PETRILAB: UMA PLATAFORMA PARA SIMULAÇÃO E GERAÇÃO DE DIAGRAMAS LADDER DE CONTROLADORES A EVENTOS DISCRETOS MODELADOS POR REDES DE PETRI

Anderson Linhares de Souza

Projeto de Graduação apresentado ao Corpo Docente do Departamento de Engenharia Elétrica da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro Eletricista.

Orientadores: Lilian Kawakami de Carvalho
João Carlos dos Santos Basílio

Rio de Janeiro
Março de 2015

PETRILAB: UMA PLATAFORMA PARA SIMULAÇÃO E GERAÇÃO DE DIAGRAMAS LADDER DE CONTROLADORES A EVENTOS DISCRETOS MODELADOS POR REDES DE PETRI

Anderson Linhares de Souza

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA ELÉTRICA DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DE GRAU DE ENGENHEIRO ELETRICISTA.

Examinado por:

Prof. Lilian Kawakami de Carvalho, D.Sc.
(Orientadora)

Prof. João Carlos dos Santos Basílio, Ph.D.
(Orientador)

Prof. Marcos Vicente de Brito Moreira, D.Sc.

Prof. Richard Magdalena Stephan, Dr.-Ing.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2015

Souza, Anderson Linhares de

PETRILab: Uma Plataforma para Simulação e Geração de Diagramas Ladder de Controladores a Eventos Discretos Modelados por Redes de Petri / Anderson Linhares de Souza. – Rio de Janeiro: UFRJ/Escola Politécnica, 2015.

103 p. : il.; 29,7cm.

Orientadores: Lilian Kawakami de Carvalho e João Carlos dos Santos Basílio

Projeto de Graduação – UFRJ/Escola Politécnica/Departamento de Engenharia Elétrica, 2015.

Referências Bibliográficas: p. 79.

1. Redes de Petri. 2. Controladores Lógicos Programáveis. 3. Conversão. 4. Guia do Usuário. 5. Exemplo. I. Souza, Anderson Linhares de. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Departamento de Engenharia Elétrica. III. PETRILab: Uma Plataforma para Simulação e Geração de Diagramas Ladder de Controladores a Eventos Discretos Modelados por Redes de Petri.

Agradecimentos

Primeiramente, agradeço aos meus pais, que, apesar de todas as dificuldades, sempre se sacrificaram pra que eu tivesse a melhor educação possível. Sem eles nada disso teria sido possível.

Agradeço também a minha orientadora, professora Lilian Kawakami, que veio me procurar com a proposta deste projeto final. Durante todo o processo de desenvolvimento do projeto, ela confiou na minha capacidade, e soube me cobrar no ritmo certo. Também devo grande gratidão ao meu orientador, professor Basílio, pela dedicação no suporte com relação a confecção dessa monografia, além das inúmeras correções sugeridas.

Por fim, devo agradecer aos meus amigos, que fizeram essa jornada de cinco anos de curso passar em um instante.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletricista

PETRILAB: UMA PLATAFORMA PARA SIMULAÇÃO E GERAÇÃO DE DIAGRAMAS LADDER DE CONTROLADORES A EVENTOS DISCRETOS MODELADOS POR REDES DE PETRI

Anderson Linhares de Souza

Março/2015

Orientadores: Lilian Kawakami de Carvalho
João Carlos dos Santos Basílio

Departamento: Engenharia Elétrica

O PETRILab é um programa desenvolvido em Python para auxiliar na modelagem e implementação de controladores de Sistemas a Eventos Discretos (SEDs). A partir de uma interface gráfica simples e intuitiva, o usuário é capaz de modelar e simular uma Rede de Petri Interpretada para Controle (RPIC), um dos modelos de SEDs que melhor representa um sistema de automação. Além disso, com apenas um clique, é possível converter a RPIC modelada em um diagrama Ladder. Este trabalho tem como objetivo apresentar o PETRILab e todos os seus recursos, e prover um guia com todos seus comandos. Para ilustrar a eficácia do programa, um exemplo prático de projeto de um controlador a eventos discretos é apresentado.

