Ricardo Jorge Pombeiro Moreira Feio

Licenciatura em Ciências de Engenharia
Electrotécnica e de Computadores

IOPT2IEC61131 – Execução de redes de Petri em Autómatos industriais (PLCs)

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Orientador: Luís Filipe dos Santos Gomes, Prof. Doutor, FCT/UNL

Júri:

Presidente: Doutor João Almeida das Rosas – FCT/UNL

Vogais: Doutor Luís Filipe dos Santos Gomes – FCT/UNL
Doutor Filipe de Carvalho Moutinho – FCT/UNL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2016

IOPT2IEC61131 – Execução de Redes de Petri em Autómatos Industriais (PLCs)

Em memória ao meu pai Jorge Feio

que nos deixou tão cedo

Agradecimentos

Ao Professor Doutor Luís Filipe dos Santos Gomes, pelo apoio e orientação que me deu ao longo deste trabalho, sem a qual não teria sido possível concluir;

Ao Professor Doutor João Almeida das Rosas, pelo material de *software* e *hardware* disponibilizado, sem o qual não teria sido possível realizar os testes e simulações necessárias;

Ao Professor Filipe Moutinho, pela ajuda e atenção que me dedicou, sem a qual não teria sido possível concluir o protótipo;

Ao Professores de Mestrado, pela dedicação e profissionalismo que demonstraram ao longo destes anos;

Aos meus colegas de Mestrado, pela ajuda disponibilizada por todos individual e colectivamente;

À minha mãe Zulmira Feio, por ter estado sempre presente e disponível nos bons e maus momentos;

Ao meu irmão Mário Feio, pela paciência de me aturar há tantos anos;

E a todos os outros, que directa ou indirectamente contribuíram para a realização e conclusão deste trabalho.

Resumo

Os autómatos industriais têm sido continuamente usados para múltiplos fins no nosso dia-a-dia, desde as mais simples portas automáticas até às mais complexas máquinas de linhas de montagem. Para normalizar a programação dos autómatos industriais ou controladores lógicos programáveis (CLPs), foi criada a norma internacional *IEC61131* e em particular a *IEC61131-3*, com o intuito de obrigar os fabricantes e programadores de autómatos a respeitar determinadas regras. No entanto, a norma *IEC61131-3* contem linguagens de programação com certas limitações no que diz respeito à capacidade de modelar ou simular seus sistemas de controlo. As *IOPT Tools*, por outro lado, são uma ferramenta gráfica que permitem modelar e simular sistemas de controlo para autómatos industriais com redes de Petri *IOPT*. Surge o problema de traduzir a linguagem das redes de Petri para as da norma *IEC61131-3*. Este trabalho visa encontrar soluções para o problema da tradução de redes de Petri, em particular da ferramenta *IOPT*, para linguagens dos CLPs. Para tal, propõe-se um conjunto de regras de tradução de redes de Petri da classe *IOPT* para a linguagem de Diagramas *Ladder*/Lista de Instruções da Norma *IEC61131-3*.

Palavras-chave: Redes de Petri, RdP, CLP, Norma *IEC61131*, *Ladder*, *IOPT*

Abstract

Programmable logic controllers have been continuously used for multiple purposes in our daily lifes, from the simple automatic doors, to the more complex assembly line machines. In order to standardize the control programming of the programmable logic controllers (*PLC*s), the international *IEC61131* standard has been created, and in particular the *IEC61131-3*, with the objective of making PLCs manufacturers and programmers to follow certain rules. However, the *IEC61131* standard has several programming languages with certain limitations regarding the modeling and simulation of its control systems. *IOPT Tools*, on the other hand, are a graphical tool that support modeling and simulation of programmable logic controllers systems with *IOPT*-nets. The problem now is the translation of the Petri nets language to the ones from the *IEC61131-3* standard. This work seeks to find solutions for this translation problem. For this a set of translation rules from *IOPT*-nets to the *IEC61131* standard's Ladder Diagrams/Instruction List languages are proposed.

Keywords: Petri nets, PN, PLC, *IEC61131* Standard, *Ladder*, IL, *IOPT*

Índice de Conteúdo

Índice de Figuras

Lista de Acrónimos

CEM – Compatibilidade Electromagnética

CIE – Comissão Internacional de Electrotécnica

CLP – Controladores Lógicos Programáveis

EMC – Electromagnetic Compatibility

GALS – Globalmente Assíncronos Localmente Síncronos

IEC – International Electrotechnical Commission

PLC – Programmable Logic Controllers

IOPT – Input Output Place Transition

RdP – Redes de Petri

Simbologia e Notações

Geral

=	igual a
≠	diferente de
>	maior do que
≥	maior ou igual a
<	menor do que
≤	menor ou igual a
+	"ou" lógico
.	"e" lógico

Redes de Petri

P	lugar
T	transição

1 Introdução

1.1 Motivação

Nos dias que correm, o uso de controladores lógicos programáveis (CLPs) estende-se em praticamente todo o tipo de indústria, desde linhas de montagem, até ao controlo de montanhas russas, ou até mesmo de casas inteligentes.

Muitos destes sistemas de controlo devem seguir uma programação normalizada através de uma norma internacional criada com o intuito de permitir produtores de diversas marcas de CLPs encontrarem uma plataforma de trabalho comum, de forma a evitar problema de compatibilidade, entre outros factores. Está-se a falar obviamente na norma *IEC61131*, que contem um leque de diversas linguagens de programação de baixo nível especificamente para CLPs [1].

Com o evoluir constante das tecnologias e com o constante aumento de complexidade de seus sistemas de controlo, tem-se tornado cada vez mais difícil modelar e simular estes sistemas de controlo unicamente através das linguagens disponibilizadas pela norma *IEC61131* [2][3], visto estas serem linguagens de baixo nível.

Uma boa forma de criar modelos destes sistemas de controlo e proceder a sua simulação é fazer uso de uma outra linguagem gráfica conhecida por redes de Petri (RdPs). As redes de Petri, mais especificamente as da classe *IOPT (Input-Output Place-Transition)* [4], são muito adequadas para modelar desde os

sistemas de controlo mais pequenos e simples até aos maiores e mais complexos, vistos estas estarem presentes nas *IOPT Tools* disponibilizando muitas outras funcionalidades, como por exemplo capacidade de simular modelos *IOPT* [5].

Para aplicar o resultado da modelação de sistemas de controlo em redes de Petri, será necessário posteriormente proceder a sua tradução para uma das linguagens de programação da norma *IEC61131,* tal como visto em [6].

1.2 Objectivos

O objectivo principal desta dissertação, é desenvolver as regras de tradução que irão permitir transformar modelos expressos em redes de Petri *IOPT*, utilizando a ferramenta *IOPT Tools*, em duas das linguagens de programação da norma *IEC61131*.

Para tal, será necessário fazer um estudo prévio das diversas classes e propriedades comportamentais das redes de Petri, mais concretamente da classe *IOPT*, analisar o modo de funcionamento de algumas das linguagens de programação da norma internacional *IEC61131,* para depois se poder proceder à criação das regras de tradução e finalmente realizar testes e simulações em modelos de redes *IOPT* com determinadas propriedades suportadas por esta dissertação.

1.3 Estrutura da Dissertação

Este trabalho encontra-se dividido em seis capítulos:

Nos primeiros três capítulos apresenta-se uma introdução à dissertação, bem como um estudo de algumas características das redes de Petri e de algumas linguagens de Programação da norma *IEC61131*, constituindo respectivamente os capítulos um, dois e três.

1 – Introdução: Neste capítulo são descritos os motivos, objectivos e estrutura desta dissertação.

2 – Redes de Petri: Neste capítulo faz-se um estudo teórico das RdPs (Redes de Petri), inspeccionam-se algumas das diferentes classes de redes de Petri incluindo uma apresentação da classe *IOPT* bem como algumas das suas funcionalidades.

3 – Norma IEC61131: Neste capítulo são abordadas as várias partes que constituem a norma *IEC61131* e é feito um estudo mais detalhado da parte 3 da norma referente às linguagens de programação suportadas pela mesma, em particular, Diagramas *Ladder* e Lista de Instruções.

Após uma breve introdução às Redes de Petri e à norma internacional segue-se o capítulo quatro, onde são apresentadas as regras de tradução que irão suportar a passagem de redes de Petri da classe *IOPT* para as linguagens de Diagramas *Ladder* e Lista de Instruções.

4 – Regras de Tradução: No que diz respeito as regras de tradução criadas, é feita uma abordagem não só da perspectiva das redes de Petri seguras mas também das redes de Petri limitadas e várias propriedades das redes de Petri *IOPT*.

Finalmente é apresentado a aplicação das regras de tradução do capítulo anterior, sendo também feito um balanço do trabalho realizado, constituindo respectivamente os capítulos cinco e seis.

5 – Aplicação Prática: Neste capítulo apresenta-se um exemplo de aplicação, onde é construído o modelo RdP do mesmo para depois ser feita a sua tradução para Diagramas *Ladder* e Lista de Instruções tendo dado uso a algumas das regras de tradução propostas no capítulo anterior e fazendo no fim uma simulação para demonstrar a sua funcionalidade.

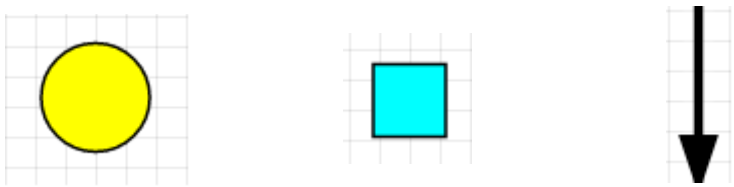
6 – Conclusões: Neste capítulo é feito um balanço dos objectivos cumpridos bem como uma análise dos resultados obtidos ao longo dos vários testes e simulações realizadas.

2 Redes de Petri

2.1 Conceitos Básicos

2.1.1 Lugares, Transições e Arcos

As redes de Petri têm dois tipos de nós, nomeadamente, os lugares e as transições, conforme ilustrado nas Figuras 2. 1a, b e c. Um lugar é regra geral representado por uma elipse ou uma circunferência, enquanto as transições são representadas por uma barra ou quadrado. Lugares e transições são sempre conectados por arcos, normalmente representados por segmentos de recta.



(a) (b) (c)

Figura 2. 1 - Representação de um lugar (a), transição (b) e arco (c)

O número de lugares e transições tem de ser finito e superior a zero. Os arcos ora se ligam de um lugar para uma transição, como se ligam de uma transição para um lugar e nunca de lugar para um lugar ou de uma transição para uma transição. Os arcos podem ainda conter diferentes pesos, representados normalmente com o número do respectivo peso. O peso pode

ainda ser visto como o número total de arcos ligados entre o respectivo lugar e transição ou vice-versa [7].

2.1.2 Marcação

As RdPs, para além de conterem lugares, transições e arcos, contêm ainda um quarto elemento gráfico: as marcas. Estas marcas são, normalmente, representadas por um ponto dentro de um ou mais lugares, podendo ainda ser representadas por um número inteiro, maior que zero, indicando o respectivo número de marcas do lugar, normalmente usado quando o número de marcas de um lugar é demasiado elevado para conter pontos, como ilustrado na Figura 2. 2. Nas *IOPT Tools*, que são uma ferramenta que permite modelar redes de Petri *IOPT*, conforme descrito mais adiante, o número de marcas de um lugar é representado por um número e não por pontos [5].

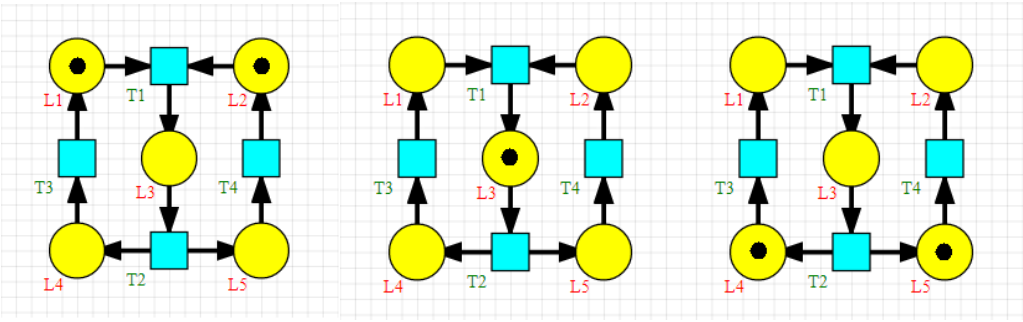
Figura 2. 2 - Duas formas de representar um lugar marcado

Cada lugar pode ainda começar marcado, ou não. Diz-se que um lugar é seguro quando contem no máximo uma marca; e quando todos os lugares de uma rede de Petri são seguros, estamos na presença de uma rede segura [8].

2.1.3 Disparo de Transições

Numa RdP não-autónoma, as transições são os nós que podem disparar em função de um sinal de entrada, e, no caso das *IOPT Tools,* em função de uma guarda de sinais de entrada, ou ainda possíveis eventos de entrada. Para que estas transições possam disparar, é necessário verificar a seguinte condição: todos os lugares ligados à entrada da transição têm de conter pelo menos uma marca ou tantas marcas quanto o peso do arco de entrada (lugar-transição).

Verificada esta condição, diz-se que a transição está apta ao disparo. Aplicando um sinal de entrada à transição, ou no caso das *IOPT Tools* um possível conjunto de sinais de entrada associados à guarda e ocorrência de eventos de entrada, é então retirado uma ou mais marcas de cada um dos lugares ligados à entrada da transição, em função do peso do arco de entrada, e posteriormente adicionado uma ou mais marcas a cada um dos lugares ligados à saída da transição, mais uma vez em função do peso do arco de saída (transição – lugar) [7] [9].



(a) (b) (c)

Figura 2. 3 – RdP com marcação inicial (a), RdP após disparo da transição T1 (b), RdP após disparo da transição T2 (c)

Como se pode verificar pela Figura 2. 3a, a transição T1 é a única que se encontra apta a disparar, assim como na Figura 2. 3b, a transição T2 é a única apta a disparar. Já na Figura 2. 3c, temos duas transições aptas a disparar: T3 e T4. Caso cada uma destas dispare pela sequência prevista, voltaremos à RdP com marcação inicial como indicado na Figura 2. 3a.

2.2 Propriedades das redes de Petri

2.2.1 Redes de Petri Limitadas, Seguras, Vivas ou Bloqueadas

As redes de Petri podem conter várias propriedades comportamentais que as tornam bastantes distintas umas das outras. Algumas dessas propriedades são [8]:

- Redes de Petri Limitadas (*Bounded Petri Nets*);

- Redes de Petri Seguras (*Safe Petri Nets*);

- Redes de Petri Vivas (*Live Petri Nets*);

- Redes de Petri com Bloqueio (*Deadlock*);

- Redes de Petri com Conflitos;

Para que um lugar de uma rede de Petri seja k-limitado, é necessário que o número de marcas desse lugar não exceda um determinado número finito k de marcas. Quando todos os lugares de uma rede de Petri forem limitados, então diz-se que a rede de Petri é k-limitada ou simplesmente limitada.

Como já foi referido anteriormente, uma rede de Petri diz-se segura quando para qualquer sequência de disparo, o número de marcas por lugar nunca ultrapassa uma marca. Neste caso podemos dizer que cada lugar é limitado a k=1, tornando se este um caso particular das redes de Petri limitadas.

Para que uma rede de Petri seja considerada viva, é necessário que para qualquer sequência de disparo, a partir da marcação inicial, nenhuma transição se torne permanentemente inapta a disparar. Por outras palavras, se numa rede de Petri, houver uma transição apta a disparar um número finito de vezes, a rede já não é viva e essa transição ganha o nome de transição quase-viva.

Bloqueio acontece numa rede de Petri, quando se chega a uma situação em que nenhuma transição está apta a disparar.

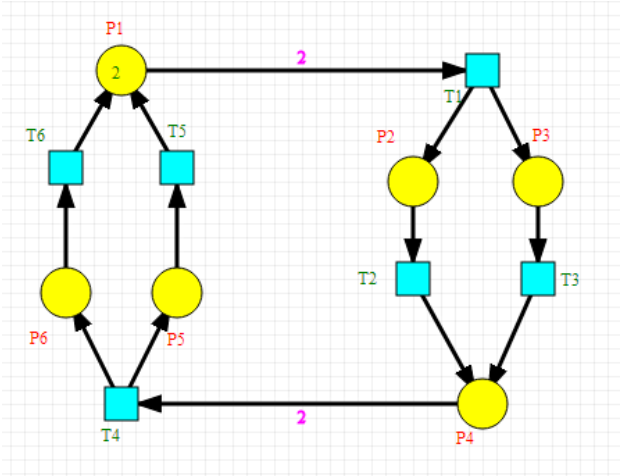
Conflitos acontecem quando duas ou mais transições partilham de um mesmo lugar de entrada com um número de marcas inferior ao somatório do peso dos arcos de entrada entre esse lugar e suas transições. Uma forma de resolver este tipo de conflito é enumerar as transições com diferentes graus de prioridade, como é feito nas redes de Petri *IOPT* utilizando as *IOPT Tools*.

Algumas das formas de verificar algumas destas propriedades será a partir da construção e análise de grafos de marcações e árvores de cobertura, como será apresentado seguidamente.

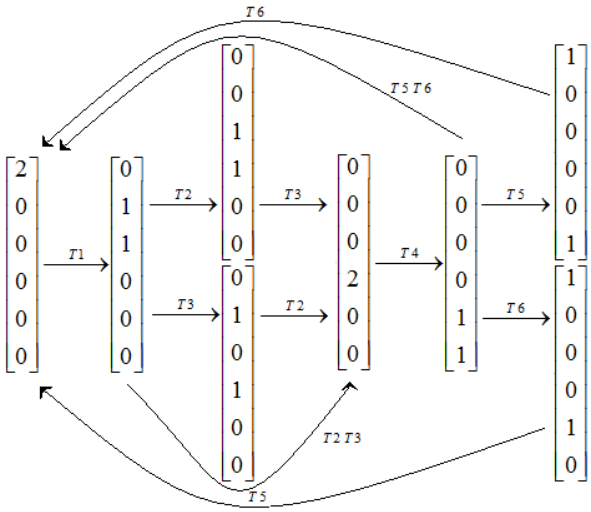
2.2.2 Grafo de Marcações e Árvore de Cobertura

Grafos e Árvores são algumas das possíveis representações de espaço de estados aplicáveis em redes de Petri. Estas podem variar em função da sua acessibilidade ou alcançabilidade e cobertura [8].

Os grafos permitem uma representação individual de cada estado, mas só podem ser usados quando o número de estados e transições (de estados) é finito, isto é na presença de uma rede de Petri limitada. Num grafo cada nó é representado por uma marcação e cada arco contem o disparo de um conjunto de transições. Um exemplo de um grafo de marcações pode ser visto na Figura 2. 4b, construída a partir da rede de Petri da Figura 2. 4a.



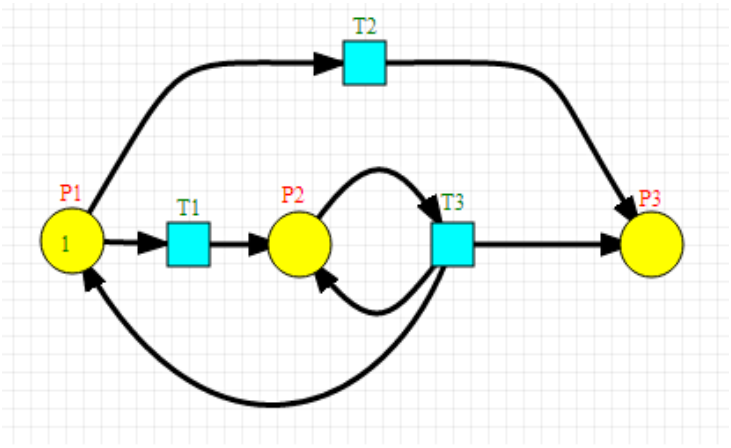
(a)



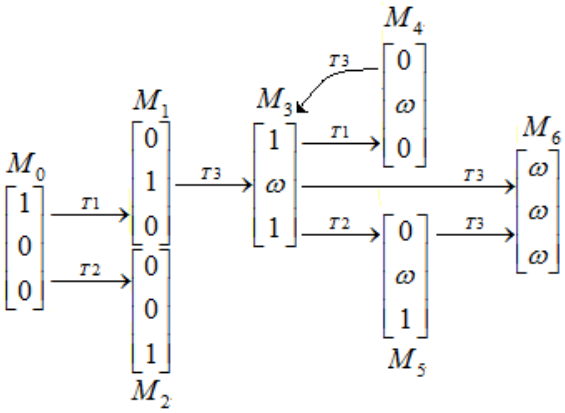
(b)

Figura 2. 4 – Rede de Petri (a) e respectivo grafo de marcações (b)

No caso de uma rede de Petri não ser limitada, é necessário recorrer a uma árvore de cobertura, de forma a conseguir uma representação do espaço de estados finito da mesma. Para se construir esta árvore de cobertura é necessário recorrer a um algoritmo descrito em [8]. O resultado será algo como representado na Figura 2. 5.



(a)



(b)

Figura 2. 5 – Rede de Petri (a) e respectiva árvore de cobertura (b)

Outra forma de verificar as propriedades anteriormente referidas, é fazer uso de uma ferramenta presente nas *IOPT Tools*, para construção do espaço de estados da rede de Petri e geração do grafo de marcações da mesma.

Sabendo a marcação inicial da rede, esta ferramenta permite saber quantos ciclos tem, o número mínimo e máximo de marcas em cada lugar, o número

total de estados e ligações, bem como uma contagem de situações de conflito, situações inválidas ou situações de bloqueio (Figura 2. 6) [5].

