPS03], is further partitioned into three profiles, each providing a different trade-off between complexity and expressivity.

Independently of the semantics chosen, when dealing with dynamic environments (as for instance, the Web), it is often important to guarantee properties over the evolution of our knowledge, i.e., over the actions issued to change and update our knowledge base. For instance, one may want to guarantee that the knowledge base is always left consistent independently of when and what actions were executed; or in the case where we need to

issue a sequence of actions, to ensure that the failure of one of the actions in the sequence implies that the sequence of actions is considered to have never happened.

In this thesis we investigate how to ensure these properties of executed actions in abstract dynamic reactive domains.

## 1.1 Dynamic Reactive Environments

Most real-world scenarios are not standalone or independent, but have to deal simultaneously with an internal and an external component, which can change over time by the execution of actions.

A clear example of this is the Semantic Web, an evolution of the World Wide Web to capacitate the Web with machine-processable data. In fact, while the World Wide Web is already an incredibly powerful tool that dramatically changed the way we communicate and share knowledge, its content is hard to be understood by automated processes. Traditionally, data published on the Web was designed mainly for humans and made available in strict formats such as CSV or XML, or marked up as HTML tables, sacrificing much of its structure and semantics. The Semantic Web, led by an original proposal of Tim Berners-Lee [BLHL01], aims to disrupt this tendency by explicitly defining the semantics of contents and links for machine consumption. Underpinning this evolution is a demand to provide data as "raw", enabling the construction of powerful linked data mashups across heterogeneous data source collections, without further programming effort. This movement has gained such popularity that, today, the amount of links between datasets, as well as the quality of these links, has largely increased, paving the way to a *Web of Data* with the ultimate goal to use the web like a single global database.

Nevertheless, before using the web as a huge database, there are still several research quests that need to be addressed. Namely, for such realization, it is essential to ensure (at least some) properties that one is used to see in standard databases.

Clearly, the (Semantic) Web as envisioned should be able to perform more activities than just querying. Communication platforms such as wikis (where several users can modify the same document), or online market places, are examples of existing web applications that require updates according to client requests or actions. Moreover, just like the Semantic Web is based on a cooperative behavior to publish web content in standard format (namely, OWL and RDF), it makes sense to promote a similar behavior to provide the web with the tools it needs to automatically evolve. In this context, W3C recommendations like SPARQL-Update [GPP13] and RIF-PRD [SMHP10] show that an important collaborative effort is being made to give semantics to changes and actions in the Semantic Web.

However, from the moment that actions and updates are possible, it appears the need to ensure some integrity properties regarding the outcome of performing such actions. As an illustration, imagine some RIF-PRD production rule stating that: ***If*** *a customer reaches $5000 of cumulative purchases during the current year* ***then*** *its status becomes* Gold *and*

*a golden customer card will be printed and sent to him within one week*. In RIF-PRD syntax this is translated into:

```
Prefix(ex <http://example.com/2008/prd1#>)
Forall ?customer ?purchasesYTD (
 If   And( ?customer#ex:Customer
           ?customer[ex:purchasesYTD->?purchasesYTD]
           External(pred:numeric-greater-than(?purchasesYTD 5000)))
 Then Do( And( Modify(?customer[ex:status->"Gold"])
               Execute(act:printCard(?customer,"Gold")))))
```

One obvious requirement of applying such rule is atomicity, that is, if the action could not be performed completely, then it should not be performed at all. In this particular example, a customer should not become a gold customer without the emission of the corresponding card, neither a card should be delivered to a customer whose status is not *gold*.

This kind of problems is generally solved in databases with the use of transactions. Transactions ensure atomicity, consistency, isolation and durability of a special set of actions. These properties, constituting the ACID model, play a fundamental role in providing reliability to standard databases.

Nevertheless, when comparing the Semantic Web to standard databases, we can find several important differences. In particular, while one fully controls how data changes in a database, this is may not be true in the Semantic Web. To provide a better understanding, consider figure 1.1 illustrating a random node's point of view in this context. Intuitively, the node's knowledge integrates both its internal knowledge and all the knowledge of the external nodes connected to it. In addition, this knowledge is not static but is constantly evolving due to the execution of actions. However, these actions clearly have very different characteristics depending on the domain where they are executed. Whereas the node fully controls the actions performed internally over its own knowledge, it cannot control the actions performed externally over other nodes' knowledge. Thus, when ensuring that these internal and external actions follow a transactional model, we cannot guarantee that external actions follow the same transaction model as internal actions, as we have a different control over them.

In addition, an obvious requirement of the Semantic Web is the ability to detect complex changes on the environment, and react to them in some way. In fact, since the node's knowledge also includes the knowledge of other nodes, it is important to automatically issue actions in response to changes made on the environment by others, but while guaranteeing transactional properties of the actions executed as a response.

As another example, intelligent agents in a multi-agent system [Woo09] also share all of the previously mentioned requirements. In the context of AI, an intelligent agent is an autonomous system that interacts and acts upon an external environment, to achieve a

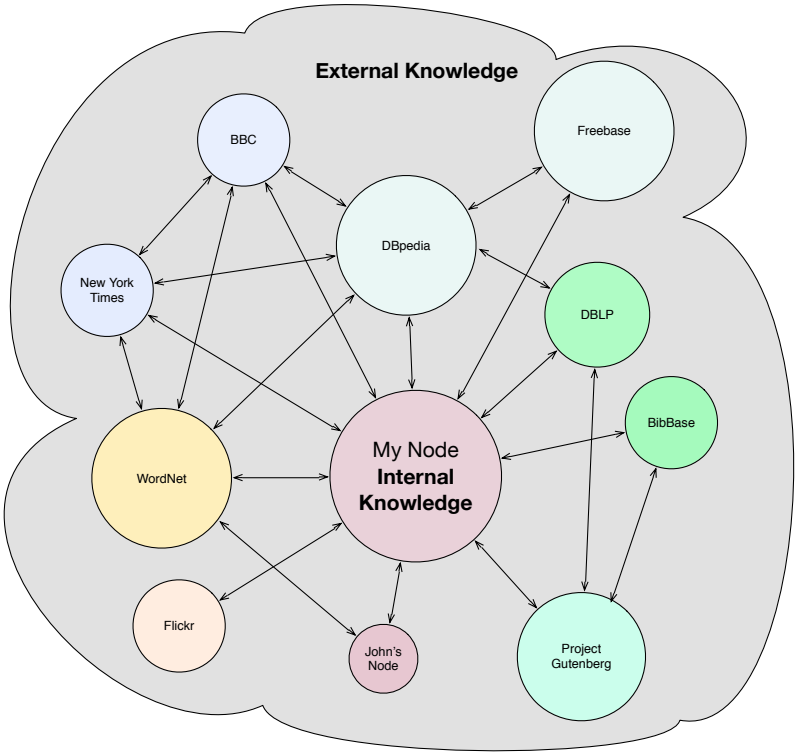



Figure 1.1: Node's point of view in the Semantic Web

set of goals. Agents are considered to be rational, and can adapt their knowledge about the environment and their goals, by observation and learning.

When modeling how an agent interacts with the world and with other agents, we can clearly distinguish between the actions that an agent does over its internal knowledge, and the actions that it does over the external environment. While the agent has full control on the actions done over its internal knowledge, this is not the case when performing actions in the external environment, which the agent cannot control, nor predict the full outcome. Additionally, the agent clearly needs to be able to detect changes made in the environment (caused by its own actions and other agents' actions) and react to them in some way.

Furthermore, in some situations it is crucial to ensure transactional properties over the outcome of the agent's actions. To better illustrate this need, consider figure 1.2, exemplifying the scenario of an agent in a medical context. In it, the agent has both an internal knowledge base comprising information about diseases, treatments and patients' treatment history, and interacts with an external entity: a patient. With the goal to make patients better, the agent can eventually give treatments to the patient, when believing that the treatment will improve the patient's condition given her symptoms.

Clearly, we need to guarantee properties over the execution of such a medical agent. Namely, we need to guarantee that if the treatment fails (e.g. because the patient reacts badly to it) then something is done to preserve consistency, i.e., to leave the patient in a stable health state.

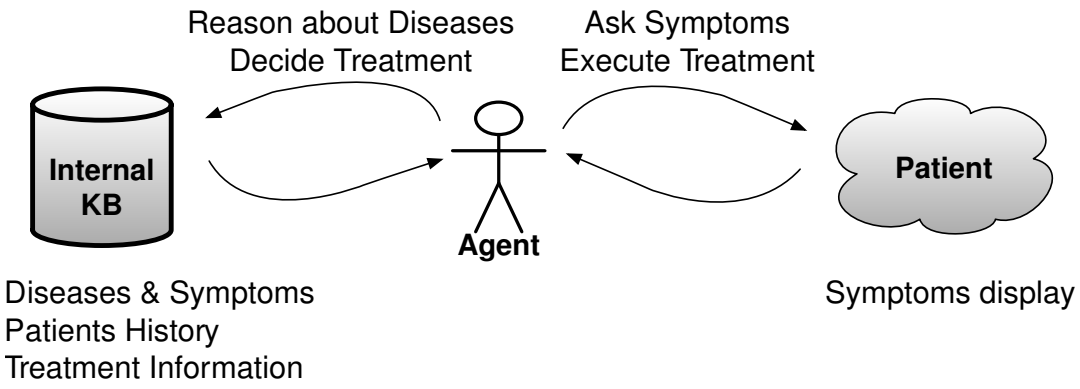



Figure 1.2: Intelligent Agent in a medical scenario

## 1.2 Transactional Properties in Dynamic Reactive Environments

Traditionally, achieving transactional properties means to execute actions according to the aforementioned ACID model. This model was originally proposed by the database community, and constitutes the standard paradigm to achieve reliability in databases.

Nevertheless, the literature has proven that the ACID model is inadequate for domains that stand out from standard relational databases. This is e.g. the case, when the domain is required to handle external knowledge [GMS87], or when availability of resources is more important than ensuring some properties of the model [Cat11].

In this work, we address the problem of relaxing the transactional ACID model to make it suitable for the dynamic reactive environments presented earlier, adapting it to deal with external knowledge and reactivity.

As previously mentioned, the problem with executing actions over external knowledge is that these actions cannot be fully controled. In particular, atomicity is achieved by means of rollback operations, and these rollback operations are unattainable over external knowledge. A rollback is defined as the operation of reversing back all the effects of the executed actions, by restoring the knowledge base state before the transaction was executed. Hence, since we cannot control a knowledge base which is *external*, state reversing is normally impossible. This is clear, for instance in the example of figure 1.2, where the agent cannot simply restore the patient's state before executing the treatment.

Another important requirement of dynamic environments is reactivity, i.e., the ability to detect changes in the environment and react automatically to them. This need emerges both in multi-agent systems, where an agent may need to act upon changes made by others in the environment; or in a Semantic Web context, where nodes need to update their information according to changes detected in other nodes' data. However, reactivity is hard to accomplish when transactions are involved. Since a transaction is executed in an all-or-nothing manner, the detection of changes normally needs to be postponed until the transaction commits, preventing changes to be handled immediately. Furthermore,

in a transactional context, the reaction itself should also be executed in an all-or-nothing way. In database systems this also implies that, if a transaction triggers the execution of some reaction and this fails, both the changes made by the reaction and the original transaction are to be rolled back. Due to these restrictions, transactions and reactivity are normally only combined in languages where the expressivity of actions is very limited (as e.g. in relational databases where actions are restricted to inserts and deletes).

To address transactional properties in these environments we need a logic-based solution ensuring that the execution of actions follows the transaction model. Since several proposals to represent knowledge and state change exist, this solution needs to be as abstract as possible, and suitable to work with several different semantics, so as to be applicable in scenarios like the Semantic Web, intelligent agents, among many others.

Finally, we argue that such a solution should offer a declarative syntax to represent transactions. Declarative languages provide ways to define programs that are clear and quickly understandable by their users. They have the advantage of being substantially concise and self-explanatory, as they state what is to be computed, or executed, rather than how it is to be executed. As a result, declarative programs are inherently high-level, where the details of the computation are left to the abstract machine. As such, declarative languages facilitate the investigation of formal properties over the programs written, which is meaningful, e.g., when one wants to guarantee that the actions defined in the program follow the transaction model.

Subsequently, declarative languages promote clarity, re-usability and interoperability, which are crucial characteristics for solutions aiming to be applicable in a large spectrum of application scenarios. In particular, these features are especially important in a Web domain context, where the tremendously large amount of users and data demands clear and simple tools to represent knowledge, as well as the actions to be executed over this knowledge.

## 1.3 Contributions and Roadmap

As a general contribution of this thesis we provide a novel logic to reason and execute transactions over dynamic and reactive environments where knowledge is partitioned over an internal and external domain. Additionally, this logic can be parametrized with different knowledge representation semantics, and thus be useful in a wide set of domains.

With this in mind, our work builds upon the well-established Transaction Logic, and provides the following contributions.

**A logic for reasoning and executing external actions.** We introduce *External Transaction Logic*, an extension of Transaction Logic to accommodate interaction and execution of actions in an external entity, as e.g. external web-sources, web-services, or agents.

Transactions are defined in a logic programming style by the composition of internal

and external primitives. These primitives are incorporated in a quite general manner, as a parameter of the External Transaction Logic theory, allowing one to specify transactions that integrate knowledge and actions from multiple sources and semantics.

Since one has different control over internal and external domains, different transaction properties are ensured depending on where actions are executed. Namely, internal actions executed in a knowledge base that we fully control follow the standard transactional model. On the contrary, transactional properties over actions executed externally need to be relaxed, as it is impossible to roll back actions executed in a domain that is external. To deal with this, external actions can be defined along with compensating operations. If a transaction fails after executing some external action, then these compensations are executed in a backward order to achieve a relaxed model of atomicity.

To reason about transactions in External Transaction Logic, we provide a model theory to reason about the execution of transactions that require the issuing of both internal and external actions on abstract knowledge bases with potentially different state semantics. To account for execution, we present a corresponding proof theory (sound and complete w.r.t. the model theory) that enables us to execute such transactions in a top-down manner. Then, we also show how External Transaction Logic can be used with several different semantics to represent the dynamics of the external world, providing examples and oracles instantiations for languages like the Situation Calculus, Action Languages, Event Calculus and Description Logics.

**Computing automatic repairs using Action Languages for multi-agent systems.** Based on the previously defined External Transaction Logic, we show how it can be used for reasoning about the behavior of agents. Particularly, for agents that have to operate in a two-fold environment in a transactional way: with an internal knowledge base defining the agent's internal knowledge and rules of behavior, and an external world where the agent executes actions and interacts with other entities.

Moreover, actions performed by the agent in the external world may fail, e.g. because their preconditions are not met or because they violate some norm of the external environment. The failure to execute some action must lead, in the internal knowledge base, to its complete rollback, following the standard transaction model. On the other hand, since it is impossible to roll back external actions performed in the outside world, external consistency must be achieved by executing compensating operations (or repairs) that revert the effects of the initial executed actions.

In the previous External Transaction Logic, repairs are stated explicitly in the program by the user, (i.e., they are explicitly associated with the corresponding external action), and do not guarantee that the effects of the executed actions are indeed reverted. To handle this, we show how to automatically compute external compensations in case of failure in External Transaction Logic. External Transaction Logic is parametric on the semantics of both the internal and external domain, and since a compensation should obtain an equivalent state prior to the execution of the external action, automatically

inferring such compensations requires making real assumptions about the semantics of external states. Due to this, we fix Action Languages as the semantics for the external environment, and show how to automatically compute the repairs for external actions in External Transaction Logic for this semantics.

**A logic for reasoning and executing reactive transactions.** We introduce *Transaction Logic with Events*, a non-monotonic logic programming language to handle transactions that react automatically to complex events resulting from the combination of internal changes and external requests. With this as a goal, we first show that, while Transaction Logic can be used to encode most complex event patterns expressible in Complex Event Processing algebras like SNOOP [AC06] or ETALIS [AFRSSS10] and reason about their occurrence over paths, Transaction Logic is insufficient to simultaneously deal with complex events and the execution of transactions as a response. Then, to achieve such combination we propose Transaction Logic with Events, an extension of Transaction Logic where transactions are required to respond to all (complex) events that occur in a given execution trace, i.e., a transaction can only succeed in an execution path if every event fired in the path is responded to. For that, Transaction Logic with Events is based on a non-monotonic theory imposing that transaction formulas must fail over paths where an unanswered event occurs. This goes in line with database trigger behavior where failing to respond to a trigger precludes the commit of the main transaction in which the trigger was fired.

Finally, we explain how Transaction Logic with Events can be used as an Event-Condition-Transaction language, providing a non-trivial proof procedure to detect complex events and automatically execute their response rules as a transaction.

**An unified logic for reasoning and executing reactive transactions involving external actions.** We provide an approach for combining External Transaction Logic and Transaction Logic with Events as one unified logical solution. The resulting logic is a non-monotonic extension of Transaction Logic that reasons about reactive transactions involving the execution of internal and external actions.

In it, we can define complex events as the combination of internal and external events; define the transactions that should be executed in response of these events; and define these transactions as the combination of internal and external actions.

This solution guarantees that a transaction can only succeed over paths where all the events triggered are properly responded to as a transaction. Moreover, it also guarantees external consistency is achieved in case of failure, by issuing the execution of compensations externally, and rolling back actions, internally.

The remainder of this document is organized as follows:

**Chapter 2 – Background.** We provide a general overview over the related paradigms

and solutions in the context of representing and reasoning about actions, database systems, transactions and reactivity.

**Chapter 3 – Transaction Logic.** We present the syntax and semantics of Transaction Logic. For that, we review Transaction Logic's model theory and proof theory, providing several examples, and explaining the role of the oracles in the semantics.

**Chapter 4 – Motivation: why do we need external actions and transactions.** We motivate for the need to achieve transactional properties over actions that execute over an internal and external environment. With that in mind, we provide several use-case examples of scenarios where this type of actions are important. Then, we explore what kind of problems arise when executing external actions in a transactional environment, and how compensations can be used to overcome them.

**Chapter 5 – $\mathcal{ETR}$: Extending Transaction Logic with External Actions.** Based on the theory of Transaction Logic, we provide an abstract logic, called External Transaction Logic, to execute and reason about transactions involving the execution of both internal and external actions. For this purpose, we start by extending Transaction Logic theory with an additional external oracle parameter, to abstract the semantics of states and updates of the external component. Then, we formally define what it means for a transaction to fail over a path and recover from this failure by executing external compensations and internal rollbacks. Subsequently, we integrate this notion in a new model theory where we model the success of a transaction over a path by the paths where the transaction executes without any failure, or the paths where, although a failure occurs, the transaction can recover from this failure and succeed after that. Additionally, we formalize the correspondence between External Transaction Logic and Transaction Logic, showing that both logics prove the same formulas, when the program does not have external actions. We also provide a proof theory for External Transaction Logic, which extends the one of Transaction Logic with an additional oracle, and the possibility of an execution to fail, recover, and succeed after the failure.

**Chapter 6 – $\mathcal{ETR}$ usage and the role of (external) oracles.** In this chapter we explore the role of the new external oracle of External Transaction Logic, by providing several external oracle instantiations for the context of the Semantic Web and intelligent agents. Namely, we define how this oracle can be instantiated with Description Logics semantics (for the Semantic Web), and with Action Languages, Event Calculus and Situation Calculus (for intelligent agents).

**Chapter 7 – Automatic compensations in $\mathcal{ETR}$.** In the External Transaction Logic defined in chapter 5 recovery is only possible if the correct compensations are given by the programmer when writing the program. Realizing that this can be a problem when the programmer does not have enough knowledge about the external world, we

investigate how to automatically compute these compensations in External Transaction Logic. To achieve this, we fix the semantics for the external knowledge base with the Action Language $\mathcal{C}$, and use the external oracle to compute the correct compensations for each external action at a given time.

**Chapter 8 – $\mathcal{ETR}$ Discussion.** In this chapter we provide a general discussion of the theory of External Transaction Logic, including a comparison with related work on the field.

**Chapter 9 – Motivation: why should transactions be truly reactive.** We motivate for the need to combine transactions with reactive features, arguing why it is useful in several scenarios, and introducing the main problems deriving from this combination.

**Chapter 10 – Using $\mathcal{TR}$ to encode Event Algebras and Transactions.** We show that Transaction Logic can express and reason about expressive and complex events. For this purpose, we show how it can express most operators of SNOOP and ETALIS event algebras, and how to use its model theory to reason about the complex events that occurred over a given path. Afterwards, we also show that, although Transaction Logic can be used to reason about events and transaction execution, it cannot deal with both concepts simultaneously.

**Chapter 11 – $\mathcal{TR}^{ev}$: Reactive Transaction Logic.** Building upon the previous two chapters and upon the theory of Transaction Logic, we define an abstract logic to reason and execute reactive transactions. This logic, called Transaction Logic with Events, provides means to express complex event rules, but also to define what transactions should be executed as a response of these event occurrences. Moreover, using this logic, we can guarantee that a transaction only succeeds over paths where all occurring events are properly responded to as a transaction. Subsequently, we show that, although this logic is non-monotonic and Transaction Logic is monotonic, it is still a conservative extension of Transaction Logic, as both prove the same properties when the program does not contain events. Finally, we also provide a non-trivial procedure that executes reactive transactions according to the theory.

**Chapter 12 – $\mathcal{TR}^{ev}$ Discussion.** In this chapter we provide a general discussion of Transaction Logic with Events, comparing it with relevant related solutions.

**Chapter 13 – Reactive Transactions with External Actions.** We explore how our two previous solutions can be combined as a single logic solution. To this end, we merge the model theory of External Transaction Logic, with the theory of Transaction Logic with Events. In it, events can be be defined by the combination of internal and external events, and we can guarantee that a transaction only succeeds over paths where all occurring events are responded to. Moreover, a transaction can formally fail over a path, if some action cannot be executed, or if we cannot respond

to some event triggered by this transaction. If such a failure occurs after the execution of external actions, we can still recover from this failure by combining internal rollbacks with external compensations.

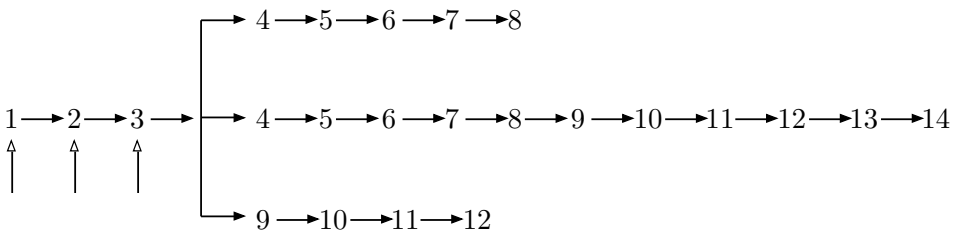
**Chapter 14 – Conclusions and Future Directions.** In this final chapter, we summarize the main contributions of the thesis, and discuss several interesting future lines of research.

**Appendices – Proofs.** We present the proofs of the theoretical results of chapters 5, 7, 10 and 11.

This thesis is organized in several parts which, according to the reader's preferences and background, can be more relevant than others. Part I, composed by chapters 1, 2, 3, introduces the general problem and makes an historical overview over existing solutions of actions, state change, database systems and transactions. Some of its contents can be skipped if the reader is familiar with the related work and with Transaction Logic.

Part II, comprising chapters 4, 5, 6, 7 and 8, deals with the problem of achieving transactional properties over domains that have an internal and external component. Part III addresses the problem of combining transactions with reactive features, and includes the chapters 9, 10, 11 and 12. Part IV, composed by chapter 13 handles the joint problem of executing reactive transactions over domains that have both an internal and external component.

Parts II and III are mainly independent of one another, and can be read in any order, or even skipped without compromising understanding of the other part. Nevertheless, both of them are essential for part IV's comprehension. While the best way to read this document is by reading the chapters in the order they appear in, we also envisage other reading paths:



# 2

# Background

In this chapter we introduce some background paradigms and results in the context of representing and reasoning about actions, transactions and reactivity. The purpose of this chapter is to give a general historical overview, to help the reader better understand our contributions. We start by introducing some of the seminal solutions in reasoning about abstract actions in Artificial Intelligence (section 2.1). Then, we talk about the main concepts and challenges when achieving ACID transaction properties in database systems, and how these systems have have evolved since their appearance (section 2.2). Afterwards, we introduce some work in the context of representing and modeling abstract transactions in a logical way (section 2.3). Finally, we present and motivate reactive solutions (section 2.4). We should note that we do not go into technical details of these solutions in this chapter, and that some of the work referred herein, will be further developed ahead in this thesis.

## 2.1 Logics to reason about actions

A number of solutions have been proposed in the literature to represent expressive dynamic “real-world"-like domains. These are normally denoted as action theories, or domain description languages, and with them, one can reason about the properties of the world, about the conditions on which actions can be executed, and determine what are the direct and indirect effects in the world after an action’s execution.

Initially proposed to imitate human reasoning in general AI, these solutions offer a highly expressive syntax to encode a description of the domain, and its dynamics. In this context, the most well-known examples include the Situation Calculus [McC63], Event

Calculus [KS86], Action Languages [GL98a], Fluent Calculus [Thi97], among many others. These offer different features like the possibility to encode non-deterministic actions and uncertainty, concurrency, causality, explicit representation of time, knowledge and belief updates, etc.

One of the main focus in all of these logics is how to handle the so-called *frame problem* [MH69]. When modeling changes and their effects in dynamic environments, the frame problem corresponds to the difficulty in representing what does not change when some action is executed. Since one does not want to define rules for every possible combination of actions and possible world states, the problem is how to express non-effects concisely. Thus, to be able to talk about dynamic environments, all of the aforementioned solutions have some form of frame axioms, which specify what remains unchanged after the execution of some action. Interestingly, the notion of non-monotonic reasoning was first introduced to address the frame problem, and afterwards to allow modeling and reasoning with defaults and exceptions. By providing a way to state what holds *in general*, along with the notion of exceptions, non-monotonic reasoning can overcome "the impossibility of naming every conceivable thing that may go wrong" [MH69]. The name *non-monotonic* comes from the behavior where additional information may invalidate previous conclusions. A classical example of this is "birds typically fly. Penguins are birds that do not fly. And tweety is a bird". From this knowledge, non-monotonic reasoning allow make one conclude that tweety is flying bird. However, later one may learn that tweety is actually a penguin, and be forced to revise the conclusions made previously.

The Situation Calculus, introduced by John McCarthy in 1963 [McC63], is a first-order language to represent general dynamic environments, in which all of the changes result from the action of some agent in the world. States of the world are represented as situations, while fluents define properties that hold in a given situation, and actions are defined with pre-condition axioms stating what fluents need to hold for an action execution be applicable in a situation. To better illustrate how one can express knowledge and actions in the Situation Calculus, consider the Yale Shooting example, first proposed in [HM86].

**Example 1** (Yale Shooting – Situation Calculus)**.** *In this example we represent and model the killing of a turkey named Fred. In it, initially Fred is alive, and the gun is unloaded. The actions of loading the gun, and shooting Fred afterwards, should kill Fred.*

*Representing such an initial setting in Situation Calculus is done by stating what holds in the initial situation* $s_0$*:*

$$Holds(Alive(fred), s_0) \qquad \neg Holds(loaded, s_0)$$

*where* $alive(fred)$ *and* $loaded$ *are fluents specifying the truth of Fred being alive, and the gun being loaded, respectively.*

*Afterwards we can say that shooting Fred with a loaded gun, will change the fluent* $alive(fred)$

*to false, and make the gun unloaded:*

$$\forall s.Holds(loaded, s) \rightarrow \neg Holds(alive(fred), \texttt{do}(shoot, s))$$
$$\forall s.\neg Holds(loaded, \texttt{do}(shoot, s))$$

*Finally, we can say that loading a gun makes the gun loaded:*

$$\forall s.Holds(loaded, \texttt{do}(load, s)).$$

The frame problem is an important issue of the Situation Calculus, as the amount of frame axioms necessary to describe situations adequately is considerably large (up to $2 \times \mathcal{F} \times \mathcal{A}$ in a naive solution) in comparison to the domain of fluents ($\mathcal{F}$) and actions ($\mathcal{A}$). This is especially problematic when considering that, in general, most fluents are unaffected by most actions. For instance, imagine if in example 1 we add an extra action of waiting after loading the gun and before shooting Fred. Then, the statement $\neg Holds(alive(fred), \texttt{do}(shoot, \texttt{do}(wait, \texttt{do}(load, s_0))))$ will not be true with the above formalization, as we still need to specify that the waiting action does not make the previously loaded gun to be unloaded.

Kowalski and Sergot introduced the Event Calculus in [KS86] with the goal to overcome some limitations of the Situation Calculus. Namely, they have addressed the possibility to describe change continuously over time, and to represent concurrent actions in a native way. In it, the world is described using a notion of linear time, instead of a global situation like in Situation Calculus, allowing one to talk about simultaneous and partially ordered actions. In addition, actions are denoted as events and occur at a given time point, whereas fluents can occur over an interval (i.e., their truth value may change within time points).

An important feature of Event Calculus is that it can be expressed by means of Horn clauses augmented with negation as failure, which allows it to be easily translated into a Prolog program.

**Example 2** (Yale Shooting – Event Calculus)**.** *To illustrate the Event Calculus, we formalize again the Yale Shooting scenario, as we did in example 1. We assume the special predicates $initially$, $initiates$ and $terminates$, where $initially(f)$, denotes that fluent $f$ holds at time 0; $initiates(f, a, t)$, states that action $a$ initiates fluent $f$ at time $t$; and $terminates(f, a, t)$ states that action $a$ terminates fluent $f$ at time $t$. Moreover, $happens$ defines what events happen in which time points, while $holdsAt$ defines what fluents hold at which time points. Afterwards, the initial setting can be represented as:*

$$initially(alive(fred)).$$

*Then, we can state that doing action $shoot$ at any time $T$ terminates the truth of $alive(fred)$, if the gun is loaded at that same time; and that shooting a gun terminates the truth value of the*

*fluent loaded:*

$$terminates(alive(fred), shoot, T) \leftarrow holdsAt(loaded, T)$$
$$terminates(loaded, shoot)$$

*Finally, we state that the action load initiates the truth value of the fluent loaded:*

$$initiates(loaded, load, T)$$

Event Calculus was mainly developed to avoid the frame problem. This is done by qualifying relations with time periods, instead of using global situations, and by giving the possibility of using non-monotonic reasoning. Subsequently, it includes general axioms, stating that by default things (or fluents) remain true until they are terminated. This is expressed in Event Calculus as follows:

$$\begin{aligned} holdsAt(P,T) &\leftarrow 0 \leq T, initially(P), \textbf{not } clipped(0,P,T). \\ holdsAt(P,T) &\leftarrow happens(E_1,T_1), T_1 < T, initiates(E_1,P,T_1), \textbf{not } clipped(T_1,P,T). \\ clipped(T_1,P,T) &\leftarrow happens(E_2,T_2), T_1 < T_2, T_2 < T, terminates(E_2,P,T_2). \end{aligned}$$

Similarly to the Event Calculus, Action Languages proposed by Gelfond and Lifschitz [GL92; GL98a] are one of the many action formalisms extending the Situation Calculus with non-monotonic reasoning. An important aspect of Action Languages is they comprise a family of languages, rather than one single language. As a family of languages, they offer different ways to express reasoning, and different choices on the possibility to express frame axioms, concurrency, non-deterministic actions and indirect effects of actions. Common to all Action Languages, is the possibility to model the dynamic world as a transition system, where nodes are the possible states and arcs are labeled with actions defined in the domain. As illustration of this formalism, we pick the most simple Action Language, language $\mathcal{A}$, and show how it can encode the Yale Shooting problem.

**Example 3** (Yale Shooting – Action Language $\mathcal{A}$)**.** *In language $\mathcal{A}$ the initial setting is described by using the statement* initially*:*

$$\textbf{initially } alive$$

*Then the effects of actions can be expressed by the dynamic laws: $A$* **causes** *$F$* **if** *$P_1; P_2; \ldots; P_n$, where $F$ is the result of performing action $A$ in a state where the preconditions $P_1; P_2; \ldots; P_n$ hold.*

$$\begin{aligned} shoot &\ \textbf{causes}\ \neg alive \quad \textbf{if } loaded. \\ shoot &\ \textbf{causes}\ \neg loaded. \\ load &\ \textbf{causes}\ loaded. \end{aligned}$$

*The semantics of Action Language $\mathcal{A}$ can also be given by a translation, based on the Situation Calculus, into logic programs with default negation (under the answer-set semantics [GL92]).*

*The translation of the Yale Shooting specification in $\mathcal{A}$ above is:*

$$
\begin{array}{ll}
Holds(alive, s_0) & \neg Holds(loaded, s_0) \\
Holds(loaded, \texttt{do}(load, S)) & Noninertial(loaded, load, S) \\
\neg Holds(loaded, \texttt{do}(shoot, S)) & Noninertial(loaded, shoot, S)
\end{array}
$$

$$
\begin{aligned}
\neg Holds(alive, \texttt{do}(shoot, S)) &\leftarrow Holds(loaded, S) \\
Noninertial(alive, shoot, S) &\leftarrow \textbf{not } \neg Holds(loaded, S) \\
Holds(loaded, S) &\leftarrow Holds(alive, S), \neg Holds(alive, \texttt{do}(shoot, S)) \\
\neg Holds(loaded, S) &\leftarrow Holds(alive, \texttt{do}(shoot, S))
\end{aligned}
$$

*To deal with the frame problem, every translated program always includes the four rules:*

$$
\begin{aligned}
Holds(F, \texttt{do}(A, S)) &\leftarrow Holds(F, S), \textbf{not } Noninertial(F, A, S) \\
\neg Holds(F, \texttt{do}(A, S)) &\leftarrow \neg Holds(F, S), \textbf{not } Noninertial(F, A, S) \\
Holds(F, S) &\leftarrow Holds(F, \texttt{do}(A, S)), \textbf{not } Noninertial(F, A, S) \\
\neg Holds(F, S) &\leftarrow \neg Holds(F, \texttt{do}(A, S)), \textbf{not } Noninertial(F, A, S)
\end{aligned}
$$

Fluent Calculus [Thi97] is another popular solution which can also be seen as a modern extension of the Situation Calculus. It is action-centered, rather than fluent-centered as the Situation Calculus, which means that for each action it specifies the affects that it has in a state, and the frame problem is handled by state update axioms.

**Example 4** (Yale Shooting – Fluent Calculus)**.** *As illustration of the Fluent Calculus, we repeat the formalization of the Yale Shooting problem. Fluent Calculus, extends Situation Calculus by introducing a notion of states. Then, situations are seen as representations of states, and statement $State(S)$ denotes the state of the world in situation $S$.*

*An important aspect of this formalism is the operator $\circ$. This is used for composing fluents, allowing one to infer non-effects of actions without extra frame axioms:*

$$State(\texttt{do}(shoot, S \circ alive(fred) \circ loaded)) = S \tag{2.1}$$

$$State(\texttt{do}(load, S)) = S \circ loaded \tag{2.2}$$

*where eq. (2.1) states that the result of doing action $shoot$ in a situation where $alive(fred)$ and $loaded$ is true, is the same situation where these two fluents are not true; and eq. (2.2) states that doing the action $load$ in a situation $S$ results in the same $S$ but where $loaded$ is true.*

## 2.2 Transaction properties, database systems and active database systems

While all of the aforementioned solutions can deal with the general problem of reasoning about expressive dynamic domains, they also come with a very high reasoning cost, making them hard to be used in practice.

Contemporaneously to the development of action theories, database systems appear as a solution to deal with representing, storing and querying simpler knowledge in an efficient way, and which could be used by a transverse number of application domains. Although they have first appeared in industry applications around the 1960s, database systems became popular with the advent of the relational model, proposed by Edgar F. Codd in 1970 [Cod70], along with the formalization of transactions by Jim Gray [Gra80; Gra81], and the ACID model [HR83] in the 80s. Afterwards, the SQL language was proposed as a standard language to manipulate databases by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) as ANSI/ISO SQL in 1986, and is now the lingua franca of databases systems.

In databases, the relational model is what provides the theoretical background to reason and manipulate knowledge in a simple way. In it, data is represented in the form of tuples and organized by relations (or tables), and can be manipulated, selected and projected by means of relational algebra operators. Subsequently, the ACID transaction model provides a paramount reliability, by defining a series of properties that must be followed when updating knowledge in database systems.

Obviously, when compared with the previous mentioned solutions to reason about actions, the relational model is considerably limited. Nevertheless, its simplicity, declarative syntax, efficiency and fault-tolerance capabilities, made databases the primary solution to model and query knowledge in industry applications until today. Actions are limited to simple inserts and deletes of tuples, and closed-world assumption (a form of non-monotonic reasoning) [Rei77] is employed, meaning that everything that can not be derived by a database system is considered as false. As such, since there are no indirect effects in the state of the database, and except for the inserted and deleted tuple, everything stays exactly the same, the frame problem is not an issue.

A very important contribution of database systems is the ACID model of transactions. The ACID model provides crucial fault-tolerance characteristics to databases, by defining a series of properties that all database operations must follow, and by specifying how such a database system can recover in case of property violation. Then, a transaction is defined as a set of actions that must be executed according to the ACID model, where ACID stands for atomicity, consistency, isolation and durability. In this model, atomicity requires the transaction to be executed in an indivisible *all-or-nothing* way, meaning that either the whole set of actions can be executed, or if something fails, the entire transaction fails, and the database must be left unchanged as if nothing happened before the execution of the transaction. Consistency ensures that the database is always left in a valid state at the end of a transaction execution, where no database constraints are violated. Isolation ensures that, in case of concurrent execution of transactions, the end result is equivalent to execute each transaction serially. In other words, a concurrent execution of transactions must be equivalent to execute transactions individually, one after the other. Finally, durability means that once a transaction is considerably successful, its changes will persist, even in the event of hardware failure like power loss, crashes, etc.

In order to achieve such a model, databases rely on concepts like commit, rollback, synchronization, locks, logging and redundancy. In this context, whenever a transaction can succeed following the ACID model, then its changes are committed to the database, i.e., they are made permanent and can be accessed by other transactions. In opposition, when a transaction fails, e.g. because some constraint is violated, then the transaction aborts, and changes are rolled back to a previous consistent state. Here, a rollback can be seen as the undoing of the (partial) effects of a transaction, and is this behavior of committing and rolling back what achieves the properties of atomicity and consistency. Afterwards, isolation is achieved through synchronization methods which prevent two transactions to change the same piece of data at the same time. One of the most important synchronization methods is the lock, where a transaction can ask to lock tuples or tables, blocking other transactions to access their partial (and possibly not consistent) changes. Finally, durability is normally achieved through logging and hardware redundancy. Normally, every database systems writes transactions into a transaction log that can be used to recreate the last consistent system state, after any (hardware) failure.

In the late 80s, a trend of research [Day88; HLM88; MD89] gained strength to extend database with reactive capabilities. This led to the development of active database systems, which extended the traditional (passive) databases with an event-driven architecture. In it, one could write event-condition-action rules specifying how a database should react to a situation of interest (also known as an event). In this context, event-condition-action (ECA) rules are rules of the following form:

*on* **event** *if* **condition** *do* **action** (ECA-rule)

where, whenever an event is known to be true, the condition is checked to hold in the current database state and, if that is the case, the action is executed. This kind of event-driven architecture has shown to be a powerful mechanism in database systems to enforce richer integrity constraints, implement alerts, maintain derived data, enforce access constraints, implement version control policies, gather statistics, etc. [DHW95].

This architecture was later incorporated in SQL:1999 in the form of database *triggers*, which are now the standard to encode reactivity in modern databases systems, by providing an elegant solution to monitor and act upon changes in SQL data.

While triggers are implemented differently upon database management systems, normally triggering events are restricted to inserts, deletes and updates in tables, views and tuples. Whenever a trigger event occurs, the piece of procedure code defined in the trigger's function is called and executed.

An important aspect of a trigger behavior is that, if the execution of the trigger's procedure code fails, e.g. because it takes the database into an inconsistent state, then the transaction associated with the trigger fails as well. In this sense, a trigger is always attached to the transaction causing it. If an error occurs during the trigger execution, that error causes transaction rollback, and the firing action will be rolled back too.

Since every trigger is coupled to a transaction, the authors of [HLM88] introduced the idea of providing different *coupling modes*, which were further developed in the context of the HiPAC System [DBBCHLMRSCLJ88]. In an Event-Condition-Action architecture, where E is an event, C the condition, and A the action, coupling modes allow one to define when (i.e., in which transaction) should the condition be tested w.r.t. the event (E-C mode), and when should the action be executed (C-A mode). Coupling modes can be: *immediate* meaning an immediate execution in the same transaction; *deferred* indicating that the execution is performed at the end of the current transaction (but still in the same transaction); and *decoupled* stating that the execution is in a separate transaction. Consequently, coupling modes allow a low-level control over how active rules are processed relative to the transaction that triggered it. However, they complicate maintenance and understanding of active rules, and because of that, they are not implemented in commercial database system.

Another important concept in database theory is the notion of *long-lived transactions*, or sagas [GMS87]. These denote transactions that can last for relatively long periods of time, holding on to database resources (as table locks), and delaying the termination of shorter and more common transactions. Reasons behind this time consumption can be many-fold, like interaction with the user; the amount of computation and access to many database objects; or interaction with external entities and external services. Since these transactions can have a huge impact on the termination of other transactions, Garcia-Molina and Salem, proposed in [GMS87] the idea of breaking up a long-lived transaction into several subtransactions and provide compensating operations for each of them. In this sense, a compensation is an action (or a set of actions) that undoes the effects of the previous committed subtransaction, but without necessarily reverting the previous state before execution. As such, by using compensations we obtain an optimized execution, but with the cost of relaxing the notion of atomicity. A rollback can be seen as a special case, where the compensation always takes the database into the exact previous state prior to execution.

Nevertheless, this notion of compensations has become paramount to achieve database consistency, whenever external actions are part of the transaction. The problem with executing external actions is that they are executed in domains which are external to databases. This is e.g. the case when transactions needs to send an email, or interact with an external service to book a hotel room. In these scenarios, the database does not control the external database over which actions are performed, and thus rolling back such external actions is impossible. Compensations are the standard paradigm to manage this rollback impediment, and provide a relaxed model of atomicity for these transactions.

## 2.3 Logics to reason about Transactions

Based on the prominent success of database systems, the community of general knowledge representation dedicated a considerable amount of research in modeling transaction

programs and reason about their general properties. In fact, although originally based in predicate logic, relational database solutions are mostly procedural, lacking from a clear model-theoretical semantics, and making it hard to reason about the properties of their execution.

To solve this, works like [Kow92; Rei92] give a formal logic background to reason about database updates using the Situation Calculus and the Event Calculus as the basic framework. For that, these works extend databases tuples with time, allowing one to reason about the past and future database states, make historical queries, and prove properties about transaction database execution. Additionally, extensive work was also developed on formalizing the dynamics of active databases like in [Zan95; BLT97; BPV98; NB00; KR03; BDR04] using action theories and logic programming formalisms as a framework to represent transaction change.

In a different context, other logical-based solutions were proposed to natively reason about transactions. Successful examples of these include Transaction Logic [BK93; BK98a], Statelog [LLM98] and ULTRA [WFF98] language.

Transaction Logic [BK93] is a modal-like logic to reason about the behavior and execution of abstract transactions. Its theory is parametric on a pair of oracles which specify the primitive actions and updates, the dynamics and effects of actions on the states, as well as the intended resolution for the frame problem. Because of this design, Transaction Logic can reason about actions that follow the transaction model, *independently* of the state semantics chosen. In this sense, if one wants to use Transaction Logic for database updates, then oracles are defined according to the relational database semantics, where primitive actions are inserts, deletes and updates, and a state is a set of tuples. Nonetheless, several other oracle instantiations are possible like, for instance, for first-order-logic, well-founded semantics, etc.

An interesting characteristic of Transaction Logic is that its theory centers on the correct execution of a transaction according to the ACID transaction model. Because of that, interpretations are mappings from formulas and sequences of states (also known as paths) to truth values. Then, a formula and path are evaluated to true, if that formula executes transactionally over that path, and to false otherwise. Based on this, Transaction Logic's model theory can talk about all the paths that correctly execute a transaction formula, and reason about their properties and invariants. A proof procedure sound and complete w.r.t. the semantics, along with efficient implementations [FK10] also exist, allowing one to construct, in a top-down manner, the path that correctly executes a given transaction goal. Transaction Logic also has a concurrent version, which can reason about the behavior of concurrent ACID transactions.

ULTRA [WFF98] is a rule-based language that, like Transaction Logic, is able to model concurrent transactions in arbitrary domains. Its language is similar to Transaction Logic and, in fact, they are proven to have the same expressive power in their sequential version. ULTRA semantics is based on the concept of deferred updates, where updates are not immediately executed during evaluation, but postponed to a materialization phase.

As such, during the evaluation phase, the semantics computes the minimal update set needed to satisfy a transaction goal according to the ACID model. And only afterwards, these updates are made persistent and committed in the materialization phase. By employing this notion of deferred updates, ULTRA is able to elegantly specify set-oriented updates (or bulk update).

For executing transactions, ULTRA also provides an implementation and operational model [FWF99], which are based on the concept of nested transactions. Simply put, a nested transaction is a model where transactions can be defined inside another transactions, as subtransactions, with the goal to better control the commit and abort behavior of the subtransaction. In this context, changes committed by a subtransaction are only made persistent if and when the top level transaction commits, and failure of a subtransaction does not force the parent transaction to abort. Although not part of ULTRA's theory, ULTRA's implementation uses this concept of nested transaction. In it, the ACID model is achieved for top-level transactions, while subtransactions may relax the notion of atomicity by means of compensating operations which, as explained in section 2.2, can be seen as a relaxed alternative to rollbacks.

Statelog is a logic-based framework, proposed in [LLM98] to reason simultaneously about active and deductive rules in the context of database systems. In contrast to the previous Transaction Logic and ULTRA, Statelog is a reactive language, able to react to (internal and external) events and execute external actions. To reason about transaction evolution, Statelog extends Datalog with states, and hard-wires the information of states in every predicate, as $[S]p$, where $S$ denotes the index of the state where $p$ is true. Then, updates are defined by progressive rules, which also incorporate frame rules, and allow referring to the current and past states. The authors also define classes of Statelog programs and prove properties like termination and determinism for each of these classes.

Statelog can detect events, where internal events are simply defined as an atom to be true at a given state $S$, and external events are special atoms that are added to the database at some state, corresponding to the instant when the external event occurs.

In its simpler version, known as flat Statelog, the semantics takes an initial database, and an initial state containing all the facts and a list of external events, and computes the perfect model [Prz88] successfully for every (progressive) state, until it reaches a fixed point.

In its full version, Statelog can also deal with nested transactions, by means of Statelog procedures. A procedure is defined as a set of Statelog rules, and a Statelog program is composed of a main procedure, and a set of subprocedures. Then, the computation of a procedure is isolated from the rest of the program, and the semantics of these procedures is given by a model-theoretic Kripke style semantics.

## 2.4 Complex Events and Reactive Systems

While commercial database systems have become stagnant in the development of triggers and reactive rules, the development of the event-driven architecture sparked off a considerable amount of research in achieving an efficient and expressive way to reason about active rules and complex events.

The latter led to the appearance of Complex Event Processing (CEP) languages like [CM94; ADGI08; KS09; ARFS12], which focus on how to express complex and composite events, and how to detect them in the most efficient way possible. Here, events can be defined in broader terms, and are not confined to detection of simple insert and delete actions like in database systems.

In the context of CEP, events can arrive as a *stream* of continuous information and the goal is to detect interesting patterns as soon as possible. This research has been applied to business process management systems like IBM's WebSphere, Oracle Event Processing, StreamBase or ESPER, and has been used in many application scenarios, as algorithmic trading in financial services, process monitoring, fraud detection, etc.

Complex event processing languages normally have some kind of event algebra, over which one can specify complex event patterns. Then, CEP solutions provide some procedures and algorithms based e.g., in finite automata (NFA), Petri Nets, RETE algorithm, etc. to detect such event patterns. An important notion is that CEP systems adhere to a principle of decoupling [EN10], meaning that, the possible consequences of an event detection are not predetermined upon declaration of the event pattern.

CEP systems normally support different consumption policies (or event contexts). Since there may be several event occurrences of the same event which can be used to satisfy a complex event pattern, a consumption policy selects which event occurrences should be applicable, and how the multiple occurrences are consumed [CKAK94]. When an event is selected to satisfy a complex event, it is said to be consumed, and depending on the policy selected, that event may not be available to satisfy other complex events. As such, the choice of what consumption policy to follow has a huge impact in the semantics of an CEP, and goes against the principle of declarative programming, in the sense that the order of evaluation of event patterns matters [Ani11]. The most widely used consumption policies are: unrestricted, recent and chronological. In the unrestricted policy, all occurrences are valid and no event is said to be consumed when satisfying a pattern. In the recent policy, only the most recent event of its type is considered to construct complex events, consuming events from the stream in a last-in-first-out manner. Conversely, the chronological policy consumes events in chronological order, i.e., in a first-in-first-out order.

In opposition to CEP systems, which are decoupled from the consequences of detecting an event, Event-Condition-Action (ECA) languages explicitly define what should the system execute as a response to the detection of event patterns. While research in ECA rules started in the context of active databases, it soon left the realm of database

systems and started to be applied in other domains like the Semantic Web, multi-agent systems and AI, by providing a standard and clear way to define what should be done, when a given event pattern occurs. For this purpose, they support the syntax showed on eq. (ECA-rule) on page 21, and to be useful in practice, they allow both the event and the action component to be specified as complex. In this sense, they normally come with an event algebra do specify event patterns, just like in CEP systems, and have an additional action algebra to specify event responses. They are today the common standard to represent and model the behavior of reactive systems.

However, in opposition to active database systems, recent and general ECA languages [ABB11; BEP06; CLN03] have abandoned the notion of ensuring transactions when executing the (re)action component.

ECA languages can also support different consumption policies (as in CEP systems), and different event selection strategies. The latter specifies how events should be selected, when more than one event needs to be responded to at a given time (e.g., because they occur simultaneously). These choices define the *operational behavior* of an ECA language, and are implemented differently among the several languages.

# 3

# Preliminaries: Transaction Logic

In this chapter we provide a complete overview over the Transaction Logic ($\mathcal{TR}$) framework, including its model and proof theory, which will be used as the basic theory for our contributions.

$\mathcal{TR}$'s syntax is very similar to that of logic programming. Atoms have the form $p(t_1, \ldots, t_n)$ where $p$ is a predicate symbol and $t_i$'s are terms (variables, constants, function terms). To simplify the presentation, and without loss of generality [BK98b], we work with a Herbrand instantiation of the language where the Herbrand base $\mathcal{B}$ is the set of all ground atoms that can be constructed with the functions and constants of the language, and a Herbrand structure is any subset of $\mathcal{B}$.

To build complex logical formulas, $\mathcal{TR}$ uses the classical logic connectives $\wedge, \vee, \neg, \rightarrow$ and a new connective $\otimes$, called the *serial conjunction* operator. Informally, the formula $\phi \otimes \psi$ represents an action composed of an execution of $\phi$ followed by an execution of $\psi$. Additionally, $\phi \wedge \psi$ defines the action of executing simultaneously $\phi$ and $\psi$; while $\phi \vee \psi$ defines the non-deterministic choice of either executing $\phi$, $\psi$, or both simultaneously. Finally $\phi \leftarrow \psi$ says that one way to satisfy the execution of $\phi$ is by executing $\psi$. Then a $\mathcal{TR}$ program is a set of rules of the form $h \leftarrow \phi$, where $h$ is an atom of the language and $\phi$ is any complex formula.

A key feature of $\mathcal{TR}$ is the separation of elementary operations from the logic of combining them. To achieve this, $\mathcal{TR}$'s theory is parametric to two different oracles allowing the incorporation of a wide variety Knowledge Base (KB) semantics, from classical to non-monotonic to various other non-standard logics. The goal of these oracles is to abstract the representation of KB states and how to query them – encapsulated in the data oracle $\mathcal{O}^d$; but also to abstract the way states change – defined by the transition oracle $\mathcal{O}^t$.

As a result of this design, the language of primitive queries and actions is not fixed,

and neither is the definition of what is a state. To distinguish between states, $\mathcal{TR}$ works with a set of state identifiers, each uniquely identifying a state. Then the *data oracle* $\mathcal{O}^d$ is a mapping from state identifiers to sets of formulas. Intuitively, given a state identifier $i$, $\mathcal{O}^d(i)$ returns the set of formulas true in state $i$. The *state transition oracle* $\mathcal{O}^t(i_1, i_2)$ is a function that maps pairs of KB states into sets of ground atoms called elementary transitions. Intuitively, given two state identifiers $i_1$ and $i_2$, $\mathcal{O}^t(i_1, i_2)$ returns the set of elementary transitions that make the KB change from state $i_1$ into $i_2$.

It is important to notice that these data and transition oracles are strongly related, as particularly, the state identifiers of these two oracles are defined in the same domain. To provide for a better understanding of these oracles and how they can be instantiated, we present some examples of data and transition oracles taken from [BK95].

**Relational Oracle** A state identifier $D$ is a set of ground atomic formulas, and the data oracle simply returns all these formulas, i.e., $\mathcal{O}^d(D) = D$. Moreover, for each predicate symbol $p$ in $D$, the transition oracle defines two new predicates, $p.ins$ and $p.del$, representing the insertion and deletion of single atoms, respectively. Formally, $p.ins \in \mathcal{O}^t(D_1, D_2)$ iff $D_2 = D_1 + \{p\}$. Likewise, $p.del \in \mathcal{O}^t(D_1, D_2)$ iff $D_2 = D_1 - \{p\}$. SQL-style bulk updates can also be defined by the transition oracle as shown in [BK98b].

**Well-Founded Oracle** A state id $D$ is a set of generalized Horn rules and $\mathcal{O}^d(D)$ is the set of literals in the well-founded model of $D$. Such oracles can represent any rule-base with well-founded semantics, which includes Horn rule-bases, stratified rule-bases, and locally-stratified rule-bases. For advanced applications, one may want to augment $\mathcal{O}^d(D)$ with rules in $D$. The transition oracle provides primitives for adding and deleting clauses to/from states.

**Generalized-Horn Oracle** A state $D$ is a set of generalized Horn rules and $\mathcal{O}^d(D)$ is a classical Herbrand model of $D$. Such oracles can represent Horn rule-bases, stratified rule-bases, locally-stratified rule-bases, rule-bases with stable-model semantics, or any rule-base whose meaning is given by a classical Herbrand model. Again, one may want to augment $\mathcal{O}^d(D)$ with rules in $D$.

Note that, although in the previous examples state identifiers are defined by sets of formulas, nothing precludes a state identifier to be a set of natural numbers, or some non-logical objects like an XML file. In addition, since a state identifier uniquely identifies a state, from this moment onwards we use the terms of "state" and "state identifier" interchangeably.

Importantly, to provide a logical account to reason about the execution of actions and transactions on abstract knowledge bases, satisfaction in $\mathcal{TR}$ is not related to what formulas hold in what states (as this is encapsulated by the oracles), but rather on how actions can be executed in a transactional way. More precisely, satisfaction of $\mathcal{TR}$ formulas means *execution*, and a formula is said to be true if it can be executed successfully, as

a transaction.

Notice that $\mathcal{TR}$ is different from most logic systems in the literature of reasoning about actions, which are normally limited to reasoning about what formulas are true in a KB (as e.g. in [GL88; BCMNPS03; GRS91]), or about the direct and indirect effects resulting from a given (trans)action in a KB (as e.g. in [GL98a; KS86; McC63; Thi97]). As such, besides being able to reason about general actions (as e.g. shown in [BK94; RK12]), $\mathcal{TR}$ can also talk about *how* a transaction can be executed in a KB. Consequently, and contrary to these solutions, formulas in $\mathcal{TR}$ are not evaluated on states but on *paths*, i.e., on sequences of states of the form $\langle D_1, \ldots, D_n \rangle$, where each $D_i$ represents a state. Then, $\mathcal{TR}$'s semantics deals with statements of the form $P, \langle D_1, D_2, ..., D_{n-1}, D_n \rangle \models t$ meaning that: transaction $t$ succeeds in program $P$ when executed in the state $D_1$ by changing the system into state $D_n$ through the *path*: $D_1, D_2, ..., D_{n-1}, D_n$.

**Example 5** ($\mathcal{TR}$ Financial Transactions)**.** *To illustrate $\mathcal{TR}$, consider a bank's KB (taken from [BK95]) defined by a relational database and where the balance of a bank account is given by the relation $balance(Acnt, Amt)$. To modify this relation we assume a relational oracle as described above, where we are provided with a pair of elementary update operations, namely: $balance(Acnt, Amt).del$ to delete a tuple from the relation, and $balance(Acnt, Amt).ins$ to insert a tuple into the relation. Using these two update primitives, we define four transactions: $changeBalance(Acnt, Bal, Bal')$ to change the balance of an account; $withdraw(Amt, Acnt)$ to withdraw an amount from an account in case the amount to transfer is less than the account's balance; $deposit(Amt, Acnt)$ to deposit an amount into an account, and finally, the transaction $transfer(Amt, Acnt, Acnt')$ for transferring an amount from one account to another. These transactions can be defined in $\mathcal{TR}$ in a logic programming style by the following four rules and where the operator $\otimes$, serial conjunction, denotes the sequencial execution of transactions.*

$$
\begin{aligned}
transfer(Amt, Acnt, Acnt') &\leftarrow withdraw(Amt, Acnt) \otimes deposit(Amt, Acnt') \\
withdraw(Amt, Acnt) &\leftarrow balance(Acnt, B) \otimes B \geq Amt \otimes \\
&\quad\ changeBalance(Acnt, B, B - Amt) \\
deposit(Amt, Acnt) &\leftarrow balance(Acnt, B) \otimes changeBalance(Acnt, B, B + Amt) \\
changeBalance(Acnt, B, B') &\leftarrow balance(Acnt, B).del \otimes balance(Acnt, B').ins
\end{aligned}
$$

*Intuitively, the first rule states that a transfer of amount $Amt$ from account $Acnt$ to account $Acnt'$ is performed if first a withdrawal of $Amt$ from $Acnt$ is performed, and then a deposit of the same amount to $Acnt'$ is performed. The last rule states that changing the balance of account $Acnt$ from $B$ to $B'$ is true (in a sequence of knowledge base states) in case first the truth of $balance(Acnt, B)$ is deleted from the knowledge base according to the update-oracle, and then $balance(Acnt, B')$ is inserted.*

In the example above, $P, \langle d_1, d_2, d_3 \rangle \models transfer(10, ac1, ac2)$ holds, if $d_1$ is e.g. a state where $balance(ac1, 20)$ and $balance(ac2, 30)$ are true, $d_2$ is a state obtained from $d_1$ by deleting $balance(ac1, 20)$ and adding $balance(ac1, 10)$, and $d_3$ is obtained from $d_2$ by deleting $balance(ac2, 30)$ and adding $balance(ac2, 40)$. In fact, in program $P$, according to

$\mathcal{TR}$, the transaction $transfer(10, ac1, ac2)$ is entailed on any path where the initial state the balance is at least 10, the balances are changed accordingly, and $transfer(10, a1, a2)$ is not entailed on any other path. In other words, the paths that entail the transaction are exactly those that correspond to its complete (atomic) execution. Note that, though in the example above, each defined transaction has just one rule, $\mathcal{TR}$ allows us to define several rules for a given transaction, making it also possible to deal with non-deterministic actions and transactions. In that case, several different paths may succeed (entail) a transaction from a given state, each corresponding to a non-deterministic choice.

## 3.1 $\mathcal{TR}$ Model Theory

As most logics, $\mathcal{TR}$ model theory is based on interpretation functions. An interpretation determines what atoms are true on what paths by defining mappings from paths to Herbrand structures. If $\phi \in M(\pi)$ then, in the interpretation $M$, path $\pi$ is said to be a valid execution for the formula $\phi$. Additionally, interpretations need to be compliant with the oracles, $\mathcal{O}^d$ and $\mathcal{O}^t$, which are specified as a parameter of the theory. In this context, the oracles define satisfaction of elementary primitives over states and state transitions which all interpretations must respect. By imposing such restriction on interpretations, we ensure that oracles' primitive formulas are always satisfied on the paths that the oracles define it as so.

**Definition 1** (Interpretations)**.** *An interpretation is a mapping $M$ assigning a classical Herbrand structure (or $\top$[1]) to every path. This mapping is subject to the following restrictions, for all states $D_i$ and every formula $\varphi$:*

1. $\varphi \in M(\langle D \rangle)$ *if* $\mathcal{O}^d(D) \models \varphi$
2. $\varphi \in M(\langle D_1, D_2 \rangle)$ *if* $\mathcal{O}^t(D_1, D_2) \models \varphi$

Subsequently, for defining satisfaction of complex formulas over paths, we have first to introduce some basic operations on paths. For instance, recall that the formula $\phi \otimes \psi$ means the complex (trans)action of executing $\phi$ followed by the execution of $\psi$. Thus, this formula is true (i.e., successfully executes) on a path that executes $\phi$ up to some point in the middle, and executes $\psi$ from that point onwards. To deal with this $\mathcal{TR}$ provides the notion of path split:

**Definition 2** (Path Splits)**.** *A* split *of a path $\pi = \langle D_1, \ldots, D_k \rangle$ is any pair of subpaths, $\pi_1$ and $\pi_2$, such that $\pi_1 = \langle D_1, \ldots, D_i \rangle$ and $\pi_2 = \langle D_i, \ldots, D_k \rangle$ for some $i$ $(1 \leq i \leq k)$. In this case, we write $\pi = \pi_1 \circ \pi_2$.*

[1]For not having to consider partial mappings, besides formulas, an interpretation can also return the special symbol $\top$. Whenever $M(\pi) = \top$, then the interpretation $M$ is said to satisfy every formula in path $\pi$. This guarantees that $M$ maps every possible path and is useful to identify inconsistencies. The interested reader is referred to [BK95] for more details.

Building on the notion of path splits and interpretations, we can now define the general satisfaction of formulas in $\mathcal{TR}$ as follows.

**Definition 3** ($\mathcal{TR}$ Satisfaction of Formulas)**.** *Let $M$ be an interpretation, $\pi$ a path and $\phi$ a formula. If $M(\pi) = \top$ then $M, \pi \models \phi$; otherwise:*

1. ***Base Case:*** *$M, \pi \models \phi$ iff $\phi \in M(\pi)$ for any atom $\phi$*
2. ***Negation:*** *$M, \pi \models \neg\phi$ iff it is not the case that $M, \pi \models \phi$*
3. ***"Classical" Conjunction:*** *$M, \pi \models \phi \wedge \psi$ iff $M, \pi \models \phi$ and $M, \pi \models \psi$.*
4. ***Serial Conjunction:*** *$M, \pi \models \phi \otimes \psi$ iff $M, \pi_1 \models \phi$ and $M, \pi_2 \models \psi$ for some split $\pi_1 \circ \pi_2$ of path $\pi$.*

In the sequel we also mention the satisfaction of disjunctions and implications, where as usual $\phi \vee \psi$ means $\neg(\neg\phi \wedge \neg\psi)$, and $\phi \leftarrow \psi$ means $\phi \vee \neg\psi$ (i.e., $\neg(\neg\phi \wedge \psi)$ ).

**Example 6** ($\mathcal{TR}$'s Model Theory)**.** *Assume a Relational Database Oracle as defined previously. Since every interpretation $M$ needs to be compliant with the oracles then for every $M$, $a.ins \in M(\langle\{\}, \{a\}\rangle)$ and $b.ins \in M(\{a\}, \{a, b\})$.*

*Moreover, given the above definition of satisfaction, for every interpretation $M$, it holds that $M, \langle\{\}, \{a\}\rangle \models a.ins$ and that $M, \langle\{\}, \{a\}, \{a, b\}\rangle \models a.ins \otimes b.ins$.*

## 3.2 $\mathcal{TR}$ Logical Entailment

After defining how interpretations satisfy complex formulas given a path, we now define which of these interpretations model a formula and a program.

**Definition 4** (Models)**.** *An interpretation $M$ is a model of a $\mathcal{TR}$ formula $\phi$ if $M, \pi \models \phi$ for every path $\pi$. In this case, we write $M \models \phi$. An interpretation is a model of a set of formulas if it is a model of every formula in the set.*

This notion of models is mostly used together with the notion of program. In it, a program is a set of formulas of the form $h \leftarrow \phi$ where $h$, the rule's head, is an atom in the language and $\phi$, the rule's body, is any complex formula. Since rules are just complex formulas, an interpretation $M$ models a program $P$ iff it models all its rules, where $M$ models a rule $h \leftarrow \phi$ iff, for every path $\pi$, whenever $M$ satisfies the body $\phi$ it also satisfies the head $h$ (or, equivalently, $M$ either satisfies the head $h$ or it does not satisfy the body $\phi$).

**Example 7.** *Assume a Relational Database Oracle as defined previously, and the following program $P$.*

$$\mathbf{P}: \quad \begin{array}{l} p \leftarrow a.ins \\ q \leftarrow b.ins \\ q \leftarrow c.ins \\ t \leftarrow p \otimes q \end{array}$$

*For every interpretation $M$ that models $P$ it is true that $M, \langle\{\}, \{a\}\rangle \models p$ and $M, \langle\{\}, \{a\}, \{a, b\}\rangle \models t$. Moreover, it is also true that $M, \langle\{\}, \{a\}, \{a, c\}\rangle \models t$*

Based on this notion of models, it is also possible to define the notion of entailment in the usual way.

**Definition 5** (Logical Entailment)**.** *Let $\phi$ and $\psi$ be two $\mathcal{TR}$ formulas. Then $\phi$ entails $\psi$ if every model of $\phi$ is also a model of $\psi$. In this case we write $\phi \models \psi$.*

## 3.3 Executional Entailment and Proof Theory

Besides the concept of a model of a $\mathcal{TR}$ theory, which allows one to prove properties of the theory independently of the paths chosen, $\mathcal{TR}$ also defines the notion of executional entailment. A transaction is entailed by a theory given an initial state, if there is a path starting in that state, and the transaction succeeds on that path. As such, given a transaction and an initial state, the executional entailment determines the path that the KB should follow in order to succeed the transaction. Non-deterministic transactions are possible, and in this case several successful paths exist.

This notion is formalized as follows.

**Definition 6** (Executional Entailment)**.** *Let $P$ be a program, $\phi$ be a formula and $\langle D_1, \ldots, D_n\rangle$ be a path:*

$$P, \langle D_1, \ldots, D_n\rangle \models \phi \tag{3.1}$$

*holds if $M, \langle D_1, \ldots, D_n\rangle \models \phi$ for every model $M$ of $P$. We write $P, D_1 - \models \phi$ when there exists a path $\langle D_1, \ldots, D_n\rangle$ for which (3.1) holds.*

**Example 8.** *Recall example 7. Here we have that:*

$$P, \langle\{\}, \{a\}\rangle \models p$$
$$P, \langle\{a\}, \{a, b\}\rangle \models q$$
$$P, \langle\{\}, \{a\}, \{a, b\}\rangle \models t$$

*Moreover, this latter entailment intuitively states that, given the specifications in $P$, transaction $t$ successfully executes over a path where initially the KB is empty, then it has the fact $a$, and finally the facts $a$ and $b$.*

*One can also check that:*

$$P, \langle\{a\}, \{a, c\}\rangle \models q$$
$$P, \langle\{\}, \{a\}, \{a, c\}\rangle \models t$$

*making $t$ and $q$ non-deterministic transactions.*

Based on this definition, $\mathcal{TR}$ further provides a proof theory that is sound and complete w.r.t. executional entailment, and a corresponding implementation to a special class of $\mathcal{TR}$ theories known as serial-Horn programs [BK93]. A serial-Horn program $P$ is a set of serial-Horn rules of the form $h \leftarrow b_1 \otimes \ldots \otimes b_n$ where every $b_i$ is an atom and $n \geq 0$.

This proof theory shares some similarities with the SLD-Resolution proof strategy for logic programs [Llo87]. Its goal is to construct a path that corresponds to a valid execution of formula $G$, i.e., a path $\langle D_0, D_1, \ldots, D_n \rangle$ for which $P, \langle D_0, D_2, \ldots, D_n \rangle \models G$ holds.

This derivation is parametric w.r.t. the database and transition oracles, $\mathcal{O}^d$ and $\mathcal{O}^t$ which provide the semantics for querying and updating a given state $D$.

**Definition 7** (Proof Theory for $\mathcal{TR}$ Programs)**.** *Let $P$ be a $\mathcal{TR}$ serial-Horn program and $D, D_0, D_1, D_2$ states. Let $G$ be a serial-Horn goal of the form $b_1 \otimes \ldots \otimes b_k$, where every $b_i$ is an atom and $k \geq 0$. The procedure deals with* sequents *of the form $P, D- \vdash G$. The special propositional constant $()$ expresses a tautological formula true in every path of length 1.*

*A derivation $P \cup \{G\}$ consists of a finite or infinite sequence of sequents $seq_1, seq_2, \ldots, seq_n$ where $seq_1 = P, D_0 - \vdash G$ and each $seq_i$ is either an axiom sequent or is derived from the earlier sequents by the following rules.*

**Axioms:** $P, D- \vdash ()$

**Inference Rules:** *Let $a$ be an atomic formula, and $\phi$ and $rest$ be serial goals.*

1. Applying transaction definitions*:*
   *Let $a \leftarrow \phi$ be a rule in $P$, then*
   $$\frac{P, D- \vdash \phi \otimes rest}{P, D- \vdash a \otimes rest}$$
2. Querying the knowledge base:
   *Let $\mathcal{O}^d(D) \models a$, then*
   $$\frac{P, D- \vdash rest}{P, D- \vdash a \otimes rest}$$
3. Performing elementary updates:
   *Let $\mathcal{O}^d(D_1, D_2) \models a$, then*
   $$\frac{P, D_2- \vdash rest}{P, D_1- \vdash a \otimes rest}$$

*Based on this, for a given goal $G$ an executional derivation (or proof) is said to be* successful *if the sequence $seq_1, seq_2, \ldots, seq_n$ is finite and ends in the axiom sequent $()$. In this case, we write:*

$$P, \langle D_0, D_1, \ldots, D_n \rangle \vdash G$$

*where $D_0, D_1, \ldots, D_n$ corresponds to the sequence of states appearing respectively in every sequent of the derivation.*

**Theorem 1** (Soundness and Completeness [BK95])**.** *Let $G$ be a serial-Horn goal, let $D_0, D_1, \ldots, D_n$ be a path and let $P$ be a serial-Horn program.*

$$P, \langle D_0, D_1, \ldots, D_n \rangle \vdash G \;\; \textit{iff} \;\; P, \langle D_0, D_1, \ldots, D_n \rangle \models G$$

# Part II

# Modeling transactions with external actions

# 4

# Motivation: why do we need external actions and transactions

In this chapter we introduce the problem arising when one needs to execute external actions (i.e., actions executed in an external environment or external KB), together with internal actions in a transactional way.

This type of behavior, where one needs to interact with both an internal and external environment and provide properties regarding the outcome of executing actions in each environment, appears in several applications domains, like e.g. in Semantic Web applications or multi-agent systems. For instance, intelligent agents in a multi-agent setting must work and reason over a two-fold environment: an external environment, representing the outside world where the agent acts, and which may include other agents; and an internal environment comprising the information about the agent's rules of behavior, preferences about the outside world, its knowledge and beliefs, intentions, goals, etc. In such a context, an agent may act on the external environment (by executing external actions), but also on the internal environment (where it executes internal actions).

Moreover, while in general it is considerably easy to ensure that internal actions are executed in a transactional way, it may also be important to ensure *some* properties regarding the outcome of the external actions performed externally. As an example of this requirement, imagine the scenario where one needs to prepare a weekend abroad in either Lisbon or London, and this preparation requires scheduling a flight with an airline but also booking a hotel room in the city of destination. Moreover, for the sake of this example, imagine that, for that weekend, there are no available hotels in Lisbon (e.g., because it was the weekend of the UEFA Champions League Final). If we have already

started to prepare the weekend and have already scheduled a flight to Lisbon, then something must be done concerning the previously booked flight, so we do not end up paying for a flight that will not be used.

However, it is normally impossible to guarantee the full transactional properties regarding the execution of external actions. In particular, atomicity is the first of the four basic ACID properties that a transaction needs to guarantee during execution. This property states that, either the whole set of actions of a transaction can be executed, or the database is left unchanged as if nothing happened. In practical terms, this property is achieved by performing rollback operations. If a set of actions is issued for execution and something fails in the middle of this execution, then the rollback mechanism is what allows us to restore the previous consistent database state that was true before the execution of these actions.

The main issue of external actions is that, since we cannot control the *external* environment where these actions were executed, rolling back is no longer possible. More precisely, since we do not own the external KB where these actions are executed, we cannot simply restore a state before the execution of these actions. To address this impediment, the work of [GMS87] introduced the notion of long-lived transactions, or sagas. These denote transactions that require interactions with an external entity, which may last for relatively long periods of time and delay the termination of shorter and more common transactions. For such situations, where rolling back is impractical or impossible, the authors of [GMS87] propose the idea of *compensations*. Then, every external operation is defined together with a compensating action that reverts the effects of the initially performed action. As such, upon external failure, if we execute these compensations in a reverse order, then we can achieve a consistent database state equivalent to the initial one before the execution of external actions, and achieve a relaxed model of external atomicity.

In this part we propose an extension of $\mathcal{TR}$ where transactions can involve actions executed in an internal and external domain. As we shall see, this extension guarantees that actions performed internally are always rolled back, while actions executed externally follow a relaxed model of atomicity, achieved by using compensating operations.

Since our work is based on the one of Transaction Logic, next we elaborate on why $\mathcal{TR}$ is indeed unsuitable to deal with this problem, and what are the changes needed in both its model theory and proof procedure, to properly address external actions.

## 4.1 Difficulties and limitations of Transaction Logic with external actions

Recall that $\mathcal{TR}$'s semantics works by constructing the paths where a transaction succeeds without failures. In other words, $\mathcal{TR}$ only considers the paths where the execution of a

transactions completely, and atomically, succeeds. This is acceptable because, when executing internal actions, we can assume to have a complete control over the KB, and thus we can assume that it is always possible to restore any state prior to any (internal) execution try. Consequently, the execution where the system makes a choice to execute an internal action, fails, rolls back, and succeeds by executing another action as an alternative, is just equivalent to the execution where the "right" choice is made directly.

However, when modeling the execution of external actions, the failed attempts to execute transactions cannot simply be disregarded. Contrary to a simple rollback, such execution is not equivalent to choosing the right path directly, as it requires an additional interaction with the external world that needs to be reflected in the final path. To better illustrate the problem, consider the following examples.

**Example 9** (Weekend)**.** *Recall the previous mentioned example of a weekend arrangement. For that weekend we assume three different possibilities: either go to Lisbon, to London, or stay at home. To go either to Lisbon or London, we have to make a reservation for the corresponding travel, this comprising both the booking of a hotel room and a flight. These transactions can be written (in a very simplified way) in a $\mathcal{TR}$-like form as follows:*

$$
\begin{array}{lcl}
weekendPrep(lisbon) & \leftarrow & bookTravel(lisbon, D). \\
weekendPrep(london) & \leftarrow & bookTravel(london, D). \\
weekendPrep(home). & & \\
bookTravel(City, D) & \leftarrow & findHotel(City, D, H) \otimes findFlight(City, D, F) \otimes \\
 & & reserveHotel(H, D) \otimes reserveFlight(F, D)
\end{array}
$$

*where $reserveHotel(H, D)$ is an action performed externally, e.g. by introducing a tuple corresponding to the reservation in an external KB about hotels, and similarly for flights.*

*Moreover, let us imagine that, for that weekend, there are no available hotels in Lisbon, but also no available flights to London (i.e., the corresponding transactions for Lisbon and London are not entailed in any path). According to $\mathcal{TR}$, in this situation there is a single path entailing the transaction for my weekend preparation, in which I end up staying at home, and nothing changes. Clearly, there is nothing wrong with the success of this transaction on the path where nothing changes. The problem is how, in practice, can a system come up with this only path for succeeding the transaction. Since there are several possible ways to prove $weekendPrep(X)$, any practical system could have tried the several alternatives that in the end fail to be entailed, until a correct one is possibly found.*

*When considering changes just in an internal KB, as is the case of $\mathcal{TR}$, this does not cause a problem. In fact, since one has full control over its own internal KB, in practice it should be possible to roll back the to internal KB state just before that transaction was tried and failed, and then to try another alternative execution available.*

*However, when changes are done in external environments, that is no longer possible. If when trying to find a path for succeeding the transaction of preparing my weekend, the transaction $bookTravel(london, w)$ fails because there is no flight available, then something must be done*

*about the previously reserved hotel, before another possibility is tried. But since the KB taking care of hotel reservations is external, rolling back might not even be an option. For example, a money penalty may be associated for canceling a room reservation, in which case the rollback of* $reserveHotel(H, D)$ *could not simply be the deletion of the added tuple (and which, in principle, one does not have permission to change directly).*

While in this previous travel example, all the actions executed are external, in general an interaction interleaving internal and external actions is required:

**Example 10** (Product request)**.** *Consider now a KB of a given organization storing information about customers, sales, etc. Moreover, assume that this organization needs to interact with other organizations, customers, suppliers, via web-services, or even by prompting external users to provide information. In this scenario, we may want to define a transaction for satisfying a customer's request for an amount of a product. Such a transaction could (again, in a quite simplified way) be expressed in a* $\mathcal{TR}$*-like form as follows:*

$$request(Prd, N, Cust) \leftarrow decreaseStock(Prd, N) \otimes dispatch(Prd, N, Cust)$$

*where* $decreaseStock(Prd, N)$ *is an internal update of decreasing the stock of product* $Prd$ *by* $N$ *(failing when* $N$ *is greater than the current stock), and* $dispatch(Prd, N, Cust)$ *is the action of dispatching* $N$ *units of product* $Prd$ *to customer* $Cust$*. In this case, if the dispatch action fails then, before any other possibility to satisfy the request is attempted, the update of decreasing the stock must be rolled back.*

*Now consider that, another way the organization has to satisfy the request is by asking an associated company whether it has the product, asking the customer whether she accepts that the product is supplied by that other company, and requesting the company to send it to the customer:*

$$\begin{aligned} request(Prd, N, Cust) \leftarrow \quad & askComp(Prd, N) \otimes askCust(Cust, Prd) \otimes \\ & requestDisp(Prd, N, Cust) \end{aligned}$$

*Here, if the action of asking the customer fails (e.g. because she does not accept it), then unlike the update of decreasing the stock, the action of asking the company cannot be rolled back. Note that, this action can have long lasting effects on the associated company (e.g. by reserving the product). However, since the organization does not control the KB of the associated company, all it can do is to signal that the customer did not accept it. Of course, this would have to be coded in the transaction, and in our proposal it is done by replacing* $askComp(Prd, N)$ *in the rule by e.g.* $\mathbf{ext}(askComp(Prd, N), forget(Prd, N))$*.*

*When putting these two rules together, one would expect the transaction to succeed in case the associated company has the product, the customer accepts it, and the product is dispatched by the associate, or by decreasing the stock and dispatching the product. Moreover, the transaction should also succeed on a path where the associate is asked, the customer does not accept the change, the associate is notified to forget about it, and finally by decreasing the stock and dispatching the product. This latter path, considering the possibility of failed attempts to execute a transaction,*

*can never be obtained by $\mathcal{TR}$.*

Consider another example, in the context of intelligent agents, which was previously illustrated in fig. 1.2 of chapter 1, requiring more elaborate Kbs.

**Example 11** (Diagnosis example)**.** *Imagine the scenario of an agent with the goal to help in the triage process of an emergency room. For that, the agent's internal KB is defined by an ontology comprising medical information about diseases, medication and so on. Externally, the agent needs to interact with the patient: check her temperature for fever, heart rate, blood pressure, etc. and eventually give medication for her condition. If the agent is able to infer the treatment to be performed and give the patient some medication, then the patient is put in the low priority list.*

*However, every medication can have adverse side-effects that, when present, need to be addressed immediately. If that is the case, then the internal information about the patient's priority must change, and something must be given to the patient to counter such side-effects.*

The previous examples motivate that, in contrast to what happens in $\mathcal{TR}$, a logic for specifying and reasoning about transactions in an internal KB together with an external environment, cannot ignore the failed attempts to execute transactions, whenever such attempts involved the execution of external actions.

To address this issue, next we propose $\mathcal{ETR}$, an extension of $\mathcal{TR}$ to reason and execute transactions on knowledge bases partitioned between an internal KB and an external environment. This ability to execute external actions together with internal updates is $\mathcal{ETR}$'s main innovation, and what is missing in the original $\mathcal{TR}$. While actions performed in the internal KB can always be rolled back, actions executed externally need to follow a relaxed model of atomicity, which in $\mathcal{ETR}$'s case is based on compensations.

With this in mind, in the following we propose $\mathcal{ETR}$'s theory (chapter 5) along with its syntax (section 5.1), model theory (section 5.3) and proof procedure (section 5.5). We then elaborate on what external oracle definitions can be useful for the envisioned application domains (chapter 6). Finally, for a particular external oracle definition based on Action Languages, we show how $\mathcal{ETR}$ can be used to automatically compute the correct compensations to be performed during execution (chapter 7).

# 5

# $\mathcal{ETR}$: Extending Transaction Logic with External Actions

As shown in the previous chapter, Transaction Logic is not suitable for situations where a transaction needs (besides other things) to execute actions in an external domain. To address this need, in the following we present *External Transaction Logic* ($\mathcal{ETR}$), a logic to reason about and execute transactions that involve internal and external actions. Most of the results shown here were published in [GA11; GA13a].

When compared with the original Transaction Logic, $\mathcal{ETR}$ offers two main differences: the possibility to interact with an external domain, and the possibility to talk about the paths where even though the transaction fails, it is possible to recover from this failure (by rolling back internally and compensate externally), and then succeed on an alternative branch.

As in Transaction Logic, $\mathcal{ETR}$ requires the existence of oracles to define the semantics and basic primitive operations of the internal KB. Additionally, to address the interaction with an external environment, $\mathcal{ETR}$ assumes an *external oracle* ($\mathcal{O}^e$) to define the behavior of the external environment. As it happens with the internal KB and the internal oracles, this design allows $\mathcal{ETR}$ to reason and execute transactions that require interaction with external sources without committing to any particular semantics for the external environment. As we shall see, this external oracle can then, e.g. be instantiated with Description Logics [BCMNPS03] semantics, or with logics for dynamic external domains like Action Languages [GL98a] or Event Calculus [KS86], making $\mathcal{ETR}$ suitable for a wide range of scenarios. Based on this, transactions are defined by the composition of internal and external actions in a logic-programming style, allowing the specification of programs that

integrate knowledge and actions from multiple sources and semantics.

Moreover, another important difference of $\mathcal{ETR}$, when compared to $\mathcal{TR}$, is the possibility to talk about *failed* paths. To provide for a better understanding of this, please consider the following example.

**Example 12.** *Assume the following $\mathcal{TR}$ program $P$ where $external\_a$, $external\_b$ and $external\_c$ are actions performed externally.*

$$t \leftarrow p.ins \otimes external\_a \otimes external\_b$$
$$t \leftarrow q.ins \otimes external\_c$$

*In $\mathcal{TR}$, transaction $t$ has two non-deterministic ways to succeed: if the insertion of predicate $p$ followed by the actions $external\_a$ and $external\_b$ succeed; or if the insertion of predicate $q$ followed by $external\_c$ succeed.*

*However, let's assume that $external\_b$ fails after the execution of $external\_a$. Then, $t$ is only satisfied on the path where $external\_c$ is performed after $q.ins$ (2nd rule).*

*Yet, the 1st rule defines an alternative way for executing transaction $t$, and thus it can be non-deterministically selected as a legitimate try to succeed it (in $\mathcal{TR}$'s proof theory). If this is the case, the actions $p.ins$ and $external\_a$ are meant to be rolled back (in the implementation version of the proof theory procedure), and this execution is considered to have never happened. However, since $external\_a$ corresponds to an external action (e.g. a request to a web-service) it may simply be impossible to roll back such an action. Nevertheless, succeeding $t$ in that state may still be possible. For that we need to compensate for $external\_a$ (e.g. send a message to cancel the previous request), roll back $p.ins$ and then execute the 2nd rule.*

As previously stated, the previous example shows that $\mathcal{TR}$'s model and proof theory can only center on the paths where a formula completely succeeds without failures. For instance, in the definition of $\mathcal{TR}$'s proof theory (definition 7), we say that a transaction $t$ succeeds in a path $\pi$ if we can construct a proof by making the "right" choices of execution non-deterministically. Obviously, when building proofs (and also, when executing transactions), one cannot assume that the (non-deterministic) choice is always the "right" one. But this causes no problems when executing actions over an internal KB (such as, e.g., a database), because we have a complete control over the KB. Or more precisely, because we can assume that it is always possible to restore any state before any execution is tried. Hence, from an implementation perspective, whenever the system makes a choice that leads to a non-successful derivation, then it simply rolls back to the state previous to that choice, and tries to succeed on an alternative branch. From the proof theory perspective, this execution, where a rollback is performed, is equivalent to the one where the right path was chosen directly.

However, when dealing with external environments and external actions, this is no longer the case, and we need to compensate for the external actions already executed before the failure. The main issue here is to find the legitimate paths were a transaction fails, since as we shall see, not all paths where a transaction fails need to be recovered.

In a nutshell, to precisely deal with failures, $\mathcal{ETR}$'s model theory provides three satisfaction relations for a given interpretation $M$:

**Classical Satisfaction** Equivalent to $\mathcal{TR}$'s satisfaction of formulas but integrating paths with an external component. $M, \pi \models_c \phi$ if $\phi$ can execute on $\pi$ without failures.

**Partial Satisfaction** Provides the first ingredient to define failures. $M, \pi \models_p \phi$ if either $\phi$ succeeds without failures (i.e., if $M, \pi \models_c \phi$) or if it fails because a primitive action in $\phi$ cannot be executed in a given state.

**General Satisfaction** Corresponds to the real satisfaction of formulas making use of the previous two notions. $M, \pi \models \phi$ if $\phi$ succeeds classically over path $\pi$ or; if we can split $\pi$ into $\pi = \pi_1 \circ \pi_2$ such that $\phi$ fails and recovers from this failure on $\pi_1$ (by rolling back internally and compensating externally) and succeeds on $\pi_2$.

The first two satisfaction relations represent the building blocks for defining failures and are *not* used to satisfy formulas directly. As it shall be precisely defined, a formula $\phi$ is said to fail in a path $\pi$ if $\phi$ can be partially satisfied but not classically satisfied (i.e., if $M, \pi \not\models_c \phi$ but $M, \pi \models_p \phi$). If this is the case, then recovery is in order, and for that we need to roll back internally and compensate externally. This is denoted as $M, \pi \rightsquigarrow \phi$ meaning that $\pi$ is a recovery path obtained after failing to execute $\phi$ and executing actions externally.

We continue by introducing $\mathcal{ETR}$'s syntax and external oracle (sections 5.1 and 5.2). Then, we define $\mathcal{ETR}$'s model theory and executional entailment by providing a precise meaning for these relations (sections 5.3 and 5.4). And finally, we introduce a proof procedure that we prove to be sound and complete w.r.t. $\mathcal{ETR}$'s semantics (section 5.5).

## 5.1 $\mathcal{ETR}$ Syntax

To deal with external environments and external actions, $\mathcal{ETR}$ operates over a KB including both an internal and an external component. For that, formally $\mathcal{ETR}$ works over two disjoint propositional languages: $\mathcal{L}_P$ (program language), and $\mathcal{L}_\mathcal{O}$ (oracles primitives language). Propositions in $\mathcal{L}_P$ denote actions and fluents that can be defined in the program. As usual, fluents are propositions that can be evaluated without changing the state and actions are propositions that cause evolution of states. Propositions in $\mathcal{L}_\mathcal{O}$ define the primitive actions and queries that deal with the internal and external KB. $\mathcal{L}_\mathcal{O}$ can still be partitioned into $\mathcal{L}_i$ and $\mathcal{L}_a$, where $\mathcal{L}_i$ denotes primitives that query and change the internal KB, while $\mathcal{L}_a$ defines the external actions, i.e. the primitives that can be executed externally. For convenience, it is assumed that $\mathcal{L}_a$ contains two distinct actions `failop` and `nop`, defining trivial failure in the external domain, and trivial success in the external domain without changing the external state, respectively.

As previously mentioned, one of the novelties in $\mathcal{ETR}$ is the computation of some paths where a formula is *not* successful, but where some external action was executed and

now needs to be compensated. To be better explained below, the difficulty of this notion is that there are several paths where a formula may fail, and not all of them correspond to a valid execution try. To be able to precisely deal with this, we restrict the language of $\mathcal{ETR}$ w.r.t. negation. More concretely, in our restricted language negation can only be applied to atoms:

**Definition 8** ($\mathcal{ETR}$ Atoms, Literals, and Formulas)**.** *An $\mathcal{ETR}$* atom *is either a proposition in $\mathcal{L}_P$, $\mathcal{L}_i$ or an external atom. An external atom is either a proposition in $\mathcal{L}_a$ or $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ where $a, b_i \in \mathcal{L}_a$. An $\mathcal{ETR}$ literal is either $\phi$ or $\neg\phi$ where $\phi$ is an $\mathcal{ETR}$ atom. An $\mathcal{ETR}$* formula *is either a literal, or an expression, defined inductively, of the form $\phi \wedge \psi$, $\phi \vee \psi$ or $\phi \otimes \psi$, where $\phi$ and $\psi$ are $\mathcal{ETR}$ formulas. We say that a formula is positive iff all its literals are atoms.*

Based on this definition, external actions can appear in a program in two different ways:

1) without any kind of associated compensation, i.e., $\mathbf{ext}(a, \texttt{nop})$, and in this case we also write it as $\mathbf{ext}(a)$ or simply $a$, where $a \in \mathcal{L}_a$ and;

2) with a user defined compensation, written $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ where $a, b_i \in \mathcal{L}_a$ ($1 \leq i \leq j$).

The former case should be used when defining external actions that do not need to be compensated, as for instance, external queries. Consequently, if $\mathbf{ext}(a, \texttt{nop})$ is executed but something fails afterwards, its compensation (i.e., `nop`) trivially succeeds without changing the external state. On the contrary, to define external actions that always fail to be compensated (as for instance, the action of printing a document) one should use the construct $\mathbf{ext}(a, \texttt{failop})$.

Note in case 2), that there is no explicit relation between $a$ and $b_1 \otimes \ldots \otimes b_j$, and that it is possible for external action $a$ to appear with different compensating actions in the same formula. It is thus the programmer's task to determine which is the correct compensation for action $a$ in a given point for a given rule.

To be able to refer to the external actions in formulas, we also define $\mathcal{L}_a^*$ as the augmentation of $\mathcal{L}_a$ with the formulas $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ where $a, b_i \in \mathcal{L}_a$.

The restriction on the use of negation, prevents us from defining $\phi \leftarrow \psi$ as $\phi \vee \neg\psi$[1], as it is done in $\mathcal{TR}$. However, we still want to be able to express this kind of statements in programs since, rules are instrumental for defining transactions. In fact, as the usage of $\mathcal{TR}$ shows, not having rules would surely be too severe a restriction. To overcome this, we still define programs as sets of rules of that form, where the head is always an atom from $\mathcal{L}_P$ and the body is any $\mathcal{ETR}$ formula, and then take especial care when defining satisfaction of rules (which is precisely defined in definition 20).

**Definition 9** ($\mathcal{ETR}$ Programs)**.** *An $\mathcal{ETR}$* program is a set of rules of the form $\phi \leftarrow \psi$ where $\phi$ is a proposition in $\mathcal{L}_P$ and $\psi$ is an $\mathcal{ETR}$ formula.

[1]Since $\psi$ is defined as a complex formula.

Before we advance to the definition of $\mathcal{ETR}$'s semantics, we give an example to illustrate what kind of transactions can be specified with this language in a domain where these transactions act both on an internal KB and on an external environment.

**Example 13.** *Recall example 11 where an agent has the task to triage patients in emergency rooms. To do so, the agent needs to interact with the patient, perform a small diagnosis, and assign a priority accordingly. The diagnosis and the priority decision is done internally, using the agent's internal KB defined by a Description Logic. Using this KB, the agent can identify simple cases of flu, and if so, provide treatment to the patient. If this is the case, the agent can give Phenylephrine (PLP) as a treatment. However, this treatment is not suitable for pregnant women or for people suffering from hypertension.*

*To encode this, imagine that we have a TBox which includes the following assertions (among many others) related to Flu and the PLP medicine:*

$$\begin{aligned}
&\text{Flu} \sqsubseteq \text{Fever} \sqcap \text{Headache} \sqcap \text{StuffyNose} \sqcap \neg\text{Serious}\\
&\text{PLPeligible} \sqsubseteq \neg\text{Pregnant}\\
&\text{PLPeligible} \sqsubseteq \neg\text{Hypertense}\\
&\text{StrongFever} \sqsubseteq \text{Serious}\\
&\text{HeartFailure} \sqsubseteq \text{Serious}
\end{aligned}$$

*To query and update this DL, the agent has the primitives* `dlquery()` *and* `dladd()` *respectively. A triage has three possible outcomes: green $g$, yellow $y$ and red $r$, respectively, ranging from low to high priority. Based on this, the process of triage of a given patient $X$ can be encoded in $\mathcal{ETR}$ by the following rules:*

$$\begin{aligned}
triage(X,r) &\leftarrow diagnosis(X) \otimes \texttt{dlquery}(\text{Serious(X)}) \otimes \texttt{dlquery}(\text{HeartFailure(X)})\\
triage(X,y) &\leftarrow diagnosis(X) \otimes \texttt{dlquery}(\text{Serious(X)}) \otimes \texttt{dlquery}(\neg\text{HeartFailure(X)})\\
triage(X,g) &\leftarrow diagnosis(X) \otimes \texttt{dlquery}(\neg\text{Serious(X)})\\
triage(X,g) &\leftarrow diagnosis(X) \otimes \texttt{dlquery}(\text{Flu(X)})) \otimes \texttt{dlquery}((\text{PLPeligible(X)}))\otimes\\
&\quad\ \mathbf{ext}(giveMeds(X,plp), giveMeds(X,cplp)) \otimes statsOK(X)\\
statsOK(X) &\leftarrow diagnosis(X) \otimes \texttt{dlquery}(\neg\text{Serious(X)})
\end{aligned}$$

*These (very simplified) rules state that, if we conclude that the patient's condition is serious and that she suffers from a heart failure, then she must be treated immediately, and thus her priority is defined as red. (1st rule). However, if her condition is serious but she does not show heart failure signs, then the patient's priority is defined as yellow (2nd rule). If the patient's condition is not serious then she is given the green priority (3rd rule). Additionally, if we can conclude that the patient has flu, and is eligible to receive the treatment, then the agent can give the patient some PLP medication (4th rule). However, if this medication is given, then the agent should ensure that the patient does not become worse afterwards. This is tested by $statsOK$ that re-performs the diagnosis and checks if the status of the patient has become serious (e.g. with the appearance of a strong fever or a heart failure). If this is the case, the call $statsOK$ will fail and the agent will give the patient a medicine to counter the effects of PLP (cplp). Then, depending on the patient*

*displaying symptoms of heart failure or not, the agent will employ the first or the second rule to assign the patient a higher priority. The expression* $\mathbf{ext}(giveMeds(X, plp), giveMeds(X, cplp))$ *defines an external action with compensation. In this case it says to give patient* $X$ *the medication* $plp$ *but, if something fails afterwards, to give* $cplp$ *as a compensation.*

*The agent assigns each value according to the results of the diagnosis. A diagnosis corresponds to a battery of tests to check the patient's condition. To perform it, the agent executes external actions to measure (query) the patient's stats. As queries, these actions do not have compensations and thus have the form* $\mathbf{ext}(temperature(X, Y))$ *(which returns the temperature* $Y$ *of patient* $X$*). A simplified version of a diagnosis can be encoded as follows:*

$$
\begin{array}{rcl}
diagnose(X) & \leftarrow & checkTemp(X) \otimes checkHeadache(X) \otimes \ldots \otimes checkHeartRate(X) \\
checkTemp(X) & \leftarrow & \mathbf{ext}(temperature(X, Y)) \otimes [(37 < Y < 41 \otimes \texttt{dladd}(\text{Fever(X)})) \vee \\
 & & (Y >= 41 \otimes \texttt{dladd}(\text{StrongFever(X)})) \vee (Y < 37 \otimes \texttt{dladd}(\neg\text{Fever(X)}))] \\
checkHeadache(X) & \leftarrow & \mathbf{ext}(hasHeadache(X, Y)) \otimes [(Y = \texttt{true} \otimes \texttt{dladd}(\text{Headache(X)})) \\
 & & \vee (Y = \texttt{false} \otimes \texttt{dladd}(\neg\text{Headache(X)}))]
\end{array}
$$

Note that the compensations for external actions are stated directly in the program. In this sense, it is the programmer's responsibility to state the right compensation for each case. As we shall see, this is necessary if the semantics of the external environment is left open. I.e., if we assume nothing about the semantics of states and updates in the external environment, then it is impossible to automatically infer what are the right actions to repair the effects of a particular action in a given KB.

However, if the semantics of the external environment is known, and formally defined in some logical language, then it may be possible to automatically infer from the external semantics what are the correct compensations for a given action. This notion is further explored ahead in chapter 7, but for now we assume that every compensation is explicitly defined by the programmer.