Palavras-chave: Sistemas a Eventos Discretos, Redes de Petri, Automação, Programa, Python

Abstract of Graduation Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the dregree of Electrical Engineer

PETRILAB: A PLAFTORM FOR SIMULATION AND LADDER DIAGRAMS GENERATION OF DISCRETE EVENT CONTROLLERS MODELED BY PETRI NETS

Anderson Linhares de Souza

February/2015

Advisors: Lilian Kawakami de Carvalho
João Carlos dos Santos Basílio

Department: Electrical Engineering

PETRILab is a software developed in Python to assist the modeling of controllers for Discrete Event Systems (DES). Through a simple and intuitive graphical user interface, the user is able to model and simulate a Control Interpreted Petri Net (CIPN), one of the models of DES that best represents an automation system. Furthermore, with a single click, the modeled CIPN can be converted into a Ladder diagram. The objective of this work is to present PETRILab and all its features, and to provide a guide with all its commands. In order to show the effectiveness of the software, a practical example of the project of a discrete event controller using PETRILab is presented.

Keywords: Discrete Event Systems, Petri Nets, Automation, Software, Python

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Quando pensamos em grandes indústrias, o que geralmente vem em mente são as gigantescas máquinas de produção, capazes de realizar trabalhos impossíveis para o ser humano, seja pela força ou pela precisão requerida. No entanto, um ponto de extrema importância, que muitas vezes é esquecido, é o de que essas máquinas nada seriam sem um sistema de controle para operá-las.

Diversas máquinas da indústria operam em diferentes "modos" que são determinados por sensores ou pela operação de um ser humano; máquinas com tais características podem ser denominadas Sistemas a Eventos Discretos (SED). Existem diversos modelos teóricos para SEDs, dentre os quais se destacam as Redes de Petri. Elas modelam os SEDs de forma simples e intuitiva e possuem diversas extensões que ampliam sua capacidade de modelagem. Dentre essas extensões, vale citar as Redes de Petri Interpretada para Controle (RPICs), que têm elementos que visam aproximá-las de máquinas de automação reais.

Do ponto de vista prático de sistemas de automação, um dos aparelhos de grande destaque é o Controlador Lógico Programável (CLP), que consiste em um microcomputador programável com entradas para sensores e saídas para atuadores, sendo capaz de controlar sistemas de qualquer porte. Dentre suas linguagens de programação, a mais utilizada na indústria é o Ladder, uma linguagem gráfica simples e eficaz. Para unir os aspectos de modelagem teórica com a aplicação prática do controle de Sistemas a Eventos Discretos, Moreira & Basílio propuseram em [1] um método sistemático de conversão de Redes de Petri Interpretadas para Controle em diagramas Ladder. Dessa

forma, a modelagem por Redes de Petri ficou ainda mais interessante, pois pode ser convertida e implementada diretamente em um CLP.

Dada a forma sistemática de se converter uma RPIC em um diagrama Ladder proposta em [1], é natural pensar em uma maneira automática de se realizar essa conversão. É dentro desse contexto que foi desenvolvido o PETRILab, um programa escrito em Python, que conta com interface gráfica e é capaz de modelar e simular RPICs com simplicidade e agilidade. Além disso o programa é capaz de realizar a conversão de RPICs em diagramas Ladder proposta em [1] com apenas um clique, instantaneamente. Este trabalho visa apresentar o PETRILab, exibindo seus recursos e guia para o usuário. A eficácia do PETRILab é demonstrada por meio de um exemplo real retirado da literatura [2].

Este trabalho está estruturado da seguinte forma: no Capítulo 2 são apresentados os conceitos principais de Sistemas a Eventos Discretos e, em particular, das Redes de Petri; no Capítulo 3 são apresentados os Controladores a Eventos Discretos, com destaque para o CLP; no Capítulo 4 é apresentado o método de conversão proposto em [1]; no Capítulo 5 o programa PETRILab é apresentado; no Capítulo 6 o programa é utilizado para a modelagem do controle de um motor de indução; por fim, no Capítulo 7 é apresentada a conclusão do trabalho, assim como os trabalhos futuros a serem realizados.

Capítulo 2

Redes de Petri Interpretadas para Controle

Antes de começar a apresentar o programa PETRILab, é necessário rever a teoria por trás do objeto a ser modelado: a Rede de Petri Interpretada para Controle (RPIC). Neste abordaremos os seguintes temas: na seção 2.1 será definido o conceito básico de um sistema; na seção 2.2 será apresentado o Sistema a Eventos Discretos; na seção 2.3 será introduzida a Rede de Petri; e, por fim, na seção 2.4 será apresentada a Rede de Petri Interpretada para Controle.