É também possível construir o grafo de marcações (Figura 2. 7), bem como dar resposta a certas questões que o utilizador possa ter, tais como, número máximo de marcas por lugar, contagem de conflitos, bloqueios, situações inválidas, entre outras.

Figura 2. 6 – Exemplo de um espaço de estados gerado pelas IOPT Tools

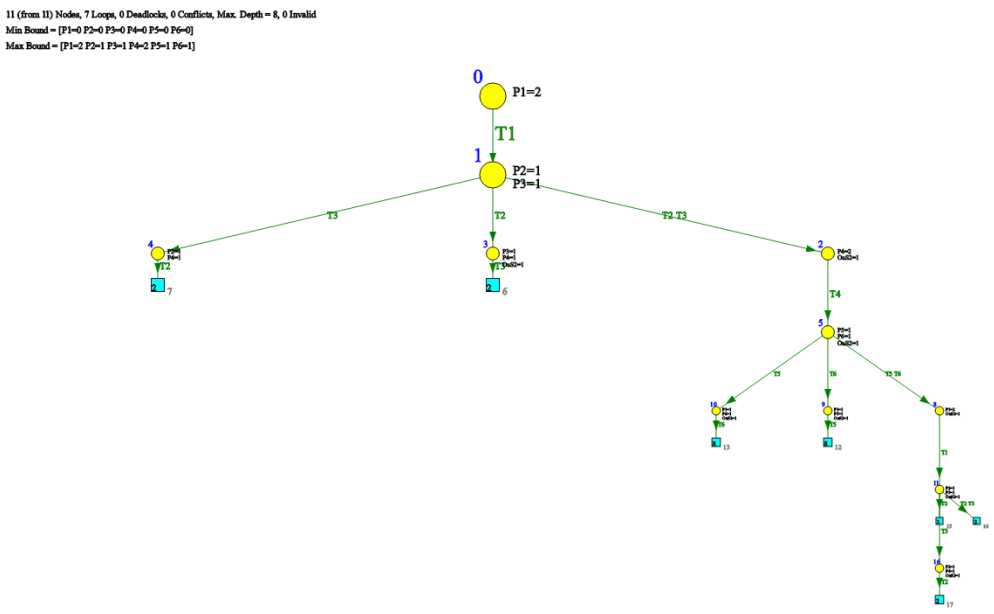


Figura 2. 7 – Grafo de marcações/espaço de estados da figura 2.6.

2.2.3 Classes não autónomas de baixo nível das redes de Petri

Redes de Petri Sincronizadas – São aquelas em que o disparo das transições está dependente de eventos externos, mesmo que a transição esteja apta a disparar, ou seja, só irá disparar quando todos os seus requisitos forem cumpridos (estar apta a disparar, seus eventos externos tiverem ocorrido e sinal de entrada/guarda respeitada) [8][10].

Redes de Petri Temporizadas – O funcionamento deste tipo de RdP está dependente do tempo. Existem dois tipos de RdP Temporizada: L-Temporizada e T-Temporizada. Nas L-Temporizada as transições só ficam aptas a disparar após as marcas do lugar de entrada da transição, se encontrarem no lugar de entrada, uma certa quantidade de tempo (constante ou variável). Já as RdPs T-Temporizadas só fazem a transição de marcas de um lugar para o seguinte, após esperarem determinada quantidade de tempo depois do disparo da transição [8][11].

Redes de Petri Interpretadas – Chamam-se RdPs interpretadas aquelas que podem ser interpretadas como modelos reais ou sistemas de controlo reais. Para que possam ser consideradas RdPs interpretadas, estas têm de respeitar os seguintes pré-requisitos: têm de ser sincronizadas, L-Temporizadas, e ter uma parte de processamento de dados aos quais são associados eventos externos de entrada às transições e eventos internos de saída aos lugares ou transições (sinais de entrada e sinais de saída) [8][12].

Redes de Petri *IOPT* (*Input-Ouput Place-Transition*) – Semelhantes às redes de Petri interpretadas, as redes *IOPT* são também elas sincronizadas e com uma componente de processamento de dados, podendo estes também estar associados a sinais externos de entrada e saída. Transições podem ter associadas eventos de entrada e guardas em função destes sinais de entrada externos. O disparo de transições pode ainda gerar sinais de saída bem como eventos de saída que por sua vez irão afectar sinais de saída. Outra forma de obter sinais de saída é através da actualização da marcação de lugares. Para além da funcionalidade de modelação de redes *IOPT*, as *IOPT Tools*, permitem ainda, geração de espaço de estados, como visto anteriormente, simulação de modelos *IOPT*, editor e resultado de consultas, gerador de código C e VHDL, entre outras funcionalidades como será visto mais adiante, estas tornaram

excelentes candidatas a modelar vários tipos de sistemas, entre estes, sistemas GALS (*Globally-Asynchronous Locally-Synchronous*), podendo considerar vários domínios temporais dentro de um modelo [9][13]. As redes de Petri das *IOPT Tools* têm muitas outras funcionalidades de serão descritas com mais detalhe na secção que se segue.

2.3 Redes de Petri IOPT e IOPT Tools

Para além das propriedades das classes de redes de Petri já referidas, as redes de Petri *IOPT* podem ainda conter outras funcionalidades, nomeadamente as presentes na ferramenta *IOPT-Tools* que farão parte das RdPs alvo do presente trabalho [5]. O editor de redes IOPT, para além de permitir o uso das várias transições, lugares, arcos, sinais e eventos de entrada e saída, todas estas têm propriedades únicas permitindo diversos comportamentos das redes IOPT como será visto de seguida.

Lugares – Para além do identificador de lugar cada lugar pode ainda conter o seu próprio nome, marcação inicial, atribuição de um número máximo de marcas, domínio de tempo para RdP de sistemas Globalmente Assíncronos, Localmente Síncronos (GALS), um campo para possíveis comentários e finalmente um campo para atribuição de acções a determinados sinais de saída em função de certas condições (Figura 2. 8).

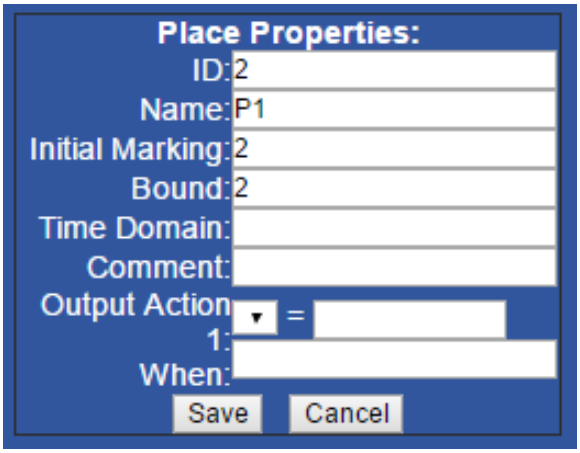


Figura 2. 8 – Propriedades dos Lugares

Transições – Tal como os lugares cada transição têm o seu próprio identificador, nome, domínio de tempo e campo para possíveis comentários. Além disso também tem um campo para atribuição de prioridades para resolução de possíveis situações de conflito entre transições, um campo para definição da guarda de sinais de entrada, um campo para associação de possíveis eventos de entrada e outra para possíveis eventos de saída, bem como um ou mais campos para atribuição de valores a determinados sinais de saída aquando do seu disparo (Figura 2. 9).

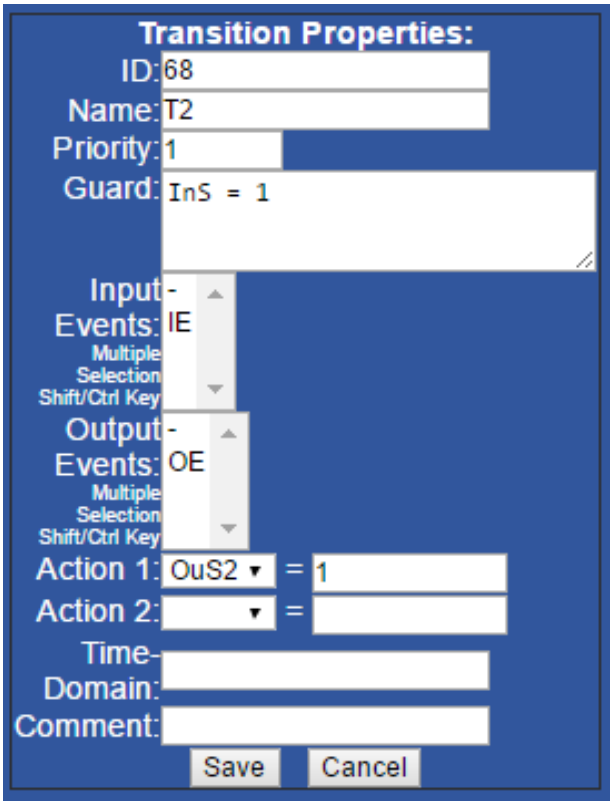


Figura 2. 9 – Propriedades das Transições

Arcos – Tal como os anteriores também este tem o seu próprio identificador, para além disso tem também um campo para atribuição do seu peso, ou seja, o número de marcas que serão produzidas ou destruídas dos lugares a que estão ligados, um campo para atribuição do tipo de arco, podendo este ser normal, de teste ou de canal (Figura 2. 10). O arco de canal não será abordado no presente trabalho no que diz respeito às regras de tradução, mas o arco de teste já o será, por isso será pertinente explicar o seu

funcionamento. Tal como o nome indica, a função desde arco, é fazer um teste à marcação do lugar de entrada à qual este se encontra ligado. Se o lugar estiver marcado a transição dispara sem que esta marca seja consumida, permanecendo este lugar sempre marcado até que esta marca seja consumida por outros meios. Posto isto, este tipo de arco só pode ser atribuído a arcos que tenham como partida um lugar e como destino uma transição e nunca o contrário, como se pode verificar pelo arco de teste a seta do meio da Figura 2. 11.

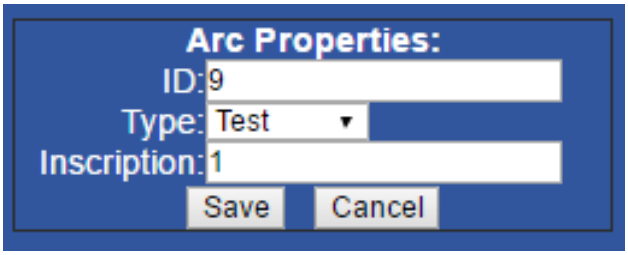


Figura 2. 10 – Propriedades dos Arcos

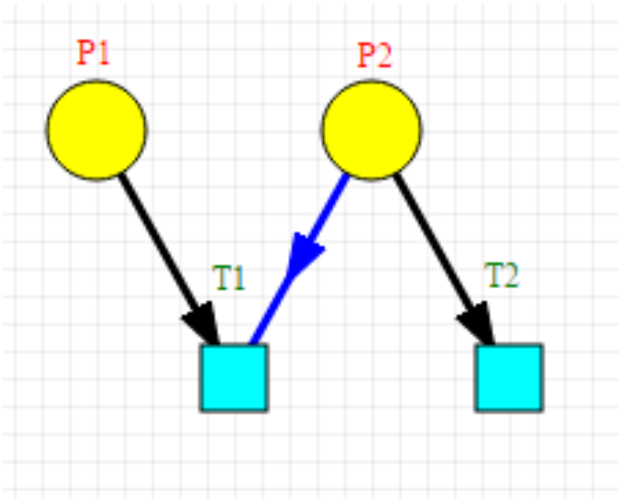


Figura 2. 11 – Exemplo de um Arco de Teste

Sinais – Os sinais, têm um campo para especificar o modo de sinal podendo este ser de entrada ou de saída, têm ainda um campo de identificador/nome, um de tipo podendo este ser booleano ou de intervalo de valores, isto é, valores menores que zero e superiores a um, um campo para especificar o valor com que esse sinal será inicializado, dois campos para

especificar o valor mínimo e valor máximo que este sinal pode ter, e um campo de especificação de um canal físico (pino) ao qual o sinal poderá ser associado (Figura 2. 12).

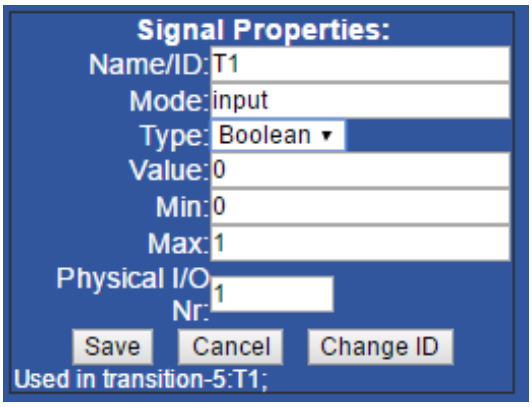


Figura 2. 12 – Propriedades dos Sinais de Entrada e Sinais de Saída

Eventos – Os eventos, tal como os sinais, têm um campo para especificar o modo de evento podendo este ser de entrada ou de saída, bem como um campo de identificador. Têm ainda uma caixa para se especificar se o evento será autónomo ou não, um campo que define de que forma o evento é afectado ou afecta o sinal de entrada ou de saída respectivamente, podendo ser afectado ou afectar sinais de forma ascendente ou descendente, um campo para definição do nível do evento, e um campo para definir qual o sinal de entrada ou de saída associado ao evento (Figura 2. 13).

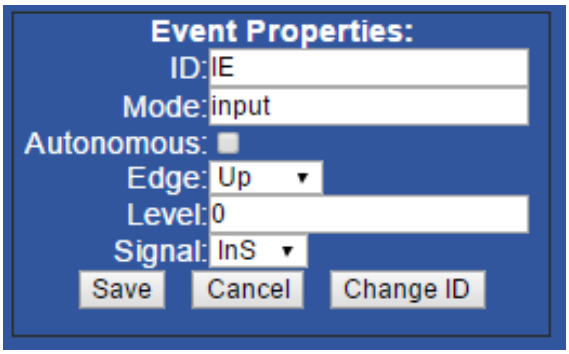


Figura 2. 13 – Propriedades dos Eventos de Entrada e Eventos de Saída

Editor de Expressões – Para finalizar, as *IOPT Tools* têm ainda um editor de expressões, por exemplo para a construção das guardas. Este editor permite incluir nas expressões operadores (tais como, lugares, sinais de entrada, sinais de saída, Tabelas e qualquer número inteiro), operações (aritmética (adições, subtracções, multiplicações, divisões, e resto de divisões), de comparação (de igualdade, maior que, menor que), lógicas (E, Ou, Ou exclusivo)), e sub-expressões (inicio e fim de sub-expressão, negação e inversão), como ilustrado pela Figura 2. 14.

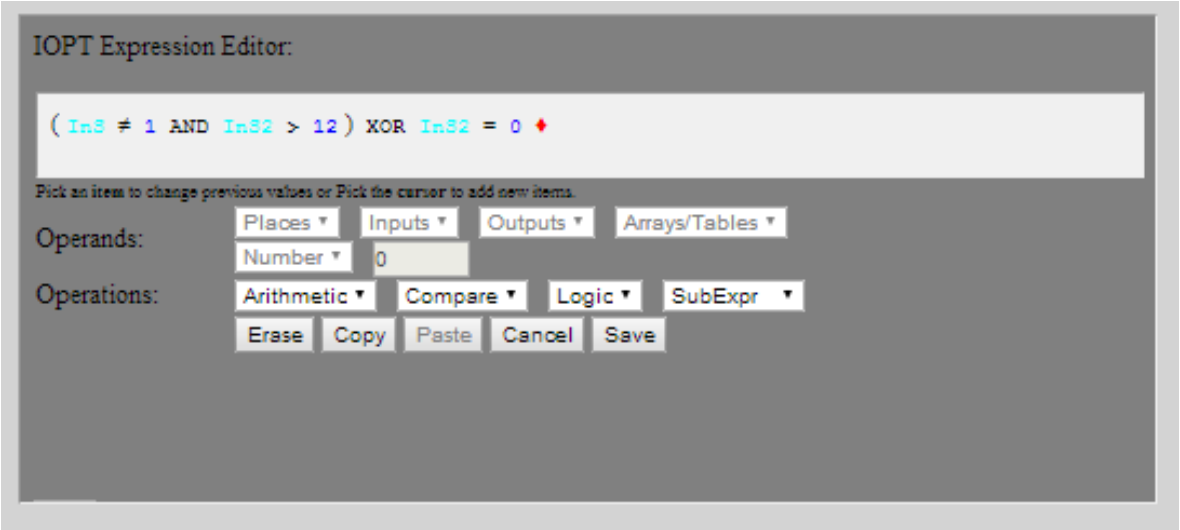


Figura 2. 14 – Editor de Expressões das IOPT Tools

As *IOPT* Tools tem ainda a capacidade de simular modelos de redes de Petri *IOPT*, gerar grafos de marcação como visto anteriormente, mostrar resultado de consultas especificadas pelo utilizador num editor de consultas das *IOPT Tools*, e através de modelos criados das redes *IOPT*, gerar código, nas seguintes linguagens de programação: *VHDL*, *C*, Java Script (apesar de esta não estar disponível para o utilizador). As redes de Petri *IOPT* são guardadas no formato *PNML* da qual são geradas as restantes linguagens, da mesma forma que será gerada por php, a linguagem da norma internacional *IEC61131*, Lista de Instruções no presente trabalho, conforme será descrito no capítulo 3 do presente trabalho. O código respeita um algoritmo dividido em três partes repetidas num ciclo [14]:

1 – Leitura de entradas;

2 – Processamento interno;

3 – Actualização de saídas;

A geração das regras de tradução das *IOPT* para Diagramas *Ladder*, irão respeitar este algoritmo, sendo analisado com mais detalhe o interior de cada uma destas partes no capítulo 4 do presente trabalho.

3 Norma IEC61131

3.1 Introdução

A comissão internacional de electrotécnica (CIE/IEC) é uma organização internacional não governamental e sem fins lucrativos, que prepara e publica normas internacionais para todo o tipo de tecnologia eléctrica, electrónica entre outras relacionadas. Existe uma grande quantidade de normas da IEC abrangidas entre os números 60000 e 79999, entre as quais a *IEC61131* que foi alvo de estudo no presente trabalho.

A norma *IEC61131* é uma das várias normas criadas pela comissão internacional de electrotécnica que visa disponibilizar ao público geral, um conjunto de regras, directrizes e características a cumprir no que diz respeito ao uso de CLPs e equipamento relacionado. A norma é actualmente constituída por nove partes, havendo ainda uma décima parte actualmente em desenvolvimento [15] [16]:

IEC61131-1: General information (Informação geral) – A primeira parte refere-se à norma IEC61131 como sendo aplicável aos CLPs e seus componentes, e fornece as definições e termos usados na norma, bem como a identificação e principais características funcionais dos sistemas de controlo programáveis [17];

IEC61131-2: Equipment requirements and tests (Requisitos de equipamento e testes) – Aqui estabelecem-se definições e identificação das principais características necessárias para a selecção e aplicação de CLPs e periféricos

associados, bem como especificar requisitos mínimos das características funcionais, eléctricas, mecânicas, ambientais e de construção, condições de serviço, segurança, compatibilidade electromagnética (CEM), programação do utilizador e os testes aplicáveis aos CLPs e periféricos associados [18];

IEC61131-3: Programming languages (Linguagens de programação) – Contem uma lista das várias linguagens de programação usadas pela norma *IEC61131* aplicadas nos CLPs, bem como suas respectivas semânticas e sintaxes [19];

IEC61131-4: User guidelines (Directrizes do utilizador) – O objectivo desta norma é a introdução da norma IEC61131 aos utilizadores de CLPs, de forma a guia-los na selecção e especificação do seu equipamento PLC [20];

IEC61131-5: Communications (Comunicação) – Esta parte especifica os aspectos de comunicação dos PCs (Controladores Programáveis) e CLPs, para com os outros dispositivos, seja para disponibilização de serviços ou requisitar serviços [21];

IEC61131-6: Functional safety (Segurança funcional) – Refere-se a questões de segurança que tanto o software como o hardware de CLPs tem de respeitar [22];

IEC61131-7: Fuzzy control programming (Programação de controlo difuso) – Permite fazer uso de sistemas de controlo baseados em lógica difusa a partir de sistemas analíticos fazendo uso de formalismos matemáticos [23];

IEC61131-8: Guidelines for application and implementation of programming languages (Directrizes para a aplicação e implementação de linguagens de programação) – Trata-se de um complemento ao "*IEC61131-4*: Directrizes de utilizador" actualizando algumas directrizes referentes ao uso das linguagens de programação contidas no "*IEC61131-3*" [24];

IEC61131-9: Single-drop digital communication interface for small sensors and actuators (SDCI) (Interface de comunicação digital "*Single Drop*" para pequenos sensores e actuadores) – Esta parte especifica os serviços de comunicação e protocolos da tecnologia de comunicação SDCI (*Single-Drop Digital Communication Interface*) tanto para sensores como actuadores, bem como requisitos de teste EMC (*Electromagnetic Compatibility*) [25];

3.1.1 IEC61131-3

A terceira parte da norma *IEC61131*, sendo ela referente às linguagens de programação usadas para a programação de CLPs, será a mais focada neste trabalho visto ser do interesse do mesmo achar formas de traduzir modelos de redes de Petri para uma destas linguagens. É actualmente constituída por seis linguagens de programação das quais quatro gráficas e duas de texto, sendo estas [19]:

Diagramas *Ladder* (*Ladder Diagram*) – Linguagem gráfica;

Diagramas de Blocos de Funções (*Fuction Block Diagram*) – Linguagem gráfica;

Texto Estruturado (*Structured Text*) – Linguagem de texto;

Lista de Instruções (*Instruction List*) – Linguagem de texto;

Gráficos de Funções Sequenciais (*Sequential Function Chart*) – Linguagem Gráfica;

Gráficos de Funções Continuas (*Continuous Funtion Chart*) – Linguagem Gráfica;

No presente trabalho só serão abordadas regras de tradução intermédia entre redes de Petri *IOPT* das *IOPT Tools* e a linguagem gráfica de Diagramas *Ladder* para posteriormente passar de Diagramas *Ladder* para a linguagem de texto de Lista de Instruções visto ser esta última a que será reconhecida pelos CLPs alvo deste trabalho.