We continue by explaining how this external oracle is incorporated in $\mathcal{ETR}$'s theory.

## 5.2 External States, and External Oracle

As in $\mathcal{TR}$, both the language and the semantics of $\mathcal{ETR}$ are parameterized by a set of oracles to reason about basic actions and queries. Consequently, besides the data oracle $\mathcal{O}^d$ and the transition oracle $\mathcal{O}^t$ that define the meaning of the internal KB, $\mathcal{ETR}$ integrates an additional external oracle $\mathcal{O}^e$ to evaluate elementary external operations and to abstract the semantics of external states.

As before, states are simply defined by state identifiers. Since $\mathcal{ETR}$ is meant to operate on both an internal KB and external environment, two disjoint sets of state identifiers are needed: one for internal states, and another for uniquely identifying states of the external domain (*external states*). The *external oracle*, $\mathcal{O}^e$, is a mapping from pairs of external states identifiers to formulas in $\mathcal{L}_a^*$. If $\mathcal{O}^e(E_1, E_2) \models \varphi$ then the primitive external action $\varphi$ is said to execute at the state identified by $E_1$ yielding the state identified by $E_2$.

Dealing with state identifiers instead of materialized states is of particular importance when considering external domains. In fact, it may be impossible for the internal system to know what does a particular state identifier mean, as e.g., when dealing with web-services as an external domain. To interact with such external domains, all we need to know is the elementary primitives that can be used to perform queries and updates, and abstract the notion of states to state identifiers. As before, in the remainder we use the terms state and state identifier interchangeably.

Since we stipulated that the actions `failop` and `nop` always belong to $\mathcal{L}_a^*$ with a precise meaning, we also enforce that, for every external oracle and every pair of external states, $\mathcal{O}^e(E_1, E_2) \not\models$ `failop` and $\mathcal{O}^e(E_1, E_1) \models$ `nop` (i.e., `failop` always fails, and `nop` always succeeds leaving the state unchanged).

External actions with compensations, $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$, are evaluated by the external oracle solely according to what is known about $a$, i.e., for every oracle we require that $\mathcal{O}^e(E_1, E_2) \models \mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ iff $\mathcal{O}^e(E_1, E_2) \models a$ for any $a, b_i \in \mathcal{L}_a$. Thus, as expected, it is not the task of the oracle (but rather of the $\mathcal{ETR}$ semantics, as we shall see) to deal with compensations.

Note that $\mathcal{TR}$ requires *two* oracles, $\mathcal{O}^d$ and $\mathcal{O}^t$, to define the semantics of queries and updates, respectively. This separation promotes the distinction between the static semantics of states and its dynamics. However, these two oracles are not independent of each other and they must share the same set of state identifiers. In practice, although this separation may ease the task of implementing these oracles (because of the separation of the two concepts), nothing prevents us from expressing both oracles using only a single mapping. Due to this, in $\mathcal{ETR}$ we assume that only one external oracle is needed to characterize the behavior of the external environment w.r.t. a given primitive (both queries and updates). Since little may be known about the external domain, it may not be possible to distinguish between an external query and an external update and thus, we assume that every external primitive can cause a state transition. If $q$ is a query, then the primitive is mapped to a pair with the same state, i.e., $q \in \mathcal{O}^e(S, S)$.

The external oracle abstracts the theory and semantics of the external domain, encapsulating the elementary operations that can be performed externally. In the sequel we see this oracle as a black box, that encodes the behavior of the external domain in a completely independent way. However, if one wants to reason about both the internal and the external KB, one can fix $\mathcal{O}^e$ by formalizing it with some logic for describing external worlds. Thus, after defining $\mathcal{ETR}$'s semantics we also elaborate upon the role of this external oracle in chapter 6, and show how it can be instantiated with Description Logics, Action Languages, Event Calculus or Situation Calculus.

## 5.3 Model Theory

As in $\mathcal{TR}$, formulas in $\mathcal{ETR}$ are evaluated on *paths*, i.e., on sequences of states. Since $\mathcal{ETR}$ deals with an external environment, a state $S$ is now a pair $(D, E)$ where $D$ represents

an internal state and $E$ denotes an external state. Based on this, a path is a sequence of states. Moreover, for convenience and to help the semantics to deal with recovery of external failures, we also record in paths the primitive actions performed between states, defined as follows.

**Definition 10** (States and Paths)**.** *An $\mathcal{ETR}$* state *$S$ is a pair $(D, E)$ where $D$ and $E$ are, respectively, internal and external states. A* path *of length $k$, or a* k-path*, is a finite sequence of states, $\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{k-1}} S_k \rangle$ where $A_i$s $(1 \leq i < k)$ are atoms from $\mathcal{L}_a^*$ or $\mathcal{L}_i$.*

In this definition, a path $\langle S_i \xrightarrow{A_i} S_{i+1} \rangle$ means that action $A_i$ caused the change from state $S_i$ into state $S_{i+1}$. If $A$ is a formula of the form $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ then this allows us to know that $b_1 \otimes \ldots \otimes b_j$ is the compensation to be performed in the event of a failure after the execution of the external action $a$.

Interpretations are then defined in the usual way, but now they also incorporate the external oracle.

**Definition 11** (Interpretations)**.** *An interpretation is a mapping $M$ assigning a classical Herbrand structure (or $\top$) to every path. This mapping is subject to the following restrictions, for all states $(D_i, E_j)$ and every atom $\varphi$ defined in $\mathcal{L}_{\mathcal{O}}$:*

1. *$\varphi \in M(\langle (D, E) \rangle)$ iff $\mathcal{O}^d(D) \models \varphi$ for any external state $E$*
2. *$\varphi \in M(\langle (D_1, E) \xrightarrow{\varphi} (D_2, E) \rangle)$ iff $\mathcal{O}^t(D_1, D_2) \models \varphi$ for any external state $E$*
3. *$\varphi \in M(\langle (D, E_1) \xrightarrow{\varphi} (D, E_2) \rangle)$ iff $\mathcal{O}^e(E_1, E_2) \models \varphi$ for any internal state $D$*

Note that $\mathcal{ETR}$ rules cannot have oracle atoms in their heads (cf. definition 9). I.e., in $\mathcal{ETR}$, contrary to $\mathcal{TR}$, it is not possible to redefine oracle primitives in programs. This restriction makes the logic cleaner by strictly separating between primitives and complex actions defined in a program, without particularly limiting the expressive power of the language. In fact, every $\mathcal{TR}$ program can be re-written to comply with this restriction. Consider for instance that the rule $a.ins \leftarrow c.ins \otimes d.ins$ exists in the program, then we can re-write it as:

$$new\_a\_ins \leftarrow a.ins$$
$$new\_a\_ins \leftarrow c.ins \otimes d.ins$$

and replace all the remaining occurrences of $a.ins$ in the program by $new\_a\_ins$. Due to this design choice, interpretation functions in $\mathcal{ETR}$ restrict oracle atoms to be true only when oracles define it so.

Before defining satisfaction of formulas, besides the operation of path splits, which is simply inherited from $\mathcal{TR}$ (cf. definition 2), we need an additional notion for ending of a path, which will be helpful to isolate the exact point of failure of a transaction.

**Definition 12** (Ending of a Path)**.** *The* ending *of a k-path $\pi$ corresponds to the 1-path $\pi_{\mathtt{end}}$ composed of the last state of $\pi$, i.e., if $\pi = \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{k-1}} S_k \rangle$ then $\pi_{\mathtt{end}} = \langle S_k \rangle$.*

$\mathcal{TR}$'s standard definition of satisfaction can now be easily adapted to accept the $\mathcal{ETR}$'s notion of paths, where a state $S$ is a pair consisting of an internal and external state $(D, E)$:

**Definition 13** (Classical Satisfaction)**.** *Let $M$ be an interpretation, $\pi$ a path and $\phi$ a formula. If $M(\pi) = \top$ then $M, \pi \models_c \phi$; otherwise:*

1. ***Base Case:*** *$M, \pi \models_c \phi$ iff $\phi$ is an atom and $\phi \in M(\pi)$*
2. ***Negation:*** *$M, \pi \models_c \neg\phi$ iff it is not the case that $M, \pi \models_c \phi$*
3. ***"Classical" Disjunction:*** *$M, \pi \models_c \phi \vee \psi$ iff $M, \pi \models_c \phi$ or $M, \pi \models_c \psi$.*
4. ***"Classical" Conjunction:*** *$M, \pi \models_c \phi \wedge \psi$ iff $M, \pi \models_c \phi$ and $M, \pi \models_c \psi$.*
5. ***Serial Conjunction:*** *$M, \pi \models_c \phi \otimes \psi$ iff $M, \pi_1 \models_c \phi$ and $M, \pi_2 \models_c \psi$ for some split $\pi_1 \circ \pi_2$ of path $\pi$.*

As discussed above, $\mathcal{TR}$'s satisfaction does not consider the possibility of failure. Since $\mathcal{ETR}$ allows external actions as transaction formulas, it must take into the account the possibility of a transaction to fail. Particularly, if a failure occurs after the execution of external actions, then we need to execute some compensating operations to invert the external actions already performed, and recover a consistent state in the external KB. As such, $\mathcal{ETR}$'s model theory also needs to address how the external recovery can be ensured in case of external failures. The partial satisfaction relation below is the first ingredient to deal with such failures.

**Example 14** (Running Example)**.** *Recall example 12, but now where the external expressions like $external\_a$ are replaced by external actions with compensations:*

$$t \leftarrow p.ins \otimes \mathbf{ext}(a, a_1 \otimes a_2) \otimes \mathbf{ext}(b, b_1)$$
$$t \leftarrow q.ins \otimes \mathbf{ext}(c, c_1)$$

*Moreover, assume that the internal KB is a relational database as formalized in chapter 3, page 28, and the external oracle includes: $\mathcal{O}^e(e_1, e_2) \models a$, (i.e., the external execution of $a$ in state $e_1$ succeeds, and makes the external world evolve into $e_2$), $\mathcal{O}^e(e_1, e_5) \models c$, and that for every state $e$, $\mathcal{O}^e(e_2, e) \not\models b$ (i.e., the execution of $b$ in state $e_2$ fails).*

*In this example, it can be easily checked that the formula $p.ins \otimes \mathbf{ext}(a, a_1 \otimes a_2)$ is classically satisfied on the path:*

$$\langle(\{\}, e_1) \xrightarrow{p.ins} (\{p\}, e_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{p\}, e_2)\rangle$$

*Similarly, the formula $q.ins \otimes \mathbf{ext}(c, c_1)$ is classically satisfied on the path:*

$$\langle(\{\}, e_1) \xrightarrow{q.ins} (\{q\}, e_1) \xrightarrow{\mathbf{ext}(c, c_1)} (\{q\}, e_5)\rangle$$

*Furthermore, given the external oracle definition above, it is easy to see that* $\mathbf{ext}(b, b_1)$ *cannot succeed on any path starting at state* $e_2$*.*

*The idea of partial satisfaction is to identify the path:*

$$\langle(\{\}, e_1)^{p.ins}\rightarrow(\{p\}, e_1)^{\mathbf{ext}(a,a_1\otimes a_2)}\rightarrow(\{p\}, e_2)\rangle$$

*as one that satisfies the formula* $p.ins \otimes \mathbf{ext}(a, a_1 \otimes a_2) \otimes \mathbf{ext}(b, b_1)$ *up to some point, though the formula eventually fails (i.e., the serial conjunction formula is satisfied up to some point, but then fails; and we use the name* partial satisfaction *to signal that only some part of the formula is satisfied).*

A formula is partially satisfied if it either completely succeeds or if it succeeds up to some point, and then fails in a primitive action. Specifically:

**Definition 14** (Partial Satisfaction)**.** *Let* $M$ *be an interpretation,* $\pi$ *a path and* $\phi$ *a formula. If* $M(\pi) = \top$ *then* $M, \pi \models_p \phi$*; otherwise:*

1. ***Base Case:*** $M, \pi \models_p \phi$ *iff* $\phi$ *is an atom and one of the following holds:*
   (a) $M, \pi \models_c \phi$
   (b) $M, \pi \not\models_c \phi$, $\phi \in \mathcal{L}_i$, $\pi = \langle(D, E)\rangle$ *and* $\neg\exists D_i$ *s.t.* $M, \langle(D, E)^{\phi}\rightarrow(D_i, E)\rangle \models_c \phi$
   (c) $M, \pi \not\models_c \phi$, $\phi \in \mathcal{L}^*_a$, $\pi = \langle(D, E)\rangle$ *and* $\neg\exists E_i$ *s.t.* $M, \langle(D, E)^{\phi}\rightarrow(D, E_i)\rangle \models_c \phi$
2. ***Negation:*** $M, \pi \models_p \neg\phi$ *iff it is not the case that* $M, \pi \models_p \phi$
3. ***"Classical" Disjunction:*** $M, \pi \models_p \phi \vee \psi$ *iff* $M, \pi \models_p \phi$ *or* $M, \pi \models_p \psi$
4. ***"Classical" Conjunction:*** $M, \pi \models_p \phi \wedge \psi$ *iff* $M, \pi \models_p \phi$ *and* $M, \pi \models_p \psi$
5. ***Serial Conjunction:*** $M, \pi \models_p \phi \otimes \psi$ *iff one of the following holds:*
   (a) $M, \pi \models_p \phi$ *and* $M, \pi \not\models_c \phi$
   (b) $\exists$ *split* $\pi_1 \circ \pi_2$ *of path* $\pi$ *s.t.* $M, \pi_1 \models_c \phi$ *and* $M, \pi_2 \models_p \psi$

There may be several reasons for a formula not to be classically satisfied in a given path, and, by design, not all of them are taken into consideration in the definition of partial satisfaction. For example, given a relational oracle, the query $a$ is not classically satisfied on the path $\langle(\{a\}, e)^{b.ins}\rightarrow(\{a, b\}, e)\rangle$, independently of the interpretation $M$ chosen. Similarly, the action $b.ins$ is never satisfied over the 1-path $\langle(\{a\}, e)\rangle$. However, these failures are not interesting in the sense that they do not correspond to a real execution-try. Particularly, $b.ins$ can classically succeed on a path starting in $\langle(\{a\}, e)\rangle$, and the query $a$ is true in the 1-paths composed of any of the singleton states $\langle(\{a\}, e)\rangle$ and $\langle(\{a, b\}, e)\rangle$. As such, partial satisfaction is defined in such a way that, although for any $M$, $M, \langle(\{a\}, e)^{b.ins}\rightarrow(\{a, b\}, e)\rangle \not\models_c a$ (i.e., $a$ fails), it is also the case that $M, \langle(\{a\}, e)^{b.ins}\rightarrow(\{a, b\}, e)\rangle \not\models_p a$.

Consequently, our definition of partial satisfaction only deals with failures that come from a real impediment to executing a primitive action from a particular state $S_0$. In the case of atomic queries, this means that the given query is not true in a particular 1-path, and in the case of atomic actions, it means that there is no possible evolution from $S_0$ that successfully satisfies the action (items 1b and 1c respectively).

With these definitions of classical and partial satisfaction, a "legitimate" failure is one where a formula is partially but not classically satisfied in a path. Moreover, cf. claim 1 of proposition 1 below, the path where this happens must always end exactly in the state prior to the failure. This is the reason why in the definition of partial satisfaction failures of primitives are constrained to 1-paths in items 1b and 1c. These 1-paths represent the state where the transaction failed.

Besides this result, Proposition 1 shows additional properties of the partial satisfaction definition. Claim 2 states that we can weaken a formula that is partially but not classically satisfied using the serial conjunctive operator $\otimes$, i.e., that in any path $\pi$ where $\phi$ can be partially but not classically executed, then $M, \pi \models_p \phi \otimes \psi$ for every formula $\psi$. Additionally, for positive formulas, the partial satisfaction is a relaxed version of the classical satisfaction (claim 3), and the two satisfaction relations coincide whenever they are evaluating atoms that are not specified by the oracles (claim 4).

**Proposition 1.** *Let $M$ be an interpretation, $\pi$ a path, $\pi_{\texttt{end}}$ the 1-path containing the last state of $\pi$, $\phi$ and $\psi$ be $\mathcal{ETR}$ formulas, $\phi'$ a positive formula, $\phi_P$ an atom from $\mathcal{L}_P$ and $a$ an atom such that $a \in \mathcal{L}_i$ or $a \in \mathcal{L}_a^*$.*

1. *If $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$ then $\exists a$ s.t. $a$ occurs in $\phi$, $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$*
2. *If $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$ then $M, \pi \models_p \phi \otimes \psi$*
3. *If $M, \pi \models_c \phi'$ then $M, \pi \models_p \phi'$*
4. *$M, \pi \models_c \phi_P$ iff $M, \pi \models_p \phi_P$*

*Proof.* This is shown in Appendix A on page 207. □

Importantly, note that the serial conjunction operator is no longer associative in this definition. More precisely, $(\phi_1 \otimes (\phi_2 \otimes \phi_3))$ is not equivalent to $((\phi_1 \otimes \phi_2) \otimes \phi_3)$ if a failure occurs when executing one of the formulas. This is so since we suspend the execution of a formula as soon as a failure occurs (i.e., whenever a formula is partially but not classically satisfied). For instance, imagine that: $M, \langle (D, E) \rangle \models_p \phi_1$ but $M, \pi \not\models_c \phi_1$, but $M, \pi_1 \models_c \phi_2 \otimes \phi_3$, where $\pi_1$ starts in path $\langle D, E \rangle$. Then $M, \pi \models_p ((\phi_1 \otimes \phi_2) \otimes \phi_3)$, but $M, \pi \not\models_p (\phi_1 \otimes (\phi_2 \otimes \phi_3))$.

Consider again example 12. In $\mathcal{TR}$, just like in logic programming, we can satisfy $t$ by satisfying either the first body or the second. But, as motivated above, in $\mathcal{ETR}$ we want consider an additional way of satisfying $t$. Namely, $t$ should also be satisfied if the first body is "tried", compensated, and then the second body is successfully executed. With the definitions above, we made precise what is meant by the "tried", viz. it is partially but

not classically satisfied. The next definitions specify what is left, i.e., how to successfully compensate a formula that is partially but not classically satisfied.

However, for this, some additional operations on paths are needed. We start by defining the notion of a *rollback path* for a given path. Intuitively, given a path $\pi$, its rollback path is obtained by keeping all externally executed actions, and rolling back on the internal state:

**Definition 15** (Rollback Path, and Sequence of External Actions)**.** *Let $\pi$ be a k-path of the following form $\langle (D_1, E_1) \xrightarrow{A_1} (D_2, E_2) \xrightarrow{A_2} \ldots \xrightarrow{A_{k-1}} (D_k, E_k)\rangle$. The* rollback *path of $\pi$ is the path obtained from $\pi$ by:*

1. *Replacing all $D_i$s by $D_1$*
2. *Keeping just the transitions where $A_i \in \mathcal{L}_a^*$.*

*The sequence of external actions of $\pi$, denoted $\texttt{Seq}(\pi)$, is the sequence of actions of the form $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ that appear in the transitions of the rollback path of $\pi$.*

**Example 15** (Rollback path)**.** *Consider the path $\pi = \langle (\{\}, e_1) \xrightarrow{p.ins} (\{p\}, e_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{p\}, e_2)\rangle$. The* rollback path *$\pi_0$ of $\pi$ is the path obtained by rolling back the internal state and keeping the external transitions. Thus, $\pi_0 = \langle (\{\}, e_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, e_2)\rangle$.*

Note that the operator $\texttt{Seq}(\pi)$ only includes the external actions that have the form $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$. Since our aim is to compensate the executed actions, then actions without compensations are skipped. Alternatively, to define compensations that always fail, one should use the primitive `failop` as in $\mathbf{ext}(a, \texttt{failop})$.

Building on this, we define the notion of a *recovery path*. After rolling back the internal state and retrieving all the necessary compensations, external recovery is achieved by executing the compensation operations defined in $\texttt{Seq}(\pi)$ in the inverse order.

**Definition 16** (Inversion, and Recovery Path)**.** *Let $S = \langle \mathbf{ext}(a_1, \mathcal{B}_1), \ldots, \mathbf{ext}(a_n, \mathcal{B}_n)\rangle$ be a sequence of actions from $\mathcal{L}_a^*$, and $\mathcal{B}_i$ is a sequence of actions of the form $(b_{1_i} \otimes \ldots \otimes b_{k_i})$. Then, the inversion of $S$ is the transaction formula $\texttt{Inv}(S) = \mathcal{B}_n \otimes \ldots \otimes \mathcal{B}_1$.*

*$\pi_r$ is a* recovery path *of $\texttt{Seq}(\pi)$ w.r.t. $M$ iff $M, \pi_r \models_c \texttt{Inv}(\texttt{Seq}(\pi))$.*

**Example 16** (Rollback and Recovery)**.** *Recall example 14 and further assume the following definition for the external oracle: $\mathcal{O}^e(e_2, e_3) \models a_1$ and $\mathcal{O}^e(e_3, e_4) \models a_2$.*

*From example 15 we know that the rollback path $\pi_0$ of $\pi = \langle (\{\}, e_1) \xrightarrow{p.ins} (\{p\}, e_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{p\}, e_2)\rangle$ is $\pi_0 = \langle (\{\}, e_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, e_2)\rangle$.*

*Thus, by definition 16, $\texttt{Seq}(\pi_0) = \langle \mathbf{ext}(a, a_1 \otimes a_2)\rangle$ and $\texttt{Inv}(\texttt{Seq}(\pi_0)) = a_1 \otimes a_2$. Finally, given our previously stated external oracle definitions, we know that:*

$$\langle (\{\}, e_2) \xrightarrow{a_1} (\{\}, e_3) \xrightarrow{a_2} (\{\}, e_4)\rangle$$

*is a recovery path of $\texttt{Seq}(\pi)$ w.r.t. any interpretation $M$.*

Equipped with these auxiliary definitions, we can finally make precise what we mean by compensating a formula that is partially but not classically satisfied. This latter is encoded in $M, \pi \rightsquigarrow \phi$, where we say that formula $\phi$ fails over $\pi$, but consistency was still achieved, i.e., all external actions executed are compensated, and the internal state is rolled back.

**Definition 17** (Compensating Path for a Transaction)**.** *Let $M$ be an interpretation, $\pi$ a k-path ($k \geq 2$) with external actions in the transitions, and $\phi$ a formula. $M, \pi \rightsquigarrow \phi$ if* all *the following conditions are true:*

1. *$\exists \pi_1$ such that $M, \pi_1 \models_p \phi$ and $M, \pi_1 \not\models_c \phi$*
2. *$\exists \pi_0$ such that $\pi_0$ is the rollback path of $\pi_1$*
3. *$\texttt{Seq}(\pi_0) \neq \emptyset$ and $\exists \pi_r$ such that $\pi_r$ is a recovery path of $\texttt{Seq}(\pi_0)$ w.r.t. $M$*
4. *$\pi_0$ and $\pi_r$ are a split of $\pi$, i.e., $\pi = \pi_0 \circ \pi_r$*

**Example 17** (Compensating Path)**.** *In the scenario of the previous examples 14–16, the statement:*

$$M, \langle (\{\}, e_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, e_2) \xrightarrow{a_1} (\{\}, e_3) \xrightarrow{a_2} (\{\}, e_4) \rangle \rightsquigarrow p.ins \otimes \mathbf{ext}(a, a_1 \otimes a_2) \otimes \mathbf{ext}(b, b_1)$$

*holds for any interpretation $M$. Note that this path does not satisfy the formula (since $\mathbf{ext}(b, b_1)$ fails). Instead, it leaves the internal and external KBs in a state somehow equivalent to the initial state: the operations done in the internal KB are rolled back, and the externally executed actions are compensated.*

A compensating path for a formula $\phi$ is one where $\phi$ is *not* successfully executed, but where external recovery can still be guaranteed. Also note that these compensating paths are only defined for cases where, besides a primitive action fails, some external actions with compensations were executed. This is so, because the operator $\texttt{Seq}(\pi_0)$ only collects external actions of the form $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$. This is as expected: if no external actions were executed on $\pi_0$ or if all the external actions executed are not meant to be compensated (e.g. if they are external queries), then $\texttt{Seq}(\pi_0) = \emptyset$. Intuitively, if this is the case then no compensations are needed, and the formula just fails (as in standard $\mathcal{TR}$).

Based on these definitions, we are finally able to formalize which (complex) formulas are true on which paths.

**Definition 18** (General Satisfaction)**.** *Let $M$ be an interpretation, $\pi$ a path and $\phi$ a formula. If $M(\pi) = \top$ then $M, \pi \models \phi$; otherwise:*

1. ***Base Case:** $M, \pi \models \phi$ if $\phi$ is an atom and $\phi \in M(\pi)$*
2. ***Negation:** $M, \pi \models \neg\phi$ if it is not the case that $M, \pi \models \phi$*
3. ***"Classical" Disjunction:** $M, \pi \models \phi \vee \psi$ if $M, \pi \models \phi$ or $M, \pi \models \psi$.*

4. ***"Classical" Conjunction:*** *$M, \pi \models \phi \wedge \psi$ if $M, \pi \models \phi$ and $M, \pi \models \psi$.*

5. ***Serial Conjunction:*** *$M, \pi \models \phi \otimes \psi$ if $M, \pi_1 \models \phi$ and $M, \pi_2 \models \psi$ for some split $\pi_1 \circ \pi_2$ of $\pi$.*

6. ***Compensating Case:*** *$M, \pi \models \phi$ if $M, \pi_1 \rightsquigarrow \phi$ and $M, \pi_2 \models \phi$ for some split $\pi_1 \circ \pi_2$ of $\pi$*

7. *For no other $M, \pi$ and $\phi$, $M, \pi \models \phi$.*

This definition strongly resembles definition 13. Intuitively, with this general notion of satisfaction, a formula $\phi$ succeeds if it succeeds classically, or if although a primitive action failed to be executed, the system can recover from the failure and $\phi$ can still succeed on an alternative path (item 6). As expected, recovery only makes sense in situations where some external actions were performed before the failure. Otherwise we can just roll back to the initial state and try to satisfy the formula in an alternative branching.

**Example 18.** *Recall examples 14–16, and further assume $\mathcal{O}^e(e_4, e_5) \models c$. Based on this, the complex formula: $(p.ins \otimes \mathbf{ext}(a, a_1 \otimes a_2) \otimes \mathbf{ext}(b, b_1)) \vee (q.ins \otimes \mathbf{ext}(c, c_1))$ is satisfied on the path:*

$$\langle (\{\}, e_1) \xrightarrow{q.ins} (\{q\}, e_1) \xrightarrow{\mathbf{ext}(c,c_1)} (\{q\}, e_5) \rangle$$

*(without using compensations), and also on the path:*

$$\langle (\{\}, e_1) \xrightarrow{\mathbf{ext}(a,a_1 \otimes a_2)} (\{\}, e_2) \xrightarrow{a_1} (\{\}, e_3) \xrightarrow{a_2} (\{\}, e_4) \xrightarrow{q.ins} (\{q\}, e_4) \xrightarrow{\mathbf{ext}(c,c_1)} (\{q\}, e_5) \rangle$$

*by using item 6 (where the first disjunct is tried, fails and is compensated).*

As previously stated, the general satisfaction is strongly related to the classical satisfaction. Particularly, besides the compensating case, the definition of general satisfaction exactly coincides with classical satisfaction (definition 13). We state this correspondence as follows.

**Theorem 2.** *Let $M$ be an interpretation, $\phi$ any formula, and $\phi'$ a positive formula and $\pi, \pi'$ paths such that $\pi'$ is a path where no external actions appear in the transitions. Then:*

$$\text{If } M, \pi \models_c \phi' \text{ then } M, \pi \models \phi' \tag{5.1}$$

$$M, \pi' \models_c \phi \text{ iff } M, \pi' \models \phi \tag{5.2}$$

*Proof.* See Appendix A, page 210 □

Models and logical entailment can now be defined as usual:

**Definition 19** (Model and entailment of a formula)**.** *An interpretation $M$ is a model of a formula $\phi$ (denoted $M \models \phi$) iff $M, \pi \models \phi$ for every path $\pi$.*
*A formula $\phi$ logically entails another formula $\psi$ ($\phi \models \psi$) if every model of $\phi$ is also a model of $\psi$.*

Since $\mathcal{ETR}$ restricts the use of negation to atoms, we have to explicitly define what it means to model a rule and to model a program. Intuitively, an interpretation models a rule if, whenever it models its body, it also models its head; as usual, it models a program if it models all its rules. Moreover, to deal with compensations, we further impose that, for models of rules, compensating paths and classical satisfaction of the rule body correspond with the compensating paths and classical satisfaction for the head:

**Definition 20** (Model of a Program)**.** *An interpretation $M$ models a rule $head \leftarrow body$ iff for every path $\pi$:*

- *If $M, \pi \models body$ then $M, \pi \models head$, and*
- *If $M, \pi \models_c body$ then $M, \pi \models_c head$, and*
- *If $M, \pi \rightsquigarrow body$ then $M, \pi \rightsquigarrow head$*

*An interpretation $M$ is a model of a program $P$ if it models all its rules. In this case we write $M \models P$.*

*A program $P$ entails another program $P'$ ($P \models P'$) if all models of $P$ are models of $P'$. Two programs $P$ and $P'$ are equivalent iff $P \models P'$ and $P' \models P$.*

## 5.4 Executional Entailment

Logical entailment, be it of formulas or programs, takes into account all the possible execution paths of a transaction formula. Hence, this entailment can be used to define general equivalence and implication of formulas, as one can express properties like "whenever transaction $\phi$ is executed, $\psi$ is also executed" ($\phi \models \psi$) or "transaction $\phi$ is equivalent to transaction $\psi$" ($\phi \models \psi$ and $\psi \models \phi$); or of programs, as one can specify that a program is equivalent to another program (if they entail one another).

Useful as this might be, sometimes one needs a simpler kind of reasoning that is concerned only with a particular execution of a formula. As such, similarly to $\mathcal{TR}$, in addition to logical entailment $\mathcal{ETR}$ supports another entailment called *executional entailment*. Whilst logical entailment allows one to *reason* about $\mathcal{ETR}$ theories, executional entailment provides a logical account of *execution* of $\mathcal{ETR}$.

**Definition 21** (Executional Entailment)**.** *Let $P$ be a program, $\phi$ be a formula and $\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{n-1}} S_n \rangle$ be a path. The statement:*

$$P, \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{n-1}} S_n \rangle \models \phi \tag{5.3}$$

*is true if $M, \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{n-1}} S_n \rangle \models \phi$ for every model $M$ of $P$. We write $P, S_1 - \models \phi$ when there exists a path $S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{n-1}} S_n$ that makes (5.3) true.*

The previously defined statement $P, \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{n-1}} S_n \rangle \models \phi$ means that, given a program $P$, the path $\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{n-1}} S_n \rangle$ represents a valid execution for transaction $\phi$.

**Example 19.** *In our running example, both statements below hold:*

$$P, \langle(\{\}, e_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, e_2) \xrightarrow{a_1} (\{\}, e_3) \xrightarrow{a_2} (\{\}, e_4) \xrightarrow{q.ins} (\{q\}, e_4) \xrightarrow{\mathbf{ext}(c, c_1)} (\{q\}, e_5)\rangle \models t$$
$$P, \langle(\{\}, e_1) \xrightarrow{q.ins} (\{q\}, e_1) \xrightarrow{\mathbf{ext}(c, c_1)} (\{q\}, e_5)\rangle \models t$$

*The latter correspond to executing, and succeeding, $t$ by the second rule. The former amounts to trying to execute $t$ by the first rule, failing, rolling back and compensating, and then executing and succeeding $t$ by the second rule.*

$P, S_1 - \models \phi$ accounts for situations where all one wants to know is whether $\phi$ can succeed starting from state $S_1$ under $P$, e.g. $P, (\{\}, e_1) - \models t$ (meaning that $t$ succeeds if executed in that initial state).

As expected, $\mathcal{ETR}$ is a conservative extension of $\mathcal{TR}$. Namely, if $\phi$ and $P$ are valid in both logics (e.g. do not contain external actions), then the two logics coincide, i.e., they satisfy the same formulas in the same paths. This is encoded in Theorem 3. Obviously, since paths in $\mathcal{ETR}$ have an additional external component compared to $\mathcal{TR}$, the paths only coincide in their shared internal path.

**Theorem 3** (Relation to $\mathcal{TR}$)**.** *Let $P$ be a transaction program and $\phi$ a transaction formula such that $P$ and $\phi$ are both well-formed in $\mathcal{TR}$'s and in $\mathcal{ETR}$'s syntax. Then for any external state $E$:*

$$P, \langle(D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi \textit{ iff } P, \langle D_1, \ldots, D_n\rangle \models_{TR} \phi$$

*Proof.* See Appendix A, page 216 □

## 5.5 A Proof Procedure for $\mathcal{ETR}$

Executional entailment determines the meaning of executing a transaction defined in a program, starting from an initial state. Our next step is to define a procedure for proving transactions in that way. In this section we extend the proof theory for the ground[2] serial-Horn $\mathcal{TR}$ fragment as described in definition 7. The advantage of this fragment is that it can be formulated as a least-fixpoint in a logic programming style.

As before, a serial-Horn program $P$ is a finite set of *serial-Horn rules*. A serial goal is a transaction formula of the form $a_1 \otimes a_2 \otimes \ldots \otimes a_n$, where each $a_i$ is an atom and $n \geq 0$. When $n = 0$, we write $()$, which denotes the empty goal. A *serial-Horn rule* has the form $b \leftarrow a_1 \otimes \ldots \otimes a_n$, where the body $a_1 \otimes \ldots \otimes a_n$ is a serial goal and the head $b$ is an atom.

The procedure verifies if $P, S_0 - \models \phi$ holds, i.e, that a transaction $\phi$ can succeed starting from the state $S_0 = (D_0, E_0)$ and, in case of success, to obtain a path starting in $S_0$ that satisfies $\phi$.

This procedure starts with a program $P$, an initial state $S_0$, a serial goal $\phi$ and manipulates *resolvents*. At each step, the procedure non-determinstically applies a series of rules to the current resolvent until it either reaches the empty goal and succeeds, or no

[2]The restriction to ground formulas is not essential and can be easily lifted. We only require it in order to simplify the presentation.

more rules are applicable and the derivation fails. Moreover, if the procedure succeeds, it also returns a path in which the goal succeeds. To cater for this last requirement, resolvents contain the information about the path obtained so far. A resolvent is of the form $\pi, S_i \Vdash_P \phi$, meaning that, we want to execute transaction $\phi$ in $P$ from the initial state $S_i$; where $\pi$ is the path history obtained so far.

With this form of resolvents, a proof, or successful derivation, for $P, S_0 - \models \phi$ starts with the resolvent $\langle S_0 \rangle, S_0 \Vdash_P \phi$ and applies the rules defined below, until eventually it reaches a resolvent $\pi, S_f \Vdash_P ()$. If such a proof is found, and since our procedure is sound and complete w.r.t. the theory (cf. theorem 4) then we further conclude that $P, \pi \models \phi$ (where $\pi$ starts with $S_0$ and ends with $S_f$). I.e., not only do we prove that transaction $\phi$ can succeed starting from $S_0$, but we also find a path where $\phi$ succeeds.

The rules for this derivation are specified in definition 22. Derivation rules $\mathbf{r_1} - \mathbf{r_4}$ are equivalent to the ones appearing in $\mathcal{TR}$'s proof theory (cf. definition 7). Rule $\mathbf{r_1}$ applies a transaction definition by unfolding its definition rule, i.e., if we are proving a given atom that is defined in the program in the head of a rule, we can replace this atom in the resolvent by the body of that rule. Rule $\mathbf{r_2}$ deals with a query to the oracle: if we are proving an atom defined in the database oracle, and the database oracle satisfies this atom in that particular state, then we can simply remove this atom. Finally, rules $\mathbf{r_3}$ and $\mathbf{r_4}$ respectively define the execution of an internal and external update. More precisely, if the atom is a primitive action $A$ that can succeed on the current state $S_1$ leading to state $S_2$ (i.e., $A$ is true in $\langle S_1 \, {}^{A}\!\!\rightarrow S_2 \rangle$) then we can remove the atom from the resolvent, and update both the path and the state appropriately. This set of rules forms the basis of the so called $SLD_{\mathcal{ETR}}$ classical derivation.

To deal with compensations, $SLD_{\mathcal{ETR}}$ has an additional rule (rule $\mathbf{r_5}$). As in $\mathcal{ETR}$'s theory, in order to compensate for the failure execution of a formula, we need to capture the usual notion of successful derivation that succeeds without any compensations (classical derivation), but also, the derivations that do not succeed, but that end in the execution of an external action that fails in the oracle. Whenever the latter is the case (i.e., whenever rule $\mathbf{r_5}.(a)$ is applicable), then we need to roll back the internal actions executed (rule $\mathbf{r_5}.(b)$), compensate for the external actions (rule $\mathbf{r_5}.(c)$), and proceed the computation (rule $\mathbf{r_5}.(d)$).

Moreover, this behavior requires the ability to handle derivations that fail, and for that purpose we define *action-failed* derivations. These correspond to the $SLD_{\mathcal{ETR}}$ derivations that end in a resolvent of the form $\pi, S_f \Vdash_P L_1 \otimes \ldots \otimes L_n$ and $L_1$ is an external action primitive that cannot be executed in $S_f$.

**Definition 22** ($SLD_{\mathcal{ETR}}$ Derivation and Classical Derivation)**.** *An $SLD_{\mathcal{ETR}}$-derivation (resp. classical derivation) for a serial goal $\phi$ in a program $P$ and state $S_0$ is finite a sequence of resolvents starting with $\langle S_0 \rangle, S_0 \Vdash_P \phi$, and obtained by non-deterministically applying the rules $r_1$–$r_5$ (resp. $r_1$–$r_4$) specified below. Let $\pi, (D_1, E_1) \Vdash_P L_1 \otimes \ldots \otimes L_n$ be a resolvent. Then*

*the next resolvent in the derivation is defined by:*

$\mathbf{r_1}$. $\pi, (D_1, E_1) \Vdash_P B_1 \otimes \ldots \otimes B_j \otimes L_2 \otimes \ldots \otimes L_n$ *if* $L_1 \leftarrow B_1 \otimes \ldots \otimes B_j \in P$

$\mathbf{r_2}$. $\pi, (D_1, E_1) \Vdash_P L_2 \otimes \ldots \otimes L_n$ *if* $\mathcal{O}^d(D_1) \models L_1$

$\mathbf{r_3}$. $\pi \circ \langle (D_1, E_1) \xrightarrow{L_1} (D_2, E_1) \rangle, (D_2, E_1) \Vdash_P L_2 \otimes \ldots \otimes L_n$ *if* $\mathcal{O}^t(D_1, D_2) \models L_1$

$\mathbf{r_4}$. $\pi \circ \langle (D_1, E_1) \xrightarrow{L_1} (D_1, E_2) \rangle, (D_1, E_2) \Vdash_P L_2 \otimes \ldots \otimes L_n$ *if* $\mathcal{O}^e(E_1, E_2) \models L_1$

$\mathbf{r_5}$. $\pi \circ \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \xrightarrow{A_k^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \rangle, S_q \Vdash_P L_1 \otimes \ldots \otimes L_n$

*if all of the following conditions hold:*

*(a) There is an action-failed classical derivation starting in* $\langle S_1 \rangle, S_1 \Vdash_P L_1 \otimes \ldots \otimes L_n$ *(where* $S_1 = (D_1, E_1)$*) ending in* $\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{j-1}} S_j \rangle, S_j \Vdash_P \phi$*, for some transaction* $\phi$

*(b)* $S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p$ *is the rollback path of* $S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{j-1}} S_j$ *(cf. definition 15)*

*(c)* $\texttt{Inv}(\texttt{Seq}(\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \rangle)) = A_k^{-1} \otimes \ldots \otimes A_1^{-1}$ *(cf. definition 16)*

*(d) There is successful classical derivation for* $\langle S_p \rangle, S_p \Vdash_P A_k^{-1} \otimes \ldots \otimes A_1^{-1}$ *ending in* $\langle S_p \xrightarrow{A_k^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \rangle, S_q \Vdash_P ()$

**Definition 23** (Successful and Action-failed Derivations)**.** *Let $P$ be a program, $\phi$ a serial goal and $S_0$ an initial state. An $SLD_{\mathcal{ETR}}$-derivation (resp. classical derivation) for $\phi$ in $P$ starting in $S_0$ is* successful *if it ends in a resolvent of the form $\pi, S_f \Vdash_P ()$. In this case we write $P, \pi \vdash \phi$ (resp. $P, \pi \vdash_c \phi$).*
*The derivation is* action-failed *if it ends in a resolvent of the form $\pi, S_f \Vdash_P L_1 \otimes \ldots \otimes L_n$ s.t.:*

*(i). $L_1 \in \mathcal{L}_i$, $\mathcal{O}^d(D_f) \not\models L_1$ and $\neg\exists D_i$ s.t. $\mathcal{O}^t(D_f, D_i) \models L_1$, or*

*(ii). $L_1 \in \mathcal{L}_a^*$ and $\neg\exists E_i$ s.t. $\mathcal{O}^e(E_f, E_i) \models L_1$*

Notice the parallel between these definitions and the satisfaction relations $\models_c$, $\models_p$ and $\models$. Classical derivation, as well as $\models_c$, does not consider the possibility of failures (and, as such, cannot use rule $\mathbf{r_5}$). Action-failed derivations, can only "fail" (and this failure be considered part of a valid derivation) in case it is impossible to execute a given primitive external action from a particular state, just like in $\models_p$. Finally, similarly to $\models$, an $SLD_{\mathcal{ETR}}$-derivation is either a classical derivation, or it includes some failed derivations from which it can recover by rolling back the internal state, compensate all the previously executed external actions, and succeed on an alternative path.

Taken together, definitions 22 and 23 determine a sound and complete procedure to find the paths that satisfy a transaction $\phi$ given a program $P$ and an initial state $S_0$. This procedure resembles an SLD-style procedure and can be seen as an extension of the inference system for serial-$\mathcal{TR}$ as presented in [BK93]. The main differences when compared to $\mathcal{TR}$'s inference system are the evaluation of external actions w.r.t to an external oracle $\mathcal{O}^e$ and the non-deterministic possibility of executing compensations in the derivation.

**Example 20** (Proof Theory)**.** *Recall our previous running examples 14-18, and imagine we want to find the proof for* $P, \langle(\{\}, e_1)\rangle - \models t$*. With this as goal, we have to start in the resolvent:*

$$\langle(\{\}, e_1)\rangle, (\{\}, e_1) \Vdash_P t \tag{5.4}$$

*In this resolvent, we can apply the rules* $\mathbf{r_1}$ *(where we unfold* $t$ *for* $p.ins \otimes \mathbf{ext}(a, a_1 \otimes a_2) \otimes \mathbf{ext}(b, b_1)$*),* $\mathbf{r_3}$ *and* $\mathbf{r_4}$ *respectively, and reach the resolvent:*

$$\pi_1, (\{p\}, e_2) \Vdash_P \mathbf{ext}(b, b_1) \tag{5.5}$$

$$\textit{where } \pi_1 = \langle \quad (\{\}, e_1) \xrightarrow{p.ins} (\{p\}, e_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{p\}, e_2)\rangle$$

*Since* $\mathbf{ext}(b, b_1) \in \mathcal{L}_a^*$ *and by the external oracle definition, there is no* $E_i$ *s.t.* $\mathcal{O}^e(e_2, e_i) \models b$*, then this derivation (from (5.4) to (5.5)) is* action-failed*. Since we have not used rule* $\mathbf{r_5}$*, this derivation is also called a* classical *action-failed derivation. Additionally, by definitions 15 and 16, we know that the rollback path of* $\pi_1$ *is the path:*

$$\pi_0 = \langle(\{\}, e_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, e_2)\rangle$$

*and that* $\texttt{Inv}(\texttt{Seq}(\pi_0)) = a_1 \otimes a_2$*. Moreover, from the resolvent:*

$$\langle(\{\}, e_2)\rangle, (\{\}, e_2) \Vdash_P a_1 \otimes a_2 \tag{5.6}$$

*we can apply the rule* $\mathbf{r_4}$ *twice obtaining:*

$$\pi_2, (\{\}, e_4) \Vdash_P () \tag{5.7}$$

$$\textit{where } \pi_2 = \langle(\{\}, e_2) \xrightarrow{a_1} (\{\}, e_3) \xrightarrow{a_2} (\{\}, e_4)\rangle$$

*Since we did not apply rule* $\mathbf{r_5}$ *and reached* $()$*, this derivation (from (5.6) to (5.7)) is a* successful classical derivation*. Building on these, we can again start in the resolvent*

$$\langle(\{\}, e_1)\rangle, (\{\}, e_1) \Vdash_P t \tag{5.8}$$

*and apply rule* $\mathbf{r_5}$*, and thus reach the resolvent*

$$\pi_1 \circ \pi_2, (\{\}, e_4) \Vdash_P t \tag{5.9}$$

*Afterwards, we can apply rule* $\mathbf{r_1}$ *(where we unfold* $t$ *for* $q.ins \otimes \mathbf{ext}(c, c_1)$*),* $\mathbf{r_3}$ *and* $\mathbf{r_4}$*, and reach the resolvent:*

$$\pi_f, (\{q\}, e_5) \Vdash_P () \tag{5.10}$$

$$\textit{where } \pi_f = \pi_1 \circ \pi_2 \circ \langle \quad (\{\}, e_4) \xrightarrow{q.ins} (\{q\}, e_4) \xrightarrow{\mathbf{ext}(c, c_1)} (\{q\}, e_5)\rangle$$

*This latter derivation (from (5.8) to (5.10)) is called a* successful $SLD_{\mathcal{ETR}}$*-derivation, and thus*

*we write* $P, \pi_f \vdash t$.

**Theorem 4** (Soundness and Completeness of $\vdash$)**.** *Let $P$ be a serial-Horn program, $\phi$ a serial-Horn goal, and $\pi$ be a path starting in state $S_0$ and ending in $S_f$. Then, $P, \pi \models \phi$ iff $P, \pi \vdash \phi$.*

*Proof.* See Appendix A.2 from page 217 to page 231 □

# 6

# $\mathcal{ETR}$ usage and the role of (external) oracles

Both the semantics and the proof procedure for $\mathcal{ETR}$ are parametric on three oracles: two of them defining the behavior of the internal KB, and a third one defining the behavior of the external environment. To be able to make any practical use of $\mathcal{ETR}$, one has to instantiate each of these oracles in such a way that they model both the behavior of the internal KB and the behavior of the external environment intended for that specific usage.

In this chapter, for the sake of illustration, we make the exercise of fully describing each of these oracles for a specific internal KB and for some external environments. Note that, while instantiating the internal oracles is required to use $\mathcal{ETR}$ (or $\mathcal{TR}$), the external oracle can be left open if all we want is to execute an $\mathcal{ETR}$ program. This is as expected since in most cases it is impossible to know how the external environment behaves, or how the external oracle is specified. However, , whenever possible, to reason about general properties of $\mathcal{ETR}$ transactions, like equivalence or implication, an external representation of states must be chosen and the external oracle must be properly defined. This leaves open the question of how this external oracle $\mathcal{O}^e$ can be instantiated and what semantics are useful to characterize the external KB.

Inspired mainly by the possible usage of $\mathcal{ETR}$ for the two application domains – Semantic Web, and intelligent agent systems – next we explore several oracle possibilities for the external environment that go beyond the approach adopted until now, i.e., where one knows nothing about how the external world behaves, and where the external oracle is seen as a "black-box".

In particular, for a Semantic Web context, we show how the internal oracles can be

instantiated to behave has a *DL-Lite* database, where the interaction with this database is made by either asking conjunctive queries to the *DL-Lite* database, or by updating the ABox part of the database. Afterwards, we instantiate the external oracle to behave as another *DL-Lite* database, but now considering the implications of dealing with an external domain. Consequently, as an *external* database, one can interact with it by querying or attempting to change the ABox, but without having complete control over these interactions.

For other application domains, like intelligent agents, the external environment requires a much richer interaction and diversity of actions that are not restricted to querying and (trying) to add or delete information from external KBs. In this context, several languages for describing this diversity of actions, as well as their effects in external domains, have been defined and studied in the literature. To make $\mathcal{ETR}$ fully integrated with external environments whose interaction is established by this variety of actions, one has to define external oracles for such languages. With this in mind, in this chapter we also provide $\mathcal{O}^e$ specifications for Action Languages [GL98a], Situation Calculus [McC63], and Event Calculus [KS86]. This list is surely non-exhaustive, as several other oracle formalizations are possible.

Afterwards, in chapter 7, we explain how one can use $\mathcal{ETR}$ to automatically infer compensations, based on the previous Action Language instantiation.

## 6.1 $\mathcal{ETR}$ Oracles for the Web

A basic requirement of the Semantic Web is the ability to reason and retrieve knowledge simultaneously from multiple web-sources described using one of the several W3C standards. Additionally, the need to reason differently according to the internal or external provenance of knowledge has been the primer motivation for the works of [HPS-BTGD+04; DAAW06; GHVD03], aiming to integrate closed and open world reasoning for this Web context. Simply put, closed world reasoning assumes that everything that is not known to be true is false. On the contrary, open world reasoning denies this principle by assuming that the current description of the world is incomplete and thus, the lack of ability to infer knowledge never implies falsity. While this latter reasoning makes sense in a open web context where one cannot assume to have complete knowledge of the environment, this is not the case when reasoning about internal knowledge. Since we fully control internal information, employing closed world assumption is useful and much more natural. To address this, several semantics like [MHRS06; MR07; KAH11] have been proposed to reason and obtain knowledge from the so called *hybrid knowledge bases*, i.e., knowledge bases described by both a non-monotonic internal KB (defined by rules) and a monotonic external Web Ontology (defined by a Description Logic).

Furthermore, the highly dynamic facet inherent to a Web environment has triggered the appearance of update operators proposals for updating and revising knowledge over Description Logics [GLPR09; LLMW11; LS12] but also over these hybrid knowledge

bases [SL10; SLS11].

Based on the development of such operators, the following step is to provide transactional properties over such evolution of knowledge. In this sense, by providing means to reason and execute transactions defined over internal and external independent domains, $\mathcal{ETR}$ can be used to achieve this goal. In fact, the flexibility obtained by the inclusion of oracles allows $\mathcal{ETR}$ to be suitable, independently of which W3C standard is selected for the particular moment. Also, it should be noted that achieving different transactional properties over actions, depending on whether they are internal or external, is in line with the previous arguments for the combination of closed and open world reasoning.

For such Semantic Web usage, specific instantiations of the oracles are in order. With this in mind, and since depending on the application Description Logics can be used to describe both the internal and external KB, next we start by providing an example of a Description Logic instantiation for these domains.

### 6.1.1 Description Logics Oracles

With the goal to make $\mathcal{ETR}$ useful in a Semantic Web context, we exemplify how the internal oracles can be defined for a Description Logic KB. Description Logics [BCMNPS03] have been largely used to describe knowledge in the Semantic Web and are the underlying representation formalism of the standard Web Ontology Language (OWL) [MVH+04].

As decidable subsets of first-order logic, Description Logics comprise a family languages with different expressivity and complexity features. Every Description Logic (DL) knowledge base $\mathcal{K}$ is composed of knowledge described over a TBox ($\mathcal{T}$) and an ABox ($\mathcal{A}$). Here, the TBox defines the concepts and terminologies of the world while the ABox defines assertions of particular instances.

Based on just this, we can abstractly define the database oracle $\mathcal{O}^d$ as a mapping from a DL knowledge base $\mathcal{K}$ to the set of formulas true in that state. Then, a state identifier $D$ can be defined as the pair $\langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox, and $\mathcal{O}^d$ is such that a formula is true in it iff it is a consequence of the TBox plus the ABox.

Different oracle instantiations can be defined for different Description Logics. Here, for the purpose of this illustration, we provide an instantiation for the *DL-Lite* Family [CDGLLR07]. *DL-Lite* is the backbone of the OWL-2 QL profile [FHH04] and known for its low computational complexity on large volumes of instance data (ABox size). OWL 2 [MPSPBFHHHRS+09] is the second edition of the standard OWL and is fragmented upon three different profiles with the goal to address different application requirements. In this sense, the OWL 2 QL is designed to deal with very large amounts of data and in contexts where query answering is the most important task.

*DL-Lite* also defines a family of languages and thus to be concrete let us pick *DL-Lite*$_{\mathcal{FR}}$ that enjoys from polynomial algorithms to update the ABox [GLPR09; CKNZ10].

#### 6.1.1.1 *DL-Lite* knowledge bases

As any DL language, elementary descriptions are partitioned between atomic *concepts* and atomic *roles*, and complex descriptions can be built from these using concept constructors. To build complex descriptions, *DL-Lite*$_{\mathcal{FR}}$ has the following constructs:

$$\begin{aligned} B &::= A \mid \exists R \\ C &::= B \mid \neg B \\ R &::= P \mid P^- \end{aligned}$$

where $A$ denotes an atomic concept, $B$ a basic concept, $C$ a general concept, $P$ an atomic role and $R$ a basic role. Based on these, an ABox $\mathcal{A}$ is a set of membership assertions of the form $B(a)$ and $P(a, b)$ where $a$ and $b$ are object constants; and a TBox $\mathcal{T}$ is a set assertions of the form:

| | | |
|---|---|---|
| $B$ | $\sqsubseteq C$ | concept inclusion assertion |
| $R_1$ | $\sqsubseteq R_2$ | role inclusion assertion |
| (`funct` $R$) | | role functionality assertion |

The semantics is defined by first-order logic interpretations. An interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ has a non-empty domain $\Delta$, and a mapping $\cdot^{\mathcal{I}}$ from individuals, concepts and roles to $\Delta$ as follows:

$$\begin{aligned} A^{\mathcal{I}} &\subseteq \Delta \\ P^{\mathcal{I}} &\subseteq \Delta \times \Delta \\ (P^-)^{\mathcal{I}} &= \{(a_2, a_1) \mid (a_1, a_2) \in P^{\mathcal{I}}\} \\ (\exists R)^{\mathcal{I}} &= \{a \mid \exists a'.(a, a') \in R^{\mathcal{I}}\} \\ (\neg B)^{\mathcal{I}} &= \Delta \setminus B^{\mathcal{I}} \end{aligned}$$

To simplify, we assume *standard names*, i.e., we assume that there is no distinction between the alphabet of constants and $\Delta$. An interpretation $\mathcal{I}$ satisfies a given TBox or ABox assertion $F$ (denoted by $\mathcal{I} \models F$) if the following is true.

- $\mathcal{I} \models D_1 \sqsubseteq D_2$, if $D_1^{\mathcal{I}} \subseteq D_2^{\mathcal{I}}$
- $\mathcal{I} \models$ (`funct` $R$), if $(a_1, a_2) \in R^{\mathcal{I}}$ and $(a_1, a_3) \in R^{\mathcal{I}}$ implies $a_2 = a_3$
- $\mathcal{I} \models B(a)$, if $a \in B^{\mathcal{I}}$
- $\mathcal{I} \models R(a, b)$, if $(a, b) \in R^{\mathcal{I}}$

An interpretation is a model of a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ (written $\mathcal{I} \models \mathcal{K}$) iff it satisfies all the assertions in $\mathcal{K}$. We say that $\mathcal{K}$ is *satisfiable* if it has at least one model, and *unsatisfiable* otherwise.

#### 6.1.1.2 *DL-Lite* database oracle

The previously defined *DL-Lite* database is needed to define the notion of an $\mathcal{ETR}$ state. Moreover, besides the notion of what is a state, we also need to specify what are the primitives that connect programs (or $\mathcal{ETR}$ formulas) and the knowledge base. Interaction with the DL oracle is made by either asking conjunctive queries to the *DL-Lite* database, via a `dlquery`() primitive, or by updating the ABox of the database, via a `dladd`() primitive.

For the `dlquery`() primitive, we can make use of *DL-Lite* features and employ a query-answering algorithm for conjunctive queries as defined in [CGLLR05]. A conjunctive query $q(\vec{x})$ over the KB $\mathcal{K}$ is an expression of the form:

$$q(\vec{x}) \equiv \exists \vec{y}.conj(\vec{x}, \vec{y})$$

where $\vec{x}$ and $\vec{y}$ are known as the *distinguished variables* and *non-distinguished variables* of the query, respectively; and $conj(\vec{x}, \vec{y})$ is a conjunction of atoms of the form $B(z)$ or $R(z_1, z_2)$ where $B$ is a basic concept and $R$ a role in $\mathcal{K}$, and $z, z_1, z_2$ are either constants in $\mathcal{K}$ or variables in $\vec{x}$ or $\vec{y}$.

The answers to a conjunctive query $q(\vec{x})$ in a knowledge base $\mathcal{K}$, denoted by $\texttt{ans}(q(\vec{x}), \mathcal{K})$ (or by $\texttt{ans}(\exists \vec{y}.conj(\vec{c}, \vec{y}), \mathcal{K})$), is the set of tuples $\vec{c} \in \Delta \times \ldots \times \Delta$ such that when the variables $\vec{x}$ are substituted with the constants $\vec{c}$, the formula $\exists \vec{y}.conj(\vec{x}, \vec{y})$ is true in every $\mathcal{I}$ that is a model of $\mathcal{K}$.

For asking conjunctive queries, our primitive takes the form $\texttt{dlquery}(\vec{c}, conj(\vec{c}, \vec{y}))$ where $conj(\vec{c}, \vec{y})$ is a conjunctive query, and $\vec{c}$ the set of constants that appear in it. With this, the database oracle $\mathcal{O}^d$ is defined as follows.

**Definition 24** (*DL-Lite* database oracle)**.** *Let $\mathcal{K}$ be a state (i.e., a TBox and an ABox in DL-Lite).*

$$\mathcal{O}^d(\mathcal{K}) \models \texttt{dlquery}(\vec{c}, conj(\vec{c}, \vec{y})) \textit{ iff } \texttt{ans}(\exists \vec{y}.conj(\vec{c}, \vec{y}), \mathcal{K}) \neq \emptyset$$

Note that, in this definition we are only dealing with boolean DL queries. This is because, from the start, we are working with Herbrand interpretations of the transaction logic formulas. As such, all rules are ground, and so are the DL queries possibly appearing in them. For the general case of rules with variables, a condition similar to DL-safety [MSS05] would have to be imposed, so as to guarantee that the instantiation of the variables in $\mathcal{ETR}$ rules would not depend on the result of the queries. In this context, DL-safety must guarantee for every transaction rule that every variable in $\vec{x}$ of a query is instantiated before the query call. This implies that every variable in $\vec{x}$ occurs before (i.e., in the rule head, or in the body before in the sequence of $\otimes$) in a predicate defined in $\mathcal{L}_P$. Since we only present the ground version of $\mathcal{ETR}$, and since for that the discussion on safety is not crucial, we do not elaborate further on this topic.

#### 6.1.1.3 *DL-Lite* transition oracle

For the `dladd()` primitive, we restrict it to instance-based updates, i.e., to updates on the membership assertions of the ABox. For those, an update $\mathcal{U}$ is simply a set of ABox assertions that is integrated into the current knowledge base $\mathcal{K}$, obtaining a new knowledge base $\mathcal{K}'$.

However, the updated information may leave $\mathcal{K}'$ unsatisfiable, and, in this case, the conflicts between $\mathcal{U}$ and the old information from $\mathcal{K}$ need to be addressed. Since the resulting knowledge base $\mathcal{K}'$ may not be expressible in the original DL where it was defined [BLMSW05], solving such conflicts may be difficult (even for the simpler case of ABox updates). To address this problem, several formal operators have been proposed either based on models and on formulas updates, and the interested reader is referred to [LLMW11] for more details.

For the purpose of this illustration, we assume the Careful Semantics Update as presented in [CKNZ10]. Nevertheless, note that any other update semantics could be as easy defined (based either on TBox, ABox updates, or both).

A careful update is defined for a *DL-Lite*$_{\mathcal{FR}}$ $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ as follows:

$$\texttt{c\_upd}(\mathcal{T}, \mathcal{A}, \mathcal{U}) := \mathcal{A}^c_m \cup \mathcal{U}$$

where $\mathcal{A}^c_m$ is the careful maximal set of assertions subset of the closure of $\mathcal{A}$ w.r.t. $\mathcal{T}$ compatible with $\mathcal{U}$. Based on this, we formally define `dladd()` as follows.

**Definition 25** (*DL-Lite* transition oracle)**.** *Let a state $S$ be a pair $\langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. Let $\mathcal{U}$ be a set of ABox assertions to update the ABox and* `c_upd` *the careful semantics update algorithm referenced earlier.*

$$\mathcal{O}^t(\langle \mathcal{T}, \mathcal{A} \rangle, \langle \mathcal{T}, \mathcal{A}' \rangle) \models \texttt{dladd}(\mathcal{U}) \textit{ iff } \mathcal{A}' = \texttt{c\_upd}(\mathcal{T}, \mathcal{A}, \mathcal{U})$$

#### 6.1.1.4 *DL-Lite* as an external oracle

In a Semantic Web environment, the external KB can also be described by a Description Logic via instantiation of $\mathcal{O}^e$. This is e.g. the case when we have an ontology distributed across several web-sources, or when a rule system interacts with an ontology on the web (like e.g. in hybrid knowledge bases). Consequently, in the following we define an $\mathcal{O}^e$ for a *DL-Lite* Description Logic similar to what we have done in the previous sections.

Particularly, if we consider the external environment to be described by a *DL-Lite* database, on which one can freely perform queries and updates, then defining the external oracle is just equivalent to defining the internal oracles $\mathcal{O}^d$ and $\mathcal{O}^t$ for that given DL. The only minor difference is that, since $\mathcal{O}^e$ works with pairs of states even when evaluating queries, querying $\mathcal{O}^e$ is equivalent to the definition of $\mathcal{O}^d(\mathcal{K})$ for a given state $\mathcal{K}$, but where the pair state does not change, i.e., $\mathcal{O}^e(\mathcal{K}, \mathcal{K})$.

However, even if the external environment is made up of ontologies described by

some Description Logic, it is not common that others can freely query and update them. In this sense, it may be useful to restrict the set of information that can be updated in $\mathcal{O}^e$ by external entities. This is a common feature of several systems where one is required to authenticate in order to have permission to access and change a given table, tuple or webpage. To achieve this, we can define in $\mathcal{O}^e$ a permission list, linking users to sets of ABox assertions that can be modified. Then, when updating the KB we must verify that the update is allowed, i.e., that only the permitted assertions are modified. Such an external oracle for a *DL-Lite* DL can be defined as follows:

$$\begin{aligned} \mathcal{O}^e(\mathcal{K},\mathcal{K}) &\models \texttt{dlquery}(\vec{c}, conj(\vec{c},\vec{y})) && \text{iff } \texttt{ans}(\exists\vec{y}.conj(\vec{c},\vec{y}),\mathcal{K}) \neq \emptyset \\ \mathcal{O}^e(\langle\mathcal{T},\mathcal{A}\rangle,\langle\mathcal{T},\mathcal{A}'\rangle) &\models \texttt{dladd}(\mathcal{U}) && \text{iff } \mathcal{A}' = \texttt{c\_upd}(\mathcal{T},\mathcal{A},\mathcal{U}) \wedge \\ & && \quad allowed(\mathcal{A},\mathcal{A}',list(user)) \end{aligned}$$

where $allowed(\mathcal{A},\mathcal{A}',List(User))$ denotes that every assertion that is present in $\mathcal{A}'$ and not in $\mathcal{A}$ is defined in the allowed list for that user. Clearly, other possibilities could be defined and explored. For example, besides a permission list for atoms that can be updated, one can also have permissions for atoms to be queried, either by a simple list, or with more sophisticated mechanisms. This, however, is beyond the scope of this thesis.

As previously argued, other languages and domains are also possible in the external KB. Thus, in the following we illustrate how the external oracle can be defined for languages with the aim of encoding the dynamic effects of actions, as Action Languages, and Situation and Event Calculus.

## 6.2 Oracles for Intelligent Agents

Intelligent agents in a multi-agent setting normally must work and reason over a two-fold environment: an external environment, representing the outside world where the agent acts, and which may include other agents; and an internal environment comprising the information about the agent's rules of behavior, preferences about the outside world, its knowledge and beliefs, intentions, goals, etc. In this context, an agent may act on the external environment (by executing external actions), but also on the internal environment (where it executes internal actions). Examples of the latter include insertions and deletions in the agent's own knowledge base, and updates on its rules of behavior or preferences.

Clearly, when performing actions, agents must take into account the possibility of action failure, and what do to upon such situation. This is especially relevant inasmuch as the agent has no control over the behavior of the external world. Consequently, external actions may e.g. fail because their preconditions are not met at the time of intended execution or, in norm regimentation, because the execution of the action would cause the violation of some norm (e.g. as allowed by 2OPL [DMG13]), or even due to a totally unknown reason to the agent.

The failure of an action should trigger some kind of "repair plan" which should undo the effects have been caused by the failed action. This is particularly important when the action is part of a plan, in which case it may be necessary to undo the effects of previous actions that have succeeded. When the action to undo is an internal action, the undo should be trivial. In fact, since the agent has full control over its own internal environment, actions and updates can be made to follow the standard transaction properties in databases and, as such, the effects made by internal actions are completely discarded. However, since in general an agent has no control over the external environment, such transactional properties cannot be guaranteed when undoing external actions.

The idea of repair plans and reversing actions to achieve consistency is already presented in the literature of agents reasoning about actions, as e.g. in [HCO04; BWH07; Das08; EEF08]. Nonetheless, these solutions only deal with external domains, and thereby are not able to model situations where an agent can reason and execute updates in an internal KB that he fully controls and where rolling back is a more convenient mean to restore consistency.

Given this, we argue that $\mathcal{ETR}$ can be used in such intelligent agent domains, by providing means to reason about and execute transactions that involve interactions with an internal and an external component. In this context, $\mathcal{ETR}$ can guarantee properties regarding the outcome of the agent's actions. Namely, it can guarantee that all internal actions follow the standard transaction model, while all external actions are compensated in case of the occurrence of some failure (if these compensations exist).

For this usage of ETR, we need to pick the oracle instantiations that can be useful in these domains.

While the external oracle $\mathcal{O}^e$ can be left open if the agent knows nothing about the external environment, in this section and for the sake of illustration, we consider the case where such knowledge about the external world exists, and it can be formalized in domain description languages like Situation Calculus [McC63; Rei91], Event Calculus [KS86] or Action Languages [GL98a]. These languages have become popular to capture and reason about dynamic (external) domains by their ability to, very expressively, formalize actions and their (possibly indirect) effects in the domains. As such, they are natural candidates to be used as the semantics of the external domain of an intelligent agent.

Moreover, and since normally the major difficulty here is how to represent the external world and the agent's interactions with this world, for this section we also fix the internal oracle as the relational database oracle defined on page 28.

### 6.2.1 Action Languages Oracles

We start by showing how an $\mathcal{ETR}$'s external oracle can be defined to incorporate Action Languages [GL98a], which afterwards will serve as the basics to define an external oracle to automatically infer repair plans in section 7.2.

Every action language defines a series of *laws* describing actions in the world and their effects. Which laws are possible as well as the syntax and semantics of each law depends on the action language in question. Several solutions like STRIPS, languages $\mathcal{A}, \mathcal{B}, \mathcal{C}$ or $PDDL$, have been proposed in the literature, each with different applications in mind. A set of laws of each language is called an action program description. Then, the semantics of each language is determined by a transition system which depends on the action program description.

Let $\langle \{\texttt{true}, \texttt{false}\}, \mathcal{F}, \mathcal{A} \rangle$ be the signature of an action language, where $\{\texttt{true}, \texttt{false}\}$ are the set of possible truth values, $\mathcal{F}$ is the set of fluent names and $\mathcal{A}$ is the set of action names in the language. Let $\langle S, V, R \rangle$ be a transition system where $S$ is the set of all possible states, $V$ is the evaluation function from $\mathcal{F} \times S$ into $\{\texttt{true}, \texttt{false}\}$, and finally $R$ is the set of possible relations in the system defined as a subset of $S \times \mathcal{A} \times S$. Based on this, we assume a function $\mathcal{T}(E)$ that from action program $E$ defines the transition system $\langle S, V, R \rangle$ associated with $E$, and the previously defined signature.

Intuitively $\mathcal{T}(E)$ defines an action languages description. Relations in $R$ have the form $\langle s, A, s' \rangle$, where $s'$ is denoted as the result of the execution of $A$ in the state $s$. Actions can also be non-deterministic, and in this case a number of end states $s'$ exist for the same action $A$ and the same starting state $s$. There is also a notion of executability, and an action is said to be executable in state $s$ if there is at least one tuple $\langle s, A, s' \rangle$ in $R$.

Based on these notions, we define $\mathcal{ETR}$'s external language $\mathcal{L}^a$ as the union of the set of possible fluents $\mathcal{F}$, and the set of possible actions, i.e., $\mathcal{L}^a = \mathcal{F} \cup \mathcal{A}$ where as usual, fluents are queries that never change the external state, while actions may cause an external state transition. Then, equipped with a function $\mathcal{T}(E)$, we can define the external state component of an $\mathcal{ETR}$ state pair, by an action language. For that, we define an $\mathcal{ETR}$ external action language state also as a pair $(E, s)$, where $E$ is an action language program description describing the external domain, and $s$ is a state of the transition system. Based on this alone, the general external oracle $\mathcal{O}^e$ can be as follows:

**Definition 26** (General Action Language External Oracle)**.** *Let $\langle \{\texttt{true}, \texttt{false}\}, \mathcal{F}, \mathcal{A} \rangle$ be the signature of an action language. Let $E$ be an action language program description, $\mathcal{T}(E)$ a function that maps an action program $E$ into a transition system $\langle S, V, R \rangle$ and let $s$ be a state from $S$.*

*Then, an external oracle $\mathcal{O}^e$ for a general Action Language is defined as:*

$$\begin{array}{lllll} \mathcal{O}^e((E,s),(E,s')) & \models action & \textit{iff} & action & \in \mathcal{A} \wedge \langle s, action, s' \rangle \in R \\ \mathcal{O}^e((E,s),(E,s)) & \models fluent & \textit{iff} & fluent & \in \mathcal{F} \wedge V(fluent, s) = \texttt{true} \end{array}$$

The previous definition specifies an $\mathcal{ETR}$ external oracle $\mathcal{O}^e$ for any action language framework, based on a transition system defined as $\langle S, V, R \rangle$. Such specification is still very general and thus, to be concrete, let us show one instantiation of this, with action language $\mathcal{C}$ [GL98b]. In the context of multi-agent systems, language $\mathcal{C}$ and its extensions like $\mathcal{C}^+$ [GLLMT04], are traditionally used to represent norms and protocols (e.g. auction,

contract formation, negotiation, rules of procedure, communication, etc.) [SC06; ASP09].

In action language $\mathcal{C}$, formulas are partitioned between state formulas and formulas. A *state formula* is a propositional combination of fluent names, while a *formula* is a propositional combination of fluent names and elementary action names. Then, an external description $E$ is a set of static and dynamic laws. A *static law* is a law of the form "**caused** $F$ **if** $G$", where $F$ and $G$ are state formulas. A *dynamic law* is of the form "**caused** $F$ **if** $G$ **after** $U$", where $F$ and $G$ are state formulas and $U$ is a formula.

In this context, a static law is used to express constraints that hold in all states, whilst a dynamic law defines what changes and what stays the same after the execution of $U$.

An important notion is that the action language $\mathcal{C}$ supports concurrent actions. As a consequence of this, in a transition $\langle s_1, A, s_2\rangle$, $A$ is not read as an individual action but as a subset of $\mathcal{A}$. Intuitively, to execute $A$ from $s_1$ to $s_2$ means to execute concurrently the "elementary actions" represented by the action symbols in $A$ changing the state $s_1$ into $s_2$ (where all elementary actions in $A$ are assumed to have the same duration).

Additionally, a state $s$ in action language $\mathcal{C}$ is defined as the interpretation of the set of fluents $\mathcal{F}$ that is closed under the static laws. I.e., for every static law "**caused** $F$ **if** $G$" defined in the action description, and every state $s$, we say that $s$ must satisfy $F$ if $s$ satisfies $G$. Based on this, the interpretation function $V$ for a state is simply defined as $V(fluent, s) = s(fluent)$.

Afterwards, the set of valid relations $R$ is defined based on the notion of *reduct* and causal explanation. For any description $E$ and any transition $\langle s_0, A, s_1\rangle$ we define the reduct of $E$ relative to $\langle s_0, A, s_1\rangle$ (denoted as $E^{\langle s_0,A,s_1\rangle}$), as the set consisting of:

- $F$ for all static laws from $E$ s.t. $s_1$ satisfies $G$
- $F$ for all dynamic laws from $E$ s.t. $s_1$ satisfies $G$ and $s_0 \cup A$ satisfies $U$

We say that $\langle s_0, A, s_1\rangle$ is *causally explained* if $s_1$ is the only state that satisfies the reduct $E^{\langle s_0,A,s_1\rangle}$.

Recall that states in action language $\mathcal{C}$ can be seen as interpretations, and thus any state $s$ contains the set of fluents that are true at a given moment w.r.t. the set of static laws (i.e., that are closed under the static laws of $A_p$). The latter means that the execution of $A$ in state $s_0$ leads to state $s_1$, if $s_1$ is completely characterized by the applicable laws given state $s_0$ and the set of actions $A$.

Additionally, since the external oracle is defined for elementary actions rather than for sets of actions, we can define the relation $R$ of $\mathcal{T}(A_p)$ as follows: $\langle s_0, a, s_1\rangle \in R$ iff $\langle s_0, A, s_1\rangle$ is causally explained w.r.t. $A_p$ and $a \in A$. Based on these notions, we can now redefine our external oracle for action language $\mathcal{C}$.

**Definition 27** (Action Language $\mathcal{C}$ External Oracle)**.**
*Let $\langle\{\texttt{true}, \texttt{false}\}, \mathcal{F}, \mathcal{A}\rangle$ be the signature of an action language. Let $E$ be an action language program description, containing a set of static laws and dynamic laws. Let $R$ be the set of tuples $\langle s_0, a, s_1\rangle$ such that $\langle s_0, a, s_1\rangle$ is causally explained w.r.t. $E$ and $a \in A$.*

*Then, an external oracle $\mathcal{O}^e$ for Action Language $\mathcal{C}$ is defined as:*

$$\begin{array}{lll} \mathcal{O}^e((E,s),(E,s')) & \models action & \textit{iff} \quad action \in \mathcal{A} \wedge \langle s, action, s' \rangle \in R \\ \mathcal{O}^e((E,s),(E,s)) & \models fluent & \textit{iff} \quad fluent \in \mathcal{F} \wedge s(fluent) = \texttt{true} \end{array}$$

### 6.2.2 Situation Calculus Oracle

In the seminal Situation Calculus [McC63], external domains are described in a second-order language with a basic ontology partitioned into *actions* ($\mathcal{A}$), *fluents* ($\mathcal{F}$) and *situations*. An action is a predicate that has the ability to change the state of the world, while a fluent is a predicate whose truth value can change over time (or more precisely, over situations). Finally, a situation represents a complete state of the universe at a given instance defined by a finite sequence of actions. More precisely, situations are either represented by a constant $s_0$ denoting an initial situation, or by $do(a, s)$ denoting the situation that results from executing action $a$ in situation $s$.

The conditions for executing actions, and their effects, are expressed using second order predicates $Poss(a, s)$, meaning that action $a$ can be executed in situation $s$, and $Holds(f, s)$, meaning that fluent $f$ is true in situation $s$.

The semantics of these predicates and operators is defined by *axioms* describing the world, actions and their effects. For the purpose of this illustration, we do not elaborate on how these axioms are defined or how the frame problem is solved, and refer e.g. to [Rei91] for more details. All we need is a satisfaction relation $\models_{SitCal}$ that satisfies primitive formulas w.r.t. a set of axioms that we define as a domain description $E$. Intuitively, a domain description is just a set of action axioms, domain axioms and frame axioms.

Based on this, the external language is $\mathcal{L}_a = \mathcal{A} \cup \mathcal{F}$, and external states are pairs $(E, S)$, where $S$ is a situation and $E$ is the external domain description. Finally, an external oracle based on Situation Calculus can be given by:

1. $\mathcal{O}^e((E,S)),(E,S)) \models f$ iff $f \in \mathcal{F}$ and $E \models_{SitCal} Holds(a, S)$
2. $\mathcal{O}^e((E,S_1),(E,S_2)) \models a$ iff $a \in \mathcal{A}$ and $E \models_{SitCal} Poss(a, S_1) \wedge S_2 = do(a, S_1)$

Note that the external oracle only executes actions that are *possible* to be executed in a given situation $s_1$. This precludes the system to evolve into an inconsistent situation that results from an action that is not allowed in that state. This also results in the possibility of failed external actions, which are then dealt in $\mathcal{ETR}$ by rolling back the internal KB and executing compensating actions externally.

Equipped with a formalism that is able to deal both with internal KBs, with transactions, and with external actions, let us show a simple illustrative examples of what it can express, and how results are obtained.

**Example 21** (Medical Diagnosis)**.** *Recall example 13. After defining the transaction rules and the internal knowledge described over a Description Logic, we left open the definition of the external KB.*

*This external KB, describing the effects of actions, and also some facts about the patient, can be modeled using Situation Calculus descriptions, e.g. including:*

$$Holds(temperature(sam, 39), s_0).$$
$$Holds(hasHeadache(sam, t), s_0).$$
$$Holds(stuffyNose(sam, t), s_0).$$
$$Holds(heartRate(sam, 80), s_0).$$
$$Holds(dyspnea(sam, f), s_0).$$
$$Holds(heartRate(sam, 160), \texttt{do}(giveMeds(sam, plp), s_0)).$$
$$Holds(dyspnea(sam, t), \texttt{do}(giveMeds(sam, plp), s_0)).$$

*Given this instantiation of the external domain, the system will conclude that $Sam$ likely suffers from flu and thus it may decide to give the medicine $plp$ as treatment. If this is the case, then Sam will experience some symptoms of heart failure: dyspnea (difficulty of breathing) and increase of heart rate. Note that if this happens, it is crucial to perform some compensation in order to make Sam feel better. In this case, the system can give Sam $cplp$ that is known to address the effects of a heart failure resulting from giving $plp$.*

### 6.2.3 Event Calculus Oracle

Similarly, in Event Calculus [KS86] predicates can be *actions* or *fluents*, where actions can change properties of the world and fluents denote these properties whose truth value may change. The main innovation of Event Calculus is that actions are *events*, i.e., changes associated with a particular moment in time that influence the state of the world. Then, fluents are evaluated w.r.t. time points usually defined by non-negative real numbers and denoting an explicit moment in the system.

One can describe the external domain in Event Calculus by using the predicates[1]: $initially(f)$, denoting that fluent $f$ holds at time 0; $initiates(f, a, t)$, stating that action $a$ initiates fluent $f$ at time $t$; $terminates(f, a, t)$ stating that action $a$ terminates fluent $f$ at time $t$; and $happens(a, t)$ denoting that action $a$ happened at time $t$. Truth of fluents at time points is obtained by the predicate $holdsAt(f, t)$, whose meaning can be obtained by a logic program:

$$\begin{aligned}
holdsAt(P, T) &\leftarrow 0 \leq T, initially(P), \textbf{not } clipped(0, P, T).\\
holdsAt(P, T) &\leftarrow happens(E_1, T_1), T_1 < T, initiates(E_1, P, T_1), \textbf{not } clipped(T_1, P, T).\\
clipped(T_1, P, T) &\leftarrow happens(E_2, T_2), T_1 < T_2, T_2 < T, terminates(E_2, P, T_2).
\end{aligned}$$

Based simply on this, one may represent states of an external domain described in Event Calculus as a pair, where the first argument is a logic program $P$ containing the description of the domain, and the second argument as a time point $t$. The definition of the oracle itself can be done in a very similar way as in the Situation Calculus case, by:

[1]We assume the simplified version of the calculus as defined in [Sha99]. More basic predicates can be found in the full version of the calculus.

1. $\mathcal{O}^e((P,t),(P,t)) \models p$ iff $P \vdash_{LP} holdsAt(p,t)$
2. $\mathcal{O}^e((P,t),(P',t+1)) \models a$ iff $P' = P \cup \{happens(a,t)\}$

Some words are in order regarding this representation of (external) states. Internal formulas (i.e., queries evaluated in $\mathcal{O}^d$, updates evaluated in $\mathcal{O}^t$, or complex formulas combining these) do not change the external state. Consequently, with our representation of states, and from the perspective of the external domain, the evaluation of all these formulas are instantaneous. In other words, this definition does not cater for cases where the external domain changes while the formulas are being evaluated. Allowing changes in the external world to occur simultaneously with the evaluation of internal formulas would require some explicit representation of the external time in the formulas of $\mathcal{ETR}$ theory, as well as a global clock with the role to instantiate correctly the time component of the external state, and this is beyond the scope of this illustration.

Another aspect worth discussing is that with this formalization of the oracle, "actions" never fail. This is so because the Event Calculus was primarily defined to reason about events, and thus it makes no sense for the occurrence of an event to fail. However, $\mathcal{ETR}$, as a logic that talks about actions and state change, assumes that actions (and, especially, external actions) can fail. In fact, it is important for the internal knowledge base to know whether a given external action can be successfully executed. Without this possibility, the need to ensure transaction properties externally, as well as the notion of compensation become redundant.

With this in mind, to include the possibility of failure, we extend the Event Calculus oracle, with $executable(A,T)$ to express that action $A$ can be executed at time $T$. Since in the end, Event Calculus can be defined as a logic program, incorporating a new predicate is as simple as defining a new rule as $executable(A,T) \leftarrow preconditions$, where the $preconditions$ denote the set of preconditions that need to be true in order for $executable(A,T)$ succeed. For instance, in the well-known Yale shooting problem [HM87], one can express the possibility of killing turkey Fred as follows:

$$executable(kill,T) \leftarrow holdsAt(alive,T), holdsAt(loaded,T).$$

Based on this, an alternative version of $\mathcal{O}^e$ can be defined as:

1. $\mathcal{O}^e((P,t),(P,t)) \models p$ iff $P \vdash_{LP} holdsAt(p,t)$
2. $\mathcal{O}^e((P,t),(P',t+1)) \models a$ iff $P \vdash_{LP} executable(a,t) \wedge P' = P \cup \{happens(a,t)\}$

**Example 22** (Ski Resort Hotel)**.** *Consider the scenario of a hotel in a ski resort where the internal KB manages room reservations. Given the location of the hotel, one possible package is to combine a hotel room with the acquisition of the ski pass for the resort. Moreover, the price of this package depends on the dates of the calendar (if it is high season or not) but also on the amount of snow on the slopes. If the amount of snow on the slopes is higher than 100cm then the quality is considered "premium", and the Hotel takes this opportunity to increase the price of the ski-pass reservation*

*by 30%. However, if the slopes are closed due to some storm or lack of snow, the ski pass cannot be sold.*

*Ski passes are external to the system and handled by the external environment which also gives information about the resort, namely: the quantity of snow on the slopes and if the resort is open or not.*

$$
\begin{array}{rcl}
priceFF(Price,T) & \leftarrow & \mathbf{ext}(\texttt{isOpen}) \otimes \mathbf{ext}(\texttt{snowCM(CM)}) \otimes\ CM <= 100 \otimes \\
 & & basePrice(Price,T) \\
priceFF(Price,T) & \leftarrow & \mathbf{ext}(\texttt{isOpen}) \otimes \mathbf{ext}(\texttt{snowCM(CM)}) \otimes\ CM > 100 \otimes \\
 & & basePrice(P,T) \otimes Price\ is\ 1.3 * P \\
reservation(N,T,X) & \leftarrow & priceFF(PF,T) \otimes priceHotel(PH,T) \otimes \\
 & & addResHotel(N,T,PF+PH) \otimes \mathbf{ext}(\texttt{printFF},\texttt{cancelFF}) \\
 & & \otimes\ X = PF + PH \otimes\ \mathbf{ext}(askPayment(N,X)) \\
addResHotel(N,T,X) & \leftarrow & roomsAvailable(Nr) > 0 \otimes roomsAvailable(Nr).del \otimes \\
 & & roomsAvailable(Nr-1).ins \otimes reservation(N,T,X)
\end{array}
$$

*In this case, the external domain of the ski resort could be described by an Event Calculus program with the following rules:*

$$
\begin{array}{rcl}
holdsAt(isClosed,T) & \leftarrow & holdsAt(stormStart,T_1), T_1 \leq T, T_1 \leq T_2 \leq T, \\
 & & \mathbf{not}\ holdsAt(stormEnd,T_2). \\
holdsAt(isOpen,T) & \leftarrow & \mathbf{not}\ holds(isClosed,T). \\
holdsAt(stormStart,150313). & & holdsAt(stormEnd,180313). \\
holdsAt(snowCM(10,150313). & & holdsAt(snowCM(150,183113).
\end{array}
$$

*External predicates like* `isOpen` *and* `snowCM(CM)` *rely on weather conditions whose truth value naturally depend on moments in time. In the example we know that between 15th and 18th of March a snow storm occurred. During this snow storm the resort was closed and thus the hotel was unable to sell reservations with ski passes for that period. However, after this storm, the amount of snow increased and the slopes on the 18 of March had around 1,5 meters of fresh snow which led to more expensive reservations.*

*Note that time is an important component of this system. It is assumed that a shared clock exists for both internal and external component. Then, whenever a new reservation request* $reservation(name,time)$ *is posed, the system must check whether the program executionally entails this transaction, given an initial internal state and external with a common appropriate value for* $time$*.*

## 6.3 Combining $n-$ary External Oracles

Some remarks are in order about the possibility of combining several external oracles to encode the dynamics of the external environment. Up until now, we have only consider external oracles that are described by a single semantics. However, this is not necessarily

the case, and the previous example 22 shows a situation where more than one external semantics is required to describe the external domain. Particularly, in such an example, besides the ski resort, the hotel system interacts with one more external entity: the client. Moreover, it is clear that the external action of asking the payment to a client is performed in a completely independent domain than the action `printFF` which requests the printing of a ski pass. Other examples of this need are common e.g. in the Semantic Web context, where a system needs to combine knowledge published across different web-sources described over different W3C standards.

Although formally $\mathcal{ETR}$ only supports integration with one external oracle, nothing prevents this oracle from being instantiated with more than one external semantics. This can be done by partitioning the external KB language ($\mathcal{L}_a$) into as many languages as needed. Then, the oracle $\mathcal{O}^e$ works as a "meta-oracle" deciding in which semantics a formula should be evaluated. In the case of example 22, to define the two domains, viz. the ski resort and the client, then two sub-oracles (one for each domain) must be defined and incorporated within $\mathcal{O}^e$.

With this in mind, one can assume a disjoint language on the two sub-oracles, allowing $\mathcal{O}^e$ to simply decide in what semantics each formula must be evaluated. Obviously, this approach can also be used to employ an arbitrary number of oracles.

# 7

# Automatic compensations in $\mathcal{ETR}$

In the previous chapters, we assumed that all compensations to be executed whenever a failure occurs, are explicitly defined by the programmer. This is natural in situations like, e.g. in example 11 of chapter 4, where "what to do to counter the side-effects of a previous unsuccessful treatment" is something that is necessarily stated by whoever programmed the agent. In other words, in that example it is reasonable to assume that the treatment can only be repaired if the agent's specification explicitly includes what are the compensating actions, or *repair plan*, to be executed in case a given treatment fails.

Nevertheless, in situations where some knowledge of the external environment is available, it should be possible to automatically infer the repair actions in a given failure situation, thus saving the programmer from that task, and from having to anticipate all possible relevant failures.

**Example 23** (Supermarket Robot)**.** *Imagine a scenario of a robot in a supermarket that has the task to fill up the supermarket's shelves with products. In its internal KB, the agent keeps information about the products' stocks and prices, but also rules on how products should be placed (e.g. "premium" products should be placed in the shelves with higher visibility). Externally, the agent needs to perform the task of putting products in a given shelf, something that can be encoded in a blocks-world usual manner. In this case, when some action fails in the context of a plan for e.g. arranging the products in some way, the agent, knowing the effects of the actions in the outside world, should be able to infer what actions to perform in order to restore the external environment to some consistent configuration, upon which some other alternative plan can be started.*

Reversible computation [Ben73; Abr05] stands for the ability to proceed computation in both a forward and backward trajectory, and is an important topic in several research areas of computer science, like automata [KW14; AG11; Pin92], programming

languages [Yok10; AGY07; MHT04] or process algebras [PU07; DK04; CKV13].

In the context of multi-agent systems, several solutions exist in the literature addressing the problem of reversing and repair an agent behavior like [HCO04; BWH07; Das08], but all of these require the reversals to be explicitly encoded in the program. On the contrary, the authors of [EEF08] introduce a solution based on Action Languages [GL98a] that reasons about what actions may revert the effects of other actions. For that, the authors define the notions of reverse action, reverse plan and conditional reversals that undo the effects of a given action or set of actions.

Building on the notions of reversals of actions defined in [EEF08], in this chapter we propose an extension of $\mathcal{ETR}$ to automatically compute compensations in case of failure. For that, we introduce the necessary notions proposed in [EEF08] needed for our solution (section 7.1), and show how these notions can be adapted for the context of $\mathcal{ETR}$ (section 7.2). Then, we formalize how $\mathcal{ETR}$ can be used to automatically infer plans when the external environment is expressed as an action language (section 7.3), and finally, we elaborate on the properties of these repair plans (section 7.3.1).

Most definitions and examples appearing in this chapter were published in [GA13b; GA14a].

## 7.1 Reverse Actions in Action Languages

Since our automatic inference of repairs for external actions is based on the work of [EEF08], before defining how to automatically infer repair plans in $\mathcal{ETR}$, we briefly overview [EEF08]'s action reverses, adapting it for the action languages framework defined above.

We start with the notion of *trajectory* of a sequence of actions. Intuitively, we say that a state $s_f$ is the trajectory of a sequence of actions applied to state $s_i$ if there is a trace from $s_i$ to $s_f$ by executing the given sequence of actions. Since nondeterministic actions are possible, this trajectory function is a mapping from $S \times \mathcal{A}$ into $\wp(S)$, and the result of $\texttt{traj}(s_0; [a_0 \otimes \ldots \otimes a_{m-1}])$ is the set of possible end states when executing $a_0 \otimes \ldots \otimes a_{m-1}$ in state $s_0$.

**Definition 28** (Trajectory of a Sequence of Actions)**.** *We define the trajectory of a sequence of actions $a_0 \otimes \ldots \otimes a_{m-1}$ in state $s_0$ as the set $S$ (denoted as $\texttt{traj}(s_0; [a_0 \otimes \ldots \otimes a_{m-1}]) = S$) iff:*

$$\forall s_f \in S \textbf{ then } \exists s'_1, \ldots, s'_m \text{ s.t. } \langle s_0, a_0, s'_1 \rangle \in R \ \wedge \ \langle s'_i, a_i, s'_{i+1} \rangle \in R \text{ and } s'_m = s_f$$

*where $(1 \leq i \leq m-1)$.*

*We say that $s_f$ is a possible trajectory of $a_0 \otimes \ldots \otimes a_{m-1}$ if it can be reached when applied to $s_0$ by executing $a_0 \otimes \ldots \otimes a_{m-1}$, i.e., if $s_f \in \texttt{traj}(s_0; [a_0 \otimes \ldots \otimes a_{m-1}])$.*

With this we can define the notion of reverse action. A reverse action states a relation between two singleton actions based on the set of their transition relations. We say that

an action $a^{-1}$ is a reverse action of $a$ if whenever we execute $a^{-1}$ after we execute $a$, we always obtain the (initial) state before the execution of $a$. This is encoded as follows.

**Definition 29** (Reverse Action)**.** *Let $a, a^{-1}$ be actions in $\mathcal{A}$. We say that an action $a^{-1}$ reverses $a$ iff:*

$$\forall s_1, s_2 \text{ if} \langle s_1, a, s_2\rangle \in R \text{ then } \exists s.\langle s_2, a^{-1}, s\rangle \in R \text{ and } \forall s.\langle s_2, a^{-1}, s\rangle \in R, s = s_1$$

*In this case we write* $\texttt{revAct}(a; a^{-1})$.

Besides the notion of reverse action, the authors of [EEF08] also introduce the notion of reverse plan. Since a single action may not be enough to reverse the effects of another action, the notion of reverse is generalized into a sequence of actions, or *plan*. A reverse plan defines what sequences of actions are able to reverse the effects of one action. This is encoded as follows.

**Definition 30** (Reverse Plan)**.** *Let $a, a_0, \ldots, a_{m-1}$ be actions in $\mathcal{A}$. We say that $a_0 \otimes \ldots \otimes a_{m-1}$ is a plan that reverses action $a$ iff $\forall s_1, s_2$ s.t. $\langle s_1, a, s_2\rangle \in R$ then $\exists s'$ s.t. $s' \in \texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$ and $\forall s'$ s.t. $s' \in \texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$ then $s' = s_1$. In this case we write* $\texttt{revPlan}(a; [a_0 \otimes \ldots \otimes a_{m-1}])$.

Intuitively, a reverse plan is a generalization of a reverse action, as every reverse action $\texttt{revAct}(a, a')$ is a reverse plan of size one: $\texttt{revPlan}(a, [a'])$.

The previous definitions show a strong relation between an action and a sequence of actions which holds for *any* state in the set of states defined in the framework. I.e., a sequence of actions is a reverse plan of a given action, if the sequence can always be applied after the execution of $a$ and, in all the transitions defined in the set $R$, the application of this sequence always leads to the state before the execution of $a$.

However, some states may prevent the existence of a reverse plan. I.e., an action may have a reverse plan under some conditions, that do not necessarily hold at every reachable state. Thus, we need a weaker notion of reverse that takes into account the information of the states, e.g. values of some fluents obtained by sensing. By restraining the states where the reverse plan is applied, we might get reverse plans that were not applicable before. This is the idea of conditional reversal plan formalized as follows.

**Definition 31** (Conditional Reversal Plan)**.** *Let $a, a_0, \ldots, a_{m-1}$ be actions in $\mathcal{A}$. We say that $a_0 \otimes \ldots \otimes a_{m-1}$ is a $\phi$;$\psi$-reverse plan that reverses action $a$ back iff:*

$$\begin{aligned}&\forall s_1, s_2 : V(s_2, \phi) = V(s_1, \psi) = \texttt{true} \text{ if } \langle s_1, a, s_2\rangle \in R \text{ then}\\ &\quad \exists s' \text{ s.t. } s' \in \texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) \text{ and}\\ &\quad \forall s' \text{ s.t. } s' \in \texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) \Rightarrow s' = s_1\end{aligned}$$

**Example 24.** *Consider the following example adapted from [EEF08] where putting a puppy into water makes the puppy wet, and drying a puppy with a towel makes it dry. The possible states and transitions of this example are illustrated in Figure 7.1.*

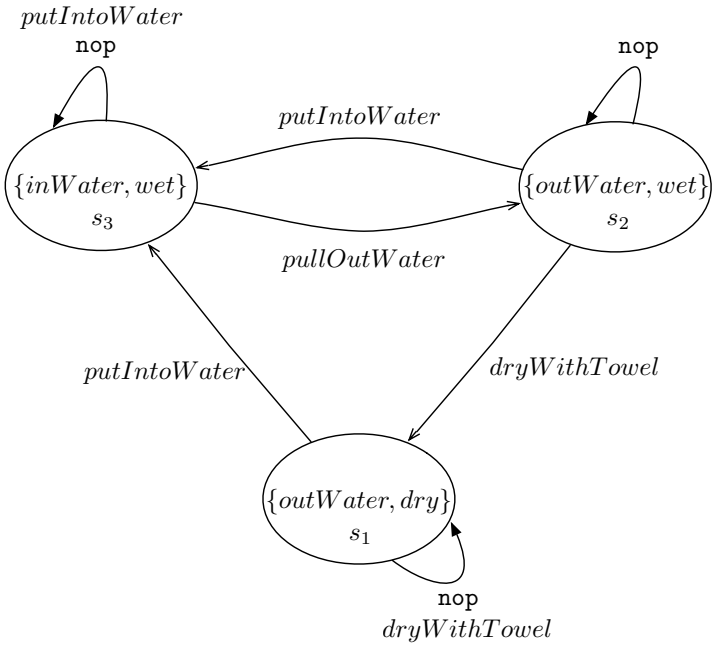



Figure 7.1: Illustration of example 24

*Based on this representation, we know that* $\texttt{revAct}(putIntoWater; pullOutWater)$*, i.e.,* $putIntoWater$ *is a* reverse action *of* $pullOutWater$*. Furthermore, while a generic reverse plan does not exist,* $[pullOutWater \otimes dryWithTowel]$ *is said to be a* $\top; \{dry\}$ *reverse plan that reverts action* $putIntoWater$*.*

## 7.2 Action Reversals for $\mathcal{ETR}$

After defining the reversals of actions for action languages, we can now elaborate on how $\mathcal{ETR}$'s external oracle can be instantiated to use these definitions and automatically infer what is the correct repair plan for each action.

However, for $\mathcal{ETR}$ we do not need such a strong and generic notion of reverse action and reverse plan as the ones proposed in [EEF08]. In fact, in [EEF08] both reverse actions and reverse plans are defined disregarding the initial state where they are being applied. On the contrary, when defining compensations or repairs of actions in $\mathcal{ETR}$, we already have information about the specific states where the repairs will be applied. This demands for a weaker notion of reverse action and reverse plan, defined for a pair of states rather than for a given action.