2.1 O Conceito de Sistema

2.1.1 Definição de Sistema

Apresentar uma definição de sistema não é uma tarefa trivial, pois definições formais podem deixar a desejar. Seguem algumas das definições de diferentes literaturas:

1. Um sistema é uma agregação ou associação de coisas combinadas pela natureza ou homem para formar um integral ou complexo todo. *(Encyclopedia Americana)*
2. Um sistema é um grupo de itens que interagem regularmente, formando um conjunto unificado. *(Webster's Dictionary)*

3. Um sistema é uma combinação de componentes que agem em conjunto para realizar uma função que não seria possível com quaisquer das partes individualmente. *(IEEE Standard Dictionary of Electrical and Eletronic Terms)*

Apesar de cada definição conter palavras-chave que as diferencia das demais, podemos, de acordo com todas elas, tomar uma postura generalizada para definir um sistema da seguinte forma: uma combinação de componentes utilizada para realizar uma determinada função.

2.1.2 Modelagem de sistemas e variáveis de estado

Diante da definição de sistema apresentada acima, fica visível que existem inúmeros sistemas no nosso dia-a-dia, como aparelhos elétricos e eletrônicos, uma empresa, o corpo humano, dentre outros. Do ponto de vista da engenharia, o principal interesse no estudo de sistemas é o de criar *modelos* para estes, de forma a possibilitar um tratamento quantitativo de suas variáveis. Com um modelo acurado, é possível então prever o comportamento de um sistema, e possivelmente controlá-lo utilizando determinadas técnicas de controle. Os modelos visam, então, representar a dinâmica de um sistema da forma mais precisa possível. O diagrama da Figura 2-1 ilustra o processo de modelagem, no qual as entradas e saídas do sistema são tratadas como variáveis, e o sistema em si é aproximado por um modelo.

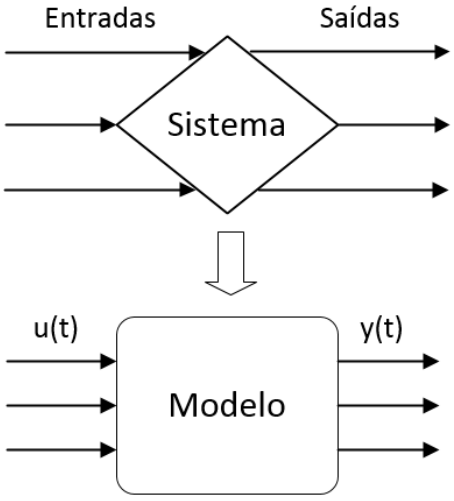


Figura 2-1: Modelagem de um sistema

Quando se trata de sistemas, é desejável que se tenham medidas que avaliem seu comportamento ao longo do tempo. Grosseiramente falando, o *estado* de um sistema é uma descrição desse comportamento. Na teoria de sistemas, no entanto, sua definição é feita mais precisamente, como segue: o estado de um sistema em um instante t_0 é a informação necessária em t_0 tal que a saída $\mathbf{y}(t)$, para todo $t \geq t_0$, seja unicamente determinada por esta informação e pelo conhecimento de $\mathbf{u}(t)$, $t \geq t_0$, em que $\mathbf{u}(t)$ e $\boldsymbol{y}(t)$ são vetores representando todas as entradas e saídas do sistema, respectivamente. [3]

Assim como as entradas e saídas, os estados também podem ser representados por um vetor $\boldsymbol{x}(t)$, e suas componentes $x_n(t)$ são denominadas variáveis de estado. As variáveis de estados podem ser classificadas primariamente em dois tipos: contínuas ou discretas. Temos variáveis contínuas quando elas podem assumir qualquer valor dentro de um conjunto não enumerável, como o conjunto dos números reais. Por outro lado, elas são denominadas discretas quando só assumem valores de um conjunto contável, como {LIGADO, DESLIGADO}, {1,2,3,4,...} ou {ALTO, BAIXO, MÉDIO}.

É comum adotarmos o tempo como uma variável contínua, afinal, isso se verifica na vida real. No entanto, na modelagem de alguns tipos de sistemas, é comum tratarmos o tempo como uma variável discreta, pois as transições de estado só ocorrem em instantes de tempo predefinidos, como é o caso de qualquer processamento computacional que tem seu *clock* constante. A este tipo de sistema damos o nome de *sistema de tempo discreto*.