3.1.1.1 Diagramas Ladder – Ladder Diagram (LD)

A lógica de Diagramas *Ladder* tem sido fortemente usada na programação de CLPs, onde o controlo sequencial do processo de uma operação é fundamental. São úteis e utilizados não só para simples mas também para sistemas de controlo mais complexos. Diagramas *Ladder* são constituídos por diversas redes que são energizadas de forma sequencial antes de se passar à

próxima rede. Cada rede pode ser constituída por diversas instruções, algumas das mais usadas são [26]:

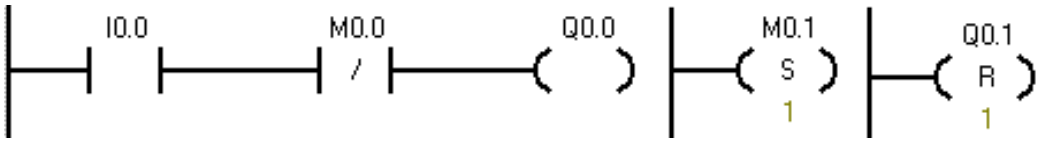
Contacto normalmente aberto (sensor) – Esta instrução fecha ligação (liga canal) se for energizada (Figura 3. 1a). Pode ser energizada por dados do tipo I (sinais de entrada), tipo Q (sinais de saída), ou tipo M (saídas de redes internas).

Contacto normalmente fechado (sensor) – Esta instrução fecha ligação (liga canal) se não for energizada (Figura 3. 1b). Também pode ser energizada por dados do tipo I (sinais de entrada), tipo Q (sinais de saída), ou tipo M (saídas de redes internas).

Bobina de Saída (Actuador) – Esta instrução é continuamente energizada enquanto a sua entrada permanecer com ligação fechada (canal ligado) (Figura 3. 1c). Pode energizar dados do tipo Q (sinais de saída), do tipo M (saída de rede interna), entre outros.

Bobina de Saída *SET* (Actuador) – Esta instrução energiza (N vezes) uma saída quando a sua entrada tiver ligação fechada (canal ligado) (Figura 3. 1d). Pode energizar dados do tipo Q (sinais de saída) e do tipo M (saída de rede interna), entre outros.

Bobina de Saída *RESET* (Actuador) – Esta instrução desenergiza (N vezes) uma saída quando a sua entrada tiver ligação fechada (canal ligado) (Figura 3. 1e). Pode desenergizar dados do tipo Q (sinais de saída) e do tipo M (saída de rede interna), entre outros.



(a) (b) (c) (d) (e)

Figura 3. 1 - Instruções de Diagrama Ladder: Contacto normalmente Aberto (a), Conctacto normalmente Fechado (b), Bobina de Saída (c), Bobina SET (d) e Bobina RESET (e)

Existem ainda muitas outras instruções, tais como, contadores, temporizadores, comparadores, relógios, conversores, interruptores, operadores lógicos, movedores, somadores, subtratores, multiplicadores,

divisores, entre muitos outros, sendo alguns destes usados no presente trabalho.

3.1.1.2 Lista de Instruções – Instruction List (IL)

Sendo uma das seis linguagens de programação da norma IEC61131 e uma das duas linguagens texto, a Lista de Instruções é uma linguagem de baixo nível, semelhante ao *Assembly*. Tal como nos Diagramas *Ladder,* cada subrotina é representada por uma rede independente. Existem várias variantes de Lista de Instruções, algumas preparadas para suportar os periféricos específicos de determinadas marcas. É exemplo disto a Lista de Instruções da Siemens também conhecida por "*Statement List*" ou "*STL*" em inglês, e "*Anweisungs-Liste*" ou "*AWL*" em alemão, sendo este último o formato do ficheiro gerado pelo *software S7-200 Micro/Win* usado no presente trabalho [26].

Cada linha de instrução é normalmente constituída por dois elementos: um operador e um operando, de entre os vários operadores os mais usados são:

Inicio de uma sub-expressão – Representado por "LD" ou "LDN" no caso do operador ser negado;

E – Representado por um "A";

Ou – Representado por um "O";

E Negado – Representado por um "AN";

Ou Negado – Representado por um "ON";

Resultado de uma função – Representado por um "=".

SET – Representado por um "S" seguido de um valor N.

RESET – Representado por um "R" seguido de um valor N.

Já os operandos podem variar entre os vários sinais de entrada (representados por "I"), sinais de saída (representados por "Q"), ou resultados de subrotinas (representados por "M"). É exemplo de uma subrotina de IL a representada pela Figura 3. 2).

O Software *S7-200 Micro/Win* usado no presente trabalho, não só faz uso das linguagens de programação Diagramas *Ladder* e Lista de Instruções, como

também é capaz de fazer tradução directa entre estas duas linguagens como se poderá verificar entre as Figura 3. 2 e Figura 3. 3.

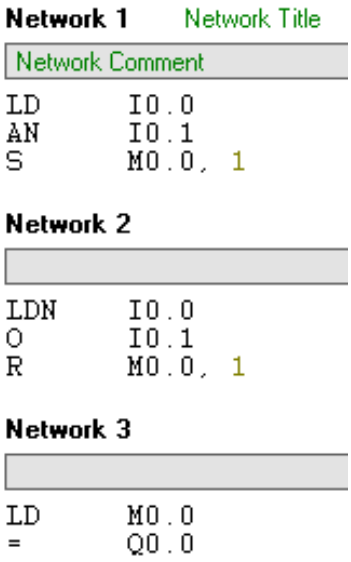


Figura 3. 2 – Exemplo de Lista de Instruções

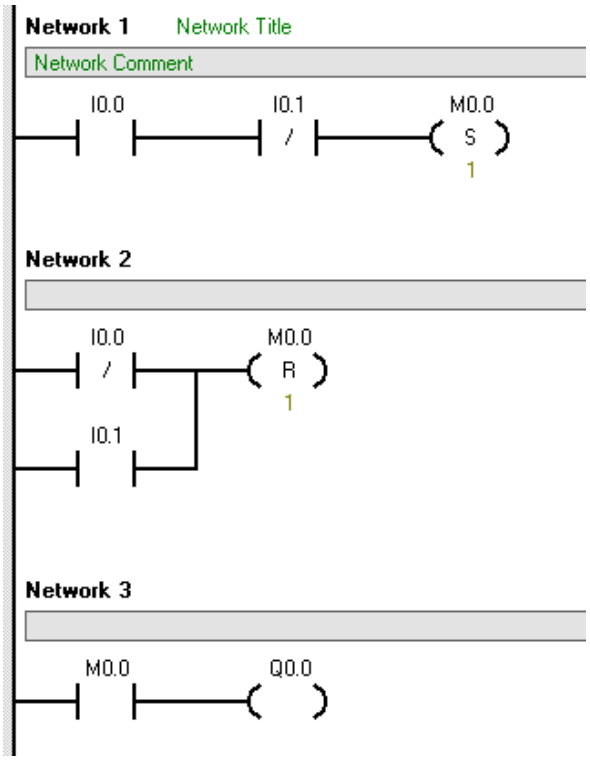


Figura 3. 3 – Diagrama Ladder equivalente à lista de Instruções da figura 3.2

4 Regras de Tradução

Neste capítulo serão abordadas as regras de tradução necessárias para se realizar a passagem de determinados modelos de redes de Petri da classe *IOPT*, para as linguagens de Diagramas *Ladder* e Lista de Instruções das norma *IEC61131*. Podendo existir vários tipos de redes de Petri, o primeiro passo é identificar que tipo de rede de Petri se vai traduzir. Como já referido no capítulo 2, podem existir redes de Petri com diferentes propriedades tornando-as muito distintas umas das outras, entre elas as mais focadas pelo presente trabalho são, as redes de Petri seguras e as redes de Petri limitadas, o que irá obrigar a diferentes tipos de abordagem na criação destas regras de tradução. Para se verificar se estas redes de Petri são seguras ou limitadas, usou-se a ferramenta de geração de espaço de estados presente nas *IOPT Tools* como já referido no capítulo 2. Se o número máximo de marcas por lugar for um, está-se perante uma rede de Petri segura, caso contrário será uma rede de Petri limitada. Será importante definir diferentes regras de tradução para diferentes tipos de redes de Petri por questões de alocação de memória e velocidades de processamento [14].

4.1 Redes de Petri IOPT Seguras

Após identificada uma rede de Petri como sendo segura, o próximo passo será identificar algumas regras de tradução. Como a sequência de processamento/alimentação dos Diagramas *Ladder* é feito de cima para baixo e

da esquerda para a direita, será importante construir o código *Ladder* com a seguinte estrutura, semelhante ao algoritmo proposto em [14]:

- Inicialização da marcação inicial;
- Leitura das entradas;
- Disparo das transições (Condições de disparo e resultado do disparo);
- Actualização das saídas.

Pegando num exemplo em concreto (Figura 4. 1), será analisado com mais detalhe cada um destes passos da estrutura e por consequentes as regras da sua tradução.

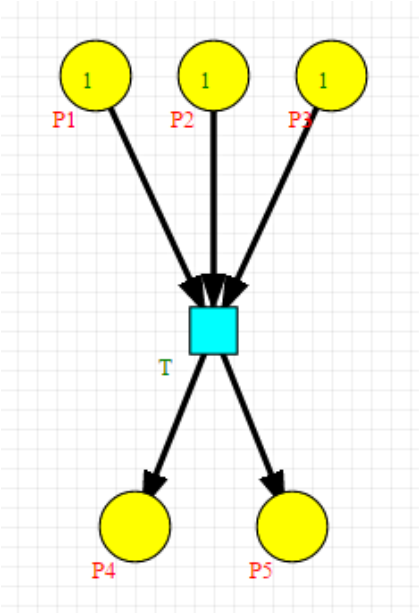


Figura 4. 1 - Exemplo de uma rede de Petri

4.1.1 Regra 1 (Inicialização da marcação inicial)

O primeiro passo de tradução de qualquer rede de Petri, passa por verificar que lugares estão marcados inicialmente, para se poder proceder à sua inicialização. Para inicializar os lugares marcados basta ver que lugares estão marcados e activar seus elementos de memória (registo). Para isso aplica-se uma bobina *SET* por cada um dos lugar marcado. Visto que a inicialização dos

elementos de memória (registos) dos lugares marcados só deverá ocorrer uma vez, uma lógica adicional é aplicada para garantir que tal aconteça, como indicado na Figura 4. 2. Outra forma de garantir que a inicialização da marcação inicial só ocorre uma vez, seria dar uso a um apontador inerente do *S7-200 Micro/Win.* Este apontador é o *SM* e tem a propriedade de conter o valor um (1) durante o primeiro ciclo de relógio (*clock)* do programa.

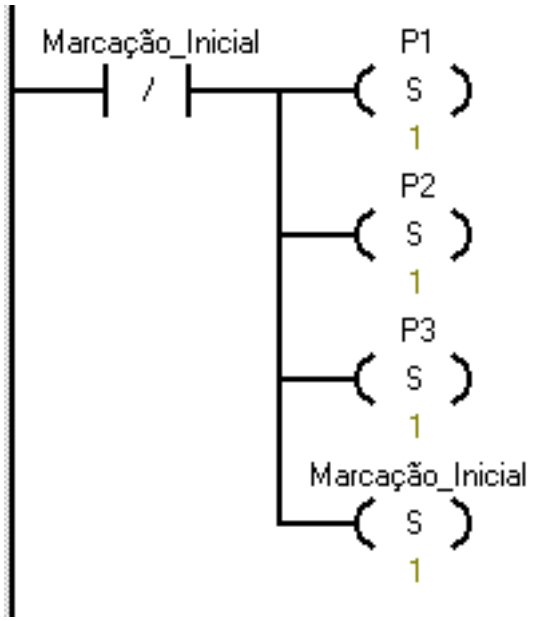


Figura 4. 2 - Inicialização da marcação inicial

4.1.2 Regra 2 (Leitura das entradas)

Para que possa ocorrer o disparo de uma transição é primeiro necessário verificar as entradas. Nas redes de Petri *IOPT* existem dois tipos de entradas: sinais de entrada e eventos de entrada. Será necessário verificar se a guarda das entradas de cada transição é respeitada, bem como os eventos de entrada. A tradução de cada uma será realizada de forma distinta como se poderá ver de seguida.

4.1.2.1 Regra 2.1 (Leitura de Guardas)

Podendo uma guarda ser constituída por um conjunto sequencial de operadores lógicos, *AND*s, *OR*s e/ou *XOR*s será necessário construir os Diagramas *Ladder* de forma a respeitar esta sequência. Estando perante um operador *AND* (por exemplo, $A \cdot \overline{B}$) basta colocar os operandos em serie como indicado na Figura 4. 3. No caso do operador ser um *OR* (por exemplo, $A + B$), colocam-se os operandos em paralelo como indicado na Figura 4. 4. Por fim, temos o operador *XOR* (por exemplo, $B \oplus C$) traduzido como indicado na Figura 4. 5. Devido à natureza do XOR, poderão surgir situações complexas de tradução (por exemplo $(A \cdot \overline{B}) \oplus (A + C)$), originando um uso excessivo de recurso que de outra forma poderão ser evitados. Para evitar estas situações, uma forma de construir o XOR em Diagramas *Ladder*, será a partir do isolamento individual de cada uma das duas entradas do *XOR*, ou seja, achar o resultado da sequência de *ANDs*/*ORs*, e usar cada um desses resultados como as entradas do XOR (como indicado na Figura 4. 6). Usando a guarda de disparo da transição T da Figura 4. 9, ($A \cdot \overline{B}$), iremos obter o Diagrama *Ladder* descrito na Figura 4. 3.

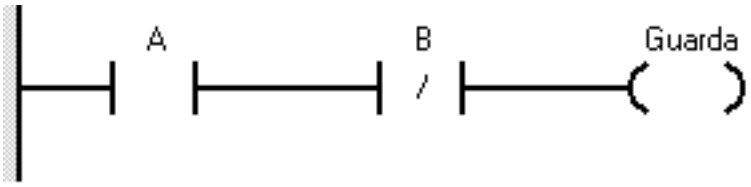


Figura 4. 3 - Entradas da transição T da Figura 4.7 ($A \cdot \overline{B}$)

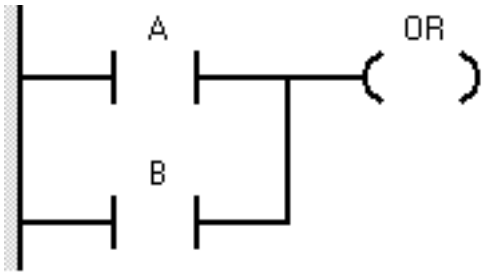


Figura 4. 4 - $A + B$ em *Ladder*

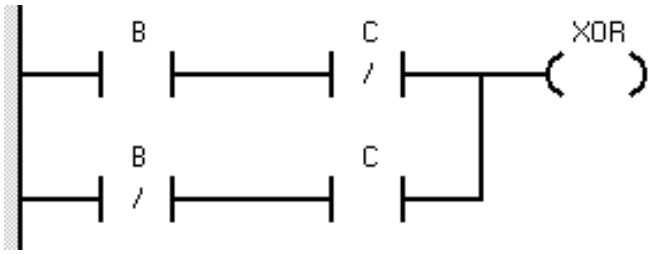


Figura 4. 5 - $B \oplus C$ **em Ladder**

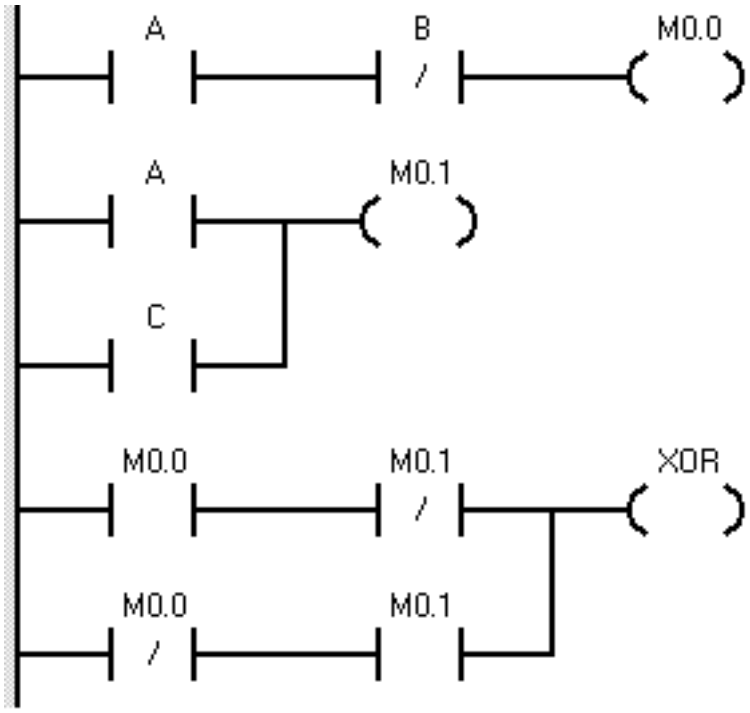


Figura 4. 6 - $(A \cdot \overline{B}) \oplus (A + C)$ **em Ladder**

Até agora tem-se assumido que todos os sinais de entrada são booleanos, mas estes podem ainda ser inteiros não booleanos. Nesse caso, será necessário recorrer a uma variante desta regra de forma a suportar guardas com tais sinais, fazendo sentindo a inclusão dos operadores de comparação. A solução envolverá usar um conjunto de vários bits de entrada físicos de forma a poder ser feita tal comparação. Reservando oito bits de sinais de entrada (por exemplo de I1.0 a I1.7), ou seja, um byte (I1), será possível suportar um valor entre cento e vinte e oito negativos e cento e vinte e sete (-128 a 127), mas como as redes

IOPT não suportam sinais com valores negativos, o intervalo de valores suportados estará entre os zero e cento e vinte e sete (0 a 127). Por exemplo, se tivermos a guarda: G > 123, o conjunto mínimo de sinais esperados que dispare a guarda será I1.0 = 0, I1.1 = 0, I1.2 = 1, I1.3 = 1, I1.4 = 1, I1.5 = 1, I1.6 = 1, I1.7 = 0, visto 01111100 ser 124 em binário, como se poderá verificar pela Figura 4. 7. Com esta regra só será suportado um sinal de entrada deste tipo, podendo haver várias guardas deste tipo desde de contenham o mesmo valor de sinal, deixando assim os sinais de I0.0 a I0.7 reservado exclusivamente a sinais booleanos, caso haja tais sinais.

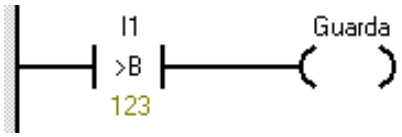


Figura 4. 7 - Guarda com sinal de entrada G > 123

4.1.2.2 Regra 2.2 (Leitura de Eventos de Entrada)

Para traduzir o disparo de possíveis eventos de entrada, é necessário verificar se a condição para o seu disparo ocorre. A cada evento de entrada está associado um sinal de entrada, e a condição para o seu disparo pode ser uma de duas situações possíveis: evolução ascendente (de 0 para 1); ou descendente (de 1 para 0), deste sinal de entrada associado e só durante o passo da transição. Para respeitar estas condições é necessário verificar o estado anterior do sinal de entrada associado. Por exemplo, com um evento de entrada de evolução descendente do sinal de entrada associado (B), irá se obter a tradução em Diagramas *Ladder*, como indicado na Figura 4. 8. No fim, é necessário actualizar o estado anterior com o registo do sinal de entrada associado.

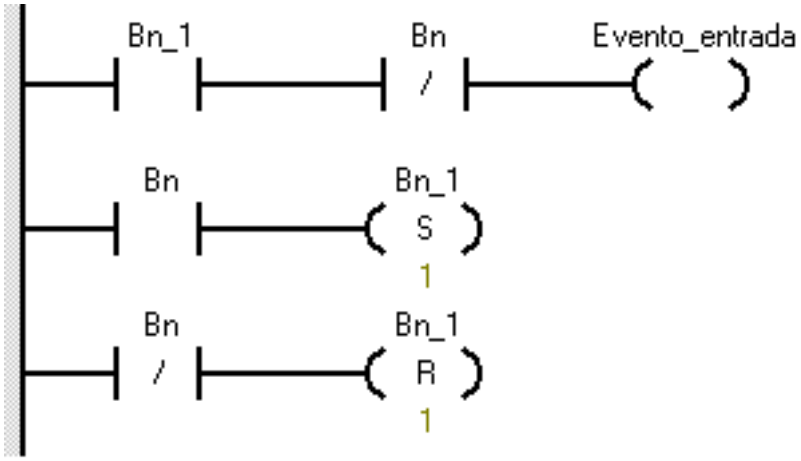


Figura 4. 8 - Evento de entrada de evolução descendente do sinal B

4.1.3 Regra 3 (Disparo da transição)

Para a execução do disparo das transições foi adoptado a separação em duas partes: verificação das condições necessárias para o seu disparo; e só depois ver o resultado do seu disparo. Tendo esta regra sido baseada nas regras de [27], mas tendo esta sido adaptada para as redes de Petri *IOPT*, visto as *IOPT Tools* terem a capacidade de saber em antemão o tipo de rede de Petri de que se está perante (seguras ou limitadas), e visto estas já estarem preparadas para resolver situações de conflito entre transições dando uso ao campo das prioridades de disparo das transições, como será visto de seguida. Para exemplificar estas duas partes da regra , pode ser visto que a rede da Figura 4. 1 foi desdobrada, respectivamente, na Figura 4. 9 e na Figura 4. 13.