**Definition 32** (Situated Reverse Action)**.** *We say that an action* $a^{-1}$ *reverses* $s_2$ *into* $s_1$ *iff* $\exists s.\langle s_2, a^{-1}, s\rangle \in R$ *and* $\forall s.\langle s_2, a^{-1}, s\rangle \in R, s = s_1$*. In this case we write* $\texttt{revAct}(s_1, s_2; a^{-1})$*.*

The previous definition defines a reverse action for a pair of states. Intuitively, we say that action $a$ is a reverse action for states $s_1$ and $s_2$ iff $a$ can be executed in state $s_2$ and *all* the transitions that exist in the set of relations $R$ w.r.t. action $a$ applied to state $s_2$ end in

state $s_1$.

As in [EEF08], instead of only considering singleton actions, we also define the notion of situated reverse plan to specify sequences of actions that are able to reverse the effects of one action. Then, $\texttt{revPlan}(s_1, s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$ states that the sequence of actions $a_0 \otimes \ldots \otimes a_{m-1}$ always restores $s_1$ when executed in state $s_2$. For that, the KB may pass through $m$ arbitrary states necessarily ending in $s_1$.

**Definition 33** (Situated Reverse Plan)**.** *We say that $a_0 \otimes \ldots \otimes a_{m-1}$ is a plan that reverses $s_2$ back to $s_1$ iff:*

$$\exists s_f \text{ s.t. } s_f \in \texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) \text{ and}$$
$$\forall s_f, \text{ if } s_f \in \texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) \text{ then } s_f = s_1$$

*In this case we write* $\texttt{revPlan}(s_1, s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$

Clearly, several reverse plans may exist restoring $s_1$ from state $s_2$. Moreover, there are better reverse plans than others. E.g., imagine that in a state $s_i$ there is an action $a_i$ that always leads us to the same state $s_i$, i.e., $\langle s_i, a_i, s_i \rangle \in R$. If a plan exists to restore the system back from $s_2$ to $s_1$ passing into state $s_i$, then there are several plans where the only difference is the amount of times we execute the "dummy" action $a_i$. For instance, in example 24 it is easy to see that the action `nop` can be incorporated as many times as desired in any reverse plan.

Since recovery is a sensitive operation, in order to minimize the amount of operations to be executed, we define the notion of shorter reverse plans. A shorter reverse plan $\texttt{revPlan}_s(s_1, s_2; [a_1 \otimes \ldots \otimes a_m])$ is a reverse plan where the number of actions to be executed is minimal (i.e. there is no other $\texttt{revPlan}(s_1, s_2; [a_1 \otimes \ldots \otimes a_n])$ with $n < m$).

**Example 25.** *Recall example 24 and the states and transitions defined in Figure 7.1. Here we can conclude that* $\texttt{revPlan}_s(s_1, s_3; [pullOutWater \otimes dryWithTowel])$ *holds, and also that* $\texttt{revAct}(s_3, s_2; putIntoWater)$.

Note that alternative minimality criteria could be defined, such as where only certain actions are minimized (e.g. `nop` actions), or by extending the action language framework with actions associated with weights or costs and, in the latter case, define minimality w.r.t. the minimal total cost that could be achieved. However, elaborating on these other criteria is outside the scope of this work.

### 7.2.1 Goal Reverse Plans

The previous notions define a reverse action or a reverse plan for a pair of states $s_1$ and $s_2$, reverting the system from state $s_2$ back to state $s_1$, and imposing that the final state obtained is *exactly* $s_1$. However, it may happen that, for some pair of states, a reverse plan does not exist. Furthermore, if some conditions are provided (e.g. by the programmer) about the state that we intend to reach, then we might still achieve a state where these conditions hold. This can be useful for instance, in cases where the agent has to find

repairs to deal with norm violations. In such cases, it may not be possible to return to the exact state before the violation, but it may be possible to reach a consistent state where the agent complies with all the norms.

This corresponds to the notion of goal reverse plans that we introduce here. Based on a state formula $\phi$ characterizing the state that we want to reach, then $\texttt{goalRev}(\phi, s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$ says that the sequence $a_0 \otimes \ldots \otimes a_{m-1}$ reverses the system from $s_2$ into a consistent state $s$ where the state formula $\phi$ holds. This is formalized as follows.

**Definition 34** (Goal Reverse Plan)**.** *We say that $a_0 \otimes \ldots \otimes a_{m-1}$ is a goal plan that reverses $s_2$ to a state where $\phi$ holds iff*

$$\exists s' \text{ s.t. } s' \in \texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) \text{ and}$$
$$\forall s', \text{if } s' \in \texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}]) \text{ then } V(\phi, s') = \texttt{true}$$

*In this case we write* $\texttt{goalRev}(\phi, s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$.

As before, to preserve efficiency of plans, we define the notion of shorter goal reverse plan. $\texttt{goalPlan}_s(\phi, s_2; [a_1 \otimes \ldots \otimes a_m])$ holds, if the sequence $a_1 \otimes \ldots \otimes a_m$ is a sequence with minimal length that takes $s_2$ into a state where $\phi$ is true.

Note that the previous definition can also be seen as a formulation of a classical planning problem. A solution to a classical planning problem consists in finding a sequence of actions $\alpha_1, \ldots, \alpha_n$ that when executed in a given initial state $S_0$, results in a final state $S_f$ in which the intended given goal $G$ holds. Similarly, Definition 34 finds the sequence of actions $a_0 \otimes \ldots \otimes a_{m-1}$ that when executed from state $S_2$ always achieves a state on which formula (or goal) $\phi$ is satisfied.

## 7.3 $\mathcal{ETR}$ with Automatic Compensations

We can now make precise how and when repairs are calculated in $\mathcal{ETR}$'s semantics, and what are the changes needed on $\mathcal{ETR}$'s language to with these automatic repairs.

In addition to automatically inferred repairs, we want to give the programmer the option of explicitly defining compensations for external actions. These are useful in scenarios where the agent knows exactly how to repair an action, even when this knowledge is not directly present in the external oracle specification. This is e.g. the case of the compensations illustrated in example 11, where the information about which treatments to choose for the patient are part of the agent's beliefs and knowledge rather than part of the external world's information.

As such, the language of $\mathcal{ETR}$ is augmented so that external actions can appear in a program in three different ways: 1) without any kind of compensation associated, i.e. $\mathbf{ext}(a, \texttt{nop})$, and in this case we write $\mathbf{ext}(a)$ or simply $a$, where $a \in \mathcal{L}_a$; 2) with a user defined repair plan, written $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ where $a, b_i \in \mathcal{L}_a$; 3) with an automatic repair plan, denoted $\mathbf{extA}(a[\phi])$, where $a \in \mathcal{L}_a$, $\phi$ is an external state formula, and an external state formula is a conjunction of external fluents. This last item corresponds to

the main change in $\mathcal{ETR}$'s syntax to accommodate automatic compensations. This can be formalized as follows:

**Definition 35.** *An $\mathcal{ETR}$* atom *is either a proposition in $\mathcal{L}_P$, $\mathcal{L}_i$ or an external atom. An external atom is either a proposition in $\mathcal{L}_a$ (where $\mathcal{L}_a = \mathcal{F} \cup \mathcal{A}$), $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ or $\mathbf{extA}(a[\phi])$ where $a, b_i \in \mathcal{L}_a$ and $\phi$ is an external state formula. An $\mathcal{ETR}$ literal is either $\phi$ or $\neg\phi$ where $\phi$ is an $\mathcal{ETR}$ atom. An external state formula is a either a literal from $\mathcal{F}$ or an expression $\phi \wedge \psi$ where $\phi$ and $\psi$ are external state formulas. An $\mathcal{ETR}$* formula *is either a literal, or an expression, defined inductively, of the form $\phi \wedge \psi$, $\phi \vee \psi$ or $\phi \otimes \psi$, where $\phi$ and $\psi$ are $\mathcal{ETR}$ formulas. An $\mathcal{ETR}$* program is a set of rules of the form $\phi \leftarrow \psi$ where $\phi$ is a proposition in $\mathcal{L}_P$ and $\psi$ is an $\mathcal{ETR}$ formula.

Intuitively, $\mathbf{extA}(a[\phi])$ stands for *"execute the external action $a$, and if something fails automatically repair the action's effects either leading to the state just before $a$ was executed, or to a state where $\phi$ holds"*. When one wants the repair to restore the system to the very state just before $a$ was executed, one may simply write $\mathbf{extA}(a)$, which is equivalent to $\mathbf{extA}(a[\bot])$.

**Example 26.** *With this modified language one can write, e.g. for the situation described in Example 23, rules like the ones below. Intuitively, these rules say that: to place a product one should decrease the stock and then place the product; one can place a product in a better shelf, or in a normal shelf in case the product is not premium. Moreover, moving a product to a given shelf is an external action that can be automatically repaired based on existing information about the external world. Consequently, $\mathbf{extA}(move(X, w, betterShelf))$ means that, if something fails after the agent has moved $X$ from the warehouse into a better shelf, then a repair plan is automatically defined for this action by the semantics and the oracle:*

$$
\begin{aligned}
placeProduct(X) \leftarrow &\quad decreaseStock(X) \otimes placeOne(X) \\
decreaseStock(X) \leftarrow &\quad stock(X, S) \otimes S > 0 \otimes stock(X, S).del \otimes stock(X, S-1).ins \\
placeOne(X) \leftarrow &\quad \mathbf{extA}(move(X, w, betterShelf)) \\
placeOne(X) \leftarrow &\quad \neg premium(X) \otimes \mathbf{extA}(move(X, w, normalShelf))
\end{aligned}
$$

*Note that, the semantics must ensure that the external world is always left consistent by the agent in any possible execution. Particularly, whenever it is not possible to place a nonpremium product in the better shelf, a repair plan is executed to put the product back in the warehouse where the agent can try to put the product in the normal shelf; if it is not possible to put the product in either shelf (or to put a premium product in the better shelf), then a repair plan is executed to put the product back in the warehouse, and the stock is rolled back to its previous value (and the transaction fails).*

*A simplified version of the external environment (oracle) can be described by the following C program which includes the definition of blocks-world-like actions (where, as usual, we use* **inertial** *$F$ to stand for* **caused** *$F$* **if** *$F$* **after** *$F$, and $\alpha$* **causes** *$F$* **if** *$G$ for $F$* **if** *$\top$* **after** *$\alpha \wedge G$):*

| | | | |
|---|---|---|---|
| **caused** | $clearBS$ | **if** $\neg on(X, betterShelf)$ | |
| **caused** | $\neg clearBS$ | **if** $on(X, betterShelf)$ | |
| **caused** | $\bot$ | **if** $on(X,Y) \wedge on(X,Z) \wedge Y \neq Z$ | |
| $move(X, w, betterShelf)$ | | **causes** $on(X, betterShelf)$ | **if** $on(X, w)$ |
| $move(X, w, normalShelf)$ | | **causes** $on(X, normalShelf)$ | **if** $on(X, w)$ |
| $move(X, betterShelf, w)$ | | **causes** $on(X, w)$ | **if** $on(X, betterShelf)$ |
| $move(X, normalShelf, w)$ | | **causes** $on(X, w)$ | **if** $on(X, normalShelf)$ |
| $move(X, w, betterShelf)$ | | **causes** $\bot$ | **if** $\neg clearBS$ |
| **inertial** $on(X,Y)$ | | | |

*where we assume that a block can only be moved into a normal or better shelf when moved from the warehouse $w$; and that a block can only be moved to the better shelf if that shelf is clear (i.e., if $\neg clearBS$ holds).*

*To illustrate the behavior of this simple example of rules and external environment, let us show the execution of $P, S- \models placeProduct(b) \otimes placeProduct(a)$, where $S$ is an initial state described as:*

$$S = \langle (premium(a), stock(a,1), stock(b,1)), (on(a,w), on(b,w), clearBS) \rangle$$

*Clearly, in this very simple example, the two paths satisfying the actions $placeProduct(b) \otimes placeProduct(a)$ are the ones presented below. The first one defines the execution where $b$ is inserted directly in a normal shelf, and no failure occurs (with some obvious abbreviations):*

$$\begin{array}{l} P, \langle ((prem(a), stock(a,1), stock(b,1)), (on(a,w), on(b,w), clearBS)) \xrightarrow{stock(b,1).del} \\ ((prem(a), stock(a,1)), (on(a,w), on(b,w), clearBS)) \xrightarrow{stock(b,0).ins} \\ ((prem(a), stock(a,1), stock(b,0)), (on(a,w), on(b,w), clearBS)) \xrightarrow{\mathbf{ext}(mv(b,w,nS), mv(b,nS,w))} \\ ((prem(a), stock(a,1), stock(b,0)), (on(a,w), on(b,nS), clearBS)) \xrightarrow{stock(a,1).del} \\ ((prem(a), stock(b,0)), (on(a,w), on(b,nS), clearBS)) \xrightarrow{stock(a,0).ins} \\ ((prem(a), stock(a,0), stock(b,0)), (on(a,w), on(b,nS), clearBS)) \xrightarrow{\mathbf{ext}(mv(a,w,bS), mv(a,bS,w))} \\ ((prem(a), stock(a,0), stock(b,0)), (on(a,bS), on(b,nS), \neg clearBS) \rangle \\ \qquad \models placeProduct(b) \otimes placeProduct(a) \end{array}$$

*However, as a valid alternative execution, the agent could have first tried to insert $b$ in a better shelf, discovered that $a$ could no longer be inserted in the better shelf because it is not clear, fail the action, remove $b$ from the better shelf back to the warehouse as a compensation, insert $b$ in the normal shelf, and finally insert $a$ in the better shelf as intended. This execution corresponds to the other path satisfying the executional entailment above:*

$$
\begin{array}{l}
P, \langle (prem(a), stock(a,1), stock(b,1)), (on(a,w), on(b,w), clearBS)) \xrightarrow{\mathbf{ext}(mv(b,w,bS), mv(b,bS,w))} \\
(prem(a), stock(a,1), stock(b,1)), (on(a,w), on(b,bS), \neg clearBS)) \xrightarrow{mv(b,bS,w)} \\
(prem(a), stock(a,1), stock(b,1)), (on(a,w), on(b,w), \neg clearBS) \xrightarrow{stock(b,1).del} \\
((prem(a), stock(a,1)), (on(a,w), on(b,w), clearBS)) \xrightarrow{stock(b,0).ins} \\
((prem(a), stock(a,1), stock(b,0)), (on(a,w), on(b,w), clearBS)) \xrightarrow{\mathbf{ext}(mv(b,w,nS), mv(b,nS,w))} \\
((prem(a), stock(a,1), stock(b,0)), (on(a,w), on(b,nS), clearBS)) \xrightarrow{stock(a,1).del} \\
((prem(a), stock(b,0)), (on(a,w), on(b,nS), clearBS)) \xrightarrow{stock(a,0).ins} \\
((prem(a), stock(a,0), stock(b,0)), (on(a,w), on(b,nS), clearBS)) \xrightarrow{\mathbf{ext}(mv(a,w,bS), mv(a,bS,w))} \\
((prem(a), stock(a,0), stock(b,0)), (on(a,bS), on(b,nS), \neg clearBS)\rangle \\
\qquad \models placeProduct(b) \otimes placeProduct(a)
\end{array}
$$

*Recall from Definition 15, that the internal changes before the failure are not reflected in the final path of execution, as they are rolled back. Thus, only the external failures (in our case, the ones corresponding to moving b from the warehouse into a better shelf, and then back to the warehouse) appear in this path where a transaction succeeds.*

*Nota also that in this very simple example the reverse of moving an object from the warehouse to a shelf, is always to move it back in the warehouse, e.g:*

$$\texttt{revPlan}((\texttt{on(a,w)}, \texttt{on(b,w)}, \texttt{clearBS}), (\texttt{on(a,w)}, \texttt{on(b,bS)}, \neg\texttt{clearBS})), [\texttt{move(b,bS,w)}])$$

Importantly, and contrary to the semantics of the original $\mathcal{ETR}$ which is independent of the defined oracles, the semantics of this new language with actions of the form $\mathbf{extA}(a[\phi])$, can only be defined given specific oracles that allow the inference of repair plans. For example, for external environments described by action languages, an external state is a pair, with the action program $E$ describing the external domain and a state of the transition system, and the external oracle $\mathcal{O}^e$ is as follows (where $\mathcal{T}(E) = \langle S, V, R\rangle$):

**Definition 36** (Action Language Oracle)**.** *Let $f, a$ be atoms in $\mathcal{L}_a$ s.t. $f$ is a fluent in $\mathcal{F}$ and $a$ is an action in $\mathcal{A}$.*

1. $\mathcal{O}^e((E,s_1),(E,s_1)) \models f$ *iff* $V(f,s_1) = \texttt{true}$
2. $\mathcal{O}^e((E,s_1),(E,s_2)) \models a$ *iff* $\langle s_1, a, s_2\rangle \in R$
3. $\mathcal{O}^e((E,s_1),(E,s_2)) \models \mathbf{ext}(a, b_1 \otimes \ldots \otimes b_n)$ *iff* $\langle s_1, a, s_2\rangle \in R$
4. $\mathcal{O}^e((E,s_1),(E,s_2)) \models \mathbf{ext}(a[\phi], a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1})$ *iff one holds:*
   - *(a)* $\langle s_1, a, s_2\rangle \in R \wedge \texttt{revPlan}_s(s_1, s_2; [a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}])$*; or else*
   - *(b)* $\langle s_1, a, s_2\rangle \in R \wedge (\neg\exists a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}$ *s.t.* $\texttt{revPlan}_s(s_1, s_2; [a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}])) \wedge \texttt{goalRev}_s(\phi, s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$

Items 3 and 4 above define how the oracle satisfies external actions with compensations. Item 3 stands for the explicitly defined compensation case. In here, if one wants to explicitly define $b_1 \otimes \ldots \otimes b_n$ as the reverse plan for action $a$, then $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_n)$ is

evaluated solely by what the oracle knows about $a$, holding in a transition iff $a$ holds in that transition of states.

Moreover, when one wants to infer a repair plan for $a$ automatically, then these repairs are computed based on the notions of reverse plan and goal reverse defined previously. Namely, the formula $\mathbf{ext}(a[\phi], a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1})$ holds, iff $a$ holds in the transition $s_1$ into $s_2$, and $a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}$ is a shorter reverse plan to repair $s_2$ back to $s_1$, or if $a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}$ is a shorter goal plan to repair $s_2$ into a state where the state formula $\phi$ holds.

Note that the order of items 4a and 4b is not arbitrary. Goal reverse plans provide an elegant solution to relax the necessary conditions to obtain repairs plans and are especially useful in scenarios where it is not possible to return to the initial state before executing the external action, as e.g. in norms or contracts violations. However, care must be taken when defining the external state formula $\phi$ of an external action $\mathbf{extA}(a[\phi])$. In fact, if $\phi$ provides a very incomplete description of the state that we want to achieve, then we might achieve a state substantially different from the intended one. Particularly, although we constrain the applicability of goal reverse plans to the ones that are shorter, there is no guarantee that the changes of these plans are minimal (w.r.t. the amount of fluents that are different from the previous state). To guarantee such a property represents a belief revision problem and is, at this moment, out of scope of this work.

Finally, compensations can be instantiated by changing the definition of interpretation (definition 11) which now determines how to deal with automatic repairs.

**Definition 37** (Interpretations)**.** *An interpretation is a mapping $M$ assigning a classical Herbrand structure (or $\top$) to every path. This mapping is subject to the following restrictions, for all states $D_i$,$E_j$ and every formula $\varphi$, every external atom $a$ and every state formula $\psi$:*

1. $\varphi \in M(\langle (D, E) \rangle)$ *iff* $\mathcal{O}^d(D) \models \varphi$ *for any external state* $E$
2. $\varphi \in M(\langle (D_1, E) \overset{\varphi}{\rightarrow} (D_2, E) \rangle)$ *iff* $\mathcal{O}^t(D_1, D_2) \models \varphi$ *for any external state* $E$
3. $\varphi \in M(\langle (D, E_1) \overset{\varphi}{\rightarrow} (D, E_2) \rangle)$ *iff* $\mathcal{O}^e(E_1, E_2) \models \varphi$ *for any internal state* $D$
4. $\mathbf{extA}(a[\psi]) \in M(\langle (D, E_1) \overset{\mathbf{ext}(a[\psi], a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1})}{\rightarrow} (D, E_2) \rangle)$ *iff* $\mathcal{O}^e(E_1, E_2) \models \mathbf{ext}(a[\psi], a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1})$ *for any internal state* $D$

Note that an external action with automatic repair plans only appears in the program in the form $\mathbf{extA}(a[\phi])$. With this previous definition, it is the interpretation's responsibility to ask the oracle to instantiate it with the correct repair plan $a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}$.

### 7.3.1 Properties of Repair Plans

In this chapter we presented a series of definitions based on the notions of reverse action, reverse plan and conditional reversals of [EEF08]. In the following we make precise the relation between the concepts presented here, and the definitions from [EEF08]. Specifically, we state that if a goal reverse plan is not considered, then $a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}$ is a

valid repair plan iff it is a $\phi;\psi$-conditional plan in [EEF08] where the $\psi$ (respectively $\phi$) represents the state formula of the initial (resp. final) state $s_1$ (resp. $s_2$).

**Theorem 5** (Relation to [EEF08])**.** *Let $F_1$ and $F_2$ be formulas that represent completely the states $s_1$ and $s_2$, respectively. Then, $\mathcal{O}^e((E,s_1),(E,s_2)) \models \mathbf{ext}(a[\bot], a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1})$ iff $a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}$ is a $F_2;F_1$-reverse plan for $a$*

*Proof.* $\mathcal{O}^e((A_p,s_1),(A_p,s_2)) \models \mathbf{ext}(a[\bot], a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1})$ iff item 4a of Definition 36 holds, i.e. if $\langle s_1, a, s_2\rangle \in R \wedge \texttt{revPlan}_s(s_1, s_2; [a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}])$. Thus this means that $\exists s_f$ s.t. $s_f \in \texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$ and $\forall s_f$ s.t. $s_f \in \texttt{traj}(s_2; [a_0 \otimes \ldots \otimes a_{m-1}])$ then $s_f = s_1$. Since the latter holds, and $F_1$ and $F_2$ represent completely the states $s_1$ and $s_2$, then by [EEF08] $a_0^{-1} \otimes \ldots \otimes a_{m-1}^{-1}$ is said to be a $F_2;F_1$-reverse plan for $a$ by Definition 31. □

We can apply the result on the sufficient condition for the existence of repairs plans from [EEF08] which is based on the notion of involutory actions. An action is said to be involutory if executing the action twice from any state where the action is executable, always results in the starting state, i.e. iff for every $s_1, s_2$ s.t. $\texttt{traj}(s_1; [a \otimes a]) = s_2$ then $s_2 = s_1$. An example of an involutory action is a toggle action, as toggling a simple switch twice will always lead the system into the initial state.

**Lemma 1.** *Let $a$ be an involutory action. For every pair of states $s_1, s_2$ s.t. $\langle s_1, a, s_2\rangle \in R$ it holds that $\mathcal{O}^e((E,s_1),(E,s_2)) \models \mathbf{ext}(a[\phi], a)$ for every state formula $\phi$.*

*Proof.* Since $a$ is an involutory action, it holds that $\forall s_1, s_2$, if $s_2 \in \texttt{traj}(s_1; [a])$ then $s_1 \in \texttt{traj}(s_2; [a])$. Thus $\texttt{revPlan}(s_1, s_2; [a])$ and by case 4a of Definition 36 we know that $\mathcal{O}^e((A_p,s_1),(A_p,s_2)) \models \mathbf{ext}(a[\phi], a)$. □

Furthermore, we can talk about safety of repairs w.r.t. programs. We say that a program is *repair safe* iff all its external actions have a repair that is guaranteed to succeed.

**Theorem 6** (Repair Safety)**.** *Let $P$ be a $\mathcal{ETR}$ program without user defined repair plans of the form $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$. If for every $\mathbf{extA}(a[\phi])$ defined in $P$ there exists a reverse plan $a_1 \otimes \ldots \otimes a_k$ s.t. $\texttt{revPlan}(a, [a_1 \otimes \ldots \otimes a_k])$ then $P$ is a repair safe program.*

*Proof.* If $P$ does not have user defined repair plans, then it means that all repairs are computed using item 4 of definition 36, and thus this is catered by the `revPlan` which require that the plan can applicable (i.e. the existence of the trajectory) but also that it necessarily reaches the intended state. Since all the possible compensations are defined using `revPlan` definition, we never reach a state where the previous computed compensated is not applicable, and thus the program is safe. □

Note that, although the conditions for a repair safe program are considerably strong, they allow us to reason about the safeness of a program *before* execution. Obviously, we do not want to restrict only to repair safe programs. However, if an agent is defined by

a repair safe program, we know that, whatever happens, the agent will always leave the external world in a consistent state.

We can also define a safe property regarding a particular execution of a transaction.

**Theorem 7** (Repair Safe Execution)**.** *Let $P$ be a program without user defined repairs, $\phi$ be a formula, $\pi$ be a path, $M$ an interpretation where $M \models P$, and $\mathcal{O}^e$ an external oracle without Item 4b.*
*If $M, \pi \models_p \phi$, $M, \pi \not\models_c \phi$ and $\texttt{Seq}(\pi) \neq \emptyset$ then $\exists \pi_0, \pi_r$ where $\pi_0$ is a rollback path of $\pi$, and $\pi_r$ is a recovery path of $\pi_0$ s.t. $\pi' = \pi_0 \circ \pi_r$ and $M, \pi' \rightsquigarrow \phi$*

*Proof.* This follows from the arguments that prove Theorem 6. If no user defined repairs exist, then all repairs are computed using item 4a of definition 36. Thus, definitions of `revPlan` ensure us that all repairs can be applicable and achieve the intended state. Since every execution of a repair is always applicable, then a recovery path exists and so does $M, \pi' \rightsquigarrow \phi$. □

This result talks about the existence of compensating paths for a given transaction $\phi$ being executed on a path $\pi$. Intuitively, if $P$ does not contain user defined transactions, the oracle only computes reverse plans, and if $\pi$ is an execution of $\phi$ that fails (i.e. $M, \pi \models_p \phi$ but $M, \pi \not\models_c \phi$) after executing some external actions (i.e. if $\texttt{Seq}(\pi) \neq \emptyset$), then there always exists a path where the execution of $\phi$ is repaired, i.e. there exists a path $\pi'$ where $M, \pi' \rightsquigarrow \phi$ holds.

Note that these theorems only provide guarantees for programs where explicit user defined repairs and goal reverses are not present. The problem with user defined repairs is that it is impossible to predict, before execution, what will be the resulting state of the external world after their execution, or to guarantee any properties about this resulting state. As such, it may be the case that the existence of user defined repairs jeopardizes the applicability of automatic repair plans. This is as expected: since the user may arbitrarily change the repair of some actions, it may certainly be the case that the specification of the external domain cannot infer any repair plan for other actions in the same sequence. To prevent this, we could preclude the possibility to define user defined repairs. However, this would make $\mathcal{ETR}$ less expressive, making it impossible to use whenever the agent does not possess enough information about the external world.

Similarly to user defined repairs, the success of goal reverses depends on a state formula $\phi$ which is specified by the user. As such, in general, without restricting the set of state formulas that can be written, nothing guarantees that in the achieved state the previous reverse plans can be applicable. However, as previously stated, reasoning about state formulas is a hard problem and is out of scope of this work.

# 8

# Discussion

In this part we have proposed External Transaction Logic ($\mathcal{ETR}$), a logic to reason and execute transactions that involve interaction with both an internal and an external domain. Building upon Transaction Logic, $\mathcal{ETR}$ is a full-fledged logic theory. Using $\mathcal{ETR}$'s model theory, logical and executional entailment one can reason about the various aspects of transactions. With it, one can either talk about the general properties of transactions that hold for every path where the transaction can successfully execute, but also about the properties of a specific executional path. In addition, and to be useful in practice, $\mathcal{ETR}$ also offers a sound and complete procedure that allows one to *execute* transactions w.r.t. the semantics.

While other works exist to reason about (trans)actions with internal and external actions, normally these have very different characteristics from $\mathcal{ETR}$. Particularly, $\mathcal{ETR}$'s theory is based on the notion of *execution paths* that satisfy a transaction formula. As such, satisfaction means execution, and the semantics assigns truth values to formulas over paths. With this, a formula is true on a path, if that path denotes a valid execution for that formula.

In contrast, other popular solutions like Action Languages [GL98a], the Situation Calculus [McC63], the Event Calculus [KS86], Process Logic [HKP82], PDL [FL79], and some of their variations like [Gab02; LRLLS97; Kow92; LT88; ZF01], can also provide means to reason about state change and the related phenomena of time and action. However, the goal of such logics is to describe, very expressively, the dynamics of a given domain, by reasoning about the possible actions that can be executed and their (direct and indirect) effects on the domain. Thus, they focus heavily on solving the frame problem in order to talk about the effects of actions, rather than how these actions can really be executed.

Nevertheless, it is important to understand that $\mathcal{ETR}$ is not meant as an alternative

to these action description languages, as they offer solutions for orthogonal problems. In particular, while action description languages are designed to model and reason about rich external environments, $\mathcal{ETR}$ deals with the interaction between a given external environment, and a given internal knowledge base. As such, $\mathcal{ETR}$ does not define what are the effects of actions in the external environment, as these other solutions, but encapsulates these decisions in the definition of the oracles. Then, assuming a model of an external environment (given as a parameter of the theory), $\mathcal{ETR}$ can reason about the execution of transactions involving actions in both an external and internal knowledge base. Consequently, $\mathcal{ETR}$, as the original Transaction Logic, can be used together with these solutions via oracles' instantiations, where $\mathcal{ETR}$ finds the execution path where a transaction is true, and these action description languages determine all the direct and indirect effects from executing a given action in the external knowledge base, and what should be the end state after that.

In another context, since $\mathcal{ETR}$'s proof theory enables the execution of transactions, one can also compare $\mathcal{ETR}$ to formalisms like [BHF04; VF12; BLZ03]. These latter provide tools to describe the interactions and communications between concurrent processes during long-running transactions. For that, they are based on algebraic systems for modeling concurrent communicating processes, as Milner's CCS [Mil83] or Hoare's CSP [Hoa85], among others. Clearly, one big difference between $\mathcal{ETR}$ and these calculus based solutions is that $\mathcal{ETR}$ does not support concurrency and synchronization. However, solutions like [BHF04; VF12; BLZ03] are conceptually very different from $\mathcal{ETR}$. Their focus is mostly on the correctness of conversations between processes executing over the same (internal) domain, and thus they provide a very powerful operational semantics to ensure correctness and termination of concurrent processes. On the contrary, $\mathcal{ETR}$ deals with two different (internal and external) domains, each with different properties and semantics. While it would be interesting to have concurrency and synchronization over actions executed in the internal domain, so as to allow multiple processes (or agents) to execute actions simultaneously, achieving concurrency over actions performed externally, i.e. over a domain which we do not control, is unfeasible. Nevertheless, providing these internal concurrency features to $\mathcal{ETR}$ is an obvious future work milestone (cf. section 14.4) and is in line with what has been done in Concurrent Transaction Logic [BK96].

Moreover, these calculus based solutions are mostly operational and thus fail to be used as knowledge representation formalisms, or to model (in a declarative way) the interaction between an internal knowledge base and an external environment. Consequently, their lack of model theory and knowledge of state makes it impossible to model what is true at each step of the execution or to specify constraints on the execution of actions based on this knowledge.

$\mathcal{ETR}$ stands in between these two worlds. It provides a clean model-theoretic semantics, parametric on the meaning of the particular KB on which it operates, allowing us

to talk about properties of transactions like equivalence and implication that hold independently of what execution path is chosen. But also, by providing a proof-theory that is sound and complete w.r.t. the semantics, $\mathcal{ETR}$ is able to talk about a particular execution of a transaction and what are the possible evolution paths for a given formula. Additionally, given its abstraction of states and primitives, $\mathcal{ETR}$ can be easily adapted for a wide range of situations, being especially useful in open contexts where several different semantics can be applicable, as e.g. in the Semantic Web.

Another interesting related work that tackles the issue of modeling transactions in arbitrary domains is the rule-based language ULTRA [WFF98], which is mentioned in section 2.3. ULTRA is based on minimal model semantics and is very similar to $\mathcal{TR}$ (and both have the same modeling power in their sequential version). Similarly to $\mathcal{ETR}$, ULTRA's implementation allows the definition of compensating subtransactions for every transaction committed. However, and in contrast to $\mathcal{ETR}$, this notion is not part of ULTRA's model theory which does not provide any means to soften the ACID transactional model. Thus, there is no formal correspondence between the procedure of ULTRA and its model theory, as in $\mathcal{ETR}$.

In [RK12] the authors propose an extension of Transaction Logic with Partially Defined Actions ($\mathcal{TR}^{PAD}$) to encode axioms defining the direct and indirect effects of actions, and to directly define partial descriptions of states. This allows $\mathcal{TR}^{PAD}$ to model external environments with incomplete information and reason about actions and their effects in the domain. Its proof-theory, being sound and complete with the model theory, allows $\mathcal{TR}^{PAD}$to also execute these actions. However, $\mathcal{TR}^{PAD}$ deals with a different problem than the one of $\mathcal{ETR}$. While $\mathcal{TR}^{PAD}$ 's expressive power makes it more interesting to deal with external domains of which we have partial knowledge, dealing simultaneously with an internal and external environment is out of scope of its theory. Moreover, it is also impossible to relax transaction properties in $\mathcal{TR}^{PAD}$ as in $\mathcal{ETR}$.

Statelog [LLM98], which is also mentioned in section 2.3, is a logic-programming like language with support for atomic transactions and has some interesting features, such as the ability to encode reactive rules, as well as results about termination of programs. A fundamental difference between Statelog and $\mathcal{ETR}$ is its Kripke-style semantics based on *states* rather than *paths*. As a result, to encode evolution, states are hard-wired in the syntax each predicate as $p[S](t_1, \ldots, t_n)$ or $p(S, t_1, \ldots, t_n)$, meaning that $p(t_1, \ldots, t_n)$ holds in state $S$. Furthermore, although simple transactions can still be defined using rules, nested transactions need the notion of *procedures*. In these, one needs to define explicitly when a transaction must fail and commit, making nested transactions harder to encode (when compared to $\mathcal{TR}$ or $\mathcal{ETR}$). Moreover, Statelog does not consider an interaction with an external entity as $\mathcal{ETR}$, and so it does not provide any mechanisms for relaxing transactional properties.

When compared to Transaction Logic, the major difference of $\mathcal{ETR}$ is the ability to execute external actions, and provide properties regarding this execution. Here, the critical issue when executing external actions is that we need to consider the possibility of

external actions to fail. In fact, while internal actions may also fail during their execution, normally this failure can be simply addressed by executing a rollback in the internal database. This means that we can always restore a consistent state independently of the internal actions executed, and in an executional perspective, trying an executional branch that fails, rolling back from this failure, and succeeding in an alternative branching, is just equivalent to trying the succeeding branch directly.

However, this is not the case regarding external actions, as one does not control the environment where these actions are performed. Consequently, and contrary to internal actions, one needs to specify what needs to be done to repair external consistency when a transaction fails. This involves executing compensating actions, or repair plans, that revert the effects of the previously executed external actions.

To incorporate transactions that involve the execution of external actions, $\mathcal{ETR}$ provides new satisfaction relations that allow us to precisely talk about failed paths. These relations enable us to exactly determine the path before the failure and that will further need to be compensated. If such failure occurs, then the semantics rolls back internally and compensates externally.

From its characteristics, we argue that $\mathcal{ETR}$ is useful in contexts where one needs to ensure properties regarding the execution of actions that require interaction with both an internal and an external component, like the Semantic Web or intelligent agent systems. With this in mind, we have elaborated on oracles instantiations aiming for these two application domains. Particularly we have shown how to instantiate the oracles with the semantics: Description Logics, Action Languages, Event Calculus and Situation Calculus.

## 8.1 $\mathcal{ETR}$ with automatic repairs

In chapter 7 we have explored the possibility to automatically infer what should be the right compensations (or repair plans) needed to restore external consistency, based on an Action Language external oracle instantiation. This feature is particularly useful to alleviate the burden of the programmer that otherwise needs to know before hand what should be the actions that revert the effects of a given action in a particular rule.

An important note here is that, while the definition of these repair plans was based on the work of [EEF08], we could have chosen another language for representing changes in the external environment like [SPS11; McC63] and alternatively specify the reversals for one of these languages. Nevertheless, we chose the reversals representation from [EEF08] since its generality makes it applicable to a wider family of action languages, like, e.g. the action language $\mathcal{C}$. Moreover, since this latter language has several extensions that are already used for norms and protocol representation in multi-agent systems [SC06; ASP09], by defining an external oracle using this action language $\mathcal{C}$ we provide means to employ such representations together with $\mathcal{ETR}$, extending them with the possibility to describe an agent's behavior in a transactional way. Furthermore, our version of goal reverse plans can also be seen as a contribution to the work of [EEF08]'s as it provides means to relax

the conditions for the existence of plans, increasing the possibility of achieving a state with some desirable consistent properties.

In the context of repair plans and multi-agent systems, several languages to describe an agent's behavior partitioned over an internal and external KB have been proposed in the literature.

In this context, AgentSpeak/Jason [BWH07], 2APL [Das08] and 3APL [HBHM99] are successful examples of agent programming languages that deal with environments with both internal updates and external actions. All of these languages have a way to address action failures, and define repairs of some form to be executed in case of failure.

AgentSpeak, became a popular multi-agent programming language mainly due to the development of its Jason interpreter. The language identifies the importance of handling plan failures given the intrinsic unpredictable characteristics of dynamic environments. Thus, for that end, it defines a form of contingency plan to be executed upon a plan execution failure that "cleans up" the effects of the previous executed actions, before attempting another plan.

Similarly, 2APL and 3APL programming languages rely on the notion of plans to achieve the agent's goals. To deal with the possibility of failure, they also include the notion of plan repair rules which specify how a plan can be repaired upon failure. Similarly to $\mathcal{ETR}$'s, their semantics first computes the prefix of the plan that caused the failure, matches this prefix to the head of a plan repair rule, and creates a new plan containing the repair plus the postfix of the plan that was not executed because of the failure. The notion of action failure is also very similar to $\mathcal{ETR}$, and when such a failure happens, then the execution of the whole plan is blocked in order to be compensated/repaired. Nevertheless, repairs need to be explicitly stated by the programmer, and if such a repair does not exist, then nothing is done, and the failed plan will be tried again, possibly causing some non-termination issues. In opposition, in $\mathcal{ETR}$ it is possible to automatically infer what should be the repairs of a given action, if there is enough information available about the external world.

It is worth mentioning that in 2APL it is possible to define plans that are not to be executed concurrently and, by using repair plans, some relaxed way of atomicity can still be achieved. However, as stated, this ability to recover depends heavily on the programmer being able to specify the correct repairs, even they affect only internal knowledge (i.e., the beliefs of the agent). Moreover, these languages have an operational semantics which allows one to say what should be the next step of computation if the system is in a particular state, and prove some properties about the agent's behavior, e.g. about termination of agents execution. However, they do not have a model-theoretic semantics like $\mathcal{ETR}$, allowing us to reason about what formulas (e.g. agents beliefs) are true during the execution.

Self-checking agents [Cos12; CG13; Cos13] can do this kind of reasoning by enabling one to express properties (or meta-constraints), by means of a linear temporal logic, and define what constraints should hold during the execution of the agent. Then, in case of

a property violation, the agent can try to execute a self-repair to restore an acceptable or desired state of affairs. These properties are stated along with applicability conditions, in reactive-like terms, defining when is the property applicable (the pre-condition), when should the property be abandoned (abandon-condition), and an optional repair to be tried upon property violation.

In $\mathcal{ETR}$ one can also define constraints over the execution of (trans)actions, in several different ways. One possible way to do it is as:

$$p \leftarrow a \wedge const$$

where the constraint $const$ should always be true during all steps of execution of transaction $p$ (and where $a$ can be a complex action formula). A violation of $const$ during the execution of $p$, always forces the system to roll back internally and to execute compensations externally in order to achieve a consistent state. Similarly, we can apply such constraints to be verified before or after some actions. E.g. $p \leftarrow const \otimes a$ and $p \leftarrow a \otimes const$ define the verification of constraint $const$ before and after the (possibly complex) action $a$, respectively.

Additionally, pre-conditions and abandon-conditions can also be expressed within the formula $const$, as:

$$const \leftarrow (precondition \otimes constraint) \wedge \neg abandon\_condition$$
$$const \leftarrow \neg precondition$$

which impose the constraint $const$ to be true unless the precondition is not applicable or the fail-condition holds eventually during the execution. Furthermore, note that such an imposition of constraints as well as the definition of the conditions, regarding individual (internal or external) actions, can also be done directly in the specification of the oracles. Clearly, the meta-constraints defined in [Cos12; CG13; Cos13] are higher-level and in some sense, easier to maintain, but they are also less expressible. Moreover, as it happens with the previous solutions, the agent can only attempt to make repairs if they are explicitly stated in the meta-constraint statement, and there is no guarantee that the repair specified by the programmer is correct, as in $\mathcal{ETR}$.

$\mathcal{ETR}$ is a first step to achieve a logic that ensures transactional properties of an interaction with an external entity where it executes actions. The next step is to interpret this interaction conversely. This means that, besides the ability to interact with the environment by executing external actions, we further want to allow our systems to be influenced back from the environment by extending $\mathcal{TR}$ with reactive features. This is the subject of the next part of this thesis.

# Part III

# Modeling Reactive Transactions

# 9

# Motivation: why should transactions be truly reactive

Reactivity stands for the ability to detect interesting complex changes (also denoted as events) in the environment and react automatically to them according to some pre-defined rules. This ability is a pre-requisite of many real-world applications, such as, web-services providing different services depending on external information, multi-agent systems adapting their knowledge and actions according to the happening of changes in the environment, or monitoring systems reacting to information detected by their sensors and issuing actions automatically in response to it.

In most reactive systems, e.g. in those based on Event-Condition-Action (ECA) languages [ABB11; BEP06; CLN03], the reaction triggered by the detection of a complex event may itself be a complex action, for instance formed by the sequential execution of several basic actions. In this part, we sustain that in various contexts, reactive systems are also required to execute *transactions* in response to events, where either the whole of the transaction is executed or, if anything fails meanwhile, nothing is changed in the KB. As an illustration scenario, consider the case of an airline web-service where an external event arrives stating that a partner airline is on strike for a given time period. Then, the airline must address this event by e.g. rescheduling flights with alternative partners, or refund tickets for passengers who do not accept the changes. Clearly, some transactional properties regarding these changes must be ensured: viz. it can never be the case that a passenger is simultaneously not refunded nor have an alternative flight; or that she is completely refunded and has a rescheduled flight.

As another example of this requirement, consider the scenario where the government

issues a lien order for a given citizen and a given amount of money. When a bank receives this event, it has to check if the citizen is a client and if it has enough money available, and in this case it seizes the amount required, notifying both the client and the government. Conversely, if the client does not have enough money, then the account must be frozen. Again here, it is important to ensure that all of these actions are treated as a transaction, in an all-or-nothing way. Namely, it can never be the case that the citizen is a client of the bank and no action is performed; or if the client withdraws the money in between the balance checking and the seizure execution, then the system has to make sure that the account is frozen instead.

Nevertheless, even though the possibility of executing transactions, obeying the traditional ACID properties, is of crucial importance in the majority of today's systems, and a must e.g. in database systems, most reactive languages do not deal with it. This is especially the case in ECA languages, one of the most popular paradigms to encode reactivity, and which normally either lack from the ability to ensure transaction properties, or are confined to a database context since they only detect atomic events defined as primitive insertions/deletes on the database, (as in [Zan95; LLM98]).

It should be noted that modeling the behavior of events with transactions is not at all trivial. Particularly, events in reactive systems should be handled and responded to as soon as possible. Additionally here, one needs to consider the possibility of non-termination, as responding to an event may cause changes that trigger other events that will further need to be responded to. On the other side of the spectrum, transactions need to guarantee of a series of properties that prevent external events to be handled immediately. Moreover, as defined in database triggers, the success of a transaction depends on the events that occur in that transaction, i.e., transactions can only commit when all events occurring during their execution are answered. This implies that, if a transaction fails to respond to a given event, then the execution fails, and all changes made by that transaction are rolled back. If events were inferred based on one of the changes of a failed transaction, then these event occurrences must be considered as to have never happened. Modeling this behavior represents a hard problem that, to the best of our knowledge, has not yet been successfully handled by the knowledge representation community. Existing solutions are either completely procedural, or have strong restrictions in either the expressivity of the events that the logic can detect, or the actions that encode a response for some event.

To solve this problem, in this part we introduce Transaction Logic with Events, $\mathcal{TR}^{ev}$, an extension of $\mathcal{TR}$ integrating the ability to reason and execute transactions over very general forms of KBs, with the ability to detect complex events.

For this, we start showing how the original $\mathcal{TR}$ can be used to express and reason about complex events, and in particular, how it can express most SNOOP and ETALIS event operators [AC06] (chapter 10). Then, we argue why $\mathcal{TR}$ alone is not able to deal with both the detection of complex events and the execution of transactions, and in particular, why it does not guarantee that all complex events detected during the execution

of a transaction are responded within that execution. (section 10.2).

Afterwards, for solving this problem, we define $\mathcal{TR}^{ev}$, its language and model theory, as well as its executional semantics (chapter 11), where we propose a procedure to execute $\mathcal{TR}^{ev}$ programs (section 11.5). Finally we end with some discussion (chapter 12).

# 10

# Using $\mathcal{TR}$ to encode Event Algebras and Transactions

In this chapter we elaborate on the problem of using Transaction Logic to reason about transactions that need to react to complex events. In particular, we demonstrate that Transaction Logic syntax is rich enough to model common event patterns, by showing how most complex events in SNOOP and ETALIS algebras can be translated into $\mathcal{TR}$, and how one can use $\mathcal{TR}$'s theory to determine whether a given pattern has occurred over a given path. Subsequently, we show that, while $\mathcal{TR}$ can reason separately about complex event patterns and transactions, it cannot reason about both simultaneously, as it lacks the proper tools to enforce transactions to respond to every (complex) event triggered during execution.

## 10.1 Representing and reasoning about complex events in $\mathcal{TR}$

In the following we show that while $\mathcal{TR}$ was primarily developed to reason about the execution of (trans)actions, it can also be used to reason about the occurrence of complex event patterns over an execution path.

In complex event processing algebras, events can either be atomic or complex, where atomic events arrive to the system, for instance as an event stream, and complex events are built upon simpler events using some temporal constructs, in a similar way to how we construct complex $\mathcal{TR}$ transactions based on the combination of primitive actions using $\mathcal{TR}$ logic connectives. As such, given an event stream (or event history) containing all the atomic events that are known to have happened, the goal of these algebras is to detect complex event patterns as efficiently as possible.

Interestingly, we can use $\mathcal{TR}$ for the same objective. Since $\mathcal{TR}$'s theory satisfies formulas over paths, for an event detection context, $\mathcal{TR}$ formulas can be interpreted as events, and an execution path can be seen as the *history* where atomic and complex events are detected. Using this, we say that an event $e$ is true on a path given a program $P$, if it *occurs* on that path according to the program $P$, and represent this as usually: $P, \pi \models e$.

Furthermore, to use $\mathcal{TR}$ for complex event detection, a $\mathcal{TR}$ program is seen as a set of complex event pattern rules, which can be defined as standard $\mathcal{TR}$ rules with the form:

$$complex\_event\_name \leftarrow pattern$$

where $pattern$ is a complex formula defined by combining simpler events using the standard $\mathcal{TR}$ connectives: $\wedge, \vee, \otimes, \neg$.

While these connectives may seem insufficient at first to represent rich event patterns, as we shall see, they are expressive enough to model most common patterns in the literature of event processing. Besides these four connectives, and as syntactic sugar, we also consider the formula `path` and the common event connective ; to represent sequence of events. Then, formula `path` denotes a tautology (e.g. $\phi \vee \neg\phi$) that always holds for paths of arbitrary dimension, and the connective ; stands for $\otimes\texttt{path}\otimes$. As such, $\phi; \psi$ means $\phi \otimes \texttt{path} \otimes \psi$ and denotes the sequence of $\phi$ followed by $\psi$, but possibly interleaved with other occurrences.

In this context, as an illustration of an event pattern definition, imagine we want to state a complex event $alarm$ triggered, whenever event $ev_1$ occurs after the events $ev_2$ and $ev_3$ simultaneously occur. This can be expressed in $\mathcal{TR}$ as the rule:

$$alarm \leftarrow (e_2 \wedge e_3); e_1 \tag{10.1}$$

With this formulation, in every $\mathcal{TR}$ model of this formula, whenever there is a (sub)path where both $e_2$ and $e_3$ are simultaneously true, followed by a (sub)path where $e_1$ holds, then $alarm$ is true in the *whole* path. In other words, whenever the event $e_1$ holds after the complex event $e_2 \wedge e_3$, then the event $alarm$ also holds.

Other complex event definitions are possible, and next we show how $\mathcal{TR}$ can represent and detect most event patterns definitions of some popular event algebras. With this in mind, next we provide a translation for the event algebras SNOOP [AC06] and ETALIS [AFRSSS10].

However, before showing how this can be done, we need first to translate event algebras histories into a $\mathcal{TR}$ path. In complex event algebras, an event history is a set of event occurrences associated to discrete points in time. Since $\mathcal{TR}$ does not have a native way to represent time, the first step when providing these translations, is to define a $\mathcal{TR}$ path to represent such an event history. An event history $H$ is a set of primitive (or atomic) event occurrences associated to a time point $i$, i.e., every primitive event $e$ is represented w.r.t. a time point $i$ as e.g. $e(i)$ or $e$<$i$>. Based on this notion, we can construct a path

as a sequence of state identifiers labeled with time, where time point $i$ takes place in the transition of states $\langle s_i, s_{i+1}\rangle$. Afterwards, to reason about events, we only consider the interpretations $M$ that are *compatible* with that history path. More precisely, we only consider the interpretations that respect the history of events that is known to occur. This can be formulated as:

**Definition 38** (Compatible Interpretations)**.** *Assume a history of path event occurrences $H$ defined as a set of primitive events bounded with a time $i$. We say that an interpretation $M$ is* compatible *with a path $\langle s_1, \ldots, s_n\rangle$ w.r.t. $H$ if, for every atomic event that is true in a time $i$ in $H$, $M$ makes the same event true over the path $\langle s_i, s_{i+1}\rangle$.*

This notion of compatibility is paramount to ensure that all atomic events known to occur are also made true by $\mathcal{TR}$ interpretations. In other words, this notion is needed to ensure that we only reason with the interpretations that are compatible with the history that is known to be true. Moreover, as we shall see, the precise formalization of this compatibility notion depends on the particular semantics of the event algebra and how such event history (or event stream) is defined.

Subsequently, we redefine the notion of executional entailment, based on the concept of compatible interpretations.

**Definition 39** (Executional Entailment given an History)**.** *Let $P$ be a program, $\phi$ a complex $\mathcal{TR}$ formula, and $H$ a history defined as a set of primitive events bounded with a time $i$.*

$$P, \pi \models_H \phi$$

*iff $M, \pi \models \phi$ for every model $M$ of $P$, where $M$ is an interpretation compatible with $H$*

Equipped with these notions, next we show how $\mathcal{TR}$ can capture most complex event patterns in SNOOP and ETALIS.

### 10.1.1 Translating SNOOP events into $\mathcal{TR}$ expressions

SNOOP [AC06; CM94] is an event specification language, designed for the context of active databases, with a formal semantics to evaluate complex events over a history of event occurrences. While SNOOP's original version [CM94] defines all events as instantaneous occurrences, SNOOP's current version [AC06] considers events to occur over a time interval rather than a single time point. The authors also define a complex event detection implementation for SNOOP's algebra [Cha97] based on an Object-Oriented DBMS.

The SNOOP syntax provides the operators: $\triangle$ (and), $\triangledown$ (or), ; (sequence), and $\neg$ (negation). Then, $E_1 \triangle E_2$ occurs when both $E_1$ and $E_2$ occur, regardless of their order of occurrence; $E_1 \triangledown E_2$ occurs when $E_1$ or $E_2$ occur; and $E_1; E_2$ occurs when $E_2$ occurs after the occurrence of $E_1$. In addition, negation determines the non-occurrence of an event in the context determined by two other events, and is represented as $\neg(E_3)[E_1; E_2]$. The latter

expression occurs if $E_3$ does not occur in the closed interval formed by the end time of $E_1$ and the start time of $E_2$.

Besides the previous operators, the SNOOP algebra also defines Aperiodic $(A, A^*)$ and Periodic $(P, P^*)$ operators, and the operator $PLUS$. In this context, $A(E_1, E_2, E_3)$ denotes the event signalled each time $E_2$ occurs in the interval defined by the occurrences of $E_1$ and $E_3$. In other words, the event is triggered if there is no occurrence of $E_3$ within the interval defined by the ending of $E_1$ and the ending of $E_2$. Similarly, $A^*$ defines the cumulative version of this operator, where the complex event is only fired once when $E_3$ occurs, accumulating the occurrences of $E_2$.

Moreover, $P(E_1, t, E_2)$ occurs whenever a time period $t$ passes within the interval specified by $E_1$ followed by $E_2$, where $t$ is a time string. A cumulative version of this event also exists, where all the occurrences are accumulated, and the event is triggered when $E_2$ occurs ($P^*(E_1, t, E_2)$). Finally, the $PLUS$ event operator defines a relative temporal event started by the occurrence of an event and is signalled after a specified time period. In it, $PLUS(E_1, t)$ occurs when time $t$ passed after the occurrence of $E_1$.

Subsequently, the SNOOP history $H$ is a set of primitive events associated with time $t$ of the form $e^i_j[t]$, where $t$ is the time where the event $e_j$ occurs, and $i$ denotes the numbered occurrence of event $e_j$ in history $H$. For instance, $e^3_i[5]$ means that the third occurrence of event $e_i$ happened at time point 5. Afterwards, SNOOP semantics defines $E[H]$ as the set of time intervals over which event $E$ occurs. If $[t_i, t_f] \in E[H]$, then event $E$ is said to occur in the interval $[t_i, t_f]$.

Based on these notions, in theorem 8 we prove that, given a history of past event occurrences, if an event expression is true in SNOOP, then there is a translation into a $\mathcal{TR}$ formula, where such formula is true over that same history. However, we should stress that, since $\mathcal{TR}$ does not support a native representation of time, we do not translate the events requiring an explicit time reference, like $PLUS(E_1, t)$ or $P(E_1, t, E_3)$.

**Definition 40** (Translating SNOOP Algebra into $\mathcal{TR}$)**.** *Let $E$ be a* SNOOP *algebra expression without periodic and $PLUS$ operators, $H$ be a history containing the set of all SNOOP primitive events $e^i_j[t_1]$ that have occurred over the time interval $t_1, t_{max}$, and $\langle s_1, \ldots s_{max+1}\rangle$ be a path with size $t_{max} - t_1 + 1$. We define $\tau$ as the translation from SNOOP into $\mathcal{TR}$ by the following function:*

$$
\begin{array}{lrl}
\textbf{\textit{Primitive:}} & \tau(E) & = \mathbf{o}(E) \textit{ where } E \textit{ is a primitive event} \\
\textbf{\textit{Sequence:}} & \tau(E_1; E_2) & = \tau(E_1) \otimes \texttt{path} \otimes \tau(E_2) \\
\textbf{\textit{Or:}} & \tau(E_1 \triangledown E_2) & = \tau(E_1) \vee \tau(E_2) \\
\textbf{\textit{And:}} & \tau(E_1 \triangle E_2) & = [(\tau(E_1) \otimes \texttt{path}) \wedge (\texttt{path} \otimes \tau(E_2))] \vee \\
 & & \quad [(\tau(E_2) \otimes \texttt{path}) \wedge (\texttt{path} \otimes \tau(E_1))] \\
\textbf{\textit{Not:}} & \tau(\neg(E_3)[E_1, E_2]) & = (\tau(E_1) \otimes \neg(\texttt{path} \otimes \tau(E_3) \otimes \texttt{path}) \otimes \tau(E_2)) \\
\textbf{\textit{Aperiodic:}} & \tau(A(E_1, E_2, E_3)) & = (\tau(E_1) \otimes [(\texttt{path} \otimes \tau(E_2)) \wedge \\
 & & \quad \neg(\texttt{path} \otimes \tau(E_3) \otimes \texttt{path})])
\end{array}
$$

**Theorem 8.** *Let $E$ be a* SNOOP *algebra expression without periodic and $PLUS$ operators, $H$ be a history containing the set of all SNOOP primitive events $e^i_j[t_1]$ that have occurred over the time interval $t_1, t_{max}$, and $\langle s_1, \ldots s_{max+1}\rangle$ be a path with size $t_{max} - t_1 + 1$. Let $\tau$ be the function defined in definition 40.*

$$\text{If } [t_i, t_f] \in E[H] \text{ then for all interpretations } M \text{ compatible with } H,$$
$$M, \langle s_{t_i}, \ldots, s_{t_{f+1}}\rangle \models_{\mathcal{TR}} \tau(E)$$

*where, cf. [AC06], $E[H]$ is the set of time intervals $(t_i, t_f)$ where $E$ occurs over $H$ in an unrestricted context, and where $M$ is compatible with $H$ if, for each $e^i_j[t_i] \in H$: $M, \langle s_{t_i}, s_{t_{i+1}}\rangle \models_{\mathcal{TR}} \mathbf{o}(e_j)$.*

*Proof.* See appendix B, page 233. □

Note that, in the previous theorem, we did not provide a translation for the cumulative version of the aperiodic event $A^*$. This event is used in the context of database systems to accumulate the information of events. For example, an application wanting to record how the temperature of an object changes from the beginning of the experiment to the end of that experiment, can be modeled by this operator as:

$$A^*(beginningExperiment, changeTemperature(X), endExperiment)$$

Although we cannot translate such a behavior of $A^*$ into a single expression, we can still do it by using event rules, and accumulating this information as a parameter of $A^*$:

$$\mathbf{o}(A^*(\mathbf{o}(e_1), \mathbf{o}(e_2(all)), \mathbf{o}(e_3)), \mathbf{o}(e_2(X))) \leftarrow \mathbf{o}(e_1); \mathbf{o}(e_2(X)); \mathbf{o}(e_3)$$
$$\mathbf{o}(A^*(\mathbf{o}(e_1), \mathbf{o}(e_2(all)), \mathbf{o}(e_3))) \leftarrow \mathbf{o}(A^*(\mathbf{o}(e_1), \mathbf{o}(e_2(all)), \mathbf{o}(e_3)), \mathbf{o}(e_2(X)))$$

where the event $\mathbf{o}(A^*(\mathbf{o}(e_1), \mathbf{o}(e_2(all)), \mathbf{o}(e_3)), \mathbf{o}(e_2(X)))$ is true, whenever the pattern $\mathbf{o}(e_1); \mathbf{o}(e_2(X)); \mathbf{o}(e_3)$ holds, storing the information relative to the occurrence of $\mathbf{o}(e_2(X))$ in that occurrence, which can be used later by the intended application to trigger the appropriate response. Moreover, $\mathbf{o}(A^*(\mathbf{o}(e_1), \mathbf{o}(e_2(all)), \mathbf{o}(e_3)))$ corresponds to the event $A^*(e_1, e_2, e_3)$, where $all$ should be interpreted as a skolem constant. As intended, the event $\mathbf{o}(A^*(\mathbf{o}(e_1), \mathbf{o}(e_2(all)), \mathbf{o}(e_3)))$ is triggered only once, when $e_3$ occurs after $e_1$ and at least one occurrence of $e_2$ occurs between $e_1$ and $e_3$.

To illustrate how SNOOP expressions are translated into $\mathcal{TR}$, consider the following example.

**Example 27** (SNOOP to $\mathcal{TR}$ translation)**.** *Let $H$ be a SNOOP history of the following form $H = \{e^1_2[1], e^1_1[2], e^1_3[3], e^2_2[4]\}$. From this history, we know for a $M$ compatible with $H$ that: $\mathbf{o}(e_2) \in M(\langle s_1, s_2\rangle)$, $\mathbf{o}(e_1) \in M(\langle s_2, s_3\rangle)$, $\mathbf{o}(e_3) \in M(\langle s_3, s_4\rangle)$ $\mathbf{o}(e_2) \in M(\langle s_4, s_5\rangle)$. From these, we know the following:*

- $(e_1; e_2)[H] = \{(2, 4)\}$ *and* $M, \langle s_2, s_3, s_4, s_5\rangle \models \mathbf{o}(e_1) \otimes \texttt{path} \otimes \mathbf{o}(e_2)$

- $(e_1 \triangle e_2)[H] = \{(1,2),(2,4)\}$, $M, \langle s_1, s_2, s_3\rangle \models (\mathbf{o}(e_2) \otimes \mathtt{path}) \wedge (\mathtt{path} \otimes \mathbf{o}(e_1))$ *and* $M, \langle s_2, s_3, s_4, s_5\rangle \models (\mathbf{o}(e_1) \otimes \mathtt{path}) \wedge (\mathtt{path} \otimes \mathbf{o}(e_2))$
- $(e_1 \vee e_3)[H] = \{(2,2),(3,3)\}$ *and* $M, \langle s_2, s_3\rangle \models \mathbf{o}(e_1)$ *and* $M, \langle s_3, s_4\rangle \models \mathbf{o}(e_3)$
- $\neg[e_3](e_2,(e_1;e_2))[H] = \{(1,4)\}$ *and* $M, \langle s_1, s_2, s_3, s_4, s_5\rangle \models \mathbf{o}(e_1) \otimes \neg\mathbf{o}(e_3) \otimes \mathbf{o}(e_1) \otimes \mathtt{path} \otimes \mathbf{o}(e_2)$
- $A(e_2, e_3, (e_1; e_2)) = \{(2,3)\}$ *and* $M, \langle s_1, s_2, s_3, s_4\rangle \models \mathbf{o}(e_2) \otimes [(\mathtt{path} \otimes \mathbf{o}(e_3)) \wedge \neg(\mathtt{path} \otimes \mathbf{o}(e_1) \otimes \mathtt{path} \otimes \mathbf{o}(e_2) \otimes \mathtt{path})]$

### 10.1.2 Translating ETALIS events into $\mathcal{TR}$ expressions

We repeat the previous translation exercise, now showing how $\mathcal{TR}$ can encode event patterns expressed in the ETALIS event algebra. The ETALIS event algebra has some roots in $\mathcal{TR}$, and its initial versions [AFSS09a; AFSS09b] share most of its syntax and connectives with $\mathcal{TR}$. Because of this, it is natural to try to translate ETALIS back to $\mathcal{TR}$, showing how one can write ETALIS events in it.

When compared to the SNOOP algebra, the ETALIS algebra is richer, especially due to its constructors to encode parallelism. With those, one can e.g. express, $E_1$ `PAR` $E_2$, which denotes that event $E_1$ occurs in parallel with event $E_2$, i.e., that $E_2$ starts before the ending of $E_1$ (or the converse). Similarly, $E_1$ `STARTS` $E_2$ denotes that event $E_1$ and $E_2$ starts at the exact same time, but $E_2$ ends after $E_1$. Conversely, $E_1$ `FINISHES` $E_2$ denotes that event $E_1$ and $E_2$ end at the exact same time, but $E_2$ starts before $E_1$

Besides these events defining parallel occurrence patterns, ETALIS also has the operators: `SEQ`, `AND`, `OR`, `EQUALS`, `MEETS`, `DURING`, `FINISHES`, `NOT`; and the event expressions: $E$ `WHERE` $t$ and $(P).q$. Then, $E_1$ `SEQ` $E_2$ occurs whenever $E_2$ occurs after $E_1$. The pattern $E_1$ `AND` $E_2$ occurs when both $E_1$ and $E_2$ occur irrespective of their occurrence order. The pattern $E_1$ `OR` $E_2$ occurs when either $E_1$ or $E_2$ occurs. The pattern $E_1$ `EQUALS` $E_2$ occurs when $E_1$ and $E_2$ occur at the exact same time. The pattern $E_1$ `MEETS` $E_2$ occurs when $E_2$ occurs after $E_1$, but where the ending of $E_1$ coincides with the starting of $E_2$. The pattern $E_1$ `DURING` $E_2$ occurs when $E_1$ occurs during the interval specified by the occurrence of $E_2$. Moreover, `NOT`$(E_3)[E_1, E_2])$ represents the negated pattern, and occurs if $E_3$ does not occur in the closed interval defined by the ending of $E_1$ and the starting of $E_2$. Then, $(P).q$ detects the occurrence of $P$ if it happens within an interval of length $q$, where $q$ is a number representing the maximum time window. And finally, $E$ `WHERE` $t$ occurs when the event pattern $E$ occurs at time $t$.

ETALIS history of events is represented by an event stream $\epsilon$ containing a set of all primitive events $e$ associated to time point <$t$>. Its semantics is based on a minimal model semantics, where an interpretation $\mathcal{I}$ is said to be a model of a given rule set $R$ and event stream $\epsilon$, if it satisfies every atomic occurrence in the stream $\epsilon$ and, whenever it satisfies the body of a rule in $R$, it also satisfies the head. Finally, the minimal model, is the unique model that minimizes the amount of events satisfied. In this context, $\mathcal{I}(E)$ is the set of

| pattern | $\mathcal{I}_\mu$(pattern) |
|---|---|
| $\mathtt{pr}(t_1,\ldots,t_n)$ | $\mathcal{I}(\mathtt{pr}(\mu^*(t_1),\ldots,\mu^*(t_n)))$ |
| $p$ WHERE $t$ | $\mathcal{I}_\mu(p)$ if $\mu^*(t) = true$<br>$\emptyset$ otherwise. |
| $q$ | $\{\langle q,q\rangle\}$ for all $q{\in}\mathbb{Q}^+$ |
| $(p).q$ | $\mathcal{I}_\mu(p) \cap \{\langle q_1,q_2\rangle \mid q_2 - q_1 = q\}$ |
| $p_1$ SEQ $p_2$ | $\{\langle q_1,q_4\rangle \mid \langle q_1,q_2\rangle{\in}\mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle{\in}\mathcal{I}_\mu(p_2)$ and $q_2{<}q_3\}$ |
| $p_1$ AND $p_2$ | $\{\langle \mathsf{min}(q_1,q_3),\mathsf{max}(q_2,q_4)\rangle \mid \langle q_1,q_2\rangle{\in}\mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle{\in}\mathcal{I}_\mu(p_2)\}$ |
| $p_1$ PAR $p_2$ | $\{\langle \mathsf{min}(q_1,q_3),\mathsf{max}(q_2,q_4)\rangle \mid \langle q_1,q_2\rangle{\in}\mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle{\in}\mathcal{I}_\mu(p_2)$ and $\mathsf{max}(q_1,q_3){<}\mathsf{min}(q_2,q_4)\}$ |
| $p_1$ OR $p_2$ | $\mathcal{I}_\mu(p_1) \cup \mathcal{I}_\mu(p_2)$ |
| $p_1$ EQUALS $p_2$ | $\mathcal{I}_\mu(p_1) \cap \mathcal{I}_\mu(p_2)$ |
| $p_1$ MEETS $p_2$ | $\{\langle q_1,q_3\rangle \mid \langle q_1,q_2\rangle{\in}\mathcal{I}_\mu(p_1)$ and $\langle q_2,q_3\rangle{\in}\mathcal{I}_\mu(p_2)\}$ |
| $p_1$ DURING $p_2$ | $\{\langle q_3,q_4\rangle \mid \langle q_1,q_2\rangle{\in}\mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle{\in}\mathcal{I}_\mu(p_2)$ and $q_3{<}q_1{<}q_2{<}q_4\}$ |
| $p_1$ STARTS $p_2$ | $\{\langle q_1,q_3\rangle \mid \langle q_1,q_2\rangle{\in}\mathcal{I}_\mu(p_1)$ and $\langle q_1,q_3\rangle{\in}\mathcal{I}_\mu(p_2)$ and $q_2{<}q_3\}$ |
| $p_1$ FINISHES $p_2$ | $\{\langle q_1,q_3\rangle \mid \langle q_2,q_3\rangle{\in}\mathcal{I}_\mu(p_1)$ and $\langle q_1,q_3\rangle{\in}\mathcal{I}_\mu(p_2)$ and $q_1{<}q_2\}$ |
| NOT$(p_1).[p_2,p_3]$ | $\mathcal{I}_\mu(p_2$ SEQ $p_3) \setminus \mathcal{I}_\mu(p_2$ SEQ $p_1$ SEQ $p_3)$ |

Figure 10.1: ETALIS operators semantics [AFRSSS10].

time intervals <$t_i, t_f$> where the event pattern expression $E$ occurs over $\epsilon$ w.r.t. the rule set $R$. If <$t_i, t_f$> $\in \mathcal{I}(E)$, then $E$ is said to occur over the interval <$t_i, t_f$>.

Based on these notions, fig. 10.1 (originally from [AFRSSS10]) shows the formal definition of ETALIS operators, where $\mu$ represents a variable assignment, and is defined as a mapping assigning a value to every variable, in the usual way.

To translate ETALIS parallel events, we have to be able to manipulate and talk about each part of an event expression, so as to say that two events start or end at the same time. As such, we define every event pattern expression translation to be composed by a starting, a middle and an ending expression, and manipulate each of these parts. For instance, the expression $e_1$ `SEQ` $e_2$ `SEQ` $e_3$, translated to $\mathcal{TR}$ as $\mathbf{o}(e_1);\mathbf{o}(e_2);\mathbf{o}(e_3)$ has as starting $\mathbf{o}(e_1)$, as ending $\mathbf{o}(e_3)$, and as middle $\mathbf{o}(e_2)$, while the expression $e_1$, translated to $\mathbf{o}(e_1)$, has as starting and ending $\mathbf{o}(e_1)$, and its middle is not defined.

Based on this, an ETALIS translation of some expression returns three arguments in $\mathcal{TR}$ syntax: the full event pattern expression (i.e., the full translation of the expression in $\mathcal{TR}$ syntax), its starting expression, and its ending expression, Moreover, as expected, the starting and ending expressions are always included within the full expression. This is encoded as follows.

**Definition 41** (ETALIS translation expressions)**.** *An ETALIS translation (denoted $T(E)$) is a sequence of three expression of the following form:*

$$T(E) = \langle T(E).expr, T(E).start, T(E).end\rangle$$

*where $T(E).expr$ denotes the full expression in $\mathcal{TR}$'s syntax, $T(E).start$ some initial part of the expression, and $T(E).end$ some final part of the expression.*

Afterwards, we also define how these expressions can be manipulated.

**Definition 42** (ETALIS expression manipulation)**.** *Let $E_1$ and $E_2$ be ETALIS translation expressions, and $T(E)$ an ETALIS translation. We define the subtract operation $E_1 \setminus E_2 = E_3$ where $E_3$ is the expression resulting from removing $E_2$ from $E_1$. If $E_2 = E_1$ then $E_3 = ()$, i.e., a tautology that holds on paths of size 1.*

*In addition, whenever $T(E).expr \neq T(E).start$ we also define:*

$$T(E).middle \equiv (T(E).expr \setminus T(E).start) \setminus T(E).end)$$

*and*

$$T(E).expr \equiv T(E).start\ ;\ T(E).middle\ ;\ T(E).end$$

*where if $M, \langle s_1, \ldots s_{f+1}\rangle \models_{TR} T(E).expr$ then $M, \langle s_1, s_2\rangle \models T(E).start$, $\exists i, j$ where $1 < i \leq j < f$ where $M, \langle s_i, s_{j+1}\rangle \models T(E).middle$ and $M, \langle s_f, s_{f+1}\rangle \models T(E).end$*

The previous definition 42 provides the tools to manipulate ETALIS translation expressions, allowing one to talk about the beginning of the expression (i.e., $T(E).start$), its middle ($T(E).middle$) and its ending ($T(E).end$). In this sense, $T(E).start$ denotes the beginning of the expression $T(E).expr$ that necessarily holds in the first transition, while $T(E).end$ holds in the last transition.

In addition, for expressions where $T(E).expr \neq T(E).start$, we can say something more, namely that $T(E).expr$ can be seen as a sequence of expressions of the form: $T(E).start; T(E).middle; T(E).end$, where the middle expression $T(E).middle$ can be simply inferred from $T(E).expr, T(E).start, T(E).end$. However, note that for cases where $T(E).expr = T(E).start$ this does not hold, and $T(E).middle$ is undefined.

As previously mentioned, these operations are crucial to translate event patterns based on parallelism, like $E_1$ `PAR` $E_2$. Moreover, it also allows us to express two events starting or ending simultaneously ($E_1$ `STARTS` $E_2$, and $E_1$ `FINISHES` $E_2$, respectively); or the case where one event is contained in the occurrence of the other ($E_1$ `DURING` $E_2$).

Based on the latter notions, we can formulate the ETALIS translation to $\mathcal{TR}$ as follows:

**Definition 43** (Translating ETALIS into $\mathcal{TR}$)**.** *Let $E_1, E_2$ be any* ETALIS *algebra expressions without the events patterns that explicitly reference time: $E$ **WHERE** $t$ and $(E).q$.*

*Let $\epsilon$ be a history, or event stream, containing the set of all primitive events $e$ associated to time point <t> that have occurred over the time interval $t_1, t_{max}$, and $\langle s_1, \ldots s_{max+1}\rangle$ be a path with size $t_{max} - t_1 + 1$. We define $\tau$ as the translation from ETALIS into $\mathcal{TR}$ by the following*

*function:*

***Primitive:*** $T(e) = \langle \mathbf{o}(e), \mathbf{o}(e), \mathbf{o}(e) \rangle$ *where* $E$ *is a primitive event*

***SEQ:*** $T(E_1 \texttt{ SEQ } E_2) = \langle (T(E_1).expr ; T(E_2).expr), T(E_1).start, T(E_2).end \rangle$

***OR:*** $T(E_1 \texttt{ OR } E_2) = \langle (T(E_1).expr \vee T(E_1).expr),$
$(T(E_1).start \vee T(E_2).start), (T(E_1).end \vee T(E_2).end) \rangle$

***EQUALS:*** $T(E_1 \texttt{ EQUALS } E_2) = \langle (T(E_1).expr \wedge T(E_2).expr),$
$(T(E_1).start \wedge T(E_2).start), (T(E_1).end \wedge T(E_2).end) \rangle$

***NOT:*** $T(\texttt{NOT}(E_3)[E_1, E_2]) = \langle (T(E_1).expr \otimes \neg(\texttt{path} \otimes T(E_3).expr \otimes \texttt{path}) \otimes T(E_2).expr),$
$T(E_1).start, T(E_2).end \rangle$

***AND:*** $T(E_1 \texttt{ AND } E_2) = \langle (T(E_1).expr \ ; \ T(E_2).expr) \vee (T(E_2).expr \ ; \ T(E_1).expr),$
$(T(E_1).start \vee T(E_2).start), (T(E_1).end \vee T(E_2).end) \rangle$

***MEETS:*** $T(E_1 \texttt{ MEETS } E_2) = \langle (T(E_1).expr \setminus T(E_1).end); (T(E_1).end \wedge T(E_2).start) \ ;$
$(T(E_2).expr \setminus T(E_2).start), T(E_1).start, T(E_2).end \rangle$

***DURING:*** $T(E_1 \texttt{ DURING } E_2) = \langle (T(E_2^*).start \otimes [(\texttt{path} \otimes T(E_1).expr \otimes \texttt{path}) \wedge T(E_2).middle] \otimes$
$T(E_2).end), T(E_2).start, T(E_2).end \rangle$

***STARTS:*** $T(E_1 \texttt{ STARTS } E_2) = \langle (T(E_1).start \wedge T(E_2^*).start) \ ;$
$((T(E_1).expr \setminus T(E_1).start \otimes \texttt{path}) \wedge T(E_2^*).middle) \ ; \ T(E_2).end,$
$(T(E_1).start \wedge T(E_2).start), T(E_2).end \rangle$

***FINISHES:*** $T(E_1 \texttt{ FINISHES } E_2) = \langle T(E_2).start \ ;$
$((\texttt{path} \otimes (T(E_1).expr \setminus T(E_1).end)) \wedge T(E_2).middle) \ ;$
$(T(E_2).end \wedge T(E_1).end),$
$T(E_1).start, (T(E_1).end \wedge T(E_2).end) \rangle$

***PAR:*** $T(E_1 \texttt{ PAR } E_2) = \langle [(T(E_1).start \wedge T(E_2).start) \vee (T(E_1).start; T(E_2).start) \vee$
$(T(E_2).start; T(E_1).start)];$
$[(\texttt{path} \otimes T(E_1).middle \otimes \texttt{path}) \wedge (\texttt{path} \otimes T(E_2).middle \otimes \texttt{path})]$
$[(T(E_1).end \wedge T(E_2).end) \vee (T(E_1).end; T(E_2).end) \vee$
$(T(E_2).end; T(E_1).end)],$
$(T(E_1).start \wedge T(E_2).start) \vee T(E_1).start \vee T(E_2).start,$
$(T(E_1).end \wedge T(E_2).end) \vee T(E_1).end \vee T(E_2).end \rangle$

Finally, we formalize the correspondence between the formulas proven by ETALIS and $\mathcal{TR}$ as follows.

**Theorem 9.** *Let* $E_1, E_2$ *be any* ETALIS *algebra expressions without the events patterns that explicitly reference time:* $E$ ***WHERE*** $t$ *and* $(E).q$.

*Let* $\epsilon$ *be a history, or event stream, containing the set of all primitive events* $e$ *associated to time point <t> that have occurred over the time interval* $t_1, t_{max}$*, and* $\langle s_1, \ldots s_{max+1} \rangle$ *be a path with size* $t_{max} - t_1 + 1$*. Let* $\tau$ *be the translating function as defined in definition 43.*

*If* $< t_i, t_f > \in \mathcal{I}(E)$ *then for all* $M$ compatible with $\epsilon$, $M, \langle s_{t_i}, \ldots, s_{t_{f+1}} \rangle \models_{\mathcal{TR}} T(E).expr$

*where, cf. [AFRSSS10],* $\mathcal{I}$ *is the minimal model of* $\epsilon$ *and an empty rule set* $R$*, and* $\mathcal{I}(E)$ *is the set of time intervals <*$t_i, t_f$*> where* $E$ *occurs over* $\epsilon$ *w.r.t. the rule set* $R$*; and where* $M$ *is compatible with* $\epsilon$ *if, for each <*$t_i$*>* $\in \epsilon(e_j)$*:* $M, \langle s_{t_i}, s_{t_{i+1}} \rangle \models_{\mathcal{TR}} \mathbf{o}(e_j)$.

*In addition, if we try to translate* $E.middle$*, and* $E.middle$ *is undefined (because* $E.expr = E.start$*), then the whole expression is translated to* $\perp$*, which is false in any path of any length.*

*Proof.* See appendix B page 236 □

Note that the translation of ETALIS expressions involving parallel constructs is only defined for some expressions. Namely, translation of the constructs `DURING`, `STARTS` and `FINISHES` is only defined when the second argument is a complex expression $E_2$ such that $E_2.expr \neq E_2.start$; and the translation of `PAR` is only defined when both these expressions follow this same property. In fact, if $e_1$ and $e_2$ are atomic (instantaneous) events, then $\mathcal{I}(e_1 \texttt{ PAR } e_2)$ will always return the empty set, as there will be no interval satisfying such an expression. Since to define some sort of parallel event, we need to be able to talk about the middle of the expression, expressions that do not follow this property are translated to $\bot$ (which states e.g. $\phi \wedge \neg\phi$), as they will never be satisfied by the ETALIS minimal model $\mathcal{I}$.

**Example 28** (ETALIS to $\mathcal{TR}$ translation)**.** *Consider the following history $\epsilon$ in ETALIS $\epsilon = \{e_2(1), e_1(2), e_3(3), e_2(4)\}$. From this history, we know for a $M$ compatible with $\epsilon$ that: $\mathbf{o}(e_2) \in M(\langle s_1, s_2\rangle)$, $\mathbf{o}(e_1) \in M(\langle s_2, s_3\rangle)$, $\mathbf{o}(e_3) \in M(\langle s_3, s_4\rangle)$ $\mathbf{o}(e_2) \in M(\langle s_4, s_5\rangle)$. From these, we know the following:*

- $\mathcal{I}(e_1 \texttt{ SEQ } e_2) = \{$ <2, 4> $\}$, $T(e_1 \texttt{ SEQ } e_2) = \langle(\mathbf{o}(e_1); \mathbf{o}(e_2)), \mathbf{o}(e_1), \mathbf{o}(e_2)\rangle$ *and that* $M, \langle s_2, s_3, s_4, s_5\rangle \models (\mathbf{o}(e_1); \mathbf{o}(e_2))$

- $\mathcal{I}(e_1 \texttt{ AND } e_2) = \{$ <1, 2>, <2, 4> $\}$, $T(e_1 \texttt{ AND } e_2) = \langle((\mathbf{o}(e_1); \mathbf{o}(e_2)) \vee (\mathbf{o}(e_2); \mathbf{o}(e_1))), \mathbf{o}(e_1) \vee \mathbf{o}(e_2), \mathbf{o}(e_2) \vee \mathbf{o}(e_1)\rangle$, $M, \langle s_1, s_2, s_3\rangle \models ((\mathbf{o}(e_1); \mathbf{o}(e_2)) \vee (\mathbf{o}(e_2); \mathbf{o}(e_1)))$ *and* $M, \langle s_2, s_3, s_4, s_5\rangle \models ((\mathbf{o}(e_1); \mathbf{o}(e_2)) \vee (\mathbf{o}(e_2); \mathbf{o}(e_1)))$

- $\mathcal{I}((e_1 \texttt{ SEQ } e_3) \texttt{ STARTS } (e_1 \texttt{ SEQ } e_2)) = \{$<2, 4>$\}$, $T((e_1 \texttt{ SEQ } e_3) \texttt{ STARTS } (e_1 \texttt{ SEQ } e_2)) = \langle(\mathbf{o}(e_1) \wedge \mathbf{o}(e_1)); (\mathbf{o}(e_3) \otimes \texttt{path}) \wedge (); \mathbf{o}(e_2), (\mathbf{o}(e_1) \wedge \mathbf{o}(e_1)), \mathbf{o}(e_2)\rangle$. *Notice the latter is equivalent to* $\langle \mathbf{o}(e_1); (\mathbf{o}(e_3) \otimes \texttt{path}); \mathbf{o}(e_2), \mathbf{o}(e_1), \mathbf{o}(e_2)\rangle$

  *Additionally we have* $M, \langle s_2, s_3, s_4, s_5\rangle \models \mathbf{o}(e_1); (\mathbf{o}(e_3) \otimes \texttt{path}); \mathbf{o}(e_2)$

- $\mathcal{I}(e_3 \texttt{ DURING } e_1 \texttt{ SEQ } e_2) = \{$<2, 4>$\}$, $T(e_3 \texttt{ DURING } e_1 \texttt{ SEQ } e_2) = \langle(\mathbf{o}(e_1) \otimes [(\texttt{path} \otimes \mathbf{o}(e_3) \otimes \texttt{path}) \wedge ()] \otimes \mathbf{o}(e_2)), \mathbf{o}(e_1), \mathbf{o}(e_2)\rangle$ *Notice the latter is equivalent to the expression* $\langle(\mathbf{o}(e_1); \mathbf{o}(e_3); \mathbf{o}(e_2))\mathbf{o}(e_1), \mathbf{o}(e_2)\rangle$

  *Additionally we have* $M, \langle s_2, s_3, s_4, s_5\rangle \models \mathbf{o}(e_1); \mathbf{o}(e_3); \mathbf{o}(e_2)$

In both of the previous translations we used the tag $\mathbf{o}(e)$ to denote the occurrence of atomic events. This is to be interpreted as a normal $\mathcal{TR}$ atom, and this tag usage is solely to help differentiate (atomic) formulas syntactically.

It is important to stress that, while $\mathcal{TR}$ can translate most events in ETALIS and SNOOP, it lacks the constructs to explicit refer time points. As a consequence of this, in all of the previous translations, $\mathcal{TR}$ fails to encode the event patterns that require such a reference like e.g., SNOOP's periodic event $P(E_1, t, E_3)$, which holds whenever the time string $t$ occurs in the interval formed by the end time of the event $E_1$ and the starting

time of the event $E_3$; or the ETALIS event $(E_1).3$ which detects the occurrence of $E_1$ if it happens within an interval of time length 3.

Nonetheless, $\mathcal{TR}$'s theory and constructs are still rich enough to capture almost all of the event patterns available in SNOOP and ETALIS, as previously shown. Moreover, since most event algebras share the same constructors, note that alternative CEPs could be chosen for this exercise like [HV02; MM09] achieving similar results. In this sense, while SNOOP algebra was chosen for its wide popularity and for being one of the CEP algebra pioneers, ETALIS was chosen because of its origins in $\mathcal{TR}$.

However, while $\mathcal{TR}$ is suitable to individually model transactions and (as seen here) event patterns, it still fails to model both concepts simultaneously. In the following we show why this is the case, and provide clues to what changes are needed in $\mathcal{TR}$ to achieve the combination of complex event detection and execution of reactive transactions.

## 10.2 $\mathcal{TR}$ with events and reactive transactions: the problem

The previously mentioned complex event processing solutions are designed to express complex event patterns, and to decide when such patterns are true based on the occurrence of given primitive events. Reactive languages go one step ahead, and combine this notion of complex event detection with action execution. In a reactive language, one can express what event patterns are interesting and should be detected (using some event algebra, just like in SNOOP or ETALIS), but *also*, define what should be done in the system when a given pattern is detected.

This notion of "what should be done when an event occurs" is normally defined in a reactive language by some action algebra, allowing the system to execute complex actions in response to complex events. However, while a number of expressive reactive languages exist in the literature, normally they do not allow this complex action to be formulated as a transaction.

To get a better grasp on the problem of combining reactive features with transactions, recall that reactive languages are normally encoded in the form of Event-Condition-Action (ECA) rules of the form: **on** *event* **if** *condition* **do** *action*. In such a rule, the *action* component is triggered for execution whenever the *event* is known to be true, and the system is on a state defined by the *condition*.

Afterwards, imagine we want to encode the following reactive behavior into $\mathcal{TR}$:

"on alarm do action $a_1$ followed by action $a_2$".

where $alarm$ is e.g. the complex event pattern expressed by the rule (10.1), triggering the ECA-rule, and $a_1$ followed by action $a_2$ defines a transaction to be executed whenever the $alarm$ is true [1].

Clearly, $\mathcal{TR}$ can individually express and reason about the transaction $a_1 \otimes a_2$, and also about its complex event (as shown above). So the question remaining is whether $\mathcal{TR}$

[1] for simplicity, and for the purpose of this section, we omit the translation of the *condition*-part for now.

can deal with both simultaneously, i.e., if it can detect the complex event *and* execute its associated action $a_1 \otimes a_2$ transactionally. Before answering this, let us first elaborate on the consequences of this combination.

To model a behavior combining both transactions and complex events, two important issues must be tackled in the logic:

1) how to model the triggering behavior of reactive systems, where the occurrence of an event drives the execution of a transaction in its response. Or, in other words, how to detect events and automatically trigger their corresponding response execution;

2) how to model the transaction behavior that prevents transactions to commit until all occurring events are responded.

Regarding 1), the authors of $\mathcal{TR}$ show in [BKC93] how a simple reactive behavior, where events are triggered inside a transaction, can be achieved in $\mathcal{TR}$ as:

$$\begin{aligned} p &\leftarrow body \otimes ev \\ ev &\leftarrow \mathbf{r}(ev) \end{aligned} \tag{10.2}$$

Where these rules guarantee that in all paths that make $p$ true, i.e., in all executions of transaction $p$, the event $ev$ is triggered/fired (after the execution of some arbitrary $body$), and $ev$'s response, $\mathbf{r}(ev)$, is executed. Note that both $\mathbf{r}(ev)$ and $body$ are just name tags which can be defined as arbitrary formulas.

However, the encoding of (10.2) can only address a very simple and specific type of event: atomic events that are explicitly triggered by a transaction defined in the program. In general, events can be complex or atomic, and atomic events can also arrive as external events, or become true because some primitive action is executed on a path (e.g. as the database triggers – on insert/on delete). Nonetheless, answering external events can be done in $\mathcal{TR}$ by considering the paths that make the external event true. E.g., if one wants to respond to an external event $ev$ from an initial state, then all we need to do is to find the sequences of states $\pi$ starting in that state, such that $P, \pi \models ev$, where $P$ includes the last rule of (10.2), plus the rules describing how to respond to $ev$ (i.e., defining $\mathbf{r}(ev)$ execution).

Furthermore, detecting the occurrence of primitive actions can be easily tackled by item 2 of definition 1 (i.e., by $\mathcal{TR}$'s definition of interpretation), while the detection of complex events can be done as prescribed previously in this chapter. However, with the approach of [BKC93], there is no way to drive the execution of an event response when the event occurrence becomes true. For instance, the ECA-rule stated above could be encoded as:

$$\begin{aligned} \mathbf{o}(alarm) &\leftarrow (\mathbf{o}(e_2) \wedge \mathbf{o}(e_3)); \mathbf{o}(e_1) \\ \mathbf{r}(alarm) &\leftarrow a_1 \otimes a_2 \end{aligned}$$

where the tags $\mathbf{o}(alarm)$ and $\mathbf{r}(alarm)$ denote the occurrence and response of the event $alarm$, respectively. Yet, this encoding does not drive the execution of $\mathbf{r}(alarm)$ when

$\mathbf{o}(alarm)$ holds which is paramount in any reactive system. Clearly, here one has to force that, whenever $\mathbf{o}(alarm)$ holds, $\mathbf{r}(alarm)$ must be made true subsequently (i.e., the body which has $\mathbf{r}(alarm)$ has head must be executed). Of course, adding a rule $\mathbf{r}(alarm) \leftarrow \mathbf{o}(alarm)$ would not work, as it would only state that, one alternative way to satisfy the response of alarm is to make its occurrence true. And as such, with this latter rule, it would be sufficient to satisfy $\mathbf{o}(alarm)$ to make $\mathbf{r}(alarm)$ true, which obviously is not what is intended.

Clearly, this combination implies two different types of formulas with two very different behaviors: the *detection* of events which should be tested for occurrence w.r.t. a past history; and the *execution* of transactions as a response, which intends to construct paths where formulas can succeed respecting transactional properties.

Because of this, to achieve the intended behavior, $\mathcal{TR}$'s formulas need to be partitioned into event occurrence formulas and transactions formulas, and the theory needs to evaluate these formulas differently: occurrences should be tested for happening w.r.t. a path which is already fixed, while transactions should be executed similarly to what is traditionally done in $\mathcal{TR}$.

However, care must be taken when executing reactive transactions. In fact, and regarding 2), the execution of a reactive transaction depends on the events triggered. Viz., as in database triggers, an event occurring during a transaction execution can delay that transaction to commit (or succeed) until the event response is successfully executed. Moreover, the failure of such response should imply the failure of the whole transaction, and if some events were inferred based on the changes of the failed transaction, then these event occurrences must be considered as to have never happened. Encoding this behavior requires that, if an event occurs during a transaction, then its execution needs to be *expanded* with the event response. Additionally, this also precludes transactions to succeed on paths where an event occurs and is not responded to (even if the transaction would succeed in that path if the event did not exist).

For addressing these issues, in the following chapter we define Transaction Logic with Events ($\mathcal{TR}^{ev}$), a non-monotonic extension of Transaction Logic to combine the detection of complex reactive events and the execution of transactions.

# 11

# $\mathcal{TR}^{ev}$: Reactive Transaction Logic

In this chapter we introduce Transaction Logic with Events, $\mathcal{TR}^{ev}$, a logic to execute and reason about the behavior of reactive transactions, i.e., transactions that are issued automatically in response to the occurrence of complex events. For that, $\mathcal{TR}^{ev}$ integrates the ability of the original Transaction Logic to reason and execute transactions over very general forms of KBs, with the ability to detect complex events over paths of past execution.

In a nutshell, a $\mathcal{TR}^{ev}$ program is composed by transaction rules and event rules, and its semantics evaluates events and transactions differently, according to their nature. In this sense, transactions are *executed*, while events are *detected* w.r.t. to a fixed path which can be seen as the history of what has happened.

Besides the standard transactional properties already addressed by the original Transaction Logic, transactions in $\mathcal{TR}^{ev}$ are further required to answer all the events that occur during their execution, as in active database systems. To achieve this, the satisfaction of transactions is dependent on the satisfaction of events (where all detected events must be responded to), similarly to the behavior of database triggers, if an event is detected during the execution of a transaction, then this execution is *expanded* with the execution of that event response. Additionally, if it is not possible to execute such response starting in that path, then the transaction fails, even if the transaction would succeed in that path if the event did not exist. As a consequence of this behavior, and contrary to Transaction Logic, $\mathcal{TR}^{ev}$ is *non-monotonic*, since adding event rules to the program may preclude the success of transactions that would succeed otherwise.

As the original Transaction Logic, $\mathcal{TR}^{ev}$ integrates two oracles ($\mathcal{O}^d$ and $\mathcal{O}^t$) as a parameter of the theory. These are responsible to determine what primitive formulas are true in states and in the transition of states, allowing $\mathcal{TR}^{ev}$ to not commit to any particular semantics of states and updates, and to make it useful for a wide range of applications.

In addition to these oracles, $\mathcal{TR}^{ev}$ also takes as a parameter an event function $choice$ which encapsulates the response policies decisions of a reactive language. Simply put, $choice$ determines how should an event be responded, and what is the next event to be responded in case that more than one event occurs simultaneously. As it is done with the oracles, abstracting these choices from the theory allows $\mathcal{TR}^{ev}$ to be instantiated with several different response policies depending on the application in mind.

Next, we continue to formalize $\mathcal{TR}^{ev}$. We start by defining $\mathcal{TR}^{ev}$ syntax (section 11.1) and model theory (section 11.2). We elaborate on the role of the $choice$ function and its possible instantiations (section 11.3). Then, we provide the notion of executional entailment and specify some properties of the logic (section 11.4), and propose a procedure for executing $\mathcal{TR}^{ev}$ programs (section 11.5). We end this chapter by explaining how $\mathcal{TR}^{ev}$ can be used as an Event-Condition-Action Language (section 11.6). Finally, in chapter 12 we provide a discussion of $\mathcal{TR}^{ev}$'s properties and comparison with related work.

Most of the theory presented in this chapter was published in [GA14b].

## 11.1 $\mathcal{TR}^{ev}$ Syntax

$\mathcal{TR}^{ev}$'s alphabet contains an infinite number of constants $\mathcal{C}$, function symbols $\mathcal{F}$, predicate symbols $\mathcal{P}$ and variables $\mathcal{V}$. Additionally, predicate symbols are further partitioned into transaction names ($\mathcal{P}_t$), event names ($\mathcal{P}_e$), and oracle primitives ($\mathcal{P}_\mathcal{O}$). The latter are predicates defined by the data and transition oracles which, as in $\mathcal{TR}$, define the interaction (querying and updating, respectively) with the KB. Transaction names are the possible names that can appear in the program to define complex transaction procedures; and finally event names are the names of events that can occur and be responded. As before, we work with a Herbrand instantiation of the language.

To construct complex formulas, $\mathcal{TR}^{ev}$ uses the same connectives from $\mathcal{TR}$, and the new sequence connective ; which is useful to express common complex events. $\phi;\psi$ denotes that $\psi$ is true after $\phi$ but possibly interleaved with other occurrences, and it can be written in terms of $\mathcal{TR}$ connectives as: $\phi \otimes \texttt{path} \otimes \psi$ where `path` is a tautology that holds on paths of arbitrary size [BK98b], (e.g. $\texttt{path} \equiv (\varphi \vee \neg\varphi)$).

As previously mentioned, combining events and transactions implies two different and separate concepts: the detection of event occurrences, and the execution of its response rule as a transaction. Their separation is based on their intrinsic differences of meaning: while the event occurrence is meant to be detected (i.e., check for happening), event responses are meant to be executed transactionally. Nevertheless, these two concepts are naturally connected, as the detection of the former must drive the execution of the latter.

This relation between occurrences and responses of an event is reflected both in the syntax and semantics of $\mathcal{TR}^{ev}$. Formulas are partitioned into transaction and event formulas, and event occurrences and responses are syntactically represented w.r.t. a given event $e \in \mathcal{P}_e$. $\mathbf{o}(e)$ denotes the occurrence of event $e$, and $\mathbf{r}(e)$ its associated response.

Next we make precise what are transaction formulas, event formulas and $\mathcal{TR}^{ev}$ programs.