2.2 Sistemas a Eventos Discretos

Quando os estados de um sistema são descritos por um conjunto discreto, e suas transições de estado ocorrem de forma assíncrona no tempo, associamos essas transições a *eventos*, e começamos a falar em *sistema a eventos discretos (SEDs)*. Um evento pode ser interpretado como um acontecimento qualquer, seja ele uma ação, como o apertar de um botão, ou o resultado da evolução dinâmica do sistema, como, por exemplo, uma caixa chegando em uma esteira de montagem, um fluido atingindo determinado nível em um tanque, ou, ainda, o resultado de uma falha, como, por exemplo, uma máquina travando.

Enquanto alguns sistemas são considerados *dirigidos pelo tempo*, ou seja, o tempo dita quando determinada mudança de estado irá ocorrer, outros são *dirigidos por eventos*,

pois as mudanças de estado são instantâneas e ocorrem apenas quando da ocorrência de eventos. À primeira vista, ambos os tipos parecem ser o mesmo, mas existem diferenças fundamentais entre eles:

- Nos sistemas dirigidos pelo tempo, as mudanças de estado são *sincronizadas* com o *clock*. Para cada passo do *clock*, um evento (ou nenhum evento) é selecionado, o estado muda e o processo se repete. O *clock* é responsável pela transição de estados.
- Já nos sistemas dirigidos por eventos, a ocorrência dos eventos determina os instantes de tempo associados. As transições de estado são resultado apenas dessas ocorrências *assíncronas* de eventos.

Com isso, podemos, então, definir formalmente um SED, como segue:

Um Sistema a Eventos Discretos (SED) é um sistema de estados discretos e dirigido por eventos, isto é, sua evolução de estados depende inteiramente da ocorrência de eventos discretos assíncronos ao longo do tempo [3].

Como exemplo de SED, podemos citar uma agenda eletrônica, onde os estados são o número de contatos armazenados. A evolução dos estados se dá quando há adição e remoção de determinado contato. O gráfico da Figura 2-2 mostra o comportamento desse sistema ao longo do tempo.

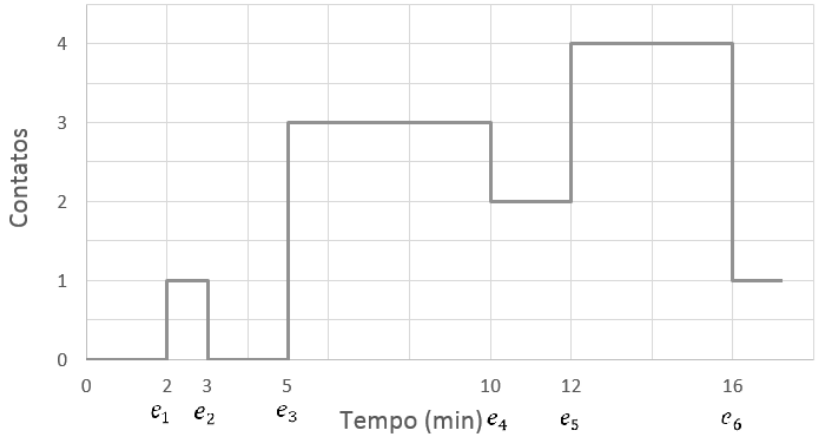


Figura 2-2: Evolução dos estados de um SED

2.3 Rede de Petri

Até agora, analisamos as características que definem um sistema a eventos discretos. Como dito anteriormente, o objetivo principal do estudo de qualquer sistema, do ponto de vista da engenharia, é o de modelá-lo. A Rede de Petri é um dos diversos modelos matemáticos possíveis para SEDs, e se destaca por sua estrutura de grafos de simples compreensão e alta capacidade de modelagem.

Nas Redes de Petri, eventos são associados a *transições*. Para uma transição ocorrer, diversas condições devem ser satisfeitas, condições essas que estão contidas nos chamados *lugares*, na forma de *fichas*. *Arcos* conectam lugares a transições e vice-versa, definindo as condições de *disparo* delas, além da dinâmica que ocorre no sistema quando essas são disparadas. Dessa forma, a Rede de Petri pode ser representada em um grafo, onde transições e lugares são nós, e são interconectados através de arcos.

Formalmente, uma Rede de Petri N é definida como uma quíntupla dada por:

$$N = (P, T, Pre, Post, x_0), \tag{1}$$

sendo:

- P o conjunto finito dos *lugares;*
- T o conjunto finito das *transições;*
- $Pre : (P \times T) \rightarrow \mathbb{N}$ a função de peso dos arcos que conectam lugares a transições;
- $Post : (T \times P) \rightarrow \mathbb{N}$ a função de peso dos arcos que conectam transições a lugares;
- $x_0 : P \rightarrow \mathbb{N}$ a função de marcação inicial.