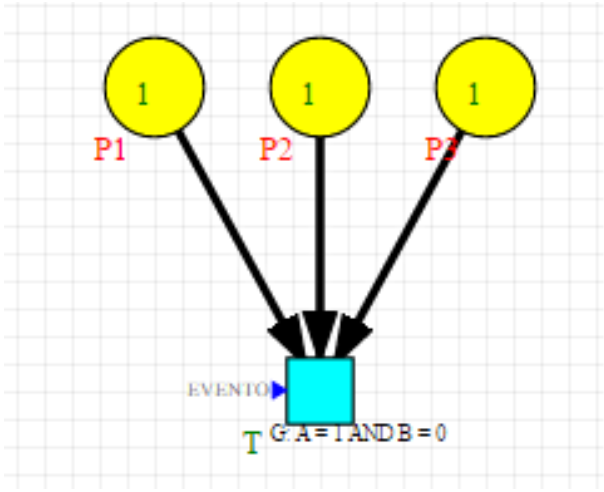


Figura 4. 9 - Condições de disparo da transição T

4.1.3.1 Regra 3.1 (Condições para o disparo das transições)

Para que possa ocorrer o disparo de uma transição, é necessário verificar três condições: lugares de entrada estarem marcados; ocorrência de um possível conjunto de eventos de entradas (eventos de entrada colocados em série no caso de se terem vários); e guarda de entradas respeitada. Após estas três condições serem verificadas, aplica-se o disparo da transição com uma bobina *SET* no registo da transição e são retiradas as marcas dos registos dos lugares de entrada usando uma bobina *RESET* em cada desses registos dos lugares de entrada, como se pode verificar na Figura 4. 10. Podendo cada transição ter determinado nível de prioridade, esta regra deverá ser aplicada de forma sequencial e ordenada de forma crescente em função do nível de prioridade de cada transição.

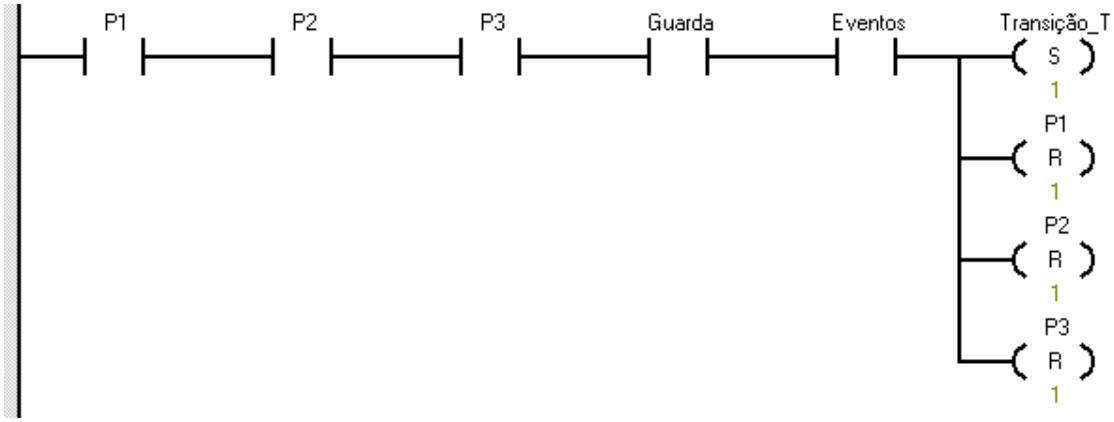


Figura 4. 10 - Resultado da aplicação da Regra 3.1

4.1.3.1.1 Regra 3.1.1 (Arcos de Teste)

Existe ainda uma variante da regra 3.1 que deve ser aplicada no caso de algum(ns) dos lugares de entrada da transição ter arco de teste em vez de arco normal. Têm de ser verificados todos os arcos para esta possibilidade e caso se comprove a sua existência, a regra 3.1 deverá sofrer uma pequena alteração. Visto que um arco de teste deverá manter a marca no lugar de entrada (P2, representado na Figura 4. 11) após o disparo da transição (T1), a não ser que esta ocorra no mesmo passo de uma transição (T2) com um arco normal a vir

deste mesmo lugar de entrada (P2), então a alteração será criar uma cópia do registo deste lugar em comum (lugar P2) e usar esta cópia (P2c) como uma das três condições da aplicação da regra 3. Por fim, actualiza-se no início de cada passo o estado do registo deste lugar (P2c) através do copiado, que no fundo não passa de criar uma nova cópia do registo deste lugar, como se poderá verificar na Figura 4. 12.

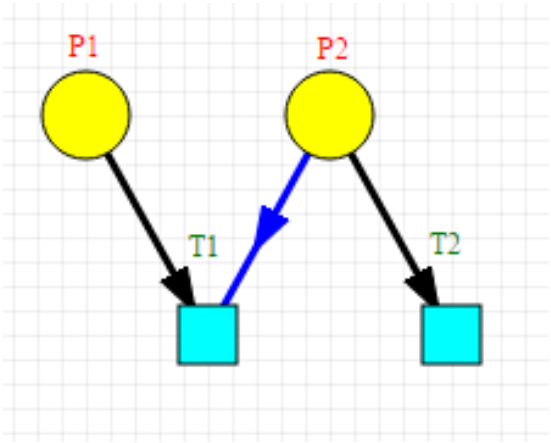


Figura 4. 11 - Exemplo do disparo de uma transição com arco de teste

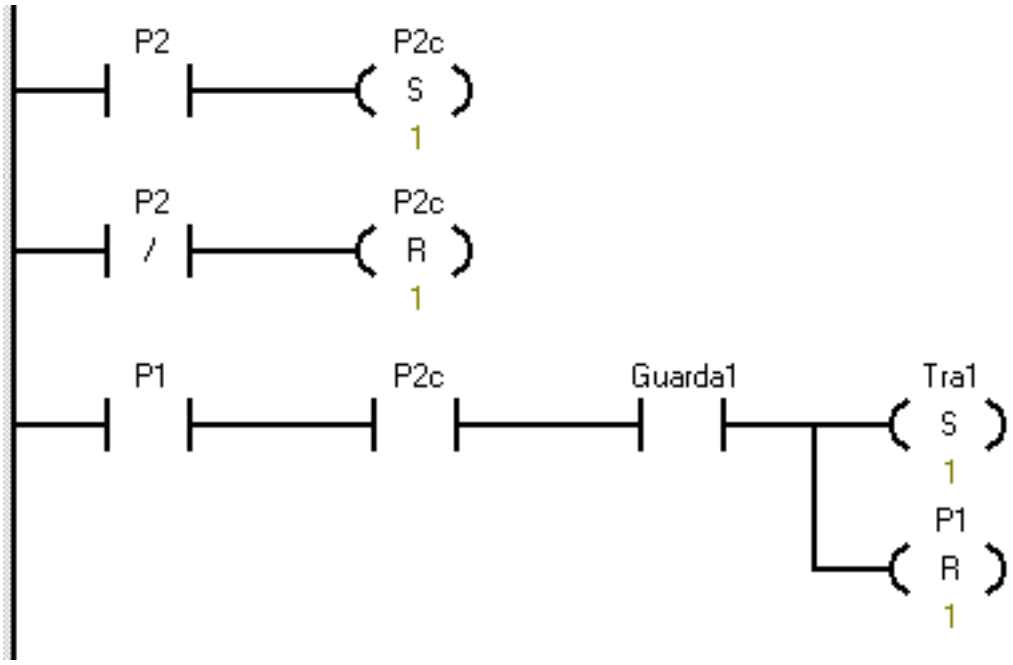


Figura 4. 12 - Aplicação da regra 3.1.1 com um arco de teste

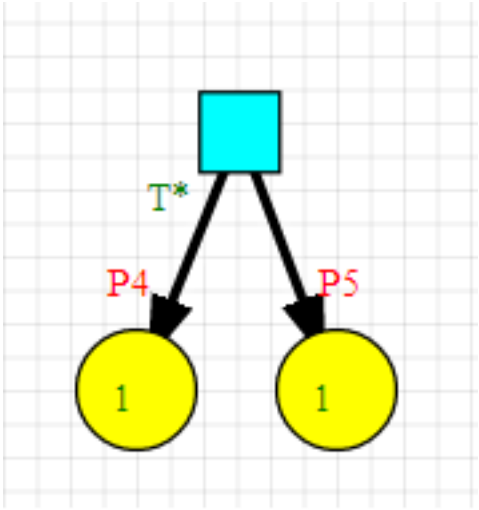


Figura 4. 13 - Resultado do disparo da transição T

4.1.3.2 Regra 3.2 (Resultado do disparo das transições)

Uma vez disparada uma transição é necessário activar (marcar) os lugares de saída da transição, proceder ao disparo dos eventos de saídas associados, e possíveis saídas associadas à transição. Para isso basta aplicar uma bobina *SET* a cada um dos registos desses lugares e a cada um dos registos destes eventos de saída, aplicar uma bobina *SET* ou *RESET* às saídas em função de possíveis condicionantes de lugares, entradas ou saídas, e finalmente desactivar essa mesma transição aplicando uma bobina *RESET* ao registo dessa transição, como indicado na Figura 4. 14.

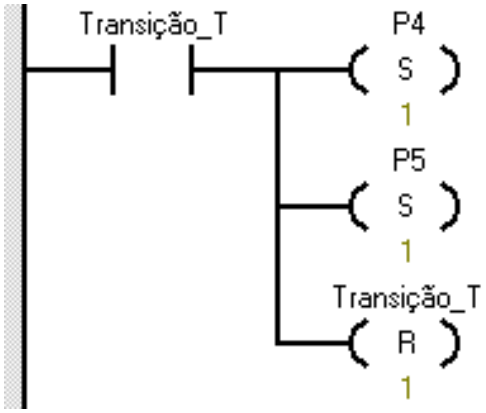


Figura 4. 14 - Resultado da aplicação da Regra 3.2

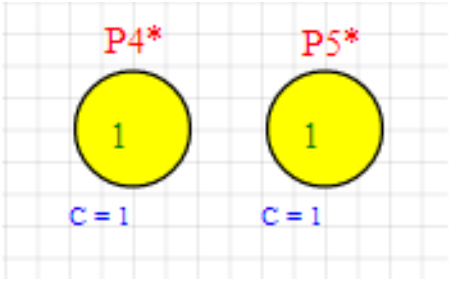


Figura 4. 15 - Actualização das saídas

4.1.4 Regra 4 (Actualização das saídas)

Após activos os elementos de memória (registos) dos lugares (resultado do disparo das transições), é necessário actualizar os sinais das saída desses lugares. Para tal, é necessário em primeiro lugar verificar com que valor a saída é inicializada. Caso seja inicializada a zero, basta activar a saída em função da activação dos registos dos lugares onde essa saída é colocada a um; Caso seja inicializada a um, deve-se activar a saída quando os resgistos dos lugares que a colocam a zero não estiverem activos. Exemplificando, a partir da Figura 4. 15, poder-se-á verificar que tanto o lugar *P4* como o lugar *P5,* têm como saída, a mesma saída *(C=1)*, esquecendo o facto, de neste caso, estes lugares serem activos ao mesmo tempo, não se pode assumir que tal aconteça, logo devem ambos ser condição necessária e suficiente para a activação da saída *C=1*, como se poderá verificar na Figura 4. 16. Isto é, se existirem outros lugares que coloquem *C=1*, os registos desses também deverão ser adicionados como *OR*s à lista de condição de activação da saída *C*, neste caso C terá sido inicializado a zero.

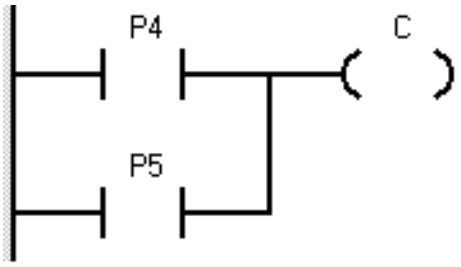


Figura 4. 16 - Resultado da aplicação da Regra 4

No caso de uma saída ser inicializada a um, usando o exemplo da Figura 4. 17, e assumindo que ambos os lugares são activos ao mesmo tempo em determinado momento da execução da rede, o resultado esperado, seguindo o critério referido anteriormente, resultará da tradução em Diagramas *Ladder* indicada na Figura 4. 18.

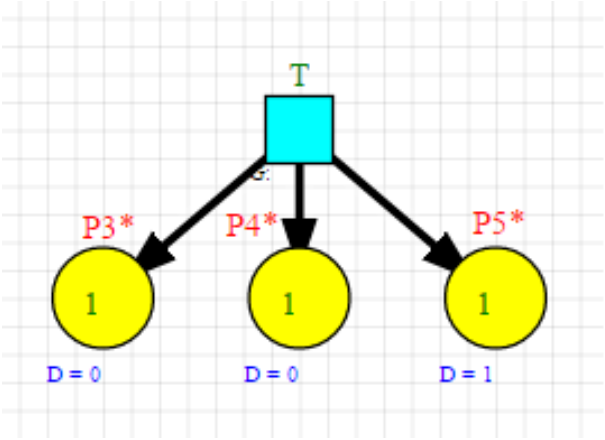


Figura 4. 17 - Actualização das saídas

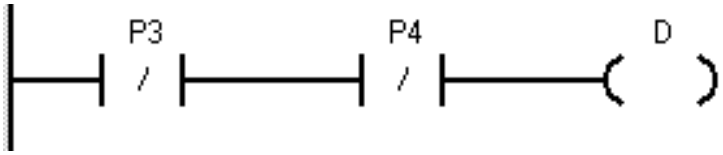


Figura 4. 18 - Resultado da aplicação da Regra 4

Como se pode verificar pelo exemplo anterior, está-se perante uma situação de conflito de saídas, neste caso a saída D não pode ser um e zero ao mesmo tempo, mas segundo o resultado da tradução para Diagramas *Ladder*, só será atribuído um valor à saída D: o valor contrário ao qual foi inicializado, que neste caso será zero, visto ter sido inicializado a um.

Tal como os sinais de entrada, os sinais de saída podem também eles ser inteiros não booleanos. Tendo já referido a regra que traduz este tipo de sinais de entrada, os sinais de saída serão tratados de uma forma semelhante. Quando na presença de um sinal de saída não booleano, reserva-se novamente oito bits de saída, por exemplo de Q1.0 a Q1.7 e realiza-se a respectiva conversão

decimal – binária (por exemplo do valor 56 para 00111000 da Figura 4. 19) após activo o respectivo lugar, será obtido Q1.3 = 1, Q1.4 = 1 e Q1.5 = 1, como se poderá verificar pela Figura 4. 20. Caso haja vários lugares a activar o mesmo sinal de saída inteiro (Q1) com diferentes valores em cada lugar, poderão surgir situações de conflito em que o valor do sinal de saída poderá ser incorrecto, como ilustrado na Figura 4. 21. Para resolver este problema será necessário dar prioridade ao lugar com o sinal de saída de valor mais elevado, especificando que os sinais de saída de valores menos elevados só podem ser activos, no caso dos lugares com sinais de saída de valores mais elevados permanecerem inactivos (usando um contacto normalmente fechado em série com a condição inicial), como ilustrado na Figura 4. 22.

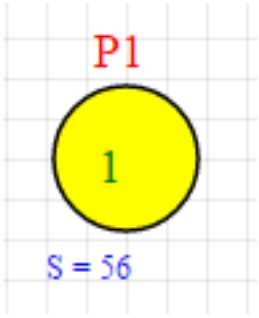


Figura 4. 19 – Lugar com sinal de saída S=56

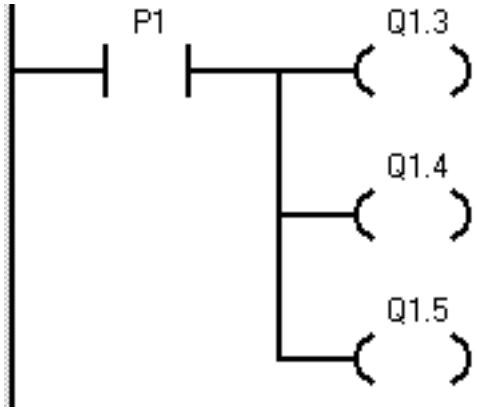


Figura 4. 20 - Sinal de saída S = 56

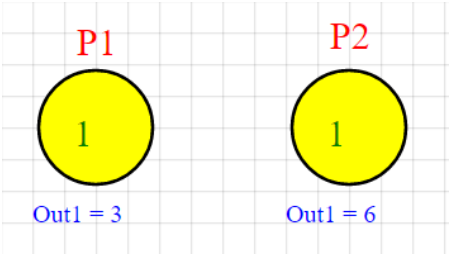


Figura 4. 21 – Dois lugares com o mesmo sinal de saída Out1

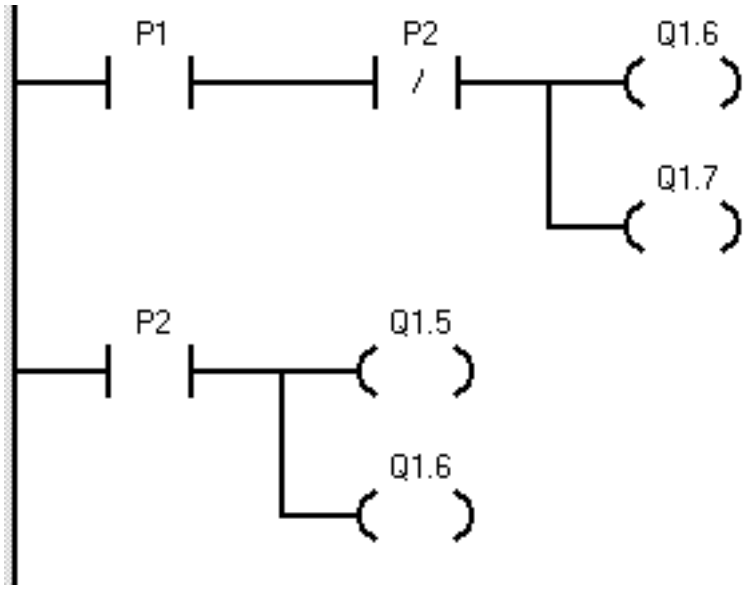


Figura 4. 22 – Sinal de saída Out1=3 e Out1=6 respectivamente

4.1.4.1 Regra 4.1 (Disparo de Eventos de Saída)

Para finalizar, será necessário actualizar as saídas associadas aos eventos de saída, neste caso basta activar uma saída no caso do registo do evento de saída associado ser de evolução ascendente, ou desactivar uma saída no caso do registo do evento de saída associado ser de evolução descendente, mas antes um passo intermédio será necessário: condição de disparo do evento de saída. Os registos dos eventos de saída disparam quando o registo da transição à qual estão associados, também disparar (como já foi referido na Regra 3.2). Para exemplificar, o disparo de um evento de saída de evolução ascendente foi indicado na Figura 4. 23, e o disparo de um evento de saída de evolução descendente foi indicado na Figura 4. 24.

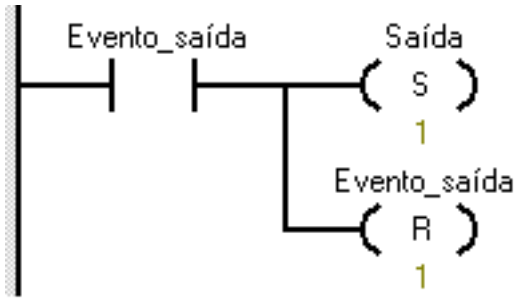


Figura 4. 23 - Resultado da aplicação da Regra 4.1 num evento de saída

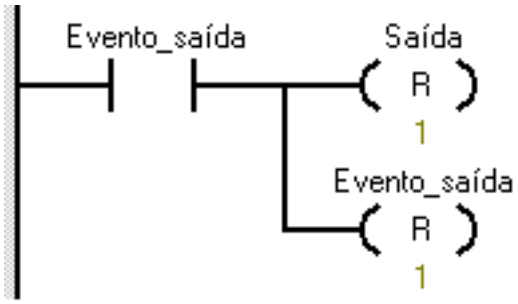


Figura 4. 24 - Resultado da aplicação da Regra 4.1 num evento de saída

4.2 Redes de Petri IOPT Limitadas

Como se pôde ver anteriormente, redes de Petri podem ter várias propriedades comportamentais, entre as quais as mais focadas pelo presente trabalho têm sido as redes de Petri seguras e as redes de Petri limitadas. Tendo já concluído as regras de tradução das redes de Petri seguras o próximo passo será estudar as regras de tradução aplicáveis para redes de Petri limitadas. Mantendo a mesma estrutura do algoritmo, e visto que a diferença entre as redes de Petri seguras para as limitadas está única e exclusivamente no número máximo de marcas por lugar e consequentemente um possível aumento no peso dos arcos, só algumas das regras já criadas para as redes de Petri seguras serão sujeitas a alterações, como será visto de seguida.