**Definition 44** (Transaction Formulas)**.** *A* transaction atom *is either a proposition in $\mathcal{P}_t$, $\mathcal{P}_e$, $\mathcal{P}_\mathcal{O}$, or $\mathbf{r}(\varphi)$ where $\varphi \in \mathcal{P}_\mathcal{O} \cup \mathcal{P}_e$. A* transaction formula *is either a transaction atom or an expression, defined inductively, of the form $\neg\phi$, $\Diamond\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, or $\phi \otimes \psi$ where $\phi$ and $\psi$ are transaction formulas.*

**Definition 45** (Event Formulas)**.** *An* event occurrence *is of the form $\mathbf{o}(\varphi)$ where $\varphi \in \mathcal{P}_e$ or $\varphi \in \mathcal{P}_\mathcal{O}$. An event formula is either an event occurrence, or an expression $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \otimes \psi$, or $\phi; \psi$ where $\phi$ and $\psi$ are event formulas.*

**Definition 46** (Rules and Programs)**.** *A* transaction rule *is a formula of the form $\varphi \leftarrow \psi$ s.t. $\varphi$ is a transaction atom and $\psi$ a transaction formula.*
*A* complex event rule *is a formula of the form $\varphi \Rightarrow \psi$ s.t. $\psi$ is an event occurrence and $\varphi$ is an event formula. A program $P$ is a set of transaction rules and complex event rules.*

As can be seen in definition 45, event occurrences are defined using the set $\mathcal{P}_\mathcal{O} \cup \mathcal{P}_e$. In this sense, event occurrences from $\mathcal{P}_e$ denote occurrences of events defined in the program by using rules (and more precisely, complex event rules), or occurrences of events that arrive as external requests (external events). Event occurrences from $\mathcal{P}_\mathcal{O}$ denote occurrences inferred from the primitive actions. These allow us to react to the occurrence of atomic changes similarly to the SQL triggers: "on insert" and "on delete". Obviously, other primitive actions and events are possible, and these depend on how the oracles are instantiated.

Note that contrary to transaction formulas, the hypothetical operator $\Diamond$ is not allowed to define event patterns. Since events are meant to be detected w.r.t. a past history, it makes little sense to define an event pattern based on the possible execution of some formula.

Another important notion is that event names (i.e., propositions in $\mathcal{P}_e$) are transaction formulas. This is what allows the triggering of *explicit* events as defined in (10.2), and is crucial to deal with external events. In fact, besides the atomic and complex event categories, we also partition events into inferred and explicit, depending on how these are triggered. Inferred events are events that become true due to the occurrence of some pattern, or the occurrence of a primitive action. In opposition, explicit events become true due to an explicit (internal or external) request, i.e., due to the execution of a transaction formula that belongs to $\mathcal{P}_e$.

As in reactive systems, $\mathcal{TR}^{ev}$ receives a sequence of external events which causes the execution of transactions in response. This is defined as $P, D_1 - \models e_1 \otimes \ldots \otimes e_n$, where $D_1$ is the initial KB state and $e_1 \otimes \ldots \otimes e_n$ is the sequence of external events that have arrived to the system. Here, we want to find the path starting from state $D_1$ encoding a KB evolution that responds to $e_1 \otimes \ldots \otimes e_n$. Triggering explicit events is a transaction formula that encodes the *action* of making an occurrence explicitly true (something not

needed in inferred events, e.g., defined by complex event rules). This explicit occurrence can then trigger complex events which, as we shall see, forces the execution of a response in $\mathcal{TR}^{ev}$.

Finally, note that event rules and transaction rules are syntactically distinguished using a different arrow as: $\phi \leftarrow \psi$ and $\psi \Rightarrow \phi$. This is to formalize the difference between transaction rules and event rules, and help the reader to distinguish between rules meant to be detected (i.e., event rules defining event patterns), and rules to specify transaction definitions (which are meant to be executed). Nevertheless, both formulas, $\phi \leftarrow \psi$ and $\psi \Rightarrow \phi$, are syntactic sugar for $\phi \vee \neg\psi$, and thus this difference is just syntactic.

## 11.2 $\mathcal{TR}^{ev}$ Model Theory

After defining the syntax of $\mathcal{TR}^{ev}$, we can now make precise how formulas are evaluated, and as usual, formulas are evaluated by interpretation functions $M$ over paths. However, paths in $\mathcal{TR}^{ev}$ are annotated with primitive action and explicit event occurrences. Thus, a path is a sequence of the form $\langle D_1 \overset{O_1}{\rightarrow} D_2 \overset{O_2}{\rightarrow} \ldots \overset{O_{k-1}}{\rightarrow} D_k \rangle$, such that $D_i$'s are states and $O_i$'s are occurrences. In it, $O_1$ is said to occur in the transition from $D_1$ to $D_2$, ..., and $O_{k-1}$ from $D_{k-1}$ to $D_k$. This information about what (occurrences) happen in state transitions is not part of $\mathcal{TR}$, and is convenient to store the *history* of primitive actions and explicit events that occur during a given execution, in a similar way to what is done in the theory of $\mathcal{ETR}$ (section 5.3).

This information about what has occurred in the transition of states is added to a path by the definition of interpretations and by their restrictions:

**Definition 47** (Interpretation)**.** *An interpretation is a mapping $M$ assigning a classical Herbrand structure (or $\top$) to every path. For a state $D_i$, primitive $\varphi \in \mathcal{P}_{\mathcal{O}}$ and occurrence $\mathbf{o}(e)$ where $e \in \mathcal{P}_e$, this mapping has the following restrictions:*

1. $\varphi \in M(\langle D \rangle)$ *if* $\varphi \in \mathcal{O}^d(D)$
2. $\{\varphi, \mathbf{o}(\varphi)\} \subseteq M(\langle D_1 \overset{\mathbf{o}(\varphi)}{\rightarrow} D_2 \rangle)$ *if* $\varphi \in \mathcal{O}^t(D_1, D_2)$
3. $\mathbf{o}(e) \in M(\langle D \overset{\mathbf{o}(e)}{\rightarrow} D \rangle)$

As in $\mathcal{TR}$, the first two points above, force all interpretations to satisfy primitive formulas on the paths where the oracles satisfy them, i.e., only the mappings that comply with the specified oracles are considered as interpretations. Additionally, point 2 also states that, whenever a primitive action $\varphi$ is made true by the oracle, the occurrence associated with the primitive action $\mathbf{o}(\varphi)$ is also made true in interpretations, and in this case, the path is annotated with the information of $\varphi$'s occurrence. As such, this restriction guarantees consistency and compliance of interpretations mappings w.r.t. the satisfaction of primitive occurrences, viz. whenever the oracle satisfies a primitive action in a transition, all the interpretations also satisfy both the primitive action, and the primitive occurrence in that same transition.

Finally, the third point guarantees that, in every interpretation, whenever an event is observed to occur in a transition, then the interpretation necessarily satisfies this occurrence. This point is an important technical detail to satisfy the action of explicitly triggering an event. By forcing interpretations to satisfy $\mathbf{o}(e)$ whenever it appears explicitly in the history of the path, we can impose compliance between the history of occurrences on a path and the set of formulas that interpretations make true on that same path. This is similar to what we have done in chapter 10 where, to reason about what (external) events were true, we had to restrict to the notion of compatible interpretations (cf. definition 38).

Note that making the occurrence of an event explicitly true does not change the KB state *per se* and thus, these transitions only take place on paths where the current state does not evolve. However, as we shall see, $\mathcal{TR}^{ev}$ theory will impose that, whenever $\mathbf{o}(e)$ is true in some part of a path (or subpath), then for a transaction to be satisfied, $\mathbf{r}(e)$ must also be true. Thus naturally, some actions may need to be executed to satisfy $\mathbf{r}(e)$ as an implicit result of making this occurrence true, which in turn, may cause changes in the KB.

**Example 29** ($\mathcal{TR}^{ev}$ interpretations)**.** *Consider a relational oracle as defined in chapter 3, page 28. Then for any interpretation $M$:*

$$\{a.ins, \mathbf{o}(a.ins)\} \subseteq M(\langle\{b\} \xrightarrow{\mathbf{o}(a.ins)} \{a, b\}), \text{ and}$$
$$\{b.del, \mathbf{o}(b.del)\} \subseteq M(\langle\{a, b\} \xrightarrow{\mathbf{o}(b.del)} \{a\}\rangle)$$

*Moreover, also for any $M$ it holds that: $\mathbf{o}(e_1) \in M(\langle\{a, b, c\} \xrightarrow{\mathbf{o}(e_1)} \{a, b, c\}\rangle)$*

As mentioned, satisfaction of *transaction formulas* means execution, and a transaction formula is satisfied on a path, if that path is a valid execution trace for the formula. Differently, satisfaction of *event formulas* means occurrence, and an event formula is satisfied on a path if the event occurs over that path. In this sense, primitive events (or atomic events), occur over particular transitions and are always bounded to 2-paths, i.e., paths of size 2. On the other hand, complex events normally occur over an interval. If a complex event formula $\phi$ is satisfied over $\langle D_0 \xrightarrow{O_1} D_1 \xrightarrow{O_2} \ldots \xrightarrow{O_k} D_k\rangle$ it means that $\phi$ occurs over this path, being initiated in the transition $D_0 \xrightarrow{O_1} D_1$ and ending in the transition $D_{k-1} \xrightarrow{O_k} D_k$. Moreover, although complex events may occur over an interval (i.e., over a path with size larger than 2), they are only said to be *fired* or *detected* in the final path transition. Whenever an event is fired, the transaction being executed needs to handle it by executing its associated response. As we shall see, this will cause a non-monotonic behavior of the transaction formulas.

In the following we specify how complex events and transactions are satisfied over paths. However, these satisfaction definitions require the prior re-definition of operations on annotated paths.

**Definition 48** (Subpath and Prefix)**.** *A subpath $\pi'$ of $\pi$ is a path composed by a subset of states and annotated occurrences of $\pi$ where both the order of the states and their annotations is preserved. A prefix $\pi_1$ of $\pi$ is any subpath of $\pi$ sharing the initial state.*

### 11.2.1 Satisfaction of Event Formulas

Since formulas in $\mathcal{TR}^{ev}$ are partitioned into transactions and events, next we start by specifying the satisfaction of event formulas. For this goal, we define the relation $\models_{ev}$, which satisfies events over general paths w.r.t. interpretation mappings $M$. In this case, $M, \pi \models_{ev} \phi$ means that in interpretation $M$, the event $\phi$ occurs due to path $\pi$.

**Definition 49** (Satisfaction of Event Formulas)**.** *Let $M$ be a interpretation, $\pi$ a path and $\phi$ an event formula. If $M(\pi) = \top$ then $M, \pi \models_{ev} \phi$; otherwise:*

1. ***Base Case:*** *$M, \pi \models_{ev} \phi$ iff $\phi \in M(\pi)$ for every event occurrence $\phi$*
2. ***Negation:*** *$M, \pi \models_{ev} \neg\phi$ iff it is not the case that $M, \pi \models_{ev} \phi$*
3. ***Disjunction:*** *$M, \pi \models_{ev} \phi \vee \psi$ iff $M, \pi \models_{ev} \phi$ or $M, \pi \models_{ev} \psi$.*
4. ***Serial Conjunction:*** *$M, \pi \models_{ev} \phi \otimes \psi$ iff there exists a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1 \models_{ev} \phi$ and $M, \pi_2 \models_{ev} \psi$*

Recall that the satisfaction of conjunctions and implications can be written in terms of $\vee$ and $\neg$, where as usual $\phi \wedge \psi$ means $\neg(\neg\phi \vee \neg\psi)$, and $\phi \Rightarrow \psi$ means $\phi \vee \neg\psi$. Also, the satisfaction of the sequence operator $\phi; \psi$ means $\phi \otimes \texttt{path} \otimes \psi$ where `path` is a tautology that holds on paths of arbitrary size.

**Example 30** (Satisfaction of events)**.** *Let's assume an interpretation $M$ such that:*

$$\mathbf{o}(e_1) \in M(\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}) \text{ and } \mathbf{o}(e_2) \in M(\{a\} \xrightarrow{\mathbf{o}(b.ins)} \{a, b\})$$

*Assuming that the latter is true, by definition 49, we have:*

$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \rangle \models_{ev} \mathbf{o}(e_1) \text{ and } M, \langle \{a\} \xrightarrow{\mathbf{o}(b.ins)} \{a, b\} \rangle \models_{ev} \mathbf{o}(e_2)$$

*Consequently, by the serial conjunction case we can conclude that*

$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(b.ins)} \{a, b\} \rangle \models_{ev} \mathbf{o}(e_1) \otimes \mathbf{o}(e_2) \text{ and}$$
$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(b.ins)} \{a, b\} \rangle \models_{ev} \mathbf{o}(e_1); \mathbf{o}(e_2)$$

*Note that, event $\mathbf{o}(e_2)$ and the complex event $\mathbf{o}(e_1); \mathbf{o}(e_2)$ are fired simultaneously, i.e., both are fired at the transition marked as $\mathbf{o}(b.ins)$. However, their occurrence is observed over different paths: $\mathbf{o}(e_2)$ is an atomic event occurring in the 2-path $\langle \{a\} \xrightarrow{\mathbf{o}(b.ins)} \{a, b\} \rangle$ while the complex event $\mathbf{o}(e_1); \mathbf{o}(e_2)$ occurs on the path $\langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(b.ins)} \{a, b\} \rangle$.*

It is important to note that $\models_{ev}$ coincides with the satisfaction relation of $\mathcal{TR}$. This is not surprising, as in the previous chapter 10 we have shown that $\mathcal{TR}$ model theory can indeed be used to reason about complex event patterns.

However, when satisfying transactions that need to react to complex events this is no longer the case and, in the following, we make precise how general transaction formulas can be satisfied over paths.

### 11.2.2 Satisfaction of Transaction Formulas

In opposition to events, transaction formulas are evaluated over the relation $\models$, where $M, \pi \models \phi$ means that in interpretation $M$, $\pi$ is a valid execution trace for transaction $\phi$.

A crucial notion is that the satisfaction relation $\models$ is dependent on the satisfaction of event formulas over $\models_{ev}$. This is to capture the database trigger behavior where transactions depend on the events triggered during its execution. Additionally, failing to respond to a trigger precludes the commit of the main transaction in which the trigger was fired. As a result, for $\pi$ to be a valid execution of $\phi$, then this means that (besides other things), all event occurrences satisfied on the path $\pi$ where $\phi$ was being executed, are responded on $\pi$. This transaction behavior where all event occurrences need to be responded is captured by the notion of path expansion, to be detailed below in definition 53.

**Definition 50** (Satisfaction of Transaction Formulas)**.** *Let $M$ be a interpretation, $\pi$ a path and $\phi$ a transaction formula. If $M(\pi) = \top$ then $M, \pi \models \phi$; otherwise:*

1. ***Base Case:*** *$M, \pi \models p$ iff there exists a prefix $\pi'$ of $\pi$ s.t. $p \in M(\pi')$ and $\pi$ is an expansion of path $\pi'$ w.r.t. $M$, for every transaction atom $p$ such that $p \notin \mathcal{P}_e$.*

2. ***Event Case:*** *$M, \pi \models e$ iff $e \in \mathcal{P}_e$ and there is a prefix $\pi'$ of $\pi$ s.t. $M, \pi' \models_{ev} \mathbf{o}(e)$ and $\pi$ is an expansion of path $\pi'$ w.r.t. $M$.*

3. ***Negation:*** *$M, \pi \models \neg\phi$ iff it is not the case that $M, \pi \models \phi$*

4. ***Disjunction:*** *$M, \pi \models \phi \vee \psi$ iff $M, \pi \models \phi$ or $M, \pi \models \psi$.*

5. ***Serial Conjunction:*** *$M, \pi \models \phi \otimes \psi$ iff there exists a prefix $\pi'$ of $\pi$ and some split $\pi_1 \circ \pi_2$ of $\pi'$ such that $M, \pi_1 \models \phi$ and $M, \pi_2 \models \psi$ and $\pi$ is an expansion of path $\pi'$ w.r.t. $M$.*

6. ***Executional Possibility:*** *$M, \pi \models \Diamond\phi$ iff $\pi$ is a 1-path of the form $\langle D \rangle$ for some state $D$ and $M, \pi' \models \phi$ for some path $\pi'$ that begins at $D$.*

Some words are in order regarding the Event Case. Besides the events that become true due to the execution of a primitive action, or due to the satisfaction of a complex event pattern, events can also be *explicitly* triggered during execution, as shown in eq. (10.2). Triggering explicit events is a transaction formula encoding the *action* of making an occurrence explicitly true, and the satisfaction of this action corresponds to the Event Case.

Additionally, note that with this definition, the serial conjunction operator is no longer an associative operator when applied to transaction formulas. This comes as a consequence of requiring the path to be expanded when satisfying transaction formulas, and when satisfying the serial conjunction. In fact, the formulas $((\phi_1 \otimes \phi_2) \otimes \phi_3)$ and $(\phi_1 \otimes (\phi_2 \otimes \phi_3))$ may not be entailed in the same path, as the order of expanding may

trigger events differently, as it is illustrated further in example 35. Due to this, in the absence of parentheses we always assume a left associative evaluation, i.e., $\phi_1 \otimes \phi_2 \otimes \phi_3 \equiv ((\phi_1 \otimes \phi_2) \otimes \phi_3)$

In addition, an *expansion* of a path $\pi_1$ w.r.t. to an interpretation $M$ is a new path $\pi_2$ where all events that occur in $\pi_1$ (and also in $\pi_2$) are completely answered.

Formalizing this notion of expansion requires the previous definition of what it means to answer an event.

**Definition 51** (Path response)**.** *For a path $\pi_1$ and an interpretation $M$ we say that $\pi$ is a response of $\pi_1$ iff $choice(M, \pi_1) = e$ and we can split $\pi$ into $\pi_1 \circ \pi_2$ such that $M, \pi_2 \models \mathbf{r}(e)$.*

The choice of what unanswered event should be answered at each moment in a path is given by an event function $choice$. This function has the role to decide what events are unanswered in a path $\pi$ w.r.t. a given interpretation $M$ and, based on a given criteria, select what event among them should be responded to first. Just like $\mathcal{TR}$ is parametric with a pair of oracles ($\mathcal{O}^d$ and $\mathcal{O}^t$), $\mathcal{TR}^{ev}$ takes the $choice$ function as an additional parameter. For now, we leave the definition and role of this function open, as we elaborate on it further, in section 11.3.

Nevertheless, and importantly, if all events that occur on a path $\pi$ are answered on $\pi$ w.r.t. an interpretation $M$, then $choice(M, \pi) = \epsilon$. With just this we can define what it means for a path to be completely answered, as follows.

**Definition 52** (Completely answered path)**.** *A path $\pi$ is said to be completely answered w.r.t. to an interpretation $M$ iff $choice(M, \pi) = \epsilon$.*

Based on these two definitions, we can now define what is an expansion of a path.

**Definition 53** (Expansion of a path)**.** *The path $\pi$ is an* expansion *of the path $\pi_1$ w.r.t. interpretation $M$ iff:*

- *$\pi$ is completely answered w.r.t. $M$, and*
- *either $\pi = \pi_1$; or there is a sequence of paths $\pi_1, \ldots, \pi$, starting in $\pi_1$ and ending in $\pi$, such that each $\pi_i$ in the sequence is a response of $\pi_{i-1}$ w.r.t. $M$.*

The latter definition specifies how to expand a path $\pi_1$ in order to obtain a path $\pi$ where all events satisfied over subpaths of $\pi$ are also answered within $\pi$. Intuitively, this expansion is obtained by sequently responding to an unanswered event in the path (given by the $choice$ function). Each path $\pi_i$ of the sequence $\pi_1, \pi_2, \ldots, \pi$ is a prefix of the path $\pi_{i+1}$, and where at least one of the events unanswered on $\pi_i$ is now answered on $\pi'$; otherwise, if all events occurring over $\pi_i$ are answered, then $\pi_i = \pi$, and the expansion is complete.

We should note that, since complex events are possible, nothing prevents $\pi_{i+1}$ to have more unanswered events than $\pi_i$. Moreover, it may not be possible to address all events

in a finite path, and thus, it may be the case that such a sequence of paths ending in a completely answered path $\pi$ does not exists. In fact, non-termination is a common problem of reactive systems, and is often undecidable for the unrestricted case [BDR04]. However, if termination is possible, then this sequence is defined, and each $\pi_i$ is an approximation of the final path $\pi$.

Before continuing, we state the paramount importance of a path to be completely answered for the satisfaction of a transaction formula on $\pi$.

**Lemma 2.** *Let $M$ be an interpretation, $\phi$ a transaction formula without negation and $\pi$ a path.*

$$M, \pi \models \phi \text{ only if } \pi \text{ is a completely answered path}$$

*Proof.* Immediately from definition 50 □

We now define what interpretations are models of a program in the standard way. In it, an interpretation is said to model a program if the interpretation satisfies all the rules in the program in every possible path. This is equivalent to say that, an interpretation satisfies a rule if, whenever it satisfies the antecedent, it also satisfies the consequent.

**Definition 54** (Model of a Program)**.** *An interpretation $M$ is a* model *of a transaction formula $\phi$ iff for every path $\pi$:*

$$M, \pi \models \phi$$

*$M$ is a model of an event formula $\psi$ iff for every path $\pi$:*

$$M, \pi \models_{ev} \psi$$

*$M$ is a model of a program $P$ (denoted $M \models P$) iff it is a model of every (transaction and complex event) rule in $P$.*

### 11.2.3 $\mathcal{TR}^{ev}$ Model Theory Examples

In the following we provide examples to illustrate $\mathcal{TR}^{ev}$'s model theory and the dynamics of its satisfaction relations $\models$ and $\models_{ev}$.

Since we have not yet defined the $choice$ function, the examples in this section were elaborated so that at most one event occurrence is satisfied in a path argument of $\pi$, and the $choice$ function simply picks it.

For brevity, we assume the responses of events inferred from primitive actions to hold trivially whenever their rules do not appear explicitly in the program. This is important as $\mathcal{TR}^{ev}$ prevents transaction formulas to hold (w.r.t. $\models$) on paths where the occurrence of an event holds (w.r.t. $\models_{ev}$). Consequently, if $p$ is an oracle primitive, $p \in \mathcal{O}^t(\pi)$ and a rule for $\mathbf{r}(p)$ is not defined in the program, then there will be some models of that program where the satisfaction of every transaction formula on path $\pi$ is constrained because $\mathbf{o}(p) \in M(\pi)$ and there is no rule specifying the execution of $\mathbf{r}(p)$. As such, whenever the

response of an oracle's primitive action is not defined, we assume that this response holds trivially in every path, i.e., that $\mathbf{r}(p) \leftarrow true$ belongs to the program. Nevertheless, note that we only assume this rule if another response rule for that primitive is not present, and only for these events arising from oracle's primitives.

**Example 31** (Non-monotonicity)**.** *Consider the following $\mathcal{TR}^{ev}$ programs $P_1$ and $P_2$.*

$$\begin{array}{l} p \leftarrow a.ins \\ \mathbf{r}(e_1) \leftarrow c.ins \end{array} \qquad (P_1) \qquad\qquad \begin{array}{l} p \leftarrow a.ins \\ \mathbf{r}(e_1) \leftarrow c.ins \\ \mathbf{o}(a.ins) \Rightarrow \mathbf{o}(e_1) \end{array} \qquad (P_2)$$

*Clearly, the two programs $P_1$ and $P_2$ differ only on one event rule, and define the same execution action for transaction $p$. In $P_1$, the transaction formula $p$ holds on the path $\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}\rangle$. This is true since all interpretations must comply with the oracles and thus we know that:*

$$\textit{for all } M\textit{: } a.ins \in M(\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}\rangle)$$

*which implies that:*

$$M, \langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}\rangle \models a.ins \qquad \textit{(in } P_1\textit{)}$$

*Moreover let's assume that $M$ is a model of the program, then it must satisfy all rules in program $P_1$, and in particular the rule $p \leftarrow a.ins$. As a consequence we have:*

$$M, \langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}\rangle \models p \qquad \textit{(in } P_1\textit{)}$$

*However, in program $P_2$, we have an additional rule saying that event $e_1$ occurs whenever the primitive $a.ins$ occurs. Additionally, (and as it also happened in $P_1$) we know that:*

$$\forall M. \mathbf{o}(a.ins) \in M(\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}\rangle)$$

*From this, assuming $M$ as a model of $P_2$ then we know that the following is also true:*

$$\mathbf{o}(e_1) \in M(\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}\rangle) \qquad \textit{(in } P_2\textit{)}$$

*Since $e_1$ has a response defined, then on the path $\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}\rangle$ the occurrence $e_1$ is unanswered and thus both the transactions $p$ and $a.ins$ cannot succeed in that path. In particular, the satisfaction of $\mathbf{o}(e_1)$ raises further requirements on the execution of* every *transaction on the path $\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}\rangle$. In order for transaction formulas to succeed, this path needs to be* expanded *with $e_1$'s response. Moreover, since:*

$$\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\}\rangle \textit{ is an expansion of } \langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}\rangle$$

*then, both transactions $p$ and $a.ins$ succeed in the* longer *path $\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\}\rangle$, i.e., for an interpretation $M$ that is a model of $P_2$:*

$$M, \langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\}\rangle \models p \textit{ and } M, \langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\}\rangle \models a.ins$$

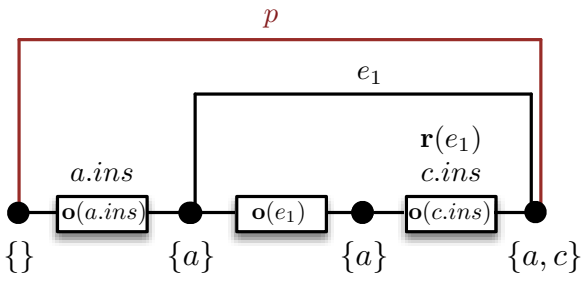
*This example illustrates the non-monotonicity of $\mathcal{TR}^{ev}$, viz. that adding a new event definition to $P_1$ falsifies the satisfaction of the transaction formulas $p$ and $a.ins$ on paths where they were previously true.*

In $\mathcal{TR}^{ev}$, as its predecessor $\mathcal{TR}$, every formula that is meant to be executed, is meant to be executed as a transaction. However, in contrast to its predecessor, executing a formula as a transaction in $\mathcal{TR}^{ev}$ requires further care to make sure that every event that occurs during this execution is properly addressed. As such in program $P_2$ of example 31, when evaluating the action primitive $a.ins$ as a transaction formula, $a.ins$ cannot succeed on the path $\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}\rangle$ because not all events are responded in that path. However, note that the action $a.ins$ belongs to every interpretation $M$ of that path (due to the restrictions in the interpretations' definition). This means that the primitive action $a.ins$ is true on the path $\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}\rangle$ although the *transaction* formula $a.ins$ is not.

Besides atomic and complex events, events in $\mathcal{TR}^{ev}$ can also be categorized as inferred events and explicit events, depending on how they are triggered. In this sense, inferred events are events defined by complex event rules, or that depend on oracle's primitive, and which are true whenever the pattern associated with them becomes true on a path. In opposition, explicit events are (atomic) events that are made *explicitly* true at a given point in execution. As such, they can appear directly in the body of a transaction rule, and arrive as external requests (and in this case are known as external events). Handling an explicit event as a *transaction* formula means to explicitly make the event occurrence true on that path and implicitly execute the responses of all events that become true due to this occurrence. This behavior is handled by the Event Case of definition 50 and is illustrated in the following example.

**Example 32** (Explicit events)**.** *Consider the following rules and the right-hand side figure, which illustrates the satisfaction of formulas over the path $\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(e_1)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\}\rangle$.*

$p \leftarrow a.ins \otimes e_1$
$\mathbf{r}(e_1) \leftarrow c.ins$




*Here we can see that transaction formula $p$ succeeds over the path $\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(e_1)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\}\rangle$,*

*This is so because, for any interpretation $M$:*

$$\begin{aligned} M, \langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\}\rangle &\models a.ins \\ M, \langle\{a\} \xrightarrow{\mathbf{o}(e_1)} \{a\}\rangle &\models e_1 \\ M, \langle\{a\} \xrightarrow{\mathbf{o}(e_1)} \{a\}\rangle &\models_{ev} \mathbf{o}(e_1) \\ M, \langle\{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\}\rangle &\models \mathbf{r}(e_1) \end{aligned}$$

*As a result for an interpretation $M$ that is a model of the program rules:*

$\langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(e_1)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\}\rangle$ *is an expansion of* $\langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(e_1)} \{a\}\rangle$

*And thus, there is a model $M$ of the program where:*

$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(e_1)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\}\rangle \models a.ins \otimes e_1 \text{ and}$$
$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(e_1)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\}\rangle \models p$$

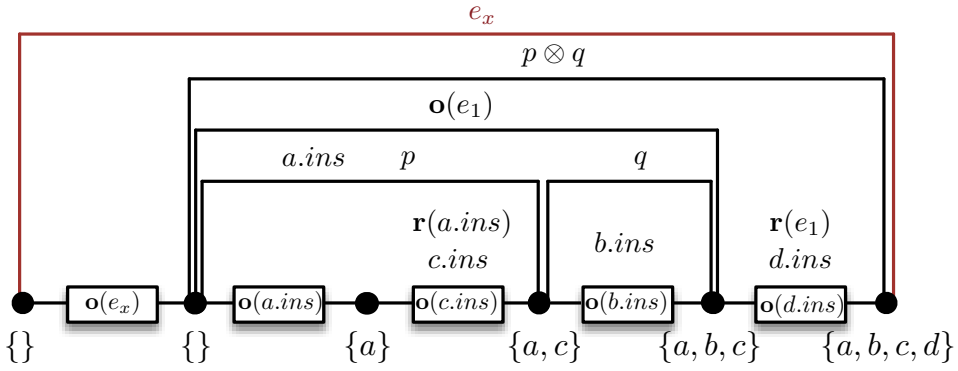
*This means that, when starting in the empty state $\{\}$, satisfying transaction $p$ given these rules means inserting $a$ in the KB and further satisfy the event transaction $e_1$. Moreover, satisfying $e_1$ as a transaction means satisfying the occurrence $e_1$ explicitly and implicitly satisfy its response. Note that, $\mathbf{o}(e_1)$ appears directly in the annotation of the path. Satisfying $\mathbf{o}(e_1)$ does not change the KB state although it implicitly forces the execution of $\mathbf{r}(e_1)$.*

In the following example, we illustrate on the satisfaction of an external event $e_x$. As mentioned, besides appearing in the body of transaction rules, explicit events can also come as external system requests (*external events*). In this case, we want to know the possible evolutions of the KB (i.e., the paths) that satisfy the event given the transaction program. This notion corresponds to executional entailment, which will be detailed further in Section 11.4.

**Example 33** (Serial-conjunction)**.** *Consider the following rules:*

$$\begin{array}{ll} p \leftarrow a.ins & \\ q \leftarrow b.ins & \\ \mathbf{r}(e_x) \leftarrow p \otimes q & \\ \mathbf{r}(e_1) \leftarrow d.ins & \mathbf{o}(a.ins); \mathbf{o}(b.ins) \Rightarrow \mathbf{o}(e_1) \\ \mathbf{r}(a.ins) \leftarrow c.ins & \end{array}$$

*Moreover, given this program the following figure illustrates the satisfaction of the (possibly external) event $e_x$ starting over the state $\{\}$.*




*As one can see, the occurrence of $e_x$ forces the satisfaction of the transaction $p \otimes q$, which is true if both its "subformulas" ($p$ and $q$) are satisfied over smaller paths. Additionally, we need to satisfy*

*all the event occurrences that become true due to the independent execution of $p$ and $q$ but also the complex events detected in the conjunction $p \otimes q$. As such, for a model $M$ of the given program, we know that:*

$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\} \rangle \models p \text{ and}$$
$$M, \langle \{a, c\} \xrightarrow{\mathbf{o}(b.ins)} \{a, b, c\} \rangle \models q$$

*Note that, by definition of the relation $\models$, all occurrences detected over the independent paths that satisfy $p$ and $q$ are already responded in those paths. Particularly, the event $\mathbf{o}(a.ins)$ occurs during the execution of $p$ and it is responded within its execution.*

*However, care must be taken to cater for the events arising from the serial conjunction execution, i.e., from the composed execution of $p$ and $q$. In this sense, the rule $\mathbf{o}(a.ins); \mathbf{o}(b.ins) \Rightarrow \mathbf{o}(e_1)$ defines one pattern for the occurrence of $e_1$ which constrains the execution of transaction $p \otimes q$ and forces the expansion of the path to satisfy $\mathbf{r}(e_1)$. Consequently, for an interpretation $M$ that is a model of the program rules:*

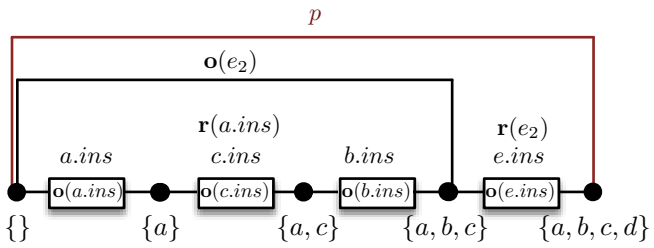
$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\} \xrightarrow{\mathbf{o}(b.ins)} \{a, b, c\} \xrightarrow{\mathbf{o}(d.ins)} \{a, b, c, d\} \rangle \models p \otimes q$$

*And finally, given this, we know that:*

$$M, \langle \{\} \xrightarrow{\mathbf{o}(e_x)} \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\} \xrightarrow{\mathbf{o}(b.ins)} \{a, b, c\} \xrightarrow{\mathbf{o}(d.ins)} \{a, b, c, d\} \rangle \models e_x$$

**Example 34** (Sequence vs. serial-conjunction)**.** *Consider the following rules and the right-hand side figure illustrating satisfaction of formulas over the path $\langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\} \xrightarrow{\mathbf{o}(b.ins)} \{a, b, c\} \xrightarrow{\mathbf{o}(e.ins)} \{a, b, c, e\} \rangle$:*

$p \leftarrow a.ins \otimes b.ins$
$\mathbf{r}(a.ins) \leftarrow c.ins$
$\mathbf{r}(e_1) \leftarrow d.ins$
$\mathbf{r}(e_2) \leftarrow e.ins$
$\mathbf{o}(a.ins) \otimes \mathbf{o}(b.ins) \Rightarrow \mathbf{o}(e_1)$
$\mathbf{o}(a.ins); \mathbf{o}(b.ins) \Rightarrow \mathbf{o}(e_2)$




*This example shows the difference between the serial conjunction operator $\otimes$ and the sequence operator ;. Recall that $\mathbf{o}(a.ins)$ and $\mathbf{o}(b.ins)$ are events that become true whenever their associated primitive actions ($a.ins$ and $b.ins$) appear in the path. Intuitively, event $e_1$ occurs whenever the primitive action $b.ins$ occurs* immediately *after the primitive $a.ins$, while, event $e_2$ occurs whenever the primitive actions $b.ins$ occurs* eventually after *$a.ins$.*

*Then, given these rules, when executing the composed transaction $a.ins \otimes b.ins$, the transaction $a.ins$ needs to respond to the event $\mathbf{o}(a.ins)$, since $\mathbf{r}(a.ins)$ is defined in the program rules. Thus, for an interpretation $M$ that is a model of the program rules $\langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \rangle$ is not an expansion of itself and:*

$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a, c\} \rangle \models a.ins$$

*i.e., starting in the empty path $\langle\{\}\rangle$ the transaction $a.ins$ is only satisfied on the path where $c.ins$ is executed: $\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a,c\}\rangle$.*

*As a result for an $M$ that is a model of the program, $\mathbf{o}(a.ins);\mathbf{o}(b.ins)$ holds on the path $\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a,c\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b,c\}\rangle$ although $\mathbf{o}(a.ins) \otimes \mathbf{o}(b.ins)$ does not:*

$$M, \langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a,c\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b,c\}\rangle \models_{ev} \mathbf{o}(a.ins);\mathbf{o}(b.ins) \text{ and}$$
$$M, \langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a,c\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b,c\}\rangle \not\models_{ev} \mathbf{o}(a.ins) \otimes \mathbf{o}(b.ins)$$

*Since $\mathbf{o}(e_2)$ holds on that path, then:*

$$\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a,c\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b,c\}\rangle \text{ is not an expansion of}$$
$$\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a,c\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b,c\}\rangle \text{ and}$$
$$M, \langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a,c\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b,c\}\rangle \not\models p$$

*This means that because of the event defined by the sequence pattern $\mathbf{o}(a.ins);\mathbf{o}(b.ins)$, the transaction $p$ cannot succeed on the path $\langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a,c\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b,c\}\rangle$, as this path needs to be expanded with $e_2$'s response. Since starting in state $\langle\{a,b,c\}\rangle$, the transaction $\mathbf{r}(e)$ holds on path $\langle\{a,b,c\} \xrightarrow{\mathbf{o}(e.ins)} \{a,b,c,e\}\rangle$, then:*

$$M, \langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a,c\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b,c\} \xrightarrow{\mathbf{o}(e.ins)} \{a,b,c,e\}\rangle \models p$$

**Example 35** (Non-associative serial-conjunction)**.** *Here we aim to show why the operator $\otimes$ is no longer an associative operator, i.e., that $\phi_1 \otimes (\phi_2 \otimes \phi_2)$ is not equivalent to $(\phi_1 \otimes \phi_2) \otimes \phi_3$. Note that in the absence of parentheses we assume all operators to be left-associative, and thus $\phi_1 \otimes \phi_2 \otimes \phi_3 \equiv (\phi_1 \otimes \phi_2) \otimes \phi_3$.*

*When evaluating $\phi_1 \otimes \phi_2$ we require the expansion of the path after the individual satisfaction of $\phi_1$ and $\phi_2$. Since to satisfy individually $\phi_1$ and $\phi_2$ all the events that become true in the individual execution of each of the formulas are necessarily responded to, this expansion will further verify which events have become true due to the* composed *execution of $\phi_1 \otimes \phi_2$. Consequently, the order of expansion will be different, depending on how we evaluate the formula, i.e., if we expand $\phi_1 \otimes \phi_2$ before $(\phi_1 \otimes \phi_2) \otimes \phi_3$; or $\phi_2 \otimes \phi_3$ before $\phi_1 \otimes (\phi_2 \otimes \phi_3)$.*

*To better illustrate this, consider the following program.*

$$\begin{array}{rll}
p & \leftarrow a.ins \otimes b.ins \otimes c.ins & \\
q & \leftarrow a.ins \otimes (b.ins \otimes c.ins) & \\
\mathbf{r}(e_1) & \leftarrow d.ins & \mathbf{o}(a.ins);\mathbf{o}(b.ins) \quad\quad \Rightarrow \mathbf{o}(e_1) \\
\mathbf{r}(e_2) & \leftarrow d.del & \mathbf{o}(a.ins);\mathbf{o}(d.ins);\mathbf{o}(c.ins) \quad \Rightarrow \mathbf{o}(e_2)
\end{array}$$

*Given these rules, we know for an interpretation $M$ that is a model of the program that:*

$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b\} \xrightarrow{\mathbf{o}(d.ins)} \{a,b,d\} \rangle \models a.ins \otimes b.ins$$

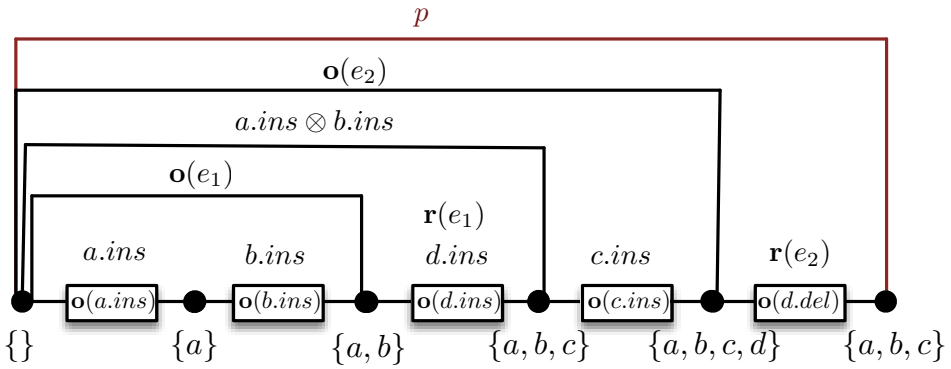
*and*

$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b\} \xrightarrow{\mathbf{o}(d.ins)} \{a,b,d\} \xrightarrow{\mathbf{o}(c.ins)} \{a,b,c,d\} \xrightarrow{\mathbf{o}(d.del)} \{a,b,c\} \rangle \models a.ins \otimes b.ins \otimes c.ins$$

*and*

$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b\} \xrightarrow{\mathbf{o}(d.ins)} \{a,b,d\} \xrightarrow{\mathbf{o}(c.ins)} \{a,b,c,d\} \xrightarrow{\mathbf{o}(d.del)} \{a,b,c\} \rangle \models p$$

*As such, this satisfaction of transaction $p$ over a path that starts on the state $\{\}$ can be illustrated by the following figure:*




*In opposition, when considering the satisfaction of the transaction formula $q$ starting in the same empty path, we know that:*

$$M, \langle \{a\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b\} \xrightarrow{\mathbf{o}(c.ins)} \{a,b,c\} \rangle \models b.ins \otimes c.ins$$

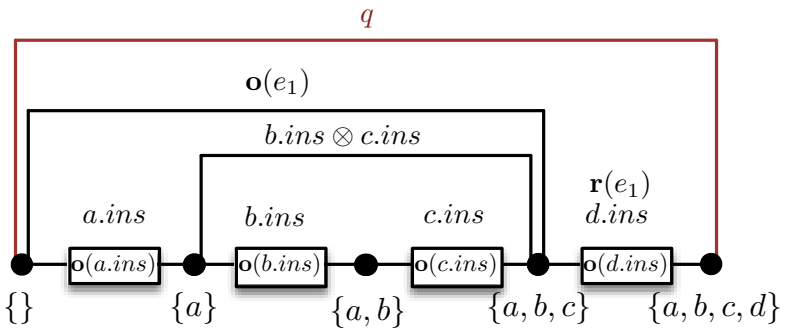
*and*

$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b\} \xrightarrow{\mathbf{o}(c.ins)} \{a,b,c\} \xrightarrow{\mathbf{o}(d.ins)} \{a,b,c,d\} \rangle \models a.ins \otimes (b.ins \otimes c.ins)$$

*and*

$$M, \langle \{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b\} \xrightarrow{\mathbf{o}(c.ins)} \{a,b,c\} \xrightarrow{\mathbf{o}(d.ins)} \{a,b,c,d\} \rangle \models q$$

*Clearly, in the latter case, event $e_1$ is triggered only after $c.ins$ is executed, and as a consequence, $e_2$ does not occur over that path which satisfies $a.ins \otimes (b.ins \otimes c.ins)$. As before, we illustrate the satisfaction of transaction $q$ on a path starting in state $\{\}$ as follows:*



## 11.3 Event Choice Function

The previous examples were defined so that only one event is triggered at each moment. However, this may not be always the case, and for that a reactive language specifies an *operational behavior* which is responsible for picking the event to be handled at a given moment in case of a conflict.

In summary, an operational semantics encases two major decisions: 1) in which order should events be responded when more than one event is detected simultaneously; and 2) how should an event be responded. In order to make $\mathcal{TR}^{ev}$ as flexible as possible, its theory was abstracted from these decisions, encapsulating them in a $choice$ function. This function is required as a parameter of the theory (similarly to the oracles $\mathcal{O}^t$ and $\mathcal{O}^d$) and precisely defines what is the next event that still needs to be responded to. However, like the oracles, to use $\mathcal{TR}^{ev}$ to reason about and execute reactive transactions, the $choice$ function needs to be precisely defined. In the following we illustrate how this function can be differently instantiated to encode different operational semantics decisions, for different application domains.

**Definition 55** ($choice$ function)**.** *Let $M$ be an interpretation and $\pi$ be a path. Then function $choice(M, \pi)$ is defined as follows:*

$$choice(M, \pi) = \mathit{firstUnans}(M, \pi, order(M, \pi))$$

Matching the two major decisions of an operational semantics, our definition of the $choice$ function is partitioned in two different functions: the $order$ function specifying how events should be sorted w.r.t. a given criteria, and a $\mathit{firstUnans}$ function which checks what events are unanswered and returns the first one based on the previous order. The former decision defines the handling order of events, i.e. given a set of occurring events, what should be responded first. This ordering can be defined e.g. based on when they have occurred (temporal order), on a priority list, or any other criteria. Then, the latter decision defines the response policy of an ECA-language, i.e., how should an event be responded. For instance, if an event occurs more than once before the system can respond to it, this specifies if such response should be issued only once or responded as many times as it occurred.

Next we illustrate how these functions can be instantiated to achieve different behaviors. We start with the $order$ function which is responsible for deciding what event should be address first, when more than one event is triggered simultaneously:

**Example 36** (Ordering-Functions)**.** *Let $\langle e_1, \ldots, e_n \rangle$ be a sequence of events, $\pi$ a path and $M$ an interpretation.*

**Temporal Ending Order** *$order(M, \pi) = \langle e_1, \ldots, e_n \rangle$ iff $\forall e_i$ s.t. $1 \leq i \leq n$ then $\exists \pi_i$ subpath of $\pi$ where $M, \pi \models_{ev} \mathbf{o}(e_i)$ and $\forall e_j$ s.t. $i < j$ then $e_j$ occurs after $e_i$*

**Temporal Starting Order** *$order(M, \pi) = \langle e_1, \ldots, e_n \rangle$ iff $\forall e_i$ s.t. $1 \leq i \leq n$ then $\exists \pi_i$ subpath of $\pi$ where $M, \pi \models_{ev} \mathbf{o}(e_i)$ and $\forall e_j$ s.t. $i < j$ then $e_j$ starts before $e_i$*

**Priority List Order** *Let $L$ be a priority list where events are linked with natural numbers starting in $1$, where $1$ is the most priority event.*

*$order_L(M, \pi) = \langle e_1, \ldots, e_n \rangle$ iff $\forall e_i \; \exists \pi_i$ subpath of $\pi$ s.t. $M, \pi_i \models_{ev} \mathbf{o}(e_i)$ and $\forall e_j$ where $1 \leq i < j \leq n$, $\pi_j$ is subpath of $\pi$ and $M, \pi_j \models_{ev} \mathbf{o}(e_j)$ then $L(e_i) \leq L(e_j)$.*

Note that all these examples require some prior notion of event ordering, which can be defined as:

**Definition 56** (Ordering of Events)**.** *Let $e_1, e_2$ be events and $\pi$ a path and $M$ an interpretation. $e_2$* occurs after *$e_1$ w.r.t. $\pi$ and $M$ iff $\exists \pi_1, \pi_2$ subpaths of $\pi$ s.t. $\pi_1 = \langle D_i \, {}^{O_i}\!\!\rightarrow \ldots {}^{O_{j-1}}\!\!\rightarrow D_j \rangle$, $\pi_2 = \langle D_n \, {}^{O_n}\!\!\rightarrow \ldots {}^{O_{m-1}}\!\!\rightarrow D_m \rangle$, $M, \pi_1 \models_{ev} \mathbf{o}(e_1)$, $M, \pi_2 \models_{ev} \mathbf{o}(e_2)$ and $D_j \leq D_m$ w.r.t. the ordering in $\pi$. $e_1$* starts before *$e_2$ w.r.t. $\pi$ if $D_i \leq D_n$*

Choosing the appropriate event ordering criteria depends, obviously, on the application in mind, as different applications require different strategies. For instance, in system monitoring applications there may exist alarms with higher priority over others that need to be addressed immediately, while in a webstore context it may be more important to treat events in the temporal orders in which they are detected.

After exemplifying how events can be ordered, it remains to show how the *firstUnans* function can be instantiated. As mentioned, this function is what defines the response policy, i.e., what requisites should be imposed w.r.t. the response executions.

In the following we illustrate two alternative instantiations. In the first, encoded in *Relaxed Response*, the function simply returns the first event $e$ such that its response is not satisfied on a path after the occurrence. As such, with this instantiation, if an event occurs more than once before it is responded, then it is sufficient to respond to it once. From its characteristics, this instantiation can be useful for instance in alarm monitoring systems, where one only needs to answer alarms (e.g. a fire alarm) once, independently of the amount of times they were fired previously.

However, depending on the application, it may also be important to address each occurrence individually (e.g., in a webstore context). This requires for an alternative definition where responses are issued explicitly for each event. This is encoded in the *Explicit Response*, where we verify if $\mathbf{r}(e_i)$ is satisfied but always w.r.t. its correct order.

**Example 37** (Answering Choices)**.** *Let $\pi$ be a path, $M$ an interpretation and $\langle e_1, \ldots, e_n \rangle$ a sequence of events.*

**Relaxed Response** *$firstUnans(M, \pi, \langle e_1, \ldots, e_n \rangle) = e_i$ if $e_i$ is the first event in $\langle e_1, \ldots, e_n \rangle$ s.t. $\exists \pi'$ subpath of $\pi$ where $M, \pi' \models_{ev} \mathbf{o}(e)$ and $\neg \exists \pi''$ s.t. $\pi''$ is also a subpath of $\pi$, $\pi''$ is after $\pi'$ and $M, \pi'' \models \mathbf{r}(e)$.*

**Explicit Response** *$firstUnans(M, \pi, \langle e_1, \ldots, e_n \rangle) = e_i$ if $e_i$ is the first event in $\langle e_1, \ldots, e_n \rangle$ s.t. $\exists \pi'$ subpath of $\pi$ where $M, \pi' \models_{ev} \mathbf{o}(e)$ and if $\exists \pi''$ subpath of $\pi$ that is after $\pi'$ where $M, \pi'' \models \mathbf{r}(e_i)$ then $\exists \pi_1, \pi_2$ subpaths of $\pi$ and $\pi_2$ is after $\pi_1$ where $M, \pi_1 \models_{ev} \mathbf{o}(e_j)$, $M, \pi_2 \models \mathbf{r}(e_j)$, $j < i$ and $\pi''$ starts before the ending of $\pi_2$*

## 11.4 Executional Entailment and Properties

As in $\mathcal{TR}$ and $\mathcal{ETR}$, besides the notion of logical entailment which reasons about the properties of transaction and event formulas that hold for *every* possible path of execution, $\mathcal{TR}^{ev}$ also uses the notion of executional entailment which is used to talk about properties of a *particular* execution path. Consequently, as before, in the following we define what we mean by executional entailment and show some properties based on this notion.

However, to reason about the execution of transactions over a specific path, care must be taken since, as described above, the satisfaction of a new occurrence on a path may invalidate transaction formulas that were previously true. As such, adding a new rule to a program may make a formula that was previously satisfied on a path $\pi$ to be false on $\pi$.

To deal with a similar behavior, non-monotonic logics rely on the concept of minimal or preferred models. Since in these logics, considering new information can falsify previously known information about the world, then, instead of considering all possible models, non-monotonic theories restrict to the most skeptical ones. Likewise, $\mathcal{TR}^{ev}$ uses the *minimal models* of a program to define entailment, whenever talking about a particular execution of a formula.

Before defining the notion of minimal models, we start by defining minimal interpretations. As usual, minimality is defined by set inclusion on the amount of predicates that an interpretation satisfies, and a minimal model of a formula or theory is a model that minimizes the amount of atoms satisfied on a path.

**Definition 57** (Ordering of Structures)**.** *If $M_1$ and $M_2$ are interpretations then $M_1 \leq M_2$ if $\forall \pi$: $M_2(\pi) = \top \vee M_1(\pi) \subseteq M_2(\pi)$*

Based on the latter notion, a minimal model of a program $P$ is an interpretation that is a model of $P$, and that is minimal w.r.t. other comparable interpretations.

**Definition 58** (Minimal Model)**.** *Let $\phi$ be a (event or transaction) formula, and $P$ a program. $M$ is a* minimal model *of $\phi$ (resp. $P$) if $M$ is a model of $\phi$ (resp. $P$) and $M \leq M'$ for every model $M'$ of $\phi$ (resp. $P$).*

As opposed to $\mathcal{TR}$'s definition of executional entailment (cf. definition 6), executional entailment in $\mathcal{TR}^{ev}$ is defined w.r.t. minimal models. The intuition is that, to know if a formula succeeds in a particular path, we can only consider the event occurrences *supported* by that path, either because they appear as occurrences in the transition of states, or because they are a necessary consequence of the program's rules given that path. This is formalized as follows.

**Definition 59** ($\mathcal{TR}^{ev}$ Executional Entailment)**.** *Let $P$ be a program, $\phi$ a transaction formula and $D_0 \overset{O_1}{\rightarrow} \ldots \overset{O_n}{\rightarrow} D_n$ a path. Then the statement:*

$$P, (D_0 \overset{O_1}{\rightarrow} \ldots \overset{O_n}{\rightarrow} D_n) \models \phi \tag{11.1}$$

*holds iff for every minimal model $M$ of $P$, $M, \langle D_0 \xrightarrow{O_1} \ldots \xrightarrow{O_n} D_n\rangle \models \phi$. $P, D_1 - \models \phi$ is said to be true, if there is a path $D_0 \xrightarrow{O_1} \ldots \xrightarrow{O_n} D_n$ that makes (11.1) true.*

**Example 38.** *Recall program $P$ of example 33 above. Since for every minimal model $M_m$ of $P$ we have $M_m, \langle\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a,c\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b,c\} \xrightarrow{\mathbf{o}(d.ins)} \{a,b,c,d\}\rangle \models p \otimes q$ then we can conclude $P, (\{\} \xrightarrow{\mathbf{o}(a.ins)} \{a\} \xrightarrow{\mathbf{o}(c.ins)} \{a,c\} \xrightarrow{\mathbf{o}(b.ins)} \{a,b,c\} \xrightarrow{\mathbf{o}(d.ins)} \{a,b,c,d\}) \models p \otimes q$*

Based on this notion of entailment, we define the notion of program equivalence. Since to execute a formula given a program, we only look into the minimal models of that program, to formalize equivalence between two programs we look into the formulas that can be executed by the two programs. Thus, we say that two programs are equivalent, if they satisfy the same formulas over the same paths.

**Definition 60** (Program equivalence)**.** *Let $P_1$ and $P_2$ be programs, and $\phi$ a formula defined in both $P_1$ and $P_2$0s alphabet. We say that $P_1 \equiv P_2$ if for all paths $\pi$:*

$$P_1, \pi \models \phi \textit{ iff } P_2, \pi \models \phi$$

Interestingly, as in logic programs, formulas satisfied by this entailment have some kind of support. This is encoded as follows:

**Lemma 3** (Support)**.** *Let $P$ be a program, $\pi$ a path, $\phi$ a transaction atom. Then, if $P, \pi \models \phi$ one of the following holds:*

1. *$\phi$ is an elementary action and either $\phi \in \mathcal{O}^d(\pi)$ or $\phi \in \mathcal{O}^t(\pi)$;*
2. *$\phi$ is the head of a transaction rule in $P$ ($\phi \leftarrow$ body) and $P, \pi \models$ body;*

*Proof.* See appendix C in page 243 □

As expected, $\mathcal{TR}^{ev}$ extends $\mathcal{TR}$. Precisely, if a program $P$ has no complex event rules, and for every elementary action $a$ defined by the oracles the only rule for $\mathbf{r}(a)$ in $P$ is $\mathbf{r}(a) \leftarrow \texttt{true}$, then executional entailment in $\mathcal{TR}^{ev}$ can be recast in $\mathcal{TR}$. As an immediate corollary of this, it follows that if $P$ is *event-free* then executional entailment in $\mathcal{TR}^{ev}$ and in $\mathcal{TR}$ coincide.

**Theorem 10** (Comparison to $\mathcal{TR}$)**.** *Let $P$ be a complex-event free program, and let $P'$ be obtained from $P$ by replacing in $P$ every event $e$ and every response $\mathbf{r}(e)$, s.t. $e \in \mathcal{P}_e$, by a new fluent $p_e$. Let $\pi$ be an annotated path and $\pi'$ be a path obtained from $\pi$ removing the annotations and for every event occurrence $\varphi$ in $\pi$ s.t.$\pi = \pi_1 \circ \langle D \xrightarrow{\varphi} D\rangle \circ \pi_2$, then $\pi' = \pi_1 \circ \langle D\rangle \circ \pi_2$. Then for every transaction formula $\phi$:*

$$P, \pi \models \phi \textit{ iff } P', \pi' \models_{\mathcal{TR}} \phi$$

*Proof.* See appendix C on page 244. □

Finally, we should mention some properties when satisfying a stream of events. As previously stated, $\mathcal{TR}^{ev}$ can be used as a reactive system, where it receives a sequence of external events as $P, D_1 - \models e_1 \otimes \ldots \otimes e_n$, and where we want to find a path starting from state $D_1$ encoding a KB evolution that responds to $e_1 \otimes \ldots \otimes e_n$, in a transactional way. Clearly, the events $e_1 \otimes e_2 \otimes \ldots \otimes e_n$ may not be known all at the same time. I.e., we may receive a sequence of events $e_1 \otimes \ldots \otimes e_n$ and afterwards another sequence $e_{n+1} \otimes \ldots \otimes e_k$.

In this case, it is important to stress that, while there may exist some paths $\pi_1 \circ \pi_2$ where $P, \pi_1 \models e_1 \otimes \ldots \otimes e_n$ and $P, \pi_2 \models e_{n+1} \otimes \ldots \otimes e_k$, we cannot guarantee that $P, \pi \models e_1 \otimes \ldots \otimes e_n \otimes e_{n+1} \otimes \ldots \otimes e_k$ holds, for a super path $\pi$ of $\pi_1 \circ \pi_2$. In fact, there may be some complex event $e_c$ triggered due to the combination of $e_1 \otimes \ldots \otimes e_n$ with $e_{n+1} \otimes \ldots \otimes e_k$ that was not triggered in any of the individual execution. Consequently, if it is impossible to respond to this $e_c$, then $P, \pi \models e_1 \otimes \ldots \otimes e_n \otimes e_{n+1} \otimes \ldots \otimes e_k$ will not hold even if $P, \pi_1 \models e_1 \otimes \ldots \otimes e_n$ and $P, \pi_2 \models e_{n+1} \otimes \ldots \otimes e_k$ do.

However, if $P, \pi \models e_1 \otimes \ldots \otimes e_n \otimes e_{n+1} \otimes \ldots \otimes e_k$ is true, we can still say something about its execution. Namely that if we pick any sub formula of $e_1 \otimes \ldots \otimes e_n \otimes e_{n+1} \otimes \ldots \otimes e_k$, this formula will be satisfied in a subpath of $\pi$. This is encoded as follows:

**Lemma 4.** *Let $P$ be a $\mathcal{TR}^{ev}$ program, $\pi$ a path, and a serial conjunction $e_1 \otimes \ldots \otimes e_n \otimes e_{n+1} \otimes \ldots \otimes e_k$ where $1 \leq n \leq k$.*

$$\textit{If } P, \pi \models e_1 \otimes \ldots \otimes e_n \otimes e_{n+1} \otimes \ldots \otimes e_k \textit{ then there is a split } \pi_1 \circ \pi_2 \textit{ of } \pi \textit{ where } P, \pi_1 \models e_1 \otimes \ldots \otimes e_n$$

*Proof.* Immediately by the Serial Conjunction Case of definition 50. □

## 11.5 A Procedure for Executing Reactive Transactions

To make $\mathcal{TR}^{ev}$ useful in practice, in this section we propose a proof procedure for executing transactions under $\mathcal{TR}^{ev}$'s semantics. This procedure is sound and complete w.r.t the execution entailment of definition 87, and it is defined for a subset of $\mathcal{TR}^{ev}$ programs, similar to the serial-Horn program restrictions of $\mathcal{TR}$'s procedure.

Given a $\mathcal{TR}^{ev}$ program and a KB state, the procedure takes a *stream* of events that are known to occur, and finds a KB evolution where all the (possibly complex) events, resulting from the direct and indirect occurrence of this stream, are responded as a transaction. More precisely, the procedure finds solutions for statements of the form $P, D_0 - \models e_1 \otimes \ldots \otimes e_k$, by finding the paths where the formula $P, \pi \models e_1 \otimes \ldots \otimes e_k$ holds. These paths encode a KB evolution that is needed to respond to the event stream in a transactional way, given the program definition $P$.

As expected, in order to define a procedure for a reactive language, we need to make choices regarding the intended operational behavior, which in $\mathcal{TR}^{ev}$'s semantics is specified by instantiating the function $choice$. With this in mind, for this procedure, we fix an event ordering based on a priority list as defined in example 36, and assume an *Explicit Response* answering choice, as defined in example 37.

In a nutshell, the procedure is partitioned into two major parts: the execution of actions, based on a top-down computation, and the detection of event patterns, based on a bottom-up computation. Then, the detection of event patterns is inspired by the ETALIS detection algorithm [AFRSSS10], while the execution of transactions is based on $\mathcal{TR}$'s proof procedure specified in section 3.3.

As in ETALIS event detection algorithm, event rules are first pre-processed in a procedure called *binarization of events*. In it, event rules are transformed so that all their bodies have at most two atoms. Particularly, if we have an event rule with a body with more than two atoms, as e.g. $\mathbf{o}(a)$ `OP` $\mathbf{o}(b)$ `OP` $\mathbf{o}(c) \Rightarrow \mathbf{o}(e)$, then the binarization of this rule leads to replacing it by $\mathbf{o}(a)$ `OP` $\mathbf{o}(b) \Rightarrow \mathbf{o}(ie_1)$ and $\mathbf{o}(ie_1)$ `OP` $\mathbf{o}(c) \Rightarrow \mathbf{o}(e)$, and adding the rule $\mathbf{r}(ie_1) \leftarrow true$, for every $ie_1$ that does not belong to the signature of $P$. Note that, in this case and similarly to what is done in ETALIS, we assume a left-association when no parenthesis are present.

Moreover, this pre-processing via binarization is done without loss of generality since we can show that the program obtained $P'$ is equivalent to the original program $P$. More precisely, we can show that the minimal models that satisfy $P'$ coincide with the minimal models that satisfy $P$:

**Proposition 2** (Binarization equivalence)**.** *Let $P$ be a $\mathcal{TR}^{ev}$ program with a rule of the form:* $\mathbf{o}(e_1)$ `OP` $\mathbf{o}(e_2)$ `OP` $\mathbf{o}(e_3) \Rightarrow \mathbf{o}(e)$ *for any events $e_1$-$e_3$ and operator* `OP`*, and let $P'$ be obtained from $P$ by removing that rule and adding* $\mathbf{o}(e_1)$ `OP` $\mathbf{o}(e_2) \Rightarrow \mathbf{o}(ie_1)$, $\mathbf{o}(ie_1)$ `OP` $\mathbf{o}(e_3) \Rightarrow \mathbf{o}(e)$ *and* $\mathbf{r}(ie_1) \leftarrow true$*. Then:*

$$P \equiv P'$$

*Proof.* See appendix C, page 257. □

Besides binarization, and as it is usual in complex event processing systems, we restrict the use of negation in the program. The major problem with negation is that it is hard to detect the non-presence of an event pattern in an unbounded interval. Because of this, complex event processing systems like [AC06; AFRSSS10] always define negation bounded to the occurrence of two other events as in $\mathbf{not}(\mathbf{o}(e_3))[\mathbf{o}(e_1), \mathbf{o}(e_2)]$. In the latter, the negation pattern holds if $e_3$ does not occur in the interval defined by the occurrence of $e_1$ followed by the occurrence of $e_2$. As shown by theorems 8 and 9 of chapter 10, this pattern can be captured in $\mathcal{TR}^{ev}$ by the formula $\mathbf{o}(e_1) \otimes \neg\mathbf{o}(e_3) \otimes \mathbf{o}(e_2)$, and thus we restrict the use of negation to this pattern, and to the expression `path`. Due to this, and to simplify the syntax, from this point on, we write this pattern using the SNOOP and ETALIS syntax: $\mathbf{not}(\mathbf{o}(e_3))[\mathbf{o}(e_1), \mathbf{o}(e_2)]$.

The detection of event patterns, as it is in ETALIS algorithm, is based on a program transformation that, as we shall see, keeps track of what events have arrived and what still needs to occur for a given pattern to be fired. As such, the program passes through two transformation phases: a first phase where the program is pre-processed so that all its event rules become binary, and a second phase where the program is transformed each time a new atomic event is learned to be true.

On the other hand, the execution of actions is based on $\mathcal{TR}$'s proof-theory [BK93]. As a consequence, this procedure is only defined for a subclass of programs where *transactions* can be expressed as serial-Horn goals, as it happens in $\mathcal{TR}$ procedures like [BK93; GA13a; FK10]. As usual, a serial goal is a transaction formula of the form $a_1 \otimes a_2 \otimes \ldots \otimes a_n$ where each $a_i$ is an atom and $n \geq 0$. When $n = 0$ we write $()$ which denotes the empty goal. Finally, a serial-Horn rule has the form $b \leftarrow a_1 \otimes \ldots \otimes a_n$, where the body $a_1 \otimes \ldots \otimes a_n$ is a serial goal and $b$ is an atom.

It is worth noting that this serial-Horn constraint is only presented at the level of transactions, and in event pattern definitions, one can still use the remaining $\mathcal{TR}^{ev}$ operators, except for negation which, as previously mentioned, is restricted to formulas of the form $\mathbf{not}(e_3)[e_1, e_2]$, and `path`.

The procedure starts with a program $P$, an initial state $D$, a serial goal $e_1 \otimes \ldots \otimes e_k$ (also known as event stream) and iteratively manipulates *resolvents*. A resolvent in the iterative procedure has the form $\pi, ESet \Vdash^{id'}_{P'} L_1 \otimes L_2 \otimes \ldots \otimes L_k$, where $\pi$ is the path obtained by the procedure so far, $ESet$ is the set of events that were previously triggered and still need to be responded to, $L_1 \otimes L_2 \otimes \ldots \otimes L_k$ is the remaining goal that needs to be proven/executed, and $id$ is an auxiliar state count identifier (whose usage is made clear below). Finally, the program $P'$ is the current program, containing all the rules from the original program $P$ plus temporary event rules to help deal with the detection of event patterns. A successful derivation for $P, D- \vdash e_1 \otimes \ldots \otimes e_k$ starts with the resolvent $\langle D \rangle, \emptyset \Vdash^{1}_{P} e_1 \otimes \ldots \otimes e_k$ and non-deterministically applies the derivation rules of definition 61 at each step of the procedure, until it eventually reaches the resolvent $\pi, \emptyset \Vdash^{id}_{P'} ()$. If such a derivation exists, then $P, \pi \vdash e_1 \otimes \ldots \otimes e_k$ holds.

Most of the derivation rules in definition 61 have a direct correspondence with $\mathcal{TR}$'s proof theory (cf. definition 7), but now incorporating the notion of path expansion and event detection. Rule 1 replaces a transaction atom $L$ by the $Body$ of a program rule whose head is $L$; rule 2 deals with a query to the oracle deleting it from the set of goals whenever this query is true in the current state (i.e., the last state of $\pi$); rule 3 executes actions according to the transition oracle definition (i.e., if an action $A$ can be executed in the last state $D_1$ of path $\pi$, reaching the state $D_2$, then we add the path $D_1 \xrightarrow{\mathbf{o}(A)} D_2$ to our current path, but also computes the path *expansion*, specified in definition 62, resulting from answering the events that have directly or indirectly become true because of $\mathbf{o}(A)$); finally, rule 4 deals with the triggering of explicit events – if an event $e$ is explicitly triggered and $D_1$ is the last state of the current path $\pi$, then we add the information that $\mathbf{o}(e)$ occurred in state $D_1$ to our path, and expand it with the KB evolution needed to answer all events that are now true because of $e$'s occurrence.

**Definition 61** (Execution). *A derivation for a serial goal $\phi$ in a program $P$ and state $D$ is a sequence of resolvents starting with $\langle D \rangle, \emptyset \Vdash^{1}_{P} \phi$, and obtained by non-deterministically applying the following rules.*

*Let $\pi, ESet \Vdash^{id}_{P_1} L_1 \otimes L_2 \otimes \ldots \otimes L_k$ be a resolvent. The next resolvent is:*

1. ***Unfolding of Rule:***
   $\pi, ESet \Vdash^{id}_{P_1} Body \otimes L_2 \otimes \ldots \otimes L_k$ *if* $L_1 \leftarrow Body \in P$

2. ***Query:***
   $\pi, ESet \Vdash^{id}_{P_1} L_2 \otimes \ldots \otimes L_k$ *if* $\texttt{last}(\pi) = D_1$ *and* $\mathcal{O}^d(D_1) \models L_1$ *or if* $L_1 = true$

3. ***Update primitive:***
   $\pi', ESet' \Vdash^{id'}_{P_2} L_2 \otimes \ldots \otimes L_k$ *if:*
   - $\texttt{last}(\pi) = D_1$ *and* $\mathcal{O}^t(D_1, D_2) \models L_1$
   - *ExpandPath*$(P_1, L_1, \pi \circ \langle D_1 \xrightarrow{\mathbf{o}(L_1)} D_2 \rangle, ESet, id) = (P_2, \pi', ESet', id')$

4. ***Explicit event request:***
   $\pi', ESet' \Vdash^{id'}_{P_2} L_2 \otimes \ldots \otimes L_k$ *if*
   - $\texttt{last}(\pi) = D_1$ *and* $L_1 \in \mathcal{P}_e$
   - *ExpandPath*$(P_1, L_1, \pi \circ \langle D_1 \xrightarrow{\mathbf{o}(L_1)} D_1 \rangle, ESet, id) = (P_2, \pi', ESet', id')$

*An execution for $\phi$ in program $P$, and state $D$ is* successful *if it ends in a resolvent of the form $\pi, \emptyset \Vdash^{id'}_{P'} ()$. In this case we write $P, \pi \vdash \phi$.*

The $ExpandPath$ function, which appears in the previous definition, is called whenever an event $A$ (either an explicit event or a primitive action execution) occurs. Then, $ExpandPath$ is responsible for expanding the current path with the response of the event made true. Moreover, given the event rules in the program, other event occurrences may become true, and in that case, these events must also be responded by this function. The set of events that become true due to the occurrence of $A$ are computed by the $Closure$ function (defined below in definition 64). The $ExpandPath$ computation only stops when no more unanswered events exist.

**Definition 62** (Expand Path)**.**
*Input: program $P$, primitive $A$, path $\pi$, event set $ESet$, $id$*
*Output: program $P'$, path $\pi'$, event set $ESet'$, $id'$*

*Define* $ESet' := ESet \cup \{\mathbf{o}(A)\{id, id+1\}\}$, $\pi' := \pi$, $id' := id + 1$, $P' := P$
***while*** $\texttt{needResponse}(ESet', id, id') \neq \emptyset$ ***{***

1. *Let* $Closure(ESet', P', id') = (ESet_1, P_1)$
2. *Let* $FirstInOrder(\texttt{needResponse}(ESet_1, id, id')) = \mathbf{o}(e)$
3. *Let $D$ be the final state of $\pi'$, and a derivation starting in* $\langle D \rangle, \emptyset \Vdash^{id'}_{P_1} \mathbf{r}(e)$ *and ending in* $\pi_i, ESet_i \Vdash^{id_i}_{P_i} ()$
4. *Define* $\pi' := \pi' \circ \pi_i$, $id' := id_i$, $P' := P_i$, $ESet' := (ESet_1 \cup ESet_i) \setminus \{\mathbf{o}(e)\}$

*}*
*If* $\texttt{needResponse}(ESet', id, id') = \emptyset$ *then the computation is said to be successful.*
*In this case* ***return*** $(P', \pi', ESet', id')$

Let us examine closer the details of this $ExpandPath$ function. The $ESet'$ variable contains, at each moment, the set of events that have happened during the execution. Each event in this set is associated with a pair $\{id_i, id_f\}$ specifying the exact interval where the event happened. Afterwards, based on these $id$s, one can recast the path of occurrence. More precisely, if $e\{id_1, id_2\} \in ESet$, and $\pi = \langle D_1 \, {}^{O_1}\!\!\rightarrow \ldots {}^{O_{k-1}}\!\!\rightarrow D_k\rangle$ is the current path, then event $e$ is said to occur on $\pi_{<id_i,id_f>}$, where $\pi_{<id_i,id_f>}$ is the path obtained from $\pi$ by trimming it from state $D_{id_i}$ to state $D_{id_f}$: $\langle D_{id_i} \, {}^{O_{id_i}}\!\!\rightarrow \ldots {}^{O_{id_f-1}}\!\!\rightarrow D_{id_f}\rangle$.

At each iteration, the $ExpandPath$ function collects all the events in $ESet'$ that still need to be responded to w.r.t. that iteration. In other words, the function obtains the events whose occurrence holds on a path starting after the initial state of the function call. This is denoted in $\texttt{needResponse}(ESet, id_i, id_j)$ which is defined as:

**Definition 63** ($\texttt{needResponse}(ESet, id_i, id_j)$)**.** *Let* $id_1, id_2, id_i$ *be identifiers and* $ESet$ *a set of events of the form* $e\{id_1, id_2\}$*, where* $id_1$ *and* $id_2$ *define the starting and ending of* $e$*, respectively.* $\texttt{needResponse}(ESet, id_i, id_j)$ *is the subset of* $ESet$ *s.t.* $id_i \leq id_1 \leq id_2 \leq id_j$*.*

Then, the $FirstInOrder$ function simply sorts the events according to the chosen order function, according to the semantics (cf. example. 36), and returns the first event w.r.t. that order:

**Example 39** (Order function - Priority List)**.** $FirstInOrder(Set) = e$ *such that* $e \in Set \wedge \forall e'$ *where* $e' \in Set \wedge Priority(e) \geq Priority(e')\}$

Finally, the $Closure$ computation is crucial for this procedure as it is responsible to detect event patterns. Given a pre-processed program where all event rules are binary, the $Closure$ computation matches event occurrences with the bodies of event rules, produces new temporary rules containing information about what events still need to occur to trigger an event pattern, and returns a new set of (complex) events that have become true. During this computation, the event component of the program suffers constant transformations and becomes partitioned into *permanent rules* and *temporary rules*.

Permanent rules have the form $body \Rightarrow e$ and come from pre-processing the original program. They can never expire (i.e., they are never deleted during the computation) and are always available for activation.

Temporary rules have the form $body \; \texttt{OP} \overset{id_1}{\underset{id_2}{\Rightarrow}} e$ and arise from *partially* satisfying a permanent rule. For instance, if we have the (permanent) rule $\mathbf{o}(e_1) \wedge \mathbf{o}(e_2) \Rightarrow \mathbf{o}(e)$ in the program, and $e_1$ is learned to occur in the interval labeled with $\{id_i, id_f\}$, then a temporary rule $\mathbf{o}(e_2) \wedge \overset{id_i}{\underset{id_f}{\Rightarrow}} e$ is added to the program, stating that we are waiting for $e_2$ to occur to fire event $e$.

Subsequently, temporary rules are valid only for some particular iterations of the procedure (unless, as we shall see, their $id$s are open). Then, they are either deleted (in case

they are expired without being satisfied) or transformed (if they are partially satisfied). We say that temporary rules are expired if the difference between the current global $id$ and the rule's ending $id$ is greater than 1. Moreover, temporary rules also have the information about the operator `OP` which specifies what constraints should be checked in order to satisfy an event pattern.

The Closure computation determines the set of events that are true at a given iteration $id$ of the procedure. At each step of its computation, we try to match the events appearing in the bodies of event rules with the set of events that are known to have occurred (i.e., events belonging to $ESet$). Whenever we can make such a match for a given event $e$, one of the two things happens: 1) if the body of that rule has more events that need to be triggered, then we add the information that $e$ occurred at this iteration by adding a temporary rule to $P$, saying what still needs to occur for that rule to be triggered; 2) if $e$ was the only event in the body of that rule, then we add the head's rule to the $ESet$ (cf. Base Cases of definition 64). This computation continues until we reach a fixed point, i.e., until nothing more is added to the $ESet$ or no (temporary) rules are added to the program.

Based on this, we define the Closure computation as follows:

**Definition 64** (Closure)**.**
*Input:* $ESet, P, id$
*Output:* $ESet', P'$

***repeat*** *{*
*Define* $ESet' = ESet$, $P' = P$
*For each* $\mathbf{o}(e)\{id_i, id_f\} \in ESet$*:*

**Base Cases:**

1. ***If*** $\mathbf{o}(e) \Rightarrow head \in P'$ ***then***
   $ESet := ESet \cup \{head\{id_i, id_f\}\}$
2. ***If*** $\mathbf{o}(e) \otimes^{id_1}_{id_2} head \in P'$ *and* $id_i = id_2$ ***then***
   $ESet := ESet \cup \{head\{id_1, id_f\}\}$ *and* $P' := P' \setminus \{\mathbf{o}(e) \otimes^{id_1}_{id_2} head\}$
3. ***If*** $\mathbf{o}(e) \wedge^{id_1}_{id_2} head \in P'$, $id_1 = id_i, id_2 = id_f$ ***then***
   $ESet := ESet \cup \{head\{id_i, id_f\}\}$ *and* $P' := P' \setminus \{\mathbf{o}(e) \wedge^{id_1}_{id_2} head\}$
4. $\mathbf{o}(e) \otimes^{*}_{*} head \in P'$ ***then***
   $ESet := ESet \cup \{head\{*id_i, id_f\}\}$ *and* $P' := P' \setminus \{\mathbf{o}(e) \otimes^{*}_{*} head\}$

**Operations Cases:**

1. ***If*** $\mathbf{o}(e) \otimes \mathbf{o}(e_1) \Rightarrow head \in P'$ ***then***
   $P' := P' \cup \{\mathbf{o}(e_1) \otimes^{id_i}_{id_f} head\}$

2. ***If*** $\mathbf{o}(e) \wedge \mathbf{o}(e_1) \Rightarrow head \in P'$ ***then***
   $P' := P' \cup \{\mathbf{o}(e_1) \underset{id_f}{\overset{id_i}{\wedge\Rightarrow}} head\}$
3. ***If*** $\mathbf{o}(e)\ ;\ \mathbf{o}(e_1) \Rightarrow head \in P'$ ***then***
   $P' := P' \cup \{\mathbf{o}(e_1) \underset{id_f^*}{\overset{id_i}{\otimes\Rightarrow}} head\}$

**Negation Case:**

1. ***If*** $\mathbf{not}(\mathbf{o}(e_3))[\mathbf{o}(e), \mathbf{o}(e_2)] \Rightarrow head \in P'$ ***then***
   $P' := P' \cup \{\mathbf{o}(e_2) \underset{id_f^*}{\overset{id_i}{\otimes\Rightarrow}} head, \mathbf{o}(e_3) \underset{id_f^*}{\overset{id_i}{\neg\Rightarrow}} (\mathbf{o}(e_2) \underset{id_f^*}{\overset{id_i}{\otimes\Rightarrow}} head)\}$

**Path Cases:**

1. ***If*** $\texttt{path} \otimes \mathbf{o}(e_1) \Rightarrow head \in P'$ ***then***
   $P' := P' \cup \{\mathbf{o}(e_1) \underset{*}{\overset{*}{\otimes\Rightarrow}} head\}$
2. ***If*** $\texttt{path} \underset{id_2}{\overset{id_1}{\otimes\Rightarrow}} head \in P'$ ***then***
   $ESet := ESet \cup \{head\{id_1, id_2^*\}\}$ *and* $P' := P' \setminus \{\texttt{path} \underset{id_2}{\overset{id_1}{\otimes\Rightarrow}} head\}$

} ***until*** $ESet = ESet'$
*For each:*
$rule_j \underset{id_2*}{\overset{id_1}{\neg\Rightarrow}} \mathbf{o}(e) \in P'$ *and* $\mathbf{o}(e)\{id_i, id_f\} \in ESet$ ***do***
$P' = P' \setminus \{rule_j \underset{id_2*}{\overset{id_1}{\neg\Rightarrow}} \mathbf{o}(e), rule_j\}$
***Return*** $ESet', P'$

Regarding the previous definition, some words are in order concerning the detection of event patterns that use the expression `path`, and negation (defined by the Path and Negation Case above, respectively). Recall from chapter 10 that the expression `path` is instrumental to define standard complex event operators. More precisely, `path` is very useful to relax (or expand) the interval where a given event pattern holds. In this sense, $\mathbf{o}(e_1) \otimes \texttt{path} \otimes \mathbf{o}(e_2)$ (normally written as $\mathbf{o}(e_1); \mathbf{o}(e_2)$) is satisfied in the same intervals as $\mathbf{o}(e_1) \otimes \mathbf{o}(e_2)$, and also when $e_2$ occurs with a starting $id$ that is equal or greater the ending of $e_1$. Note that to make $\mathbf{o}(e_1) \otimes \mathbf{o}(e_2)$ true, the starting $id$ of $e_2$ must be equal to the ending $id$ of $e_1$.

As another example, in rule $\mathbf{o}(a.ins) \otimes \texttt{path} \Rightarrow \mathbf{o}(e)$, the usage of `path` makes $\mathbf{o}(e)$ true in all paths $\langle D_1 \xrightarrow{\mathbf{o}(a.ins)} D_2 \rangle$ where $\mathbf{o}(a.ins)$ holds, but also in paths that can be constructed by expanding $\langle D_1 \xrightarrow{\mathbf{o}(a.ins)} D_2 \rangle$ to the right, (e.g. as $\langle D_1 \xrightarrow{\mathbf{o}(a.ins)} D_2 \xrightarrow{\mathbf{o}(b.ins)} D_3 \xrightarrow{\mathbf{o}(c.ins)} D_4 \rangle$). Conversely, $\texttt{path} \otimes \mathbf{o}(a.ins) \Rightarrow \mathbf{o}(e)$ makes $\mathbf{o}(e)$ true in all paths that can be constructed by expanding $\langle D_1 \xrightarrow{\mathbf{o}(a.ins)} D_2 \rangle$ to the left. To cope with this behavior, the procedure deals with three *open ids*: $^*id$, $id^*$ and $*$. Here, $*$ states that the right or left interval of an event pattern occurrence is unknown, while $^*id$ and $id^*$ state that the starting (respectively ending) point of an event is any point before (respectively after) or equal to $id$. Since these markers can propagate to several events, we have to define

comparisons operations for these ids, and say that $\forall id.\ id = *$, $id = {}^{*}id_1$ if $id \leq {}^{*}id_1$ and finally, $id = id_1^*$ if $id \geq id_1$.

Regarding negation, and as it was previously mentioned, it can only appear in the body of an event pattern rule as: $\mathbf{not}(\mathbf{o}(e_3))[\mathbf{o}(e_1), \mathbf{o}(e_2)] \Rightarrow head$, which is syntactic sugar for $\mathbf{o}(e_1) \otimes \neg\mathbf{o}(e_3) \otimes \mathbf{o}(e_2) \Rightarrow head$. Such an event pattern is said to start with the occurrence of $\mathbf{o}(e_1)\{id_i, id_f\}$ in $ESet$. Then, when $e_1$ happens, we add two different rules to the program: $\mathbf{o}(e_2) \overset{id_i}{\underset{id_f^*}{\otimes\!\Rightarrow}} head$ and $\mathbf{o}(e_3) \overset{id_i}{\underset{id_f^*}{\neg\!\Rightarrow}} (\mathbf{o}(e_2) \overset{id_i}{\underset{id_f^*}{\otimes\!\Rightarrow}} head)$. The former rule says that the not-event becomes true when $\mathbf{o}(e_2)$ appears in the $ESet$. The latter rule determines that, in case $\mathbf{o}(e_3)$ appears in the $ESet$ (before $\mathbf{o}(e_2)$), then the former temporary rule is removed from the program, so that a later occurrence of $e_2$ no longer makes the not-event true. One important detail is that the removal of temporary rules that arise from such negation patterns, is performed after the fixed point is achieved, separating the monotonic construction of the $ESet$, from the non-monotonic behavior of the rule $\neg\!\Rightarrow$.

**Theorem 11** (Soundness and Completeness of the Procedure)**.** *Let $P$ be a program, $\pi$ a path, and $\phi$ a transaction formula.*

$$P, \pi \models \phi \textit{ iff } P, \pi \vdash \phi$$

*Proof.* See appendix C.3 from page 261 onwards. And proof C.3.3 on page 271. □

To better illustrate of how $\mathcal{TR}^{ev}$'s procedure behaves, consider the following example, where we exhaustively detail, step by step, the execution of the procedure, until it reaches the desired solution path.

**Example 40** (Proof Procedure)**.** *Recall example 33 which is composed by a program $P$ with the following rules:*

$$\begin{aligned}
&p \leftarrow a.ins \\
&q \leftarrow b.ins \\
&\mathbf{r}(e_x) \leftarrow p \otimes q \\
&\mathbf{r}(e_1) \leftarrow d.ins \qquad\qquad \mathbf{o}(a.ins); \mathbf{o}(b.ins) \Rightarrow \mathbf{o}(e_1) \\
&\mathbf{r}(a.ins) \leftarrow c.ins \\
&\mathbf{r}(b.ins) \leftarrow true \\
&\mathbf{r}(c.ins) \leftarrow true \\
&\mathbf{r}(d.ins) \leftarrow true
\end{aligned}$$

*where we explicitly add the rules $\mathbf{r}(b.ins) \leftarrow true$, $\mathbf{r}(c.ins) \leftarrow true$ and $\mathbf{r}(d.ins) \leftarrow true$ to the program, as defined in section 11.2.3, where we assume a rule $\mathbf{r}(primitive) \leftarrow true$, to every $primitive$ of the oracle defined in the program's signature.*

*In program $P$, for any interpretation $M$ the following is true (cf. example 33):*

$$M, \langle \{\} \overset{\mathbf{o}(e_x)}{\rightarrow} \{\} \overset{\mathbf{o}(a.ins)}{\rightarrow} \{a\} \overset{\mathbf{o}(c.ins)}{\rightarrow} \{a, c\} \overset{\mathbf{o}(b.ins)}{\rightarrow} \{a, b, c\} \overset{\mathbf{o}(d.ins)}{\rightarrow} \{a, b, c, d\} \rangle \models e_x$$

*In the following we show that the procedure is able to compute the same path for the event request $e_x$ and starting in path $\langle\{\}\rangle$.*

*The procedure starts with the resolvent:*

$$\langle\{\}\rangle, \emptyset \Vdash^1_P e_x$$

*Since $e_x \in \mathcal{P}_e$, then the only rule that can be applied from definition 61 is item 4, where we call the Expand Path definition to compute the next resolvent:*

$$\textbf{\textit{Call:}}\ ExpandPath(P, e_1, \langle\{\}\rangle, \emptyset, 1)$$

*For that we start by defining:*

$$ESet' = \emptyset \cup \{\mathbf{o}(e_x)\{1,2\}\},\ \pi' = \langle\{\}\rangle,\ id' = 2 \text{ and } P' = P$$

*Since* $\texttt{needResponse}(\{\mathbf{o}(e_x)\{1,2\}\}, 1, 2) \neq \emptyset$ *(as $\mathbf{o}(e_x)$ still needs to be responded to) we start iterating the **while** cycle:*

**iteration 1.** *Initial Scope $id$ = 1*

1.a) *As the first step, we start computing the Closure$(\{\mathbf{o}(e_x)\{1,2\}\}, P', 2)$. Since $\mathbf{o}(e_x)$ is not in the body of any complex event rule, then the computation returns $ESet_1 = \{\mathbf{o}(e_x)\{1,2\}\}$ and $P' = P'$.*

1.b) *For step two, we call $FirstInOrder(\texttt{needResponse}(ESet_1, 1, 2))$. Since the only event in the $ESet_1$ is $\mathbf{o}(e_x)\{1,2\}$, $FirstInOrder(\texttt{needResponse}(ESet_1, 1, 2)) = \mathbf{o}(e_x)$.*

1.c) *For step three we start a derivation for $\mathbf{r}(e_x)$. Since $\langle\{\}\rangle$ is the last (and only) state of $\pi'$, the derivation starts in the following resolvent:*

$$\langle\{\}\rangle, \emptyset \Vdash^2_{P'} \mathbf{r}(e_x)$$

*From this we can apply the unfolding step (item 1 of definition 61) and obtain the resolvent:*

$$\langle\{\}\rangle, \emptyset \Vdash^2_{P'} p \otimes q$$

*Similarly, we can again apply the unfolding step (item 1 of definition 61) and obtain the resolvent:*

$$\langle\{\}\rangle, \emptyset \Vdash^2_{P'} a.ins \otimes q$$

*Then, since $a.ins$ is defined in the oracle we apply item 3 of definition 61. Since $\mathcal{O}^t(\{\}, \{a\}) \models a.ins$, we call $ExpandPath$ for the action $a.ins$ and the new path:*

$$\textbf{\textit{Call:}}\ ExpandPath(P', a.ins, \langle\{\} \overset{\mathbf{o}(a.ins)}{\rightarrow} \{a\}\rangle, \emptyset, 2)$$

*(note that we are still inside an $ExpandPath$ call).*

*For that we start by defining:*

$$ESet' = \emptyset \cup \{\mathbf{o}(a.ins)\{2,3\}\},\ \pi' = \langle\{\}^{\mathbf{o}(a.ins)}\!\!\rightarrow\{a\}\rangle,\ id' = 3 \text{ and } P' = P'$$

*Afterwards, since* $\texttt{needResponse}(\{\mathbf{o}(a.ins)\{2,3\}\},2,3) \neq \emptyset$ *we start iterating the* ***while*** *cycle:*

**iteration 1.1.** *Initial Scope* $id$ *= 2*

*1.1.a) As the first step, we start computing the Closure*$(\{\mathbf{o}(a.ins)\{2,3\}\},P',3)$. *Since* $\mathbf{o}(a.ins)$ *is in the body of the rule:* $\mathbf{o}(a.ins);\mathbf{o}(b.ins) \Rightarrow \mathbf{o}(e_1)$ *we apply the* ***Operation Case*** *item 3. and transform* $P'$ *into* $P_1 = P' \cup \{\mathbf{o}(b.ins) \otimes\overset{2}{\underset{3*}{\Rightarrow}} \mathbf{o}(e_1)\}$ *Then, since there is no more event in the* $ESet'$*, then the computation stops, returning the new* $P_1$ *which has now a new rule, and the set* $ESet'$ *which is unchanged.*

*1.1.b) For step two, we call* $FirstInOrder(\texttt{needResponse}(ESet',id,id'))$. *Since the only event in the* $ESet'$ *is* $\mathbf{o}(a.ins)\{2,3\}$*, then:*

$$FirstInOrder(\texttt{needResponse}(ESet',2,3)) = \mathbf{o}(a.ins)$$

*1.1.c) For step three we start a derivation for* $\mathbf{r}(a.ins)$. *Since* $\langle\{a\}\rangle$ *is the last state of* $\pi'$*, then the derivation starts in the resolvent:*

$$\langle\{a\}\rangle,\emptyset \Vdash^{3}_{P_1} \mathbf{r}(a.ins)$$

*From this we can apply the unfolding step (item 1 of definition 61) and obtain:*

$$\langle\{a\}\rangle,\emptyset \Vdash^{3}_{P_1} c.ins$$

*Then, since* $c.ins$ *is defined in the oracle we apply item 3 of definition 61. Since* $\mathcal{O}^t(\{a\},\{a,c\}) \models c.ins$*, we call* $ExpandPath$ *for* $c.ins$ *and for this new path:*

***Call:*** $ExpandPath(P_1,c.ins,\langle\{a\}^{\mathbf{o}(c.ins)}\!\!\rightarrow\{a,c\}\rangle,\emptyset,3)$

*(note that we are still inside two* $ExpandPath$ *calls).*
*We start by setting:*

$$ESet' = \emptyset \cup \{\mathbf{o}(c.ins)\{3,4\}\},\ \pi' = \langle\{a\}^{\mathbf{o}(c.ins)}\!\!\rightarrow\{a,c\}\rangle,\ id' = 4 \text{ and } P' = P_1$$

*Since* $\texttt{needResponse}(ESet',3,4) \neq \emptyset$ *(because* $\mathbf{o}(c.ins)$ *is in the* $ESet'$*) we start iterating the* ***while*** *cycle:*

**iteration 1.1.1.** *Initial Scope* $id$ *= 3*

*1.1.1.a) As the first step, we start computing the Closure*$(ESet',P',4)$ *Since the only event in* $ESet'$ *is* $\mathbf{o}(c.ins)\{3,4\}$ *which is not in the body of any event rule, then this computation returns trivially without changing the* $ESet'$ *and the program* $P'$.

*1.1.1.b) For step two, we call* $FirstInOrder(\texttt{needResponse}(ESet',id,id'))$.