No grafo associado às Redes de Petri, as transições são representadas como barras, enquanto os lugares são representados como círculos. As fichas podem ser representadas por números ou bolinhas, e os arcos por setas.

Na descrição de Redes de Petri, é comum utilizarmos $I(t_j)$ para representar o conjunto de lugares de entrada da transição t_j. De forma similar, $O(t_j)$ representa o conjunto de lugares de saída da transição t_j. A função $x : P \rightarrow \mathbb{N}$ é a função de marcação, ou seja, para um dado lugar p_i, $x(p_i)$ denota o número de fichas presentes nele.

A transição t_j é dita estar *habilitada* quando o número de fichas em cada um dos seus lugares de entrada for maior ou igual que o peso dos arcos conectando esses lugares à transição, ou seja:

$$x(p_i) \geq Pre(p_i, t_j), \text{ para todo } p_i \in I(t_j) \quad (2)$$

Se a transição t_j estiver habilitada para uma dada marcação $\underline{x}$ e t_j disparar, uma nova marcação $\underline{\bar{x}}$ é obtida, e sua evolução é dada pela seguinte equação:

$$\bar{x}(p_i) = x(p_i) - Pre(p_i, t_j) + Post(t_j, p_i), i = 1,2, \dots, n \quad (3)$$

A Figura 2-3 mostra um exemplo de um grafo de uma Rede de Petri. Para essa Rede de Petri, $P = \{p_0, p_1, p_2\}$, $T = \{t_0\}$, $Pre(p_0, t_0) = 1$, $Pre(p_1, t_0) = 2$, $Post(t_0, p\_2) = 3$ e $x_0 = [2\ 3\ 0]^T$. Vale notar que, caso não haja arco de entrada ou saída em uma transição, vinda de um determinado lugar, é definido que as funções *Pre* e *Post* tem valor zero. No exemplo, temos então que $Pre(p_2, t_0) = 0$, $Post(t_0, p_0) = 0$ e $Post(t_0, p_1) = 0$. Para a transição *t0* estar habilitada, todos os seus lugares de entrada devem ter um número de fichas maior do que o peso do arco que os conectam a *t0*, como dita a Equação (2). É fácil verificar que a transição está habilitada neste caso.

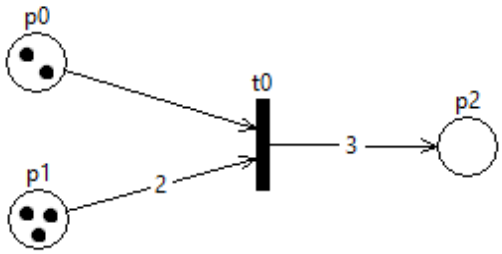


Figura 2-3: Exemplo de grafo de Rede de Petri

Ao disparar- t_0, o estado do sistema evolui segundo a Equação (3), e o novo estado pode ser visto no grafo da Figura 2-4. Observe que os lugares p_0 e p_1 perderam, respectivamente, uma e duas fichas e o lugar p_2 ganhou três fichas. Observe, também, que agora a condição da Equação (2) não é mais verdadeira, pois o número de fichas em p_1 é menor do que o peso do arco que o interconecta com t_0.

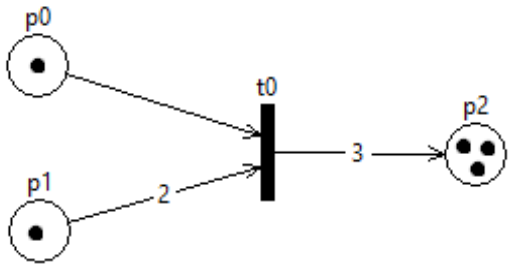


Figura 2-4: Transição t_0 disparada uma vez

Por fim, convém definir que um lugar p_i é um *lugar seguro* quando $x(p_i) \leq 1$ para qualquer marcação alcançável a partir do estado inicial $\underline{x_0}$. No exemplo da Figura 2-3 e Figura 2-4, nenhum lugar é seguro.

2.4 Redes de Petri Interpretadas para Controle

A Rede de Petri apresentada na seção anterior é apenas a forma mais genérica e simplificada dentre todos os tipos existentes. Existem diversas extensões para Redes de Petri que possuem alguns formalismos adicionais, ou modificam algumas das características originais, visando uma aplicação mais específica. Uma dessas extensões é a denominada Rede de Petri Interpretada para Controle (RPIC), que é voltada para a modelagem de sistemas implementados em máquinas de automação.