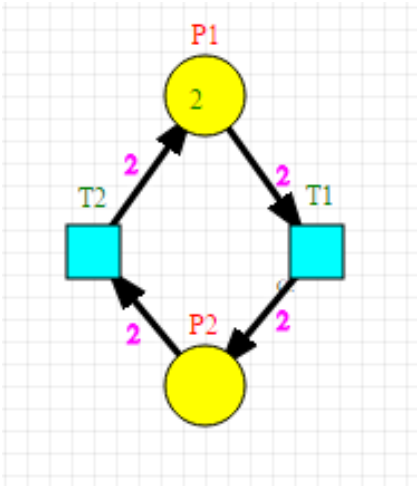


Figura 4. 25 - Rede de Petri limitada

4.2.1 Regra 5 (Inicialização da marcação inicial)

Tal como nas redes de Petri seguras, o primeiro passo na tradução de uma rede de Petri, passa novamente por inicializar os lugares com a marcação inicial. Sabendo quais os lugares e quantas marcas cada lugar tem no inicio, aplica-se a instrução "*MOVE_W*", que move um valor inteiro para o registo de cada lugar marcado (16 bits do tipo "*Word*", sendo este o tipo de todos os lugares das redes de Petri limitadas), indicando ainda o valor que se quer mover para o lugar. Deve-se também mover o valor zero (0) para o registo de cada lugar não marcado para garantir que cada um dos outros lugares é inicializado com um valor. Usando o exemplo da Figura 4. 25 obtemos o resultado presente na Figura 4. 26.

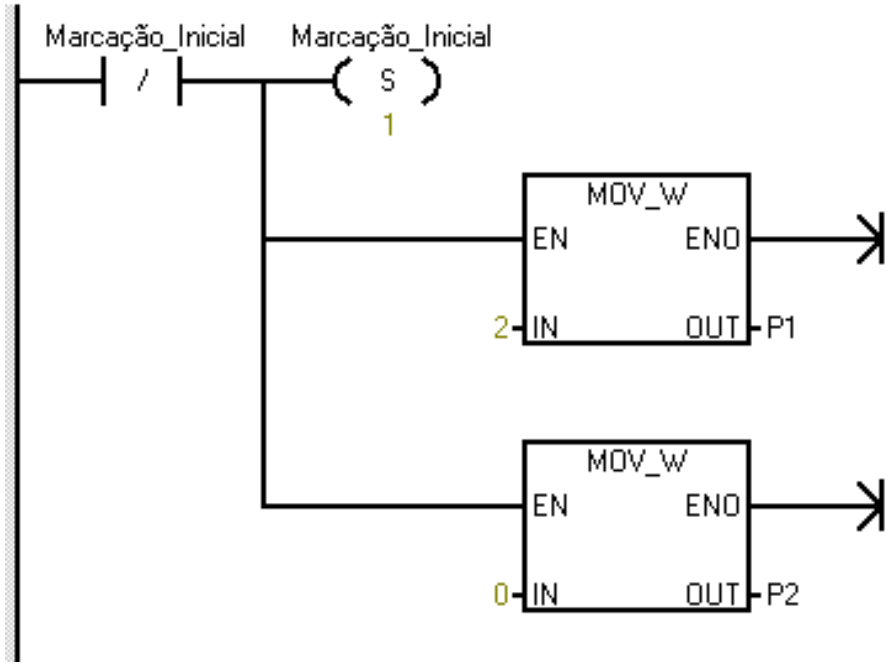


Figura 4. 26 – Inicialização da marcação inicial

4.2.2 Regra 2 (Leitura das entradas)

Esta regra e todas as suas variantes permanecerão inalteradas para as redes de Petri limitadas visto estas não terem quaisquer influências nos sinais e eventos de entrada.

4.2.3 Regra 6 (Disparo da transição)

O disparo das transições das redes de Petri limitadas serão separadas em várias sub-regras, da mesma forma que se separou o disparo de transições para as redes Petri seguras. Ir-se-á começar pelas condições necessárias para o seu disparo procedendo depois para o resultado do seu disparo, vendo ainda as diferenças da regra de tradução dos arcos de teste.

4.2.3.1 Regra 6.1 (Condições para o disparo das transições)

Para que possa ocorrer o disparo de uma transição numa rede de Petri limitada, é necessário verificar as três condições: lugares de entrada terem tantas ou mais marcas quanto o peso do arco de entrada (usando um comparador do tipo inteiro "*>=I*"); ocorrência de um possível conjunto de

eventos de entradas; e guarda de entradas respeitada. Após estas três condições serem verificadas, aplica-se o disparo do registo da transição e são retiradas ao lugar de entrada tantas marcas quanto o peso do arco de entrada (usando uma instrução matemática do tipo inteiro "*SUB_I*" para cada um dos registos dos lugares de entrada), como se pode verificar pela Figura 4. 27. Podendo cada transição ter determinado nível de prioridade, esta regra deverá também ela ser aplicada de forma sequencial e ordenada de forma crescente em função do grau de prioridade de cada transição.

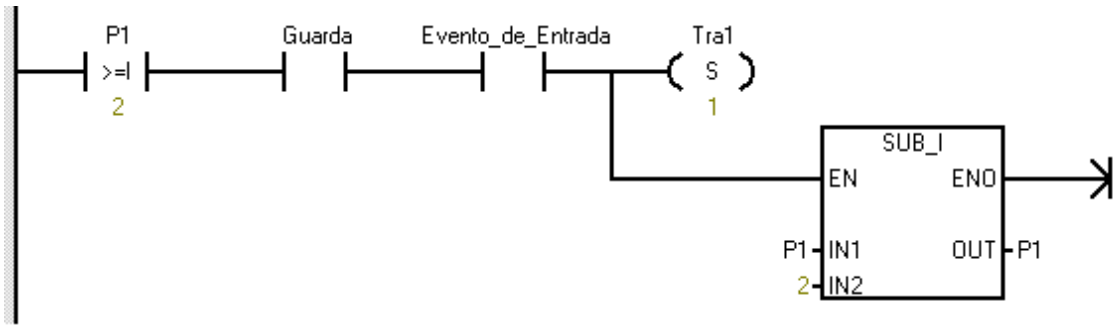


Figura 4. 27 - Resultado da aplicação da Regra 6.1

4.2.3.1.1 Regra 6.1.1 (Arcos de Teste)

Existe ainda a variante da regra 6.1 referente à regra de tradução das condições necessárias para o disparo de uma transição caso esta tenha um arco de teste. Da mesma forma, têm de ser verificados todos os arcos para esta possibilidade e caso se comprove a sua existência, a regra 6.1 será sujeita à seguinte alteração. A alteração será novamente criar uma cópia do lugar de onde parte o arco de teste e usar esta cópia como uma das três condições da aplicação da regra 6.1. Por fim, actualiza-se no início de cada passo o estado do registo deste lugar cópia através do copiado (usando um "*Move_W*"), como se poderá verificar na Figura 4. 28.

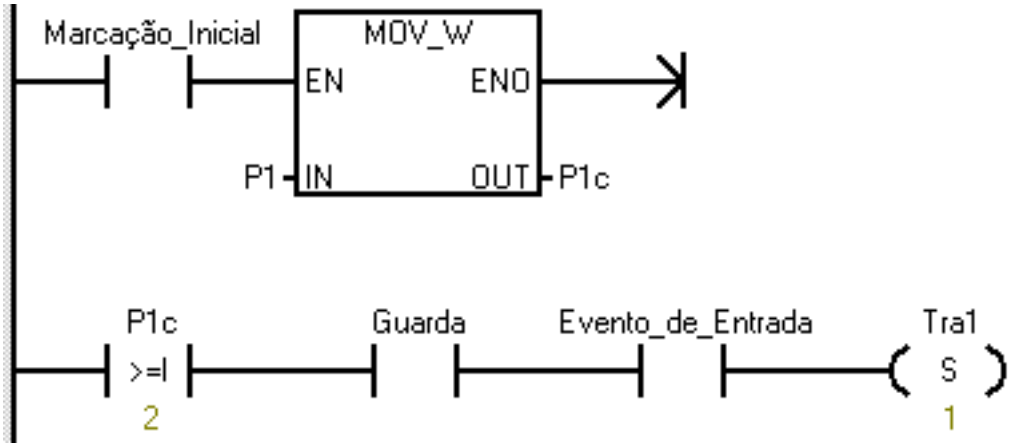


Figura 4. 28 - Aplicação da regra 6.1.1 com um arco de teste

4.2.3.2 Regra 6.2 (Resultado do disparo das transições)

A regra de tradução do disparo das transições de redes de Petri limitadas será feito tendo em conta o possível aumento do número de marcas a colocar num lugar de saída em função do peso do arco de saída (usando a instrução matemática do tipo inteiro "ADD_I" para cada um dos registos dos lugares de saída). Também será necessário proceder novamente ao disparo de possíveis eventos de saídas associados, e possíveis saídas associadas à transição para finalmente se desactivar o registo dessa mesma transição, como indicado na Figura 4. 29.

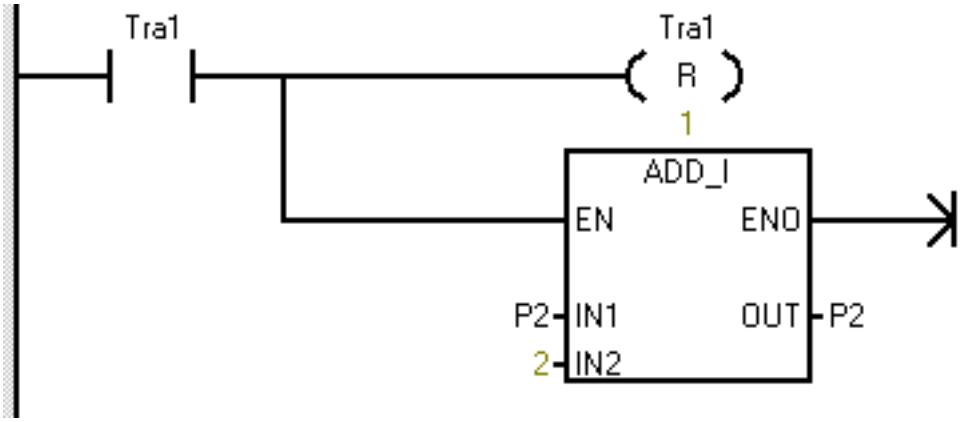


Figura 4. 29 - Resultado da aplicação da Regra 6.2

4.2.4 Regra 4 (Actualização das saídas)

Apesar das redes de Petri limitadas não terem quaisquer alterações nos sinais de saída quando comparadas com as redes de Petri seguras , na regra 4 (Actualização das saídas), a actualização das saídas é feita em função da marcação de lugares, logo esta terá de ser alterada para que possa ser suportada pelas redes de Petri limitadas. Mais uma vez, caso o sinal de saída seja inicializado a zero, activa-se o sinal de saída em função da presença de uma ou mais marcas nos registos dos lugares onde essa saída é activada a um, como ilustrado na Figura 4. 30. No caso de o sinal de saída ser inicializado a um, deve-activar o sinal de saída quando todos os registos dos lugares que o colocam a zero, tiverem zero marcas. Para sinais de saída inteiros não booleanos, será semelhante ao que foi apresentado para as redes de Petri seguras, mas mais uma vez haverá a necessidade de verificar a presença de uma ou mais marcas no registo do lugar onde o sinal de saída é evocado.

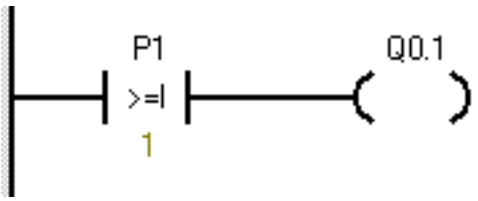


Figura 4. 30 – Actualização de um sinal de saída de um lugar

5 Aplicação

5.1 Programa de Tradução

Ao longo do presente trabalho foi desenvolvido um programa com o objectivo de integrar na ferramenta das *IOPT Tools*, mais uma ferramenta de tradução de redes de Petri para outra linguagem de programação, neste caso mais concretamente Lista de Instruções. O programa em questão foi desenvolvido através da linguagem "*Php*" no ambiente "*Eclipse*", tendo ainda necessitado de fazer uso do gerador de servidores virtuais da "*WAMPServer*", para que se pudesse proceder aos respectivos testes e simulações. Um excerto de código *Php* usado poderá ser visto na Figura 5. 1, que tratará de preparar o ficheiro "*.pnml*" para posteriormente ser corrido pelo algoritmo de tradução.

newfile.php | index.php

```
1  <?php
2  $pnml = new DOMDocument();
3  $pnml->load('2CarrosLugaresSeguros.pnml');
4  include 'newfile.php';
5  $pnml = newfile($pnml);
6
7  $pnml->save('result.awl');
8
9
10
11 ?>
```

Figura 5. 1 - Excerto de código Php

5.1.1 Algoritmo

O longo do programa foram construídas um serie de tabelas, algumas de três dimensões, das quais serão usadas para guardar toda a informação necessária para a correcta tradução das redes de Petri, informação essa retirada a partir do ficheiro "*.pnml*" gerado pelas *IOPT Tools*. O programa em questão foi desenvolvido usando uma estrutura como será apresentado de seguida.

5.1.1.1 Criação de tabelas de sinais e eventos de entrada

Estas serão as primeiras tabelas a ser criadas visto não necessitarem de informações de outras tabelas e por serem precisos certos dados desta tabela para a construção de outras tabelas, como se poderá ver de seguida. Para cada sinal de entrada foi criada uma linha com as seguintes colunas: Identificador do sinal de entrada gerado pelas *IOPT Tools*; Identificador do sinal de entrada para a Lista de instruções; tipo de sinal (booleano, range). Para cada evento de saída foi criada uma linha com as seguintes colunas: Identificador do evento de entrada gerado pelas *IOPT Tools*; Identificador do sinal de entrada associado; Identificador do evento de entrada para a Lista de instruções; Identificador do evento de entrada do passo anterior para a Lista de instruções; Tipo de transição (ascendente ou descendente). Tendo aplicado esta parte do programa, o resultado esperado pode ser visto na Figura 5. 2, em que se pode verificar, por exemplo neste caso, não irão haver valores para a tabela dos eventos de entrada.

Tabela de Sinais de Entrada
RdP ID | IL ID
A1 I0.2 boolean
A2 I0.3 boolean
B1 I0.0 boolean
B2 I0.1 boolean
BACK I1.1 boolean
GO I1.0 boolean

Tabela de Eventos de Entrada
RdP ID | ID do Sinal Associado | IL ID -1 | IL ID | Edge

Figura 5. 2 - Tabela de sinais e eventos de entrada do exemplo prático anterior

5.1.1.2 Criação de tabela de lugares

Esta tabela, tal como as anteriores, não necessita de quaisquer dados de outras tabelas para se poder proceder à sua criação. O contrário não será verdade, tornando estas excelentes candidatas ao segundo lugar da criação de tabelas do algoritmo. Para cada lugar foi criada uma linha com as seguintes colunas: Identificador do lugar gerado pelas *IOPT Tools*; Identificador do lugar para a Lista de instruções; Valor da marcação inicial; Nome do sinal de saída associado; Valor do sinal de saída associado. Na Figura 5. 3 pode ser vista esta tabela.

Tabela de Lugares
RdP ID | IL ID | Marcacao inicial
4 M0.0 0 Move1 1 Dir1 1
6 M0.1 0 Move2 1 Dir2 1
12 M0.2 0 Move2 1
13 M0.3 0 Move1 1
64 M0.4 1
65 M0.5 1
75 M0.6 0
77 M0.7 0

Figura 5. 3 - Tabela de lugares do exemplo prático anterior

5.1.1.3 Criação de tabelas de sinais e eventos de saída

Estas tabelas já irão necessitar de retirar informação da tabela de lugares anteriormente criada, fazendo sentido cria-las depois da tabela de lugares. Para cada sinal de saída foi criada uma linha com as seguintes colunas: Identificador do sinal de saída gerado pelas *IOPT Tools*; Identificador do sinal de saída para a Lista de instruções; valor com que o sinal de saída é inicializado; Identificador do lugar ao qual o sinal de saída foi associado; Valor do sinal de saída associado; Número total de colunas da tabela. O número de colunas será usado para mais tarde calcular a quantidade de sinais de saída associados ao lugar. Para cada evento de saída foi criada uma linha com as seguintes colunas: Identificador do evento de saída gerado pelas *IOPT Tools*; Identificador do sinal

de saída associado; Identificador do evento de saída para a Lista de instruções; Tipo de transição (ascendente ou descendente). Tal como na tabela de eventos de entrada, pode-se verificar pela Figura 5. 4, que a tabela de eventos de saída, neste exemplo em particular, se encontrará vazia.

Tabela de Sinais de Saidas
RdP ID | IL ID | Valor | Place ID | Num Colunas
Move1 Q0.0 0 M0.0 1 M0.3 1 7
Move2 Q0.2 0 M0.1 1 M0.2 1 7
Dir1 Q0.1 0 M0.0 1 5
Dir2 Q0.3 0 M0.1 1 5

Tabela de Eventos de Saida
RdP ID | ID do Sinal Associado | IL ID | Edge

Figura 5. 4 - Tabela de sinais e eventos de saída do exemplo prático anterior

5.1.1.4 Criação de tabelas de transições

Esta tabela irá necessitar de retirar dados de tabelas já criadas tais como as tabelas de eventos de entrada e de eventos de saída. Para cada transição foi criada uma linha com as seguintes colunas: Identificador da transição gerado pelas *IOPT Tools*; Identificador da transição para a Lista de instruções; Prioridade de transição; Guarda da transição; Eventos de entrada associados; Eventos de saída associados. Esta tabela poderá ser vista na Figura 5. 5.

Tabela das Transicoes
RdP ID | IL ID | Prioridade | Guarda | Eventos de Entrada/Saida
9 M1.0 1 B1 = 1
10 M1.1 1 B2 = 1
11 M1.2 1 BACK = 1
14 M1.3 NaN A1 = 1
15 M1.4 1 A2 = 1
67 M1.5 1 GO = 1

Figura 5. 5 - Tabela de transições do exemplo prático

5.1.1.5 Criação de arcos de entrada (lugar – transição)

Tendo já criadas as tabelas de lugares e de transições, pode-se agora proceder à criação das tabelas de arcos (de entrada e de saída, ou seja, nos sentidos lugar – transição e transição – lugar, respectivamente). Para cada arco de entrada foi criada uma linha com as seguintes colunas: Identificador do arco de entrada gerado pelas *IOPT Tools*; Identificador Lista de instruções do lugar origem; Identificador Lista de instruções da transição destino; Peso do arco; Tipo de arco (normal ou de teste); Identificador Lista de instruções do lugar cópia. Esta tabela poderá ser vista da Figura 5. 6.

Tabela dos Arcos de Entrada

RdP ID	IL Sourse ID	IL Target ID	Arc Value	Type(Normal/Test)	Test Arc ID
47	M0.0	M1.0	1	normal	
53	M0.1	M1.1	1	normal	
59	M0.2	M1.4	1	normal	
60	M0.3	M1.3	1	normal	
72	M0.5	M1.5	1	normal	
73	M0.4	M1.5	1	normal	
89	M0.6	M1.2	1	normal	
91	M0.7	M1.2	1	normal	

Figura 5. 6 - Tabela de arcos de entrada (lugar - transição) do exemplo prático

5.1.1.6 Criação de arcos de saída (transição - lugar)

Para cada arco de saída foi criada uma linha com as seguintes colunas: Identificador do arco de saída gerado pelas *IOPT Tools*; Identificador Lista de instruções da transição origem; Identificador Lista de instruções do lugar destino; Peso do arco. Esta tabela poderá ser vista na Figura 5. 7.

Tabela dos Arcos de Saida
RdP ID | IL Sourse ID | IL Target ID | Arc Value
29 M1.2 M0.3 1
56 M1.2 M0.2 1
68 M1.3 M0.4 1
69 M1.4 M0.5 1
79 M1.0 M0.6 1
81 M1.1 M0.7 1
86 M1.5 M0.0 1
88 M1.5 M0.1 1

Figura 5. 7 - Tabela de arcos de saída (transição - lugar) do exemplo prático

Tendo criado todas estas tabelas com todos os dados necessários para a aplicação das regras de tradução anteriormente propostas, o resto do algoritmo baseia-se na escrita do código Lista de Instruções fazendo uso da informação destas tabelas e aplicando cada uma das regras de tradução pela ordem com que foram apresentadas.

5.1.2 Funções de criação de código de Lista de Instruções

No início do programa foram criadas algumas funções necessárias para o correcto funcionamento do código Lista de instruções gerado. As funções usadas foram as seguintes:

Função "*Network_increment()*" – A linguagem Lista de Instruções requer que o inicio de cada processo/rede seja identificada com o número da rede seguinte. Para tal, foi necessário criar uma função que gerasse de forma automática cada uma destas novas redes sempre que necessário.

Função "*input_pnr_creation($input_pnr)*" – Como já foi referido, cada sinal de entrada terá de vir da rede de Petri, identificado com o número de um porto físico por forma a se poder atribui o respectivo canal de entrada na Lista de Instruções, podendo este variar entre I0.0 até I0.7 e I1.0 até I1.7, podendo este ultimo intervalo ser atribuído a um sinal de entrada de alcance superior a um (1).

Função "*output_pnr_creation($output_pnr)*" – Faz a mesma coisa que a função anterior mas para sinais de saída e usando antes os intervalos de Q0.0

até Q0.7 e Q1.0 até Q1.7, podendo este ultimo intervalo também ser atribuído a um sinal de saída de alcance superior a um (1).

Função "*il_id_creation()*" - Na Lista de Instruções, sempre que se cria um resultado interno binário numa rede, esse resultado tem de vir identificado como um registo binário "*M(número de byte).(número de bit)*", de forma semelhante aos sinais de entrada e saída (I0.0 ou Q0.0 respectivamente), para que mais tarde possa ser evocado em outra rede. Esta função quando evocada , trata de gerar de forma dinâmica todos estes registos sempre que se precisa de um novo.

Função "*il_word_creation()*" – Na Lista de Instruções, sempre que se precisa de um registo inteiro não binário (Word) "*VW(número par)*" (para redes de Petri limitadas onde os lugares podem conter mais que uma marca). Esta função quando evocada, trata de gerar de forma dinâmica todos estes registos, sempre que se encontra no *.pnml* um novo lugar limitado.