*Since the only event in the $ESet'$ is $\mathbf{o}(c.ins)\{3,4\}$, then:*

$$FirstInOrder(\texttt{needResponse}(ESet', 3, 4)) = \mathbf{o}(c.ins)$$

*1.1.1.c) In step three, we start a new derivation for $\mathbf{r}(c.ins)$. Since $\langle\{a, c\}\rangle$ is the last state of $\pi'$, this derivation starts in the resolvent:*

$$\langle\{a, c\}\rangle, \emptyset \Vdash^4_{P'} \mathbf{r}(c.ins)$$

*From this we can apply the unfolding step (item 1 of definition 61) and obtain:*

$$\langle\{a, c\}\rangle, \emptyset \Vdash^4_{P'} true$$

*Then, we can apply the query step (item 2 of definition 61) and obtain:*

$$\langle\{a, c\}\rangle, \emptyset \Vdash^4_{P'} ()$$

*We can now stop this derivation, and consider it successful, where $P, \langle\{a, c\}\rangle \vdash \mathbf{r}(c.ins)$*

*1.1.1.d) We end this iteration by setting $\pi' = \langle\{a\}^{\mathbf{o}(c.ins)}\rightarrow \{a, c\}\rangle$, $id' = 4$, $P' = P'$, $ESet' = (ESet' \cup \emptyset) \setminus \{\mathbf{o}(c.ins)\} = \emptyset$*

*Now we go back to **iteration 1.1.1**.*
*Since $ESet' = \emptyset$, then $\texttt{needResponse}(ESet', 3, 4) = \emptyset$ we succeed the **while** cycle and return: $(P', \langle\{a\}^{\mathbf{o}(c.ins)}\rightarrow \{a, c\}\rangle, \emptyset, 4)$*

*Then we return to item 1.1.c). where*

$$ExpandPath(P', c.ins, \langle\{a\}^{\mathbf{o}(c.ins)}\rightarrow \{a, c\}\rangle, \emptyset, 3) = (P', \langle\{a\}^{\mathbf{o}(c.ins)}\rightarrow \{a, c\}\rangle \circ \langle\{a, c\}\rangle, \emptyset, 4)$$

*and we get the resolvent:*

$$\langle\{a\}^{\mathbf{o}(c.ins)}\rightarrow \{a, c\}\rangle, \emptyset \Vdash^4_{P'} ()$$

*Again we can now stop this derivation, and consider it successful, where $P, \langle\{a\}^{\mathbf{o}(c.ins)}\rightarrow \{a, c\} \vdash c.ins$*

*1.1.d) We end this iteration by setting $\pi' = \langle\{a\}^{\mathbf{o}(c.ins)}\rightarrow \{a, c\}\rangle$, $id' = 4$, $P' = P'$, $ESet' = (ESet' \cup \emptyset) \setminus \{\mathbf{o}(a.ins)\} = \emptyset$*

*I Since $ESet' = \emptyset$, then $\texttt{needResponse}(ESet', 2, 4) = \emptyset$ we succeed the **while** cycle and return: $(P_1, \langle\{\}^{\mathbf{o}(a.ins)}\rightarrow \{a\}^{\mathbf{o}(c.ins)}\rightarrow \{a, c\}\rangle, \emptyset, 4)$*

*Then we return to item 1.c). where*

$$ExpandPath(P', a.ins, \langle\{\} \overset{\mathbf{o}(a.ins)}{\rightarrow} \{a\}\rangle, \emptyset, 2) = \\ (P_1, \langle\{\} \overset{\mathbf{o}(a.ins)}{\rightarrow} \{a\} \overset{\mathbf{o}(c.ins)}{\rightarrow} \{a, c\}\rangle, \emptyset, 4)$$

*and from this we obtain the non-complete resolvent:*

$$\langle\{\} \overset{\mathbf{o}(a.ins)}{\rightarrow} \{a\} \overset{\mathbf{o}(c.ins)}{\rightarrow} \{a, c\}\rangle, \emptyset \Vdash^{4}_{P_1} q$$

*Subsequently, we can again apply the unfolding step (item 1 of definition 61) and obtain the resolvent:*

$$\langle\{\} \overset{\mathbf{o}(a.ins)}{\rightarrow} \{a\} \overset{\mathbf{o}(c.ins)}{\rightarrow} \{a, c\}\rangle, \emptyset \Vdash^{4}_{P_1} b.ins$$

*Then, since $b.ins$ is defined in the oracle, we apply item 3 of definition 61. Since $\mathcal{O}^t(\{a, c\}, \{a, b, c\}) \models b.ins$, we call $ExpandPath$ for $b.ins$ and for this new path:*

***Call:***

$$ExpandPath(P_1, b.ins, \langle\{\} \overset{\mathbf{o}(a.ins)}{\rightarrow} \{a\} \overset{\mathbf{o}(c.ins)}{\rightarrow} \{a, c\} \overset{\mathbf{o}(b.ins)}{\rightarrow} \{a, b, c\}\rangle, \emptyset, 4)$$

*(note that we are still inside one $ExpandPath$ calls).*

*We start by setting:*

$$ESet' = \emptyset \cup \{\mathbf{o}(b.ins)\{4, 5\}\},$$
$$\pi' = \langle\{\} \overset{\mathbf{o}(a.ins)}{\rightarrow} \{a\} \overset{\mathbf{o}(c.ins)}{\rightarrow} \{a, c\} \overset{\mathbf{o}(b.ins)}{\rightarrow} \{a, b, c\}\rangle, id' = 5 \text{ and } P' = P_1$$

*Then, since* $\texttt{needResponse}(ESet', 4, 5) \neq \emptyset$ *(as $\mathbf{o}(b.ins)$ still needs to be responded to) we start iterating the **while** cycle again:*

**iteration 1.2** *Initial Scope $id$ = 4*

*1.2a) As the first step, we start computing the Closure($\{\mathbf{o}(b.ins)\{4, 5\}\}, P', 5$). Recall that $P'$ has now a temporary rule $\mathbf{o}(b.ins) \otimes \overset{2}{\underset{3*}{\Rightarrow}} \mathbf{o}(e_1)$. By definition $3* = 4$ is true since $3*$ is an open id. As such, we add $\mathbf{o}(e_1)$ to the $ESet'$, i.e.*

$$ESet' = \{\mathbf{o}(b.ins)\{4, 5\}, \mathbf{o}(e_1)\{2, 5\}\}$$

*and delete the previous temporary rule from $P'$ (which is now equal to the original program $P$).*

*Then, since $e_1$ is not in the body of any event rule, the computation stops, returning a new $P_2$ which does not have temporary rules, and the new set $ESet'$.*

*1.2b) For step two, we call $FirstInOrder(\texttt{needResponse}(ESet', 4, 5))$. Now we have two events in the set, but only one can be responded at this time, because we are in the scope $(4, 5)$ and the event $e_1$ is in scope $(2, 5)$*

*Thus:*

$$FirstInOrder(\texttt{needResponse}(ESet', 4, 5)) = \mathbf{o}(b.ins)$$

*1.2c) For step three we start a new derivation for $\mathbf{r}(b.ins)$. Since $\langle\{a, b, c\}\rangle$ is the last state of $\pi'$, this derivation starts in the resolvent:*

$$\langle\{a, b, c\}\rangle, \emptyset \Vdash^{5}_{P_2} \mathbf{r}(b.ins)$$

*From this we can apply the unfolding step (item 1 of definition 61) and obtain:*

$$\langle\{a, b, c\}\rangle, \emptyset \Vdash^{5}_{P_2} true$$

*Then, we can apply the query step (item 2 of definition 61) and obtain:*

$$\langle\{a, b, c\}\rangle, \emptyset \Vdash^{5}_{P_2} ()$$

*We can now stop this derivation, and consider it successful, where $P, \langle\{a, b, c\}\rangle \vdash \mathbf{r}(b.ins)$*

*1.2d) We end this iteration by setting $\pi' = \langle\{\}^{\mathbf{o}(a.ins)}\!\!\rightarrow\{a\}^{\mathbf{o}(c.ins)}\!\!\rightarrow\{a, c\}^{\mathbf{o}(b.ins)}\!\!\rightarrow \{a, b, c\}\rangle$, $id' = 5$, $P' = P_2$, $ESet' = ESet' \setminus \{\mathbf{o}(b.ins)\} = \{\mathbf{o}(e_1)\}$*

*Now we go back to **iteration 1.2** and notice that $e_1$ is still in the $ESet$, but $\texttt{needResponse}(ESet', 4, 5) = \emptyset$ (because the event started before state id 4) Then we end this iteration and return: $(P_2, \langle\{\}^{\mathbf{o}(a.ins)}\!\!\rightarrow\{a\}^{\mathbf{o}(c.ins)}\!\!\rightarrow\{a, c\}^{\mathbf{o}(b.ins)}\!\!\rightarrow \{a, b, c\}\rangle, \{\mathbf{o}(e_1)\{2, 5\}\}, 5)$*

*From this we come back to iteration 1.c) ending the call:*

$$ExpandPath(P_1, b.ins, \langle\{\}^{\mathbf{o}(a.ins)}\!\!\rightarrow\{a\}^{\mathbf{o}(c.ins)}\!\!\rightarrow\{a, c\}^{\mathbf{o}(b.ins)}\!\!\rightarrow\{a, b, c\}\rangle, \emptyset, 4) = (P_2, \langle\{\}^{\mathbf{o}(a.ins)}\!\!\rightarrow\{a\}^{\mathbf{o}(c.ins)}\!\!\rightarrow\{a, c\}^{\mathbf{o}(b.ins)}\!\!\rightarrow\{a, b, c\}\rangle, \{\mathbf{o}(e_1)\{2, 5\}\}, 5)$$

*1.d) We end this iteration by setting $\pi' = \langle\{\}^{\mathbf{o}(a.ins)}\rightarrow \{a\}^{\mathbf{o}(c.ins)}\rightarrow \{a, c\}^{\mathbf{o}(b.ins)}\rightarrow \{a, b, c\}\rangle$, $id' = 5$, $P' = P_2$, $ESet' = \{\mathbf{o}(e_1)\}$*

*We end iteration 1. and go back to the **while** cycle where the initial $id = 1$.*

*Notice that $\texttt{needResponse}(ESet', 1, 5) \neq \emptyset$ as we now need to respond to $e_1$. Consequently, we iterate the while again.*

**Iteration 2.** *Initial Scope $id$ = 1*

*2.a) As the first step, we start computing the Closure$(\{\mathbf{o}(e_1)\{1, 5\}\}, P', 5)$. Since $e_1$ is not in the body of any complex event rule, then the computation ends trivially without changing $ESet'$ nor $P'$.*

*2.b) For step two, we call* $FirstInOrder(\texttt{needResponse}(ESet', 2, 5))$. *Since the only event in the* $ESet'$ *is* $\mathbf{o}(e_1)\{2, 5\}$, *then:*

$$FirstInOrder(\texttt{needResponse}(ESet', 1, 5)) = \mathbf{o}(e_1)$$

*2.c) For step three we start a new derivation for* $\mathbf{r}(e_1)$. *Since* $\langle\{a, b, c\}\rangle$ *is the last state of* $\pi'$, *this derivation starts in the resolvent:*

$$\langle\{a, b, c\}\rangle, \emptyset \Vdash^{5}_{P'} \mathbf{r}(e_1)$$

*From this we can apply the unfolding step (item 1 of definition 61) and obtain:*

$$\langle\{a, b, c\}\rangle, \emptyset \Vdash^{5}_{P'} d.ins$$

*Then since* $d.ins$ *is defined in the oracle, we apply item 3 of definition 61. Since* $\mathcal{O}^t(\{a, b, c\}, \{a, b, c, d\}) \models d.ins$, *we call* $ExpandPath$ *for* $d.ins$ *and this new path.*

$$\textbf{\textit{Call:}}\ ExpandPath(P', d.ins, \langle\{a, b, c\} \xrightarrow{\mathbf{o}(d.ins)} \{a, b, c, d\}\rangle, \emptyset, 5)$$

*(note that we are still inside one* $ExpandPath$ *call).*
*We start by setting:*

$$ESet' = \emptyset \cup \{\mathbf{o}(d.ins)\{5, 6\}\},\ \pi' = \langle\{a, b, c\} \xrightarrow{\mathbf{o}(d.ins)} \{a, b, c, d\}\rangle,\ id' = 5 \text{ and } P' = P'$$

*Then since* $\texttt{needResponse}(ESet', 5, 6) \neq \emptyset$ *(as* $\mathbf{o}(d.ins)$ *needs to be addressed) we start iterating the* ***while*** *cycle:*

**iteration 2.1** *Initial Scope* $id$ *= 5*

*2.1a) As the first step, we start computing the* $Closure(ESet', P', 6)$ *Since the only event in* $ESet'$ *is* $\mathbf{o}(d.ins)\{5, 6\}$ *which is not in the body of any event rule, then this computation returns trivially without changing the* $ESet'$ *and the program* $P'$.

*2.1b) For step two, we call* $FirstInOrder(\texttt{needResponse}(ESet', id, id'))$. *Since the only event in the* $ESet'$ *is* $\mathbf{o}(d.ins)\{5, 6\}$, *then:*

$$FirstInOrder(\texttt{needResponse}(ESet', 5, 6)) = \mathbf{o}(d.ins)$$

*2.1c) For step three we start a new derivation for* $\mathbf{r}(d.ins)$. *Since* $\langle\{a, b, c, d\}\rangle$ *is the last state of* $\pi'$, *this derivation starts in the resolvent:*

$$\langle\{a, b, c, d\}\rangle, \emptyset \Vdash^{6}_{P'} \mathbf{r}(d.ins)$$

*From this we can apply the unfolding step (item 1 of definition 61) and obtain:*

$$\langle\{a, b, c, d\}\rangle, \emptyset \Vdash^{6}_{P'} true$$

*Then, we can apply the query step (item 2 of definition 61) and obtain:*

$$\langle\{a,b,c,d\}\rangle, \emptyset \Vdash^{6}_{P'} ()$$

*We can now stop this derivation, and consider it successful, where* $P, \langle\{a,b,c,d\}\rangle \vdash \mathbf{r}(d.ins)$

*2.1d) We end this iteration by setting* $\pi' = \langle\{a,b,c\}^{\mathbf{o}(d.ins)}\rightarrow\{a,,b,c,d\}\rangle$, $id' = 6$, $P' = P'$, $ESet' = (ESet' \cup \emptyset) \setminus \{\mathbf{o}(d.ins)\} = \emptyset$

*We go back to* ***iteration 2.1****, where now* $\texttt{needResponse}(ESet', 5, 6) = \emptyset$ *because* $ESet' = \emptyset$*. With this we return the computation.*

*We come back to iteration 2.c) ending the call:*

$$ExpandPath(P', d.ins, \langle\{a,b,c,\}^{\mathbf{o}(d.ins)}\rightarrow\{a,b,c,d\}\rangle, \emptyset, 5) = (P', \langle\{a,b,c,\}^{\mathbf{o}(d.ins)}\rightarrow\{a,b,c,d\}\rangle, \emptyset, 6)$$

*And obtaining the resolvent:*

$$\langle\{a,b,c\}^{\mathbf{o}(d.ins)}\rightarrow\{a,b,c,d\}\rangle, \emptyset \Vdash^{6}_{P'} ()$$

*which denotes the successful resolvent, and thus the derivation is complete and successful, where* $P, \langle\{a,b,c\}^{\mathbf{o}(d.ins)}\rightarrow\{a,b,c,d\}\rangle \vdash d.ins$

*2.d) We end this iteration by setting* $\pi' = \langle\{\}^{\mathbf{o}(a.ins)}\rightarrow\{a\}^{\mathbf{o}(c.ins)}\rightarrow\{a,c\}^{\mathbf{o}(b.ins)}\rightarrow\{a,b,c\}^{\mathbf{o}(d.ins)}\rightarrow\{a,b,c,d\}\rangle$, $id' = 6$, $P' = P'$, $ESet' = \emptyset$

*Now, in the end of the while cycle we compute* $\texttt{needResponse}(ESet', 1, 6)$ *which returns* $\emptyset$ *because* $ESet' = \emptyset$*, and the iteration stops.*

*Finally, we return to the first call* $ExpandPath(P, e_1, \langle\{\}\rangle, \emptyset, 1)$ *where*

$$ExpandPath(P, e_1, \langle\{\}\rangle, \emptyset, 1) = (P, \langle\{\}^{\mathbf{o}(a.ins)}\rightarrow\{a\}^{\mathbf{o}(c.ins)}\rightarrow\{a,c\}^{\mathbf{o}(b.ins)}\rightarrow\{a,b,c\}^{\mathbf{o}(d.ins)}\rightarrow\{a,b,c,d\}\rangle, \emptyset, 6)$$

*And thus we obtain the resolvent:*

$$\langle\{\}^{\mathbf{o}(a.ins)}\rightarrow\{a\}^{\mathbf{o}(c.ins)}\rightarrow\{a,c\}^{\mathbf{o}(b.ins)}\rightarrow\{a,b,c\}^{\mathbf{o}(d.ins)}\rightarrow\{a,b,c,d\}\rangle, \emptyset \Vdash^{6}_{P} ()$$

*Since this is the successful resolvent, the derivation is said to be complete and successful. Consequently, we have proven that:*

$$P, \langle\{\}^{\mathbf{o}(a.ins)}\rightarrow\{a\}^{\mathbf{o}(c.ins)}\rightarrow\{a,c\}^{\mathbf{o}(b.ins)}\rightarrow\{a,b,c\}^{\mathbf{o}(d.ins)}\rightarrow\{a,b,c,d\}\rangle \vdash e_x$$

*as intended.*

Recall that in section 11.4 we mentioned some desired properties when responding to a stream of events $e_1 \otimes \ldots \otimes e_n \otimes \ldots \otimes e_k$. In particular, we mentioned that we cannot guarantee that $P, \pi \models e_1 \otimes \ldots \otimes e_n \otimes e_{n+1} \otimes \ldots \otimes e_k$ holds, for a super path $\pi$ of $\pi_1 \circ \pi_2$, when $P, \pi_1 \models e_1 \otimes \ldots \otimes e_n$ and $P, \pi_2 \models e_{n+1} \otimes \ldots \otimes e_k$ hold, since the events that happened in $\pi_1$ cannot be taken into account in path $\pi_2$, as the latter represents a new (and "clean") execution.

Although this property cannot be achieved by our executional entailment theory, as it will require us some notion of *history* (which is discussed further in section 14.4), this property can be achieved by our procedure, as we keep track of the occurrence history by using temporary rules. This can be formulated as follows:

**Theorem 12.** *Let $P$ be a program, $\phi_1$ $\phi_2$ be serial-Horn transaction formulas, $\pi_1, \pi_2$ be paths, where $D$ is the initial state of $\pi_1$ and $D_f$ is its end state.*

*If there is a derivation starting in $\langle D \rangle, \emptyset \Vdash^{1}_{P} \phi_1$ and ending in $\pi_1, \emptyset \Vdash^{id_1}_{P_1} ()$, and a derivation starting in $\langle D_f \rangle, \emptyset \Vdash^{id_1}_{P_1} \phi_2$ and ending in $\pi_2, \emptyset \Vdash^{id_2}_{P_2} ()$ then:*

$$P, \pi_1 \circ \pi_2 \vdash \phi_1 \otimes \phi_2$$

*Proof.* Trivially true, since there is a derivation starting in $\langle D \rangle, \emptyset \Vdash^{1}_{P} \phi_1$ and ending in $\pi_1, \emptyset \Vdash^{id_1}_{P_1} ()$, then there is also a derivation starting in $\langle D \rangle, \emptyset \Vdash^{1}_{P} \phi_1 \otimes \phi_2$, obtaining the resolvent $\pi_1, \emptyset \Vdash^{id_1}_{P_1} \phi_2$. Moreover, since there is a derivation starting in $\langle D_f \rangle, \emptyset \Vdash^{id_1}_{P_1} \phi_2$ and ending in $\pi_2, \emptyset \Vdash^{id_2}_{P_2} ()$, then the resolvent $\pi_1, \emptyset \Vdash^{id_1}_{P_1} \phi_2$ will also end in $\pi_1 \circ \pi_2, \emptyset \Vdash^{id_2}_{P_2} ()$, and thus $P, \pi_1 \circ \pi_2 \vdash \phi_1 \otimes \phi_2$ holds. □

## 11.6 $\mathcal{TR}^{ev}$ as an Event-Condition-Language

As already argued in chapter 2, Event-Condition-Action (ECA) languages are the commonly accepted paradigm to support reactivity. Originally proposed in the context of active database systems, ECA-languages have been applied to countless scenarios by defining a standard rule syntax, which specifies the reactive behavior of a system. In its syntax, a reactive (or ECA-rule) is of the following form:

**On** *event* **if** *condition* **do** *action*

which states that, whenever the event happens, the system is tested to check if it is in a specific state where the condition holds, and if that is the case, the action is executed as a response. Moreover, in order to be useful in practice, all of its ingredients (event, condition and action) can be complex and defined by expressive event and action expressions, and some logic language for the condition.

ECA-rules can be very easily encoded in $\mathcal{TR}^{ev}$. Moreover, contrary to the majority of ECA-languages, in $\mathcal{TR}^{ev}$ the action part can be formulated as a transaction, executed in an all-or-nothing way, and where every event is necessarily responded to as a transaction. Since $\mathcal{TR}^{ev}$ forces every formula $\mathbf{r}(ev)$ (denoting the response of a given event $ev$) to be

true whenever $\mathbf{o}(ev)$ (the occurrence of $ev$) is learned to hold, then this *event-condition-transaction* behavior can be simply encoded as:

$$\begin{aligned} \mathbf{r}(ev) &\leftarrow \Diamond cond \otimes action \\ \mathbf{r}(ev) &\leftarrow \neg\Diamond cond \end{aligned} \tag{11.2}$$

where $\Diamond cond$ is a test (and which necessarily does not cause changes in the KB) to determine if the condition $cond$ holds, and if this is the case, the $action$ is necessarily executed in the KB. Note that all of its components can be complex. In this sense, the event $ev$ can be an event pattern defined by some event rule ($body \Rightarrow \mathbf{o}(ev)$), while both the action and the condition can be defined by some declarative transaction rule.

In addition, as was previously discussed in section 11.3, the several operational behaviors of an ECA-language can be obtained by different instantiations of the $choice$ function, according to the application needs.

As an illustration of how to write ECA-rules in $\mathcal{TR}^{ev}$ consider the following example.

**Example 41** (Lien Execution)**.** *Recall the scenario from chapter 9 in page 99, regarding the government issuing a lien execution order for a given citizen and a given amount of money. In such a scenario, when a bank receives this event, it has to check if the citizen is a client and then if it has enough money available. If that is the case, then the bank seizes the amount required, notifying both the client and the government. Conversely, if the client does not have enough money, the account is frozen. Note here that it is crucial to ensure that these actions are treated as a transaction. Namely: it can never be the case that the citizen is a client of the bank and no action is performed; or if the client withdraws the money in between the balance checking and the seizure execution, then the system has to make sure that the account is frozen instead.*

*Using the previous encoding, the event-condition-transactions rules in this bank system can be written in $\mathcal{TR}^{ev}$ as:*

$$\begin{aligned} &\mathbf{r}(lien(Amt, X)) \leftarrow hasBalance(Amt, X) \otimes liensExec(Amt, X) \\ &\mathbf{r}(lien(Amt, X)) \leftarrow noBalance(Amt, X) \otimes freezeAcct(X) \\ &hasBalance(X, Amt) \leftarrow account(Cl, Ac) \otimes avalBalance(Ac, B) \otimes Amt <= B \\ &noBalance(X, Amt) \leftarrow account(Cl, Ac) \otimes avalBalance(Ac, B) \otimes Amt > B \end{aligned}$$

*where, since $hasBalance$ and $noBalance$ are queries to the KB, we can drop the $\Diamond$ constructor; $freezeAcct(X)$ defines the act of freezing the account of person $X$, and $lienExec$ the transaction that executes the lien by seizing a giving amount from an account $X$.*

*Additionally, the transaction $seize$ for seizing the money of a bank's client, and a transaction $lienExec$ to be triggered by the lien event can be defined in $\mathcal{TR}^{ev}$ e.g. as:*

$$\begin{aligned} &seize(Am, Ac) \leftarrow avalBalance(Ac, B).del \otimes avalBalance(Ac, B - Am).ins \\ &lienExec(Am, Cl) \leftarrow account(Cl, Ac) \otimes seize(Am, Ac) \otimes notify(Cl) \otimes notify(st) \end{aligned}$$

As another example assume the following scenario of fraud detection in a telecommunications company.

**Example 42** (Fraud Detection)**.** *Consider the case of a telecommunications company, which needs to detect abusive and fraudulent behavior, and act upon it, to prevent monetary losses. Examples of a probable abusive behavior are the attempt for a given number to generate two calls simultaneously for different end numbers; or the case where the client receives credit for incoming calls and then it is flagged by the system due to an abnormal call length activity.*

*This could be expressed in $\mathcal{TR}^{ev}$ in a simplified way as:*

$$\begin{aligned}&\mathbf{o}(initiateCall(X,Y).ins) \wedge \mathbf{o}(initiateCall(X,Z).ins) \wedge Y \neq Z \Rightarrow \\ &\qquad \mathbf{o}(cardDuplicationFraud(X)) \\ &\mathbf{o}(incomingCallCredit(X,Y,Amt)) \ ; \ \mathbf{o}(flagAbnormalCallsLenght(X)) \Rightarrow \\ &\qquad \mathbf{o}(creditReceiveFraud(X))\end{aligned}$$

*where we assume that $initiateCall(X,Y).ins$ denotes the action of generating a call, with origin in X and ending in Y, and thus $\mathbf{o}(initiateCall(X,Y).ins)$ is an atomic event triggered due to the execution of an oracle's primitive. In opposition, we assume $\mathbf{o}(incomingCallCredit(X,Y,Amt))$ to be an external event, and $\mathbf{o}(flagAbnormalCallsLenght(X))$ an internal event explicitly triggered by the system.*

*Whenever one of those situations happens, the company will pose a request for the situation to be analyzed by the fraud team, and will limit the usage of that number. The latter means that the client will not be able to generate calls, but still can receive them. Moreover, if a possible fraud happened before for the same client with number $X$, then the system considers it as a much likely serious fraud scenario. In this case, it blocks the number $X$ directly (for generated, and incoming calls) and will request a more serious and quick analysis from the fraud team.*

*This behavior can be simply addressed by ECA-rules, which can be represented in $\mathcal{TR}^{ev}$ as follows.*

$$\begin{aligned}&\mathbf{o}(creditReceiveFraud(X)) \Rightarrow \mathbf{o}(possibleFraud(X)) \\ &\mathbf{o}(cardDuplicationFraud(X)) \Rightarrow \mathbf{o}(possibleFraud(X)) \\ &\mathbf{o}(possibleFraud(X)) \ ; \ \mathbf{o}(possibleFraud(X)) \Rightarrow \mathbf{o}(fraud(X))\end{aligned}$$

$$\begin{aligned}&\mathbf{r}(possibleFraud(X)) \leftarrow restrictNumber(X) \otimes analyzeCase(X) \\ &\mathbf{r}(fraud(X)) \leftarrow blockNumber(X) \otimes blackList(X).ins \otimes analyzeCaseUrgent(X)\end{aligned}$$

In the following we illustrate another scenario, now in the context of agent systems.

**Example 43.** *Imagine a multi-agent system in a chemical laboratory scenario, where we need to specify an event $e_c$ that is true whenever the alarm indicating the decrease of the external temperature below 0°C occurs after a given compound $c$ is taken out of the oven. Thus, event $e_c$ is a complex event that results from the temporal combination of an external event (the external temperature) and an internal event (the action of taking out compound $c$ from the oven). Whenever*

*event $e_c$ is detected, the agent needs to move the compound $c$ to a particular container in order to preserve its characteristics. This can be expressed as:*

$$\mathbf{o}(put(c, oven).del); \mathbf{o}(temperature\_lower\_0) \Rightarrow \mathbf{o}(e_c)$$
$$\mathbf{r}(e_c) \leftarrow move(c, X, container)$$
$$move(Obj, X, Y) \leftarrow is(Obj, X) \otimes is(Obj, X).del \otimes is(Obj, Y).ins$$

*where $\mathbf{o}(is(c, oven).del)$ denotes the occurrence associated with the action of taking solution $c$ out from the oven, and $\mathbf{o}(temperature\_lower\_0)$ the external event occurrence stating that the external temperature is below 0°C.*

*Moreover imagine we want to state that whenever the agent puts an object on a surface $X$ that is dirty, then the action $clean(X)$ needs to be issued. This is a typical ECA-rule of the form:* ***on*** *$is(Obj, X).ins$* ***if*** *$isDirty(X)$* ***then*** *$clean(X)$ and can be expressed in $\mathcal{TR}^{ev}$ as follows:*

$$\mathbf{o}(is(Obj, X).ins) \Rightarrow \mathbf{o}(checkClean(X))$$
$$\mathbf{r}(checkClean(X)) \leftarrow isDirty \otimes clean(X)$$
$$\mathbf{r}(checkClean(X)) \leftarrow \neg isDirty$$

# 12

# Discussion

In this part, we have presented Transaction Logic with Events ($\mathcal{TR}^{ev}$), an extension of Transaction Logic to execute and model the behavior of transactions that react automatically to complex events. For that, $\mathcal{TR}^{ev}$ builds upon the definitions of Transaction Logic, and extends them with the ability to detect and react to complex events. In fact, as shown in chapter 10, Transaction Logic can be used to either reason about complex actions or reason about complex events, but it lacks the proper tools to reason about both simultaneously. The main problem is that, when executing actions to react to complex events, the logic needs to guarantee that *all* events detected (including the ones that become true during this reaction) are also responded as a transaction. As a consequence of this, transactions need to impose that all events occurring during their execution are properly responded to, and that no transaction can succeed (or commit) without this guarantee.

Interestingly, $\mathcal{TR}^{ev}$ shows that such a transactional reactive behavior, where all occurring events need to be responded, can be elegantly encoded in a non-monotonic manner. In it, transaction formulas and event formulas are evaluated differently according to distinct satisfaction relations (respectively $\models$ and $\models_{ev}$). Then, satisfaction of transaction formulas over a given path depends on what event formulas are satisfied (by $\models_{ev}$) over that same path. If in a path $\pi$ there is an atomic event formula $\mathbf{o}(e)$ satisfied which is not responded (i.e., if $\mathbf{r}(e)$ does not hold after $\mathbf{o}(e)$), then no transaction formula can hold in $\pi$, and $\pi$ needs to be expanded with the satisfaction of $\mathbf{r}(e)$.

This behavior is also presented in $\mathcal{TR}^{ev}$'s procedure, where we require paths to be expanded with the events that are learned to be true. Additionally, this procedure is based on both $\mathcal{TR}$'s proof theory and ETALIS's event detection algorithm, in which the program is transformed to keep track, at each moment, of what events patterns are partially satisfied, and what atomic events still needs to occur (and when) in order for a given

pattern be triggered.

Moreover, as in Transaction Logic, $\mathcal{TR}^{ev}$ abstracts the KB theory and its primitives from the logic that reasons about their reactive behavior. To achieve this, both $\mathcal{TR}^{ev}$ model theory and procedure are parameterized with a pair of oracles defining the KB semantics and its primitive actions, and thereby being useful in a wide range of scenarios. This is also true for the operational choices of a reactive language, that are encoded in $\mathcal{TR}^{ev}$'s $choice$ function, and which define what is the next event to be responded and in what conditions should it be responded. However, while this $choice$ function is also a parameter of $\mathcal{TR}^{ev}$'s model theory, for the definition of the proof procedure, specific choices regarding this operational behavior had to be made. This is as expected since, it is hard to abstract all the executional details when specifying how a procedure executes. In this sense, while the ordering of events is still abstracted by the function $FirstInOrder$, this procedure fixes a given response policy (i.e., a $firstUnans$ definition) for the definition of the $Closure$ computation function.

Additionally, as it happens with the primer semantics of ETALIS, all instances of an event will be selected by $\mathcal{TR}^{ev}$'s semantics to trigger an event pattern, which means that $\mathcal{TR}^{ev}$ assumes an unrestricted event context for event consumption. Clearly, while other event consumption patterns could be easily implemented, as pointed out by [Ani11], changing the semantics to only consider some of the event instances that have happened could damage the declarative property of the language, as they cause different orders in which event rules are evaluated and triggered.

From its characteristics, $\mathcal{TR}^{ev}$ represents an important step in the context of reactive languages, by providing the ability to combine complex event detection with transactional execution. Moreover, $\mathcal{TR}^{ev}$ does this in a declarative model-theoretic way, which makes it suitable to reason about properties of reactive transactions and reactive programs. To the best of our knowledge, $\mathcal{TR}^{ev}$ is the first full-fledged logic that can execute *and* reason about the behavior of reactive transactions in arbitrary theories.

Nevertheless, it is worth mentioning that in the existing literature, one can find several solutions to reason about complex events and execute (trans)actions. In this sense, complex event processing (CEP) systems as [AC05; WDR06; AFRSSS10] can reason very efficiently with large streams of data and detect complex events. These support a rich specification of events based on event pattern rules combining atomic events with some temporal constructs. As shown in theorem 8, $\mathcal{TR}$ and $\mathcal{TR}^{ev}$ can express most event patterns of SNOOP and ETALIS, failing only to translate the expressions that require the explicit specification of time in the language. ETALIS [AFRSSS10] CEP system even uses $\mathcal{TR}$'s syntax and connectives, although abandoning $\mathcal{TR}$'s model theory and providing a different satisfaction definition. However, CEP systems, which include ETALIS and SNOOP are decoupled from the consequences of detecting events, which means that they do not deal with the problem of executing (trans)actions in reaction of the events detected. Since we have proposed a procedure based on ETALIS algorithm with exactly this feature, one can see the procedure presented in section 11.5 as an extension of ETALIS

algorithm with the ability to execute transactions in response to a stream of events.

To reason about actions and transactions, solutions like the Situation Calculus, Event Calculus, Action Languages, etc., are popular for their ability to model, very expressively, the direct and indirect effects of actions in KBs. Motivated by the success of relational and deductive databases, several extensions of these languages like [BLT97; BPV98] have been proposed to reason about the effects of a subset of actions that follows a transactional behavior and, some of these solutions can also reason about events that behave similarly to database triggers. However, as in database triggers, these events are restricted to the detection of simple actions like "on insert/delete" and thus have a very limited expressivity that fails to encode arbitrary complex events. Furthermore, although the examples herein are all based on a relational oracle, nothing prevents $\mathcal{TR}^{ev}$ to use alternative KBs definitions and subsequently primitive actions. In fact, note that all the oracles instantiations defined in chapter 6 can also be used for $\mathcal{TR}^{ev}$, with some small changes (to adapt external oracles into internal oracles). As such, we could e.g., assume an Action Language oracle and use $\mathcal{TR}^{ev}$ to model reactive transactions where atomic events are the actions defined in a Action Language transition relation. In this sense, $\mathcal{TR}^{ev}$ can also be seen as a logic that gives the user the ability to combine transactions and complex events with such well-established solutions that reason efficiently about actions and their effects (like Action Languages, Situation Calculus, etc.).

On another perspective, to simultaneously reason about actions and complex events, ECA languages [ABB11; BEP06; CLN03] are the standard paradigm.

However, ECA languages rarely allow the action component of the language to be defined as a transaction, and when they do, they lack from a declarative semantics as in [PPW06]; or they are based on active databases and can only detect atomic events defined as insertions/deletes as in [Zan95; LLM98]. In opposition, $\mathcal{TR}^{ev}$ can define complex events and, since its theory is parametric on a database and transition oracle, atomic events are arbitrary.

Additionally, as shown in eq. (11.2) of section 11.6, ECA-rules can be naturally encoded in $\mathcal{TR}^{ev}$. As such, $\mathcal{TR}^{ev}$ represents an important contribution to the research in ECA-languages, where it can be seen as an Event-Condition-Transaction language, where the action component is not only executed, but executed as a transaction. This means that, by using $\mathcal{TR}^{ev}$, we can guarantee that either the whole set of actions defined as a transaction is executed or, if anything fails meanwhile, the KB is left unchanged. As shown in section 11.6, this is paramount in several application domains that require properties on the outcome of their execution actions, and $\mathcal{TR}^{ev}$ gives a very important contribution in modeling such behavior.

Moreover, by providing different instantiations of the choice function, $\mathcal{TR}^{ev}$ can also offer different operational behaviors depending on the application needs. Finally, the procedure presented herein gives an important contribution to implement such an Event-Condition-Transaction language, and closing the existing gap to use it in real scenarios.

# Part IV

# Reactive Transactions with External Actions

# 13

# Combining reactivity and the execution of external actions

In parts II and III, we have presented two independent $\mathcal{TR}$ extensions: $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$ to deal with the problem of executing external actions, and reacting to complex events by issuing transactions, respectively.

However, several scenarios demand the combination between events and external actions, when executing transactions. In fact, interaction with an external entity is normally perceived as a "two-way street", where the system may execute actions externally (as in $\mathcal{ETR}$), but also react automatically to changes made by others (as in $\mathcal{TR}^{ev}$). Moreover, as before, we focus on the scenarios where such a complex interaction is needed, but where it is also important to achieve transactional properties over that interaction. As illustration, consider the following example from $\mathcal{ETR}$.

**Example 44** (Product Request (continued))**.** *Recall example 10 from page 40 of part II, where we model the interaction between the customer and an organization selling products across several associated company stores.*

*In it, one way the organization has to satisfy a customer's request for buying a product, is by asking an associated company whether it has the product in the asked quantity $N$, asking the customer whether she accepts that the product is supplied by that other company, and requesting the company to send it to the customer:*

$$
\begin{aligned}
request(Prd, N, Cust, OrderID) \leftarrow \\
&\mathbf{ext}(askComp(Prd, N, OrderID), forget(Prod, N, OrderID)) \\
&\otimes askCust(Cust, Prd, OrderID) \\
&\otimes requestDisp(Prd, N, Cust, OrderID)
\end{aligned}
$$

*However, the external action $askCust(Cust, Prd)$ requires an answer from the customer which may arrive later in time. This can be more precisely modeled as an event occurrence of the form $\mathbf{o}(reply(Cust, OrderID, Answer))$, signaling that a given customer $Cust$ replied with a yes or no (encoded in $Answer$) to the request identified with $OrderID$. Assuming, that such an event arrives to the system, then we need to capture this event occurrence, and respond to it accordingly. We can model such interaction with the customer, using events and external actions as follows:*

$$
\begin{aligned}
&request(Prd, N, Cust, OrderID) \leftarrow \\
&\qquad \mathbf{ext}(askComp(Prd, N, OrderID), forget(Prd, N, OrderID)) \\
&\qquad \otimes\ askCust(Cust, Prd, OrderID) \\
&\mathbf{r}(reply(Cust, OrderID, yes)) \leftarrow requestDisp(Prd, N, Cust, OrderID) \\
&\mathbf{r}(reply(Cust, OrderID, no)) \leftarrow forget(Prod, N, OrderID)
\end{aligned}
$$

*where we assume the event $\mathbf{o}(reply(Cust, OrderID, Answer))$ to arrive from the exterior, after the external action $askCust(Cust, Prd, OrderID)$ is performed. With this modeling, the request asking the associated company to send the product is only issued, if the event arrives with a positive answer from the client. Otherwise, if the event arrives containing a negative answer from the client, then the associated company is informed accordingly.*

*Note that if the action of asking the customer fails for any reason (e.g. because the client could not be reached), then the associated company is also informed to forget about the order.*

The previous example illustrates the need to detect internal and external events *and* issue actions in response to these events, in a transactional-way. This kind of behavior can actually be observed in most of $\mathcal{ETR}$ examples. For instance, in example 22 of $\mathcal{ETR}$, we model the scenario of a hotel in a ski resort, making the prices of the hotel's ski package product dependent on the weather conditions of the external ski resort. In it, the prices of the ski package should be higher if the ski conditions are "premium", and lower otherwise. Thus, whenever the weather conditions change, the ski resort must update its own knowledge base accordingly. Intuitively, such an interaction, where the hotel receives information about the ski conditions, can be naturally modeled using events.

On the other hand, the need to execute internal and external actions, while simultaneously reacting to complex events can also be found in $\mathcal{TR}^{ev}$'s examples. For instance, in example 41 of $\mathcal{TR}^{ev}$, we illustrate a system that receives requests from the state to issue a given lien execution order for a given client, and a given amount of money. As specified in the example, the execution of a lien by the system is modeled by the rule:

$$lienExec(Am, Cl) \leftarrow account(Cl, Ac) \otimes seize(Am, Ac) \otimes notify(Cl) \otimes notify(st)$$

Clearly, these actions of notifying the client and the state (i.e., $notify(Cl)$ and $notify(st)$, respectively) are external actions, that cannot be rolled back (something that is not taken care by $\mathcal{TR}^{ev}$ semantics).

In order to deal with scenarios involving (internal and external) actions and complex

events, we need a logic combining $\mathcal{TR}^{ev}$'s reactive features, and $\mathcal{ETR}$'s features to execute actions in an external environments, in a transactional-like way.

Although $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$ should be perceived as the main contributions of this thesis, they were also designed to be orthogonal and compatible with each other. In this chapter we show how these two logics can indeed be combined into one single logic theory, which deals simultaneously with external failures (from $\mathcal{ETR}$) and path expansions (from $\mathcal{TR}^{ev}$). Moreover, this unified logic theory is mostly achieved by simply combining $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$ definitions though, as expected, we still need to cater for some technical details when integrating the possibility of failure and external compensations with the detection of complex events arising from internal and external actions.

Namely, $\mathcal{TR}^{ev}$'s definitions (as it happens with $\mathcal{TR}$'s) do not consider the notion of a failed execution path for a transaction. More precisely, since the logic only deals with internal domains, and rolling back is always possible, there is no need to model the failed execution of a (trans)action over a path. As such, in $\mathcal{TR}^{ev}$'s model theory, the path where a transaction fails, rolls back the internal state, and succeeds, is equivalent to the path where a transaction succeeds directly.

However, when allowing external actions to be executed in response to events, care must be taken with paths where transactions do not completely succeed since external actions and external events can no longer be rolled back. Thus, as in $\mathcal{ETR}$, whenever a failure happens, we need to compensate for all the external actions executed before the failure. Additionally, since we can now define event patterns based on the actions performed externally (including compensations), we need also to cater for the patterns that might become true because of the external actions which were compensated (and which cannot be deleted/rolled back), and due to the execution of compensations themselves.

Consequently, the main difficulty of this combination is to properly define what it means for an action to fail, when responding to events. In particular, now an action may fail, because it is impossible to execute it, or because it is not possible to execute a response of an event that becomes true due to that action.

In $\mathcal{ETR}$, the detection of a failure is done by providing two auxiliary definitions – the partial ($\models_p$) and the classical ($\models_c$) satisfaction relations. Then, we say that a formula $\phi$ "fails" on a path in a way that can be compensated iff $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$.

In this combination, we provide the same notion to define paths where formulas fail but can be compensated. However, such a failure can also occur during the expansion step, i.e., when expanding a path with the responses of the events triggered in that path. As we shall see, to handle a failure when expanding a path requires further notions of partial, classical and general expansion, and changes in the $choice$ function. Then, a classical expansion path responds to events without considering the possibility of failure (and corresponds to $\mathcal{TR}^{ev}$'s definition of path expansion). The partial expansion considers the possibility of failure, stopping the expansion of that path whenever a failure occurs. And finally, the general expansion expands a path considering the possibility of a formula to fail, as long as it rolls back internally, compensates externally, and succeeds

on a path starting from that failure.

Next we formalize $\mathcal{ETR}^{ev}$, External Transaction Logic with Events, which combines the theory of our main contributions $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$ in one single unified theory. $\mathcal{ETR}^{ev}$ is able to automatically react to complex events that arise from internal and external changes, and execute transactions involving both internal and external actions, in response to these events.

In the following we start by formalizing $\mathcal{ETR}^{ev}$'s syntax (section 13.1), its model theory (section 13.2), the event *choice* function (section 13.2.1) and the notion of models and entailment (section 13.2.2). Finally, we end with some discussion (section 13.3).

## 13.1 $\mathcal{ETR}^{ev}$'s Syntax

The language of $\mathcal{ETR}^{ev}$ results from combining $\mathcal{ETR}$'s and $\mathcal{TR}^{ev}$'s operators and syntax. In a nutshell, formulas of the language are partitioned into transaction formulas and event formulas (like in $\mathcal{TR}^{ev}$). And like $\mathcal{ETR}$, they are constructed based on oracle primitives, which are also partitioned into internal and external primitives.

Formally, $\mathcal{ETR}^{ev}$'s alphabet contains an infinite number of constants $\mathcal{C}$, function symbols $\mathcal{F}$, predicate symbols $\mathcal{P}$ and variables $\mathcal{V}$. Predicate symbols are further partitioned into transaction names ($\mathcal{P}_t$), event names ($\mathcal{P}_e$), and oracle primitives ($\mathcal{P}_\mathcal{O}$). In this context, transaction names are the names that can appear in the program to define complex transaction procedures (i.e., the names appearing as heads of transaction rules); and event names denote the names of events that can occur and be responded (i.e., the names appearing as heads of event rules).

Propositions in $\mathcal{P}_\mathcal{O}$ define primitive actions and queries to deal with the internal and external KB. Consequently, and just like in $\mathcal{ETR}$, $\mathcal{P}_\mathcal{O}$ is still partitioned into $\mathcal{L}_i$ the internal domain's language, and $\mathcal{L}_a$ the external domain's language. Here, the former defines primitives to query and change the internal KB, while $\mathcal{L}_a$ denotes the external primitives that can be executed externally. For convenience, we assume that $\mathcal{L}_a$ contains two distinct actions `failop` and `nop`, respectively defining trivial failure in the external domain, and trivial success in the external domain without changing the external state. As before, we work with a Herbrand instantiation of the language, in the usual way.

As in $\mathcal{ETR}$, negation in transaction formulas is restricted to atoms. However, its use is unrestricted in event formulas.

**Definition 65** (Transaction Formulas)**.** *A* transaction atom *is either a proposition in $\mathcal{P}_t$, $\mathcal{P}_e$, $\mathcal{P}_\mathcal{O}$, $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ where $a, b_i \in \mathcal{L}_a$, or $\mathbf{r}(\varphi)$ where $\varphi \in \mathcal{P}_\mathcal{O} \cup \mathcal{P}_e$. A transaction literal is either $\phi$ or $\neg\phi$ where $\phi$ is a transaction atom. A* transaction formula *is either a transaction literal or an expression, defined inductively, of the form: $\phi \wedge \psi$, $\phi \vee \psi$, or $\phi \otimes \psi$ where $\phi$ and $\psi$ are transaction formulas.*

As usual, transaction formulas are the formulas whose satisfaction means execution. Thus, a transaction formula is said to be true on a path $\pi$, if that path corresponds to a

valid (transactional) execution of that formula. Just like in $\mathcal{ETR}$, executing a transaction formula may include executing *internal* and *external* actions. As before, to be able to refer to the external actions in formulas, we define $\mathcal{L}_a^*$ as the augmentation of $\mathcal{L}_a$ with the formulas $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ where $a, b_i \in \mathcal{L}_a$. Moreover, and just like in $\mathcal{TR}^{ev}$, explicitly triggering an event is also a transaction formula.

Events are different from transactions, as their satisfaction means occurrence rather than execution. In this sense, an event is said to be true on a path, if we observe its occurrence in that path.

**Definition 66** (Event Formulas)**.** *An* event occurrence *is of the form* $\mathbf{o}(\varphi)$ *where* $\varphi \in \mathcal{P}_e$ *or* $\varphi \in \mathcal{P}_\mathcal{O}$*. An event formula is either an event occurrence, or an expression* $\neg\phi$*,* $\phi \wedge \psi$*,* $\phi \vee \psi$*,* $\phi \otimes \psi$*, or* $\phi; \psi$ *where* $\phi$ *and* $\psi$ *are event formulas.*

The latter definition is very similar to $\mathcal{TR}^{ev}$'s definition 45 in page 119. However, contrary to $\mathcal{TR}^{ev}$, event formulas can now be constructed based on the occurrence of external primitives, and those can be used to specify complex event patterns.

Based on these definitions of transaction formulas and event formulas, we can define transaction rules, event rules and programs in the usual way.

**Definition 67** (Rules and Programs)**.** *A* transaction rule *is a formula of the form* $\varphi \leftarrow \psi$ *s.t.* $\varphi$ *is a transaction atom and* $\psi$ *a transaction formula.*
*A* complex event rule *is a formula of the form* $\varphi \Rightarrow \psi$ *s.t.* $\psi$ *is an event occurrence and* $\varphi$ *is a event formula. A program* $P$ *is a set of transaction rules and complex event rules.*

**Example 45** (Running Example)**.** *To illustrate the language and dynamics of* $\mathcal{ETR}^{ev}$ *consider the following program.*

$$
\begin{array}{ll}
t \leftarrow \mathbf{ext}(a, a_1 \otimes a_2) \otimes p.ins & \\
t \leftarrow q.ins \otimes e_1 & \\
 & \mathbf{r}(e_1) \leftarrow \mathbf{ext}(c, c_1) \\
\mathbf{o}(a)\ ;\ \mathbf{o}(p.ins) \Rightarrow \mathbf{o}(e_2) & \mathbf{r}(e_2) \leftarrow v.ins \otimes \mathbf{ext}(b, b_1) \\
\mathbf{o}(a)\ ;\ \mathbf{o}(a_2) \Rightarrow \mathbf{o}(e_3) & \mathbf{r}(e_3) \leftarrow s.ins \\
\mathbf{o}(e_2)\ ;\ \mathbf{o}(c) \Rightarrow \mathbf{o}(e_4) & \mathbf{r}(e_4) \leftarrow \mathbf{ext}(d, d_1)
\end{array}
$$

*where* $e_1, e_2, e_3, e_4 \in \mathcal{P}_e$*,* $t \in \mathcal{P}_t$*,* $p, q, s, v \in \mathcal{L}_i$ *and* $a, a_1, a_2, b, b_1, c, c_1, d, d_1 \in \mathcal{L}_a$*.*

## 13.2 Model Theory

In this section, we formally combine $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$ as an unified theory. However, this integration is not straightforward, since the two theories have different requirements for a formula execution to be considered valid.

A crucial point of $\mathcal{ETR}$'s theory is how to identify that a transaction formula *failed* over a particular path, in a way that this failed path represents a legitimate try for executing the formula. To that end, $\mathcal{ETR}$ employs three transaction satisfaction definitions:

classical, partial, and general satisfaction relations. With them, a formula is said to "legitimately" fail, if it is partially satisfied on a path $\pi$, over which it is not classically satisfied. As such, the partial satisfaction definition has the important responsibility to identify all the possible sources of failure, and constrain the evolution of the path whenever such a failure happens.

Based on these notions, executing a formula in $\mathcal{ETR}$ corresponds to a normal $\mathcal{TR}$ transaction execution with the additional possibility of executing external actions; or to an execution where a failure occurs, external actions are compensated, and the formula succeeds on a path started after the compensations. On the other hand, in $\mathcal{TR}^{ev}$ a transaction execution corresponds to the execution of the actions specified in that transaction, *plus* the execution of the event responses whose events occurred over that path. The latter is done by *expanding* the first path with the responses to all the events triggered.

$\mathcal{TR}^{ev}$ does not deal with the notion of failure, since every action is internal, and rollbacks are always a valid option. In particular, for the model theory perspective, the execution where one executes an action, rolls back its effects and then succeeds in an alternative branch, is just equivalent to executing the successful branch directly.

Similarly, $\mathcal{ETR}$ does not include the possibility to detect (complex) events, nor is able to make satisfaction of transactions dependent on the satisfaction of the event responses. As one could expect, one possible way to achieve this combination is by extending $\mathcal{ETR}$'s satisfaction definitions with the notion of path expansion. However, in that case, we need to consider the expansion as another possible source of failure. More precisely, when executing a formula $\phi$, a failure may happen because $\phi$ cannot be executed, or because it is not possible to respond to the events triggered by the execution of $\phi$ (during the expansion). Independently of the cause of failure, we need to roll back all the internal actions, and compensate for the external actions executed before the failure.

Importantly, since external actions are also primitive events, executing compensations can also trigger events, and we need to make sure that every event is responded to, even after we compensate for a given formula.

To define such a theory, as usual, we start by formally specifying what are interpretations. In this context, an interpretation is a mapping from a pair of states, into a set of formulas. We adopt the definition of states and paths of $\mathcal{ETR}$ where a state is a pair containing a representation of the internal state together with a representation of the external state, and a path is just a sequence of states. Additionally, like in $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$, paths are annotated with the (external and internal) actions that have been executed; and with the explicit events that have been triggered in the path.

**Definition 68** (Interpretations)**.** *An interpretation is a mapping $M$ that assigns a classical Herbrand structure (or $\top$) to every path. This mapping is subject to the following restriction, for all states $D_i$ and every formula $\varphi$*

1. $\varphi \in M(\langle (D, E) \rangle)$ *if* $\mathcal{O}^d(D) \models \varphi$ *for any external state* $E$
2. $\{\varphi, \mathbf{o}(\varphi)\} \subseteq M(\langle (D_1, E) \xrightarrow{\varphi} (D_2, E) \rangle)$ *if* $\mathcal{O}^t(D_1, D_2) \models \varphi$

3. $\{\varphi, \mathbf{o}(\varphi)\} \subseteq M(\langle (D, E_1) \xrightarrow{\varphi} (D, E_2) \rangle)$ *if* $\mathcal{O}^e(E_1, E_2) \models \varphi$

4. $\{\mathbf{ext}(\varphi, \psi), \mathbf{o}(\varphi)\} \subseteq M(\langle (D, E_1) \xrightarrow{\mathbf{ext}(\varphi,\psi)} (D, E_2) \rangle)$ *if* $\mathcal{O}^e(E_1, E_2) \models \varphi$

5. $\mathbf{o}(e) \in M(\langle (D, E) \xrightarrow{\mathbf{o}(e)} (D, E) \rangle)$ *for any internal and external states* $D, E$

The previous definition states what oracle primitives, and what explicit events are satisfied over 1-paths and 2-paths. Like in $\mathcal{TR}^{ev}$, we force interpretations to satisfy both the formula, and their occurrence on the paths where the oracles satisfy them (items 1, 2 and 3). Moreover, to be able to detect the occurrence of a given external action, whenever that action is executed in the context of an action that needs compensations, we include item 4, which satisfies the occurrence of $\phi$ whenever $\mathbf{ext}(\phi, \psi)$ appears in the path, and is satisfied by the oracle. Finally, and as in $\mathcal{TR}^{ev}$, item 5 states compliance between the occurrences observed in a path, and occurrences made true by an interpretation. Namely, whenever an occurrence appears annotated in the path transition, then every interpretation makes this occurrence true in that path.

Based on this notion of interpretation, we can now define satisfaction of general formulas over general paths. We assume that the operations over paths (like split and prefix) are defined as usual, and as in $\mathcal{TR}^{ev}$, satisfaction of formulas is partitioned between transaction formulas and event formulas. We start by formalizing the satisfaction of event formulas, which corresponds to the classical definition of $\mathcal{TR}$.

**Definition 69** (Satisfaction of Event Formulas)**.** *Let $M$ be a interpretation, $\pi$ a path and $\phi$ an event formula. If $M(\pi) = \top$ then $M, \pi \models_{ev} \phi$; otherwise:*

1. ***Base Case:*** *$M, \pi \models_{ev} p$ iff $p \in M(\pi)$ for every event atom $p$*

2. ***Negation:*** *$M, \pi \models_{ev} \neg\phi$ iff it is not the case that $M, \pi \models_{ev} \phi$*

3. ***"Classical" Conjunction:*** *$M, \pi \models_{ev} \phi \wedge \psi$ iff $M, \pi \models_{ev} \phi$ and $M, \pi \models_{ev} \psi$.*

4. ***Serial Conjunction:*** *$M, \pi \models_{ev} \phi \otimes \psi$ iff exists a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1 \models_{ev} \phi$ and $M, \pi_2 \models_{ev} \psi$*

*As usual we assume $\psi \Rightarrow \phi$ as syntactic sugar for $\neg\psi \vee \phi$; $\psi \vee \phi$ as syntactic sugar for $\neg\psi \wedge \neg\phi$; and $\phi; \psi$ as $\phi \otimes \mathtt{path} \otimes \psi$.*

Like in $\mathcal{ETR}$, satisfaction of complex transaction formulas requires two further auxiliary definitions – the classical and the partial satisfaction – to deal with external failures.

Moreover, similarly to $\mathcal{TR}^{ev}$, satisfying a transaction formula on a path requires (among other things) the response satisfaction of all the events occurring on that path. More precisely, it requires the *expansion* of that path, to ensure that every event triggered is properly responded to.

Importantly, since every satisfaction relation has different requirements, the notion of expansion needs to be different, according to the satisfaction relation in mind. Since the

classical satisfaction aims to satisfy formulas over paths without considering the possibility of failure, its expansion does not consider this possibility. On the contrary, since the goal of the partial satisfaction is exactly to identify failures and constrain the evolution of the path whenever such a failure happens, its expansion has to be different from the classical expansion. In particular, this partial expansion must identify when the response of an event fails (i.e., when it succeeds partially but not classically over a path), and in that case immediately stop the execution of the responses. Finally, the expansion for the general satisfaction must be able to deal with failures and compensate for them in the expansion step, like $\mathcal{ETR}$ does in the general satisfaction definition.

In the following we specify the three satisfaction relations along with their notion of path response and expansion. We start with the notion of the classical path response and classical expansion, which correspond to $\mathcal{TR}^{ev}$'s response and expansion path (cf. definition 53 and definition 51).

**Definition 70** (Classical path response)**.** *For a path $\pi_1$ and an interpretation $M$ we say that $\pi$ is a response of $\pi_1$ iff $choice(M, \pi_1, \models_c) = e$ and we can split $\pi$ into $\pi_1 \circ \pi_2$ such that $M, \pi_2 \models_c \mathbf{r}(e)$.*

Based on these two definitions, we can now define what is an expansion of a path.

**Definition 71** (Classical Expansion of a path)**.** *The path $\pi$ is a* classical expansion *of the path $\pi_1$ w.r.t. interpretation $M$ iff:*

- *$\pi$ is completely answered w.r.t. $M$ and $\models_c$, and*
- *either $\pi = \pi_1$; or there is a sequence of paths $\pi_1, \ldots, \pi$, starting in $\pi_1$ and ending in $\pi$, such that each $\pi_i$ in the sequence is a response of $\pi_{i-1}$ w.r.t. $M$.*

As expected, classical expansion is defined as the expansion of a path, where the responses to all events that occur on that path are executed w.r.t. the classical satisfaction relation (to be defined ahead in definition 72). As such, each path response $\pi_i$ can be seen as $\pi_{i-1} \circ \pi'$ where $\pi_{i-1}$ is the previous path in the sequence, $choice(M, \pi_{i-1}, \models_c) = e$ and $M, \pi' \models_c \mathbf{r}(e)$. Alternatively, if $\pi_1$ is a completely answered path w.r.t. $M$ and $choice(M, \pi_1, \models_c)$, the expansion of $\pi_1$ is $\pi_1$ itself.

As we shall see in section 13.2.1, events are evaluated to be answered or unanswered w.r.t. a given interpretation and satisfaction relation (cf. definition 82). Afterwards, the $choice$ function returns the first unanswered event given a predefined ordering, an interpretation, and a satisfaction relation, similarly to what happens in $\mathcal{TR}^{ev}$. Building on these definitions, we formalize what (complex) formulas are classically true on what paths. This definition corresponds exactly to the $\mathcal{TR}^{ev}$'s definition of transaction formulas.

**Definition 72** (Classical Satisfaction of Transaction Formulas)**.** *Let $M$ be a interpretation, $\pi$ a path and $\phi$ a transaction formula. If $M(\pi) = \top$ then $M, \pi \models_c \phi$; otherwise:*

1. ***Base Case:*** *$M, \pi \models_c p$ iff there is a prefix $\pi'$ of $\pi$ such that $p \in M(\pi')$ and $\pi$ is a classical expansion of path $\pi'$ w.r.t. $M$, for every transaction atom $p$ such that $p \notin \mathcal{P}_e$.*

2. ***Event Case:*** *$M, \pi \models_c e$ iff $e \in \mathcal{P}_e$, there is a prefix $\pi'$ of $\pi$ where $M, \pi' \models_{ev} \mathbf{o}(e)$ and $\pi$ is a classical expansion of path $\pi'$ w.r.t. $M$.*

3. ***Negation:*** *$M, \pi \models_c \neg\phi$ iff it is not the case that $M, \pi \models_c \phi$*

4. ***"Classical" Disjunction:*** *$M, \pi \models_c \phi \vee \psi$ iff $M, \pi \models_c \phi$ or $M, \pi \models_c \psi$.*

5. ***"Classical" Conjunction:*** *$M, \pi \models_c \phi \wedge \psi$ iff $M, \pi \models_c \phi$ and $M, \pi \models_c \psi$.*

6. ***Serial Conjunction:*** *$M, \pi \models_c \phi \otimes \psi$ iff there exists a prefix $\pi'$ of $\pi$ and some split $\pi_1 \circ \pi_2$ of $\pi'$ such that $M, \pi_1 \models_c \phi$, $M, \pi_2 \models_c \psi$ and $\pi$ is a classical expansion of path $\pi'$ w.r.t. $M$.*

To illustrate the previous definitions, consider the following example.

**Example 46** (Classical Satisfaction)**.** *Recall the rules defined in example 45, assume an internal oracle based on relational oracle, and an external oracle where $\mathcal{O}(s_1, s_6) \models c$, for the external states $s_1, s_6$.*

*Regarding the classical satisfaction of formula $q.ins \otimes e_1$ we have:*

$$M, \langle(\{\}, s_1) \xrightarrow{q.ins} (\{q\}, s_1)\rangle \models_c q.ins$$

*And, since $\mathbf{o}(e_1) \in \langle(\{q\}, s_1) \xrightarrow{\mathbf{o}(e_1)} (\{q\}, s_1)\rangle$:*

$$M, \langle(\{q\}, s_1) \xrightarrow{\mathbf{o}(e_1)} (\{q\}, s_1)\rangle \models_{ev} \mathbf{o}(e_1)$$

*Consequently, on the latter path, $e_1$ occurs and is not responded to. Since $M, \langle(\{q\}, s_1) \xrightarrow{\mathbf{ext}(c,c_1)} (\{q\}, s_6)\rangle \models_c \mathbf{r}(e_1)$, then:*

$$\langle(\{q\}, s_1) \xrightarrow{\mathbf{o}(e_1)} (\{q\}, s_1) \xrightarrow{\mathbf{ext}(c,c_1)} (\{q\}, s_6)\rangle \text{ is a classical expansion of path } \langle(\{q\}, s_1) \xrightarrow{\mathbf{o}(e_1)} (\{q\}, s_1)\rangle \text{ and } M, \langle(\{q\}, s_1) \xrightarrow{\mathbf{o}(e_1)} (\{q\}, s_1) \xrightarrow{\mathbf{ext}(c,c_1)} (\{q\}, s_6)\rangle \models_c e_1$$

*Finally:*

$$M, \langle(\{\}, s_1) \xrightarrow{q.ins} (\{q\}, s_1) \xrightarrow{\mathbf{o}(e_1)} (\{q\}, s_1) \xrightarrow{\mathbf{ext}(c,c_1)} (\{q\}, s_6)\rangle \models_c q.ins \otimes e_1$$

After defining the classical satisfaction of transaction formulas, and its related definitions, we now define the partial satisfaction relation. As before, we start by specifying how to expand paths when partially satisfying formulas.

**Definition 73** (Partial Expansion of a path)**.** *The path $\pi$ is a* partial expansion *of the path $\pi_1$ w.r.t. interpretation $M$ iff either one of the following is true:*

1. *$\pi$ is a classical expansion of path $\pi_1$ w.r.t. interpretation $M$; or*

2. *there is a sequence of paths $\pi_1, \ldots, \pi_j$, starting in $\pi_1$ and ending in $\pi_j$, such that each $\pi_i$ in the sequence is a* classical *response of $\pi_{i-1}$ w.r.t. $M$, $choice(M, \pi_j, \models_c) = e$ and there is a path $\pi'$ s.t. $\pi = \pi_j \circ \pi'$, $M, \pi' \models_p \mathbf{r}(e)$ but $M, \pi' \not\models_c \mathbf{r}(e)$; or*

3. *there is a path $\pi'$ such that $\pi = \pi_1 \circ \pi'$, $M, \pi' \models_p \mathbf{r}(e)$ but $M, \pi' \not\models_c \mathbf{r}(e)$.*

When compared to the classical expansion, the partial expansion of a path has the additional responsibility to detect failures during the execution of event responses, and to stop the evolution of a path whenever such a failure is detected. As such, $\pi$ is the partial expansion of path $\pi_1$ if $\pi$ is a classical expansion of path $\pi_1$, or if $\pi$ can be split as $\pi = \pi_1 \circ \pi_2 \circ \pi'$ where $\pi_2$ is a (possibly non-existent) sequence of paths obtained by classically responding to events in $\pi_1$ and $\pi'$ is a path obtained when trying to respond to events in $\pi_1 \circ \pi_2$.

It is important to note that the partial expansion definition always tries to execute responses according to the classical satisfaction relation. As a consequence, a partial expansion path can be seen as an extension of the classical expansion path, in the sense that all paths obtained by classical expansions are also obtained by partial expansions. This is reflected in the following lemma.

**Lemma 5.** *If path $\pi$ is a classical expansion of path $\pi_1$ w.r.t. interpretation $M$, then $\pi$ is also a partial expansion of $\pi_1$ w.r.t. $M$.*

*Proof.* Immediate consequence of definition 73. □

Based on the definition of partial expansion, we can define the partial satisfaction of transaction formulas as follows.

**Definition 74** (Partial Satisfaction of Transaction Formulas)**.** *Let $M$ be an interpretation, $\pi$ a path and $\phi$ a transaction formula. If $M(\pi) = \top$ then $M, \pi \models_p \phi$; otherwise:*

1. ***Base Case:** $M, \pi \models_p \phi$ iff $\phi$ is an atom and one of the following holds:*

   *(a)* $M, \pi \models_c \phi$

   *(b)* $M, \pi \not\models_c \phi$, $\phi \in \mathcal{L}_i$, $\pi = \langle (D, E) \rangle$ *and* $\neg \exists D_i$ *s.t.* $\phi \in M(\langle (D, E)^{\phi} \rightarrow (D_i, E) \rangle)$

   *(c)* $M, \pi \not\models_c \phi$, $\phi \in \mathcal{L}_a^*$, $\pi = \langle (D, E) \rangle$ *and* $\neg \exists E_i$ *s.t.* $\phi \in M(\langle (D, E)^{\phi} \rightarrow (D, E_i) \rangle)$

   *(d)* $\phi \in M(\pi_1)$ *and $\pi$ is a partial expansion of $\pi_1$ w.r.t. $M$.*

2. ***Event Case:** $M, \pi \models_p e$ iff $e \in \mathcal{P}_e$ and one of the following holds:*

   *(a)* $M, \pi \models_c e$

   *(b)* $\exists$ *split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1 \models_{ev} \mathbf{o}(e)$, and $\pi$ is a partial expansion of $\pi_1$ w.r.t. $M$.*

3. ***Negation:** $M, \pi \models_p \neg\phi$ iff it is not the case that $M, \pi \models_p \phi$*

4. ***"Classical" Disjunction:*** $M, \pi \models_p \phi \vee \psi$ *iff* $M, \pi \models_p \phi$ *or* $M, \pi \models_p \psi$
5. ***"Classical" Conjunction:*** $M, \pi \models_p \phi \wedge \psi$ *iff* $M, \pi \models_p \phi$ *and* $M, \pi \models_p \psi$
6. ***Serial Conjunction:*** $M, \pi \models_p \phi \otimes \psi$ *iff one of the following holds:*
   (a) $M, \pi \models_c \phi \otimes \psi$
   (b) $M, \pi \models_p \phi$ *and* $M, \pi \not\models_c \phi$
   (c) $\exists$ *split* $\pi_1 \circ \pi_2$ *of path* $\pi$ *s.t.* $M, \pi_1 \models_c \phi$, $M, \pi_2 \models_p \psi$ *and* $M, \pi_2 \not\models_c \psi$
   (d) $\exists$ *split* $\pi_1 \circ \pi_2$ *of path* $\pi'$ *s.t.* $M, \pi_1 \models_c \phi$, $M, \pi_2 \models_c \psi$ *and* $\pi$ *is a partial expansion of* $\pi'$ *w.r.t.* $M$.

The previous definition can be seen as an extension of $\mathcal{ETR}$'s partial satisfaction definition (cf. definition 14), but where the failure can also arise from the expansion. For instance, in the Base Case item 1, a formula is partially satisfied if: it is classically satisfied; if it cannot be executed from that given state; or if after executing that formula, some events are triggered and they cannot be responded to properly.

Similarly, a serial conjunction formula $\phi \otimes \psi$ is partially satisfied on a path $\pi$ if: it can be executed classically on that path (i.e., if nothing fails at all); if the execution of $\phi$ fails over $\pi$ (and in this case we stop the execution); if there is a split of $\pi$ where $\phi$ succeeds and $\psi$ fails (and in this case we stop the execution); if $\phi$ and $\psi$ succeed in splits of $\pi$ but an expansion needed to respond to the events that have become true because of this execution, fails.

As before, whenever a formula fails to be executed, then the (failed) execution is stopped immediately. In other words, even if other formulas were meant to be executed, we stop the execution of a (complex) formulas as soon as the failure occurs. In particular, if this failure happens before the expansion step, then the expansion is not calculated, as it makes little sense to trigger additional events for an execution that is known to fail.

**Example 47** (Partial Satisfaction)**.** *Recall example 45, and assume* $\mathcal{O}(s_1, s_2) \models a$ *and* $\forall s.\ \mathcal{O}(s_2, s) \not\models b$*. When partially satisfying the formula* $\mathbf{ext}(a, a_1 \otimes a_2) \otimes p.ins$ *we have for any interpretation* $M$*:*

$$M, \langle (\{\}, s_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, s_2) \rangle \models_c \mathbf{ext}(a, a_1 \otimes a_2)$$
$$M, \langle (\{\}, s_2) \xrightarrow{p.ins} (\{p\}, s_2) \rangle \models_c p.ins$$

*However, we also have that:*

$$M, \langle (\{\}, s_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, s_2) \xrightarrow{p.ins} (\{p\}, s_2) \rangle \models_{ev} \mathbf{o}(a)\ ;\ \mathbf{o}(p.ins)$$
$$M, \langle (\{\}, s_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, s_2) \xrightarrow{p.ins} (\{p\}, s_2) \rangle \models_{ev} \mathbf{o}(e_2)$$

*And thus we need to expand the path with the response of* $e_2$*, i.e., with the expansion that satisfies* $v.ins \otimes \mathbf{ext}(b, b_1)$*. Moreover, we have:*

$$M, \langle (\{p\}, s_2) \xrightarrow{v.ins} (\{p, v\}, s_2) \rangle \models_c v.ins$$

*However, since* $\forall s.\ \mathcal{O}(s_2, s) \not\models b$*:*

$$M, \langle(\{p, v\}, s_2)\rangle \models_p \mathbf{ext}(b, b_1)$$
$$M, \langle(\{p, v\}, s_2)\rangle \not\models_c \mathbf{ext}(b, b_1)$$

*And thus:*

$$\langle(\{\}, s_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, s_2) \xrightarrow{p.ins} (\{p\}, s_2) \xrightarrow{v.ins} (\{p, v\}, s_2)\rangle$$
*is a partial expansion of* $\langle(\{\}, s_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, s_2) \xrightarrow{p.ins} (\{p\}, s_2)\rangle$

*which makes us conclude:*

$$M, \langle(\{\}, s_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, s_2) \xrightarrow{p.ins} (\{p\}, s_2) \xrightarrow{v.ins} (\{p, v\}, s_2)\rangle \models_p \mathbf{ext}(a, a_1 \otimes a_2) \otimes p.ins$$

After a failure is known to be true, we need to check if some external actions were executed, and in that case, compensate for the failure. Like in $\mathcal{ETR}$, this behavior is captured by the notions of rollback path, $\mathtt{Seq}(\pi)$, and compensating path, which are defined in the usual way.

**Definition 75** (Rollback Path, and Sequence of External Actions)**.** *Let* $\pi$ *be a k-path of the following form* $\langle(D_1, E_1) \xrightarrow{A_1} (D_2, E_2) \xrightarrow{A_2} \ldots \xrightarrow{A_{k-1}} (D_k, E_k)\rangle$*. The* rollback *path of* $\pi$ *is the path obtained from* $\pi$ *by:*

1. *Replacing all* $D_i$*s by* $D_1$
2. *Keeping just the transitions where* $A_i \in \mathcal{L}_a^*$*.*

*The sequence of external actions of* $\pi$*, denoted* $\mathtt{Seq}(\pi)$*, is the sequence of actions of the form* $\mathbf{ext}(a, b_1 \otimes \ldots \otimes b_j)$ *that appear in the transitions of the rollback path of* $\pi$*.*

It is worth noting that, this definition of rollback path deletes from the original path, not only the transitions and states related to internal actions, but also the ones related to explicit event transitions. Recall that explicit events are transaction formulas that make the occurrence of an event explicitly true, and which may force the execution of event responses as a reaction. They can be seen as the internal action of triggering a given event, and thus, they are rolled back just like internal actions and internal events. As a consequence of this, only external actions (i.e. the actions belonging to $\mathcal{L}_a^*$) will appear in the rollback path.

**Example 48** (Rollback path)**.** *Let* $\pi_1 = \langle(\{\}, s_1) \xrightarrow{q.ins} (\{q\}, s_1) \xrightarrow{\mathbf{o}(e_1)} (\{q\}, s_1) \xrightarrow{\mathbf{ext}(c, c_1)} (\{q\}, s_6)\rangle$ *and* $\pi_2 = \langle(\{\}, s_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, s_2) \xrightarrow{p.ins} (\{p\}, s_2) \xrightarrow{v.ins} (\{p, v\}, s_2)\rangle$

$\pi_1' = \langle(\{\}, s_1) \xrightarrow{\mathbf{ext}(c, c_1)} (\{\}, s_6)\rangle$ *is the rollback path of* $\pi_1$ *and* $\pi_2' = \langle(\{\}, s_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, s_2)\rangle$ *is the rollback path of* $\pi_2$

*Moreover,* $\mathtt{Seq}(\pi_1') = \mathbf{ext}(c, c_1)$ *and* $\mathtt{Seq}(\pi_2') = \mathbf{ext}(a, a_1 \otimes a_2)$*.*

Subsequently, we define the notion of inversion, recovery path and compensating path for a transaction.

**Definition 76** (Inversion, and Recovery Path). *Let* $S = \langle \mathbf{ext}(A_1, A_1^{-1}), \ldots, \mathbf{ext}(A_n, A_n^{-1})\rangle$ *be a sequence of actions from* $\mathcal{L}_a^*$. *Then, the inversion of* $S$ *is the transaction formula* $\texttt{Inv}(S) = A_n^{-1} \otimes \ldots \otimes A_1^{-1}$.

$\pi_r$ *is a* recovery path *of* $\texttt{Seq}(\pi)$ *w.r.t.* $M$ *iff* $M, \pi_r \models_c \texttt{Inv}(\texttt{Seq}(\pi))$.

**Definition 77** (Compensating Path for a Transaction). *Let* $M$ *be an interpretation,* $\pi$ *a path and* $\phi$ *a formula.* $M, \pi \rightsquigarrow \phi$ *iff* all *the following hold:*

1. $\exists \pi_1$ *such that* $M, \pi_1 \models_p \phi$ *and* $M, \pi_1 \not\models_c \phi$
2. $\exists \pi_0$ *such that* $\pi_0$ *is the rollback path of* $\pi_1$
3. $\texttt{Seq}(\pi_1) \neq \emptyset$ *and* $\exists \pi_r$ *such that* $\pi_r$ *is a recovery path of* $\texttt{Seq}(\pi_1)$ *w.r.t.* $M$
4. $\pi_0$ *and* $\pi_r$ *are a split of* $\pi$, *i.e.* $\pi_c = \pi_0 \circ \pi_r$

Notice that, in the latter definition, we restrict the execution of some event responses when compensating for a formula. In particular, while all events triggered directly in the recovery path are addressed (due to the definition of $\models_c$), we do not address the events that may become true on the rollback path, nor on the path composed by the rollback path and the recovery path. The intuition is that, since as we are trying to restore consistency of a failed execution, a compensation should be seen a particular sensitive operation. As such, it is important to minimize the possibility of an execution to fail. Thus, we only address the events that directly depend on the execution of the compensation (according to the $\models_c$ relation), and leave the others to be addressed by the general satisfaction relation in definition 80.

**Example 49** (Compensating Path). *Recall examples 45, 47 and 48, and assume that* $\mathcal{O}(s_2, s_3) \models a_1$ *and* $\mathcal{O}(s_3, s_4) \models a_2$.

*Assume a path* $\pi_1$ *of the form:* $\pi_1 = \langle(\{\}, s_1)^{\mathbf{ext}(a, a_1 \otimes a_2)} \rightarrow (\{\}, s_2)^{p.ins} \rightarrow (\{p\}, s_2)^{v.ins} \rightarrow (\{p, v\}, s_2)\rangle$. *For* $\pi_1$, *we know that* $M, \pi_1 \models_p \mathbf{ext}(a, a_1 \otimes a_2) \otimes p.ins$ *and* $M, \pi_1 \not\models_c \mathbf{ext}(a, a_1 \otimes a_2) \otimes p.ins$

*Then, since* $\langle(\{\}, s_1)^{\mathbf{ext}(a, a_1 \otimes a_2)} \rightarrow (\{\}, s_2)\rangle$ *is the rollback path of* $\pi_1$, $\texttt{Inv}(\texttt{Seq}(\pi_1)) = a_1 \otimes a_2$ *and:* $M, \langle(\{\}, s_2)^{a_1} \rightarrow (\{\}, s_3)^{a_2} \rightarrow (\{\}, s_4)\rangle \models_c a_1 \otimes a_2$ *we can say that:*

$$M, \langle(\{\}, s_1)^{\mathbf{ext}(a, a_1 \otimes a_2)} \rightarrow (\{\}, s_2)^{a_1} \rightarrow (\{\}, s_3)^{a_2} \rightarrow (\{\}, s_4)\rangle \rightsquigarrow \mathbf{ext}(a, a_1 \otimes a_2) \otimes p.ins$$

Similar to $\mathcal{ETR}$, the previous classical and partial satisfaction relations are *not* used to satisfy formulas directly, but only as auxiliary functions. As before, to satisfy transaction formulas we rely on the notion of general satisfaction, in which it is possible to satisfy a formula even if some failures take place during its execution. Moreover, since $\mathcal{ETR}^{ev}$ is a reactive language, we need to ensure that formulas are only satisfied on paths where every occurring event is properly responded to. To achieve all this, we assume an additional notion of general path response and path expansion, defined in the usual way.

**Definition 78** (Path response)**.** *For a path $\pi_1$ and an interpretation $M$ we say that $\pi$ is a response of $\pi_1$ iff $choice(M, \pi_1, \models) = e$ and we can split $\pi$ into $\pi_1 \circ \pi_2$ such that $M, \pi_2 \models \mathbf{r}(e)$.*

**Definition 79** (Expansion of a path)**.** *The path $\pi$ is an* expansion *of the path $\pi_1$ w.r.t. interpretation $M$ iff:*

- *$\pi$ is completely answered w.r.t. $M$ and satisfaction relation $\models$, and*
- *either $\pi = \pi_1$; or there is a sequence of paths $\pi_1, \ldots, \pi$, starting in $\pi_1$ and ending in $\pi$, such that each $\pi_i$ in the sequence is a response of $\pi_{i-1}$ w.r.t. $M$.*

Based on the previous definitions, we can now make precise the general satisfaction of formulas.

**Definition 80** (General Satisfaction of Transaction Formulas)**.** *Let $M$ be a interpretation, $\pi$ a path and $\phi$ a transaction formula. If $M(\pi) = \top$ then $M, \pi \models_c \phi$; otherwise:*

1. ***Base Case:*** *$M, \pi \models p$ if there exists a prefix $\pi'$ of $\pi$ such that $p \in M(\pi')$ and $\pi$ is an expansion path of $\pi'$ w.r.t. $M$, for every transaction atom $p$ such that $p \notin \mathcal{P}_e$.*
2. ***Event Case:*** *$M, \pi \models e$ if $e \in \mathcal{P}_e$, $M, \pi' \models_{ev} \mathbf{o}(e)$ and $\pi$ is an expansion path of $\pi'$ w.r.t. $M$.*
3. ***Negation:*** *$M, \pi \models \neg\phi$ if it is not the case that $M, \pi \models \phi$*
4. ***"Classical" Disjunction:*** *$M, \pi \models \phi \vee \psi$ if $M, \pi \models \phi$ or $M, \pi \models \psi$.*
5. ***"Classical" Conjunction:*** *$M, \pi \models \phi \wedge \psi$ iff $M, \pi \models \phi$ and $M, \pi \models \psi$.*
6. ***Serial Conjunction:*** *$M, \pi \models \phi \otimes \psi$ if there exists a prefix $\pi'$ of $\pi$ and some split $\pi_1 \circ \pi_2$ of $\pi'$ such that $M, \pi_1 \models \phi$ and $M, \pi_2 \models \psi$ and $\pi$ is an expansion path of $\pi'$ w.r.t. $M$.*
7. ***Compensating Case:*** *$M, \pi \models \phi$ if $M, \pi'_1 \rightsquigarrow \phi$, $\pi_1$ is an expansion path of $\pi'_1$ w.r.t. $M$, and $M, \pi_2 \models \phi$ for some split $\pi_1 \circ \pi_2$ of $\pi_i$ s.t. $\pi$ is an expansion path of $\pi_i$ w.r.t. $M$.*
8. *For no other $M, \pi, \phi$, $M, \pi \models \phi$.*

Like in $\mathcal{ETR}$, the main difference of the latter definition, when compared to classical satisfaction, is the inclusion of the compensating case (item 7). In it, $M, \pi \rightsquigarrow \phi$ means that, in the *failed* attempt to execute $\phi$, a sequence of external actions was performed, but since it is impossible to roll back to the point before the execution of these actions, consistency is ensured by performing a sequence of compensating external actions in the reverse order (if compensating actions are defined for those actions). In this case, the internal state is rolled back (i.e., the initial state before the execution of the transaction is restored), and all external actions are compensated.

Additionally, also in item 7, since complex events may become true due to the composition of the compensating path and the path that satisfies a formula, we need also to

expand that path, in order to ensure that every occurring event is responded to. Moreover, note that we may expand such path twice: first to address the events that become true in the consistency preserving path (i.e., in $M, \pi'_1 \rightsquigarrow \phi$), and then to address the events that become true in the final path i.e., in the path composed by the failed attempt to satisfy a formula, and the successful execution started after that attempt.

**Example 50** (General Satisfaction). *Recall examples 45, 47, 48 and 49, and assume* $\mathcal{O}(s_4, s_6) \models c$. *We have:*

$$M, \langle (\{\}, s_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, s_2) \xrightarrow{a_1} (\{\}, s_3) \xrightarrow{a_2} (\{\}, s_4) \rangle \rightsquigarrow \mathbf{ext}(a, a_1 \otimes a_2) \otimes p.ins$$

*Additionally, since:*

$$M, \langle (\{\}, s_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, s_2) \xrightarrow{a_1} (\{\}, s_3) \xrightarrow{a_2} (\{\}, s_4) \rangle \models_{ev} \mathbf{o}(e_3)$$

*we need to expand the path with the response of the event* $e_3$. *Since:*

$$M, \langle (\{\}, s_4) \xrightarrow{s.ins} (\{s\}, s_4) \rangle \models \mathbf{r}(e_3)$$

*And:*

$$M, \langle (\{s\}, s_4) \xrightarrow{q.ins} (\{s, q\}, s_4) \xrightarrow{\mathbf{o}(e_1)} (\{s, q\}, s_4) \xrightarrow{\mathbf{ext}(c, c_1)} (\{s, q\}, s_6) \rangle \models q.ins \otimes e_1$$
$$M, \langle (\{s\}, s_4) \xrightarrow{q.ins} (\{s, q\}, s_4) \xrightarrow{\mathbf{o}(e_1)} (\{s, q\}, s_4) \xrightarrow{\mathbf{ext}(c, c_1)} (\{s, q\}, s_6) \rangle \models t$$

*we have:*

$$M, \langle (\{\}, s_1) \xrightarrow{\mathbf{ext}(a, a_1 \otimes a_2)} (\{\}, s_2) \xrightarrow{a_1} (\{\}, s_3) \xrightarrow{a_2} (\{\}, s_4) \xrightarrow{s.ins} (\{s\}, s_4)(\{s\}, s_4) \xrightarrow{q.ins}$$
$$(\{s, q\}, s_4) \xrightarrow{\mathbf{o}(e_1)} (\{s, q\}, s_4) \xrightarrow{\mathbf{ext}(c, c_1)} (\{s, q\}, s_6) \rangle \models t$$

*Moreover since:*

$$M, \langle (\{\}, s_1) \xrightarrow{q.ins} (\{q\}, s_1) \xrightarrow{\mathbf{o}(e_1)} (\{q\}, s_1) \xrightarrow{\mathbf{ext}(c, c_1)} (\{q\}, s_6) \rangle \models q.ins \otimes e_1$$

*we also have:*

$$M, \langle (\{\}, s_1) \xrightarrow{q.ins} (\{q\}, s_1) \xrightarrow{\mathbf{o}(e_1)} (\{q\}, s_1) \xrightarrow{\mathbf{ext}(c, c_1)} (\{q\}, s_6) \rangle \models t$$

### 13.2.1 Event Choice Function

The *choice* function has the important feature of deciding for each path, if there are events considered unanswered and, if more than one event exist in this condition, to determine which one should be responded to first.

However, by definition, an event $e$ is unanswered on a path $\pi$, if $\mathbf{o}(e)$ occurs on a subpath $\pi_1$ of $\pi$ (w.r.t. the event satisfaction definition), and its response is not satisfied after the occurrence on a subpath $\pi_1$. Yet, whereas in $\mathcal{TR}^{ev}$ only one satisfaction relation

exists for transaction formulas, in $\mathcal{ETR}^{ev}$ we have three: classical, partial, and general satisfaction. As a result of this, defining a completely answered path depends on the satisfaction relation in question, since an event can be considered unanswered w.r.t. the classical satisfaction relation, but answered w.r.t. the general satisfaction relation.

Consequently, and to be useful for all definitions, the *choice* function abstracts the satisfaction relation, and assumes it as a parameter of the function. Apart from that, all the remaining definitions and decisions of the *choice* function in $\mathcal{ETR}^{ev}$ are exactly the same as the ones argued for $\mathcal{TR}^{ev}$ in section 11.3.

**Definition 81** (*choice* function)**.** *Let $M$ be an interpretation and $\pi$ be a path. Then function $choice(M, \pi)$ is defined as follows:*

$$choice(M, \pi, \texttt{ rel }) = firstUnans(M, \pi, order(M, \pi), \texttt{ rel })$$

Afterwards, we formalize what it means for a path to be completely answered w.r.t. an interpretation and a satisfaction relation `rel`:

**Definition 82** (Completely answered path)**.** *A path $\pi$ is said to be completely answered w.r.t. to an interpretation $M$ and satisfaction relation* `rel` *iff $choice(M, \pi, \texttt{ rel }) = \epsilon$.*

And finally, we can instantiate *firstUnans* just as in example 37:

**Example 51** (Answering Choices)**.** *Let $\pi$ be a path, $M$ an interpretation,* `rel` *a satisfaction relation and $\langle e_1, \ldots, e_n \rangle$ a sequence of events.*

**Relaxed Response** *$firstUnans(M, \pi, \langle e_1, \ldots, e_n \rangle) = e_i$ if $e_i$ is the first event in $\langle e_1, \ldots, e_n \rangle$ s.t. $\exists \pi'$ subpath of $\pi$ where $M, \pi' \models_{ev} \mathbf{o}(e)$ and $\neg \exists \pi''$ s.t. $\pi''$ is also a subpath of $\pi$, $\pi''$ is after $\pi'$ and $M, \pi''$* `rel` $\mathbf{r}(e)$.

**Explicit Response** *$firstUnans(M, \pi, \langle e_1, \ldots, e_n \rangle) = e_i$ if $e_i$ is the first event in $\langle e_1, \ldots, e_n \rangle$ s.t. $\exists \pi'$ subpath of $\pi$ where $M, \pi' \models_{ev} \mathbf{o}(e)$ and if $\exists \pi''$ subpath of $\pi$ that is after $\pi'$ where $M, \pi''$* `rel` $\mathbf{r}(e_i)$ *then $\exists \pi_1, \pi_2$ subpaths of $\pi$ and $\pi_2$ is after $\pi_1$ where $M, \pi_1 \models_{ev} \mathbf{o}(e_j)$, $M, \pi_2$* `rel` $\mathbf{r}(e_j)$*, $j < i$ and $\pi''$ starts before the ending of $\pi_2$.*

### 13.2.2 Models and Entailment

After defining how transaction formulas and events can be satisfied over paths, we can now define the notion of model of a formula and of a program in the standard way. In it, an interpretation is said to model a formula, if and only if it satisfy the formula in every possible path.

**Definition 83** (Model of a formula)**.** *An interpretation $M$ is a* model *of a transaction formula $\phi$ iff for every path $\pi$:*

$$M, \pi \models \phi$$

*$M$ is a model of a event formula $\psi$ iff for every path $\pi$:*

$$M, \pi \models_{ev} \psi$$

*A formula $\phi$ logically entails another formula $\psi$ ($\phi \models \psi$) if every model of $\phi$ is also a model of $\psi$.*

Based on the latter definition, we say that an interpretation is model of a program if it models every rule of the program, i.e., if it satisfies all the rules in the program in every possible path. An interpretation satisfies a rule if, whenever it satisfies the antecedent, it also satisfies the consequent.

**Definition 84** (Model of a Program)**.** *An interpretation $M$ is a model of an event rule $\psi \Rightarrow \phi$ ) iff for every path $\pi$, whenever $M, \pi \models_{ev} \psi$ then $M, \pi \models_{ev} \phi$*
*An interpretation $M$ models a transaction rule $head \leftarrow body$ iff for every path $\pi$:*

- *If $M, \pi \models body$ then $M, \pi \models head$ and;*
- *If $M, \pi \models_c body$ then $M, \pi \models_c head$ and;*
- *If $M, \pi \rightsquigarrow body$ then $M, \pi \rightsquigarrow head$*

*An interpretation $M$ is a model of a program $P$ if it models all its rules. In this case we write $M \models P$.*

*A program $P$ entails another program $P'$ ($P \models P'$) if all models of $P$ are models of $P'$. Two programs $P$ and $P'$ are equivalent iff $P \models P'$ and $P' \models P$.*

In the following we define the notion of executional entailment which can be used to talk about properties of a *particular* execution path. Additionally, and similarly to $\mathcal{TR}^{ev}$, the executional entailment is based on the notion of *minimal models*. Since we are talking about a specific execution path, care must be taken since satisfying a new occurrence onn a path may invalidate transaction formulas that were previously true. Consequently, adding a new rule to a program may make a formula that was previously satisfied on a path $\pi$ to be false on $\pi$.

To define these minimal models, we assume the same notion of minimal interpretations as defined in section 11.4.

**Definition 85** (Ordering of Structures)**.** *If $M_1$ and $M_2$ are interpretations then $M_1 \leq M_2$ if $\forall \pi$: $M_2(\pi) = \top \vee M_1(\pi) \subseteq M_2(\pi)$*

Based on the latter notion, a minimal model of a program $P$ is an interpretation that is a model of $P$, and that is minimal w.r.t. other comparable interpretations.

**Definition 86** (Minimal Model)**.** *Let $\phi$ be a (event or transaction) formula, and $P$ a program. $M$ is a* minimal model *of $\phi$ (resp. $P$) if $M$ is a model of $\phi$ (resp. $P$) and $M \leq M'$ for every model $M'$ of $\phi$ (resp. $P$).*

Finally, we can make precise the notion of execution entailment. Like in $\mathcal{TR}^{ev}$, to know if a formula succeeds in a particular path, only the event occurrences *supported* by that path are considered, either because they appear as occurrences in the transition of states, or because they are a necessary consequence of the program's rules given that path. This is formalized as follows.

**Definition 87** (Executional Entailment)**.** *Let $P$ be a program, $\phi$ a transaction formula and $D_0 \xrightarrow{O_1} \ldots \xrightarrow{O_n} D_n$ a path. Then the statement:*

$$P, (D_0 \xrightarrow{O_1} \ldots \xrightarrow{O_n} D_n) \models \phi \tag{13.1}$$

*holds iff for every minimal model $M$ of $P$, $M, \langle D_0 \xrightarrow{O_1} \ldots \xrightarrow{O_n} D_n \rangle \models \phi$. $P, D_0 - \models \phi$ is said to be true, if there is a path $D_0 \xrightarrow{O_1} \ldots \xrightarrow{O_n} D_n$ that makes (13.1) true.*

**Example 52** (Executional Entailment)**.** *In all of the previous examples 45, 47, 48, 49 and 50, we have talked about formulas that hold in any models of the following program:*

$$\begin{array}{ll}
t \leftarrow \mathbf{ext}(a, a_1 \otimes a_2) \otimes p.ins & \\
t \leftarrow q.ins \otimes e_1 & \\
 & \mathbf{r}(e_1) \leftarrow \mathbf{ext}(c, c_1) \\
\mathbf{o}(a) \ ; \ \mathbf{o}(p.ins) \Rightarrow \mathbf{o}(e_2) & \mathbf{r}(e_2) \leftarrow v.ins \otimes \mathbf{ext}(b, b_1) \\
\mathbf{o}(e_2) \ ; \ \mathbf{o}(c) \Rightarrow \mathbf{o}(e_3) & \mathbf{r}(e_4) \leftarrow \mathbf{ext}(d, d_1) \\
\mathbf{o}(a) \ ; \ \mathbf{o}(a_2) \Rightarrow \mathbf{o}(e_4) & \mathbf{r}(e_3) \leftarrow s.ins
\end{array}$$

*From what is showed in those examples, we have that for every model, and, thus, also for every minimal model $M$ of that program:*

$$M, \langle (\{\}, s_1) \xrightarrow{q.ins} (\{q\}, s_1) \xrightarrow{\mathbf{o}(e_1)} (\{q\}, s_1) \xrightarrow{\mathbf{ext}(c,c_1)} (\{q\}, s_6) \rangle \models t$$

*and*

$$\begin{aligned} M, \langle (\{\}, s_1) &\xrightarrow{\mathbf{ext}(a,a_1 \otimes a_2)} (\{\}, s_2) \xrightarrow{a_1} (\{\}, s_3) \xrightarrow{a_2} (\{\}, s_4) \xrightarrow{s.ins} (\{s\}, s_4)(\{s\}, s_4) \xrightarrow{q.ins} \\ &(\{s, q\}, s_4) \xrightarrow{\mathbf{o}(e_1)} (\{s, q\}, s_4) \xrightarrow{\mathbf{ext}(c,c_1)} (\{s, q\}, s_6) \rangle \models t \end{aligned}$$

*Thus, we can conclude that:*

$$P, ((\{\}, s_1) \xrightarrow{q.ins} (\{q\}, s_1) \xrightarrow{\mathbf{o}(e_1)} (\{q\}, s_1) \xrightarrow{\mathbf{ext}(c,c_1)} (\{q\}, s_6)) \models t$$

*and that:*

$$\begin{aligned} P, ((\{\}, s_1) &\xrightarrow{\mathbf{ext}(a,a_1 \otimes a_2)} (\{\}, s_2) \xrightarrow{a_1} (\{\}, s_3) \xrightarrow{a_2} (\{\}, s_4) \xrightarrow{s.ins} (\{s\}, s_4)(\{s\}, s_4) \xrightarrow{q.ins} \\ &(\{s, q\}, s_4) \xrightarrow{\mathbf{o}(e_1)} (\{s, q\}, s_4) \xrightarrow{\mathbf{ext}(c,c_1)} (\{s, q\}, s_6)) \models t \end{aligned}$$

## 13.3 Discussion

In this chapter we proposed $\mathcal{ETR}^{ev}$, a logic to reason about transactions simultaneously dealing with external actions and complex events, and which is based on a combination of $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$. Although $\mathcal{ETR}^{ev}$ should not be seen as the main contribution of this thesis, it still gives important pointers on how to model transactions that need to execute external actions, but also detect and react to complex events.

$\mathcal{ETR}^{ev}$ can be applicable in all of the scenarios of $\mathcal{TR}^{ev}$ and $\mathcal{ETR}$, although its theory is considerably more complex than any of its two based logics. This complexity arises from the need to consider additional points of failure, and consider additional sources of events. In this sense, in $\mathcal{ETR}^{ev}$ a formula $\phi$ can fail and be compensated if, $\phi$ cannot be executed starting from that state, or if after $\phi$ was executed, it is impossible to respond to all the events that become true because of the execution. Similarly, besides the events detected in $\mathcal{TR}^{ev}$, $\mathcal{ETR}^{ev}$ also detects atomic and complex events depending on the execution of external actions (even if these actions were executed in a failed execution). In fact, while events based on internal actions can be discarded because internal actions are rolled back after a failure, events based on external actions never disappear, even if they are compensated afterwards.

As a consequence of this, one important detail that arises from this combination is the handling of event patterns that become true because of the rollback and recovery step. In particular, it is important to note that, some complex events which were not true on the original failed path, may become true on the path obtained by the rollback operation. To illustrate this behavior, consider the following example.

**Example 53** (Events on rollback path)**.** *Imagine a program containing the following rules, based on a relational database internal oracle, and an external oracle where* $\mathcal{O}^e(e_1, e_2) \models a$, $\mathcal{O}^e(e_2, e_3) \models b$, *and* $\forall e.\mathcal{O}^e(e_3, e) \not\models c$.

$$t \leftarrow \mathbf{ext}(a, a_1) \otimes p.ins \otimes \mathbf{ext}(b, b_1) \otimes \mathbf{ext}(c, c_1)$$
$$t \leftarrow q.ins$$

$$\begin{array}{rll} \mathbf{o}(a) \otimes \mathbf{o}(b) & \Rightarrow \mathbf{o}(e_1) & \mathbf{r}(e_1) \leftarrow \mathbf{ext}(d, \mathtt{nop}) \\ \mathbf{not}(\mathbf{o}(p.ins))[a, b] & \Rightarrow \mathbf{o}(e_2) & \mathbf{r}(e_2) \leftarrow r.ins \end{array}$$

*With these rules for any minimal model of the program we have:*

$$M, \langle(\{\}, e_1) \xrightarrow{\mathbf{ext}(a,a_1)} (\{\}, e_2) \xrightarrow{p.ins} (\{p\}, e_2) \xrightarrow{\mathbf{ext}(b,b_1)} (\{p\}, e_3)\rangle \models_p t$$
$$\text{and } M, \langle(\{\}, e_1) \xrightarrow{\mathbf{ext}(a,a_1)} (\{\}, e_2) \xrightarrow{p.ins} (\{p\}, e_2) \xrightarrow{\mathbf{ext}(b,b_1)} (\{p\}, e_3)\rangle \not\models_c t$$

*Since for that path* $\pi = \langle(\{\}, e_1) \xrightarrow{\mathbf{ext}(a,a_1)} (\{\}, e_2) \xrightarrow{p.ins} (\{p\}, e_2) \xrightarrow{\mathbf{ext}(b,b_1)} (\{p\}, e_3)\rangle$ *we have* $M, \pi \models_p t$ *and* $M, \pi \not\models_c t$, *we can try to achieve a compensating path for* $t$ *based on* $\pi$, *in order to try to succeed transaction* $t$ *even after the action* $\mathbf{ext}(c, c_1)$ *failed.*

*To achieve this goal, we start by constructing the rollback path* $\pi_0$ *of* $\pi$, *where we have* $\pi_0 = \langle(\{\}, e_1) \xrightarrow{\mathbf{ext}(a,a_1)} (\{\}, e_2) \xrightarrow{\mathbf{ext}(b,b_1)} (\{\}, e_3)\rangle$

*However, note that both* $\mathbf{o}(e_1)$ *and* $\mathbf{o}(e_2)$ *are now true in* $\pi_0$, *although they were not true in the original path* $\pi$. *In fact,* $\mathbf{o}(e_1)$ *is true on a path where* $b$ *is executed* immediately *after* $a$, *whereas* $\mathbf{o}(e_2)$ *is true on a path where* $p.ins$ *does not occur in the interval defined by occurrences of* $a$ *and* $b$. *As such, both* $\mathbf{o}(e_1)$ *and* $\mathbf{o}(e_2)$ *are not true in* $\pi$ *because of the internal action* $p.ins$ *occurs. When we remove* $p.ins$ *from the path (because of the rollback), we obtain a different path* $\pi_0$, *capturing a different execution, and over which these two events become true.*

*As a result of this, if we want to execute $t$ after the failure, we need to respond to $e_1$ and $e_2$, and only after that we can try to succeed $t$. More precisely, if $\mathcal{O}^e(e_3, e_4) \models b_1$, $\mathcal{O}^e(e_4, e_5) \models a_1$ and $\mathcal{O}^e(e_5, e_6) \models d$ then, $t$ succeeds on the path:*

$$P, \langle (\{\}, e_1) \xrightarrow{\mathbf{ext}(a,a_1)} (\{\}, e_2) \xrightarrow{\mathbf{ext}(b,b_1)} (\{\}, e_3) \xrightarrow{b_1} (\{\}, e_4) \xrightarrow{a_1} (\{\}, e_5) \xrightarrow{\mathbf{ext}(d,\mathtt{nop})} (\{\}, e_6) \xrightarrow{r.ins} (\{r\}, e_6) \xrightarrow{q.ins} (\{r, q\}, e_6) \rangle \models t$$

*where we assume $\mathbf{o}(e_1)$ has higher priority over $\mathbf{o}(e_2)$.*

The previous example shows a somewhat unexpected behavior of events when combined with compensations, where event patterns can become true on paths resulting after an internal rollback, even if they were not true before the rollback. However, note that the compensating path is in fact the real outcome execution of a transaction, and all the internal actions rolled back must be considered as to have never happened (and invisible in the final path).

Additionally, one could argue whether it makes sense to detect and trigger event patterns based on an event that was compensated. However, unless we are using the mechanisms of chapter 7, and automatically computing the correct compensations for each action, there is no guarantee that a given compensation actually reverts the effects of the initial action. Moreover, it may be interesting for the programmer to write event patterns like:

$$\mathbf{o}(\mathbf{ext}(a, b)) \; ; \; \mathbf{o}(b) \Rightarrow \mathbf{o}(e)$$

where event $e$ is triggered if we execute the action $a$ followed by its compensation $b$. In other words, it may be interesting to define a complex event pattern triggered whenever an external action was compensated, so as to guarantee that some constraint is true in that after the compensation execution, or to execute some internal update based on that occurrence (and in this case, they these responses can be specified in the body of $\mathbf{r}(e)$).

Note that this behavior, where we can define complex events based on the execution of an external action and its compensation, is present not only in $\mathcal{ETR}^{ev}$, but in all solutions combining detection of event patterns (based on internal and external actions), with the possibility to compensate for external actions. This is an important detail, and one which the programmer needs to be aware of, and write her programs accordingly.

It is also important to stress that, as in $\mathcal{ETR}$, $\mathcal{ETR}^{ev}$ does not support the hypothetical construct, since it makes little sense to talk about hypothetical execution of actions involving external actions.

Moreover, $\mathcal{ETR}^{ev}$ is very powerful and flexible, as transactions can be compensated at any given time (including during the expansion). In addition, events can be triggered based on any (internal and external) action execution, including compensations. As such, the behavior where we fail to respond to an event, may cause the execution of compensations, which further cause the execution of more events and path expansions.

While this gives the logic an important and interesting flexibility, it also makes it considerably complex. To be useful in practice, this flexibility needs to be restricted,

and as such, in opposition to $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$, this result should not be seen as a "ready to use" logic, but as a basic framework combining transactions, reactivity and external actions, and which requires further investigation. Because of this, we have not developed a proof theory for $\mathcal{ETR}^{ev}$ and leave it as a future goal. A more extensive discussion on the possible future work can be found in section 14.4.

Notwithstanding, $\mathcal{ETR}^{ev}$ is an interesting first step when combining external actions with reactive transactions, providing a basic semantics for future (simpler) solutions to be compared with.

Since $\mathcal{ETR}^{ev}$ results from combination of $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$, it can be compared with all the solutions already discussed in chapter 8 and in chapter 12. To the best of our knowledge, $\mathcal{ETR}^{ev}$ is the only knowledge-based solution that address simultaneously complex events and transactions, where external actions are compensated in case of failure.

# Part V

# Wrapping up and Moving on

# 14

# Conclusions and Future Directions

In this final chapter we overview the main contributions of this thesis, and discuss the most interesting lines of related future research.

In the previous parts, we presented several results to achieve transaction properties over the execution of actions in reactive dynamic domains that have an internal and external component. In part II we defined *External Transaction Logic*, an extension of Transaction Logic to deal with transactions involving both an internal and external component. Then in part III, we addressed the problem of combining events and reactive features with transactions properties, proposing *Transaction Logic with Events*, an extension of Transaction Logic to reason and execute transactions that need to react to complex events. Finally, in part IV we showed that the proposal made in parts II and III can be put together in order to define a unified logic that can capture both the ability of executing transactions involving internal and external domains, and the ability to detect and react to complex events.

In the following sections we elaborate on the conclusions drawn from each of our results. Then, in section 14.4, we point out and discuss desirable future developments.

## 14.1 Transactions involving an internal and external component

In chapter 5 we provided an extension of Transaction Logic to model and execute abstract transactions involving the execution of actions in environments with an internal and external component. The main problem with such hybrid environments is that, internal and external actions can achieve different transactional properties, depending on where they are executed. In particular, while traditionally, transaction properties over internal actions are ensured by rollback mechanisms which revert the knowledge base to

the state before the execution; external actions cannot be rolled back since they are executed in a component that is not fully controlled. To overcome this, *External Transaction Logic* ($\mathcal{ETR}$) combines the possibility of rolling back internal actions, and compensate for external actions, capturing this behavior in the logic.

$\mathcal{ETR}$'s model theory may reason about all the possible paths where a transaction formula succeeds (in an all-or-nothing manner) given a specific program $P$, and about the properties that are true in all of these executions. Importantly, possible execution paths include the ones where a transaction formula succeeds without failures, but also the ones where a failure occurs, is recovered (by internal rollbacks and external compensations), and the transaction succeeds after the recovery.

In addition, $\mathcal{ETR}$ also provides the notion of executional entailment, where it can talk and reason about *a* particular path of execution. For this latter notion, we have also provided a sound and complete proof procedure which can be used to *construct* such a path, giving support for an implementation to execute transactions according to $\mathcal{ETR}$ semantics.

Both $\mathcal{ETR}$'s theory and procedure receive as a parameter three mapping functions, called *oracles*, which define the meaning of internal and external states and the set of possible primitives that can be executed in each state and transitions of states. This parametrization makes $\mathcal{ETR}$ flexible enough to be used in a wide set of domains. The logic itself does not commit to any particular semantics of state change, centering on how can formulas execute *transactionally* based on the specification of states received as a parameter.

An important result of $\mathcal{ETR}$ is the formal definition of a formula *legitimately* failing over a path, and recovering from this failure. These notions are essential, because there are several reasons for a formula to fail to be executed over a path, and which are not interesting to be recovered. Namely, a formula can fail over a path because the path does not represent a valid execution try for that transaction formula. As such, a legitimate fail path is one where we really tried to execute the transaction, but this execution was impossible according to the oracles. Building upon this notion, we say that a transaction legitimately fails over a path, if the transaction needs to execute a primitive action (or query) in a state, and this primitive fails according to the specification of the oracles. More precisely, because there is not a possible transition from the current state that satisfies the primitive action according to the oracle; or because the primitive is not true in the current state (in the case of queries).

To capture this failed behavior, $\mathcal{ETR}$ provides two auxiliary definitions – the partial and classical satisfaction relations (definition 13 and definition 14) – and a formula is said to legitimately fail if it is partially satisfied on a path, over which it is not classically satisfied. Afterwards, a formula can still succeed on a path after a failure, if we can compensate externally, rollback the internal state, and succeed in an alternative execution, started after the recovery.

To illustrate how $\mathcal{ETR}$'s new external oracle can be used in different scenarios, in

chapter 6 we explored several possibilities and provided external oracle instantiations for Description Logics, Action Languages, Situation Calculus and Event Calculus. Building upon those, we exemplified how $\mathcal{ETR}$ can be applicable in the Semantic Web context, where the external component is defined by a given Description Logic; and also in intelligent agents, where the external world is described by Action Languages, Situation Calculus or Event Calculus.

An important aspect of $\mathcal{ETR}$ is that compensations are assumed to be explicitly provided by the programmer. As a result, it is the programmer's responsibility to know what actions reverse the effects of the original executed actions, at each point in the program. However, this may be a too big a burden, as the programmer may not have enough knowledge to predict all the possible scenarios, and be ending up defining compensations that do not revert the effect of the executed actions, unintentionally. To address this, in chapter 7 we explored the possibility of automatically computing such compensations, and showed how this can be done for a particular external oracle instantiation. This result is provided for the case where the external oracle is instantiated with the action language $\mathcal{C}$, building upon the notion of action reversals of [EEF08]. Besides saving the programmer from the need to specify all the compensations, this approach also ensures that the executed compensations can indeed correctly revert the effects of actions. In addition, to cater for the situations where such compensating actions do not exist (because they are simply not reversible), in chapter 7 we also provided the notion of *goal reverse* that still tries to achieve a state where a given goal formula is true.

Due to all of its characteristics, $\mathcal{ETR}$ has shown to be an important first step to achieve transactional properties in dynamic reactive domains that have an internal and external component. However, $\mathcal{ETR}$ is still missing the ability to detect complex changes in the domain, and to react automatically to them. This led us to explore, in part III, what happens when we combine transactions and reactive features, and how Transaction Logic can be extended to handle this behavior.

## 14.2 Transactions with complex reactive features

In chapters 9 and 10, we investigated how to combine events and transactions. In particular, in chapter 10 we showed how the original Transaction Logic can be used to reason about what events are true on what paths. For that, we provided translations from the expressive event algebras SNOOP [AC06] and ETALIS [AFRSSS10] into Transaction Logic, showing that, if we exclude the expressions requiring explicit reference of time points, Transaction Logic can express and model the same event expressions as these algebras (theorems 8 and 9).

While these results showed that Transaction Logic can indeed be used to talk and reason about complex events and transactions over paths, in chapter 10 we also demonstrated that, Transaction Logic cannot deal with events and transactions simultaneously. In fact, when combined with reactivity, transactions need further to guarantee that every

event that occurs is properly responded to within the transaction itself. This behavior is observed e.g. in active databases, where a transaction fails if it cannot execute the triggers that become true during the execution. Similarly, Transaction Logic cannot handle reactive transactions, because it is not possible to force a transaction to respond to an event whenever its occurrence is detected over a path.

To overcome this, in chapter 11 we define *Transaction Logic with Events* ($\mathcal{TR}^{ev}$), which extends Transaction Logic with reactivity. As intended, $\mathcal{TR}^{ev}$ guarantees that transactions only succeed on paths where all occurring events are properly responded to. Its formulas are partitioned into event formulas and transaction formulas, and the satisfaction of transaction formulas over a path, w.r.t. a given interpretation, is dependent on what event formulas the interpretation makes true over the same path. Moreover, we imposed that whenever an event occurs and it is not responded to, then the path is *expanded* with its response.

As a consequence of this behavior, and in opposition to the original Transaction Logic, $\mathcal{TR}^{ev}$ is a non-monotonic logic, as adding new event rules to the program can make a formula to fail over a path where it was originally true, in the case a new event is triggered over that path and is not responded to.

$\mathcal{TR}^{ev}$ is also dependent on a pair of oracles defining the states and primitives of the knowledge base. Since primitive actions are also events that can be detected, then these oracles are also responsible for defining a subset of the atomic events that can be triggered. As any reactive language, $\mathcal{TR}^{ev}$ requires the definition of an operational behavior, responsible for selecting the next event to be responded, in case more than one event is unanswered at a given time. In order to maintain its flexibility, $\mathcal{TR}^{ev}$ is also parametric on a $choice$ function, that determines these operational choices. Following this line of research, in section 11.3, we explored how this function can be instantiated according to different reactive scenarios.

Moreover, in theorem 10 we showed that $\mathcal{TR}^{ev}$ is a conservative extension of Transaction Logic, proving the same transaction formulas over the same paths whenever the program does not contain event formulas.

Subsequently, besides providing a model theory to reason about every possible path of execution of reactive transactions, in section 11.5, we also provided a proof procedure to construct execution paths. Specified for a particular event $choice$ function, this proof procedure combines a bottom-up detection of event patterns, with a top-down execution of transactions, and can be used as basis to implement reactive transactions that execute according to the $\mathcal{TR}^{ev}$ semantics.

Finally, we illustrated how $\mathcal{TR}^{ev}$ can be used as an Event-Condition-Transaction language, modeling a similar behavior as that of event-condition-action rules (which can be considered the standard paradigm to encode reactivity), but where the action part behaves as a transaction.

While $\mathcal{TR}^{ev}$ can satisfactorily solve the problem of combining transactions and reactive features, it does not address the possibility to execute external actions in response

to events (even if the events arrive as external events) like $\mathcal{ETR}$. This made us combine $\mathcal{TR}^{ev}$ with $\mathcal{ETR}$ in one unified framework in part IV.

## 14.3 Reactive transactions involving internal and external actions

After developing two independent extensions of $\mathcal{TR}$ to handle the execution of external actions, and to integrate the possibility to react to complex events, we showed that these two extensions are orthogonal and compatible with each other, and explored how they can be combined in one unified logic.

With that in mind, in chapter 13 we developed *External Transaction Logic with Events*, a logic combining the theory of $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$, and which is able to guarantee for every path on which a transaction succeeds that every event triggered on a path is properly answered; and that in case of failure, consistency can be restored by executing compensations externally and rolling back the state internally.

The logic maintains the flexibility of $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$, by assuming three oracles ($\mathcal{O}^d$, $\mathcal{O}^t$ and $\mathcal{O}^e$), and a $choice$ function, as a parameter of the theory.

Like $\mathcal{TR}^{ev}$, to guarantee that every occurring event is properly responded to, this logic is non-monotonic, as adding a new event rule to a program may falsify transactions on paths where they were previously true, in case an unanswered event becomes true on that path because of the new rule. Like in $\mathcal{ETR}$, this logic formally defines what it means for a transaction formula to fail over a path in a legitimate way. However, now we have additional sources of failure. Namely, a formula can fail because some of its primitives could not be executed; or because it was not possible to answer all the events that have become true during the execution. In other words, a transaction formula can fail in a path, if some primitive fails to be executed, or because there is a failure when expanding the path.

Nevertheless, we still say that a transaction legitimately fails over a path, if it succeeds partially on that path, but not classically. In this case, we can still succeed after the failure if we rollback internally, compensate externally for the executed actions, and satisfy the formula on a path starting after the failure. An important detail, is that we need to cater also for the events that arise in the recovery path, i.e., on the path where we combine rolling back with compensations. In fact, since events can be defined based on primitive (internal and external) actions, we can define complex events based on the execution of an external action and its compensation, and respond to them in some way (e.g. by internally marking that a failure happened and was compensated).

While this logic should not be seen as the main contribution of this thesis, $\mathcal{ETR}^{ev}$ is still a very interesting first step by providing a powerful semantics combining transactions, reactivity and external actions.

## 14.4 Future Directions

The previously discussed results motivate further investigation in several different directions, which are summarized as follows:

**Achieving concurrent reactive transactions.** In our work, we assumed that each node/agent has only one process running at the time, and that he completely controls its own internal KB. However, whenever more than one process or agent are changing the KB by executing actions, it becomes important to model how these processes can execute concurrently.

Concurrency is a crucial property of computer systems, which allows several processes to be executed simultaneously, sharing the same resources, and interacting with each other. In practice, most of transaction systems implement concurrency control mechanisms to guarantee the correct execution of parallel transactions, e.g. to prevent two transactions to change the same tuple simultaneously (lost update problem), or to access data which is being concurrently modified by another transaction (dirty reads).

To capture the executional behavior of transactions in a concurrent setting, the authors of Transaction Logic have proposed Concurrent Transaction Logic [BK96]. This logic extends Transaction Logic theory with the notion of multi-paths, modeling the execution of a transaction which can be interleaved with other executions. In it, a transaction can execute over paths where it has continuous execution segments, interleaved with periods where it is suspended, to prevent conflicts with other transactions.

Extending any of our resulting logics with this notion of multi-paths is a natural and desirable improvement that would be interesting to address in the future. Nonetheless, we should note that while supporting concurrency is important when executing (trans)actions over an internal domain, as to allow reasoning about the execution of multiple processes (or agents); achieving concurrency over external actions is, in most cases, unreasonable. In particular, since we do not control the domain over which the actions are executed, it is unfeasible to reason about how these (external) resources can be shared, or how our external actions interact with external actions performed by other entities in the environment.

**Restricting event detection using history and time-windows.** In order to reason about what events happened in a given interval, both $\mathcal{TR}^{ev}$ and $\mathcal{ETR}^{ev}$ require the whole evolution of knowledge base states and transitions characterizing that interval. This implies that, while $P, \pi_1 \models \phi_1$ and $P, \pi_2 \models \phi_2$ may hold, the statement $P, \pi_1 \circ \pi_2 \models \phi_1 \otimes \phi_2$ may not[1]. Namely, there may be events arising from the combination of

[1]Note that the converse is always true, i.e., if $P, \pi \models \phi_1 \otimes \phi_2$ then there is a subpath of $\pi'$ of $\pi$ s.t. $P, \pi_1 \models \phi_1$, $P, \pi_2 \models \phi_2$ and $\pi' = \pi_1 \circ \pi_2$ (cf. lemma 4).

$\pi_1 \circ \pi_2$ that still need to be responded to on a path $\pi'$ (and in that case $P, \pi_1 \circ \pi_2 \circ \pi' \models \phi_1 \otimes \phi_2$ holds); or worse, there may not be such a path $\pi'$ where these events can be responded given the program $P$, and $\phi_1 \otimes \phi_2$ fails. Consequently, whenever we want to execute $\phi_2$, but taking into account the events that happened in the past execution of $\phi_1$, we have to store the knowledge about the execution of $P, \pi_1 \models \phi_1$ and use this to execute $\phi_1 \otimes \phi_2$.

However, in practice, it is unreasonable to keep all the information about these executions, and about all the events that happened in the past, especially when considering event-processing systems where events arrive as an avalanche of data. To deal with this limitation, complex event-processing systems employ the notion of sliding window. This window defines a boundary (in time or memory) on which events are processed, thereby restricting the amount data to be computed. Events occurring outside of this window are discarded.

This encourages us to extend $\mathcal{TR}^{ev}$ (and $\mathcal{ETR}^{ev}$) with the notion of event windows, or event history, to minimize the amount of information to be stored, and using it to talk about executions that take into account a bounded event history. With it, one could e.g. define statements of the form: $P, \pi \models^H \phi$ where $\phi$ is said to execute over a path $\pi$, given a program $P$, and a *history* $H$ containing a window of events that have occurred in the past.

With this notion, one could have a property stronger than the one of lemma 4. This property would ensure that the results would be the same if: (1) one completely stops the execution after responding to all available events, and resumes it only after new events appear; or (2) the execution never stops. More precisely, it would allows us to say that if $P, \pi_1 \models \phi_1$, and $P, \pi_2 \models^{H_1} \phi_2$, then $P, \pi_1 \circ \pi_2 \models \phi_1 \otimes \phi_2$, where $H_1$ is the history of events occurring in $\pi_1$, and which are taking into account in the execution of $\phi_2$.

Nevertheless, we should stress that such a behavior is already achieved by $\mathcal{TR}^{ev}$'s procedure. In it, we already store the history of what events occurred by adding temporary rules to the program. Consequently, if $\phi_1$ successfully executes over a program $P$ and path $\pi_1$, changing it into $P'$, then if we execute $\phi_2$ over $P'$ starting in the end path of $\pi_1$, we will achieve the same results as executing $\phi_1 \otimes \phi_2$ over $P$ and starting in the first state of $\pi_1$ (cf. theorem 12).

**Predicting termination and non-termination.** Termination stands for the ability to guarantee that any given execution of a program will definitely terminate. Clearly, $\mathcal{TR}$ is a Turing-complete language, and thus by Turing's halting problem we know that this problem is undecidable in the general case.

Nevertheless, this is an important problem in reactive languages, because the cascading behavior where events can be triggered during their response execution may easily lead to a non-termination behavior, which may not be at all trivial to detect.

This demands for a further investigation on the subclasses of programs that can be considered safe and unsafe for execution. Moreover, since termination depends on the underlying execution model, then this means that in $\mathcal{TR}^{ev}$ it also depends on the instantiation of the *choice* function, and its role should be further explored and examined in this context.

**Restricting External Transaction Logic with Events.** As proposed, $\mathcal{ETR}^{ev}$ has an interesting flexibility which is hard to handle in practice. In particular, the logic allows non-deterministic recovery at any point (even after a compensations were executed) and to any point back (even the beginning of a very long transaction).

In real scenarios this behavior is impractical and demands for further investigation on how to restrict the execution of $\mathcal{ETR}^{ev}$ formulas. In this context, it would be interesting to explore the possibility of partially committing changes of subtransactions, and to model the behavior of practical mechanisms like savepoints to prevent the rollback of the entire transaction.

**Proof Procedure for executing $\mathcal{ETR}^{ev}$ programs.** Like in $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$, the theory of $\mathcal{ETR}^{ev}$ demands the development of procedures to execute transactions that react to events and execute external actions, according $\mathcal{ETR}^{ev}$'s theory. Nevertheless, given the flexible characteristics of $\mathcal{ETR}^{ev}$, it probably makes sense to provide such procedures for restricted version of the logic only.

**Implementations.** The proof procedures provided for $\mathcal{ETR}$ and $\mathcal{TR}^{ev}$ can be seen as an important backbone to write implementations to execute transactions that include external actions and react to events. To develop such implementations, the works of [FK10] and [ARFS12] are interesting starting points, the latter in the context of detecting complex events.

**Relation with multi-context systems.** Multi-context systems are a popular framework to represent knowledge partitioned over several contexts using several semantics. Given the similarities of multi-context systems and $\mathcal{ETR}$, it might be interesting to explore if and how multi-context systems could be embedded in $\mathcal{ETR}$, with the goal to obtain transaction properties in these scenarios.

In summary, our results show how transactional properties can be achieved over reactive and dynamic environments requiring the interaction with an internal and external component. This is done by proposing two different logics that handle reactive transactions and external transaction execution of actions in abstract settings. On the other hand, the general problem of combining reactive transactions with external actions execution is complex, and requires further exploration and restrictions to be useful in practice. In this thesis we achieved a first approximation in ensuring transactional properties in this context, in an interesting and unified logical framework.

# Bibliography


[Abr05] S. Abramsky. "A structural approach to reversible computation". In: *Theor. Comput. Sci.* 347.3 (2005), pp. 441–464. DOI: 10.1016/j.tcs.2005.07.002. URL: http://dx.doi.org/10.1016/j.tcs.2005.07.002.

[AC05] R. Adaikkalavan and S. Chakravarthy. "Formalization and Detection of Events Using Interval-Based Semantics". In: *COMAD*. Ed. by J. R. Haritsa and T. M. Vijayaraman. Computer Society of India, 2005, pp. 58–69.

[AC06] R. Adaikkalavan and S. Chakravarthy. "SnoopIB: Interval-based event specification and detection for active databases". In: *Data Knowl. Eng.* 59.1 (2006), pp. 139–165.

[ADGI08] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. "Efficient pattern matching over event streams". In: *SIGMOD Conference*. Ed. by J. T.-L. Wang. ACM, 2008, pp. 147–160. ISBN: 978-1-60558-102-6.

[ABB11] J. J. Alferes, F. Banti, and A. Brogi. "Evolving reactive logic programs". In: *Intelligenza Artificiale* 5.1 (2011), pp. 77–81.

[APP98] J. J. Alferes, L. M. Pereira, and T. C. Przymusinski. "`Classical' Negation in Nonmonotonic Reasoning and Logic Programming". In: *J. Autom. Reasoning* 20.1 (1998), pp. 107–142. DOI: 10.1023/A:1005900924623. URL: http://dx.doi.org/10.1023/A:1005900924623.

[Ani11] D. Anicic. "Event Processing and Stream Reasoning with ETALIS". PhD thesis. Karlsruher Institut für Technologie, 2011.

[AFRSSS10] D. Anicic, P. Fodor, S. Rudolph, R. Stühmer, N. Stojanovic, and R. Studer. "A Rule-Based Language for Complex Event Processing and Reasoning". In: *RR*. Ed. by P. Hitzler and T. Lukasiewicz.

Vol. 6333. Lecture Notes in Computer Science. Springer, 2010, pp. 42–57. ISBN: 978-3-642-15917-6.

[AFSS09a] D. Anicic, P. Fodor, N. Stojanovic, and R. Stühmer. "An approach for data-driven and logic-based complex Event Processing". In: *DEBS*. Ed. by A. S. Gokhale and D. C. Schmidt. ACM, 2009. ISBN: 978-1-60558-665-6.

[AFSS09b] D. Anicic, P. Fodor, R. Stühmer, and N. Stojanovic. "Event-Driven Approach for Logic-Based Complex Event Processing". In: *CSE (1)*. IEEE Computer Society, 2009, pp. 56–63.

[ARFS12] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic. "Stream reasoning and complex event processing in ETALIS". In: *Semantic Web* 3.4 (2012), pp. 397–407.

[ASP09] A. Artikis, M. J. Sergot, and J. V. Pitt. "Specifying norm-governed computational societies". In: *ACM Trans. Comput. Log.* 10.1 (2009).

[AG11] H. B. Axelsen and R. Glück. "A Simple and Efficient Universal Reversible Turing Machine". In: *Language and Automata Theory and Applications - 5th International Conference, LATA 2011, Tarragona, Spain, May 26-31, 2011. Proceedings*. Ed. by A. H. Dediu, S. Inenaga, and C. Martín-Vide. Vol. 6638. Lecture Notes in Computer Science. Springer, 2011, pp. 117–128. ISBN: 978-3-642-21253-6. DOI: 10.1007/978-3-642-21254-3_8. URL: http://dx.doi.org/10.1007/978-3-642-21254-3_8.

[AGY07] H. B. Axelsen, R. Glück, and T. Yokoyama. "Reversible Machine Code and Its Abstract Processor Architecture". In: *Computer Science - Theory and Applications, Second International Symposium on Computer Science in Russia, CSR 2007, Ekaterinburg, Russia, September 3-7, 2007, Proceedings*. Ed. by V. Diekert, M. V. Volkov, and A. Voronkov. Vol. 4649. Lecture Notes in Computer Science. Springer, 2007, pp. 56–69. ISBN: 978-3-540-74509-9. DOI: 10.1007/978-3-540-74510-5_9. URL: http://dx.doi.org/10.1007/978-3-540-74510-5_9.

[BCMNPS03] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003. ISBN: 0-521-78176-0.

[BLMSW05] F. Baader, C. Lutz, M. Milicic, U. Sattler, and F. Wolter. "Integrating Description Logics and Action Formalisms: First Results". In: *AAAI*. 2005, pp. 572–577.

[BDR04] J. Bailey, G. Dong, and K. Ramamohanarao. "On the decidability of the termination problem of active database systems". In: *Theor. Comput. Sci.* 311.1-3 (2004), pp. 389–437.

[BLT97] C. Baral, J. Lobo, and G. Trajcevski. "Formal Characterizations of Active Databases: Part II". In: *DOOD*. Ed. by F. Bry, R. Ramakrishnan, and K. Ramamohanarao. Vol. 1341. Lecture Notes in Computer Science. Springer, 1997, pp. 247–264. ISBN: 3-540-63792-3.

[Ben73] C. Bennett. "Logical Reversibility of Computation". In: *IBM Journal of Research and Development* 17.6 (1973), pp. 525–532. ISSN: 0018-8646. DOI: 10.1147/rd.176.0525.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. "The Semantic Web". In: *Scientific American* May (2001), pp. 29–37.

[BPV98] L. E. Bertossi, J. Pinto, and R. Valdivia. "Specifying Active Databases in the Situation Calculus". In: *SCCC*. IEEE Computer Society, 1998, pp. 32–39.

[BLZ03] L. Bocchi, C. Laneve, and G. Zavattaro. "A Calculus for Long-Running Transactions". In: *FMOODS*. Ed. by E. Najm, U. Nestmann, and P. Stevens. Vol. 2884. Lecture Notes in Computer Science. Springer, 2003, pp. 124–138. ISBN: 3-540-20491-1.

[BK93] A. J. Bonner and M. Kifer. "Transaction Logic Programming". In: *ICLP*. Ed. by D. S. Warren. MIT Press, 1993, pp. 257–279. ISBN: 0-262-73105-3.

[BK94] A. J. Bonner and M. Kifer. "Applications of Transaction Logic to Knowledge Representation". In: *ICTL*. Ed. by D. M. Gabbay and H. J. Ohlbach. Vol. 827. Lecture Notes in Computer Science. Springer, 1994, pp. 67–81. ISBN: 3-540-58241-X.

[BK95] A. J. Bonner and M. Kifer. *Transaction Logic Programming (or a logic of declarative and procedural knowledge)*. Tech. rep. CSRI-323. University of Toronto, 1995.

[BK96] A. J. Bonner and M. Kifer. "Concurrency and Communication in Transaction Logic". In: *JICSLP*. MIT Press, 1996, pp. 142–156.

[BK98a] A. J. Bonner and M. Kifer. "A Logic for Programming Database Transactions". In: *Logics for Databases and Information Systems*. Ed. by J. Chomicki and G. Saake. Kluwer, 1998, pp. 117–166.

[BK98b] A. J. Bonner and M. Kifer. "Results on Reasoning about Updates in Transaction Logic". In: *Transactions and Change in Logic Databases*. Springer, 1998, pp. 166–196.

[BKC93] A. J. Bonner, M. Kifer, and M. P. Consens. "Database Programming in Transaction Logic". In: *DBPL*. 1993, pp. 309–337.

[BWH07] R. H. Bordini, M. Wooldridge, and J. F. Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007. ISBN: 0470029005.

[BEP06] F. Bry, M. Eckert, and P.-L. Patranjan. "Reactivity on the Web: Paradigms and Applications of the Language XChange". In: *J. Web Eng.* 5.1 (2006), pp. 3–24.

[BHF04] M. J. Butler, C. A. R. Hoare, and C. Ferreira. "A Trace Semantics for Long-Running Transactions". In: *25 Years Communicating Sequential Processes*. Ed. by A. E. Abdallah, C. B. Jones, and J. W. Sanders. Vol. 3525. Lecture Notes in Computer Science. Springer, 2004, pp. 133–150. ISBN: 3-540-25813-2.

[CDGLLR07] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. "Tractable reasoning and efficient query answering in Description Logics: The DL-Lite family". In: *Journal of Automated reasoning* 39.3 (2007), pp. 385–429.

[CGLLR05] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. "DL-Lite: Tractable Description Logics for Ontologies". In: *AAAI*. 2005, pp. 602–607.

[CKNZ10] D. Calvanese, E. Kharlamov, W. Nutt, and D. Zheleznyakov. "Updating ABoxes in DL-Lite". In: *AMW*. Ed. by A. H. F. Laender and L. V. S. Lakshmanan. Vol. 619. CEUR Workshop Proceedings. CEUR-WS.org, 2010.

[Cat11] R. Cattell. "Scalable SQL and NoSQL data stores". In: *ACM SIGMOD Record* 39.4 (2011), pp. 12–27.

[Cha97] S. Chakravarthy. "SENTINEL: An Object-Oriented DBMS With Event-Based Rules". In: *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*. Ed. by J. Peckham. ACM Press, 1997, pp. 572–575. DOI: 10.1145/253260.253409. URL: http://doi.acm.org/10.1145/253260.253409.

[CKAK94] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. "Composite Events for Active Databases: Semantics, Contexts and Detection". In: *VLDB*. Ed. by J. B. Bocca, M. Jarke, and C. Zaniolo. Morgan Kaufmann, 1994, pp. 606–617. ISBN: 1-55860-153-8.

[CM94] S. Chakravarthy and D. Mishra. "Snoop: An Expressive Event Specification Language for Active Databases". In: *Data Knowl. Eng.* 14.1 (1994), pp. 1–26. DOI: 10.1016/0169-023X(94)90006-X. URL: http://dx.doi.org/10.1016/0169-023X(94)90006-X.

[CLN03] J. Chomicki, J. Lobo, and S. A. Naqvi. "Conflict Resolution Using Logic Programming". In: *IEEE Trans. Knowl. Data Eng.* 15.1 (2003), pp. 244–249.

[Cod70] E. F. Codd. "A Relational Model of Data for Large Shared Data Banks". In: *Commun. ACM* 13.6 (June 1970), pp. 377–387. ISSN: 0001-0782. DOI: 10.1145/362384.362685. URL: http://doi.acm.org/10.1145/362384.362685.

[CG13] S. Costantini and G. D. Gasperis. "Meta-level Constraints for Complex Event Processing in Logical Agents". In: *Informal Proc. of Commonsense 2013, 11th International Symposium on Logical Formalizations of Commonsense Reasoning*. 2013. URL: {http://www.commonsense2013.cs.ucy.ac.cy/program.html}.

[Cos12] S. Costantini. "Self-checking Logical Agents". In: *LA-NMR*. Ed. by M. Osorio, C. Zepeda, I. Olmos, J. L. Carballido, and R. C. M. Ramírez. Vol. 911. CEUR Workshop Proceedings. CEUR-WS.org, 2012, pp. 3–30.

[Cos13] S. Costantini. "Self-checking logical agents". In: *AAMAS*. Ed. by M. L. Gini, O. Shehory, T. Ito, and C. M. Jonker. IFAAMAS, 2013, pp. 1329–1330. ISBN: 978-1-4503-1993-5.

[CKV13] I. Cristescu, J. Krivine, and D. Varacca. "A Compositional Semantics for the Reversible p-Calculus". In: *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*. IEEE Computer Society, 2013, pp. 388–397. ISBN: 978-1-4799-0413-6. DOI: 10.1109/LICS.2013.45. URL: http://doi.ieeecomputersociety.org/10.1109/LICS.2013.45.

[DAAW06] C. V. Damásio, A. Analyti, G. Antoniou, and G. Wagner. "Supporting Open and Closed World Reasoning on the Web". In: *Principles and Practice of Semantic Web Reasoning, 4th International Workshop, PPSWR 2006, Budva, Montenegro, June 10-11, 2006, Revised Selected Papers*. Ed. by J. J. Alferes, J. Bailey, W. May, and U. Schwertel. Vol. 4187. Lecture Notes in Computer Science. Springer, 2006, pp. 149–163. ISBN: 3-540-39586-5. DOI: 10.1007/11853107_11. URL: http://dx.doi.org/10.1007/11853107_11.

[DK04] V. Danos and J. Krivine. "Reversible Communicating Systems". In: *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*. Ed. by P. Gardner and N. Yoshida. Vol. 3170. Lecture Notes in Computer Science. Springer, 2004, pp. 292–307. ISBN: 3-540-22940-X. DOI: 10.1007/978-3-540-28644-8_19. URL: http://dx.doi.org/10.1007/978-3-540-28644-8_19.

[DMG13] M Dastani, J.-J. C. Meyer, and D Grossi. "A logic for normative multi-agent programs". In: *J. Log. Comput.* 23.2 (2013), pp. 335–354.

[Das08] M. Dastani. "2APL: a practical agent programming language". In: *Autonomous Agents and Multi-Agent Systems* 16.3 (2008), pp. 214–248.

[Day88] U. Dayal. "Active Database Management Systems". In: *JCDKB*. 1988, pp. 150–169.

[DBBCHLMRSCLJ88] U. Dayal, B. T. Blaustein, A. P. Buchmann, U. S. Chakravarthy, M. Hsu, R. Ledin, D. R. McCarthy, A. Rosenthal, S. K. Sarin, M. J. Carey, M. Livny, and R. Jauhari. "The HiPAC Project: Combining Active Databases and Timing Constraints". In: *SIGMOD Record* 17.1 (1988), pp. 51–70.

[DHW95] U. Dayal, E. N. Hanson, and J. Widom. "Active Database Systems". In: *Modern Database Systems*. 1995, pp. 434–456.

[EEF08] T. Eiter, E. Erdem, and W. Faber. "Undoing the effects of action sequences". In: *J. Applied Logic* 6.3 (2008), pp. 380–415.

[EN10] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications Company, 2010, pp. I–XXIV, 1–360. ISBN: 978-1-935182-21-4.

[FWF99] A. Fent, C.-A. Wichert, and B. Freitag. "Logical Update Queries as Open Nested Transactions". In: *FMLDO - Selected Papers*. Ed. by G. Saake, K. Schwarz, and C. Türker. Vol. 1773. Lecture Notes in Computer Science. Springer, 1999, pp. 45–66. ISBN: 3-540-67201-X.

[FHH04] R. Fikes, P. Hayes, and I. Horrocks. "OWL-QL: a language for deductive query answering on the Semantic Web". In: *Web semantics: Science, services and agents on the World Wide Web* 2.1 (2004), pp. 19–29.

[FL79] M. J. Fischer and R. E. Ladner. "Propositional Dynamic Logic of Regular Programs". In: *J. Comput. Syst. Sci.* 18.2 (1979), pp. 194–211.

[FK10] P. Fodor and M. Kifer. "Tabling for transaction logic". In: *PPDP*. Ed. by T. Kutsia, W. Schreiner, and M. Fernández. ACM, 2010, pp. 199–208. ISBN: 978-1-4503-0132-9.

[FDKV98] B. Freitag, H. Decker, M. Kifer, and A. Voronkov, eds. *Transactions and Change in Logic Databases, International Seminar on Logic Databases and the Meaning of Change, Schloss Dagstuhl, Germany, September 23-27, 1996 and ILPS '97 Post-Conference Workshop on (Trans)Actions and Change in Logic Programming and Deductive Databases, (DYNAMICS'97) Port Jefferson, NY, USA, October 17, 1997, Invited Surveys and Selected Papers*. Vol. 1472. Lecture Notes in Computer Science. Springer, 1998. ISBN: 3-540-65305-8.

[Gab02] A. Gabaldon. "Non-Markovian Control in the Situation Calculus". In: *AAAI/IAAI*. Ed. by R. Dechter and R. S. Sutton. AAAI Press / The MIT Press, 2002, pp. 519–525.

[GMS87] H. Garcia-Molina and K. Salem. "Sagas". In: *SIGMOD Conference*. Ed. by U. Dayal and I. L. Traiger. ACM Press, 1987, pp. 249–259.

[GPP13] P. Gearon, A. Passant, and A. Polleres. *SPARQL Query Language for RDF*. W3C Recommendation. W3C Recommendation http://www.w3.org/TR/2013/REC-sparql11-update-20130321/. 2013.

[GRS91] A. V. Gelder, K. A. Ross, and J. S. Schlipf. "The Well-Founded Semantics for General Logic Programs". In: *J. ACM* 38.3 (1991), pp. 620–650.

[GL88] M. Gelfond and V. Lifschitz. "The Stable Model Semantics for Logic Programming". In: *ICLP/SLP*. Ed. by R. A. Kowalski and K. A. Bowen. MIT Press, 1988, pp. 1070–1080. ISBN: 0-262-61056-6.

[GL92] M. Gelfond and V. Lifschitz. "Representing Actions in Extended Logic Programming". In: *JICSLP*. Ed. by K. R. Apt. MIT Press, 1992, pp. 559–573. ISBN: 0-262-51064-2.

[GL98a] M. Gelfond and V. Lifschitz. "Action Languages". In: *Electron. Trans. Artif. Intell.* 2 (1998), pp. 193–210.

[GLPR09] G. D. Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. "On Instance-level Update and Erasure in Description Logic Ontologies". In: *J. Log. Comput.* 19.5 (2009), pp. 745–770.

[GLLMT04] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. "Nonmonotonic causal theories". In: *Artif. Intell.* 153.1-2 (2004), pp. 49–104.

[GL98b] E. Giunchiglia and V. Lifschitz. "An Action Language Based on Causal Explanation: Preliminary Report". In: *AAAI/IAAI*. Ed. by J. Mostow and C. Rich. AAAI Press / The MIT Press, 1998, pp. 623–630. ISBN: 0-262-51098-7.

[GA11] A. S. Gomes and J. J. Alferes. "Transaction Logic with External Actions". In: *LPNMR*. 2011, pp. 272–277.

[GA13a] A. S. Gomes and J. J. Alferes. "Extending Transaction Logic with External Actions". In: *TPLP* 13.4-5-Online-Supplement (2013).

[GA13b] A. S. Gomes and J. J. Alferes. "External Transaction Logic with Automatic Compensations". In: *CLIMA*. Ed. by J. Leite, T. C. Son, P. Torroni, L. van der Torre, and S. Woltran. Vol. 8143. Lecture Notes in Computer Science. Springer, 2013, pp. 239–255. ISBN: 978-3-642-40623-2.

[GA14a] A. S. Gomes and J. J. Alferes. "Combining Transactions and Automatic Repairs". In: *Journal of Logic and Computation* To Appear.CLIMA XIV's Special Issue (2014).

[GA14b] A. S. Gomes and J. J. Alferes. "Transaction Logic with (Complex) Events". In: *Theory and Practice of Logic Programming, On-line Supplement* To appear (2014).

[Gra80] J. Gray. "A Transaction Model". In: *ICALP*. Ed. by J. W. de Bakker and J. van Leeuwen. Vol. 85. Lecture Notes in Computer Science. Springer, 1980, pp. 282–298. ISBN: 3-540-10003-2.

[Gra81] J. Gray. "The Transaction Concept: Virtues and Limitations (Invited Paper)". In: *VLDB*. IEEE Computer Society, 1981, pp. 144–154.

[GHVD03] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. "Description Logic Programs: combining Logic Programs with Description Logic". In: *WWW*. Ed. by G. Hencsey, B. White, Y.-F. R. Chen, L. Kovács, and S. Lawrence. ACM, 2003, pp. 48–57. ISBN: 1-58113-680-3.

[HR83] T. Haerder and A. Reuter. "Principles of transaction-oriented database recovery". In: *ACM Computing Surveys (CSUR)* 15.4 (1983), pp. 287–317.

[HM87] S. Hanks and D. McDermott. "Nonmonotonic logic and temporal projection". In: *Artif. Intell.* 33.3 (Nov. 1987), pp. 379–412. ISSN: 0004-3702. DOI: 10.1016/0004-3702(87)90043-9. URL: http://dx.doi.org/10.1016/0004-3702(87)90043-9.

[HM86] S. Hanks and D. V. McDermott. "Default Reasoning, Nonmonotonic Logics, and the Frame Problem". In: *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, August 11-15, 1986. Volume 1: Science.* Ed. by T. Kehler. Morgan Kaufmann, 1986, pp. 328–333. URL: http://www.aaai.org/Library/AAAI/1986/aaai86-054.php.

[HKP82] D. Harel, D. Kozen, and R. Parikh. "Process Logic: Expressiveness, Decidability, Completeness". In: *J. Comput. Syst. Sci.* 25.2 (1982), pp. 144–170.

[HCO04] H. Hayashi, K. Cho, and A. Ohsuga. "A New HTN Planning Framework for Agents in Dynamic Environments". In: *CLIMA*. Ed. by J. Dix and J. A. Leite. Vol. 3259. Lecture Notes in Computer Science. Springer, 2004, pp. 108–133. ISBN: 3-540-24010-1.

[HBHM99] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. C. Meyer. "Agent Programming in 3APL". In: *Autonomous Agents and Multi-Agent Systems* 2.4 (1999), pp. 357–401.

[HV02] A. Hinze and A. Voisard. "A Parameterized Algebra for Event Notification Services". In: *TIME*. IEEE Computer Society, 2002, pp. 61–63. ISBN: 0-7695-1474-X.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985. ISBN: 0-13-153271-5.

[HPSBTGD+04] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, et al. "SWRL: A Semantic Web rule language combining OWL and RuleML". In: *W3C Member submission* 21 (2004), p. 79.

[HLM88] M. Hsu, R. Ladin, and D. R. McCarthy. "An Execution Model for Active Data Base Management Systems". In: *JCDKB*. 1988, pp. 171–179.

[KR03] I. Kiringa and R. Reiter. "A Unifying Semantics for Active Databases Using Non-Markovian Theories of Actions". In: *DBPL*. Ed. by G. Lausen and D. Suciu. Vol. 2921. Lecture Notes in Computer Science. Springer, 2003, pp. 110–129. ISBN: 3-540-20896-8.

[KCM04] G. Klyne, J. J. Carroll, and B. McBride. "Resource description framework (RDF): Concepts and abstract syntax". In: *W3C recommendation* 10 (2004).

[KAH11] M. Knorr, J. J. Alferes, and P. Hitzler. "Local closed world reasoning with description logics under the well-founded semantics". In: *Artif. Intell.* 175.9-10 (2011), pp. 1528–1554.

[Kow92] R. A. Kowalski. "Database Updates in the Event Calculus". In: *J. Log. Program.* 12.1&2 (1992), pp. 121–146.

[KS86] R. A. Kowalski and M. J. Sergot. "A Logic-based Calculus of Events". In: *New Generation Comput.* 4.1 (1986), pp. 67–95.

[KS09] J. Krämer and B. Seeger. "Semantics and implementation of continuous sliding window queries over data streams". In: *ACM Trans. Database Syst.* 34.1 (2009).

[KW14] M. Kutrib and T. Worsch. "Degrees of Reversibility for DFA and DPDA". In: *Reversible Computation - 6th International Conference, RC 2014, Kyoto, Japan, July 10-11, 2014. Proceedings*. Ed. by S. Yamashita and S. Minato. Vol. 8507. Lecture Notes in Computer Science. Springer, 2014, pp. 40–53. ISBN: 978-3-319-08493-0. DOI: 10.1007/978-3-319-08494-7_4. URL: http://dx.doi.org/10.1007/978-3-319-08494-7_4.

[LT88] K. G. Larsen and B. Thomsen. "A Modal Process Logic". In: *LICS*. IEEE Computer Society, 1988, pp. 203–210. ISBN: 0-8186-0853-6.

[LLM98] G. Lausen, B. Ludäscher, and W. May. "On Active Deductive Databases: The Statelog Approach". In: *Transactions and Change in Logic Databases*. Ed. by B. Freitag, H. Decker, M. Kifer, and A. Voronkov. Vol. 1472. Lecture Notes in Computer Science. Springer, 1998, pp. 69–106. ISBN: 3-540-65305-8.

[LS12] M. Lenzerini and D. F. Savo. "Updating inconsistent Description Logic knowledge bases". In: *ECAI*. Ed. by L. D. Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, and P. J. F. Lucas. Vol. 242. Frontiers in Artificial Intelligence and Applications. IOS Press, 2012, pp. 516–521. ISBN: 978-1-61499-097-0.

[LRLLS97] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. "GOLOG: A Logic Programming Language for Dynamic Domains". In: *J. Log. Program.* 31.1-3 (1997), pp. 59–83.

[LLMW11] H. Liu, C. Lutz, M. Milicic, and F. Wolter. "Foundations of instance level updates in expressive Description Logics". In: *Artif. Intell.* 175.18 (2011), pp. 2170–2197.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987. ISBN: 3-540-18199-7.

[MM04] F. Manola and E. Miller. *RDF Resource Description Framework*. W3C Recommendation http://www.w3.org/RDF/. 2004.

[MD89] D. R. McCarthy and U. Dayal. "The Architecture Of An Active Data Base Management System". In: *SIGMOD Conference*. Ed. by J. Clifford, B. G. Lindsay, and D. Maier. ACM Press, 1989, pp. 215–224.

[McC63] J. McCarthy. *Situations, Actions, and Causal Laws*. Tech. rep. Reprinted in MIT Press, Cambridge, Mass., 1968 pages 410-417. Stanford University, 1963.

[MH69] J. Mccarthy and P. J. Hayes. "Some Philosophical Problems from the Standpoint of Artificial Intelligence". In: *Machine Intelligence*. Edinburgh University Press, 1969, pp. 463–502.

[MH04] D. L. Mcguinness and F. van Harmelen. *OWL Web Ontology Language Overview*. W3C Recommendation. W3C Recommendation http://www.w3.org/TR/owl-features/. 2004.

[MVH+04] D. L. McGuinness, F. Van Harmelen, et al. "OWL web ontology language overview". In: *W3C recommendation* 10.2004-03 (2004), p. 10.

[MM09] Y. Mei and S. Madden. "ZStream: a cost-based query processor for adaptively detecting composite events". In: *SIGMOD Conference*. Ed. by U. Çetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul. ACM, 2009, pp. 193–206. ISBN: 978-1-60558-551-2.

[Mil83] R. Milner. "Calculi for Synchrony and Asynchrony". In: *Theor. Comput. Sci.* 25 (1983), pp. 267–310.

[MHRS06] B. Motik, I. Horrocks, R. Rosati, and U. Sattler. "Can OWL and Logic Programming Live Together Happily Ever After?" In: *International Semantic Web Conference*. Ed. by I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo. Vol. 4273. Lecture Notes in Computer Science. Springer, 2006, pp. 501–514. ISBN: 3-540-49029-9.

[MPSPBFHHHRS+09] B. Motik, P. F. Patel-Schneider, B. Parsia, C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler, et al. "OWL 2 web ontology language: Structural specification and functional-style syntax". In: *W3C recommendation* 27 (2009), p. 17.

[MR07] B. Motik and R. Rosati. "A Faithful Integration of Description Logics with Logic Programming". In: *IJCAI*. 2007, pp. 477–482.

[MSS05] B. Motik, U. Sattler, and R. Studer. "Query Answering for OWL-DL with rules". In: *J. Web Sem.* 3.1 (2005), pp. 41–60.

[MHT04] S. Mu, Z. Hu, and M. Takeichi. "An Injective Language for Reversible Computation". In: *Mathematics of Program Construction, 7th International Conference, MPC 2004, Stirling, Scotland, UK, July 12-14, 2004, Proceedings*. Ed. by D. Kozen and C. Shankland. Vol. 3125. Lecture Notes in Computer Science. Springer, 2004, pp. 289–313. ISBN: 3-540-22380-0. DOI: 10.1007/978-3-540-27764-4_16. URL: http://dx.doi.org/10.1007/978-3-540-27764-4_16.

[NB00] M. Nakamura and C. Baral. "Invariance, Maintenance, and Other Declarative Objectives of Triggers - A Formal Characterization of Active Databases". In: *Computational Logic*. Ed. by J. W. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey. Vol. 1861. Lecture Notes in Computer Science. Springer, 2000, pp. 1210–1224. ISBN: 3-540-67797-6.

[PPW06] G. Papamarkos, A. Poulovassilis, and P. T. Wood. "Event-condition-action rules on RDF metadata in P2P environments". In: *Comp. Networks* 50.10 (2006), pp. 1513–1532.

[PU07] I. C. C. Phillips and I. Ulidowski. "Reversing algebraic process calculi". In: *J. Log. Algebr. Program.* 73.1-2 (2007), pp. 70–96. DOI: 10.1016/j.jlap.2006.11.002. URL: http://dx.doi.org/10.1016/j.jlap.2006.11.002.

[Pin92] J. Pin. "On Reversible Automata". In: *LATIN '92, 1st Latin American Symposium on Theoretical Informatics, São Paulo, Brazil, April 6-10, 1992, Proceedings*. Ed. by I. Simon. Vol. 583. Lecture Notes in Computer Science. Springer, 1992, pp. 401–416. ISBN: 3-540-55284-7. DOI: 10.1007/BFb0023844. URL: http://dx.doi.org/10.1007/BFb0023844.

[Prz88] T. C. Przymusinski. "On the Declarative Semantics of Deductive Databases and Logic Programs". In: *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, 1988, pp. 193–216. ISBN: 0-934613-40-0.

[Rei77] R. Reiter. "On Closed World Data Bases". In: *Logic and Data Bases*. 1977, pp. 55–76.

[Rei91] R. Reiter. "The frame problem in the Situation Calculus: a simple solution (sometimes) and a completeness result for goal regression". In: *AI and mathematical theory of computation*. Ed. by V. Lifschitz. San Diego, CA, USA: Academic Press, 1991, pp. 359–380.

ISBN: 0-12-450010-2. URL: http://dl.acm.org/citation.cfm?id=132218.132239.

[Rei92] R. Reiter. "Formalizing Database Evolution in the Situation Calculus". In: *FGCS*. 1992, pp. 600–609.

[RK12] M. Rezk and M. Kifer. "Transaction Logic with Partially Defined Actions". In: *J. Data Semantics* 1.2 (2012), pp. 99–131.

[SMHP10] C. de Sainte Marie, G. Hallmark, and A. Paschke. *RIF Production Rule Dialect*. W3C Recommendation http://www.w3.org/TR/rif-prd/. 2010.

[SC06] M. J. Sergot and R. Craven. "The Deontic Component of Action Language nC+". In: *DEON*. Ed. by L. Goble and J.-J. C. Meyer. Vol. 4048. Lecture Notes in Computer Science. Springer, 2006, pp. 222–237. ISBN: 3-540-35842-0.

[Sha99] M. Shanahan. "The Event Calculus Explained". In: *AI Today*. 1999, pp. 409–430.

[SL10] M. Slota and J. Leite. "Towards closed world reasoning in dynamic open worlds". In: *TPLP* 10.4-6 (2010), pp. 547–563.

[SLS11] M. Slota, J. Leite, and T. Swift. "Splitting and updating hybrid knowledge bases". In: *TPLP* 11.4-5 (2011), pp. 801–819.

[SPS11] T. C. Son, E. Pontelli, and C. Sakama. "Formalizing Commitments Using Action Languages". In: *DALT*. Ed. by C. Sakama, S. Sardiña, W. Vasconcelos, and M. Winikoff. Vol. 7169. Lecture Notes in Computer Science. Springer, 2011, pp. 67–83. ISBN: 978-3-642-29112-8.

[Thi97] M. Thielscher. "Ramification and Causality". In: *Artif. Intell.* 89.1-2 (1997), pp. 317–364.

[VF12] C. Vaz and C. Ferreira. "On the analysis of compensation correctness". In: *J. Log. Algebr. Program.* 81.5 (2012), pp. 585–605.

[WFF98] C.-A. Wichert, B. Freitag, and A. Fent. "Logical Transactions and Serializability". In: *Transactions and Change in Logic Databases*. Ed. by B. Freitag, H. Decker, M. Kifer, and A. Voronkov. Vol. 1472. Lecture Notes in Computer Science. Springer, 1998, pp. 134–165. ISBN: 3-540-65305-8.

[Woo09] M. J. Wooldridge. *An Introduction to MultiAgent Systems (2. ed.)* Wiley, 2009. ISBN: 978-0-470-51946-2.

[WDR06] E. Wu, Y. Diao, and S. Rizvi. "High-performance complex event processing over streams". In: *SIGMOD Conference*. Ed. by S. Chaudhuri, V. Hristidis, and N. Polyzotis. ACM, 2006, pp. 407–418. ISBN: 1-59593-256-9.

[Yok10] T. Yokoyama. "Reversible Computation and Reversible Programming Languages". In: *Electr. Notes Theor. Comput. Sci.* 253.6 (2010), pp. 71–81. DOI: 10.1016/j.entcs.2010.02.007. URL: http://dx.doi.org/10.1016/j.entcs.2010.02.007.

[Zan95] C. Zaniolo. "Active Database Rules with Transaction-Conscious Stable-Model Semantics". In: *DOOD*. Ed. by T. W. Ling, A. O. Mendelzon, and L. Vieille. Vol. 1013. Lecture Notes in Computer Science. Springer, 1995, pp. 55–72. ISBN: 3-540-60608-4.

[ZF01] D. Zhang and N. Y. Foo. "EPDL: A Logic for Causal Reasoning". In: *IJCAI*. Ed. by B. Nebel. Morgan Kaufmann, 2001, pp. 131–138. ISBN: 1-55860-777-3.

# Proofs: External Transaction Logic

This appendix contains the proofs of lemmas, propositions and theorems of chapter 5.

## A.1 $\mathcal{ETR}$ Properties

**Proposition 1**. *Let $M$ be an interpretation, $\pi$ a path, $\pi_{\texttt{end}}$ the 1-path containing the last state of $\pi$, $\phi$ and $\psi$ be $\mathcal{ETR}$ formulas, $\phi'$ a positive formula, $\phi_P$ an atom from $\mathcal{L}_P$ and $a$ an atom such that $a \in \mathcal{L}_i$ or $a \in \mathcal{L}_a^*$.*

1. *If $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$ then $\exists a$ s.t. $a$ occurs in $\phi$, $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$*
2. *If $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$ then $M, \pi \models_p \phi \otimes \psi$*
3. *If $M, \pi \models_c \phi'$ then $M, \pi \models_p \phi'$*
4. *$M, \pi \models_c \phi_P$ iff $M, \pi \models_p \phi_P$*

*Proof.* Next we prove the proposition 1 from page 53.

We prove each item separately.

1. If $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$ then $\exists a$ s.t. $a$ occurs in $\phi$, $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$

   We prove by induction on the structure of $\phi$:
   **Base Case:** If $\phi$ is an atom then, by the Base Case of the definitions of Classical and Partial satisfaction (Case 1. of definitions 13 and 14) we know that $\phi \in \mathcal{L}_i \cup \mathcal{L}_a^*$ and for a path $\pi$ s.t. $\pi$ is a 1-path: $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$. Since $\phi$ is an action, and $\pi$ is a 1-path, this statement holds for action $a = \phi$.

   **Induction Step:**

*Conjunction*: Assume that the result holds for $\phi$ and $\psi$. We need to prove that if $M, \pi \models_p \phi \wedge \psi$ and $M, \pi \not\models_c \phi \wedge \psi$ then $\exists a$ s.t. $a$ occurs in $\phi \wedge \psi$ $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$.

Simplifying, we have to prove that if $M, \pi \models_p \phi$ and $M, \pi \models_p \psi$, and $M, \pi \not\models_c \phi$ or $M, \pi \not\models_c \psi$, then $\exists a$ s.t. $a$ occurs in $\phi \wedge \psi$, $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$. We have two possible cases:
1) Suppose that $M, \pi \models_p \phi$ and $M, \pi \models_p \psi$ and $M, \pi \not\models_c \phi$ hold. Then we can apply apply the induction hypothesis to $\phi$ and thus conclude that $\exists a$ s.t. $a$ occurs in $\phi$ $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$. Since $a$ occurs in $\phi$, then it also occurs in $\phi \wedge \psi$, and thus the hypothesis holds in this case.

2) Suppose that $M, \pi \models_p \phi$ and $M, \pi \models_p \psi$ and $M, \pi \not\models_c \psi$ hold. Similarly to the previous case, we can apply the induction hypothesis to $\psi$ and conclude that $\exists a$ s.t. $a$ occurs in $\psi$, $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$. Since $a$ occurs in $\psi$, then it also occurs in $\phi \wedge \psi$, and thus the hypothesis holds in this case.

*Disjunction*: Assume that the result holds for $\phi$ and $\psi$. We need to prove that if $M, \pi \models_p \phi \vee \psi$ and $M, \pi \not\models_c \phi \vee \psi$ then $\exists a$ s.t. $a$ occurs in $\phi \vee \psi$, $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$.

Simplifying, we have to prove that if $M, \pi \models_p \phi$ or $M, \pi \models_p \psi$, and $M, \pi \not\models_c \phi$ and $M, \pi \not\models_c \psi$ then $\exists a$ s.t. $a$ occurs in $\phi \vee \psi$, $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$. Now we have two cases:
1) If $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$ hold, then we can apply the induction hypothesis to $\phi$ and thus conclude that $\exists a$ s.t. $a$ occurs in $\phi$, $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$. Since $a$ occurs in $\phi$, then it also occurs in $\phi \vee \psi$, and thus the hypothesis holds in this case.

2) If $M, \pi \models_p \psi$ and $M, \pi \not\models_c \psi$ hold, then we can apply the induction hypothesis to $\psi$ and thus conclude that $\exists a$ s.t. $a$ occurs in $\psi$. $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$. Since $a$ occurs in $\psi$, then it also occurs in $\phi \vee \psi$, and thus the hypothesis holds in this case.

*Serial Conjunction*: Assume the result holds for $\phi$ and $\psi$. We need to prove that if $M, \pi \models_p \phi \otimes \psi$ and $M, \pi \not\models_c \phi \otimes \psi$ then $\exists a$ s.t. $a$ occurs in $\phi \otimes \psi$, $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$.

By definition of serial conjunction in $\models_p$ and $\models_c$, this is equivalent to ( [ (a) $\exists \pi_1, \pi_2 : \pi_1 \circ \pi_2 = \pi$ and $M, \pi_1 \models_c \phi$ and $M, \pi_2 \models_p \psi$] or [ (b) $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$] )
And $\forall \pi_3, \pi_4 : \pi_3 \circ \pi_4 = \pi, M, \pi_3 \not\models_c \phi$ or $M, \pi_4 \not\models_c \psi$

Suppose (b) is the case. We can apply the Induction Hypothesis for $\phi$, and conclude that $\exists a$ s.t. $a$ occurs in $\phi$, $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$. Since $a$ occurs in $\phi$, then $a$ occurs in $\phi \otimes \psi$, and thus the hypothesis holds in this case.

Suppose (a) is the case and $\forall \pi_3, \pi_4 : \pi_3 \circ \pi_4 = \pi$ either $M, \pi_3 \not\models_c \phi$ or $M, \pi_4 \not\models_c \psi$ (because $M, \pi \not\models_c \phi \otimes \psi$). Since this latter statement holds for every split of $\pi$, then

in particular it also holds for $\pi = \pi_1 \circ \pi_2$. As such, we know that $M, \pi_1 \models_c \phi$ and $M, \pi_2 \models_p \psi$ and ($M, \pi_1 \not\models_c \phi$ or $M, \pi_2 \not\models_c \psi$).

Since $M, \pi_1 \models_c \phi$ and $M, \pi_1 \not\models_c \phi$ cannot be true together, we can conclude that $M, \pi_1 \models_c \phi$, $M, \pi_2 \models_p \psi$ and $M, \pi_2 \not\models_c \psi$. From this, we can apply the induction hypothesis to $\psi$ for the path $\pi_2$ and conclude that $\exists a$ s.t. $a$ occurs in $\psi$. $M, \pi'_{\texttt{end}} \models_p a$ and $M, \pi'_{\texttt{end}} \not\models_c a$ for path $\pi_2$. Since $\pi$ is composed of $\pi_1 \circ \pi_2$, $\pi_2$ is in the *ending* of path $\pi$ then this result holds for path $\pi$ and $\pi'_{\texttt{end}} = \pi_{\texttt{end}}$. Since $a$ occurs in $\psi$, then it also occurs in $\phi \otimes \psi$ and thus the hypothesis holds for this case.

*Negation*: Let's assume that the result holds for $\phi$. We need to prove that if $M, \pi \models_p \neg\phi$ and $M, \pi \not\models_c \neg\phi$ then $\exists a$ s.t. $a$ occurs in $\neg\phi$, $M, \pi_{\texttt{end}} \models_p a$ and $M, \pi_{\texttt{end}} \not\models_c a$.

By definition of negation in the partial and classical satisfactions, we know that this is equivalent to $M, \pi \not\models_p \phi$ and $M, \pi \models_c \phi$. However, this latter is always false as $M, \pi \models_c \phi$ implies $M, \pi \models_p \phi$ by definition of $\models_p$. Since the antecedent of the implication is always false, the statement is trivially satisfied. □

2. If $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$ then $M, \pi \models_p \phi \otimes \psi$
   This result holds immediately a consequence of item 5.a) of the definition of partial satisfaction (Definition 14).

3. Let $\phi$ be a positive formula: If $M, \pi \models_c \phi$ then $M, \pi \models_p \phi$
   We prove this result by induction on the structure of $\phi$:
   **Base Case:** If $\phi$ is an atom, then this statement holds immediately by item 1.a) of the definition of partial satisfaction (Definition 14).
   **Induction Step:**
   *Conjunction*: Let's assume that the result holds for $\phi$ and $\psi$. We want to prove that if $M, \pi \models_c \phi \wedge \psi$ then $M, \pi \models_p \phi \wedge \psi$. As such, assume that $M, \pi \models_c \phi \wedge \psi$, and thus, by definition of classical satisfaction, we have $M, \pi \models_c \phi$ and $M, \pi \models_c \psi$. Moreover, applying the induction hypothesis, we know that $M, \pi \models_p \phi$ and $M, \pi \models_p \psi$ and thus by the classical conjunction case of definition 14 we have that $M, \pi \models_p \phi \wedge \psi$

   *Disjunction*: Let's assume that the result holds for $\phi$ and $\psi$. We want to prove that if $M, \pi \models_c \phi \vee \psi$ then $M, \pi \models_p \phi \vee \psi$. Assuming $M, \pi \models_c \phi \vee \psi$ then we know that either $M, \pi \models_c \phi$ or $M, \pi \models_c \psi$ holds. Consequently, we have two cases:
   1) $M, \pi \models_c \phi$ and since the result holds for $\phi$ then we know that $M, \pi \models_p \phi$ and thus by definition of the disjunction case $M, \pi \models_p \phi \vee \psi$ holds for any transaction $\psi$. 2) $M, \pi \models_c \psi$ and since the result holds for $\psi$ then we know that $M, \pi \models_p \psi$ and thus by definition of the disjunction case $M, \pi \models_p \phi \vee \psi$ holds for any transaction $\phi$.

   *Serial Conjunction*: Assume that the result holds for $\phi$ and $\psi$. We want to prove that if $M, \pi \models_c \phi \otimes \psi$ then $M, \pi \models_p \phi \otimes \psi$. Since we know that $M, \pi \models_c \phi \otimes \psi$ then there must exist a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1 \models_c \phi$ and $M, \pi_2 \models_c \psi$. If this is the case, then by hypothesis we know that $M, \pi_1 \models_p \phi$ and $M, \pi_2 \models_p \psi$. By applying item b) of the serial conjunction case of partial satisfaction, it follows that $M, \pi \models_p \phi_1 \otimes \phi_2$.

4. $M, \pi \models_c \phi_P$ iff $M, \pi \models_p \phi_P$
   This statement follow easily from the definition of partial satisfaction. Since $\phi_P$ is an atom from $\mathcal{L}_P$, then item 1b. of Definition 14 is never applicable. As such, if $M, \pi \models_p \phi_P$ then $M, \pi \models_c \phi_P$. The reverse holds by item 1. of Definition 14

□

**Theorem 2.** *Let $M$ be an interpretation, $\phi$ any formula, and $\phi'$ a positive formula and $\pi, \pi'$ paths such that $\pi'$ is a path where no external actions appear in the transitions. Then:*

$$\text{If } M, \pi \models_c \phi' \text{ then } M, \pi \models \phi' \tag{A.1}$$

$$M, \pi' \models_c \phi \text{ iff } M, \pi' \models \phi \tag{A.2}$$

*Proof.* Next we show the proof of theorem 2 from page 56.

We prove each item separately,

- If $M, \pi \models_c \phi$ then $M, \pi \models \phi$
  This result is proven by induction on the structure of $\phi$:
  **Base Case:** If $\phi$ is an atom, then this statement holds trivially by item 1 of definitions 18 and 13.
  **Induction Step:**
  *Disjunction*: Assume that the result holds for $\phi$ and $\psi$. We want to prove that if $M, \pi \models_c \phi \vee \psi$ then $M, \pi \models \phi \vee \psi$. Since $M, \pi \models_c \phi \vee \psi$ then one of the following holds: (1) $M, \pi \models_c \phi$ or (2) $M, \pi \models_c \phi \vee \psi$. If (1) is the case then, by hypothesis, we know that $M, \pi \models \phi$, and thus by definition $M, \pi \models \phi \vee \psi$. If (2) is the case instead, then, by hypothesis, it follows that $M, \pi \models \psi$, and so by definition $M, \pi \models \phi \vee \psi$.

  *Conjunction*: Let's assume that this result holds for $\phi$ and $\psi$. We want to prove that if $M, \pi \models_c \phi \wedge \psi$ then $M, \pi \models \phi \wedge \psi$. Since $M, \pi \models_c \phi \wedge \psi$ then both $M, \pi \models_c \phi$ and $M, \pi \models_c \psi$. So, by induction hypothesis, $M, \pi \models \phi$ and $M, \pi \models \psi$ holds, and thus, by definition, $M, \pi \models \phi \wedge \psi$.

  *Serial Conjunction*: Assume now that the result holds for $\phi$ and $\psi$. We want to prove that if $M, \pi \models_c \phi \otimes \psi$ then $M, \pi \models \phi \otimes \psi$. Since $M, \pi \models_c \phi \otimes \psi$ holds, then there must exist a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1 \models_c \phi$ and $M, \pi_2 \models_c \psi$. Then, by induction hypothesis, we can conclude $M, \pi_1 \models \phi$ and $M, \pi_2 \models \psi$. Since $\pi_1 \circ \pi_2$ are splits of $\pi$, then by the serial conjunction case of the definition of general satisfaction (definition 18) if follows that $M, \pi \models \phi \otimes \psi$.

- For the second item of this Theorem, we have to prove that $M, \pi \models_c \phi$ iff $M, \pi \models \phi$, where $\pi$ is a path without external actions in the annotated transitions.
  Since $\pi$ does not contain annotations for external actions, we know that any subpath of $\pi_1$ of $\pi$, $\pi_1$ does not have external actions. Moreover, we know by Definiton 17 that if a path $\pi_1$ does not contain external actions in the annotations, then

$M, \pi_1 \rightsquigarrow \phi$ is impossible for any formula $\phi$. Consequently, since $M, \pi_1 \rightsquigarrow \phi$ is impossible for any subpath $\pi_1$ of $\pi$, then the Compensating Case of definition 18 is never applicable for path $\pi$. Thus, since the definitions of general and partial satisfaction (Definitions 18 and 13) are exactly the same except for this case, then for such a path $\pi$, it follows that $M, \pi \models \phi$ iff $M, \pi' \models \phi$.

□

Before proving Theorem 3 on page 216 we need two auxiliary lemmas 6 and 7, as well as definitions 88 and 89, which are as follows.

**Definition 88.** *Let $P$ be a program without external actions, well-formed in both $\mathcal{TR}$ and $\mathcal{ETR}$. Let $M$ be a $\mathcal{TR}$ model of $P$. We define $M^{ETR}$ to be the interpretation obtained from $M$ as follows*

1. *If $\phi \in \mathcal{L}_P$ then, whenever $\phi \in M(\langle (D_0, \ldots, D_n)\rangle)$ then $\phi \in M^{ETR}(\langle (D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E)\rangle)$ for every external state $E$ and every $A_i$ $(1 \leq i \leq n)$ s.t. $\mathcal{O}^t(D_{i-1}, D_i) \models A_i$;*

2. *If $\phi \in \mathcal{L}_{\mathcal{O}}$ then, for every $\pi$ satisfying the restrictions in Definition 11, $\phi$ belongs to $M^{ETR}(\pi)$;*

3. *nothing else belongs to $M^{ETR}(\pi)$.*

**Lemma 6.** *Let $P$ be a program without external actions, well-formed in both $\mathcal{TR}$ and $\mathcal{ETR}$ and $\phi$ be a formula without external actions, well-formed in both $\mathcal{TR}$ and $\mathcal{ETR}$. Let $M$ be a $\mathcal{TR}$ model of $P$ and $M^{ETR}$ the $\mathcal{ETR}$ interpretation obtained from $M$ as described in Definition 88. Then:*

- *$M^{ETR}$ is a $\mathcal{ETR}$ model of $P$,* and
- *$M^{ETR}, \langle (D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E)\rangle \models_{ETR} \phi$ iff $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi$*

*Proof.* We start by proving the equivalence in the second item, and do this by proving each direction separately. First we show ($\Rightarrow$ direction) that $M^{ETR}, \langle (D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E)\rangle \models_{ETR} \phi$ implies $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi$.
$\Rightarrow$:
We make this proof by induction on the structure of $\phi$:

- *Base Case:* $\phi$ is an atom, and thus by definition there are two possible scenarios:

  - $\phi \in \mathcal{L}_i$.
    Since $\phi$ is a primitive of the oracle we know that $M^{ETR}, \pi \models_{ETR} \phi$ if one of the two cases holds:

    1. $\pi = \langle (D, E)\rangle$ and $\mathcal{O}^d(D) \models \phi$. In this case, by definition of $\mathcal{TR}$ interpretation, we know that $\phi \in M(\langle D\rangle)$ and thus that $M, \langle D\rangle \models_{TR} \phi$

2. $\pi = \langle (D_1, E) \xrightarrow{\phi} (D_2, E) \rangle$ and $\mathcal{O}^t(D_1, D_2)$
   $\models \phi$. By definition of $\mathcal{TR}$ interpretation, we know that $\phi \in M(\langle D_1, D_2 \rangle)$ and thus that $M, \langle D_1, D_2 \rangle \models_{TR} \phi$

– $\phi \in \mathcal{L}_P$.

$M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_{ETR} \phi$ and since this path does not have external actions, we can apply the results of Theorem 2 and $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_{ETR} \phi$ iff $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_c \phi$. Since $\phi \in \mathcal{L}_P$, this is equivalent to say that $\phi \in M^{ETR}(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle)$. Then, by definition, $\phi \in M^{ETR}(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle)$ because $\phi \in M(\langle D_0, \ldots, D_n \rangle)$. Thus, $M, \langle D_0, \ldots, D_n \rangle \models_{TR} \phi$.

- *Conjunction:* $\phi = \phi_1 \wedge \phi_2$

  $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_{ETR} \phi_1 \wedge \phi_2$, and since this path does not have external actions in its transitions, from Theorem 2: $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_c \phi_1 \wedge \phi_2$. Moreover, this is equivalent to $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_{ETR} \phi_1$ and $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_{ETR} \phi_2$. From this, we can apply our Induction Hypothesis individually to $\phi_1$ and $\phi_2$, concluding $M, \langle D_0, \ldots, D_n \rangle \models_{TR} \phi_1$ and $M, \langle D_0, \ldots, D_n \rangle \models_{TR} \phi_2$. Finally, by definition of $\models_{TR}$ we know that in this case $M, \langle D_0, \ldots, D_n \rangle \models_{TR} \phi_1 \wedge \phi_2$

- *Disjunction:* $\phi = \phi_1 \vee \phi_2$

  $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_{ETR} \phi_1 \vee \phi_2$, and since this path does not have external actions in its transitions, from Theorem 2: $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_c \phi_1 \vee \phi_2$. Moreover, this is equivalent to $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_{ETR} \phi_1$ or $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_{ETR} \phi_2$. We can then apply our Induction Hypothesis to $\phi_1$ and $\phi_2$ individually, concluding that $M, \langle D_0, \ldots, D_n \rangle \models_{TR} \phi_1$ or $M, \langle D_0, \ldots, D_n \rangle \models_{TR} \phi_2$. By definition of $\models_{TR}$ we can conclude that $M, \langle D_0, \ldots, D_n \rangle \models_{TR} \phi_1 \vee \phi_2$.

- *Serial Conjunction:* $\phi = \phi_1 \otimes \phi_2$

  $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_{ETR} \phi_1 \otimes \phi_2$, and since this path does not have external actions in its transitions, from Theorem 2: $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_c \phi_1 \otimes \phi_2$. Moreover, this is equivalent to saying that there is an $i$, where $0 \leq i \leq n$, such that $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_i, E) \rangle \models_{ETR} \phi_1$ and $M^{ETR}, \langle (D_i, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E) \rangle \models_{ETR} \phi_2$. We can then apply our Induction Hypothesis to $\phi_1$ and $\phi_2$ individually, concluding that $M, \langle D_0, \ldots, D_i \rangle \models_{TR} \phi_1$ and $M, \langle D_i, \ldots, D_n \rangle \models_{TR} \phi_2$. By definition of $\models_{TR}$ we know that this is equivalent to $M, \langle D_0, \ldots, D_n \rangle \models_{TR} \phi_1 \otimes \phi_2$.

- *Negation:* $\phi = \neg\phi_1$

$M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \neg\phi$, since this path does not have external actions in its transitions, from Theorem 2 we know that $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_c \neg\phi_1$. Moreover, since $\phi_1$ is an atom, this is equivalent to say $\phi_1 \notin M^{ETR}(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle)$ Then, by definition 88 we know that $\phi_1 \notin M(\langle D_0, \ldots, D_n\rangle)$ (as otherwise it would lead to a contradiction). Since $\phi_1$ is an atom, $M, \pi \models_{TR} \phi_1$ iff $\phi_1 \in M(\pi)$. Thus, it is not the case that $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1$ which implies $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \neg\phi_1$.

$\Leftarrow$:
We show $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi$ implies $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi$. This is also proven by induction on the structure of $\phi$:

- *Base Case:* $\phi$ is an atom.

  By satisfaction in $\mathcal{TR}$, $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi$ iff $\phi \in M(\langle D_0, \ldots, D_n\rangle)$. If the latter is true, by Definition 88, it follows that $\phi \in M^{ETR}(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle)$ which, by definition of $\models_{ETR}$, leads to $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi$

- *Conjunction:* $\phi = \phi_1 \wedge \phi_2$

  From satisfaction definition in $\mathcal{TR}$ we know that $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1 \wedge \phi_2$ implies that $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1$ and $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_2$. Applying the Induction Hypothesis individually to $\phi_1$ and $\phi_2$, it follows that $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_1$ and $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_2$. From this we can conclude as intended: $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_1 \wedge \phi_2$

- *Disjunction:* $\phi = \phi_1 \vee \phi_2$ This is proven exactly as for conjunction, replacing $\wedge$ by $\vee$.

- *Serial Conjunction:* $\phi = \phi_1 \otimes \phi_2$

  Given $\mathcal{TR}$'s satisfaction relation as presented in Definition 3: $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1 \otimes \phi_2$ implies that there is an $i$ (where $0 \leq i \leq n$) such that $M, \langle D_0, \ldots, D_i\rangle \models_{TR} \phi_1$ and $M, \langle D_i, \ldots, D_n\rangle \models_{TR} \phi_2$. By applying the Induction Hypothesis individually to $\phi_1$ and $\phi_2$ we obtain that $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_i, E)\rangle \models_{ETR} \phi_1$ and $M^{ETR}, \langle (D_i, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_2$. From this, it follows that, as desired, $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_1 \otimes \phi_2$

- *Negation:* $\phi = \neg\phi_1$

  $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \neg\phi_1$. From definition of $\models_{TR}$ and since $\phi_1$ must be an atom, this implies that $\phi_1 \notin M(\langle D_0, \ldots, D_n\rangle)$. We need to show $\phi_1 \notin M^{ETR}(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle)$, and we do this by contradiction.

  Assume $\phi_1 \in M^{ETR}(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle)$. By definition of $M^{ETR}$ this can only be the case if there is a rule in $P$, s.t. $\phi_1 \leftarrow \mathit{body}$ and $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \mathit{body}$.

From the $\Rightarrow$'s proof, we know $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} body$ implies $M, \langle D_0, \ldots, D_n\rangle \models_{TR} body$. Since $M$ is a model of $P$, then it also models this rule and $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1$ holds. Since $\phi_1$ is an atomic formula, this implies that $\phi_1 \in M(\langle D_0, \ldots, D_n\rangle)$ which is impossible. Consequently, $\phi_1 \notin M^{ETR}(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle)$ and $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \neg\phi_1$

Since $P$ does not have external actions, $M, \pi \rightsquigarrow body$ does not hold for any of the rules in $P$. As such the first result of this lemma comes directly from Definition 88 and the fact that $M$ is a model of $P$. □

**Definition 89.** *Let $M$ be an $\mathcal{ETR}$ interpretation. We define $M^{TR}$ to be the interpretation obtained from $M$ as follows*

1. *If $\phi \in M(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle)$ then $\phi \in M^{TR}(\langle D_0, \ldots D_n\rangle)$ for every $E$ (note that it is always the same $E$ in every state in the path), and every $A_i$ s.t. $A_i \in \mathcal{L}_i$*
2. *nothing else belongs to $M^{TR}$*

**Lemma 7.** *Let $P$ be a program without external actions well-formed in both $\mathcal{TR}$ and $\mathcal{ETR}$ and $\phi$ be a formula without external actions well-formed in both $\mathcal{TR}$ and $\mathcal{ETR}$. Let $M$ be an $\mathcal{ETR}$ model of $P$ and $M^{TR}$ the $\mathcal{TR}$ interpretation obtained by Definition 89. Then:*

$$M^{TR} \text{ is a } \mathcal{TR} \text{ model of } P, \text{ and } M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi \text{ iff } M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi$$

*Proof.* We start to prove the second claim of this lemma: $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi$ iff $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi$ As previously, we split this proof in two, proving the claim individually for each direction. For the $\Rightarrow$ direction we prove that $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi$ implies $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi$. Then, for the $\Leftarrow$ we show that $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi$ implies $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi$.

$\Rightarrow$:

We prove this by induction on the structure of $\phi$.

- *Base Case:* $\phi$ is an atom

  $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi$ implies $\phi \in M(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle)$. Then by definition 89 we know that it must be the case that $\phi \in M(\langle D_0, \ldots, D_n\rangle)$ which implies by $\models_{TR} M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{TR} \phi$

- *Conjunction:* $\phi = \phi_1 \wedge \phi_2$

  $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_1 \wedge \phi_2$, and since this path does not have external actions in its transitions, we know from Theorem 2 that $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_c \phi_1 \wedge \phi_2$. Moreover, this is equivalent to say that $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_1$ and $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_2$. From this, we can apply our Induction Hypothesis individually to the formulas $\phi_1$ and $\phi_2$,

concluding $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1$ and $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_2$. Finally, by definition of $\models_{TR}$ we know that this is equivalent to $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1 \wedge \phi_2$

- *Disjunction:* $\phi = \phi_1 \vee \phi_2$

  $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_1 \vee \phi_2$, and since this path does not have external actions in its transitions, we know from Theorem 2 that $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_c \phi_1 \vee \phi_2$. Moreover, this is equivalent to say $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_1$ or $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_2$. We can then apply our Induction Hypothesis to $\phi_1$ and $\phi_2$, concluding $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1$ or $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_2$. By definition of $\models_{TR}$ we know that this is equivalent to $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1 \vee \phi_2$.

- *Serial Conjunction:* $\phi = \phi_1 \otimes \phi_2$

  $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_1 \otimes \phi_2$, and since this path does not have external actions in its transitions, we know from Theorem 2 that $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_c \phi_1 \otimes \phi_2$. Moreover, this is equivalent to say that there is a $i$ where $0 \leq i \leq n$, such that $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_i, E)\rangle \models_{ETR} \phi_1$ and $M^{TR}, \langle (D_i, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_2$. We can then apply our Induction Hypothesis to $\phi_1$ and $\phi_2$ where: $M^{TR}, \langle D_0, \ldots, D_i\rangle \models_{TR} \phi_1$ and $M^{TR}, \langle D_i, \ldots, D_n\rangle \models_{TR} \phi_2$. By definition of $\models_{TR}$ we know that this is equivalent to $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1 \otimes \phi_2$.

- *Negation:* $\phi = \neg\phi_1$

  $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \neg\phi$, since this path does not have external actions in its transitions, from Theorem 2 we know that $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_c \neg\phi_1$. Moreover, since $\phi_1$ is an atom, this is equivalent to say $\phi_1 \notin M(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle)$ Then, by definition 89 we know that $\phi_1 \notin M^{TR}(\langle D_0, \ldots, D_n\rangle)$ (as otherwise it would lead to a contradiction). Since $\phi_1$ is an atom, $M^{TR}, \pi \models_{TR} \phi_1$ iff $\phi_1 \in M^{TR}(\pi)$. Thus, it is not the case that $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1$ which implies $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \neg\phi_1$.

$\Leftarrow$:

We prove this by induction on the structure of $\phi$.

- *Base Case:* $\phi$ is an atom.

  $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi$ implies that $\phi \in M(\langle D_0, \ldots, D_n\rangle)$. If the latter is true, by definition 89 $\phi \in M(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle)$ which by definition of $\models_{ETR}$ implies that $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi$

- *Conjunction:* $\phi = \phi_1 \wedge \phi_2$

  $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1 \wedge \phi_2$.

From $\models_{TR}$'s definition, this implies $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1$ and $M, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_2$. We are now in conditions to apply our Induction Hypothesis individually to $\phi_1$ and $\phi_2$ and conclude that $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_1$ and $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_2$. From this, as intended, we conclude: $M^{ETR}, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_1 \wedge \phi_2$

- *Disjunction:* $\phi = \phi_1 \vee \phi_2$

  Similar to the proof of conjunction, but where $\wedge$ is replaced by $\vee$, and "and" by "or".

- *Serial Conjunction:* $\phi = \phi_1 \otimes \phi_2$

  From $\mathcal{TR}$'s definition of satisfaction, $\models_{TR}$, if $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \phi_1 \otimes \phi_2$ then there must exist an $i$ such that $M^{TR}, \langle D_0, \ldots, D_i\rangle \models_{TR} \phi_1$ and $M^{TR}, \langle D_i, \ldots, D_n\rangle \models_{TR} \phi_2$ (where $0 \leq i \leq n$). Applying the Induction Hypothesis individually to $\phi_1$ and $\phi_2$ one concludes that $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_i, E)\rangle \models_{ETR} \phi_1$ and $M, \langle (D_i, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_2$. Thus, given the definition of $\models_{ETR}$: $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi_1 \otimes \phi_2$

- *Negation:* $\phi = \neg\phi_1$

  $M^{TR}, \langle D_0, \ldots, D_n\rangle \models_{TR} \neg\phi_1$. From definition of $\models_{TR}$ and since $\phi_1$ must be an atom, this implies that $\phi_1 \notin M(\langle D_0, \ldots, D_n\rangle)$. By definition 89 this also implies that $\phi_1 \notin M(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle)$ (otherwise it would lead to a contradiction). And thus $M, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \neg\phi_1$

We now prove the first claim: If $M$ is a model of $P$, then $M^{TR}$ is also model of $P$ in $\mathcal{TR}$. To prove this claim we need to show for every rule $head \leftarrow body$ in $P$ that, whenever $M^{TR}, \pi \models_{TR} body$ then $M^{TR}, \pi \models_{TR} head$, for every path $\pi = \langle D_1, \ldots, D_n\rangle$

By the second claim that was previously proven we know that $M^{TR}, \pi \models_{TR} body$ iff $M, \pi' \models_{ETR} body$, where $\pi' = \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle$. Moreover, since $M$ is a model of $P$ we know that $M, \pi' \models_{ETR} head$ and, since $head$ is an atomic formula, we also know that $head \in M(\langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle)$. By definition 89 this implies $head \in M^{TR}(\pi)$ and $M, \pi \models \phi$. □

**Theorem 3**. *Let $P$ be a transaction program and $\phi$ a transaction formula such that $P$ and $\phi$ are both well-formed in $\mathcal{TR}$'s and in $\mathcal{ETR}$'s syntax. Then for any external state $E$:*

$$P, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi \textit{ iff } P, \langle D_1, \ldots, D_n\rangle \models_{TR} \phi$$

*Proof.* In the following we show theorem 3 from page 58

$$P, \langle (D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)\rangle \models_{ETR} \phi \text{ iff}$$
$$P, \langle D_1, \ldots, D_n\rangle \models_{TR} \phi$$

We start by showing $P, ((D_0, E) \xrightarrow{A_1} \ldots \xrightarrow{A_n} (D_n, E)) \models_{ETR} \phi$ implies $P, (D_1, \ldots, D_n) \models_{TR} \phi$ ($\Rightarrow$ direction), and then show the converse, i.e., that $P, (D_1, \ldots, D_n) \models_{TR} \phi$ implies

$P, ((D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E)) \models_{ETR} \phi$ (the $\Leftarrow$ direction). For this proof, we take into account the result of Theorem 2, which says that for paths $\pi$ without external actions, $M, \pi \models_c \phi$ iff $M, \pi \models \phi$ in $\mathcal{ETR}$.

$\Rightarrow$: This proof follows by contradiction.

First recall that if $P, ((D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E)) \models_{ETR} \phi$ holds, then, by definition of the executional entailment, for every $\mathcal{ETR}$ model $M$ of program $P$, it is the case that $M, \langle (D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E) \rangle \models_{ETR} \phi$.

Let's assume that $P, (D_1, \ldots, D_n) \not\models_{TR} \phi$. By definition this implies that there is a $\mathcal{TR}$ model of $P$, $M_1$, s.t. $M_1, \langle D_1, \ldots, D_n \rangle \not\models \phi$. Additionally, we can construct an $\mathcal{ETR}$ interpretation $M_1^{ETR}$, as defined in the auxiliary Definition 88 below, that, as proven by auxiliary Lemma 6, is an $\mathcal{ETR}$ model of $P$. This Lemma 6 also tells us that $M_1, \langle D_1, \ldots, D_n \rangle \models_{TR} \phi$ iff $M_1^{ETR}, \langle (D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E) \rangle \models_{ETR} \phi$, and thus $M_1, \langle D_1, \ldots, D_n \rangle \not\models \phi$ implies that $M_1^{ETR}, \langle (D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E) \rangle \not\models_{ETR} \phi$. Since $M_1^{ETR}$ is also an $\mathcal{ETR}$ model, this leads to a contradiction, and thus $P, (D_1, \ldots, D_n) \models_{TR} \phi$ must hold.

$\Leftarrow$: Once again, we make the proof by contradiction.

Start by recalling that if $P, (D_1, \ldots, D_n) \models_{TR} \phi$, by definition of the executional entailment in $\mathcal{TR}$, it follows that for every $\mathcal{TR}$ model $M$ of $P$: $M, \langle D_1, \ldots, D_n \rangle \models_{TR} \phi$.

By contradiction, assume that $P, ((D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E)) \not\models_{ETR} \phi$ . By definition this implies that there must exist a $M_1$, s.t. $M_1$ is an $\mathcal{ETR}$ model of $P$ and $M_1, \langle (D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E) \rangle \not\models_{ETR} \phi$. If this is the case, then we can construct a $\mathcal{TR}$ interpretation $M_1^{TR}$, as defined below (auxiliary Definition 89) that, c.f. auxiliary Lemma 7, is a $\mathcal{TR}$ model of $P$. Moreover, this Lemma 7 also tells us that $M_1^{TR}, \langle D_1, \ldots, D_n \rangle \models_{TR} \phi$ iff $M_1, \langle (D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E) \rangle \models_{ETR} \phi$, and thus $M_1, \langle (D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E) \rangle \not\models_{ETR} \phi$, implies that $M_1^{TR}, \langle D_1, \ldots, D_n \rangle \not\models_{TR} \phi$. Since $M_1^{TR}$ is also a $\mathcal{TR}$ model, this leads to a contradiction, and $P, ((D_0, E) \overset{A_1}{\rightarrow} \ldots \overset{A_n}{\rightarrow} (D_n, E)) \models_{ETR} \phi$ must hold. □

## A.2 Soundness and Completeness of $\mathcal{ETR}$ Procedure

Next we show the auxiliary results to show Theorem 4 of page 62, which establishes the Soundness and Completeness of $\mathcal{ETR}'$ derivation procedure $\vdash$.

### A.2.1 Soundness of $\vdash$

The auxiliary results up to Lemma 14 contribute to the soundness proof. We start by proving the soundness of $\vdash_c$ w.r.t. to classical satisfaction; then show that replacing in resolvents, an atom which is the head of some rule by the rule's body is a sound operation; then that action failed derivations corresponds to finding formulas that are partial but not classically satisfied, and that rule-5 of the procedure builds compensations. Finally we prove that each of the rules of the procedure is sound w.r.t. the general executional entailment.

**Lemma 8** (Soundness $\vdash_c$)**.** *Let $P$ be a serial-Horn program, $M$ be a model of $P$, $\pi$ be a path, $\phi$ a serial goal.*

$$\textit{If } P, \pi \vdash_c \phi \textit{ then } M, \pi \models_c \phi$$

*Proof.*

**Soundness of Axiom**:

$P, \pi \vdash_c ()$ only holds for paths $\pi$ with size 1. Since $()$ represents the empty transaction, which is tautologically true in any path of length 1, the result follows trivially, and $M, \pi \models_c ()$.

**Soundness of Rules**:

We prove each of the rules, $r_1$–$r_4$, separately:

r.1 Assume there is a rule $L_1 \leftarrow B_1 \otimes \ldots \otimes B_j$ in $P$. We want to prove that if $M, \pi \models_c B_1 \otimes \ldots \otimes B_j \otimes L_2 \otimes \ldots \otimes L_k$ then $M, \pi \models_c L_1 \otimes L_2 \otimes \ldots \otimes L_k$
Since we have $M, \pi \models_c B_1 \otimes \ldots \otimes B_j \otimes L_2 \otimes \ldots \otimes L_k$ then we know that there is a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1 \models_c B_1 \otimes \ldots \otimes B_j$ and $M, \pi_2 \models_c L_2 \otimes \ldots \otimes L_k$. Then, since $M$ is a model of $P$ and $L_1 \leftarrow B_1 \otimes \ldots \otimes B_j$ is a serial-Horn rule, it follows that $M, \pi_1 \models_c L_1$ and thus $M, \pi \models_c L_1 \otimes L_2 \otimes \ldots \otimes L_k$

r.2 Assume that $\mathcal{O}^d(D_1) \models L_1$. We want to prove that if $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_f} (D_f, E_f) \rangle \models_c L_2 \otimes \ldots \otimes L_k$ then $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_f} (D_f, E_f) \rangle \models_c L_1 \otimes L_2 \otimes \ldots \otimes L_k$ holds.
Since $\mathcal{O}^d(D_1) \models L_1$, we know that for every interpretation, $M, \langle (D_1, E_1) \rangle \models_c L_1$. Thus, by definition of the serial conjunction case of $\models_c$:
$M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_f} (D_f, E_f) \rangle \models_c L_1 \otimes L_2 \otimes \ldots \otimes L_k$

r.3 Assume that $\mathcal{O}^t(D_0, D_1) \models L_1$. We want to prove that if $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_f} (D_f, E_f) \rangle \models_c L_2 \otimes \ldots \otimes L_k$ then $M, \langle (D_0, E_1) \xrightarrow{L_1} (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_f} (D_f, E_f) \rangle \models_c L_1 \otimes L_2 \otimes \ldots \otimes L_k$
Since $\mathcal{O}^t(D_0, D_1) \models L_1$, we know that:
$M, \langle (D_0, E_1) \xrightarrow{L_1} (D_1, E_1) \rangle \models_c L_1$. Thus, by definition of the serial conjunction case of $\models_c$:
$M, \langle (D_0, E_1) \xrightarrow{L_1} (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_f} (D_f, E_f) \rangle \models_c L_1 \otimes L_2 \otimes \ldots \otimes L_k$

r.4 Assume $\mathcal{O}^e(E_0, E_1) \models L_1$. We want to prove that if $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_f} (D_f, E_f) \rangle \models_c L_2 \otimes \ldots \otimes L_k$ then $M, \langle (D_1, E_0) \xrightarrow{L_1} (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_f} (D_f, E_f) \rangle \models_c L_1 \otimes L_2 \otimes \ldots \otimes L_k$
Since $\mathcal{O}^e(E_0, E_1) \models L_1$, we know:
$M, \langle (D_1, E_0) \xrightarrow{L_1} (D_1, E_1) \rangle \models_c L_1$. Thus, by definition of the serial conjunction case of $\models_c$:
$M, \langle (D_1, E_0) \xrightarrow{L_1} (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_f} (D_f, E_f) \rangle \models_c L_1 \otimes L_2 \otimes \ldots \otimes L_k$

□

**Definition 90** (Unfolding of formulas)**.** *Let $P$ be a serial-Horn program, and $\phi$ a serial-goal of the form: $\phi = \phi_1 \otimes \ldots \otimes \phi_i \otimes \ldots \otimes \phi_k$. A one-step unfolding of $\phi$ is a serial goal obtained from*

*$\phi$ by replacing one atom $\phi_i$ ($1 \leq i \leq k$) in $\phi$ by a serial goal body, in case $\phi_i \leftarrow$ body is a rule in $P$. An unfolding of $\phi$ is a serial goal obtained from $\phi$ by iteratively applying one-step unfolding a finite number of times.*

*A serial-Horn goal is completely unfolded if it is empty, or if every atom in it is an action formula defined in the oracles.*

**Lemma 9.** *Let $\phi$ be a completely unfolded serial-Horn goal formula, and $M$ an interpretation s.t. $M \models P$.*

*If $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$, then for all $\psi$ s.t. $\psi$ is a serial-Horn goal, it is the case that $M, \pi \models_p \phi \otimes \psi$ and $M, \pi \not\models_c \phi \otimes \psi$*

*Proof.* We prove this by induction on the size of the serial-Horn goal $\phi = \phi_1 \otimes \ldots \otimes \phi_k$:

**Inductive Hypothesis:** Let $\pi$ be a path, and $M$ a model of a program $P$. For completely unfolded serial goals $\psi$ and $\phi_1 \otimes \ldots \otimes \phi_k$:

If $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_k$ and $M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_k$ then $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_k \otimes \psi$ and $M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_k \otimes \psi$

**Base, k = 1:**

Let $\phi$ be a serial-Horn formula of size 1. Since $\phi$ is an atom, $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$ only if $\pi$ is a 1-path. By definition of $\models_p$, it follows immediately that $M, \pi \models_p \phi \otimes \psi$. Moreover, since $\pi$ is a 1-path, and $M, \pi \not\models_c \phi$ then by definition of the serial conjunction $\otimes$ in $\models_c$ the only split that can be done from $\pi$ is $\pi = \pi \circ \pi$ and thus it follows that $M, \pi \not\models_c \phi \otimes \psi$.

**Induction Step:**

Assume our hypothesis is true for values up to $k$. We prove that it is still true for $\phi_1 \otimes \ldots \otimes \phi_{k+1}$.

Assume that $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_{k+1}$ and $M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_{k+1}$. By definition of serial conjunction in $\models_p$, this is equivalent to: $[(M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_k \wedge M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_k) \vee (\exists \pi_1 \circ \pi_2 = \pi \; s.t. \; M, \pi_1 \models_c \phi_1 \otimes \ldots \otimes \phi_k \wedge M, \pi_2 \models_p \phi_{k+1})] \wedge (\forall \pi_3 \circ \pi_4 = \pi$ s.t. $M, \pi_3 \not\models_c \phi_1 \otimes \ldots \otimes \phi_k \vee M, \pi_4 \not\models_c \phi_{k+1})$

We consider each of these two cases individually:

(1) Assume $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_k \wedge M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_k$ is true. By induction hypothesis we know that $\forall \psi_1$ s.t. $\psi_1$ is a serial-Horn goal, then $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_k \otimes \psi_1$ and $M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_k \otimes \psi_1$. Moreover, since this holds for any formula $\psi_1$ s.t. $\psi_1$ is a serial-Horn goal, then it is also true for $\psi_1 = \phi_{k+1} \otimes \psi$ and thus $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_{k+1} \otimes \psi$ and $M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_{k+1} \otimes \psi$.

(2) Assume that $\exists \pi_1 \circ \pi_2 = \pi \; s.t. \; M, \pi_1 \models_c \phi_1 \otimes \ldots \otimes \phi_k \wedge M, \pi_2 \models_p \phi_{k+1}$ and that $\forall \pi_3 \circ \pi_4 = \pi. \; M, \pi_3 \not\models_c \phi_1 \otimes \ldots \otimes \phi_k \vee M, \pi_4 \not\models_c \phi_{k+1}$.