5.2 Exemplo de Aplicação

Tendo já visto no capítulo 4 as regras de tradução necessárias para transformar redes de Petri em Diagramas *Ladder* o próximo passo será mostrar um exemplo prático destas regras em acção. Para melhor visualização da aplicação das regras, será abordado um exemplo bastante usado e testado no presente trabalho: Controlo de dois vagonetes (Figura 5. 8). Este exemplo irá cobrir construção do modelo e simulações no ambiente das *IOPT Tools* do ponto de vista das redes de Petri seguras, bem como a construção e simulações do resultado da tradução para Diagramas *Ladder*/Lista de Instruções.

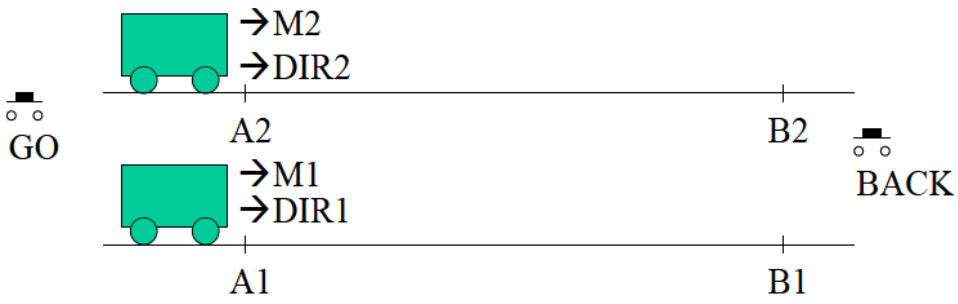


Figura 5. 8 - Controlo de dois Vagonetes

Neste exemplo o objectivo é controlar o movimento e direcção de dois vagonetes de forma independente. Para tal irão existir um conjunto de vários sensores e actuadores para os diferentes vagonetes: um botão "*GO*" que inicia o movimento de ambas as vagonetes (para tal é necessário que ambas estejam no ponto de partida detectadas pelos sensores "*A1*" e "*A2*"); um botão "*BACK*" que inicia o movimento de ambas as vagonetes no sentido contrário (para tal é também necessário ambas as vagonetes estarem no ponto de chegada detectadas pelos sensores "*B1*" e "*B2*"); Por fim os actuadores "*M1*" e "*M2*" referentes aos motores que iniciam o movimento das vagonetes 1 e 2 respectivamente; "*DIR1*" e "*DIR2*" que definem a direcção a que os motores "*M1*" e "*M2*" rodam (*DIR*=1 desloca-se para o lado direito enquanto que *DIR*=0 desloca-se para o lado esquerdo).

Tendo identificado todas as entradas e saídas do controlador, pode-se então dar inicio à construção da rede de Petri no ambiente *IOPT Tools*, como indicado na Figura 5. 9. Como se pode verificar ir-se-á ter um total de seis transições (uma para cada sensor) e quatro lugares para cada vagonete (um para a posição inicial, um para indicar o movimento para o lado direito, outro para indicar a posição final e por fim um para indicar o movimento para o lado esquerdo).

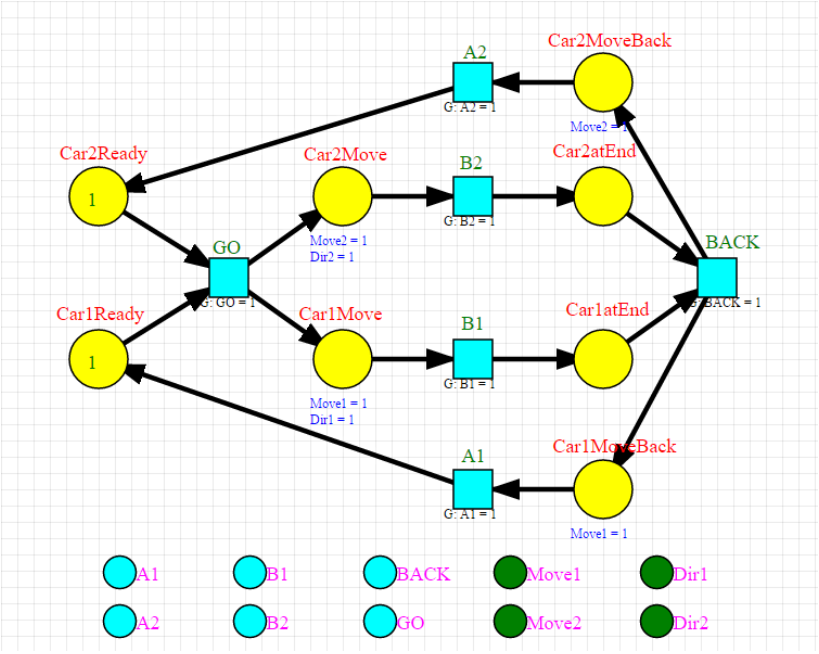


Figura 5. 9 - Rede de Petri do exemplo dos dois vagonetes

5.2.1 Simulação nas IOPT Tools

Para se testar a funcionalidade desta rede foi utilizado a ferramenta de simulação das *IOPT Tools*. Com ela pode-se verificar o seu comportamento e concluir que o modelo da rede de Petri produzido respeita o comportamento esperado pelo exemplo proposto, como será mostrado seguidamente.

A Figura 5. 10 mostra o resultado do disparo da transição "*GO*" após aplicado o sinal de entrada do botão "*GO*" (Guarda da transição "*GO*": *GO*=1), marcando os lugares "*Car1Move*" e "*Car2Move*" e dando início ao movimento dos motores "*M1*" e "*M2*", para o lado direito como indicado pelas saídas "*DIR1*" e "*DIR2*".

A Figura 5. 11 mostra o resultado da activação dos sensores de chegada "*B1*" e "*B2*", marcando os lugares "*Car1atEnd*" e "*Car2atEnd*", indicando que ambos os carros chegaram ao fim do seu percurso e desligando cada um dos seus motores "*M1*" e "*M2*" bem como os actuadores de direcção "*DIR1*" e "*DIR2*".

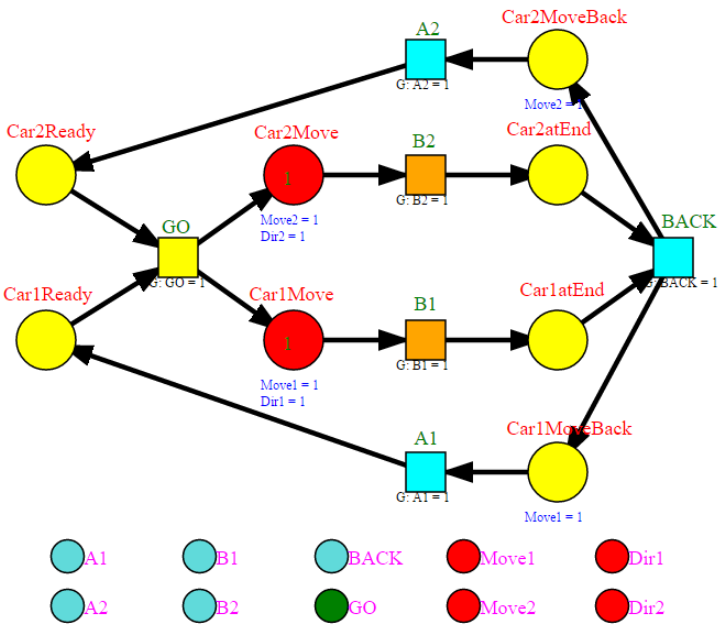


Figura 5. 10 - Resultado do disparo da transição GO

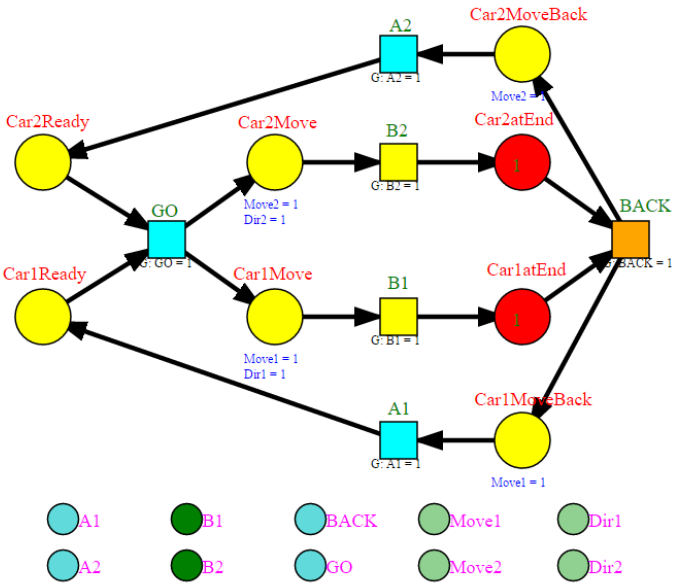


Figura 5. 11 - Resultado do disparo das transições B1 e B2

A Figura 5. 12 mostra o resultado da activação da transição "*BACK*" após aplicado o sinal de entrada do botão "*BACK*" (Guarda da transição "*BACK*": *BACK*=1), marcando os lugares "*Car1MoveBack*" e "*Car2MoveBack*" e dando início ao movimento dos motores "*M1*" e "*M2*", mas desta vez para o lado esquerdo como indicado pela ausência dos sinais de saída "*DIR1*" e "*DIR2*".

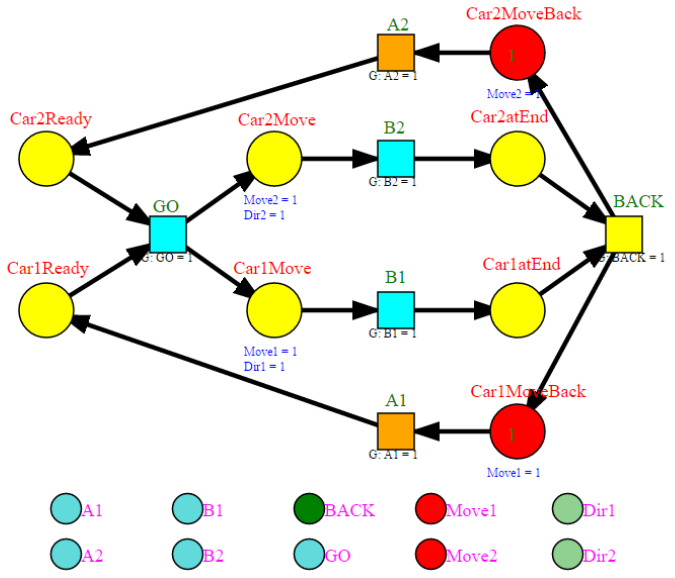


Figura 5. 12 - Resultado do disparo da transição BACK

Para concluir, o resultado da activação dos sensores de chegada "*A1*" e "*A2*", originará a marcação dos lugares "*Car1Ready*" e "*Car2Ready*" e o desligar dos motores "*M1*" e "*M2*", voltando a marcação inicial como indicado na Figura 5. 9. Pode-se ver ainda com mais detalhe, a mesma rede de Petri previamente exemplificada, a evolução de todos os lugares (as oito primeiras, a laranja), transições (as seguintes seis, a amarelo) e sinais de entrada/saída (as últimas dez, a azul) representada na Figura 5. 13.

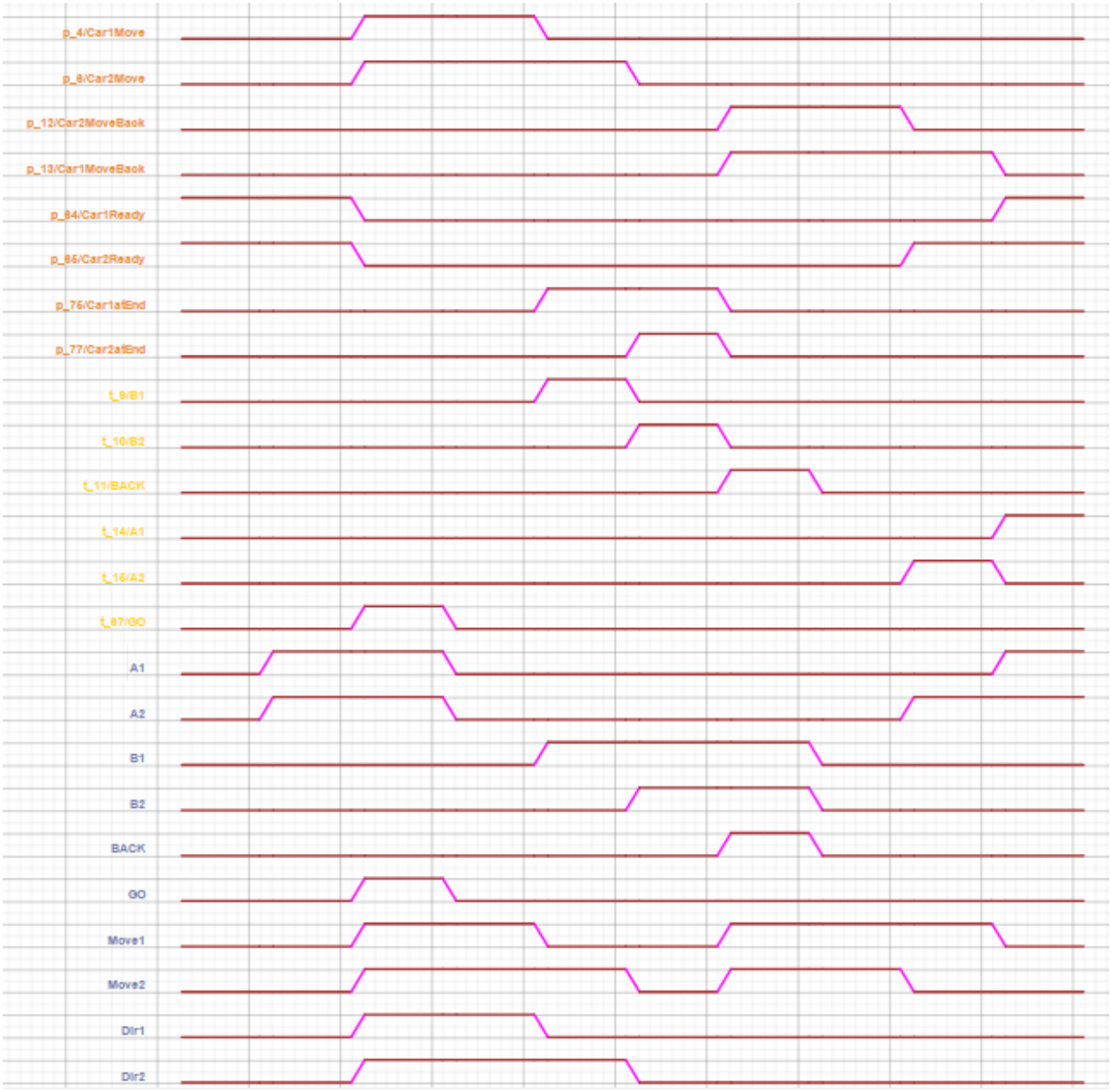


Figura 5. 13 - Simulação usando o Waveform Editor das IOPT-Tools

5.3 Tradução da rede de Petri

Após verificada a funcionalidade da rede de Petri poder-se-á proceder a sua tradução de acordo com as regras descritas no capítulo 4.1.1, mas antes será necessário confirmar que esta rede de Petri se trata realmente de uma rede de Petri segura. Para tal, usa-se a ferramenta de geração de espaço de estados das *IOPT Tools* para verificar que o número máximo de marcas por lugares é igual a um, como se pode ver na linha "*MAX Bounds*" da Figura 5. 14.

Com esta ferramenta de análise de espaço de estados, para além de se poder ver o número máximo de marcas por lugar, podem-se observar outras informações, tais como, número mínimo de marcas por lugar, contagem de situações de bloqueio, contagem de conflitos, ou tipo de situações inválidas, que poderão ser úteis, mas neste caso, visto não haver quaisquer tipos de problemas, terá sido útil para identificar redes de Petri seguras.

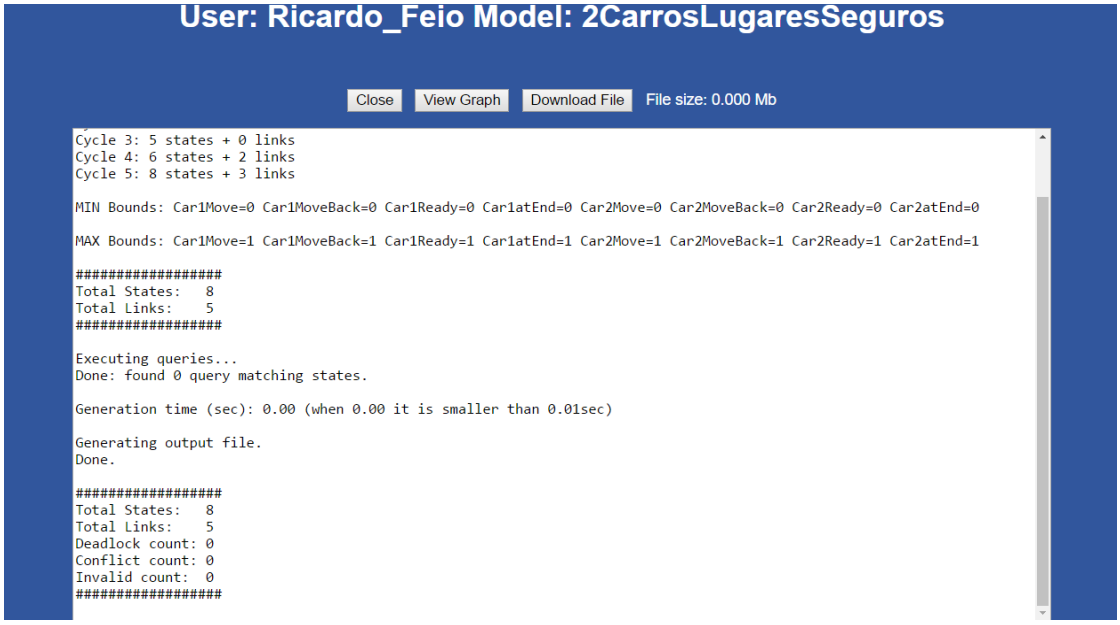


Figura 5. 14 - Espaço de estados do exemplo dos vagonetes

Tendo verificado que se está perante uma rede de Petri segura, o próximo passo será aplicar as regras de tradução de redes de Petri para Diagramas *Ladder*:

Aplicação da regra 1 (Inicialização da marcação inicial) – como esta rede de Petri só tem os lugares "*Car1Ready*" e "*Car2Ready*" com marcação inicial, então a aplicação desta regra resultará no aplicar de uma bobina *SET* no registo destes dois lugares, mais a lógica adicional que garante que este processo só corre uma vez, resultando no Diagrama *Ladder* indicado na Figura 5. 15 e Figura 5. 16.

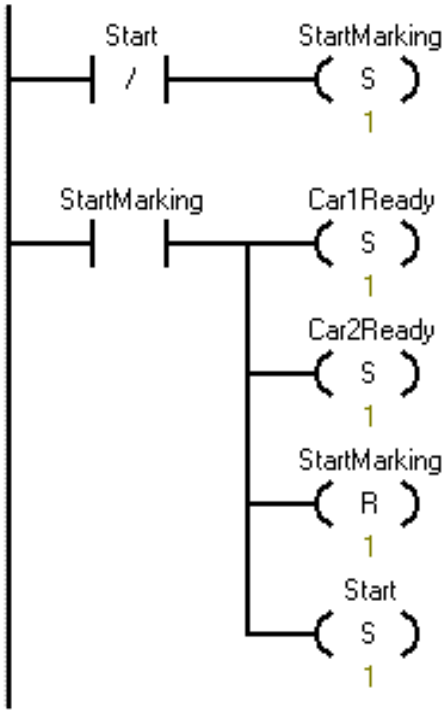


Figura 5. 15 - Aplicação da regra 1 de tradução RdP-DL.

```
Network 1
LDN     Start
S       StartMarking, 1

Network 2
LD      StartMarking
S       Car1Ready, 1
S       Car2Ready, 1
R       StartMarking, 1
S       Start, 1
```

Figura 5. 16 - Lista de Instruções equivalente da figura 5. 15

Aplicação da regra 2 (Leitura das entradas) – olhando para cada transição podemos verificar que cada uma tem a sua guarda de sinais de entrada, fazendo um total de seis guardas, e nenhum evento de entrada. Posto isto será aplicada somente a regra 2.1 para a construção das guardas.

Aplicação da regra 2.1 (Leitura de Guardas) – Cada uma destas guardas tem de ser analisada e traduzida de forma a respeitar a guarda. Olhando para a transição "*GO*" pode-se ver que tem a 1ª guarda: GO=1; Já a transição "*B1*" tem a 2ª guarda: B1=1; A transição "*B2*" tem a 3ª guarda: B2=1; A transição "*BACK*" tem a 4ª guarda: BACK=1; A transição "*A1*" tem a 5ª guarda: A1=1; E finalmente a transição "*A2*" tem a 6ª guarda A2=1. O resultado desta tradução pode ser verificado na Figura 5. 17 e Figura 5. 18.