Since $M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_{k+1}$ and $M, \pi_1 \models_c \phi_1 \otimes \ldots \otimes \phi_k$, it follows that $M, \pi_2 \models_p \phi_{k+1}$ and $M, \pi_2 \not\models_c \phi_{k+1}$. Since $\phi_{k+1}$ is a completely unfolded serial-Horn formula of size 1, we are in the base case that was already proven, and we can conclude that $M, \pi_2 \not\models_c \phi_{k+1} \otimes \psi$ and $M, \pi_2 \models_p \phi_{k+1} \otimes \psi$. Then by definition of $\models_p$ we know that $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_k \otimes \psi$ Moreover, recall from the base case proof, $\pi_2$ must be a 1-path, and by definition of split,

$\pi_1 = \pi$. Since $M, \pi \models_c \phi_1 \otimes \ldots \otimes \phi_k$ and $M, \pi_2 \not\models_c \phi_{k+1} \otimes \psi$ then $M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_k \otimes \phi_{k+1} \otimes \psi$ □

**Lemma 10.** *Let $M$ be an interpretation, $\phi$ any transaction formula, $\psi$ an atomic action defined in the oracles and $\pi$ a path s.t. $\pi = \pi_1 \circ \pi_2$. If $M, \pi_1 \models_c \psi$ and $M, \pi_2 \not\models_c \phi$ then $M, \pi \not\models_c \psi \otimes \phi$*

*Proof.* We make this proof by contradiction. Assume $M, \pi \models_c \psi \otimes \phi$ holds. Then there is a split $\pi_3, \pi_4$ s.t. $M, \pi_3 \models_c \psi$ and $M, \pi_4 \models_c \phi$. Clearly, this property only holds if $\pi_4 \neq \pi_2$ which implies that $\pi_1 \neq \pi_3$

Since $\psi$ is an atom, for any path $\pi'$ $M, \pi' \models_c \psi$ iff $\psi \in M(\pi')$. Since $\psi$ is an *action defined in the oracles*, by definition of interpretation (Definition 11) we know that this $\pi'$ must be a 2-path. Since $\pi'$ must be a 2-path to satisfy $\psi$, and $\pi_1$ and $\pi_3$ must always be a prefix of $\pi$ (by the definition of path split), we can conclude that $\pi_1 = \pi_3$ and thus that $\pi_2 = \pi_4$. Consequently, $M, \pi_4 \not\models_c \phi$ ,and the assumption is contradicted. □

**Lemma 11** (Soundness of action-failed derivation w.r.t. $\models_c$ and $\models_p$)**.** *If there is an action-failed derivation starting in $\langle S_1 \rangle, S_1 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$ and ending in $\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} S_f \rangle, S_f \Vdash_P \psi$, for some serial-goal $\psi$, then for all models $M$ of $P$, $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_k$ and $M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_k$, for $\pi = \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} S_f \rangle$*

*Proof.* We start by proving by induction on the size $k$ of the serial-Horn formula $\phi_1 \otimes \ldots \otimes \phi_k$ that:
**Inductive Hypothesis:** If there is an action-failed derivation starting in $\langle S_1 \rangle, S_1 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_i$ and ending in $\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} S_f \rangle, S_f \Vdash_P \psi$, for some serial-goal $\psi$, then $\forall M$ s.t. $M \models P$, $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_k$ and $M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_k$ for $\pi = \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} S_f \rangle$
**Base Case** $k = 1$
If $k = 1$, then, by definition of action-failed derivation, $\pi$ must be a 1-path, $\langle S_f \rangle = \langle (D_f, E_f) \rangle$, the derivation starts and ends in the following resolvent:
$\langle (D_f, E_f) \rangle, (D_f, E_f) \Vdash_P \phi_1$, and one of the two cases must occur:

(i). $\phi_1 \in \mathcal{L}_i$, $\mathcal{O}^d(D_f) \not\models \phi_1$ and $\neg\exists D_i$ s.t. $\mathcal{O}^t(D_f, D_i) \models \phi_1$, or

(ii). $\phi_1 \in \mathcal{L}_a^*$ and $\neg\exists E_i$ s.t. $\mathcal{O}^e(E_f, E_i) \models \phi_1$

Since $\phi_1$ belongs to the language of the oracles, then it means that, for any $M$, we know that $\phi_1 \notin M(\langle (D_f, E_f) \rangle)$; but also, that $\neg\exists D_i, E_f$ such that $\phi_1 \in M\langle (D_f, E_f) \xrightarrow{\phi_1} (D_i, E_f) \rangle$ or that $\phi_1 \in M\langle (D_f, E_f) \xrightarrow{\phi_1} (D_f, E_i) \rangle$. As a result, by the definitions of $\models_p$ and $\models_c$, we have $M, \langle (D_f, E_f) \rangle \not\models_c \phi_1$, and $M, \langle (D_f, E_f) \rangle \models_p \phi_1$.
**Induction Step:**
Assume there is an action-failed derivation starting in $\langle S_1 \rangle, S_1 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_{k+1}$ and ending in $\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} S_f \rangle, S_f \Vdash_P \psi$, for some serial-goal $\psi$. Since $\phi_1 \otimes \ldots \otimes \phi_{k+1}$ is a completely unfolded formula, $\psi$ must be a subformula of $\phi_1 \otimes \ldots \otimes \phi_{k+1}$.

More precisely, there must exist a resolvent:
$\langle S_1, \ldots, S_f \rangle, S_f \Vdash_P \phi_i \otimes \ldots \otimes \phi_{k+1}$, $(1 \leq i \leq k+1)$ where one of the following conditions are true:

(i). $\phi_i \in \mathcal{L}_i$, $\mathcal{O}^d(D_f) \not\models L_1$ and $\neg\exists D_i$ s.t.
$\mathcal{O}^t(D_f, D_i) \models \phi_i$, or

(ii). $\phi_i \in \mathcal{L}_a^*$ and $\neg\exists E_i$ s.t. $\mathcal{O}^e(E_f, E_i) \models \phi_i$

In this case there are two possibilities:

1. $(1 \leq i \leq k)$ and by definition there is a classical derivation starting in $\langle S_1 \rangle, S_1 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$ and ending in $\pi, S_f \Vdash_P ()$. Since $i \leq k$, by induction hypothesis we know that $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_i$ and $M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_i$ and by Lemma 9 we know that $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_i \otimes \ldots \otimes \phi_{k+1}$ and $M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_i \otimes \ldots \otimes \phi_{k+1}$ for all $\phi_{i+1} \otimes \ldots \phi_{k+1}$

2. $(i = k+1)$ and by definition there is a classical derivation starting in $\langle S_1 \rangle, S_1 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$ and ending in $\pi, S_f \Vdash_P ()$. If so, we know, by Lemma 8, that $M, \pi \models_c \phi_1 \otimes \ldots \otimes \phi_k$. Moreover, we also have an action-failed derivation starting in $\langle S_f \rangle, S_f \Vdash_P \phi_{k+1}$ and ending in $\langle S_f \rangle, S_f \Vdash_P \phi_{k+1}$. Since $\phi_{k+1}$ is of size 1, as proven in the base case, we know $M, \langle S_f \rangle \models_p \phi_{k+1}$ and $M, \langle S_f \rangle \not\models_c \phi_{k+1}$. Since $\pi = \langle S_1 \, {}^{A_1}\!\!\rightarrow \ldots \, {}^{A_{f-1}}\!\!\rightarrow S_f \rangle$ we can conclude that $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_k \otimes \phi_{k+1}$ and $M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_k \otimes \phi_{k+1}$.

**Soundness of completely unfolded rules**
To prove the soundness of rules we need to prove the following:

r.1 Assume there is a rule $\phi_1 \leftarrow \psi \in P$. If $M, \pi \models_p \psi \otimes \phi_2 \otimes \ldots \otimes \phi_k$ and $M, \pi \not\models_c \psi \otimes \phi_2 \otimes \ldots \otimes \phi_k$ then $M, \pi \models_p \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$ and $M, \pi \not\models_c \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$

r.2 Assume $\mathcal{O}^d(D_1) \models \phi_1$
If $M, \pi \not\models_c \phi_2 \otimes \ldots \otimes \phi_k$ and $M, \pi \models_p \phi_2 \otimes \ldots \otimes \phi_k$ then it holds:
$M, \pi \not\models_c \phi_1 \otimes \ldots \otimes \phi_k$ and $M, \pi \models_p \phi_1 \otimes \ldots \otimes \phi_k$

r.3 Assume $\mathcal{O}^d(D_1, D_2) \models L_1$
If $M, \langle (D_2, E_2) \, {}^{A_2}\!\!\rightarrow \ldots \, {}^{A_{f-1}}\!\!\rightarrow (D_f, E_f) \rangle \not\models_c \phi_2 \otimes \ldots \otimes \phi_k$ and $M, \langle (D_2, E_2) \, {}^{A_2}\!\!\rightarrow \ldots \, {}^{A_{f-1}}\!\!\rightarrow (D_f, E_f) \rangle \models_p \phi_2 \otimes \ldots \otimes \phi_k$ then it holds:
$M, \langle (D_1, E_2) \, {}^{\phi_1}\!\!\rightarrow (D_2, E_2) \, {}^{A_2}\!\!\rightarrow \ldots \, {}^{A_{f-1}}\!\!\rightarrow (D_f, E_f) \rangle \not\models_c \phi_1 \otimes \ldots \otimes \phi_k$ and $M, \langle (D_1, E_2) \, {}^{\phi_1}\!\!\rightarrow (D_2, E_2) \, {}^{A_2}\!\!\rightarrow \ldots \, {}^{A_{f-1}}\!\!\rightarrow (D_f, E_f) \rangle \models_p \phi_1 \otimes \ldots \otimes \phi_k$

r.4 Assume $\mathcal{O}^e(E_0, E_1) \models \phi_1$
If $M, \langle (D_1, E_1) \, {}^{A_1}\!\!\rightarrow \ldots \, {}^{A_{f-1}}\!\!\rightarrow (D_f, E_f) \rangle \not\models_c \phi_2 \otimes \ldots \otimes \phi_k$ and $M, \langle (D_1, E_1) \, {}^{A_1}\!\!\rightarrow \ldots \, {}^{A_{f-1}}\!\!\rightarrow (D_f, E_f) \rangle \models_p \phi_2 \otimes \ldots \otimes \phi_k$ then it holds:
$M, \langle (D_1, E_0) \, {}^{\phi_1}\!\!\rightarrow (D_1, E_1) \, {}^{A_1}\!\!\rightarrow \ldots \, {}^{A_{f-1}}\!\!\rightarrow (D_f, E_f) \rangle \not\models_c \phi_1 \otimes \ldots \otimes \phi_k$ and $M, \langle (D_1, E_0) \, {}^{\phi_1}\!\!\rightarrow (D_1, E_1) \, {}^{A_1}\!\!\rightarrow \ldots \, {}^{A_{f-1}}\!\!\rightarrow (D_f, E_f) \rangle \models_p \phi_1 \otimes \ldots \otimes \phi_k$

We prove each rule individually:

r.1 Not applicable since $\phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$ is a completely unfolded goal

r.2 Assume $\mathcal{O}^d(D_1) \models \phi_1$, $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} (D_f, E_f) \rangle \not\models_c \phi_2 \otimes \ldots \otimes \phi_k$ and $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} (D_f, E_f) \rangle \models_p \phi_2 \otimes \ldots \otimes \phi_k$.

Since $\mathcal{O}^d(D_1) \models \phi_1$, $\phi_1 \in M(\langle (D_1, E_1) \rangle)$ holds, by the serial conjunction case of the partial satisfaction definition, we can conclude that $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} (D_f, E_f) \rangle \models_p \phi_1 \otimes \ldots \otimes \phi_k$. Moreover, since $\phi_1$ is an oracle primitive, we can apply Lemma 10 and conclude that: $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} (D_f, E_f) \rangle \not\models_c \phi_1 \otimes \ldots \otimes \phi_k$

r.3 Assume $\mathcal{O}^t(D_0, D_1) \models \phi_1$, $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} (D_f, E_f) \rangle \not\models_c \phi_2 \otimes \ldots \otimes \phi_k$ and $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} (D_f, E_f) \rangle \models_p \phi_2 \otimes \ldots \otimes \phi_k$

Since $\mathcal{O}^t(D_0, D_1) \models \phi_1$, we know that $\phi_1 \in M(\langle (D_0, E_1) \xrightarrow{L_1} (D_1, E_1) \rangle)$. Thus, by the serial conjunction case of the partial satisfaction definition we can conclude: $M, \langle (D_0, E_1) \xrightarrow{L_1} (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} (D_f, E_f) \rangle \models_p \phi_1 \otimes \ldots \otimes \phi_k$. Moreover, since $\phi_1$ is an oracle primitive, we can apply Lemma 10 and conclude: $M, \langle (D_0, E_1) \xrightarrow{L_1} (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} (D_f, E_f) \rangle \not\models_c \phi_1 \otimes \ldots \otimes \phi_k$

r.4 Assume $\mathcal{O}^e(E_0, E_1) \models \phi_1$,
$M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} (D_f, E_f) \rangle \not\models_c \phi_2 \otimes \ldots \otimes \phi_k$ and $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} (D_f, E_f) \rangle \models_p \phi_2 \otimes \ldots \otimes \phi_k$

Since $\mathcal{O}^e(E_0, E_1) \models \phi_1$ we know:
$\phi_1 \in M(\langle (D_1, E_0) \xrightarrow{\phi_1} (D_1, E_1) \rangle)$. Thus, by the serial conjunction case of the partial satisfaction definition we can conclude that $M, \langle (D_1, E_0) \xrightarrow{L_1} (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} (D_f, E_f) \rangle \models_p \phi_1 \otimes \ldots \otimes \phi_k$. Moreover, since $\phi_1$ is an oracle primitive, we can apply Lemma 10 and conclude: $M, \langle (D_1, E_0) \xrightarrow{L_1} (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{f-1}} (D_f, E_f) \rangle \not\models_c \phi_1 \otimes \ldots \otimes \phi_k$

□

**Lemma 12** (Soundness of rule-5. w.r.t. $\rightsquigarrow$)**.** *Let $P$ be a serial-Horn program, $\phi_1 \otimes \ldots \otimes \phi_k$ be a completely unfolded serial-Horn goal.*

*If all the following conditions are true:*

1. *There is an action-failed classical derivation starting in $\langle S_1 \rangle, S_1 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$ (where $S_1 = (D_1, E_1)$) ending in $\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{j-1}} S_j \rangle, S_j \Vdash_P \psi$, for some serial-goal $\psi$*
2. *$S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p$ is the rollback path of the path $S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{j-1}} S_j$ (cf. Definition 15)*
3. *$\texttt{Inv}(\texttt{Seq}(\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \rangle)) = A_k^{-1} \otimes \ldots \otimes A_1^{-1}$ (cf. Definition 16)*
4. *$P, \langle S_p \xrightarrow{A_k^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \rangle, S_q \vdash_c A_k^{-1} \otimes \ldots \otimes A_1^{-1}$*

*then $M, \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \xrightarrow{A_{p-1}^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \rangle \rightsquigarrow \phi_1 \otimes \ldots \otimes \phi_k$ for all models $M$ of $P$*

*Proof.* From Lemma 11 we know that if $\phi_1 \otimes \ldots \otimes \phi_k$ is a completely unfolded serial-Horn goal and there is an action-failed classical derivation starting in $\langle S_1 \rangle, S_1 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$ (where $S_1 = (D_1, E_1)$) ending in $\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{j-1}} S_j \rangle, S_j \Vdash_P \psi$, for some serial-goal $\psi$ then $M, \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{j-1}} S_j \rangle \models_p \phi_1 \otimes \ldots \otimes \phi_k$ and $M, \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{j-1}} S_j \rangle \not\models_c \phi_1 \otimes \ldots \otimes \phi_k$.

Note that $S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p$ is the rollback path of the path $S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{j-1}} S_j$ (applying Definition 15) and $\texttt{Inv}(\texttt{Seq}(\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \rangle)) = A_k^{-1} \otimes \ldots \otimes A_1^{-1}$ (applying Definition 16).

By Lemma 8 we know that:
$P, \langle S_p \xrightarrow{A_k^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \rangle, S_q \vdash_c A_k^{-1} \otimes \ldots \otimes A_1^{-1}$ then $M, \langle S_p \xrightarrow{A_k^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \rangle, S_q \models_c A_k^{-1} \otimes \ldots \otimes A_1^{-1}$.

Then, by Definition 17 we can conclude that if conditions 1-4 hold then: $M, \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \xrightarrow{A_{p-1}^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \rangle \rightsquigarrow \phi_1 \otimes \ldots \otimes \phi_k$ □

**Definition 91.** *$M_{def}$ is an interpretation defined w.r.t. a program $P$ as follows:*

- *$\phi \in M_{def}(\pi)$ and $\phi \in \mathcal{L}_{\mathcal{O}}$ iff $\mathcal{O}^d(\pi) \models \phi$ or $\mathcal{O}^t(\pi) \models \phi$ or $\mathcal{O}^e(\pi) \models \phi$.*
- *$\phi \in M_{def}(\pi)$ and $\phi \in \mathcal{L}_P$ iff there is a rule $\phi \leftarrow body \in P$ and $M, \pi \models body$*
- *$M_{def}(\pi) \rightsquigarrow \phi$ and $\phi$ is a complex formula iff conditions 1-4 of Definition 17 are true.*
- *$M_{def}(\pi) \rightsquigarrow \phi$ and $\phi$ is an atom iff there is a rule $\phi \leftarrow body \in P$ and $M, \pi \rightsquigarrow body$*

**Remark 1.** *It follows easily from the definition, that $M_{def}$ is a valid $\mathcal{ETR}$ interpretation and a model of program $P$.*

**Lemma 13.** *Let $P$ be a program containing the rule $\psi \leftarrow \phi_1$. For every model $M$ of $P$:*
*If $M, \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ then $M, \pi \models \psi \otimes \phi_2 \otimes \ldots \otimes \phi_j$.*

*Proof.* We will apply induction on the size $k$ of path $\pi = \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{n-1}}, S_k \rangle$
**Induction Hypothesis:** For every model $M$ of $P$ where $P$ contains the rule $\psi \leftarrow \phi_1$, if $M, \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ then $M, \pi \models \psi \otimes \phi_2 \otimes \ldots \otimes \phi_j$.
**Base**
If $k = 1$, since it is impossible to construct a compensating path that has size 1, $M, \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ is only true by the serial conjunction case. Moreover, since $\pi$ has size 1, $M, \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_1$ iff $M, \pi \models \phi_1$, $M, \pi \models \phi_2$, ..., $M, \pi \models \phi_j$. Since for every path $\pi_1$: if $M, \pi_1 \models \phi_1$ then, since $M$ is a model, we have $M, \pi_1 \models \psi$. Since $M, \pi \models \psi$, and $M, \pi \models \phi_2 \otimes \ldots \otimes \phi_j$, and so we also have $M, \pi \models \psi \otimes \phi_2 \otimes \ldots \otimes \phi_j$.
**Step**
Let's assume that the hypothesis is true for paths up to size $j$. Here we prove that it is also true for paths of size $j + 1$.

By definition of $\models$, we know that $M, \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ holds if either of the cases occurs:

(a) Serial Conjunction Case: $M, \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ is true because there is a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1 \models \phi_1$ and $M, \pi_2 \models \phi_2 \otimes \ldots \otimes \phi_j$. Since $M$ is a model of $P$, we know $M, \pi_1 \models \psi$ and then by the definition of Serial Conjunction, $M, \pi \models \psi \otimes \phi_2 \otimes \ldots \otimes \phi_j$.

(b) Compensating Case: $M, \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ is true because there is a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1 \rightsquigarrow \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ and $M, \pi_2 \models \phi \otimes \phi_2 \otimes \ldots \otimes \phi_j$.

In this case $\pi_2$ is strictly smaller than $\pi$ as otherwise $\pi_1$ would be a 1-path and $M, \pi_1 \rightsquigarrow \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ would not hold. Consequently, we can apply the Induction Hypothesis to $\pi_2$ and conclude $M, \pi_2 \models \psi \otimes \phi_2 \otimes \ldots \otimes \phi_j$.

If $M, \pi_1 \rightsquigarrow \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$, there is a path $\pi'$ s.t. $M, \pi' \models_p \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ and $M, \pi' \not\models_c \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$. By definition of the serial conjunction case in the partial satisfaction relation we know that:

(b1) $M, \pi_1 \rightsquigarrow \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ because rules 1-4 of Definition 17 hold, and the failure occurs in $\phi_1$, i.e., $M, \pi' \models_p \phi_1$ and $M, \pi' \not\models_c \phi_1$. Since $M$ is a model of program, we know $M, \pi_1 \rightsquigarrow \phi_1$ implies $M, \pi_1 \rightsquigarrow \psi$ and thus $M, \pi_1 \rightsquigarrow \psi \otimes \phi_2 \otimes \ldots \otimes \phi_j$. Since $M, \pi_2 \models \psi \otimes \phi_2 \otimes \ldots \otimes \phi_j$ we can conclude $M, \pi \models \psi \otimes \phi_2 \otimes \ldots \otimes \phi_j$.

(b2) $M, \pi_1 \rightsquigarrow \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ because rules 1-4 of Definition 17 hold, and the failure occurs after $\phi_1$. I.e., there is a path $\pi'$ with a split $\pi'_1 \circ \pi'_2$ of $\pi'$ s.t. $M, \pi'_1 \models_c \phi_1$, $M, \pi'_2 \models_p \phi_2 \otimes \ldots \otimes \phi_j$ and $M, \pi'_2 \not\models_c \phi_2 \otimes \ldots \otimes \phi_j$. Since $M$ is a model of $P$ and $M, \pi'_1 \models_c \phi_1$ it holds $M, \pi'_1 \models_c \psi$, and thus $M, \pi_1 \rightsquigarrow \psi \otimes \phi_2 \otimes \ldots \otimes \phi_j$. From this and $M, \pi_2 \models \psi \otimes \phi_2 \otimes \ldots \otimes \phi_j$ we know that $M, \pi \models \psi \otimes \phi_2 \otimes \ldots \otimes \phi_j$

(b3) $M, \pi_1 \rightsquigarrow \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ and rules 1-4 of Definition 17 do not hold. However, this must be true for every model $M$ of $P$, and thus it must hold also for model $M_{def}$ specified in Definition 91. By $M_{def}$ definition, $M_{def}, \pi_1 \rightsquigarrow \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ and rules 1-4 of Definition 17 do not hold, only if $\phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j$ is an atom, i.e. if $\phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_j = \phi_1$. If this is the case, then by definition of what is a model, $M, \pi_1 \rightsquigarrow \phi_1$ implies $M, \pi \rightsquigarrow \psi$ and thus $M, \pi \models \psi$ as intended.

□

**Lemma 14** (Soundness $\Vdash_P$)**.** *Let $P$ be a serial-Horn program, $\phi_1 \otimes \ldots \otimes \phi_k$ ($k \geq 1$) be a serial-Horn goal and $\pi_1$,$\pi_2$ be paths. Let $\pi_1 \circ \pi_2, S_j \Vdash_P \psi$ be the next derivation step of $\pi_1, S_i \Vdash_P \phi$.*

*If $P, \pi_1 \models \phi$ then $P, \pi_1 \circ \pi_2 \models \psi$*

*Proof.* We have to prove the following:

r.1 Assume $\pi, S_i \Vdash_P \psi \otimes \phi_2 \otimes \ldots \otimes \phi_k$, $\pi, S_i \Vdash_P \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$ and $\phi_1 \leftarrow \psi$ is a rule in $P$.

If $P, \pi \models \psi \otimes \phi_2 \otimes \ldots \otimes \phi_k$ then $P, \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$

r.2 Assume $\pi, (D_i, E_i) \Vdash_P \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$, $\pi, (D_i, E_i) \Vdash_P \phi_2 \otimes \ldots \otimes \phi_k$, $\mathcal{O}^d(D_1) \models \phi_1$, and $\pi = \langle (D_1, E_1) \overset{A_1}{\hookrightarrow} \ldots \overset{A_{i-1}}{\hookrightarrow} (D_i, E_i) \rangle$

If $P, \pi \models \phi_2 \otimes \ldots \otimes \phi_k$ then $P, \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$

r.3 Assume $\pi, (D_i, E_i) \Vdash_P \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$, $\pi, (D_i, E_i) \Vdash_P \phi_2 \otimes \ldots \otimes \phi_k$, $\mathcal{O}^t(D_0, D_1) \models \phi_1$, and $\pi = \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i) \rangle$

If $P, \pi \models \phi_2 \otimes \ldots \otimes \phi_k$ then $P, \langle (D_0, E_1) \xrightarrow{\phi_1} (D_1, E_1) \rangle \circ \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$

r.4 Assume $\pi, (D_i, E_i) \Vdash_P \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$, $\pi, (D_i, E_i) \Vdash_P \phi_2 \otimes \ldots \otimes \phi_k$, $\mathcal{O}^e(E_0, E_1) \models \phi_1$, and $\pi = \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i) \rangle$

If $P, \pi \models \phi_2 \otimes \ldots \otimes \phi_k$ then $P, \langle (D_1, E_0) \xrightarrow{\phi_1} (D_1, E_1) \rangle \circ \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$

r.5 Assume $\langle S_q \xrightarrow{A_q} \ldots \xrightarrow{A_{i-1}} S_i \rangle, (D_i, E_i) \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$ and $\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \xrightarrow{A_{p-1}^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \xrightarrow{A_q} \ldots \xrightarrow{A_{i-1}} S_i \rangle, (D_i, E_i) \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$ and all the following conditions hold:

(a) There is an action-failed derivation starting in $\langle S_1 \rangle, S_1 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$ and ending in $\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{j-1}} S_j \rangle, S_j \Vdash_P \psi$, for some serial-goal $\psi$

(b) $S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p$ is the rollback path of the path $S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{j-1}} S_j$ (cf. Definition 15)

(c) $\texttt{Inv}(\texttt{Seq}(\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \rangle)) = A_k^{-1} \otimes \ldots \otimes A_1^{-1}$ (cf. Definition 16)

(d) $P, \langle S_p \xrightarrow{A_k^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \rangle, S_q \vdash_c A_k^{-1} \otimes \ldots \otimes A_1^{-1}$

If $P, \langle S_q \xrightarrow{A_q} \ldots \xrightarrow{A_{i-1}} S_i \rangle \models \phi_1 \otimes \ldots \otimes \phi_k$ then $P, \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \xrightarrow{A_{p-1}^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \xrightarrow{A_q} \ldots \xrightarrow{A_{i-1}} S_i \rangle \models \phi_1 \otimes \ldots \otimes \phi_k$

We prove each item in turn.

r.1 We know $P, \pi \models \psi \otimes \phi_2 \otimes \ldots \otimes \phi_k$ and $\phi_1 \leftarrow \psi$ is a rule in $P$. By definition, this is equivalent to saying: for every model $M$ of $P$, $M, \pi \models \psi \otimes \phi_2 \otimes \ldots \otimes \phi_k$. Then, by Lemma 13, we know that $M, \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$, and thus $P, \pi \models \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$

r.2 Assume that $\mathcal{O}^d(D_1) \models \phi_1$ and $P, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i) \rangle \models \phi_2 \otimes \ldots \otimes \phi_k$ holds. By definition of $\mathcal{ETR}$ interpretations we know that for every $M$, and in particular, for every $M$ that models $P$: $M, \langle (D_1, E_1) \rangle \models \phi_1$.

Since $P, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i) \rangle \models \phi_1 \otimes \ldots \otimes \phi_k$ then we also know $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i) \rangle \models \phi_2 \otimes \ldots \otimes \phi_k$ for every $M$ that models $P$. By the serial conjunction case we can conclude that
$M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i) \rangle \models \phi_1 \otimes \ldots \otimes \phi_k$ for every $M$ that models $P$. Consequently, as expected, it holds that $P, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i) \rangle \models \phi_1 \otimes \ldots \otimes \phi_k$

r.3 Assume that $\mathcal{O}^t(D_0, D_1) \models \phi_1$ and $P, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i) \rangle \models \phi_2 \otimes \ldots \otimes \phi_k$ holds. By definition we know that for every $M$ (and in particular, for every $M$ that models $P$):
$M, \langle (D_0, E_1) \xrightarrow{\phi_1} (D_1, E_1) \rangle \models \phi_1$. Since $P, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i) \rangle \models \phi_1 \otimes \ldots \otimes \phi_k$ then $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i) \rangle \models \phi_2 \otimes \ldots \otimes \phi_k$ for every $M$ that models $P$. By the serial conjunction case we can conclude that $M, \langle (D_0, E_1) \xrightarrow{\phi_1}$

$(D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i)\rangle \models \phi_1 \otimes \ldots \otimes \phi_k$. Consequently, as intended, it holds that $P, \langle (D_0, E_1) \xrightarrow{\phi_1} (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i)\rangle \models \phi_1 \otimes \ldots \otimes \phi_k$

r.4 Assume that $\mathcal{O}^e(E_0, E_1) \models \phi_1$ and $P, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i)\rangle \models \phi_2 \otimes \ldots \otimes \phi_k$ holds. By definition we know that for every $M$ (and in particular, for every $M$ that models $P$):
$M, \langle (D_1, E_0) \xrightarrow{\phi_1} (D_1, E_1)\rangle \models \phi_1$. Since we know $P, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i)\rangle \models \phi_1 \otimes \ldots \otimes \phi_k$ then $M, \langle (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i)\rangle \models \phi_2 \otimes \ldots \otimes \phi_k$ for every $M$ that models $P$. By the serial conjunction case we can conclude that $M, \langle (D_1, E_0) \xrightarrow{\phi_1} (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i)\rangle \models \phi_1 \otimes \ldots \otimes \phi_k$. Consequently, as intended, it holds that $P, \langle (D_1, E_0) \xrightarrow{\phi_1} (D_1, E_1) \xrightarrow{A_1} \ldots \xrightarrow{A_{i-1}} (D_i, E_i)\rangle \models \phi_1 \otimes \ldots \otimes \phi_k$

r.5 Assume the following:

(a) There is an action-failed derivation starting in $\langle S_1\rangle, S_1 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$ ending in $\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{j-1}} S_j\rangle, S_j \Vdash_P \psi$, for some serial-goal $\psi$

(b) $S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p$ is the rollback path of $S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{j-1}} S_j$ (cf. Definition 15)

(c) $\texttt{Inv}(\texttt{Seq}(\langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p\rangle)) = A_k^{-1} \otimes \ldots \otimes A_1^{-1}$ (cf. Definition 16)

(d) $P, \langle S_p \xrightarrow{A_k^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q\rangle, S_q \vdash_c A_k^{-1} \otimes \ldots \otimes A_1^{-1}$

and $P, \langle S_q \xrightarrow{A_q} \ldots \xrightarrow{A_{i-1}} S_i\rangle \models \phi_1 \otimes \ldots \otimes \phi_k$.

We know that for every model $M$ of $P$ that $M, \langle S_q \xrightarrow{A_q} \ldots \xrightarrow{A_{i-1}} S_i\rangle \models \phi_1 \otimes \ldots \otimes \phi_k$. If $\phi_1 \otimes \ldots \otimes \phi_k$ is completely unfolded, then the pre-conditions for the application of Lemma 12 are all verified. Otherwise, there is a complete unfolding, $Goal$, of $\phi_1 \otimes \ldots \otimes \phi_k$ iff $Goal$ can be obtained by a finite number of applications of rule 1 (something that follows directly from the definition of unfolding). In this case also, all pre-conditions for the application of Lemma 12 are verified. Thus, in both cases, there is a serial goal $\phi'_1 \otimes \ldots \otimes \phi'_{k'}$ (either the original one, or the unfolded $Goal$) for which we can apply Lemma 12, and $M, \langle S_q \xrightarrow{A_q} \ldots \xrightarrow{A_{i-1}} S_i\rangle \models \phi'_1 \otimes \ldots \otimes \phi'_{k'}$.

Applying Lemma 12 it follows that for every model $M$ of $P$: $M, \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \xrightarrow{A_{p-1}^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q\rangle \rightsquigarrow \phi'_1 \otimes \ldots \otimes \phi'_{k'}$, and since $M, \langle S_q \xrightarrow{A_q} \ldots \xrightarrow{A_{i-1}} S_i\rangle \models \phi'_1 \otimes \ldots \otimes \phi'_{k'}$, we can also conclude that: $M, \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \xrightarrow{A_{p-1}^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \xrightarrow{A_q} \ldots \xrightarrow{A_{i-1}} S_i\rangle \models \phi'_1 \otimes \ldots \otimes \phi'_{k'}$. Since rule 1. was already proven sound, then we can conclude that $M, \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \xrightarrow{A_{p-1}^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \xrightarrow{A_q} \ldots \xrightarrow{A_{i-1}} S_i\rangle \models \phi_1 \otimes \ldots \otimes \phi_k$

Finally, by definition of the executional entailment it holds: $P, \langle S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{p-1}} S_p \xrightarrow{A_{p-1}^{-1}} \ldots \xrightarrow{A_1^{-1}} S_q \xrightarrow{A_q} \ldots \xrightarrow{A_{i-1}} S_i\rangle \models \phi_1 \otimes \ldots \otimes \phi_k$.

□

## A.3 Completeness of $\vdash$

We now show the auxiliary results needed for the proof of completeness of the $\vdash$ procedure.

We start by constructing a canonical model of the program $P$ to link the model theory and the proof theory. We continue by proving that there are classical derivations for all serial conjunctions satisfied by that model, then consider the case of compensations, and finally prove that every atom or serial conjunction satisfied (with the general satisfaction relation) by the canonical model is obtained by the procedure. We end this section by proving that the canonical model is indeed a model.

**Definition 92** (Canonical Model)**.** *The canonical model of a program $P$ is the interpretation defined as follows:*

- $M_P(\pi) = \{a \in (\mathcal{L}_P \cup \mathcal{L}_\mathcal{O}) \mid P, \pi \vdash_c a\}$
- $M_P, \langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle \rightsquigarrow a$ *and $a$ is an atom* ***only if****: $a \in \mathcal{L}_P$ and rule $\mathbf{r_5}$. of the Definition 22 is applicable to the resolvent $\langle S_0\rangle, S_0 \Vdash_P a$, resulting in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_p \Vdash_P a$*

**Remark 2.** *Let $\phi$ and $\psi$ be serial-Horn formulas:*

1. *If $P, \pi_1 \vdash_c \phi$, $P, \pi_2 \vdash_c \psi$, and $\pi_1 \circ \pi_2$ a split of $\pi$ then $P, \pi \vdash_c \phi \otimes \psi$*
2. *If $P, \pi_1 \vdash \phi$, $P, \pi_2 \vdash \psi$, and $\pi_1 \circ \pi_2$ a split of $\pi$ then $P, \pi \vdash \phi \otimes \psi$*
3. *If rule $\mathbf{r_5}$. of the Definition 22 is applicable to the resolvent $\langle S_0\rangle, S_0 \Vdash_P \phi$ resulting in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_p \Vdash_P \phi$ and $P, \langle S_p \xrightarrow{A_{p+1}} \ldots \xrightarrow{A_{n-1}} S_n\rangle \vdash \phi$ then $P, \langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p \xrightarrow{A_{p+1}} \ldots \xrightarrow{A_{n-1}} S_n\rangle \vdash \phi$*
4. *If $P, \pi \vdash \phi$ then either $P, \pi \vdash_c \phi$ or there is a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $\pi_1 = \langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle$, there is a derivation starting in the resolvent $\langle S_0\rangle, S_0 \Vdash_P \phi$ resulting in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_p \Vdash_P \phi$ where rule $\mathbf{r_5}$. of the Definition 22 was applicable and $P, \pi_2 \vdash \phi$.*

**Lemma 15.** *If $M_P, \pi \models_c \phi_1 \otimes \ldots \otimes \phi_k$ then $P, \pi \vdash_c \phi_1 \otimes \ldots \otimes \phi_k$*

*Proof.* We prove by induction on the size of the formula $\phi_1 \otimes \ldots \otimes \phi_k$.
*Base Case 1 (k = 0)*: $M_P, \pi \models_c ()$ iff $\pi$ is a 1-path $\langle S\rangle$. If this is the case, then the derivation starting in the resolvent $\langle S\rangle, S \Vdash_P ()$ succeeds, for every state $S$. Since we have not used rule $\mathbf{r_5}$. for this derivation, we can conclude $P, \pi \vdash_c ()$
*Base Case 2 (k = 1)*: $M_P, \pi \models_c \phi$ and $\phi$ is an atom, then by definition of $M_P$ we know that $P, \pi \vdash_c \phi$
*Induction Step (k = j+1)*: Suppose the hypothesis is true for $\phi_1 \otimes \ldots \otimes \phi_j$.

$M_P, \pi \models_c \phi_1 \otimes \ldots \otimes \phi_{j+1}$ implies that there is a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M_P, \pi_1 \models_c \phi_1 \otimes \ldots \otimes \phi_j$ and $M_P, \pi_2 \models_c \phi_{j+1}$. Applying the Induction Hypothesis to $\pi_1$ we know that $P, \pi_1 \vdash_c \phi_1 \otimes \ldots \otimes \phi_j$. Since $\phi_{j+1}$ is an atom, from Base Case 2 follows $P, \pi_2 \vdash_c \phi_{j+1}$.

Since $P, \pi_1 \vdash_c \phi_1 \otimes \ldots \otimes \phi_j$ and $P, \pi_2 \vdash_c \phi_{j+1}$ we conclude $P, \pi \vdash_c \phi_1 \otimes \ldots \otimes \phi_j \otimes \phi_{j+1}$ □

**Lemma 16.** *If* $M_P, \langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle \rightsquigarrow \phi_1 \otimes \ldots \otimes \phi_k$ *then rule* $\mathbf{r_5}$. *of the Definition 22 is applicable to the resolvent* $\langle S_0\rangle, S_0 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$ *resulting in the resolvent* $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_p \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$

*Proof.* We prove by induction on the size of the formula $\phi_1 \otimes \ldots \otimes \phi_k$. Note that $M, \pi \rightsquigarrow ()$ is impossible for every path $\pi$.
*Base Case (k = 1)*: $M_P, \langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle \rightsquigarrow \phi$ and $\phi \in \mathcal{L}_P$ (as $\phi \in \mathcal{L}_{\mathcal{O}}$ then $M, \pi \rightsquigarrow \phi$ is impossible for every path $\pi$).

Then by definition of $M_P$ we know that rule $\mathbf{r_5}$. of the Definition 22 is applicable to the resolvent $\langle S_0\rangle, S_0 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$ resulting in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_p \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$
*Induction Step (k = j+1)*: Suppose the hypothesis is true for $\phi_1 \otimes \ldots \otimes \phi_j$.

$M_P, \pi \rightsquigarrow \phi_1 \otimes \ldots \otimes \phi_{j+1}$. In this case we have two possible scenarios:

1. $M_P, \pi \rightsquigarrow \phi_1 \otimes \ldots \otimes \phi_j$ and thus by Induction Hypothesis we know that rule $\mathbf{r_5}$. of the Definition 22 is applicable to the resolvent $\langle S_0\rangle, S_0 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_j$ resulting in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_p \Vdash_P \phi_1 \otimes \ldots \otimes \phi_j$. If this is the case, by definition of rule $\mathbf{r_5}$. we know that it is also applicable to the resolvent $\langle S_0\rangle, S_0 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_j \otimes \phi_{j+1}$ resulting in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_0 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_j \otimes \phi_{j+1}$, for a path $\pi = \langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle$.

2. $M_P, \pi \not\rightsquigarrow \phi_1 \otimes \ldots \otimes \phi_j$ and $M_P, \pi \rightsquigarrow \phi_1 \otimes \ldots \otimes \phi_{j+1}$. In this case, we know that there exist a path $\pi_1, \pi_0, \pi_r$ s.t. $M_P, \langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_j} S_j\rangle \models_c \phi_1 \otimes \ldots \otimes \phi_j$ and $M_P, \langle S_j\rangle \models_p \phi_{j+1}$ $M_P, \langle S_j\rangle \not\models_c \phi_{j+1}$, $\pi_0$ a the rollback path of $\pi_1$ and, a path $M_P, \pi_r \models_c \texttt{Inv}(\texttt{Seq}(\pi_0))$ and $\pi = \pi_0 \circ \pi_r$. If this is the case, by Lemma 15 there must exist a classical derivation $P, \langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_j} S_j\rangle \vdash_c \phi_1 \otimes \ldots \otimes \phi_j$. By definition of a classical action-failed derivation we also know that there is a classical action-failed derivation starting in $\langle S_j\rangle, S_j \Vdash_P \phi_{j+1}$ and thus there is a classical action-failed derivation starting in $\langle S_0\rangle, S_0 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_j \otimes \phi_{j+1}$ and ending in $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_j} S_j\rangle, S_j \Vdash_P \phi_{j+1}$.

   Since, by Lemma 15 it follows that $P, \pi_r \vdash_c \texttt{Inv}(\texttt{Seq}(\pi_0))$, then all conditions of rule $\mathbf{r_5}$. of the Definition 22 hold, and so we can apply it to the resolvent $\langle S_0\rangle, S_0 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_j \otimes \phi_{j+1}$ resulting in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_p \Vdash_P \phi_1 \otimes \ldots \otimes \phi_j \otimes \phi_{j+1}$, where $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle = \pi_0 \circ \pi_r$.

□

**Lemma 17.** *Let* $\pi$ *be a 1-path of the form* $\pi = \langle S\rangle$. *If* $M_P, \pi \models \phi_1 \otimes \ldots \otimes \phi_j$ *then* $P, \pi \vdash \phi_1 \otimes \ldots \otimes \phi_j$

*Proof.* We apply induction on the size of the serial-Horn formula $\phi_1 \otimes \ldots \otimes \phi_j$:
*Base Case 1 (j = 0):*
$M_P, \pi \models ()$. If this is the case, then there is a derivation starting in the resolvent $\langle S\rangle, S \Vdash_P$

() and successfully ending in the resolvent $\langle S\rangle, S \Vdash_P ()$. And thus both $P, \pi \vdash_c ()$ and $P, \pi \vdash ()$ are true.
*Base Case 2 (j = 1):*
$M_P, \pi \models \phi$ and $\phi$ is an atom. If this is the case, and since $M_P, \pi \rightsquigarrow \phi$ is not true for any path smaller than size 2, then $M_P, \pi \models \phi$ iff $\phi \in M_P(\pi)$. Moreover, by definition of $M_P$ this implies that $P, \pi \vdash_c \phi$ which implies $P, \pi \vdash \phi$.
*Inductive Case (j = k+1):*
Suppose the hypothesis holds for values up to $k$.

Since $\pi$ is a 1-path and $M_P, \pi \rightsquigarrow \phi_1 \otimes \ldots \otimes \phi_k \otimes \phi_{k+1}$ is not defined for paths smaller than size 2, then $M_P, \pi \models \phi_1 \otimes \ldots \otimes \phi_k \otimes \phi_{k+1}$ iff there are $k+1$ splits of $\pi$ s.t. $M, \pi_i \models \phi_i$ $(1 \leq i \leq k+1)$.

However, since $\pi$ is a 1-path, the only possible split to make is $\pi_i = \pi$. Thus for $M_P, \pi \models \phi_1 \otimes \ldots \otimes \phi_k \otimes \phi_{k+1}$ to hold, $M, \pi \models \phi_i$ must hold for all formulas $\phi_i$ in $\phi_1 \otimes \ldots \otimes \phi_k \otimes \phi_{k+1}$. If this is the case, by Base Case 2, for every formula, $P, \pi \vdash \phi_i$, and thus $P, \pi \vdash \phi_1 \otimes \ldots \otimes \phi_k \otimes \phi_{k+1}$ □

**Lemma 18.** *Let $a$ be an atom. If $M_P, \pi \models a$ then $P, \pi \vdash a$*

*Proof.* We apply induction on the size of the k-path $\pi$.
*Base Case (k = 1):*
$M_P, \pi \models a$ iff $a \in M_P(\pi)$ and thus $P, \pi \vdash_c a$ and $P, \pi \vdash a$
*Induction step (k = m+1):*
$M_P, \pi \models a$ if:

- $a \in M_P(\pi)$ and thus $P, \pi \vdash_c a$ and also $P, \pi \vdash a$; or
- $M_P, \pi_1 \rightsquigarrow a$ and $M_P, \pi_2 \models a$. If this is the case, then $\pi_2$ must be strictly smaller than $\pi$, as $\pi_1$ must be at least a 2-path. So, by Induction Hypothesis, we can conclude that $P, \pi_2 \vdash a$.

  Moreover, by definition of $M_P$, we know that $M_P, \pi_1 \rightsquigarrow a$ implies that there is a derivation starting in the resolvent $\langle S_0\rangle, S_0 \Vdash_P a$ where rule $\mathbf{r_5}$. of the Definition 22 is applicable and resulting in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_p \Vdash_P a$. In this case $\pi_1 = \langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle$. From this, we conclude $P, \pi \vdash a$.

□

**Lemma 19.** *If $M_P, \pi \models \phi_1 \otimes \ldots \otimes \phi_j$ then $P, \pi \vdash \phi_1 \otimes \ldots \otimes \phi_j$*

*Proof.* We prove by lexicographic induction first on the size of the k-path $\pi$ and then on the size of the serial-Horn formula $\phi_1 \otimes \ldots \otimes \phi_j$
*Base Case (k = 1)*:
$\pi$ is a 1-path and $M_P, \pi \models \phi_1 \otimes \ldots \otimes \phi_j$. In this case we can apply Lemma 17 and conclude $P, \pi \vdash \phi_1 \otimes \ldots \otimes \phi_j$.

*Induction step (k = m+1)*:
Assume the hypothesis holds for paths with size up to $m$.

$M_P, \pi \models \phi_1 \otimes \ldots \otimes \phi_j$ holds if one the two cases is true:

- *Serial Conjunction Case:* There is a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1 \models \phi_1$ and $M, \pi_2 \models \phi_2 \otimes \ldots \otimes \phi_j$. In this case we know that either $\pi_1$ or $\pi_2$ are strictly smaller than $\pi$ and both $\phi_1$ and $\phi_2 \otimes \ldots \otimes \phi_j$ are strictly smaller than $\phi_1 \otimes \ldots \otimes \phi_j$. As such, and since we know by Lemma 18 that for an atomic formula $\phi$ $M_P, \pi \models \phi$ then $P, \pi \vdash \phi$. Then by Induction Hypothesis we have $P, \pi_1 \vdash \phi_1$ and $P, \pi_2 \vdash \phi_2 \otimes \ldots \otimes \phi_j$. From this we know $P, \pi \vdash \phi_1 \otimes \ldots \otimes \phi_j$

- *Compensating Case:* There is a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1 \rightsquigarrow \phi_1 \otimes \ldots \otimes \phi_j$ and $M, \pi_2 \models \phi_1 \otimes \ldots \otimes \phi_j$. Since $\pi_1$ must be at least a 2-path, then we know that both $\pi_1$ and $\pi_2$ are strictly smaller than $\pi$. As such we can apply the Induction Hypothesis to $\pi_2$ and conclude $P, \pi_2 \vdash \phi_1 \otimes \ldots \otimes \phi_j$.

  Moreover, since we know $M, \pi_1 \rightsquigarrow \phi_1 \otimes \ldots \otimes \phi_j$ then by Lemma 16 we know that rule $\mathbf{r_5}$. of the Definition 22 is applicable to the resolvent $\langle S_0 \rangle, S_0 \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$ resulting in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p \rangle, S_p \Vdash_P \phi_1 \otimes \ldots \otimes \phi_k$, for $\pi_1 = \langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p \rangle$. Consequently, since $\pi_1 \circ \pi_2$ are splits of $\pi$, it follows that $P, \pi \vdash \phi_1 \otimes \ldots \otimes \phi_j$

□

**Lemma 20.** *Let $a$ be an atom.*

*If $P, \pi \vdash a$ then $M_P, \pi \models a$*

*Proof.* We prove by induction on the size of $\pi$.
*Base Case (k = 1):*
If $P, \pi \vdash a$ then either $a$ is an oracle primitive and $\mathcal{O}^d(\pi) \models a$ (i.e. rule $\mathbf{r_2}$.) , or we can apply rule $\mathbf{r_1}$. together with rule $\mathbf{r_2}$. an arbitrary number of times to reach the resolvent $\pi, S \Vdash_P ()$. Note that rule $\mathbf{r_5}$. is never applicable in this case.

As such, if $\pi$ is a 1-path $P, \pi \vdash a$ implies $P, \pi \vdash_c a$ and thus $a \in M(\pi)$ and $M, \pi \models a$
*Inductive step (k = m+1):*
Assume this holds for paths of size up to $m$.

If $P, \pi \vdash a$ then one of the two cases is true:

1. $P, \pi \vdash_c a$ and in this case by definition of $M_P$ we know that $a \in M_P(\pi)$ and thus $M_P, \pi \models a$

2. There is a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $\pi_1 = \langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p \rangle$, where the resolvent $\langle S_0 \rangle, S_0 \Vdash_P \phi$ results in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p \rangle, S_p \Vdash_P \phi$ by applying rule $\mathbf{r_5}$. of the Definition 22 and $P, \pi_2 \vdash \phi$. Since $\pi_2$ is strictly smaller than $\pi$, then by Induction Hypothesis we know that $M, \pi_2 \vdash \phi$. Moreover by definition of $M_P$, since there is a resolvent $\langle S_0 \rangle, S_0 \Vdash_P \phi$ which results in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p}$

$S_p\rangle, S_p \Vdash_P \phi$ by applying rule $\mathbf{r_5}$. of the Definition 22, it follows that $M_P, \pi_1 \rightsquigarrow a$. Consequently, by the compensating case of $\models$, it follows that $M_P, \pi \models a$.

□

**Lemma 21.** *$M_P$ is a model of $P$.*

*Proof.* For $M_P$ to be a model of $P$ we have to prove for every rule $h \leftarrow body$:

(1) if $M_P, \pi \models body$ then $M_P, \pi \models head$

(2) if $M_P, \pi \models_c body$ then $M_P, \pi \models_c head$

(3) if $M_P, \pi \rightsquigarrow body$ then $M_P, \pi \rightsquigarrow head$

We prove each claim in turn:

(1) Assume $M_P, \pi \models body$, then by Lemma 19 we know that $P, \pi \vdash body$. By definition of $\vdash$, since the rule $h \leftarrow body$ exists in $P$, this is equivalent to $P, \pi \vdash head$. Then, by Lemma 20, $M_P, \pi \models head$.

(2) Assume $M_P, \pi \models_c body$, then by Lemma 15 we know that $P, \pi \vdash_c body$. By definition of $\vdash$, since the rule $h \leftarrow body$ exists in $P$, this is equivalent to $P, \pi \vdash_c head$. Then by definition of $M_P$ we know that $M_P, \pi \models_c head$.

(3) Assume now that $M_P, \pi \rightsquigarrow body$. Then by Lemma 16 we know that, for $\pi = \langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle$, rule $\mathbf{r_5}$. of the Definition 22 is applicable to the resolvent $\langle S_0\rangle, S_0 \Vdash_P body$ resulting in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_p \Vdash_P body$.

From this we can apply rule $\mathbf{r_1}$. and conclude $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_p \Vdash_P head$. If this is the case, then we could have started in the resolvent $\langle S_0\rangle, S_0 \Vdash_P head$ and apply $\mathbf{r_5}$. resulting in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_p \Vdash_P head$ (as rule $\mathbf{r_1}$. could be performed as part of the classical action-failed derivation in the first step of rule $\mathbf{r_5}$.).

As such, there is a derivation starting in $\langle S_0\rangle, S_0 \Vdash_P head$ resulting in the resolvent $\langle S_0 \xrightarrow{A_1} \ldots \xrightarrow{A_p} S_p\rangle, S_p \Vdash_P head$. Consequently, by definition of $M_P$, $M_P, \pi \rightsquigarrow head$.

□

Finally, we can show soundness and completeness of $\vdash$ as follows.

**Theorem 4**. *Let $P$ be a serial-Horn program, $\phi$ a serial-Horn goal, and $\pi$ be a path starting in state $S_0$ and ending in $S_f$. Then, $P, \pi \models \phi$ iff $P, \pi \vdash \phi$.*

*Proof.*

$$P, \pi \models \phi \text{ iff } P, \pi \vdash \phi$$

**Soundness**

*Soundness of Axiom*:

Assume $P, \pi \vdash ()$. Then, by definition of $\vdash$, $\pi$ must be a 1-path. Since we defined $()$ as an

empty transaction that holds for all paths of size 1, then $M, \pi \models ()$ for all interpretations $M$, and so $P, \pi \models ()$.
*Soundness of Rules*: This proof follow by separately proving the soundness of each of the rules in the definition of $SLD_{\mathcal{ETR}}$-derivation (rules r1–r5 of definition 22) as detailed in the auxiliary Lemma 14.

**Completeness**
This proof is inspired by the completeness proof of $\mathcal{TR}$'s Proof Theory [BK95]. Similarly to the proof for $\mathcal{TR}$, we construct a canonical interpretation (auxiliary Definition 92) of the program $P$, $M_P$, that intuitively collects all the results obtained by the proof theory. We then prove that $M_P$ is a model of $P$ (auxiliary Lemma 46).

Saying that $P, \pi \models \phi$ is equivalent to saying that for every model $M$ of $P$, $M, \pi \models \phi$. Given that, as we've established, $M_P$ is a model of $P$, it follows that $M_P, \pi \models \phi$.

Given that, all that remains to be proven is that if $M_P$ satisfies a formula $\phi$ in a given path $\pi$, then the procedure proves $\phi$ in that path, i.e., $P, \pi \vdash \phi$. This is done in auxiliary Lemmas 18 and 19. □

# B

# Proofs: Translating Event Algebras into Transaction Logic

In this appendix we prove the theorems of chapter 10.

**Theorem 8**. *Let $E$ be a* SNOOP *algebra expression without periodic and aperiodic operators, $H$ be a history containing the set of all SNOOP primitive events $e^i_j[t_1]$ that have occurred over the time interval $t_1, t_{max}$, and $\langle s_1, \ldots s_{max+1}\rangle$ be a path with size $t_{max} - t_1 + 1$. Let $\tau$ be the following function:*

$$
\begin{array}{lr}
\textbf{\textit{Primitive:}} & \tau(E) = \mathbf{o}(E) \textit{ where } E \textit{ is a primitive event} \\
\textbf{\textit{Sequence:}} & \tau(E_1; E_2) = \tau(E_1) \otimes \mathtt{path} \otimes \tau(E_2) \\
\textbf{\textit{Or:}} & \tau(E_1 \triangledown E_2) = \tau(E_1) \vee \tau(E_2) \\
\textbf{\textit{AND:}} & \tau(E_1 \triangle E_2) = [(\tau(E_1) \otimes \mathtt{path}) \wedge (\mathtt{path} \otimes \tau(E_2))] \vee \\
 & \qquad [(\tau(E_2) \otimes \mathtt{path}) \wedge (\mathtt{path} \otimes \tau(E_1))] \\
\textbf{\textit{NOT:}} & \tau(\neg(E_3)[E_1, E_2]) = (\tau(E_1) \otimes (\neg\tau(E_3)) \otimes \tau(E_2))
\end{array}
$$

*Then:*

$$\textit{If } [t_i, t_f] \in E[H] \textit{ then } \forall \text{M compatible with } H,\ M, \langle s_{t_i}, \ldots, s_{t_{f+1}}\rangle \models_{\mathcal{TR}} \tau(E)$$

*where, cf. [AC06], $E[H]$ is the set of time intervals $(t_i, t_f)$ where $E$ occurs over $H$ in an unrestricted context, and where $M$ is compatible with $H$ if, for each $e^i_j[t_i] \in H$: $M, \langle s_{t_i}, s_{t_{i+1}}\rangle \models_{\mathcal{TR}} \mathbf{o}(e_j)$.*

*Proof.* Next we prove the theorem 8 from page 106.

For that we enunciate our induction hypothesis as follows:

If $[t_i, t_f] \in E[H]$ then for all $M$*compatible with* $H$, $M, \langle s_{t_i}, \ldots, s_{t_{f+1}} \rangle \models_{\mathcal{TR}} \tau(E)$

We prove this claim by induction on the structure of $E$.

**Base Primitive:** $E = E_j$

Assume $[t_i, t_f] \in E_j[H]$. Since $E_j$ is a primitive event, it must exist a $e_j^i[t_i] \in H$ where $t_i = t_f$. Then, since $M$ is compatible with $H$ we have that $M, \langle s_{t_i}, s_{t_{i+1}} \rangle \models_{\mathcal{TR}} \mathbf{o}(e_j)$ as intended.

**Sequence:** $E = E_1; E_2$

Assume $[t_i, t_f] \in (E_1; E_2)[H]$.

By SNOOP's definition we know that this implies:

(1) $\exists t_{f1}, t_{i2}$ s.t. $t_i \leq t_{f1} < t_{i2} \leq t_f$ and

(2) $[t_i, t_{f1}] \in E_1[H]$, $[t_{i2}, t_f] \in E_2[H]$.

Then, we can apply the I.H. to (1) and (2) and respectively conclude that ∀M compatible with $H$, $M, \langle s_{t_i}, \ldots, s_{t_{f1+1}} \rangle \models_{\mathcal{TR}} \tau(E_1)$ and $M, \langle s_{t_{i2}}, \ldots, s_{t_{f+1}} \rangle \models_{\mathcal{TR}} \tau(E_2)$.

Additionally note that $\langle s_{t_i}, \ldots, s_{t_{f1+1}} \rangle$ and $\langle s_{t_{i2}}, \ldots, s_{t_f} \rangle$ are subsets of $\langle s_1, \ldots s_{max+1} \rangle$ where the former subpath occurs before the latter.

As a result, by the $\mathcal{TR}$'s satisfaction relation definition 3 and its serial-conjunction case, we know that $M, \langle s_{t_i}, \ldots, s_{t_{f1+1}}, \ldots s_{t_{i2}}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} \tau(E_1) \otimes \texttt{path} \otimes \tau(E_2)$ which is equivalent to $M, \langle s_{t_i}, \ldots, s_{t_{f1+1}}, \ldots s_{t_{i2}}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} \tau(E_1); \tau(E_2)$

**Or:** $E = E_1 \triangledown E_2$

Assume $[t_i, t_f] \in (E_1 \triangledown E_2)[H]$.

By SNOOP's definition we know that one of the following is true:

(1) $[t_i, t_f] \in E_1[H]$ or;

(2) $[t_i, t_f] \in E_2[H]$.

Then applying the I.H. to (1) and (2) we can conclude that ∀M compatible with $H$, either $M, \langle s_{t_i}, \ldots, s_{t_{f+1}} \rangle \models_{\mathcal{TR}} \tau(E_1)$ or $M, \langle s_{t_i}, \ldots, s_{t_{f+1}} \rangle \models_{\mathcal{TR}} \tau(E_2)$. From this, and from $\mathcal{TR}$'s satisfaction relation definition 3 we know that $M, \langle s_{t_i}, \ldots, s_{t_{f+1}} \rangle \models_{\mathcal{TR}} \tau(E_1) \vee \tau(E_2)$.

**AND:** $E = \tau(E_1 \triangle E_2)$

Assume $[t_i, t_f] \in (E_1 \triangle E_2)[H]$.

By SNOOP's definition we know that $\exists t_{f'}, t_{i'}$ s.t. $t_i \leq t_{f'} \leq t_{i'} \leq t_f$ and either one of the following two cases holds:

(1) $[t_i, t_{f'}] \in E_1[H]$, $[t_{i'}, t_f] \in E_2[H]$; or

(2) $[t_i, t_{f'}] \in E_2[H]$, $[t_{i'}, t_f] \in E_1[H]$

Let's assume (1). Then by applying the I.H. to this case we know that ∀M compatible with $H$, $M, \langle s_{t_i}, \ldots, s_{t_{f'}} \rangle \models_{\mathcal{TR}} \tau(E_1)$ and $M, \langle s_{t_{i'}}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} \tau(E_2)$.

Moreover, since $t_{f'} \leq t_f$ we know by $\mathcal{TR}$'s definition that $M, \langle s_{t_i}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} (\tau(E_1) \otimes \texttt{path})$ and $M, \langle s_{t_i}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} (\texttt{path} \otimes \tau(E_2))$. Thus $M, \langle s_{t_i}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} (\tau(E_1) \otimes \texttt{path}) \wedge (\texttt{path} \otimes \tau(E_2))$ and $M, \langle s_{t_i}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} [(\tau(E_1) \otimes \texttt{path}) \wedge (\texttt{path} \otimes \tau(E_2))] \vee [(\tau(E_2) \otimes \texttt{path}) \wedge (\texttt{path} \otimes \tau(E_1))]$

Let's assume (2). Then by applying the I.H. to this case we know that ∀M compatible with $H$, $M, \langle s_{t_i}, \ldots, s_{t_{f'}} \rangle \models_{\mathcal{TR}} \tau(E_2)$ and $M, \langle s_{t_{i'}}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} \tau(E_1)$.

Moreover, since $t_{f'} \leq t_f$ we know by $\mathcal{TR}$'s definition that $M, \langle s_{t_i}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} (\tau(E_2) \otimes \texttt{path})$ and $M, \langle s_{t_i}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} (\texttt{path} \otimes \tau(E_1))$. Thus $M, \langle s_{t_i}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} (\tau(E_2) \otimes \texttt{path}) \wedge (\texttt{path} \otimes \tau(E_1))$ and $M, \langle s_{t_i}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} [(\tau(E_1) \otimes \texttt{path}) \wedge (\texttt{path} \otimes \tau(E_2))] \vee [(\tau(E_2) \otimes \texttt{path}) \wedge (\texttt{path} \otimes \tau(E_1))]$.

**NOT:** $E = \neg(E_3)[E_1, E_2]$

Assume $[t_i, t_f] \in \neg(E_3)[E_1, E_2][H]$.

By SNOOP's definition we know that $\exists t_{f1}, t_{i2}$ s.t. $t_i \leq t_{f1} < t_{i2} \leq t_f$ s.t.:

(1) $[t_i, t_{f1}] \in E_1[H]$;

(2) $[t_{i2}, t_f] \in E_2[H]$ and;

(3) it is not the case that $\exists t_{i3}, t_{f3}$ where $t_{f1} < t_{i3} \leq t_{f3} < t_{i2}$ and $[t_{i3}, t_{f3}] \in E_3(H)$

From the case proof of sequence defined above and from (1) and (2), we entail that $M, \langle s_{t_i}, \ldots, s_{t_{f1+1}}, \ldots s_{t_{i2}}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} \tau(E_1); \tau(E_2)$, or in other notation, that $M, \langle s_{t_i}, \ldots, s_{t_{f1+1}}, \ldots s_{t_{i2}}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} \tau(E_1) \otimes \texttt{path} \otimes \tau(E_2)$.

Moreover, from the definition of compatibility it follows directly that, if $E$ is an atomic event $[t'_i, t'_f] \in E[H]$ iff $\forall M$ compatible with $H$ $M, \langle s_{t'_i}, \ldots s_{t'_{f+1}} \rangle \models \tau(E)$. Thus, if it is not the case that $\exists t_{i3}, t_{f3}$ where $t_{f1} < t_{i3} \leq t_{f3} < t_{i2}$ and $[t_{i3}, t_{f3}] \in E_3(H)$ then it is also not the case that $M, \langle s_{t_{i3}} \ldots s_{t_{f3}} \rangle \models \tau(E_3)$, and by definition of $\mathcal{TR}$'s satisfaction of negation $M, \langle s_{t_{i3}} \ldots s_{t_{f3}} \rangle \models \neg\tau(E_3)$.

As a result, we can conclude that $M, \langle s_{t_i}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} \tau(E_1) \otimes (\neg\tau(E_3)) \otimes \tau(E_2)$ which, by definition of path, is equivalent to say $M, \langle s_{t_i}, \ldots, s_{t_f} \rangle \models_{\mathcal{TR}} \tau(E_1) \otimes \neg(\texttt{path} \otimes \tau(E_3) \otimes \texttt{path}) \otimes \tau(E_2)$

□

**Theorem 9**. *Let $E$ be an* ETALIS *algebra expression without the events patterns that explicitly reference time: $E$* ***WHERE*** *$t$ and $(E).q$, and $R$ be a rule set of event rules of the form: $atom \leftarrow pattern$, where $pattern$ is an ETALIS algebra expression that also respects the previous restriction. Let $\epsilon$ be a history, or event stream, containing the set of all primitive events $e$ associated with time point $< t >$ that have occurred over the time interval $t_1, t_{max}$, and $\langle s_1, \ldots s_{max+1} \rangle$ be a path with size $t_{max} - t_1 + 1$. Let $\tau$ be the following function:*

$$
\begin{array}{rr}
\textbf{\textit{Primitive:}} & T(e) = \{\mathbf{o}(e), \mathbf{o}(e), \mathbf{o}(e)\} \textit{ where } E \textit{ is a primitive event} \\
\textbf{\textit{SEQ:}} & T(E_1 \texttt{ SEQ } E_2) = \{(T(E_1).expr; T(E_2).expr), T(E_1).start, T(E_2).end\} \\
\textbf{\textit{OR:}} & T(E_1 \texttt{ OR } E_2) = \{(T(E_1).expr \vee T(E_1).expr), \\
 & (T(E_1).start \vee T(E_2).start), (T(E_1).end \vee T(E_2).end)\} \\
\textbf{\textit{EQUALS:}} & T(E_1 \texttt{ EQUALS } E_2) = \{(T(E_1).expr \wedge T(E_2).expr), \\
 & (T(E_1).start \wedge T(E_2).start), (T(E_1).end \wedge T(E_2).end)\} \\
\textbf{\textit{NOT:}} & T(\texttt{NOT}(E_3)[E_1, E_2]) = \{(T(E_1).expr \ ; \ \neg T(E_3).expr \ ; \ T(E_2).expr), T(E_1).start, \\
 & T(E_2).end\} \\
\textbf{\textit{AND:}} & T(E_1 \texttt{ AND } E_2) = \{(T(E_1).expr \ ; \ T(E_2).expr) \vee (T(E_2).expr \ ; \ T(E_1).expr), \\
 & (T(E_1).start \vee T(E_2).start), (T(E_1).end \vee T(E_2).end)\} \\
\textbf{\textit{MEETS:}} & T(E_1 \texttt{ MEETS } E_2) = \{(T(E_1).expr \setminus T(E_1).end); (T(E_1).end \wedge T(E_2).start) \ ; \\
 & (T(E_2).expr \setminus T(E_2).start), T(E_1).start, T(E_2).end\} \\
\textbf{\textit{STARTS:}} & T(E_1 \texttt{ STARTS } E_2) = \{(T(E_1).start \wedge T(E_2).start) \ ; \\
 & ((T(E_1).expr \setminus T(E_1).start \otimes \texttt{path}) \wedge T(E_2).middle) \ ; \ T(E_2).end, \\
 & (T(E_1).start \wedge T(E_2).start), T(E_2).end\} \\
\textbf{\textit{FINISHES:}} & T(E_1 \texttt{ FINISHES } E_2) = \{T(E_2).start \ ; \\
 & ((\texttt{path} \otimes T(E_1).expr \setminus T(E_1).end) \wedge T(E_2).middle) \ ; \\
 & (T(E_2).end \wedge T(E_1).end), \\
 & T(E_1).start, (T(E_1).end \wedge T(E_2).end)\} \\
\textbf{\textit{PAR:}} & T(E_1 \texttt{ PAR } E_2) = \{[(T(E_1).start \wedge T(E_2).start) \vee (T(E_1).start; T(E_2).start) \vee \\
 & (T(E_2).start; T(E_1).start)]; \\
 & [(\texttt{path} \otimes T(E_1).middle \otimes \texttt{path}) \wedge (\texttt{path} \otimes T(E_2).middle \otimes \texttt{path})] \\
 & [(T(E_1).end \wedge T(E_2).end) \vee (T(E_1).end; T(E_2).end) \vee \\
 & (T(E_2).end; T(E_1).end)], \\
 & (T(E_1).start \wedge T(E_2).start) \vee T(E_1).start \vee T(E_2).start, \\
 & (T(E_1).end \wedge T(E_2).end) \vee T(E_1).end \vee T(E_2).end\}
\end{array}
$$

*If* $< t_i, t_f > \in \mathcal{I}(E)$ *then* $\forall$M compatible with $\epsilon$, $M, \langle s_{t_i}, \ldots, s_{t_{f+1}} \rangle \models_{\mathcal{TR}} T(E).expr$

*where, cf. [AFRSSS10], $\mathcal{I}$ is the minimal model of $\epsilon$ and an empty rule set $R$, and $\mathcal{I}(E)$ is the set of time intervals $< t_i, t_f >$ where $E$ occurs over $\epsilon$ w.r.t. the rule set $R$; and where $M$ is compatible with $\epsilon$ if, for each $< t_i > \in \epsilon(e_j)$: $M, \langle s_{t_i}, s_{t_{i+1}} \rangle \models_{\mathcal{TR}} \mathbf{o}(e_j)$.*

*In addition, if we try to translate $E.middle$, and $E.middle$ is not defined (because $E.expr = E.start$), then the whole expression is translated to $\perp$, which is false in any path of any length.*

*Proof.* In the following we prove theorem 9 as stated in page 111

We aim to prove the following hypothesis:

$$< t_i, t_f > \in \mathcal{I}(E) \text{ then } M, \langle s_{t_i}, \ldots, s_{t_{f+1}} \rangle \models_{\mathcal{TR}} T(E).expr$$

We prove by induction on the structure of $E$.

**Base Case:** $E$ is a primitive event true in $< t_i, t_i >$ and $T(E).expr = \mathbf{o}(E)$.

Since $E$ is a primitive event, then $< t_i, t_i > \in \mathcal{I}(E)$ iff $(t_i) \in \epsilon(E)$. Since $M$ must be compatible with $\epsilon$, then we know by definition of $M$ that $M, \langle s_i, s_{i+1} \rangle \models \mathbf{o}(E)$.

**Sequence:** $E$ is $E_1$ `SEQ` $E_2$ and we want to prove that $M, \langle s_i, \ldots, s_{f+1} \rangle \models T(E_1).expr \ ; \ T(E_2).expr$

We know $< t_i, t_f > \in \mathcal{I}(E_1$ `SEQ` $E_2)$, then by definition of ETALIS, we know that:

$$\exists t_{f1}, t_{i2} \text{ s.t. } < t_i, t_{f1} > \in \mathcal{I}(E_1) \text{ and } < t_{i2}, t_f > \in \mathcal{I}(E_2).$$

where $t_i \leq t_{f1} < t_{i2} \leq t_f$.

From this, applying the I.H. to each of the previous statements we know that:

$$M, \langle s_i, \ldots, s_{f1+1} \rangle \models_{\mathcal{TR}} T(E_1).expr \text{ and } M, \langle s_{i2}, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).expr$$

Then, since $t_i \leq t_{f1} < t_{i2} \leq t_f$ we know that $s_{f1+1} \leq s_{i2}$ and thus by definition of $\otimes$ and `path`:

$$M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).expr \otimes \texttt{path} \otimes T(E_2).expr$$

which is equivalent to: $M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).expr \ ; \ T(E_2).expr$

**Or:** $E$ is $E_1$ `SEQ` $E_2$ and we want to prove that $M, \langle s_i, \ldots, s_{f+1} \rangle \models T(E_1).expr \vee T(E_2).expr$

We know $< t_i, t_f > \in \mathcal{I}(E_1$ `OR` $E_2)$, then by definition of ETALIS, we know that:

$$< t_i, t_f > \in \mathcal{I}(E_1) \text{ or } < t_i, t_f > \in \mathcal{I}(E_2).$$

From this, applying the I.H. to each of the previous statements we know that:

$$M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).expr \text{ or } M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).expr$$

which by definition of $\vee$ and $\mathcal{TR}$ is equivalent to:

$$M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).expr \vee T(E_2).expr$$

**Equals:** $E$ is $E_1$ `SEQ` $E_2$ and we want to prove that $M, \langle s_i, \ldots, s_{f+1} \rangle \models T(E_1).expr \wedge T(E_2).expr$

We know $< t_i, t_f > \in \mathcal{I}(E_1$ `AND` $E_2)$, then by definition of ETALIS, we know that:

$$< t_i, t_f > \in \mathcal{I}(E_1) \text{ and } < t_i, t_f > \in \mathcal{I}(E_2).$$

From this, applying the I.H. to each of the previous statements we know that:

$$M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).expr \text{ and } M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).expr$$

which by definition of $\vee$ and $\mathcal{TR}$ is equivalent to:

$$M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).expr \wedge T(E_2).expr$$

**Not:** $E$ is $\texttt{NOT}(E_3)[E_1, E_2]$ and we want to prove that
$M, \langle s_i, \ldots, s_{f+1} \rangle \models T(E_1).expr \ ; \ \neg T(E_3).expr \ ; \ T(E_2).expr$

We know $< t_i, t_f > \in \mathcal{I}(\texttt{NOT}(E_3)[E_1, E_2])$, then by definition of ETALIS, we know that:

1) $\exists t_{f1}, t_{i2}$ s.t. $< t_i, t_{f1} > \in \mathcal{I}(E_1), < t_{i2}, t_f > \in \mathcal{I}(E_2)$ and;
2) $\neg\exists t_{i3}, t_{f3}$ s.t. $< t_{i3}, t_{f3} \in \mathcal{I}(E_3)$ and $t_{f1} < t_{i3} \leq t_{f3} < t_{i2}$

From applying I.H. to 1) we know that:

$$M, \langle s_i, \ldots, s_{f1+1} \rangle \models_{\mathcal{TR}} T(E_1).expr \text{ and } M, \langle s_{i2}, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).expr$$

and $M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).expr \ ; \ T(E_2).expr$

Recall that by definition of negation in ETALIS, $E_3$ must be an atomic event. Moreover, from the definition of compatibility it follows directly that, if $E_3$ is an atomic event $< t'_i, t'_f > \in \epsilon(E_3)$ iff $\forall M$ compatible with $\epsilon$ $M, \langle s_{t'_i}, \ldots s_{t'_{f+1}} \rangle \models \tau(E_3)$. Thus, if it is not the case that $\exists t_{i3}, t_{f3}$ where $t_i < t_{i3} \leq t_{f3} < t_f$ and $< t_{i3}, t_{f3} > \in \mathcal{I}(E_3)$ then it is also not the case that $M, \langle s_{t_{i3}} \ldots s_{t_{f3}} \rangle \models T(E_3).expr$, and by definition of $\mathcal{TR}$'s satisfaction of negation $M, \langle s_{t_{i3}} \ldots s_{t_{f3}} \rangle \models \neg T(E_3).expr$.

As a result we conclude $M, \langle s_i, \ldots, s_{f+1} \rangle \models T(E_1).expr \otimes \neg(\texttt{path} \otimes T(E_3).expr \otimes \texttt{path}) \otimes (E_2).expr$

**And:** $E$ is $E_1$ $\texttt{AND}$ $E_2$ and we want to prove that
$M, \langle s_i, \ldots, s_{f+1} \rangle \models (T(E_1).expr \ ; \ T(E_2).expr) \vee (T(E_2).expr \ ; \ T(E_1).expr)$

We know $< t_i, t_f > \in \mathcal{I}(E_1 \ \texttt{AND} \ E_2)$, then by definition of ETALIS, we know that there are $t_{f1}, t_{i2}$ such that $t_i \leq t_{f1} < t_{i2} \leq t_f$, and one of the following cases is true:

1) $< t_i, t_{f1} > \in \mathcal{I}(E_1)$ and $< t_{i2}, t_f > \in \mathcal{I}(E_2)$ or
2) $< t_i, t_{f1} > \in \mathcal{I}(E_2)$ and $< t_{i2}, t_f > \in \mathcal{I}(E_1)$

From this, applying the I.H. to 1) we know that:

$$M, \langle s_i, \ldots, s_{f1+1} \rangle \models_{\mathcal{TR}} T(E_1).expr \text{ and } M, \langle s_{i2}, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).expr$$

Conversely, by applying the I.H. to 2) we know that:

$$M, \langle s_i, \ldots, s_{f1+1} \rangle \models_{\mathcal{TR}} T(E_2).expr \text{ and } M, \langle s_{i2}, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).expr$$

Then, since $t_i \leq t_{f1} < t_{i2} \leq t_f$ we know that $s_{f1+1} \leq s_{i2}$ and thus by definition of $\otimes$ and $\texttt{path}$: $M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).expr \ ; \ T(E_2).expr$ or $M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).expr \ ; \ T(E_1).expr$ which by definition of $\vee$ in $\models_{\mathcal{TR}}$ is equivalent to:
$M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} (T(E_1).expr \ ; \ T(E_2).expr) \vee (T(E_2).expr \ ; \ T(E_1).expr)$

**Starts:** $E$ is $E_1$ `STARTS` $E_2$ and we want to prove that
$M, \langle s_i, \ldots, s_{f+1} \rangle \models (T(E_1).start \wedge T(E_2).start)\ ;((T(E_1).expr \backslash T(E_1).start \otimes \texttt{path}) \wedge T(E_2).middle)\ ;\ T(E_2).end$

We know $< t_i, t_f > \in \mathcal{I}(E_1$ `STARTS` $E_2)$, then by definition of ETALIS, we know that there is a $t_j$ s.t. $t_i \leq t_j < t_f$ and:

$$< t_i, t_j > \in \mathcal{I}(E_1) \text{ and } < t_i, t_f > \in \mathcal{I}(E_2)$$

From this, applying the I.H. for each of this statements we know that:

a) $M, \langle s_i, \ldots, s_{j+1} \rangle \models_{\mathcal{TR}} T(E_1).expr$

b) $M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).expr$

From this, by definition of $T(E)$ we know

a1) $M, \langle s_i, s_{i+1} \rangle \models_{\mathcal{TR}} T(E_1).start$

a2) $M, \langle s_{i+1}, \ldots, s_{j+1} \rangle \models_{\mathcal{TR}} T(E_1).expr \setminus T(E_1).start$

a3) $M, \langle s_{i+1}, \ldots, s_f \rangle \models_{\mathcal{TR}} T(E_1).expr \setminus T(E_1).start \otimes \texttt{path}$ (since $t_j < t_f$)

And also, since $T(E_2).start \neq T(E_2).expr$:

b1) $M, \langle s_i, s_{i+1} \rangle \models_{\mathcal{TR}} T(E_2).start$

b2) $M, \langle s_{i+1}, \ldots, s_f \models_{\mathcal{TR}} T(E_2).middle$

b3) $M, \langle s_f, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).end$

Then by a1) and b1) we know that $M, \langle s_i, s_{i+1} \rangle \models_{\mathcal{TR}} T(E_1).start \wedge T(E_2).start$. Moreover, by a3) and b2) we know:
$M, \langle s_{i+1}, \ldots, s_f \rangle \models_{\mathcal{TR}} T(E_1).middle \wedge (T(E_1).expr \setminus T(E_1).start \otimes \texttt{path})$

From this, we can conclude that:
$M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} (T(E_1).start \wedge T(E_2).start)\ ;((T(E_1).expr \backslash T(E_1).start \otimes \texttt{path}) \wedge T(E_2).middle)\ ;\ T(E_2).end$

**Finishes:** $E$ is $E_1$ `FINISHES` $E_2$ and we want to prove that
$M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).start\ ;\ ((\texttt{path} \otimes T(E_1).expr \setminus T(E_1).end) \wedge T(E_2).middle)\ ;$
$(T(E_2).end \wedge T(E_1).end)$

We know $< t_i, t_f > \in \mathcal{I}(E_1$ `STARTS` $E_2)$, then by definition of ETALIS, we know that there is a $t_j$ s.t. $t_i < t_j \leq t_f$ and:

$$< t_j, t_f > \in \mathcal{I}(E_1) \text{ and } < t_i, t_f > \in \mathcal{I}(E_2)$$

From this, applying the I.H. for each of this statements we know that:

a) $M, \langle s_j, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).expr$

b) $M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).expr$

From this, by definition of $T(E)$ we know

a1) $M, \langle s_j, \ldots, s_f \rangle \models_{\mathcal{TR}} T(E_1).expr \setminus T(E_1).end$

a2) $M, \langle s_j, \ldots, s_f \rangle \models_{\mathcal{TR}} \texttt{path} \otimes T(E_1).expr \setminus T(E_1).end$ (since $t_i < t_j$)

a3) $M, \langle s_f, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).end$

And also $T(E_2).start \neq T(E_2).expr$:

b1) $M, \langle s_i, s_{i+1} \rangle \models_{\mathcal{TR}} T(E_2).start$

b2) $M, \langle s_{i+1}, \ldots, s_f \rangle \models_{\mathcal{TR}} T(E_2).middle$

b3) $M, \langle s_f, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).end$

By a3) and $b3$) we know: $M, \langle s_f, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).end \wedge T(E_1).end$.

By a2) and b2) we know:
$M, \langle s_{i+1}, \ldots, s_f \rangle \models_{\mathcal{TR}} T(E_2).middle \wedge (\texttt{path} \otimes T(E_1).expr \setminus T(E_1).end)$

and finally from the latter and b1):
$M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).start \; ; \; ((\texttt{path} \otimes T(E_1).expr \setminus T(E_1).end) \wedge T(E_2).middle) \; ;$
$(T(E_2).end \wedge T(E_1).end)$

**Parallel:** $E$ is $E_1$ PAR $E_2$ and we want to prove that:

$$
\begin{aligned}
&M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} \\
&\quad [(T(E_1).start \wedge T(E_2).start) \vee (T(E_1).start; T(E_2).start) \vee (T(E_2).start; T(E_1).start)]; \\
&\quad [(\texttt{path} \otimes T(E_1).middle \otimes \texttt{path}) \wedge (\texttt{path} \otimes T(E_2).middle \otimes \texttt{path})] \\
&\quad [(T(E_1).end \wedge T(E_2).end) \vee (T(E_1).end; T(E_2).end) \vee (T(E_2).end; T(E_1).end)]
\end{aligned}
$$

We know if $< t_i, t_f > \in \mathcal{I}(E_1$ PAR $E_2)$, then by definition of ETALIS, we know that there is a $t_j, t_k$ s.t. $t_i \leq t_j < t_k \leq t_f$ and one of the two statements are true:

S1: $< t_i, t_k > \in \mathcal{I}(E_1)$ and $< t_j, t_f > \in \mathcal{I}(E_2)$; or

S2: $< t_i, t_k > \in \mathcal{I}(E_2)$ and $< t_j, t_f > \in \mathcal{I}(E_1)$

Based on this we can apply the I.H.:

S1a) $M, \langle s_i, \ldots s_{k+1} \rangle \models_{\mathcal{TR}} T(E_1).expr$

S1b) $M, \langle s_j, \ldots s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).expr$

and

S2a) $M, \langle s_i, \ldots s_{k+1} \rangle \models_{\mathcal{TR}} T(E_2).expr$

S2b) $M, \langle s_j, \ldots s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).expr$

From this, and since $T(E_1).start \neq T(E_1).expr$ and $T(E_2).start \neq T(E_2).expr$ we know:

S1a1) $M, \langle s_i, s_{i+1} \rangle \models_{\mathcal{TR}} T(E_1).start$

S1a2) $M, \langle s_{i+1}, \ldots, s_k \rangle \models_{\mathcal{TR}} T(E_1).middle$

S1a3) $M, \langle s_k, s_{k+1} \rangle \models_{\mathcal{TR}} T(E_1).end$

S1b1) $M, \langle s_j, s_{j+1} \rangle \models_{\mathcal{TR}} T(E_2).start$

S1b2) $M, \langle s_{j+1} \ldots, s_f \rangle \models_{\mathcal{TR}} T(E_2).middle$

S1b3) $M, \langle s_f, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_2).end$

S2a1) $M, \langle s_i, s_{i+1} \rangle \models_{\mathcal{TR}} T(E_2).start$

S2a2) $M, \langle s_{i+1}, \ldots, s_k \rangle \models_{\mathcal{TR}} T(E_2).middle$

S2a3) $M, \langle s_k, s_{k+1} \rangle \models_{\mathcal{TR}} T(E_2).end$

S2b1) $M, \langle s_j, s_{j+1} \rangle \models_{\mathcal{TR}} T(E_1).start$

S2b2) $M, \langle s_{j+1} \ldots, s_f \rangle \models_{\mathcal{TR}} T(E_1).middle$

S2b3) $M, \langle s_f, s_{f+1} \rangle \models_{\mathcal{TR}} T(E_1).end$

So by S1a1 and S1b1 we know regarding the starting of the expression:

$t_i < t_j \ : M, \langle s_i, \ldots s_{j+1} \rangle \models_{\mathcal{TR}} T(E_1).start; T(E_2).start$ or

$t_i = t_j \ : M, \langle s_i, \ldots s_{i+1} \rangle \models_{\mathcal{TR}} T(E_1).start \wedge T(E_2).start$

and from S2a1 and S2b1:

$t_i < t_j \ : M, \langle s_i, \ldots s_{j+1} \rangle \models_{\mathcal{TR}} T(E_2).start; T(E_1).start$ or

$t_i = t_j \ : M, \langle s_i, \ldots s_{i+1} \rangle \models_{\mathcal{TR}} T(E_1).start \wedge T(E_2).start$

And from these we can conclude that:
$M, \langle s_i, \ldots s_{j+1} \rangle \models_{\mathcal{TR}} (T(E_1).start \wedge T(E_2).start) \vee (T(E_1).start; T(E_2).start) \vee (T(E_2).start; T(E_1).start)$

Applying the same deduction for the end of the expression we can conversely conclude: $M, \langle s_k, \ldots s_{f+1} \rangle \models_{\mathcal{TR}} (T(E_1).end \wedge T(E_2).end) \vee (T(E_1).end; T(E_2).end) \vee (T(E_2).end; T(E_1).end)$

Moreover from S1a2, S1b2, S2a1 and S2b2, it holds:

$M, \langle s_{i+1} \ldots s_f \rangle \models_{\mathcal{TR}} \texttt{path} \otimes T(E_1).middle \otimes \texttt{path}$
$M, \langle s_{i+1} \ldots s_f \rangle \models_{\mathcal{TR}} \texttt{path} \otimes T(E_2).middle \otimes \texttt{path}$ and
$M, \langle s_{i+1} \ldots s_f \rangle \models_{\mathcal{TR}} (\texttt{path} \otimes T(E_1).middle \otimes \texttt{path}) \wedge (\texttt{path} \otimes T(E_2).middle \otimes \texttt{path})$

From all these, and since $t_i \leq t_j < t_k \leq t_f$ we know:

$$
\begin{array}{l}
M, \langle s_i, \ldots, s_{f+1} \rangle \models_{\mathcal{TR}} \\
\quad [(T(E_1).start \wedge T(E_2).start) \vee (T(E_1).start; T(E_2).start) \vee (T(E_2).start; T(E_1).start)]; \\
\quad [(\texttt{path} \otimes T(E_1).middle \otimes \texttt{path}) \wedge (\texttt{path} \otimes T(E_2).middle \otimes \texttt{path})] \\
\quad [(T(E_1).end \wedge T(E_2).end) \vee (T(E_1).end; T(E_2).end) \vee (T(E_2).end; T(E_1).end)]
\end{array}
$$

□

# Proofs: Transaction Logic with Events

This appendix contains proofs of propositions and theorems of chapter 11. In order to simplify the notation, in this chapter we write $\texttt{exp}_M(\pi)$ as the set of possible expansion paths obtained from path $\pi$ w.r.t. $M$. As such, if $\pi' \in \texttt{exp}_M(\pi)$ then $\pi'$ is an extension of path $\pi$ w.r.t. $M$ (cf. definition 53).

**Lemma 3**.*[Support] Let $P$ be a program, $\pi$ a path, $\phi$ a transaction atom. Then, if $P, \pi \models \phi$ one of the following holds:*

1. *$\phi$ is an elementary action and either $\phi \in \mathcal{O}^d(\pi)$ or $\phi \in \mathcal{O}^t(\pi)$;*

2. *$\phi$ is the head of a transaction rule in $P$ ($\phi \leftarrow$ body) and $P, \pi \models$ body.*

*Proof.* Next we prove the lemma 3 from page 135.

Clearly by definition 47 and definition 58 both items are true in all minimal models of program $P$. As such, it remains to show that every $\phi$ that holds in a path arises from these. Let's assume that $\phi$ does not fall in either of the two previous cases. Then, since $\phi$ is a transaction atom, then by definition of $\mathcal{TR}^{ev}$ language, $\phi$ must either be (1) primitive defined in the oracles but where $\phi \notin \mathcal{O}^d(\pi)$ and $\phi \notin \mathcal{O}^t(\pi)$; or (2) a transaction name that does not occur in any the head; or (3) $\phi$ appears in the body of a rule whose head is *not* true;

Let's consider each of these cases individually.
(1) if $\phi$ is an oracle primitive, then $\phi$ cannot appear in the head of rules in $P$. Additionally, since the oracles do not make $\phi$ true in $\pi$, then there is no reason for an interpretation that is a minimal model of $P$ to make $\phi$ true in $\pi$. And thus, a minimal model that only respects the oracle primitives and the rules of $P$ does not make $\phi$ true in $\pi$, and thus $\phi$ is not true in *all* minimal models of $P$, and $P, \pi \not\models \phi$.

(2) if $\phi$ is a transaction name that is not the head of a transaction rule, then $\phi$ cannot be true in any minimal model of $P$. Moreover, if $\phi$ appears in the head of a rule where $P, \pi \not\models body$, then it means that at least one minimal models fails to satisfy $body$ in path $\pi$. As such, it means that at in that minimal model $\phi$ does not need to be true from that rule in path $\pi$.
(3) The latter case is the trickiest and is the case where for all minimal models $M$, they must satisfy a rule $head \leftarrow body$ in $\pi$ and the head is known to be false in all minimal models for path $\pi$. However, for a given head to be false in all minimal models, it means that $M, \pi \models head$ is not the case in every minimal model of $M$. For that to be true, and since such rule exists and $body$ is true in $\pi$ for all minimal models, then $\neg head$ must be a direct consequence of the program. Since negation cannot appear in the head of rules, and the failure of a primitive oracle does not directly cause failure in the interpretation (cf. definition 47), then this case is impossible.

□

## C.1 Comparison with $\mathcal{TR}$

In the following we show all the auxiliary lemmas and definitions to prove theorem 10, i.e., to show that $\mathcal{TR}^{ev}$ indeed extends $\mathcal{TR}$. More precisely, we show that if $P$ does not have complex event rules and if $P'$ is the program obtained from $P$ as defined in definition 94, then both programs prove the same formulas in a transformed path (cf. definition 93).

With that as goal, given an interpretation $M$ in $\mathcal{TR}^{ev}$, we construct an equivalent interpretation $t(M)$ in $\mathcal{TR}$ (cf. definition 95). Then we prove that all formulas are satisfied in $M$ iff they are satisfied in $t(M)$, and if an interpretation $M$ is a model of $P$ then $t(M)$ is also a model of $P'$. Conversely, given an interpretation $M$ in $\mathcal{TR}$ we construct an interpretation $M^{-1}$ in $\mathcal{TR}^{ev}$ (cf. definition 97), and prove that both interpretations model the same formulas in the same paths, and that if $M$ is a model of $P'$, then the $M^{-1}$ is a model of $P$. Then we show that $t(M)$ and $M^{-1}$ are bijective functions, and that they achieve minimal models, if they are constructed based on minimal models. With this, we can show that for every model of $P$, and every model of $P'$, all minimal models prove the same formulas in the same paths.

**Definition 93** (Path transformation)**.** *Let $\pi$ be an annotated path from $\mathcal{TR}^{ev}$.*

*We define the $\mathcal{TR}$ path obtained from $\pi$ as the path $\pi^{\mathcal{TR}}$ which is the path obtained from $\pi$ removing the annotations and for every event occurrence $\varphi$ in $\pi$ s.t.$\pi = \pi_1 \circ \langle D \xrightarrow{\varphi} D \rangle \circ \pi_2$, then $\pi^{\mathcal{TR}} = \pi_1 \circ \langle D \rangle \circ \pi_2$.*

**Definition 94** (Program transformation)**.** *Let $P$ be a $\mathcal{TR}^{ev}$ program s.t. it does not contain complex event rules.*

*$P'$ is the program obtained from $P$ as follows:*

- *for every transaction formula $e$ and $\mathbf{r}(e)$ s.t. $e \in \mathcal{P}_e$, replace it with $p_e$ (where $p_e$ does not belong to $P$).*
- *If $\mathbf{r}(\phi)$ is in the head of some rule in $P$ and $\phi \in P_{\mathcal{O}}$, then substitute all instances of $\phi$ in $P'$ by $new_\phi$ and add the rule: $new_\phi \leftarrow \phi \otimes \mathbf{r}(\phi)$.*
- *keep all the remaining rules of $P$ in $P'$.*

To prove this result we will first construct an interpretation $t$ that maps an interpretation in $\mathcal{TR}^{ev}$ into an interpretation in $\mathcal{TR}$ (definition 95). Afterwards we will show that, if $M$ is a model of $P$ in $\mathcal{TR}^{ev}$, then $t(M)$ is also a model of $P$ in $\mathcal{TR}$ (lemma 29).

**Definition 95** ($\mathcal{TR}$ function)**.** *We define $t$ as a function from interpretations to interpretations defined as follows:*

- *$\forall \phi, \pi$ such that $\phi$ is an atomic formula in $\mathcal{P}_{\mathcal{O}} \cup \mathcal{P}_t$:*

$$\text{If } \phi \in M(\pi) \text{ then } \phi \in t(M)(\pi^{\mathcal{TR}})$$

- *$\forall e \in \mathcal{P}_e$ such that $e \in M(\pi_1 \circ \pi_2) \wedge \mathbf{o}(e) \in M(\pi_1) \wedge \mathbf{r}(e) \in M(\pi_2) \wedge \pi = \pi_1 \circ \pi_2$ then $p_e \in t(M)(\pi^{\mathcal{TR}})$*
- *$\forall \phi \in \mathcal{P}_{\mathcal{O}}$ such that $\pi_1 \circ \pi_2 = \pi$, $\phi \in M(\pi_1)$, $\mathbf{r}(\phi) \in M(\pi_2)$, then $p_\phi \in t(M)(\pi^{\mathcal{TR}})$*
- *Nothing else belongs to $t(M)$*

*where $\pi^{\mathcal{TR}}$ is the $\mathcal{TR}$ path obtained from $\pi$ as defined in definition 93*

**Lemma 22.** *Let $M_1$ and $M_2$ be interpretations in $\mathcal{TR}^{ev}$*

$$M_1 \subset M_2 \Rightarrow t(M_1) \subset t(M_2)$$
$$M_1 = M_2 \Rightarrow t(M_1) = t(M_2)$$

*Proof.* Immediate by definition 95 □

**Lemma 23.** *Let $M$ be a minimal model of a program $P$ which does not have complex event rules. Then $\pi \notin \texttt{exp}_M(\pi)$ iff one of the following is true:*

*a) $\exists \phi \in \mathcal{P}_{\mathcal{O}}$ s.t. $\phi \in M(\pi_1)$ where $\pi_1$ is a subpath of $\pi$, $\mathbf{r}(\phi) \leftarrow body \in P$ and $M, \pi_1 \not\models body$. In this case we say that $\phi \notin \mathcal{O}_{static}$.*

*b) $\exists e \in \mathcal{P}_{\mathcal{O}}$ s.t. $e \in M(\langle D \xrightarrow{\mathbf{o}(e)} D \rangle)$ where $\langle D \xrightarrow{\mathbf{o}(e)} D \rangle$ is a subpath of $\pi$, $\mathbf{r}(e) \leftarrow body \in P$ and $M, \langle D \xrightarrow{\mathbf{o}(e)} D \rangle \not\models body$*

*Proof.* Immediately by definition of minimal model and $\texttt{exp}_M(\pi)$ (cf. definition 53). □

**Definition 96** ($\mathcal{O}_{static}$)**.** *We define $\mathcal{O}_{static}$ as the subset of $\mathcal{P}_{\mathcal{O}}$ w.r.t. a program $P$ s.t. $\phi$ only has a response rule $\mathbf{r}(\phi) \leftarrow body$ where $body = \texttt{true}$. We define $\mathcal{O}_{reactive}$ as the complement of $\mathcal{O}_{static}$ w.r.t. $\mathcal{P}_{\mathcal{O}}$ .*

**Lemma 24.** *Let $\phi$ be a formula in $\mathcal{P}_t$. Let $M$ be a minimal model of a program $P$.*

*If $\phi \in M(\pi)$ then $\pi \in \texttt{exp}_M(\pi)$.*

*Proof.* Trivially, by definition of minimal models (definition 58) and the base case of definition 50. □

**Lemma 25.** *Let $M$ be a minimal model of a program $P$ without complex event rules. Let $\pi_1$ and $\pi_2$ be splits of path $\pi$*

*If $\pi_1 \in \texttt{exp}_M(\pi_1)$ and $\pi_2 \in \texttt{exp}_M(\pi_2)$ then $\pi \in \texttt{exp}_M(\pi)$*

*Proof.* Trivially since $M$ is a minimal model (definition 58) and $P$ does not have complex event rules. □

**Lemma 26.** *Let $M$ be a minimal model of a program $P$ which does not have complex event rules. Then for every path $\pi^{\mathcal{TR}}$ in $\mathcal{TR}$, and every formula $\phi$ which does not contain formulas in $\mathcal{O}_{reactive}$ and in $\mathcal{P}_e$*

$$t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi \textit{ iff } M, \pi \models \phi$$

*where $\pi$ is the path obtained from $\pi^{\mathcal{TR}}$ by only adding annotated transitions.*

*Proof.* We prove this statement for each direction.

($\Rightarrow$):

For this proof we assume $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$ and show that $M, \pi \models \phi$ We show this by induction on the structure of $\phi$.

**Base Case** $\phi \in \mathcal{O}_{static}$ or $\phi \in \mathcal{P}_t$

If $\phi$ is an atom, then $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$ iff $\phi \in t(M)()^{\mathcal{TR}}$.

Then by definition 95, we know that $\phi \in t(M)()^{\mathcal{TR}}$ only if $\phi \in M(\pi)$.

Let's assume $\phi \in \mathcal{O}_{static}$. Since $\mathbf{r}(\phi) \leftarrow body \notin P$ then $\pi \in \texttt{exp}_M(\pi)$ and $M, \pi \models \phi$

Conversely, assume $\phi \in \mathcal{P}_t$, then by lemma 24 we know that $\pi \in \texttt{exp}_M(\pi)$ and thus $M, \pi \models \phi$ as we intended to prove.

**Serial Conjunction** $\phi = \phi_1 \otimes \phi_2$

By definition of $\mathcal{TR}$ satisfaction, we know that: $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1 \otimes \phi_2$ iff there is a split of $\pi^{\mathcal{TR}} = \pi_1^{\mathcal{TR}} \circ \pi_2^{\mathcal{TR}}$ s.t. $t(M), \pi_1^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ and $t(M), \pi_2^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_2$.

From this, we can apply the Induction Hypothesis to $\phi_1$ and $\phi_2$ and conclude: $M, \pi_1 \models \phi_1$ and $M, \pi_2 \models \phi_2$. From the latter we know that $\pi_1 \in \texttt{exp}_M(\pi_1)$ and $\pi_2 \in \texttt{exp}_M(\pi_2)$ and by lemma 25 we know that $\pi \in \texttt{exp}_M(\pi)$, and thus $M, \pi \models \phi_1 \otimes \phi_2$

**Negation** $\phi = \neg\phi$

By definition of $\mathcal{TR}$ satisfaction, we know that: $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \neg\phi$ iff it is not the case that $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$. Since by induction hypothesis $\phi$ $M, \pi \models \phi$ iff $t(M)\pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$, then if it is not the case that $M, \pi \models \phi$, and thus $M, \pi \models \neg\phi$.

**Disjunction** $\phi = \phi_1 \vee \phi_2$

By definition of $\mathcal{TR}$ satisfaction, we know that: $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1 \vee \phi_2$ iff either $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ or $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_2$.

Then by Induction Hypothesis, we know that either $M, \pi \models \phi_1$ or $M, \pi \models \phi_2$, which allows us to conclude $M, \pi \models \phi_1 \vee \phi_2$

**Executional Possibility** $\phi = \Diamond\phi$

By definition of $\mathcal{TR}$ we know that: $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \Diamond\phi$ iff $\pi^{\mathcal{TR}}$ is a 1-path of the form $\langle D \rangle$ for some state $D$ and $t(M), \pi'^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$ for some path $\pi'^{\mathcal{TR}}$ that begins at $D$.

Moreover, we can apply the Induction Hypothesis for $\phi$ and conclude that $M, \pi' \models \phi$ for a path $\pi'$ that begins at $D$. Then by the definition of $\mathcal{TR}^{ev}$ satisfaction of transaction formulas we know that $M, \langle D \rangle \models \Diamond\phi$.

($\Leftarrow$):

**Base Case** $\phi \in \mathcal{O}_{static}$ or $\phi \in \mathcal{P}_t$

If $\phi$ is an atom, then $M, \pi \models \phi$ iff $\phi \in M(\pi')$ and $\pi \in \mathtt{exp}_M(\pi')$. Since $\phi \in \mathcal{O}_{static}$ and $\mathbf{r}(\phi) \leftarrow body \notin P$ then $\phi \in M(\pi)$ and $\pi \in \mathtt{exp}_M(\pi)$. From this, we know that $\phi \in t(M)(\pi^{\mathcal{TR}})$ and thus $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$

Conversely, assume $\phi \in \mathcal{P}_t$, then by lemma 24 we know that $\pi \in \mathtt{exp}_M(\pi)$ and thus $M, \pi \models \phi$ if $\phi \in M(\pi)$. Then by definition 95 we know that $\phi \in t(M)(\pi^{\mathcal{TR}})$ and thus $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$

**Serial Conjunction** $\phi = \phi_1 \otimes \phi_2$

$M, \pi \models \phi_1 \otimes \phi_2$ iff there is a prefix $\pi'$ s.t. $\pi \in \mathtt{exp}_M(\pi')$ and $M, \pi_1 \models \phi_1$ and $M, \pi_2 \models \phi_2$ for a split $\pi_1 \circ \pi_2$ of $\pi'$. By lemma 25 we know that $\pi \in \mathtt{exp}_M(\pi')'$ and thus that $\pi' = \pi$

By Induction Hypothesis we know that $t(M), \pi_1^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ and $t(M), \pi_2^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_2$, which allows us to conclude: $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1 \otimes \phi_2$.

**Negation** $\phi = \neg\phi$

$M, \pi \models \neg\phi$ iff it is not the case that $M, \pi \models \phi$. Let's assume that $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$ then by $\Rightarrow$ proof we know that $M, \pi \models \phi$, which contradicts our assumption. Then, it is not the case that $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$ and thus $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \neg\phi$.

**Disjunction** $\phi = \phi_1 \vee \phi_2$

$M, \pi \models \phi_1 \vee \phi_2$ iff $M, \pi \models \phi_1$ or $M, \pi \models \phi_2$

By Induction Hypothesis we know that $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ or $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_2$, and thus by definition of $\models_{\mathcal{TR}}$: $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1 \vee \phi_2$.

**Executional Possibility** $M, \langle D \rangle \models \Diamond\phi$ iff $\pi$ is a 1-path of the form $\langle D \rangle$ for some state $D$ and $M, \pi' \models \phi$ for some path $\pi'$ that begins at $D$.

We can apply the Induction Hypothesis for $\phi$ and conclude that $t(M), \pi'^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$ for a path $\pi'^{\mathcal{TR}}$ that begins at $D$. From this we know that $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \Diamond\phi$.

□

**Lemma 27.** *Let $M$ be a minimal model of a program $P$ which does not have complex event rules, and $P'$ the program obtained by definition 94. Then for every path $\pi^{\mathcal{TR}}$ in $\mathcal{TR}$, and every formula $\phi \in \mathcal{O}_{reactive}$:*

$$t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_\phi \textit{ iff } M, \pi \models \phi$$

*where $\pi$ is the path obtained from $\pi^{\mathcal{TR}}$ by only adding annotated transitions.*

*Proof.*
(⇒):
By definition 95 we know $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_\phi$ implies: $\phi \in M(\pi_1)$ and $\mathbf{r}(\phi) \in M(\pi_2)$. Since $\mathbf{r}(\phi) \in \mathcal{P}_t$, then by lemma 24 we know that $M, \pi_2 \models \mathbf{r}(\phi)$ and thus $\pi_2 \in \mathtt{exp}_M(\pi_2)$.

Additionally, since $\mathbf{r}(\phi)$ is the response of $\phi$, since there are no complex events, since $\phi \in M(\pi_1)$, $\pi = \pi_1 \circ \pi_2$ and $M, \pi_2 \models \mathbf{r}(\phi)$, then we know that $\pi_1 \circ \pi_2 \in \mathtt{exp}_M(\pi_1)$. and thus $M, \pi \models \phi$
(⇐):
$M, \pi \models \phi$ if there is a prefix s.t. $\phi \in M(\pi_1)$ and $\pi \in \mathtt{exp}_M(\pi_1)$. Since there are no complex events in $P$, $\pi_1 \notin \mathtt{exp}_M(\pi_1)$ iff $\phi \in \mathcal{O}_{reactive}$ and $\mathbf{r}(\phi) \notin M(\pi_1)$.

Let's assume $\mathbf{r}(\phi) \notin M(\pi_1)$ (otherwise the proof is trivial), then there must exist a $\pi_2$ s.t. $M, \pi_2 \models \mathbf{r}(\phi)$, $\mathbf{r}(\phi) \in M(\pi_2)$ and $\pi_1 \circ \pi_2 = \pi$. Since $\mathbf{r}(\phi) \in M(\pi_2)$ and $\phi \in M(\pi_1)$, then by definition 95 we know $p_\phi \in t(M)(\pi^{\mathcal{TR}})$ and $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_\phi$ □

**Lemma 28.** *Let $M$ be a minimal model of a program $P$ which does not have complex event rules, and $P'$ the program obtained by definition 94. Then for every path $\pi^{\mathcal{TR}}$ in $\mathcal{TR}$, and every formula $\phi \in \mathcal{P}_e$:*

$$t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_\phi \textit{ iff } M, \pi \models \phi$$

*where $\pi$ is the path obtained from $\pi^{\mathcal{TR}}$ by only adding annotated transitions.*

*Proof.* (⇒):
By definition 95 we know $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_\phi$ implies $e \in M(\pi_1 \circ \pi_2) \wedge \mathbf{o}(e) \in M(\pi_1) \wedge \mathbf{r}(e) \in M(\pi_2) \wedge \pi = \pi_1 \circ \pi_2$.

Since no complex events are possible, $e \in M(\pi)$ then $\pi \in \mathtt{exp}_M(\pi)$ and $M, \pi \models e$.

(⇐):
$M, \pi \models e$ implies that there is a prefix $\pi_1$ of $\pi$ and $\mathbf{o}(e) \in M(\pi_1)$. Since no complex events are possible, $\pi \in \mathtt{exp}_M(\pi_1)$ iff $\pi = \pi_1 \circ \pi_2$ and $M, \pi_2 \models \mathbf{r}(e)$. If that is the case, then $p_\phi \in t(M)(\pi^{\mathcal{TR}})$ and $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_\phi$. □

**Lemma 29.** *Let $M$ be an interpretation, and $P$ be a program without complex event rules. If $M$ is a minimal model of $P$ in $\mathcal{TR}^{ev}$ then $t(M)$ is also a model of $P'$ in $\mathcal{TR}$, where $P'$ is obtained by applying the program transformation defined in definition 94*

*Proof.* In order to prove that $t(M)$ is a model of $P'$ it is sufficient to show for every rule of $P'$ that, whenever it satisfies the body of the rule in $P'$ it also satisfies the head.

I.e., $\forall h \leftarrow \phi \in P'$, we need to prove that if $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$ then $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$. Assume $h \leftarrow \phi \in P$ and $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$ holds, then we need to prove that $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$ holds. We do this by induction in the structure of $\phi$:

- **Base Case 1**: $\phi \in \mathcal{P}_{\mathcal{O}} \cup \mathcal{P}_t$
  Then $\phi \in t(M)(\pi^{\mathcal{TR}})$ and by definition 95 $\phi \in M(\pi)$ and one of the two cases must be true:

  - $h \leftarrow \phi \in P$ and $h \leftarrow \phi \in P'$

    If this is the case, then $\phi \in \mathcal{O}_{static}$, $\pi \in \texttt{exp}_M(\pi)$ and since $M$ is a model of $P$, $h \in M(\pi)$. Then by definition 95 $h \in t(M)(\pi^{\mathcal{TR}})$ and $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$.
  - $h \leftarrow \phi \in P'$ but $h \leftarrow \phi \notin P$. If $body \in \mathcal{P}_{\mathcal{O}} \cup \mathcal{P}_t$ then the only possibility is $\phi = p_\psi$ and $h \leftarrow \psi \in P$ By lemma 27 we know that if $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_e$ then $M, \pi \models \psi$ and thus $M, \pi \models h$. As a consequence $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$.

- **Event Case:** $\phi = p_e$
  Then $h \leftarrow p_e \in P'$ but $h \leftarrow e \in P$. By lemma 28 we know that if $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_e$ then $M, \pi \models e$ and thus $M, \pi \models h$. As a consequence $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$

- **Serial Conjunction Case:** $\phi = \phi_1 \otimes \phi_2$
  $t(M), \pi \models_{\mathcal{TR}} \phi_1 \otimes \phi_2$ iff it exists a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $t(M), \pi_1 \models_{\mathcal{TR}} \phi_1$ and $t(M), \pi_2 \models_{\mathcal{TR}} \phi_2$.

  Again there are two possibilities:

  - $h \leftarrow \phi \in P$ and $h \leftarrow \phi \in P'$

    If this is the case, then by lemma 26 we know that $M, \pi_1 \models \phi_1$ and $M, \pi_2 \models \phi_2$. Since no complex events may be true $\pi \in \texttt{exp}_M(\pi)$ and $M, \pi \models \phi_1 \otimes \phi_2$, $M, \pi \models h$ and $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$.
  - $h \leftarrow \phi \in P'$ but $h \leftarrow \phi \notin P$. Then $\phi_1 \vee \phi_2 \in \mathcal{P}_e \cup \mathcal{O}_{reactive}$. By lemma 26,lemma 27 and lemma 28 we know that $M, \pi_1 \models \phi_1$ and $M, \pi_2 \models \phi_2$. Since no complex events may be true, $\pi \in \texttt{exp}_M(\pi)$ and $M, \pi \models \phi_1 \otimes \phi_2$, $M, \pi \models h$ and $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$.

- **Negation:** $\phi = \neg\phi$
  $M, \pi \models \neg\phi$

  Again there are two possibilities:

  - $h \leftarrow \neg\phi \in P$ and $h \leftarrow \neg\phi \in P'$

    If this is the case, then by lemma 26 we know that $M, \pi \models \neg\phi$ Since $M$ is a model, then $M, \pi \models h$ and $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$.

– $h \leftarrow \neg\phi \in P'$ but $h \leftarrow \neg\phi \notin P$. Then $\phi \in \mathcal{P}_e \cup \mathcal{O}_{reactive}$ By lemma 26,lemma 27 and lemma 28 we know that $M, \pi \models \neg\phi_1$. Since $M$ is a model, then $M, \pi \models h$ and $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$.

- **Disjunction:** $\phi = \phi_1 \vee \phi_2$
  $M, \pi \models \phi_1 \vee \phi_2$

  Again there are two possibilities:

  – $h \leftarrow \phi_1 \vee \phi_2 \in P$ and $h \leftarrow \phi_1 \vee \phi_2 \in P'$

  If this is the case, then by lemma 26 we know that $M, \pi \models \phi_1 \vee \phi_2$. Since $M$ is a model, then $M, \pi \models h$ and $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$.

  – $h \leftarrow \phi_1 \vee \phi_2 \in P'$ but $h \leftarrow \phi_1 \vee \phi_2 \notin P$. Then $\phi_1 \vee \phi_2 \in \mathcal{P}_e \cup \mathcal{O}_{reactive}$. By lemma 26,lemma 27 and lemma 28 we know that $M, \pi \models \phi_1 \vee \phi_2$. Since $M$ is a model, then $M, \pi \models h$ and $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$.

- **Executional Possibility:** $\phi = \Diamond\phi_1$
  $M, \pi \models \Diamond\phi_1$

  Again there are two possibilities:

  – $h \leftarrow \Diamond\phi_1 \in P$ and $h \leftarrow \Diamond\phi_1 \in P'$

  If this is the case, then by lemma 26 we know that $M, \pi \models \Diamond\phi_1$ Since $M$ is a model, then $M, \pi \models h$ and $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$.

  – $h \leftarrow \Diamond\phi_1 \in P'$ but $h \leftarrow \Diamond\phi_1 \notin P$. Then $\phi_1 \vee \phi_2 \in \mathcal{P}_| \cup \mathcal{O}_{reactive}$ By lemma 26,lemma 27 and lemma 28 we know that $M, \pi \models \Diamond\phi_1$. Since $M$ is a model, then $M, \pi \models h$ and $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$.

□

**Lemma 30** (Minimal Models in $\mathcal{TR}$)**.** *For all models $M$ of $P$ and for all the minimal models $M_M$ of $P$ the following property is true:*

$$M \models_{\mathcal{TR}} \phi \textit{ iff } M_M \models_{\mathcal{TR}} \phi$$

*Proof.* We prove the claim in both directions.
⇒:
If $M \models_{\mathcal{TR}} \phi$ then $M_M \models_{\mathcal{TR}} \phi$
Trivially true. If $M \models_{\mathcal{TR}} \phi$ then $\phi$ is true in *all* models of $P$ including in the minimal ones: $M_M$.
⇐:
If $M_M \models_{\mathcal{TR}} \phi$ then $M \models_{\mathcal{TR}} \phi$
Let us assume that the latter is not true, and so $\phi$ is true in $M_M$ but not in a given model $M_F$ of the program such that $M_F$ is not a minimal model. Then there must exist a path $\pi$ such that $\phi \in M_M(\pi)$ but $\phi \notin M_F(\pi)$. However by definition of minimal models

$M_M$ is a minimal model if for every path $\pi$ and every model $M$ of $P$, $M_M(\pi) \subseteq M(\pi)$. Consequently if $\phi \in M_M(\pi)$ but $\phi \notin M_F(\pi)$ then either $M_F$ is a minimal model (and $M_M$ and $M_F$ are incomparable) or $M_M$ is not a minimal model. Since both hypothesis lead to a contradiction, then we prove that $M_M \models_{\mathcal{TR}} \phi$ then $M \models_{\mathcal{TR}} \phi$. □

**Definition 97.** *Let $M$ be an interpretation in $\mathcal{TR}$. We denote $M^{-1}$ as the interpretation in $\mathcal{TR}^{ev}$ defined w.r.t $M$ as follows:*

1. $\forall \phi \in \mathcal{P}_{\mathcal{O}} \cup \mathcal{P}_t$ *of* $\mathcal{TR}^{ev}$
   *If* $\phi \in M(\pi^{\mathcal{TR}})$ *then* $\phi \in M^{-1}(\pi)$
   *If* $p_\phi \in M(\pi^{\mathcal{TR}})$ *then* $\mathbf{r}(\phi) \in M^{-1}(\pi)$
2. $\forall e \in \mathcal{P}_e$
   *If* $p_e \in M(\pi^{\mathcal{TR}})$ *then* $\mathbf{r}(e) \in M^{-1}(\pi)$
3. *If* $\pi_1 = \langle D_i \xrightarrow{\mathbf{o}(e)} D_i \rangle$ *and* $e \in \mathcal{P}_e$
   (a) *If* $\exists \pi_2$ *s.t.* $p_e \in M(\pi_2^{\mathcal{TR}})$ *and* $\pi = \pi_1 \circ \pi_2$
   *then* $\mathbf{o}(e) \in M^{-1}(\pi_1)$
   (b) *otherwise* $\mathbf{o}(e) \in M^{-1}(\pi_1)$ *and* $\mathbf{r}(e) \in M^{-1}(\pi_1)$
4. *If* $\pi_1 = \langle D_i \xrightarrow{\mathbf{o}(\phi)} D_{i+1} \rangle$ *and* $\phi \in \mathcal{P}_{\mathcal{O}}$
   (a) *If* $\exists \pi_2$ *s.t.* $\mathbf{r}(\phi) \in M(\pi_2^{\mathcal{TR}}) \wedge \pi = \pi_1 \circ \pi_2$
   *then* $\mathbf{o}(\phi) \in M^{-1}(\pi_1)$
   (b) *otherwise* $\mathbf{o}(\phi) \in M^{-1}(\pi_1)$ *and* $\mathbf{r}(\phi) \in M^{-1}(\pi_1)$
5. *Nothing else belongs to* $M^{-1}$.

**Lemma 31.** *Let $M_1$ and $M_2$ be interpretations in $\mathcal{TR}$*

$$M_1 \subset M_2 \Rightarrow M_1^{-1} \subset M_2^{-1}$$
$$M_1 = M_2 \Rightarrow M_1^{-1} = M_2^{-1}$$

*Proof.* Immediate by definition 97 □

**Lemma 32.** *Let $M$ be an interpretation in $\mathcal{TR}^{ev}$. Let $t(M)$ be the interpretation in $\mathcal{TR}$ w.r.t. $M$ achieved by applying definition 95. Let $t(M)^{-1}$ be the interpretation in $\mathcal{TR}^{ev}$ w.r.t. $t(M)$ achieved by applying definition 97*

$$(t(M))^{-1} = M$$

*Proof.* Immediately by definition 95 and definition 97. □

**Lemma 33.** *Let $M$ be an interpretation in $\mathcal{TR}$ and $M^{-1}$ be the interpretation w.r.t. $M$ achieved by applying definition 97. Let $t(M^{-1})$ be the interpretation in $\mathcal{TR}$ w.r.t. $M$ achieved by applying definition 95.*

$$t(M^{-1}) = M$$

*Proof.* Immediately by definition 95 and definition 97 □

**Lemma 34.** *Let $M$ be an interpretation in $\mathcal{TR}$, then there is an interpretation $M_1$ in $\mathcal{TR}^{ev}$ s.t. $t(M_1) = M$*

*Proof.* We prove by contradiction. Assume there is a $\mathcal{TR}$ interpretation $M'$ s.t. $\forall M_1$: $t(M_1) \neq M$.

However, if $M'$ is an interpretation in $\mathcal{TR}$, then there is an interpretation in $M'^{-1}$ which is an interpretation in $\mathcal{TR}^{ev}$. If that is the case, then $t(M'^{-1})$ is an interpretation in $\mathcal{TR}$ and by lemma 33 we know that $t(M'^{-1}) = M'$. Since there is an interpretation s.t. $t(M'^{-1}) = M'$, we achieve a contradiction, and our claim holds. □

**Lemma 35.** *Let $M$ be an interpretation in $\mathcal{TR}^{ev}$, then there is an interpretation $M_1$ in $\mathcal{TR}$ s.t. $M_1^{-1} = M$.*

*Proof.* We prove by contradiction. Assume there is a $\mathcal{TR}^{ev}$ interpretation $M'$ s.t. $\forall M_1$: $M_1^{-1} \neq M$.

However, if $M'$ is an interpretation in $\mathcal{TR}^{ev}$, then there is an interpretation in $t(M')$ which is an interpretation in $\mathcal{TR}$. If that is the case, then $t(M')^{-1}$ is an interpretation in $\mathcal{TR}^{ev}$ and by lemma 32 we know that $t(M')^{-1} = M'$. Since there is an interpretation $M_1$ s.t. $M_1^{-1} = M'$, we achieve a contradiction, and our claim holds. □

**Lemma 36.** *Let $P$ be a $\mathcal{TR}$ program, $M$ be an interpretation in $\mathcal{TR}$ and $\phi$ a formula which does not contain $\mathcal{O}_{reactive}$ and $\mathcal{P}_e$. Then:*

$$M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi \textit{ iff } M^{-1} \models \phi$$

*Proof.* We prove this claim in each directions.

($\Rightarrow$):

We prove by induction on the structure of $\phi$.

**Base Case** $\phi \in \mathcal{O}_{static} \cup \mathcal{P}_t$

$M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$ iff $\phi \in M(\pi)$. Then by definition 97 we know that $\phi \in M^{-1}(\pi)$. Since $\phi \in \mathcal{O}_{static}$, $\pi \in \mathtt{exp}_{M^{-1}}(\pi)$ and $M^{-1}, \pi \models \phi$

**Serial Conjunction** $\phi = \phi_1 \otimes \phi_2$

$M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1 \otimes \phi_2$ iff there is a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ and $M, \pi_2^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_2$. Then by Induction Hypothesis, we know that $M^{-1}, \pi_1 \models \phi_1$ and $M^{-1}, \pi_2 \models \phi_2$. Moreover since $M^{-1}$ only satisfies occurrences in paths of size 2, and $\pi_1 \in \mathtt{exp}_{M^{-1}}(\pi_1)$ and $\pi_2 \in \mathtt{exp}_{M^{-1}}(\pi_2)$ then $\pi \in \mathtt{exp}_{M^{-1}}(\pi)$ and $M^{-1}, \pi \models \phi_1 \otimes \phi_2$

**Disjunction** $\phi = \phi_1 \vee \phi_2$

$M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1 \otimes \phi_2$ iff $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ or $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_2$. Then by Induction Hypothesis, we know that $M^{-1}, \pi \models \phi_1$ or $M^{-1}, \pi \models \phi_2$. Thus by definition of $\models$ we know $M^{-1}, \pi \models \phi_1 \vee \phi_2$

**Negation** $\phi = \neg\phi_1$

$M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \neg\phi_1$ and thus we know that $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ is not the case.

Similarly, to prove $M^{-1}, \pi \models \neg\phi_1$ we need to show that it is not the case that $M^{-1}, \pi \models \phi_1$. We prove this by contradiction. Let's assume $M^{-1}, \pi \models \phi_1$ then by ($\Leftarrow$) we can conclude $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ which contradicts our initial knowledge.

Thus, since $M^{-1}, \pi \models \phi_1$ does not hold, we can conclude $M^{-1}, \pi \models \neg\phi_1$

**Executional Possibility** $\phi = \Diamond\phi_1$

$M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \Diamond\phi_1$ iff $\pi$ is a 1-path of the form $\langle D \rangle$ for some state $D$ and $M, \pi'^{\mathcal{TR}} \models \phi_1$ for some path $\pi'^{\mathcal{TR}}$ that begins at $D$.

Since we know $M, \pi'^{\mathcal{TR}} \models \phi_1$, by Induction Hypothesis we know $M^{-1}, \pi' \models \phi_1$. From this, and from the definition of $\models$ we conclude $M^{-1}, \pi \models \Diamond\phi_1$

($\Leftarrow$):

We prove by induction on the structure of $\phi$.

**Base Case** $\phi \in \mathcal{O}_{static} \cup \mathcal{P}_t$

$M^{-1}, \pi \models \phi$ then, since $\phi \in \mathcal{O}_{static} \cup \mathcal{P}_t$ we know that $\pi \in \texttt{exp}_{M^{-1}}(\pi)$. From definition 97 we know that $\phi \in M^{-1}(\pi)$ if $\phi \in M(\pi^{\mathcal{TR}})$ and thus $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$

**Serial Conjunction** $\phi = \phi_1 \otimes \phi_2$

$M^{-1}, \pi \models \phi_1 \otimes \phi_2$ then there is a split $\pi_1 \circ \pi_2$ of a path $\pi'$, prefix of $\pi$ s.t. $M^{-1}, \pi_1 \models \phi_1$, $M^{-1}, \pi_2 \models \phi_2$ and $\pi \in \texttt{exp}_{M^{-1}}(\pi')$. Moreover, since $\pi_1 \in \texttt{exp}_{M^{-1}}(\pi_1)$ and $\pi_2 \in \texttt{exp}_{M^{-1}}(\pi_2)$ and since $M^{-1}$ only satisfies occurrences in paths of size 2, we know that $\pi \in \texttt{exp}_{M^{-1}}(\pi')'$, and thus $\pi_1 \circ \pi_2$ can be seen as splits of $\pi$.

We can apply the Induction Hypothesis to $\phi_1$ and $\phi_2$ and conclude $M, \pi_1^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ and $M, \pi_2^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_2$ which leads us to $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1 \otimes \phi_2$.

**Disjunction** $\phi = \phi_1 \vee \phi_2$

$M^{-1}, \pi \models \phi_1 \vee \phi_2$ iff $M^{-1}, \pi \models \phi_1$ or $M^{-1}, \pi \models \phi_2$.

We can apply the Induction Hypothesis to $\phi_1$ and $\phi_2$ and conclude $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ and $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_2$. From this we know $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1 \vee \phi_2$.

**Negation** $\phi = \neg\phi_1$

$M^{-1}, \pi \models \neg\phi_1$ iff it is not the case that $M^{-1}, \pi \models \phi_1$.

Similarly, to prove $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \neg\phi_1$ we need to show that $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ is not the case. We prove this by contradiction. Let's assume $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ then by ($\Rightarrow$) we can conclude $M^{-1}, \pi \models \neg\phi_1$ which contradicts our initial knowledge.

Thus, since $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$ does not hold, we can conclude $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \neg\phi_1$

**Executional Possibility** $\phi = \Diamond\phi_1$

$M^{-1}, \pi \models \Diamond\phi_1$ iff $\pi$ is a 1-path of the form $\langle D \rangle$ for some state $D$ and $M^{-1}, \pi' \models \phi_1$ for some path $\pi'$ that begins at $D$.

Since $M^{-1}, \pi' \models \phi_1$ holds, by Induction Hypothesis we know $M, \pi'^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1$, and can conclude $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \Diamond\phi$

□

**Lemma 37.** *Let $P$ be a $\mathcal{TR}$ program, and $M$ be an interpretation in $\mathcal{TR}$*

$$M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_\phi \textit{ iff } M^{-1} \models \phi$$

*where $\phi \in \mathcal{O}_{reactive} \cup \mathcal{P}_e$.*

*Proof.* Let's assume $\phi = \mathcal{P}_e$, and $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_\phi$. Then we know by definition 97 that $\mathbf{r}(\phi) \in M^{-1}(\pi^{\mathcal{TR}})$. If $\pi_1 = \langle D_1 \xrightarrow{o(\phi)} D_1 \rangle$ and $\pi_1 \circ \pi^{\mathcal{TR}}$, $\mathbf{o}(\phi) \in M(\pi_1)$ and by definition 93 we know $M^{-1}, \pi \models \phi$

Now assume $\phi \in \mathcal{O}_{reactive}$ and $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_\phi$. Then we know by definition 97 that $\mathbf{r}(\phi) \in M^{-1}(\pi^{\mathcal{TR}})$. If $\pi_1 = \langle D_1 \xrightarrow{o(\phi)} D_2 \rangle$ and $\pi_1 \circ \pi^{\mathcal{TR}}$, we know $\pi_1 \in \mathtt{exp}_{M^{-1}}(\pi_1) \circ \pi$ and thus $M^{-1}\pi_1 \circ \pi \models \phi$ □

**Lemma 38.** *Let $P$ be a $\mathcal{TR}^{ev}$ program, and $P'$ be the program obtained by definition 94 Let $M$ be a model of $P'$ in $\mathcal{TR}$ then $M^{-1}$ is also a model of $P$.*

*Proof.* To prove that $M$ is a model of $P$ it is sufficient to show for every rule of $P'$ that, whenever it satisfies the body of the rule in $P'$ it also satisfies the head.

I.e., $\forall h \leftarrow \phi \in P$, we need to prove that if $M^{-1}, \pi \models \phi$ then $M^{-1}, \pi \models h$.
Assume $h \leftarrow \phi \in P$ and $M^{-1}, \pi \models \phi$ holds, then we need to prove that $M^{-1}, \pi \models h$ holds as well. We do this by induction in the structure of $\phi$:

**Base Case:** $\phi \in \mathcal{P}_\mathcal{O} \cup \mathcal{P}_t$

Then $\phi \in M^{-1}(\pi)$ and $\pi \in \mathtt{exp}_{M^{-1}}(\pi)$ and by definition 97 we know $\phi \in M(\pi^{\mathcal{TR}})$, and $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$.

From this, one of the two cases must be true:

- $h \leftarrow \phi \in P$ and $h \leftarrow \phi \in P'$. Since $M$ is a model of $P'$ and $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi$, we know that $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$. By definition of what is a rule, $h$ is an atom in $\mathcal{P}_t$, $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$ implies $h \in M(\pi^{\mathcal{TR}})$. which implies $h \in M^{-1}(\pi)$.
  Since we know $\pi \in \mathtt{exp}_{M^{-1}}(\pi)$, then $M, \pi \models h$.
- $h \leftarrow \phi \in P$ but $h \leftarrow \phi \notin P'$. If $body \in \mathcal{P}_\mathcal{O} \cup \mathcal{P}_t$ then it must exist a rule $h \leftarrow p_\phi \in P'$. By lemma 37 we know that if $M^{-1}, \pi \models \phi$ then $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_\phi$
  Since $M$ is a model of $P'$ then, $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$. Since by definition of what is a rule, $h$ is an atom in $\mathcal{P}_t$, $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$ implies $h \in M(\pi^{\mathcal{TR}})$, which implies $h \in M^{-1}(\pi)$. Since we know $\pi \in \mathtt{exp}_{M^{-1}}(\pi)$, then $M, \pi \models h$.

**Event Case:** $\phi \in \mathcal{P}_e$

Then $\phi \in M^{-1}(\pi)$ and $\pi \in \mathtt{exp}_{M^{-1}}(\pi)$ and by definition 97 we know $p_\phi \in M(\pi^{\mathcal{TR}})$, and $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_\phi$.

From this, we know that: $h \leftarrow \phi \in P$ and $h \leftarrow p_\phi \in P'$. By lemma 37 we know that if $M^{-1}, \pi \models \phi$ then $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} p_\phi$.

Since $M$ is a model of $P'$ then, $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$. Since by definition of what is a rule, $h$ is an atom in $\mathcal{P}_t$, $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$ implies $h \in M(\pi^{\mathcal{TR}})$. which implies $h \in M^{-1}(\pi)$. Since we know $\pi \in \texttt{exp}_{M^{-1}}(\pi)$, then $M, \pi \models h$

**Serial Conjunction:** $\phi = \phi_1 \otimes \phi_2$

$M^{-1}, \pi \models \phi_1 \otimes \phi_2$ . In this case there is a split $\pi_1 \circ \pi_2$ of some prefix $\pi'$ of $\pi$ s.t. $M^{-1}, \pi_1 \models \phi_1$, $M^{-1}, \pi_2 \models \phi_2$, and $\pi \in \texttt{exp}_{M^{-1}}(\pi')$. Since $\pi_1 \in \texttt{exp}_{M^{-1}}(\pi_1)$, $\pi_2 \in \texttt{exp}_{M^{-1}}(\pi_2)$, and $M^{-1}$ does not satisfy event occurrences over paths of length greater than 2, we know that $\pi \in \texttt{exp}_{M^{-1}}(\pi')'$, i.e. $\pi' = \pi$.

From this, one of the two must be true:

- $h \leftarrow \phi \in P$ and $h \leftarrow \phi \in P'$. By lemma 36 we know that $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1 \otimes \phi_2$. Since $M$ is a model of $P'$ we know $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$ and since $h$ is an atom we know $M^{-1}, \pi \models h$
- $h \leftarrow \phi \in P$ but $h \leftarrow \phi \notin P'$. Then $\phi_1 \otimes \phi_2 \in \mathcal{P}_e \cup \mathcal{O}_{reactive}$, and an equivalent rule exists in $P'$ for $h$. By lemma 36 and lemma 37 we know $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1 \otimes \phi_2$ (or their equivalent in $P'$). $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$ and $M^{-1}, \pi \models h$.

**Negation:** $\phi = \neg\phi_1$

$M^{-1}, \pi \models \neg\phi_1$. From this one of the two must be true:

- $h \leftarrow \phi \in P$ and $h \leftarrow \phi \in P'$. And by lemma 36, $M^{-1}, \pi \models \neg\phi_1$ implies $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \neg\phi_1$ and thus $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$. Then by definition 97 we know $M^{-1}, \pi \models h$.
- $h \leftarrow \phi \in P$ and $h \leftarrow \phi \notin P'$. Then $\phi \in \mathcal{P}_\neg \cup \mathcal{O}_{reactive}$, and an equivalent rule exists in $P'$ for $h$. By lemma 36 and lemma 37 we know $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \neg\phi$ (or its equivalent in $P'$), $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$ and $M^{-1}, \pi \models h$.

**Disjunction:** $\phi = \phi_1 \vee \phi_2$ $M^{-1}, \pi \models \phi_1 \vee \phi_2$

From this one of the two must be true:

- $h \leftarrow \phi \in P$ and $h \leftarrow \phi \in P'$ And by lemma 36, $M^{-1}, \pi \models \phi_1 \vee \phi_2$ implies $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1 \vee \phi_2$ and thus $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$. Then by definition 97 we know $M^{-1}, \pi \models h$
- $h \leftarrow \phi \in P$ and $h \leftarrow \phi \notin P'$ Then $\phi_1 \vee \phi_2 \in \mathcal{P}_\neg \cup \mathcal{O}_{reactive}$, and an equivalent rule exists in $P'$ for $h$. By lemma 36 and lemma 37 we know $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi_1 \vee \phi_2$ (or their equivalent in $P'$). Then, $M, \pi^{\mathcal{TR}} \models_{\mathcal{TR}} h$ and $M^{-1}, \pi \models h$.

□

**Lemma 39.** *Let $P$ be a $\mathcal{TR}^{ev}$ program. If $M$ is a minimal model of $P$, then $t(M)$ is a minimal model of $P'$.*

*Proof.* We already know that $t(M)$ is a model of $P'$ by lemma 29, so it remains to show that $t(M)$ is also minimal. We show the latter by contradiction.

Assume that $t(M)$ is not minimal. Then there is a $M_1$ s.t. $M_1 \subset t(M)$ and $M_1$ is a minimal model of $P'$. If that is the case, then $M_1^{-1}$ which by lemma 31 $M_1^{-1} \subset t(M)^{-1}$. Since by lemma 32 $t(M)^{-1} = M$ then, $M_1^{-1} \subset M$. Since $M$ is a minimal model of $P$, then we reach a contradiction. □

**Lemma 40.** *Let $P$ be a $\mathcal{TR}^{ev}$ program, and $P'$ be the program obtained by definition 94 Let $M$ be a minimal model of $P'$ in $\mathcal{TR}$ then $M^{-1}$ is also a minimal model of $P$.*

*Proof.* By lemma 38 we know that if $M$ is a model of $P'$, then $M^{-1}$ is a model of $p$.

So, it remains to show that $M^{-1}$ is minimal if $M$ is also minimal. We show this by contradiction. Assume that it exists a $M_{min}$ such that $M_{min} \subset M^{-1}$ (they are comparable) which is a model of $P$.

If that is the case, then by lemma 22 it follows that $t(M_{min}) \subset t(M^{-1})$ (by lemma 22) Moreover, by lemma 33 $t(M^{-1}) = M$ and thus $t(M_{min}) \subseteq M$. Since $M$ is a minimal model, we reach a contradiction.

□

**Theorem 10.***[Comparison to $\mathcal{TR}$] Let $P$ be a complex-event free program, and let $P'$ be obtained from $P$ by replacing in $P$ every event $e$ and every response $\mathbf{r}(e)$, s.t. $e \in \mathcal{P}_e$, by a new fluent $p_e$. Let $\pi$ be an annotated path and $\pi'$ be a path obtained from $\pi$ removing the annotations and for every event occurrence $\varphi$ in $\pi$ s.t.$\pi = \pi_1 \circ \langle D^{\varphi} \rightarrow D \rangle \circ \pi_2$, then $\pi' = \pi_1 \circ \langle D \rangle \circ \pi_2$. Then for every transaction formula $\phi$:*

$$P, \pi \models \phi \text{ iff } P', \pi' \models_{\mathcal{TR}} \phi'$$

*Proof.* Next we prove theorem 10 stated in page 135. In order to better understand this proof, we recommend the reader to by the auxiliary lemmas, starting in page 244.

We prove the claim in each direction.

(⇒):

$P, \pi \models \phi$ iff $M, \pi \models \phi$ for all minimal models $M$ of $P$. By lemma 39 we know that for all minimal models $M$ there is a $t(M)$ which is also a minimal model of $P'$. Moreover, by lemma 34 we know that for all minimal models $M'$ of $\mathcal{TR}$ there is a minimal model $M$ s.t. $t(M) = M'$.

Thus by lemma 26, lemma 27 and lemma 28 that $M, \pi \models \phi$ implies $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi'$ for all minimal models of $\mathcal{TR}$. Finally by lemma 30 we know that if $t(M), \pi^{\mathcal{TR}} \models_{\mathcal{TR}} \phi'$ holds for all minimal models of $\mathcal{TR}$, then it holds for all $\mathcal{TR}$ models, and thus $P', \pi' \models_{\mathcal{TR}} \phi'$.

(⇐):

$P', \pi' \models_{\mathcal{TR}} \phi'$ iff for all models of $M$ of $P'$: $M, \pi' \models_{\mathcal{TR}} \phi'$. Moreover by lemma 30 we know that this is equivalent to consider only the minimal models $M$ of $P'$.

By lemma 40 and lemma 35 we know that $M^{-1}, \pi \models \phi$ for all minimal models of $\mathcal{TR}^{ev}$. Then by definition we know that $P, \pi \models \phi$ holds. □

## C.2 Binarization Equivalence in $\mathcal{TR}^{ev}$'s Procedure

**Proposition 2.***[Binarization equivalence] Let $P$ be a $\mathcal{TR}^{ev}$ program with a rule of the form: $\mathbf{o}(e_1)$ $\texttt{OP}$ $\mathbf{o}(e_2)$ $\texttt{OP}$ $\mathbf{o}(e_3) \Rightarrow \mathbf{o}(e)$ for any events $e_1$-$e_3$ and operator $\texttt{OP}$, and let $P'$ be obtained from $P$ by removing that rule and adding $\mathbf{o}(e_1)$ $\texttt{OP}$ $\mathbf{o}(e_2) \Rightarrow \mathbf{o}(ie_1)$, $\mathbf{o}(ie_1)$ $\texttt{OP}$ $\mathbf{o}(e_3) \Rightarrow \mathbf{o}(e)$ and $\mathbf{r}(ie_1) \leftarrow true$. Then:*

$$P \equiv P'$$

*Proof.* Next we prove the proposition 2 from page 137.

To prove equivalence between $P$ and $P'$ we have to show for every path $\pi$ and every formula $\phi$ that $P, \pi \models \phi$ iff $P', \pi \models \phi$

Note that the two programs $P$ and $P'$ differ only on one event rule for the event $e$, and $P'$ has a rule for $ie_1$ occurrence and response, a new auxiliary event that does not belong to the program language. Consequently, since there is no additional rule for transactions (except the fact $\mathbf{r}(ie_1) \leftarrow true$, for a formula that did not appear in the program before), it is sufficient to show for every path $\pi$, every minimal model $M$ of $P$ and every minimal model $M'$ of $P'$ that:

1. $M', \pi \models_{ev} \mathbf{o}(ie_1)$ then $M', \pi \models_{ev} \mathbf{r}(ie_1)$
2. $M, \pi \models_{ev} \mathbf{o}(\varphi)$ iff $M', \pi \models_{ev} \mathbf{o}(\varphi)$ for every $\varphi$ different from $ie_1$

Point 1. comes directly from the definition of $P'$. Since $\mathbf{r}(ie_1)$ is in $P'$, then every minimal model $M'$ of $P$ will satisfy $\mathbf{r}(ie_1)$ in every path $\pi$, and thus point 1. is necessarily true.

Now for point 2. recall that $P$ and $P'$ define exactly the same rules for every event, except for $e$. As such, to prove point 2., it is sufficient to show that:

$$M, \pi \models_{ev} \mathbf{o}(e) \text{ iff } M', \pi \models_{ev} \mathbf{o}(e)$$

Next we prove this claim in each direction:

**$\Rightarrow$:** $M, \pi \models_{ev} \mathbf{o}(e)$ **then** $M', \pi \models_{ev} \mathbf{o}(e)$

Assume $M, \pi \models_{ev} e$, if this is the case then there are three possible explanations for this satisfaction:

a) $M, \pi \models_{ev} \mathbf{o}(e)$ because $e \in \mathcal{O}^t(\pi)$ or $\pi = \langle D \xrightarrow{\mathbf{o}(e)} D \rangle$, for any state $D$.
If this is the case then by definition of what is an interpretation, $M', \pi \models_{ev} \mathbf{o}(e)$

b) $M, \pi \models_{ev} \mathbf{o}(e)$, $body \Rightarrow \mathbf{o}(e) \in P$ and $M, \pi \models_{ev} body$
In this case, there are two further possibilities:

b1) $body \Rightarrow \mathbf{o}(e) \in P'$ and in this case, we know that $M, \pi \models_{ev} body$ implies $M', \pi \models_{ev} body$ and $M', \pi \models_{ev} \mathbf{o}(e)$

b2) *body* $\Rightarrow \mathbf{o}(e) \notin P'$ and the rule in question is $\mathbf{o}(e_1)$ `OP` $\mathbf{o}(e_2)$ `OP` $\mathbf{o}(e_3) \Rightarrow \mathbf{o}(e)$
As such let's assume $M, \pi \models_{ev} \mathbf{o}(e_1)$ `OP` $\mathbf{o}(e_2)$ `OP` $\mathbf{o}(e_3)$ and show that this implies that $M', \pi \models_{ev} \mathbf{o}(e)$ For that we will look into the structure of the possible connective that can appear in `OP`:

$\wedge$) $M, \pi \models_{ev} \mathbf{o}(e_1) \wedge \mathbf{o}(e_2) \wedge \mathbf{o}(e_3)$ then we know that:

$$M, \pi \models_{ev} \mathbf{o}(e_1) \text{ and}$$
$$M, \pi \models_{ev} \mathbf{o}(e_2) \text{ and}$$
$$M, \pi \models_{ev} \mathbf{o}(e_3)$$

Consequently, since $e_1$, $e_2$ and $e_3$ are defined exactly the same in $P$ and $P'$ it is easy to see that $M', \pi \models_{ev} \mathbf{o}(e_1)$, $M', \pi \models_{ev} \mathbf{o}(e_2)$ and $M', \pi \models_{ev} \mathbf{o}(e_3)$. Then by definition of $\wedge$ in $\models_{ev}$ we know that

$$M', \pi \models_{ev} \mathbf{o}(e_1) \wedge \mathbf{o}(e_2) \wedge \mathbf{o}(e_3)$$

$\vee$) $M, \pi \models_{ev} \mathbf{o}(e_1) \vee \mathbf{o}(e_2) \vee \mathbf{o}(e_3)$ then we know that:

$$M, \pi \models_{ev} \mathbf{o}(e_1) \text{ or}$$
$$M, \pi \models_{ev} \mathbf{o}(e_2) \text{ or}$$
$$M, \pi \models_{ev} \mathbf{o}(e_3)$$

Consequently, since $e_1$, $e_2$ and $e_3$ are defined exactly the same in $P$ and $P'$ it is easy to see that $M', \pi \models_{ev} \mathbf{o}(e_1)$ or $M', \pi \models_{ev} \mathbf{o}(e_2)$ or $M', \pi \models_{ev} \mathbf{o}(e_3)$. Then by definition of $\vee$ in $\models_{ev}$ we know that

$$M', \pi \models_{ev} \mathbf{o}(e_1) \vee \mathbf{o}(e_2) \vee \mathbf{o}(e_3)$$

$\otimes$) $M, \pi \models_{ev} \mathbf{o}(e_1) \otimes \mathbf{o}(e_2) \otimes \mathbf{o}(e_3)$ then we know that there is a split $\pi_1 \circ \pi_2 \circ \pi_3$ of $\pi$ s.t.

$$M, \pi_1 \models_{ev} \mathbf{o}(e_1) \text{ and}$$
$$M, \pi_2 \models_{ev} \mathbf{o}(e_2) \text{ and}$$
$$M, \pi_3 \models_{ev} \mathbf{o}(e_3)$$

Consequently, since $e_1$, $e_2$ and $e_3$ are defined exactly the same in $P$ and $P'$ it is easy to see that $M', \pi_1 \models_{ev} \mathbf{o}(e_1)$, $M', \pi_2 \models_{ev} \mathbf{o}(e_2)$ and $M', \pi_3 \models_{ev} \mathbf{o}(e_3)$. Then by definition of $\otimes$ in $\models_{ev}$ we know that

$$M', \pi \models_{ev} \mathbf{o}(e_1) \otimes \mathbf{o}(e_2) \otimes \mathbf{o}(e_3)$$

;) $M, \pi \models_{ev} \mathbf{o}(e_1); \mathbf{o}(e_2); \mathbf{o}(e_3)$ then we know that there is a split $\pi_1 \circ \pi_1' \circ \pi_2 \circ \pi_2' \circ \pi_3$ of $\pi$ s.t.

$$M, \pi_1 \models_{ev} \mathbf{o}(e_1) \text{ and}$$
$$M, \pi_2 \models_{ev} \mathbf{o}(e_2) \text{ and}$$
$$M, \pi_3 \models_{ev} \mathbf{o}(e_3)$$

Consequently, since $e_1$, $e_2$ and $e_3$ are defined exactly the same in $P$ and $P'$ it is easy to see that $M', \pi_1 \models_{ev} \mathbf{o}(e_1)$, $M', \pi_2 \models_{ev} \mathbf{o}(e_2)$ and $M', \pi_3 \models_{ev} \mathbf{o}(e_3)$. Then by definition of $\otimes$ in $\models_{ev}$ we know that $M', \pi \models_{ev} \mathbf{o}(e_1) \otimes \texttt{path} \otimes \mathbf{o}(e_2) \otimes \texttt{path} \otimes \mathbf{o}(e_3)$ and thus

$$M', \pi \models_{ev} \mathbf{o}(e_1); \mathbf{o}(e_2); \mathbf{o}(e_3)$$

Note that in this rule negation $\neg$ cannot appear in the program, except in the form of formula `path`, and that no other operator can appear as `OP`. Moreover, since $M', \pi \models_{ev} \mathbf{o}(e_1)$ `OP` $\mathbf{o}(e_2)$ `OP` $\mathbf{o}(e_3)$ it is easy to show that because $\mathbf{o}(ie_1)$ `OP` $\mathbf{o}(e_3) \Rightarrow \mathbf{o}(e)$ and $\mathbf{o}(e_1)$ `OP` $\mathbf{o}(e_2) \Rightarrow \mathbf{o}(ie_1)$ are in $P'$ then there is a $\pi'$, $\pi''$ s.t. $M', \pi \models_{ev} \mathbf{o}(ie_1)$ and $M', \pi'' \models_{ev} \mathbf{o}(e_3)$ and $M, \pi \models_{ev} \mathbf{o}(ie_1)$ `OP` $\mathbf{o}(e_3)$ (where for `OP` $= \wedge$ and `OP` $= \vee$, $\pi'$ and $\pi''$ are equal to $\pi$, and for `OP` $= \otimes$ and `OP` $=$; $\pi'$ and $\pi''$ are path splits of $\pi$). As a consequence of this, since $M'$ is a minimal model of $P'$ we know that:

$$M', \pi \models_{ev} \mathbf{o}(e)$$

And from all these cases we conclude that $M, \pi \models_{ev} \mathbf{o}(e)$ implies $M'\pi \models_{ev} \mathbf{o}(e)$

**$\Leftarrow$:** $M', \pi \models_{ev} \mathbf{o}(e)$ **then** $M, \pi \models_{ev} \mathbf{o}(e)$

Assume $M', \pi \models_{ev} e$, if this is the case then there are three possible explanations for this satisfaction:

a) $M', \pi \models_{ev} \mathbf{o}(e)$ because $e \in \mathcal{O}^t(\pi)$ or $\pi = \langle D \xrightarrow{\mathbf{o}(e)} D \rangle$, for any state $D$.
   If this is the case then by definition of what is an interpretation, $M, \pi \models_{ev} \mathbf{o}(e)$

b) $M', \pi \models_{ev} \mathbf{o}(e)$ *body* $\Rightarrow \mathbf{o}(e) \in P'$ and $M, \pi \models_{ev}$ *body*
   In this case, there are two further possibilities:

   b1) *body* $\Rightarrow \mathbf{o}(e) \in P'$ and in this case, we know that $M, \pi \models_{ev}$ *body* implies $M', \pi \models_{ev}$ *body* and $M', \pi \models_{ev} \mathbf{o}(e)$

   b2) *body* $\Rightarrow \mathbf{o}(e) \notin P'$ and the rule in question is $\mathbf{o}(ie_1)$ `OP` $\mathbf{o}(.)e_3 \Rightarrow \mathbf{o}(e)$
   As such let's assume $M, \pi \models_{ev} \mathbf{o}(ie_1)$ `OP` $\mathbf{o}(e_3)$ and show that this implies that $M', \pi \models_{ev} \mathbf{o}(e)$ For that we will look into the structure of the possible connective that can appear in `OP`:

   **$\wedge$)** $M', \pi \models_{ev} \mathbf{o}(ie_1) \wedge \mathbf{o}(e_3)$ then we know that:

   $$M', \pi \models_{ev} \mathbf{o}(ie_1) \text{ and}$$
   $$M', \pi \models_{ev} \mathbf{o}(e_3)$$

   Then, since $\mathbf{o}(ie_1)$ can only become true because of the rule $\mathbf{o}(e_1)$ `OP` $\mathbf{o}(e_2) \Rightarrow \mathbf{o}(ie_1)$ we know that:

   $$M', \pi \models_{ev} \mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \text{ and}$$
   $$M', \pi \models_{ev} \mathbf{o}(e_3)$$

Thus since $\mathbf{o}(e_1)$, $\mathbf{o}(e_2)$ $\mathbf{o}(e_3)$ become true in the same way in $P$ and $P'$ we can conclude for $M$ that

$$M, \pi \models_{ev} \mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \text{ and}$$
$$M, \pi \models_{ev} \mathbf{o}(e_3)$$

And thus $M, \pi \models_{ev} \mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \wedge \mathbf{o}(e_3)$ which because of the rule for $\mathbf{o}(e)$ in $P$ implies:

$$M, \pi \models_{ev} \mathbf{o}(e)$$

$\vee$) $M, \pi \models_{ev} \mathbf{o}(ie_1) \vee \mathbf{o}(e_3)$ then we know that:

$$M', \pi \models_{ev} \mathbf{o}(ie_1) \text{ or}$$
$$M', \pi \models_{ev} \mathbf{o}(e_3)$$

Then, since $\mathbf{o}(ie_1)$ can only become true because of the rule $\mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \Rightarrow \mathbf{o}(ie_1)$ we know that:

$$M', \pi \models_{ev} \mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \text{ or}$$
$$M', \pi \models_{ev} \mathbf{o}(e_3)$$

Thus since $\mathbf{o}(e_1)$, $\mathbf{o}(e_2)$ $\mathbf{o}(e_3)$ become true in the same way in $P$ and $P'$ we can conclude for $M$ that

$$M, \pi \models_{ev} \mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \text{ or}$$
$$M, \pi \models_{ev} \mathbf{o}(e_3)$$

And thus $M, \pi \models_{ev} \mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \vee \mathbf{o}(e_3)$ which because of the rule for $\mathbf{o}(e)$ in $P$ implies:

$$M, \pi \models_{ev} \mathbf{o}(e)$$

$\otimes$) $M, \pi \models_{ev} \mathbf{o}(ie_1) \otimes \mathbf{o}(e_3)$ then we know that there is a split $\pi_1 \circ \pi_2$ of $\pi$ s.t.

$$M', \pi_1 \models_{ev} \mathbf{o}(ie_1) \text{ and}$$
$$M', \pi_2 \models_{ev} \mathbf{o}(e_3)$$

Then, since $\mathbf{o}(ie_1)$ can only become true because of the rule $\mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \Rightarrow \mathbf{o}(ie_1)$ we know that:

$$M', \pi_1 \models_{ev} \mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \text{ or}$$
$$M', \pi_2 \models_{ev} \mathbf{o}(e_3)$$

Thus since $\mathbf{o}(e_1)$, $\mathbf{o}(e_2)$ $\mathbf{o}(e_3)$ become true in the same way in $P$ and $P'$ we can conclude for $M$ that

$$M, \pi_1 \models_{ev} \mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \text{ or}$$
$$M, \pi_2 \models_{ev} \mathbf{o}(e_3)$$

And thus $M, \pi \models_{ev} \mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \otimes \mathbf{o}(e_3)$ which because of the rule for $\mathbf{o}(e)$ in $P$ implies:

$$M, \pi \models_{ev} \mathbf{o}(e)$$

;) $M, \pi \models_{ev} \mathbf{o}(ie_1) \otimes \mathbf{o}(e_3)$ then we know that there is a split $\pi_1 \circ \pi_1' \circ \pi_2$ of $\pi$ s.t.

$$M', \pi_1 \models_{ev} \mathbf{o}(ie_1) \text{ and}$$
$$M', \pi_2 \models_{ev} \mathbf{o}(e_3)$$

Then, since $\mathbf{o}(ie_1)$ can only become true because of the rule $\mathbf{o}(e_1)$ `OP` $\mathbf{o}(e_2) \Rightarrow \mathbf{o}(ie_1)$ we know that:

$$M', \pi_1 \models_{ev} \mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \text{ or}$$
$$M', \pi_2 \models_{ev} \mathbf{o}(e_3)$$

Thus since $\mathbf{o}(e_1)$, $\mathbf{o}(e_2)$ $\mathbf{o}(e_3)$ become true in the same way in $P$ and $P'$ we can conclude for $M$ that

$$M, \pi_1 \models_{ev} \mathbf{o}(e_1) \texttt{ OP } \mathbf{o}(e_2) \text{ or}$$
$$M, \pi_2 \models_{ev} \mathbf{o}(e_3)$$

And thus $M, \pi \models_{ev} \mathbf{o}(e_1)$ `OP` $\mathbf{o}(e_2) \otimes \texttt{path} \otimes \mathbf{o}(e_3)$ which because of the rule for $\mathbf{o}(e)$ in $P$ implies:

$$M, \pi \models_{ev} \mathbf{o}(e)$$

And from this we conclude that $M', \pi \models_{ev} \mathbf{o}(e)$ implies $M\pi \models_{ev} \mathbf{o}(e)$ as intended

□

## C.3 Soundness and Completeness of $\mathcal{TR}^{ev}$'s Procedure

In this section we prove soundness and completeness of $\mathcal{TR}^{ev}$'s proof procedure. For that purpose, we will make a correspondence between the several functions of $\mathcal{TR}^{ev}$'s procedure and $\mathcal{TR}^{ev}$'s model theory definitions. This correspondence is as follows.

| Procedure | Theory |
|---|---|
| $ESet$ | $\models_{ev}$ |
| $FirstInOrder$ | $choice$ |
| $ExpandPath$ | $\texttt{exp}_M(\pi)$ |
| $Execution$ | $\models$ |

### C.3.1 Soundness

**Lemma 41** (Soundness ESet)**.** *Let $P$ be a $\mathcal{TR}^{ev}$ program, and $M$ a minimal model of this program. Let $\pi$ be the current path of the $ExpandPath$ call on which the function $Closure$ is called, and $ESet$ the current set of events that are known to hold.*

***If*** $\mathbf{o}(e)\{id_i, id_f\} \in ESet$ ***then*** $M, \pi_{<id_i,id_f>} \models_{ev} \mathbf{o}(e)$

*Proof.* By definition, $\mathbf{o}(e)\{id_i, id_f\} \in ESet$, either because it is added by the $ExpandPath$ call, or because of the base cases of the $Closure$ function. We look into each of these cases separately.

1. *ExpandPath:*
   $\mathbf{o}(e)\{id_i, id_f\} \in ESet$ by the ExpandPath call, then either of the two following cases:

   (a) *External Action Execution:*
   $\mathbf{o}(e)\{id_i, id_f\} \in ESet$ if $e \in \mathcal{P}_{\mathcal{O}}$, $id_f = id_{i+1}$, and $\mathcal{O}^t(\pi_{<id_i,id_{i+1}>}) \models e$. In this case, we know by definition of $M$ that $M\pi_{<id_i,id_{i+1}>} \models_{ev} \mathbf{o}(e)$.

   (b) *Explicit Event Request:*
   $\mathbf{o}(e)\{id_i, id_f\} \in ESet$ if $e \in \mathcal{P}_e$, $id_f$ = $id_{i+1}$ and $\pi_{<id_i,id_{i+1}>} = \langle D^{\mathbf{o}(e)} \rightarrow D \rangle$. In this case, we know by definition of $M$ that $M\pi_{<id_i,id_{i+1}>} \models_{ev} \mathbf{o}(e)$.

2. $Closure$ Rules:
   Now we look into the definition of the $Closure$ base cases (where events are added to $ESet$).

   Base Case 1. If $\mathbf{o}(e')\{id_i, id_f\} \in ESet$ and $\mathbf{o}(e') \Rightarrow \mathbf{o}(e) \in P'$ then $\mathbf{o}(e)\{id_i, id_f\}\} \in ESet$.
   Since $\mathbf{o}(e')\{id_i, id_f\} \in ESet$, by Induction Hypothesis, we know that $M, \pi_{<id_i,id_f>} \models_{ev} \mathbf{o}(e')$.
   Since this rule is a *permanent rule* then we know that it must belong to the original program $P$, and $M$ is a model of this program, which means that whenever $M$ models the body of a rule, it must also model the head in the same path. Consequently, since $M, \pi_{<id_i,id_f>} \models_{ev} \mathbf{o}(e')$, we know $M, \pi_{<id_i,id_f>} \models_{ev} \mathbf{o}(e)$.

   Base Case 2. If $\mathbf{o}(e')\{id_i, id_f\} \in ESet$, $\mathbf{o}(e') \otimes \overset{id_1}{\underset{id_2}{\Rightarrow}} \mathbf{o}(e) \in P'$ and $id_i = id_2$ then $\mathbf{o}(e)\{id_1, id_f\} \in ESet$.
   In this case, since $\mathbf{o}(e') \otimes \overset{id_1}{\underset{id_2}{\Rightarrow}} \mathbf{o}(e) \in P'$ then it must exist a rule of one of the following forms:

   i. $\mathbf{o}(e_1) \otimes \mathbf{o}(e') \Rightarrow \mathbf{o}(e) \in P$
   If this is the case, then we know that $\mathbf{o}(e_1)\{id_1, id_2\}, \mathbf{o}(e')\{id_2, id_f\} \in ESet$. By Induction Hypothesis we know that $M, \pi_{<id_1,id_2>} \models_{ev} \mathbf{o}(e_1)$ and $M, \pi_{<id_2,id_f>} \models_{ev} \mathbf{o}(e')$. By definition of $\models_{ev}$ we know that $M, \pi_{<id_1,id_f>} \models_{ev} \mathbf{o}(e_1) \otimes \mathbf{o}(e')$. Since $M$ is a model of $P$, we can conclude that $M, \pi_{<id_1,id_f>} \models_{ev} \mathbf{o}(e)$.

   ii. $\mathbf{o}(e_1); \mathbf{o}(e') \Rightarrow \mathbf{o}(e)$
   If this is the case, then we know that $\mathbf{o}(e_1)\{id_1, id_2\}, \mathbf{o}(e')\{id_i, id_f\} \in ESet$, where $id_2 \leq id_i$. By Induction Hypothesis we know that $M, \pi_{<id_1,id_2>} \models_{ev} \mathbf{o}(e_1)$ and $M, \pi_{<id_i,id_f>} \models_{ev} \mathbf{o}(e')$. By definition of $\models_e v$ we know that $M, \pi_{<id_1,id_f>} \models_{ev} \mathbf{o}(e_1); \mathbf{o}(e')$. Since $M$ is a model of $P$, we can conclude that $M, \pi_{<id_1,id_f>} \models_{ev} \mathbf{o}(e)$.

   Base Case 3. If $\mathbf{o}(e')\{id_i, id_f\} \in ESet$, $\mathbf{o}(e') \wedge \overset{id_1}{\underset{id_2}{\Rightarrow}} \mathbf{o}(e) \in P'$, $id_1 = id_i, id_2 = id_f$ then $\mathbf{o}(e)\{id_i, id_f\} \in ESet$

In this case, since $\mathbf{o}(e') \overset{id_1}{\underset{id_2}{\wedge\Rightarrow}} \mathbf{o}(e) \in P'$ then it must exist a rule in $P$ of the form: $\mathbf{o}(e_1) \wedge \mathbf{o}(e') \Rightarrow \mathbf{o}(e)$ and $\mathbf{o}(e_1)\{id_1, id_2\} \in ESet$. From this we can apply the Induction Hypothesis to $\mathbf{o}(e_1), \mathbf{o}(e)$ and conclude $M, \pi_{<id_1,id_2>} \models_{ev} \mathbf{o}(e_1)$ and $M, \pi_{<id_1,id_2>} \models_{ev} \mathbf{o}(e')$. Then by definition $\models_{ev}$ we know that $M, \pi_{<id_1,id_2>} \models_{ev} \mathbf{o}(e_1) \wedge \mathbf{o}(e')$. Since $M$ is a model of $P$, we can conclude that $M, \pi_{<id_1,id_f>} \models_{ev} \mathbf{o}(e)$.

Base Case 4. If $\mathbf{o}(e')\{id_i, id_f\} \in ESet$, $\mathbf{o}(e') \overset{*}{\underset{*}{\otimes\Rightarrow}} \mathbf{o}(e) \in P'$ then $\mathbf{o}(e)\{^*id_i, id_f\}$

In this case, since $\mathbf{o}(e') \overset{*}{\underset{*}{\otimes\Rightarrow}} \mathbf{o}(e) \in P'$ then it must exist a rule in $P$ of the form: $\texttt{path} \otimes \mathbf{o}(e') \Rightarrow \mathbf{o}(e)$. By Induction Hypothesis we know $M, \pi_{<id_i,id_f>} \models_{ev} \mathbf{o}(e')$. Since $\texttt{path}$ is true in any path of arbitrary size, we know that $M, \pi_{<id_1,id_f>} \models_{ev} \texttt{path} \otimes \mathbf{o}(e')$ $\forall.id_1$ s.t. $id_1 \leq id_i$. Since $M$ is a model of $P$, we can conclude that $M, \pi_{<id_1,id_f>} \models_{ev} \mathbf{o}(e)$ $\forall.id_1$ s.t. $id_1 \leq id_i$.

□

**Lemma 42** (Soundness $FirstInOrder$)**.** *Let $P$ be a program and $M$ a minimal model of this program. Assume we are within an $ExpandPath$ computation, in a $FirstInOrder$ call, where $\pi$ is the current path and $ESet$ the current set of unanswered events.*

$$\textit{If } FirstInOrder(ESet) = \mathbf{o}(e) \textit{ then } choice(M, \pi) = \mathbf{o}(e)$$

*Proof.* We should note that $choice$ and $FirstInOrder(ESet)$ follow the exact same definition of priority. As such, if $FirstInOrder(ESet) = \mathbf{o}(e)$ then, it means that $e$ is the event in $ESet$ with higher priority. Since $e \in ESet$, then by lemma 41, we know that there are $id_i, id_f$ such that $M, \pi_{<id_i,id_f>} \models_{ev} \mathbf{o}(e)$. Since the priority definition is equal for $choice$ and $FirstInOrder$ then we can conclude $choice(M, \pi) = e$. □

**Lemma 43** (Soundness $Execute$ and $ExpandPath$)**.** *Since by definition $\models$ and $\texttt{exp}_M(\pi_1)$ depend on each other, then their proofs need also to be written in this way, and are shown simultaneously. In addition, it should be noted that, in every call of the $ExpandCall$, $P, \pi \vdash \phi$ is called for a $\pi$ always smaller than the $\pi_1$ of $ExpandPath$.*

*Assume $P$ be a $\mathcal{TR}^{ev}$ program, and $M$ any minimal model of $P$.*

1. *Let $P$ be $\pi$ a path and $\phi$ a transaction formula.*

$$\textit{If } P, \pi \vdash \phi \textit{ then } M, \pi \models \phi$$

2. *Assume we are inside an $Execute$ procedure in $\vdash$ where we start an $ExpandPath$ call with $P$, event $A$, path $\pi_1$.*

$$\textit{If } ExpandPath(P, A, \pi_1, \{\mathbf{o}(A)\}, id) = (P', \pi', ESet', id') \textit{ then } \pi' \in \texttt{exp}_M(\pi_1)$$

*Proof.* We show each of these items separately.

**Soundness of** $\vdash$ To prove this, we show the soundness of the axiom, and the soundness of the rules.

**Soundness of Axiom:**

Formula () is a tautology that holds in paths of size 1. Thus if derivation starts in $\langle D\rangle, \emptyset \Vdash^0_P ()$ it successfully ends in $\langle D\rangle, \emptyset \Vdash^0_P ()$, and $M, \langle D\rangle \models ()$, for any state $D$.

**Soundness of Rules:**

1. *Unfolding of Rule:*
   We need to show that if there is a rule $L_1 \leftarrow Body \in P$ and $M, \pi \models Body \otimes L_2 \otimes \ldots \otimes L_k$ then $M, \pi \models L_1 \otimes L_2 \otimes \ldots \otimes L_k$.
   We know that $M, \pi \models Body \otimes L_2 \otimes \ldots \otimes L_k$. Thus by definition of $\otimes$, there must be a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1 \models Body$ and $M, \pi_2 \models L_2 \otimes \ldots \otimes L_k$. Since $M$ is a model of $P$, then it must also model the rule $L_1 \leftarrow Body$, meaning that $M, \pi_1 \models L_1$.
   Consequently, we know $M, \pi \models L_1 \otimes L_2 \otimes \ldots \otimes L_k$
2. *Query:*
   We need to show that if $\mathcal{O}^d(D_1) \models L_1$ and $M, \pi \models L_2 \otimes \ldots \otimes L_k$ then $M, \pi \models L_1 \otimes L_2 \otimes \ldots \otimes L_k$, where $D_1$ is the first state of $\pi$.
   Since $D_1$ is the first state of $\pi$, we know that $\langle D_1\rangle \circ \pi = \pi$ and $M, \langle D_1\rangle \models L_1$ (by definition of $M$). Consequently, by definition of $\otimes$, we know $M, \pi \models L_1 \otimes L_2 \otimes \ldots \otimes L_k$.
3. *Update Primitive:*
   We need to show that if $\mathcal{O}^t(D_1, D_2) \models L_1$, $M, \pi' \models L_2 \otimes \ldots \otimes L_k$ and $ExpandPath(P, L_1, \langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_2\rangle, ESet, id) = (P_2, \langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_2\rangle \circ \pi_1, ESet', id')$ then $M, \pi \models L_1 \otimes L_2 \otimes \ldots \otimes L_k$, where $\pi = \langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_2\rangle \circ \pi_1 \circ \pi'$.
   To prove this, we need to rely on the soundness of $ExpandPath$. As we shall see later in this proof, $ExpandPath(P, L_1, \langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_2\rangle, ESet, id) = (P_2, \langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_2\rangle \circ \pi_1, ESet', id')$ implies that we have $\pi_1 \in \mathtt{exp}_M(\langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_2\rangle)$. Then, since $\mathcal{O}^t(D_1, D_2) \models L_1$, by definition of interpretation we know: $M, \langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_2\rangle \circ \pi_1 \models L_1$.
   Moreover by definition of the procedure, $\pi'$ starts in the same state as $\pi_1$ ends, and thus $M, \langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_2\rangle \circ \pi_1 \circ \pi' \models L_1 \otimes L_2 \otimes \ldots \otimes L_k$
4. *Explicit event request:*
   We need to show that if $L_1$ is an explicit event, $M, \pi' \models L_2 \otimes \ldots \otimes L_k$ and $ExpandPath(P, L_1, \langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_1\rangle, ESet, id) = (P_2, \langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_1\rangle \circ \pi_1, ESet', id')$ then $M, \pi \models L_1 \otimes L_2 \otimes \ldots \otimes L_k$, where $\pi = \langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_1\rangle \circ \pi_1 \circ \pi'$.
   To prove this, we need to rely on the soundness of $ExpandPath$. As we shall see later in this proof, $ExpandPath(P, L_1, \langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_1\rangle, ESet, id) = (P_2, \langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow D_2\rangle \circ \pi_1, ESet', id')$ implies that we have $\pi_1 \in \mathtt{exp}_M(\langle D_1 {}^{\mathbf{o}(L_1)}\!\!\rightarrow$

$D_1\rangle$). Then, since $\mathcal{O}^t(D_1, D_2) \models L_1$, by definition of interpretation we know: $M, \langle D_1 \xrightarrow{\mathbf{o}(L_1)} D_1\rangle \circ \pi_1 \models L_1$.
Moreover by definition of the procedure, $\pi'$ starts in the same state as $\pi_1$ ends, and thus $M, \langle D_1 \xrightarrow{\mathbf{o}(L_1)} D_1\rangle \circ \pi_1 \circ \pi' \models L_1 \otimes L_2 \otimes \ldots \otimes L_k$

**Soundness of** $ExpandPath$

We show that each iteration is sound w.r.t. $\mathcal{R}_M$, and then that its stopping condition, makes $ExpandPath$ sound with $\pi' \in \texttt{exp}_M(\pi_1)$

Let's look into the definition of $ExpandPath$:

- $Closure(ESet', P', id') = (ESet_1, P_1)$
- $FirstInOrder(\texttt{needResponse}(ESet_1, id, id')) = \mathbf{o}(e)$
- There is a derivation starting in $\langle D\rangle, \emptyset \Vdash^{id'}_{P_1} \mathbf{r}(e)$ and ending in $\pi_i, ESet_i \Vdash^{id_i}_{P_i} ()$, where $D$ is the final state of $\pi_1$

If this is true, then $choice(M, \pi_1) = \mathbf{o}(e)$ by lemma 42. Moreover, since, as shown previously $Execute$ is sound if $ExpandPath$ is sound, then we know $M, \pi_i \models \mathbf{r}(e)$ where $\pi' = \pi \circ \pi_i$. Note that $\pi_i$ of $Execute$ is always smaller than the path of $ExpandPath$.

Finally, if $\texttt{needResponse}(ESet_1, =, \emptyset)$ then there are no further events to respond, $choice(M, \pi_1) = \epsilon$ and thus $\pi' \in \texttt{exp}_M(\pi_1)$.

□

### C.3.2 Completeness

In our completeness proof we demonstrate that if $P, \pi \models \phi$ then $P, \pi \vdash \phi$. As in $\mathcal{ETR}$, we show this by constructing a canonical interpretation $M_P$ that proves the same events and transaction formulas as the procedure. Then we show that this interpretation is indeed a model of the program $P$, and is minimal. Afterwards, based on this interpretation, we show the completeness of the $ExpansionPath$ and $Execute$ computations.

For these proofs we assume the statement $P, \pi \vdash_{Set} \phi$ defined as follows.

**Definition 98.** *Let $P$ be a program, $\phi$ a formula and $\pi$ a path. We denote $P, \pi \vdash_{Set} \phi$ as the computation $P, \pi \vdash \phi$ and where $Set$ is a set containing all the events $\mathbf{o}(e)\{id_i, id_f\}$ that belonged to the $ESet$ during any part of the computation.*

Subsequently, we continue by defining the canonical interpretation $M_P$.

**Definition 99** (Canonical Interpretation)**.** *Let $P$ be a program and $\pi$ a path. $M_P$ is defined as follows.*

$$
\begin{aligned}
M_P(\pi) = \quad & \{a \mid a \textit{ is a transaction atom } \wedge P, \pi \vdash a\} \\
& \cup \{\mathbf{o}(e) \mid e \in \mathcal{P}_e \wedge \quad \exists \textit{ transaction formula } \phi, \textit{ a path } \pi' \textit{ s.t.} \\
& \qquad\qquad P, \pi' \vdash_{Set} \phi \wedge \mathbf{o}(e)\{id_i, id_f\} \in ESet \wedge \pi'_{<id_i, id_f>} = \pi\} \\
& \cup \{head \mid body \Rightarrow head \in P \wedge \neg\exists \phi, \pi' \textit{ s.t. } P, \pi' \vdash \phi \textit{ where } \pi \textit{ is a subpath of } \pi'\}
\end{aligned}
$$

In the following we show that if a procedure exists for a path $\pi'$ and a transaction formula $\phi$, then all events made true by any minimal model of $P$, are made true in the procedure (i.e., they belong to $Set$) on the exact same path.

**Lemma 44.** *Let $P$ be a program, $M$ any minimal model of $P$, $\mathbf{o}(e)$ an event occurrence and $\pi$ a path. Assume there is a $\pi'$ and a transaction formula $\phi$ s.t. $P, \pi' \vdash_{Set} \phi$, where $\pi$ is a subpath of $\pi'$.*

$$\textit{If } M, \pi \models_{ev} \mathbf{o}(e) \textit{ then } \mathbf{o}(e)\{id_i, id_f\} \in ESet \textit{ and } \pi'_{<id_i, id_f>} = \pi.$$

*Proof.* We show this by constructing the set of all atomic events that $Set$ makes true in paths of this form.

**Base Cases:**

- *action oracle formulas*
  If $M, \pi \models_{ev} \mathbf{o}(e)$ then $\pi = \langle D_1 \xrightarrow{\mathbf{o}(e)} D_2 \rangle$ and $\mathcal{O}^t(D_1, D_2) \models e$. In this case, and since $\pi$ is a subpath of $\pi'$, then we know that rule 3 of definition 61 of $Execute$ is called and that for some $id_i, id_{i+1}$ $\mathbf{o}(e)\{id_i, id_{i+1}\} \in Set$ where $\pi'_{<id_i, id_{i+1}>} = \langle D_1 \xrightarrow{\mathbf{o}(e)} D_2 \rangle$
- *explicit event*
  If $M, \pi \models_{ev} \mathbf{o}(e)$ then $\pi = \langle D_1 \xrightarrow{\mathbf{o}(e)} D_1 \rangle$ where $e$ is an explicit eventIn this case, and since $\pi$ is a subset of $\pi'$, then we know that rule 4 of definition 61 of $Execute$ is called and that for some $id_i, id_{i+1}$ $\mathbf{o}(e)\{id_i, id_{i+1}\} \in Set$ where $\pi'_{<id_i, id_{i+1}>} = \langle D_1 \xrightarrow{\mathbf{o}(e)} D_1 \rangle$

**Heads of the rules where** $body \in Set$

We now prove that if $\mathbf{o}(e)$ is the head of a rule in $P$ which every atomic formula of the (possibly complex) *body* belongs to $Set$, then $M, \pi \models_{ev} \mathbf{o}(e)$ implies $\mathbf{o}(e)\{id_i, id_f\} \in Set$ and $\pi'_{<id_i, id_f>} = \pi$.

To prove this, we look into the structure of *body*.

1. $body \equiv \mathbf{o}(e_1) \otimes \mathbf{o}(e_2)$

   $M, \pi \models_{ev} \mathbf{o}(e_1) \otimes \mathbf{o}(e_2)$, then there is a split $\pi_1 \circ \pi_2$ s.t. $M, \pi_1 \models_{ev} \mathbf{o}(e_1)$ and $M, \pi_2 \models_{ev} \mathbf{o}(e_2)$.

   Since $M, \pi_1 \models_{ev} \mathbf{o}(e_1)$, $\mathbf{o}(e_1) \otimes \mathbf{o}(e_2) \Rightarrow \mathbf{o}(e)$ is in the original program $P$ and $\mathbf{o}(e_1)\{id_i, id_f\} \in Set$, then we know by Rule 1. of the Operation Case of $Closure$ definition that, a temporary rule of the form $\mathbf{o}(e_2) \otimes \overset{id_i}{\underset{id_f}{\Rightarrow}} \mathbf{o}(e)$ is added to $P'$ in the procedure.

Moreover, since $M, \pi_2 \models_{ev} \mathbf{o}(e_2)$ then $\mathbf{o}(e_1)\{id_{i2}, id_{f2}\} \in Set$. If this is the case, and since $\pi = \pi_1 \circ \pi_2$, then we know $id_{i2} = id_f$ and by Rule 2. of the Base Case of $Closure$, we know $\mathbf{o}(e)\{id_i, id_{f2}\} \in Set$ and that $\pi'_{<id_i, id_{f2}>} = \pi$.

2. $body \equiv \mathbf{o}(e_1); \mathbf{o}(e_2)$

   $M, \pi \models_{ev} \mathbf{o}(e_1) \otimes \mathbf{o}(e_2)$, then there is a split $\pi_1 \circ \pi_m \circ \pi_2$ s.t. $M, \pi_1 \models_{ev} \mathbf{o}(e_1)$ and $M, \pi_2 \models_{ev} \mathbf{o}(e_2)$.

   Since $M, \pi_1 \models_{ev} \mathbf{o}(e_1)$, $\mathbf{o}(e_1) \otimes \mathbf{o}(e_2) \Rightarrow \mathbf{o}(e)$ is in the original program $P$ and $\mathbf{o}(e_1)\{id_i, id_f\} \in Set$, then we know by Rule 3. of the Operation Case of $Closure$ definition that, a temporary rule of the form $\mathbf{o}(e_2) \otimes \overset{id_i}{\underset{id_f^*}{\Rightarrow}} \mathbf{o}(e) \in P$

   Moreover, since $M, \pi_2 \models_{ev} \mathbf{o}(e_2)$ then $\mathbf{o}(e_1)\{id_{i2}, id_{f2}\} \in Set$. If this is the case, and since $\pi = \pi_1 \circ \pi_m \circ \pi_2$, then we know $id_f \leq id_{i2}$ and by Rule 2. of the Base Case of $Closure$, we know $\mathbf{o}(e)\{id_i, id_{f2}\} \in Set$ and that $\pi'_{<id_i, id_{f2}>} = \pi$.

3. $body \equiv \mathbf{o}(e_1) \wedge \mathbf{o}(e_2)$

   $M, \pi \models_{ev} \mathbf{o}(e_1) \otimes \mathbf{o}(e_2)$, then $M, \pi \models_{ev} \mathbf{o}(e_1)$ and $M, \pi \models_{ev} \mathbf{o}(e_2)$.

   Since $M, \pi \models_{ev} \mathbf{o}(e_1)$, $\mathbf{o}(e_1) \otimes \mathbf{o}(e_2) \Rightarrow \mathbf{o}(e)$ is in the original program $P$ and $\mathbf{o}(e_1)\{id_i, id_f\} \in Set$, then we know by Rule 2. of the Operation Case of $Closure$ definition that, a temporary rule of the form $\mathbf{o}(e_2) \wedge \overset{id_i}{\underset{id_f}{\Rightarrow}} \mathbf{o}(e)$ is created.

   Moreover, since $M, \pi \models_{ev} \mathbf{o}(e_2)$ then $\mathbf{o}(e_2)\{id_i, id_f\} \in Set$. If this is the case, then by Rule 3. of the Base Case of $Closure$, we know $\mathbf{o}(e)\{id_i, id_f\} \in Set$ and that $\pi'_{<id_i, id_f>} = \pi$.

4. $body \equiv \texttt{path} \otimes \mathbf{o}(e_2)$ $M, \pi \models_{ev} \mathbf{o}(e_1) \otimes \mathbf{o}(e_2)$ then there is a split $\pi_1 \circ \pi_2$ s.t. $M, \pi_1 \models_{ev}$ $\texttt{path}$ and $M, \pi_2 \models_{ev} \mathbf{o}(e_1)$. Note that, $\texttt{path}$ is a formula that is true in paths of arbitrary size.

   If $\texttt{path} \otimes \mathbf{o}(e_1) \Rightarrow \mathbf{o}(e) \in P$, then by the Path Case, Rule 1. of $Closure$ we know that $\mathbf{o}(e_1) \otimes \overset{*}{\underset{*}{\Rightarrow}} \mathbf{o}(e)$ is created in $P$. Moreover, since $M, \pi_2 \models_{ev} \mathbf{o}(e_1)$ then we know that $\mathbf{o}(e)\{id_i, id_f\} \in Set$. If this is the case, then by Rule 4. of the Base Case of $Closure$ we know that $\mathbf{o}(e)\{{}^*id_i, id_f\}$ where ${}^*id_i$ is a left open id. As such, it matches several paths when used in path trimming, and thus $\pi'_{<{}^*id_i, id_f>} = \pi$

5. $body \equiv \mathbf{not}(\mathbf{o}(e_3))[\mathbf{o}(e), \mathbf{o}(e_2)]$

   $M, \pi \models_{ev} \mathbf{not}(\mathbf{o}(e_3))[\mathbf{o}(e), \mathbf{o}(e_2)]$ then there is a split $\pi_1 \circ \pi_3 \circ \pi_2 = \pi$ s.t. $M, \pi_1 \models_{ev}$ $\mathbf{o}(e_1)$, $M, \pi_2 \models_{ev} \mathbf{o}(e_2)$ and $M, \pi_3 \not\models_{ev} \texttt{path} \otimes \mathbf{o}(e_3) \otimes \texttt{path}$.

   Since $M, \pi_1 \models_{ev} \mathbf{o}(e_1)$ then $\mathbf{o}(e_1)\{id_i, id_f\} \in Set$ and thus by Negation Case, Rule 1. that the rules $\mathbf{o}(e_2) \otimes \overset{id_i}{\underset{id_f^*}{\Rightarrow}} \mathbf{o}(e)$ and $\mathbf{o}(e_3) \neg \overset{id_i}{\underset{id_f^*}{\Rightarrow}} (\mathbf{o}(e_2) \otimes \overset{id_i}{\underset{id_f^*}{\Rightarrow}} \mathbf{o}(e))$ are added to the program.

   Moreover, we know that $M, \pi_2 \models_{ev} \mathbf{o}(e_2)$ and thus $\mathbf{o}(e_2)\{id_{i2}, id_{f2}\} \in Set$. In order to apply the Base Case Rule 2 over the first rule added to the program, we

need to make sure that $\mathbf{o}(e_3) \neg \overset{id_i}{\underset{id_f^*}{\Rightarrow}} (\mathbf{o}(e_2) \otimes \overset{id_i}{\underset{id_f^*}{\Rightarrow}} \mathbf{o}(e))$ was not triggered, i.e., that $\mathbf{o}(e_3)\{id_{i3}, id_{f3}\} \notin Set$ where $id_i \leq id_{i3} \leq id_{i2}$ and $id_f \leq id_{f3} \leq id_{f2}$.

We show this by contradiction. Assume that $\mathbf{o}(e_3)\{id_{i3}, id_{f3}\} \in Set$, then by Soundness we know that $M', \pi_i \models_{ev} \mathbf{o}(e_3)$ for any minimal model $M'$. Since this is true for any minimal model, then it is also true for $M$. Moreover since $id_i \leq id_{i3} \leq id_{i2}$ and $id_f \leq id_{f3} \leq id_{f2}$, then $M, \pi_3 \models_{ev} \texttt{path} \otimes \mathbf{o}(e_3) \otimes \texttt{path}$ and we reach a contradiction.

Consequently $\mathbf{o}(e_3)\{id_{i3}, id_{f3}\} \notin Set$ and we can apply Base Case Rule 2. and conclude $\mathbf{o}(e)\{id_i, id_{f2}\}$ where $\pi'_{<id_i, id_{f2}>} = \pi$

Node that we made no assumption about the structure of $\mathbf{o}(e_1)$ and $\mathbf{o}(e_2)$. Consequently, we can conclude that the head of the rule $\mathbf{o}(e)$ belongs to the $Set$, whenever the bodies $\mathbf{o}(e_1)$ and $\mathbf{o}(e_2)$ also belong to the $Set$. Moreover, the construction of $Set$ through this process clearly converges, since the bodies of each rule are binary, and the amount of rules is finite.

It remains to show that all events that $M$ satisfies over subpaths of $\pi'$ are in $Set$. To prove this, assume there is an occurrence $\mathbf{o}(e_x)$ s.t. $M, \pi \models_{ev} \mathbf{o}(e_x)$ and $\mathbf{o}(e_x) \notin Set$. If this is the case, then $\mathbf{o}(e_x)$ cannot be an action from the oracle, or an explicit event (otherwise by our Base Case proof, it has to be in $Set$). Additionally, $\mathbf{o}(e_x)$ cannot be the head of the rule where the head is supported by an action from the oracle or an explicit event, or any event defined in this way.

Consequently, the only possibility is for $\mathbf{o}(e_x)$ be an occurrence that is not defined in the head of any rule in $P$, or the head of a rule in $P$ with body not made true eventually by applying the previous process. However, if this is the case, then $M, \pi \models_{ev} \mathbf{o}(e_x)$ only if $M$ is not minimal, and thus we reach a contradiction.

From this we can conclude that for paths where a derivation $\vdash$ exists, then the $Set$ is complete w.r.t to all events made true by minimal models on the same paths. □

**Lemma 45.** *Let $M_P$ be the canonical interpretation of the program $P$. Then $M_P$ models every event rule in $P$.*

*Proof.* We need to show that if $M_P, \pi \models_{ev} body$ then $M_P, \pi \models_{ev} head$

By definition of $M_P$, this result is clear for paths $\pi$ where there is no superpath $\pi'$ s.t. $P, \pi' \vdash \phi$. Moreover, for the other cases, by lemma 44 we know that if there is a minimal model s.t. $\mathbf{o}(e) \in M(\pi)$ then $\mathbf{o}(e) \in Set$ and thus by definition of $M_P$, $M_p, \pi \models_{ev} \mathbf{o}(e)$.

Since $M_P$ and $M$ satisfy the same event atoms over the same paths, if $M_P, \pi \models_{ev} body$ then $M, \pi \models_{ev} body$. Since $M$ is a model of the program we know that if $M, \pi \models_{ev} body$ then $M, \pi \models_{ev} head$. Since $M$ and $M_P$ satisfy the same events we know $M, \pi \models_{ev} head$ □

**Lemma 46.** *Let $M_P$ be the canonical interpretation of the program $P$. $M_P$ is a satisfies every event formula $\mathbf{o}(e)$.*

*Proof.* We know that $M_P$ models every event rule by lemma 45, so it remains to prove that it is minimal.

This result is trivial for paths $\pi$ s.t. there is a superpath $\pi'$ of $\pi$ where $P, \pi' \vdash \phi$, since $M_P$ only makes formulas true that are true in minimal paths (by soundness of $ESet$).

Let's consider the case where $M_P, \pi \models_{ev} \mathbf{o}(e)$ and $\pi$ is not defined in the previous case. If so, we know that there must exist a rule $body \Rightarrow \mathbf{o}(e)$ and $M_P, \pi \models_{ev} body$.

From this we know that eventually for $body$ to be true over $\pi$, then there must exist a set of events that make $body$ true because there is a subpath $\pi_1$ of $\pi$ where $\mathbf{o}(e) \in M(\pi_1)$ and $P, \pi_1' \vdash_{Set} \phi$ where $\pi_1'$ is a super path of $\pi_1$ but not of $\pi$.

Since all formulas made true in these paths are also made true in any minimal model $M_{min}$, then $M_{min}$ and $M_P$ must make the same formulas true over the same paths. □

**Lemma 47** (Completeness FirstInOrder)**.** *Let $P$ be a program and $M$ a minimal model of this program. Let $\pi$ be a path s.t. there is a super path $\pi'$ and a transaction formula $\phi$, where $P, \pi' \vdash_{Set} \phi$. Let $ESet \subseteq Set$.*

$$\textit{If } choice(M, \pi) = \mathbf{o}(e) \textit{ then } FirstInOrder(ESet) = \mathbf{o}(e)$$

*Proof.* We should note that $choice$ and $FirstInOrder(ESet)$ follow the exact same definition of priority. Thus since $\models_{ev}$ is complete w.r.t. the events that appear in $Set$ for paths of this kind, then we know that $FirstInOrder(ESet)$ will follow the same definition and conclude $FirstInOrder(ESet) = \mathbf{o}(e)$ □

**Lemma 48** (Completeness $ExpansionPath$ and $Execute$)**.** *Since by definition, $\models$ and $\texttt{exp}_M(\pi_1)$ depend on each other, then their proofs need also to be written in this way, and are shown simultaneously. In addition, it should be noted that, in every call of the $ExpandCall$, $P, \pi \vdash \phi$ is called for a $\pi$ always smaller than the $\pi_1$ of $ExpandPath$.*

*Assume $P$ be a $\mathcal{TR}^{ev}$ program, and $M$ any minimal model of $P$, and $\phi$ a serial-Horn transaction formula.*

1. *If $M_P, \pi \models \phi$ then $P, \pi \vdash \phi$*
2. *If there is a path s.t. $P, \pi' \vdash_{Set} \phi$ and $\pi_2 \in \texttt{exp}_{M_P}(\pi_1)$ for paths $\pi_1$ and $\pi_2$ subpaths of $\pi'$ Then there are some $id, id'$ s.t. $ExpandPath(P, \phi, \pi_1, ESet, id) = (P_2, \pi_1 \circ \pi_2, ESet', id')$*

*Proof.* We prove each statement in turn.

1. If $M_P, \pi \models \phi$ then $P, \pi \vdash \phi$

   We prove this by induction on the $k$-size of the serial-Horn $\phi \equiv \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_k$.

   $k = 0$**:** $M_P, \pi \models ()$ iff $\pi$ is a 1-path. Then the derivation $\langle D \rangle, \emptyset \Vdash_P^1 ()$ succeeds and $P, \langle D \rangle \vdash ()$ holds for any state $D$.

$k = 1$**:** $M_P, \pi \models \phi_1$, and $\phi_1$ is an atom. Then by definition of the canonical interpretation we know that $P, \pi \vdash \phi_1$.

$k = n + 1$**:** Assume the result holds for formulas of size $n$, we need to prove that it still holds for formulas of size $n + 1$.

$M_P, \pi \models \phi_1 \otimes \ldots \otimes \phi_n \otimes \phi_{n+1}$ then by definition of $M_P$ we know that there is a split of $\pi = \pi_1 \circ \pi_2 \circ \pi_3$ s.t. $M_P, \pi_1 \models \phi_1 \otimes \ldots \otimes \phi_n$, $M_P, \pi_2 \models \phi_{n+1}$ and $\pi_1 \circ \pi_2 \circ \pi_3 \in \texttt{exp}_{M_P}(\pi_1 \circ \pi_2)$.

From this we know by Induction Hypothesis that $P, \pi_1 \vdash \phi_1 \otimes \ldots \otimes \phi_n$ and $P, \pi_2 \vdash \phi_{n+1}$. Since this is the case, then by definition of the procedure $P, \pi \vdash \phi_1 \otimes \ldots \otimes \phi_n \otimes \phi_{n+1}$ if $ExpandPath(P, \phi_{n+1}, \pi_1 \circ \pi_2, ESet, id) = (P_2, \pi_1 \circ \pi_2 \circ \pi_3, ESet', id')$.

Based on this, we can now use the proof that will be shown next, and conclude that $\texttt{exp}_{M_P}(\pi)$ implies the latter statement.

2. We need to show that if there is a path s.t. $P, \pi' \vdash_{Set} \phi$ and $\pi_2 \in \texttt{exp}_{M_P}(\pi_1)$ for paths $\pi_1$ and $\pi_2$ subpaths of $\pi'$ then there are some $id, id'$ s.t. $ExpandPath(P, \phi, \pi_1, ESet, id) = (P_2, \pi_1 \circ \pi_2, ESet', id')$.

   We know that $\pi_2 \in \texttt{exp}_{M_P}(\pi_1)$ if for all events that are true in $\pi_1$ they are responded in $\pi_2$. I.e. $\forall \mathbf{o}(e)$ s.t. $M, \pi_1' \models_{ev} \mathbf{o}(e)$ (for subpaths $\pi_1'$ of $\pi_1$), then there is a path $\pi_2'$ subpath of $\pi_1 \circ \pi_2$ where $M_P, \pi_2' \models \mathbf{r}(e)$, where events are responded in the order defined by $choice$

   If this is the case, then by lemma 44 we know that $\mathbf{o}(e)\{id_i, id_f\} \in Set$. Since $FirstInOrder$ and $choice$ give the same results as $choice$, and $M_P, \pi_2' \models \mathbf{r}(e)$ is complete w.r.t. $P, \pi \vdash \mathbf{r}(e)$ for paths smaller than $\pi$, then we have all conditions of the $ExpandPath$ function, and $ExpandPath(P, L_1, \pi_1, ESet, id) = (P_2, \pi_1 \circ \pi_2, ESet', id')$.

□

**Lemma 49.** *$M_P$ is a minimal model of $P$*

*Proof.* In order to show that $M_P$ is a minimal model, we need to show that $M_P$ models all transaction rules and event rules of $P$, and that it minimizes the amount of formulas made true in paths.

We start showing that $M_P$ is a model. By lemma 45 we know that $M_P$ models every event rule, so we need to show that it models every transaction rule.

Let's assume that $head \leftarrow body \in P$ and $M_P, \pi \models body$. Then by item 1 of lemma 48 we know that $P, \pi \vdash body$. If this is the case, then by rule 1 of the definition of the procedure, we also know that $P, \pi \vdash head$. If the latter is true, since $head$ is an atom, we know that $M_P, \pi \models head$

Finally, it remains to show that $M_P$ is minimal. We already know by lemma 46 that it is minimal w.r.t. the event formulas that it makes true over any path, so let's look into what happens with transaction formulas.

We show this by contradiction. Let's assume that $M_P$ is not minimal. If that is the case, then there is a $M_{min}$ minimal model of $P$ s.t. $M_P, \pi \models \phi$ and $M_{min}, \pi \not\models \phi$. As shown by item 1 of lemma 48, $M_P, \pi \models \phi$ implies $P, \pi \vdash \phi$. Since $P, \pi \vdash \phi$ is sound with every minimal model $M$ of $P$, then it is also sound with $M_{min}$, and thus $P, \pi \vdash \phi$ implies $M_{min}\pi \models \phi$, which leads to a contradiction. Consequently $M_P$ must be a minimal model. □

### C.3.3 Soundness and Completeness

**Theorem 11.**(Soundness and Completeness of the Procedure) *Let $P$ be a program, $\pi$ a path, and $\phi$ a transaction formula.*

$$P, \pi \vdash \phi \textit{ iff } P, \pi \models \phi$$

*Proof.* We prove this statement separately in each direction.
$\Rightarrow$: $P, \pi \vdash \phi$ implies $P, \pi \models \phi$

Immediately by lemma 43, as $P, \pi \vdash \phi$ implies $M, \pi \vdash \phi$ for every minimal model $M$ of $P$. Thus $P, \pi \models \phi$ holds.

$\Leftarrow$: $P, \pi \models \phi$ implies $P, \pi \vdash \phi$

If $P, \pi \models \phi$, then by definition, for all minimal models $M$ of $P$, $M, \pi \models \phi$. Since we know that $M_P$ is a minimal model of $P$ (lemma 49), we know that $M_P, \pi \models \phi$. Thus by item 1. of lemma 48 we know that $P, \pi \vdash \phi$. □