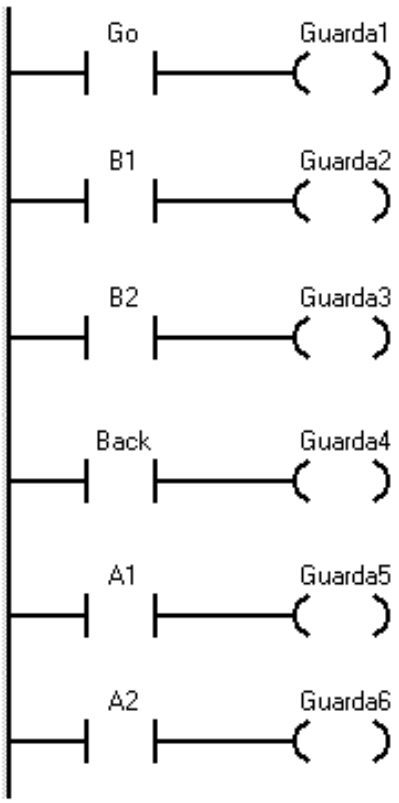


Figura 5. 17 - Aplicação da regra 2 de tradução RdP-DL

```
Network 3
LD      Go
=       Guarda1

Network 4
LD      B1
=       Guarda2

Network 5
LD      B2
=       Guarda3

Network 6
LD      Back
=       Guarda4

Network 7
LD      A1
=       Guarda5

Network 8
LD      A2
=       Guarda6
```

Figura 5. 18 - Lista de Instruções equivalente da figura 5. 17

Aplicação da regra 3 (Disparo das transições) – Como já foi referido anteriormente, a aplicação desta regra está dividida em duas fases: Verificação das condições de disparo da transição; e resultado do disparo da transição. Comecemos por ver a 1ª parte da regra:

Aplicação da regra 3.1 (Condições para o disparo das transições) – Como já foi discutido, para que o disparo de uma transição possa ocorrer, tanto as guardas e eventos de entrada tem de estar activos bem como os lugares de entrada da transição estarem marcados com tantas marcas quanto o peso do arco. Verificadas as condições de disparo das guardas e com a garantia de marcação do registo de alguns lugares que foi feito a partir da regra 1 (marcação inicial), pode-se então proceder à tradução da parte referente ao disparo das transições (Regra 3.2). Têm-se seis transições, que se traduzem em seis processos independentes, como indicados na Figura 5. 19 e Figura 5. 20:

Processo 1 – Verifica se os registos dos lugares, "*Car1Ready*" e "*Car2Ready*" estão marcados, bem como se a primeira guarda foi respeitada. Resultado: dispara a transição "*GO*" aplicando uma bobina *SET* no registo da transição, e

retirando a marca dos lugares "*Car1Ready*" e "*Car2Ready*" usando uma bobina *RESET*no registo dos lugares. Este será o primeiro processo a disparar visto serem os registos dos lugares "*Car1Ready*" e "*Car2Ready*" os marcados inicialmente.

Processo 2 – Verifica se o registo do lugar "*Car1Move*" se encontra marcado, bem como se a segunda guarda foi respeitada. Resultado: dispara a transição "*B1*", aplicando uma bobina *SET* no registo da transição, e retirando a marca do lugar "*Car1Move*" usando uma bobina *RESET* no registo do lugar.

Processo 3 – Semelhante à anterior, verifica se o registo do lugar "*Car2Move*" se encontra marcado, bem como se a terceira guarda foi respeitada. Resultado: dispara a transição "*B2*" aplicando uma bobina *SET* no registos da transição, e retirando a marca do lugar "*Car2Move*" usando uma bobina *RESET* no registo do lugar.

Processo 4 – Verifica se os registos dos lugares, "*Car1End*" e "*Car2End*" estão marcados, bem como se a quarta guarda foi respeitada. Resultado: dispara a transição "*BACK*" aplicando uma bobina *SET* no registo da transição, e retirando a marca dos lugares "*Car1End*" e "*Car2End*" usando uma bobina *RESET* no registo do lugar.

Processo 5 – Verifica se o registo do lugar "*Car1MoveBack*" se encontra marcado, bem como se a quinta guarda foi respeitada. Resultado: dispara a transição "*A1*" aplicando uma bobina *SET* no registo da transição, e retirando a marca do lugar "*Car1MoveBack*" usando uma bobina *RESET* no registo do lugar.

Processo 6 – Semelhante à anterior, verifica se o registo do lugar "*Car2MoveBack*" se encontra marcado, bem como se a sexta guarda foi respeitada. Resultado: dispara a transição "*A2*" aplicando uma bobina *SET* no registo da transição, e retirando a marca do lugar "*Car2MoveBack*" usando uma bobina *RESET* no registo do lugar.

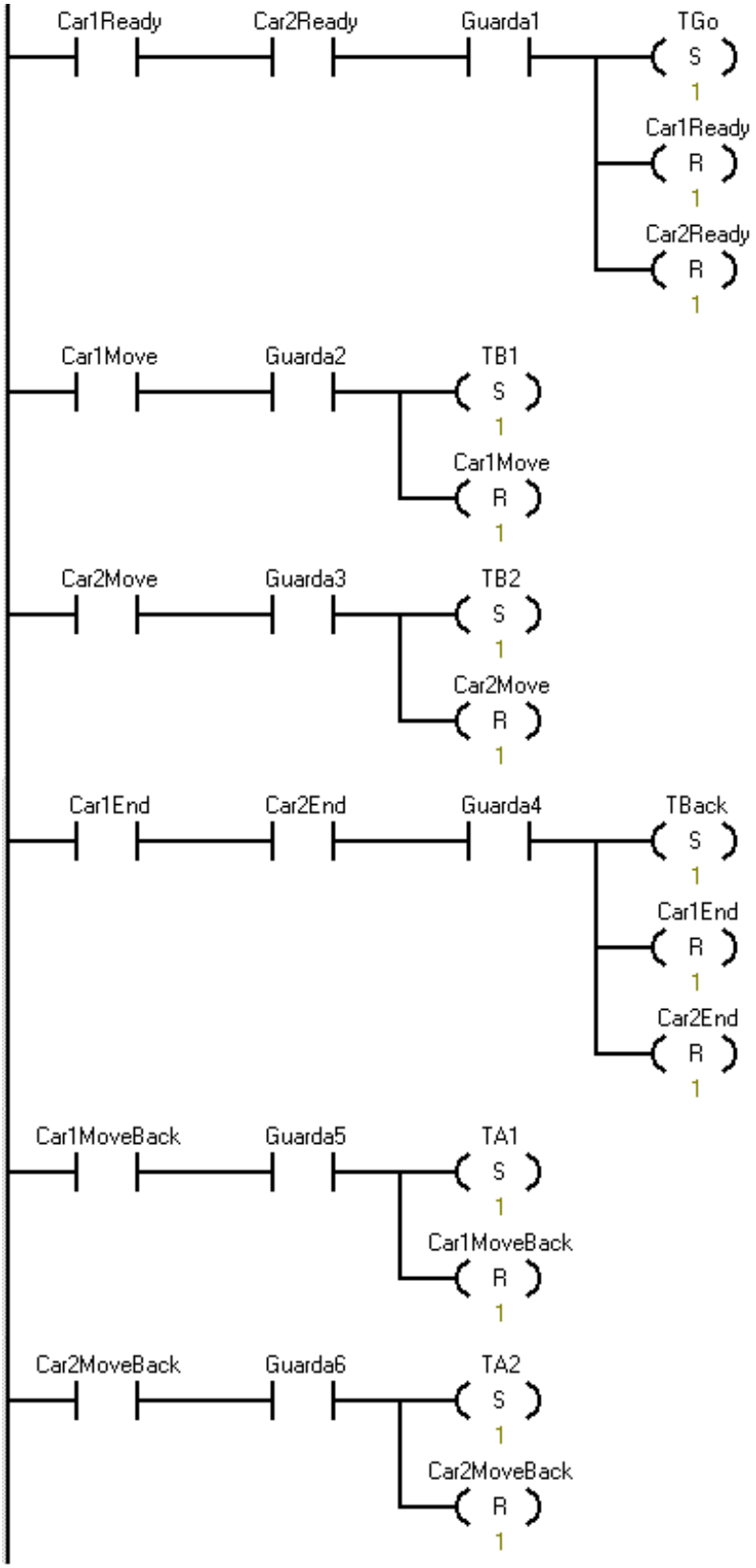


Figura 5. 19 - Aplicação da regra 3.1 de tradução RdP-DL

Network 9

```
LD      Car1Ready
A       Car2Ready
A       Guarda1
S       TGo, 1
R       Car1Ready, 1
R       Car2Ready, 1
```

Network 10

```
LD      Car1Move
A       Guarda2
S       TB1, 1
R       Car1Move, 1
```

Network 11

```
LD      Car2Move
A       Guarda3
S       TB2, 1
R       Car2Move, 1
```

Network 12

```
LD      Car1End
A       Car2End
A       Guarda4
S       TBack, 1
R       Car1End, 1
R       Car2End, 1
```

Network 13

```
LD      Car1MoveBack
A       Guarda5
S       TA1, 1
R       Car1MoveBack, 1
```

Network 14

```
LD      Car2MoveBack
A       Guarda6
S       TA2, 1
R       Car2MoveBack, 1
```

Figura 5. 20 - Lista de Instruções equivalente da figura 5. 19

Aplicação da regra 3.2 (Resultado do disparo das transições) – Tendo separado aquilo que deveria ser um processo elementar no disparo das transições, ter-se-á agora de traduzir o que será o resultado do disparo das transições após aplicada a regra 3.1. Mais uma vez, como se têm seis transições, no fim haverá seis processos independentes, como aqueles indicados na Figura 5. 21 e Figura 5. 22:

Processo 1 – Após activo o registo da transição "*GO*" marcam-se os lugares usando uma bobina *SET* nos registos de "*Car1Move*" e "*Car2Move*". No fim desactiva-se a transição "*GO*" usando uma bobina *RESET* no registo da transição.

Processo 2 – Após activo o registo da transição "*B1*" marca-se o lugar usando uma bobina SET no registo de "*Car1End*". No fim desactiva-se a transição "*B1*" usando uma bobina *RESET* no registo da transição.

Processo 3 – Após activo o registo da transição "*B2*" marca-se o lugar usando uma bobina *SET* no registo de "Car2End". No fim desactiva-se a transição "B2" usando uma bobina *RESET* no registo da transição.

Processo 4 – Após activo o registo da transição "*BACK*" marcam-se os lugares usando uma bobina *SET* no registo de"*Car1MoveBack*" e "*Car2MoveBack*". No fim desactiva-se a transição "*Back*" usando uma bobina RESET no registo da transição.

Processo 5 – Após activo o registo da transição "*A1*" marca-se o lugar usando uma bobina *SET* no registo de"*Car1Ready*". No fim desactiva-se a transição "*A1*" usando uma bobina *RESET* no registo da transição.

Processo 6 – Finalmente, após activo o registo da transição "*A2*" marca-se o lugar usando uma bobina *SET* no registo de "*Car2Ready*". No fim desactiva-se a transição "*A2*" usando uma bobina RESET no registo da transição.

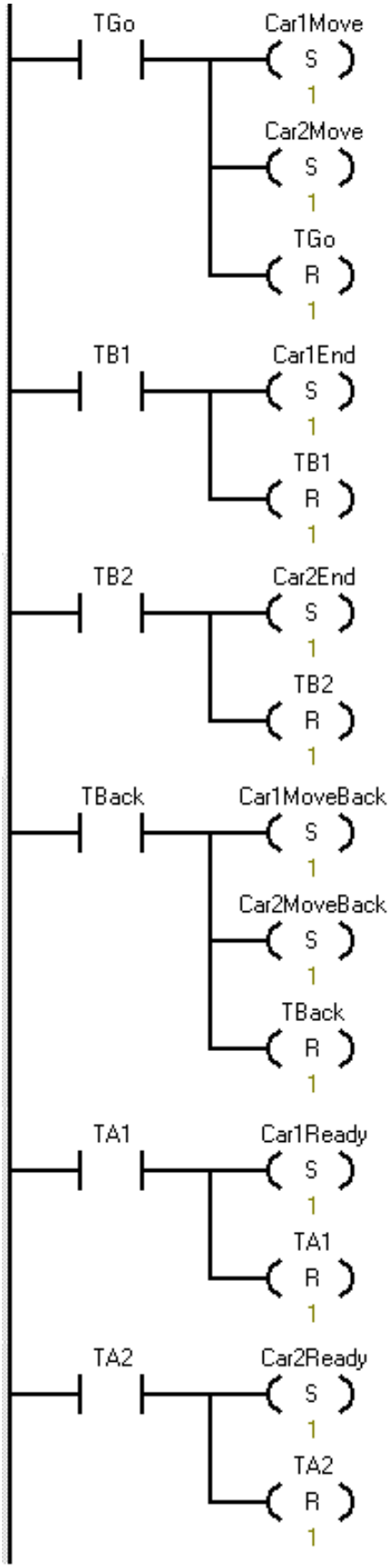


Figura 5. 21 - Aplicação da regra 3.2 de tradução RdP-DL

```
Network 15
LD      TGo
S       Car1Move, 1
S       Car2Move, 1
R       TGo, 1

Network 16
LD      TB1
S       Car1End, 1
R       TB1, 1

Network 17
LD      TB2
S       Car2End, 1
R       TB2, 1

Network 18
LD      TBack
S       Car1MoveBack, 1
S       Car2MoveBack, 1
R       TBack, 1

Network 19
LD      TA1
S       Car1Ready, 1
R       TA1, 1

Network 20
LD      TA2
S       Car2Ready, 1
R       TA2, 1
```

Figura 5. 22 - Lista de Instruções equivalente da figura 5. 21

Aplicação da regra 4 (Actualização das saídas) – Uma vez activos os registos dos lugares, poder-se-á proceder a actualização das saídas. Como foi referido na regra 4 cada saída terá direito a um processo independente evocando como condição para activação, todos os lugares onde esta possa ser actualizada. Neste caso serão quatro processos para as quatro saídas: "*M1*", "*DIR1*", "*M2*" e "*DIR2*". Desta forma ir-se-á obter a tradução indicada na Figura 5. 23 e Figura 5. 24:

Processo 1 – A saída "*M1*" pode ser activa pelos registos dos lugares "*Car1Move*" ou "Car1MoveBack".

Processo 2 – A saída "*M2*" pode ser activa pelos registos dos lugares "*Car2Move*" ou "Car2MoveBack".

Processo 3 – A saída *"DIR1"* só pode ser activa pelo registo do lugar *"Car1Move"*.

Processo 4 – A saída *"DIR2"* só pode ser activa pelo registo do lugar *"Car2Move"*.

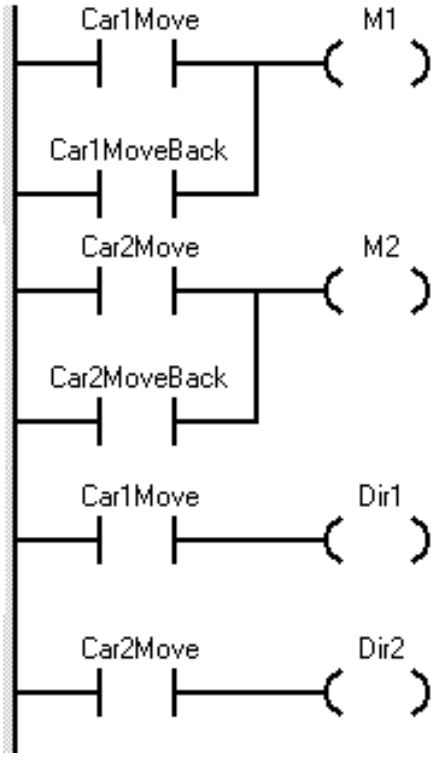


Figura 5. 23 - Aplicação da regra 4 de tradução RdP-DL

```
Network 21
LD      Car1Move
O       Car1MoveBack
=       M1

Network 22
LD      Car2Move
O       Car2MoveBack
=       M2

Network 23
LD      Car1Move
=       Dir1

Network 24
LD      Car2Move
=       Dir2
```

Figura 5. 24 - Lista de Instruções equivalente da figura 5. 23

5.3.1 Simulação da rede de Petri traduzida

Tendo concluído a aplicação de todas a regras de tradução de redes de Petri para Diagramas *Ladder* descritas no capítulo 4, e tendo produzido a Lista de Instruções dos Diagramas *Ladder* equivalente, pode-se agora passar à simulação e visualização dos resultados da aplicação destas regras. Para tal, foi utilizado a interface de utilizador gráfico *S7-200* [28] como indicado na Figura 5. 25.

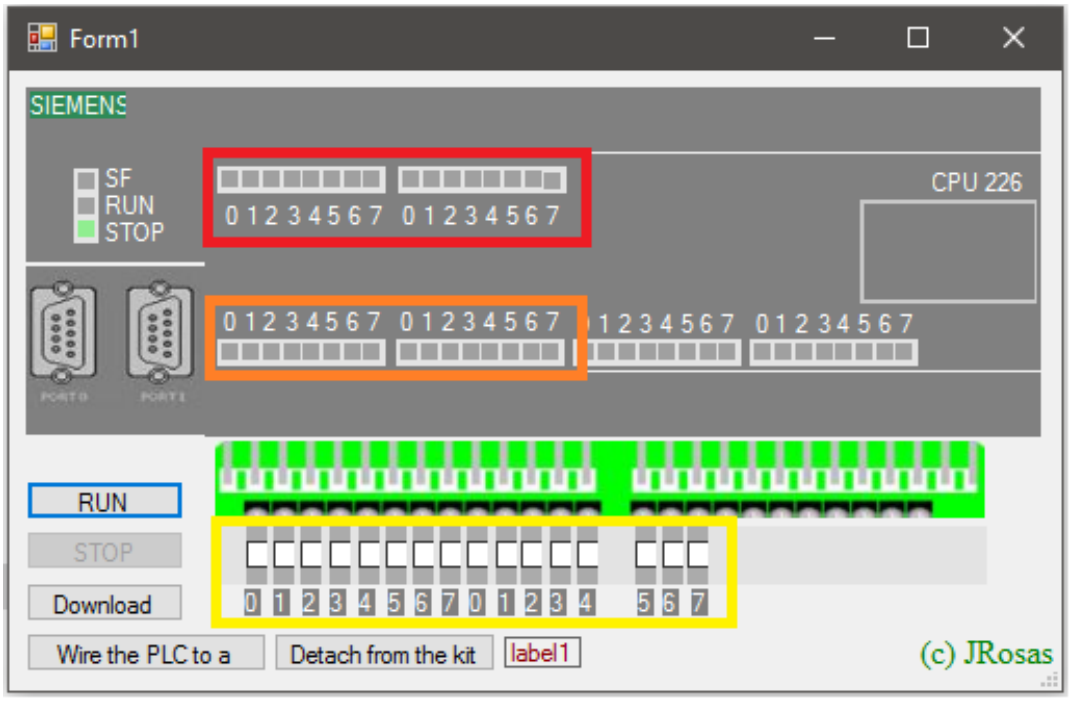


Figura 5. 25 - Interface de utilizador S7-200

Como se pode ver pela Figura 5. 25, existem várias áreas com caixas que podem estar a cinzento ou a verde, simulando o desligar ou ligar, respectivamente, de "*LEDs*", assim como caixas de selecção que podem estar seleccionadas ou não para simular o pressionar ou não de botões, respectivamente. A zona a vermelho contem dois grupos de "*LEDs*" para saídas (os quadrados de cima), que são identificadas pela interface de Q0.0 a Q0.7 da esquerda para a direita para o primeiro grupo e de Q1.0 a Q1.7 da esquerda para a direita para o segundo grupo. A zona a amarelo contem dois grupos de "*Botões*" para as entradas (as caixas de baixo), podendo estas ser usadas para

botões ou sensores, e estão identificadas pelo interface de I0.0 a I0.7 da esquerda para a direita para o primeiro grupo e de I1.0 a I1.7 da esquerda para a direita para o segundo grupo. A zona a laranja representa simples "*LEDs*" (os quadrados do meio) que acendem em função da selecção das caixas da zona amarela (de baixo). Para além disso a interface ainda contem um botão de "*Download*" para carregar o ficheiro de formado "*.awl*"que irá conter a linguagem "*Intruction List*" traduzida dos Diagramas *Ladder* resultantes da aplicação das regras de tradução anteriormente especificadas, um botão "*Wire the PLC to a*" que faz a ligação do interface com um possível emulador PLC e um botão "*Detach from the kit*" que interrompe a ligação para com esse mesmo emulador. Finalmente, tem um botão "*RUN*" para dar início ao processamento do ficheiro carregado e botão "*STOP*" para parar de correr esse mesmo ficheiro [28].

Para melhor compreender em que estado da rede de Petri se situa a simulação, foram incluídas oito saídas adicionais (uma para cada lugar), que indicará que lugares estão marcados em determinado momento. Esta atribuição foi aplicada aos seguintes "*LEDs*": Q0.0 – Car1Ready; Q0.1 – Car2Ready; Q0.2 – Car1Move; Q0.3 – Car2Move; Q0.4 – Car1End; Q0.5 – Car2End; Q0.6 – Car1MoveBack; Q0.7 – Car2MoveBack. Já as saídas referentes ao movimento e direcção dos vagonetes foram a seguinte: Q1.0 – M1; Q1.1 – DIR1; Q1.2 – M2; Q1.3 – DIR2. Para concluir, a atribuição dada aos sinais de entrada foi a seguinte: I0.0 – B1; I0.1 – B2; I0.2 – A1; I0.3 – A2; I1.0 – GO; I1.1 – BACK.

Dando início à simulação, após carregado o ficheiro para a interface de utilizador e pressionado o botão "*RUN*", obteremos a marcação inicial do "*Car1Ready*" e "*Car2Ready*", como indicado na Figura 5. 26.

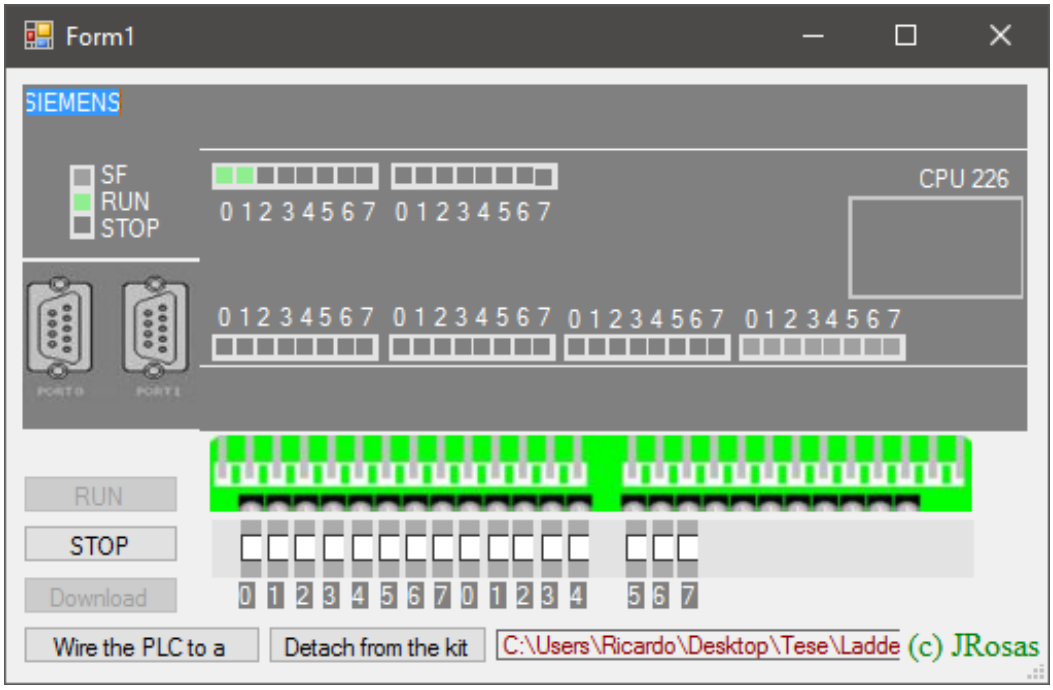


Figura 5. 26 - Marcação inicial dos vagonetes

Recorrendo à mesma sequência de activação de botões e sensores executada na simulação feita anteriormente no ambiente das *IOPT Tools*, ou seja, clicando primeiro no botão "*GO*", ir-se-á obter o estado dos lugares "*Car1Move*" e "*Car2Move*" marcados e os "*LEDs*" "*M1*", "*DIR1*", "*M2*", "*DIR2*" acesos, como indicado na Figura 5. 27.

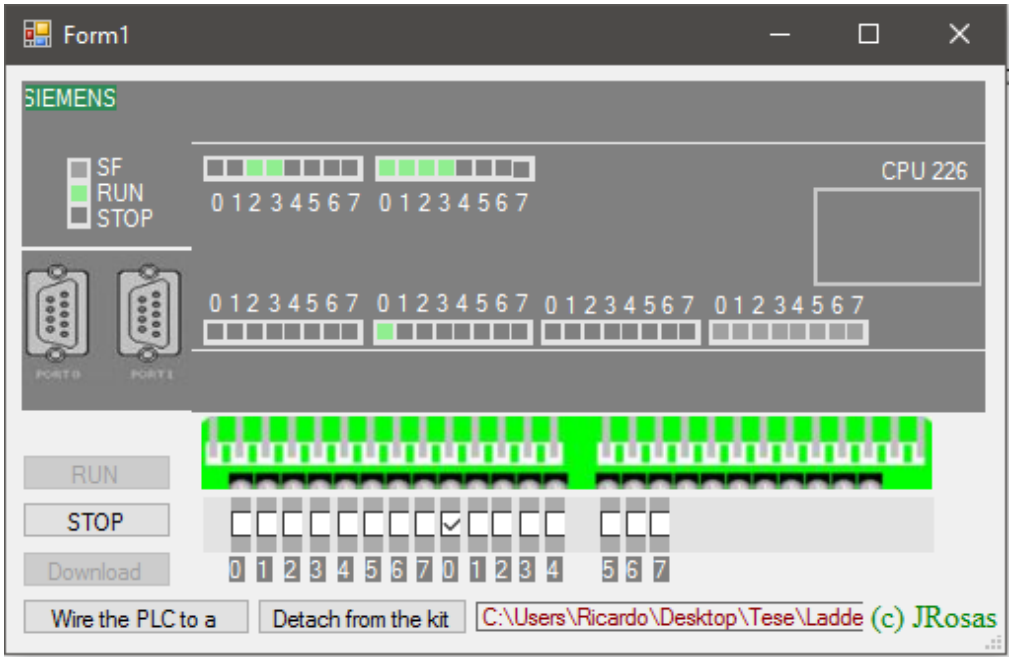


Figura 5. 27 - Vagonetes em movimento após disparo da transição GO

Seguindo a sequência, o próximo estado será aquele em que os vagonetes chegam ao fim do percurso ("*Car1End*" e "*Car2End*"), activando os sensores "*B1*" e "*B2*" e parando os motores dos vagões 1 e 2, respectivamente, como indicado na Figura 5. 28.

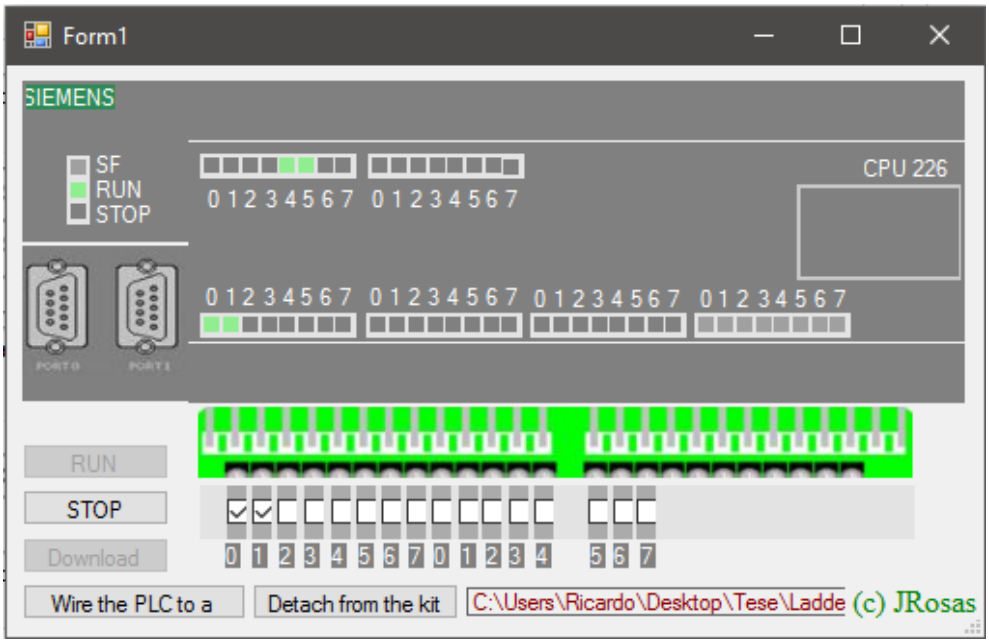


Figura 5. 28 - Vagonetes paradas depois de chegarem aos sensores B1 e B2

Uma vez activo o botão "*BACK*" as vagonetes irão retomar a marcha, desta vez no sentido oposto, activando só as saídas "*M1*"e "*M2*" nos lugares "*Car1MoveBack*" e "*Car2MoveBack*", respectivamente, como indicado na Figura 5. 29.

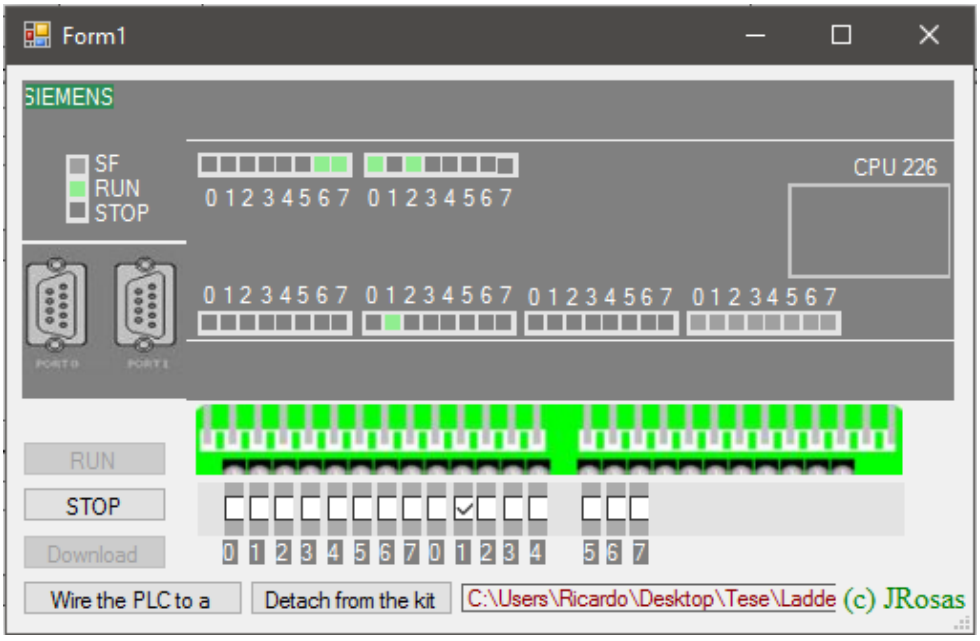


Figura 5. 29 - Vagonetes em movimento após disparo da transição BACK

Para concluir, chega-se ao estado em que ambos os vagonetes completam o seu percurso, activando os sensores "*A1*" e "*A2*" e desligando os motores "*M1*" e "*M2*" respectivamente, acabando com a marcação igual à inicial (lugares "*Car1Ready*" e "*Car2Ready*" marcados), como foi visto na Figura 5. 26.

5.3.2 Outros testes e suas simulações

Para além do exemplo prático anteriormente apresentado, que só cobre o teste e simulação da algumas das regras de tradução propostas pelo presente trabalho, muitos outros testes e simulações foram realizados de forma a cobrir todas as regras propostas. Nem todas serão demonstradas neste documento mas poderão ser verificadas através do programa desenvolvido ao longo dos

trabalhos desta dissertação, especificamente com o intuído de aplicar cada uma desta regras, por forma a traduzir redes de Petri seguras e limitadas da classe *IOPT*, para a linguagem de programação da norma *IEC61131* anteriormente referida como Lista de Instruções.

6 Conclusões

6.1 Introdução

Apesar de alguns conhecimentos anteriormente adquiridos, tanto nas redes de Petri da classe *IOPT* como na linguagem *Ladder*, a realização deste trabalho representou uma experiencia nova noutras áreas, tais como, nas linguagens de programação Listas de Instruções da norma internacional *IEC61131*, bem como em *Php*, tendo ainda aprofundado algum dos conhecimentos nas outras áreas, mesmo tendo encontrado diversas dificuldades, muitas dessas superadas com o apoio imprescindível do orientador e de outros professores do DEE. Algumas dessas dificuldades consistiram no aprofundar de alguns conhecimentos das redes de Petri das *IOPT Tools* e na elaboração de algumas das regras de tradução. Também quando se procedeu a criação do programa de tradução automática das redes de Petri *IOPT* para a linguagem Lista de Instruções, nomeadamente na leitura do ficheiro "*.pnml*" e na criação das Guardas das transições. E finalmente, na implementação do CLP físico para simulação e análise de resultados.

6.2 Cumprimento de Objectivos

De acordo com os objectivos inicialmente traçados, foi possível criar um conjunto de regras que irão permitir traduzir redes de Petri *IOPT* para a

linguagem Lista de Instruções da Norma *IEC61131*, mais concretamente redes de Petri seguras e limitada, com um certo grau de cobertura das funções e propriedades das *IOPT Tools*, não tendo sido possível abranger questões como sistemas *GALS* entre outras relacionadas.

Das simulações e testes realizados foi possível concluir que todas as regras de tradução propostas funcionam com um grau de sucesso satisfatório tendo em conta a quantidade de testes e simulações realizadas, tendo sido corrigidos muitos se não todos os problemas encontrados.

6.3 Implementação

No protótipo realizado foram implementadas com sucesso todas as regras de tradução propostas, tanto para as redes de Petri seguras como as redes de Petri limitadas.

6.4 Análise

Após a implementação do protótipo foi possível verificar a execução de todas as regras com sucesso, tendo corrigido todos os erros que foram encontrados durante os testes finais, de salientar os seguintes:

Eventos de entrada: Pelo cuidado que foi preciso ter quanto ao seu disparo visto este ter a duração de um passo por disparo.

Guardas: Demonstraram ser uma das grandes dificuldades encontradas pelo grau de complexidade que estas podem atingir tendo em conta a quantidade de operandos, operadores e sub-expressões que estas podem conter.

Prioridades: Pela atenção que foi preciso dar, de forma a garantir que a ordem de construção das regras seria implementada da forma correcta, de forma a se evitar situações de conflitos.

O disparo das transições das redes de Petri *IOPT* têm a particularidade de, caso aptas e sem situações de conflito, dispararem ao mesmo tempo, isto é, a

destruição e criação de marcas nos lugares ocorre no mesmo passo. Tendo usado a estrutura de algoritmo proposto na aplicação das regras de tradução e tendo sido tomado em consideração a forma como os Diagramas Ladder e a Lista de Instruções são processados pelos CLPs, o disparo simultâneo das transições fica assim garantido visto estes ocorrem tambem no mesmo passo de processamento da Lista de Instruções dos CLPs.

De notar que seria possível fazer uso das regras de tradução das redes de Petri limitadas para a tradução de redes de Petri seguras, visto as redes de Petri seguras serem um caso particular das redes de Petri limitadas, mas será importante aplicar cada uma das regras de tradução para o fim a que foram destinadas, por questões de alocação de recursos, visto as regras de tradução das redes de Petri limitadas consumirem mais recursos do que as regras de tradução das redes de Petri seguras.

6.5 Comentário Final e Expectativa de Futuro

O presente trabalho requer que se realizem mais testes e simulações do protótipo para que se possa corrigir possíveis erros que possam não ter sido detectados. Para além das 1945 linhas de código do protótipo, espera-se poder continuar a melhorá-lo, não só na resolução de possíveis erros não detectados mas também na implementação de outras funcionalidades das redes *IOPT* que não tenham sido aprofundadas no presente trabalho, como por exemplo eventos de entrada e saída com sinais inteiros, o suporte dos restantes operadores no editor de expressões das guardas das transições, ou até mesmo melhoria na quantidade de recursos de memória usados na implementação das regras de tradução da redes de Petri limitadas (um registo binário por cada sinal de entrada e no máximo oito registos binários (byte) para um sinal de entrada, um registo binário por cada sinal de saída e no máximo oito registos binários (byte) para um sinal de saída, dois registos binários por cada evento de entrada, um registo binário por cada evento de saída, um registo binário por cada guarda, um registo binário ou dois registos *byte* (*Word)* por cada lugar, e finalmente, um registo binário por cada transição). Enquanto uma experiencia positiva, foi-me dado novas perspectivas na abordagem de problemas, como

aqueles que foram expostos no presente trabalho, podendo no futuro tentar melhorar alguns dos aspectos menos fortes do trabalho.

Espera-se que os conhecimentos adquiridos neste trabalho bem como o protótipo realizado possam vir a ser integrados como uma ferramenta adicional das *IOPT Tools* esperando ainda abrir caminho para novas ideias não só na criação de ferramentas de tradução das *IOPT Tools* como de outras linguagens úteis na modelação de sistemas de controlo de autómatos industriais.

Referências

[1] Website: http://www.plcopen.org/pages/tc1_standards/ , visitado a 21 de Setembro de 2016

[2] Website: http://www.controleng.com/single-article/iec-61131-3-whats-the-acceptance-rate-of-this-control-programming-standard/235bfa337c311b8e0dc0386ca3b5590a.html, visitado a 03 de Dezembro de 2016

[3] Website:http://www.controleng.com/industry-news/more-news/single-article/speaking-in-tongues-understanding-the-iec-61131-3-programming-languages/4123b0e66c3f2cb8bdd60d3cb20f944d.html, visitado a 03 de Dezembro de 2016

[4] L.Gomes, J. Barros,A. Costa, and R. Nunes. "Input-Output Place Transition Petri Net Class and Associated Tools". In: *Proceedings of the 5th IEEE International Conference on Industial Informatics (INDIN'07).* Vienna, Austria, 2007

[5] Fernando Pereira, Filipe Moutinho e Luís Gomes, "*IOPT Tools User Manual Version 1.1*", Faculdade de Ciências e Tecnologias – FCT, Portugal, GRES Research Group, 2014

[6] I. Buzhinsky, C. Pang, V. Vyatkin. "Formal Modeling of Testing Software for Cyber-Physical Automation Systems". *2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki, 2015, pp. 301-306, pág.1.

[7] Tadao Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, Vol. 77, No. 4, April 1989

[8] René David e Hassane Alla, "*Petri Nets and Grafcet Tools for modelling discrete event systems*", Prentice Hall, 1992; ISBN 0-13-327537-X

[9] Luís Gomes, Filipe Moutinho, Fernando Pereira, José Ribeiro, Anikó Costa e João-Paulo Barros, "*Extending Input-Output Place-Transition Petri nets for Distributed Controller Systems development*", UNINOVA - CTS, Portugal, 2014

[10] M. Moalla, J. Pulou, and J. Sifakis, "Synchronized Petri nets: A Model for the description of non-autonomous systems", In: *Mathematical Foundations of Computer Science 1978*, **Ed. By J. Winkowski, Vol. 64, Lecture Notes in Computer Science**

[11] C. Ranchandani. "Analysis of asynchronous concurrent systems by timed Petri nets", Tech. rep. Cambridge, MA, USA, 1974

[12] Georg Frey, "PLC Programming for Hybrid Systems via Signal Interpreted Petri Nets", *4th Int. Conf on Automation of Mixed Processes, ADPM 2000*, **Dortmund**

[13] F. Mountinho, L. Gomes, "Asynchronous-Channels Within Petri Net-Based GALS Distributed Embedded Systems Modeling", In: *IEEE Transition on Industrial Informatics*, **vol. 10, no. 4, pp. 2024-2033, Nov. 2014**

[14] "From non-autonomous Petri net models to code in embedded systems design"; Luís Gomes, João Paulo Barros, Rui Pais; *DESDes'04 - 2nd International Workshop on Discrete-Event System Design*; **Dychów near Zielona Gora, Poland, September 15-17, 2004**

[15] Website: http://www.iec.ch/index.htm , visitado a 21 de Setembro de 2016

[16] Website: https://webstore.iec.ch/searchform&q=61131 visitado a 21 de Setembro de 2016

[17] International Electrotechnical Commission, "*IEC 61131-1 Programmable Controllers – Part 1: General information***", International Standard, Edition 2.0, 2003-05, Website: https://webstore.iec.ch/preview/info_iec61131-1%7Bed2.0%7Db.pdf , visitado a 21 de Setembro de 2016**

[18] International Electrotechnical Commission, "*IEC 61131-2 Programmable Controllers – Part 2: Equipment requirements and tests***", International Standard, Edition 3.0, 2007-07, Website: https://webstore.iec.ch/preview/info_iec61131-2%7Bed3.0%7Db.pdf, visitado a 21 de Setembro de 2016**

[19] International Electrotechnical Commission, "*IEC 61131-3 Programmable Controllers – Part 3: Programming languages***", International Standard, Edition 3.0, 2013-02, Website: https://webstore.iec.ch/preview/info_iec61131-3%7Bed3.0%7Db.pdf , visitado a 21 de Setembro de 2016**

[20] International Electrotechnical Commission, "*IEC 61131-4 Programmable Controllers – Part 4: User guidelines***", International Standard, Second Edition, 2004-07, Website: https://webstore.iec.ch/preview/info_iec61131-4%7Bed2.0%7Den.pdf, visitado a 21 de Setembro de 2016**

[21] International Electrotechnical Commission, "*IEC 61131-5 Programmable Controllers – Part 5: Communications***", International Standard, Edition 1.0, 2000-11, Website: https://webstore.iec.ch/preview/info_iec61131-5%7Bed1.0%7Db.pdf, visitado a 21 de Setembro de 2016**

[22] International Electrotechnical Commission, "*IEC 61131-6 Programmable Controllers – Part 6: Functional safety***", International Standard, Edition 1.0, 2012-10, Website:**

https://webstore.iec.ch/preview/info_iec61131-6%7Bed1.0%7Db.pdf, visitado a 21 de Setembro de 2016

[23] International Electrotechnical Commission, "*IEC 61131-7 Programmable Controllers – Part 7: Fuzzy control programming*", International Standard, First Edition, 2000-08, Website: https://webstore.iec.ch/preview/info_iec61131-7%7Bed1.0%7Db.pdf, visitado a 21 de Setembro de 2016

[24] International Electrotechnical Commission, "*IEC 61131-8 Programmable Controllers – Part 8: Guidelines for the application and implementation of programming languages*", International Standard, Second Edition, 2003-09, Website: https://webstore.iec.ch/preview/info_iec61131-8%7Bed2.0%7Den.pdf, visitado a 21 de Setembro de 2016

[25] International Electrotechnical Commission, "*IEC 61131-9 Programmable Controllers – Part 9: Single-drop digital communication interface for small sensors and actuators (SDCI)*", International Standard, Edition 1.0, 2013-09, Website: https://webstore.iec.ch/preview/info_iec61131-9%7Bed1.0%7Db.pdf, visitado a 21 de Setembro de 2016

[26] SIEMENS, "*SIMATIC S7-200 Programmable Controller System Manual*", Edition 08/2005, A5E00307987-02

[27] T. Perme, "Translation of extended Petri net model into ladder diagram and simulation with PLC", *J. Mech. Eng.* , vol. 55, no. 10, pp. 608-622, 2009

[28] "*S7-200GUI*", João Almeida das Rosas, Faculdade de Ciências e Tecnologias, Universidade Nova de Lisboa