Antes de apresentarmos em detalhe a RPIC, é viável realizarmos uma introdução sucinta à denominada *Rede de Petri Rotulada Estendida*, que possui alguns parâmetros adicionais em relação à Rede de Petri genérica. A Rede de Petri Rotulada Estendida N é uma óctupla dada por:

$$N = (P, T, Pre, Post, E, l, x_0, In), \tag{4}$$

sendo P, T, Pre, $Post$ e x_0 são os mesmos parâmetros da Rede de Petri da seção anterior, e:

- E o conjunto de *eventos* para rotulação das transições;
- $l : T \rightarrow E$ a função de rotulação das transições;

- $In : (P \times T) \rightarrow \mathbb{N}$ a função de peso dos *arcos inibidores.*

Nesse tipo de Rede de Petri, estando a transição t_j habilitada, ela somente será disparada na ocorrência do evento $l(t_j) \in E$ associado a ela. Vale notar que diversas transições podem estar associadas a um evento e. Nesse caso, todas que estiverem habilitadas dispararão quando da ocorrência do evento e.

Essa rede também possui um novo tipo de arco, o *arco inibidor*, que aumenta significativamente sua capacidade de modelagem. Com o arco inibidor, é possível restringir o número máximo de fichas em um lugar p_i para que uma transição t_j esteja habilitada. Às condições para a habilitação de uma transição dadas pela Equação (2), deve também ser acrescida a condição seguinte:

$$x(p_i) < In(p_i, t_j), \qquad \text{para todo } p_i \in I(t_j) \tag{5}$$

Note que, caso o número de fichas seja igual ao peso do arco, a transição estará desabilitada. O arco inibidor é representado no grafo como uma linha com um círculo vazio na ponta, como pode ser visto no grafo de exemplo da Figura 2-5. Nesse exemplo, a transição t_0 está desabilitada, pois o número de fichas em p_0 é igual ao peso do arco inibidor que o interconecta com t_0. Nota-se também que a transição está associada com o evento $\uparrow S_0$, e só disparará caso o evento ocorra (e se estiver habilitada).

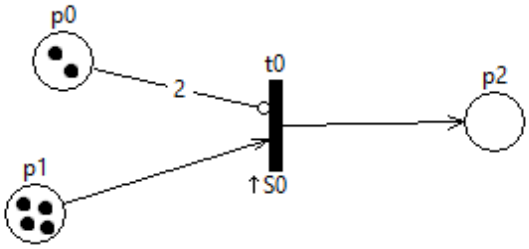


Figura 2-5: Exemplo de Rede de Petri Rotulada Estendida

A Rede de Petri Interpretada para Controle se assemelha à Rede de Petri Rotulada Estendida, porém contém estruturas adicionais para lidar com sensores e atuadores. As entradas da RPIC são sinais enviados pelos sensores para informar a ocorrência de eventos, e as saídas, associadas aos lugares, são ações impulsionais enviadas para a planta.

Para incluir temporizadores, a RPIC também tem transições temporizadas. Desse modo, o conjunto de transições T_C pode ser particionado como $T_C = T_C^0 U T_C^D$, em que T_C^0 é o conjunto de transições instantâneas, e T_C^D o conjunto de transições temporizadas. Definimos uma RPIC N_c formalmente como uma 13-tupla:

$$N_c = \left(P_C, T_C, Pre_C, Post_C, x_{0,C}, In_C, C, E_C, D, l_D, A, l_A\right), \tag{6}$$

sendo $(P_C, T_C, Pre_C, Post_C, x_0, In_C)$ uma Rede de Petri Rotulada Estendida, e:

- C o conjunto das condições de entrada associadas às transições em T_C^0;
- E_C o conjunto dos eventos de entradas associados às transições em T_C^0;
- $l_C : T_C^0 \to C \times E_C$ a função que associa cada transição em T_C^0 a um evento de E_C e uma condição de C;
- D o conjunto de atrasos associados com as transições em T_C^D;
- $l_D : T_C^D \to D$ a função de temporização que associa cada transição em T_C^D a um atraso em D;
- A o conjunto de ações, associadas a lugares seguros;
- $l_A : P_{Cs} \to 2^A$ a função de associação às ações, em que $P_{Cs} \subseteq P_C$ é o conjunto de lugares seguros.

Em uma RPIC, admite-se que todas as transições instantâneas $t_j \in T_C^0$ são associadas a uma condição de C e a um evento de E através da função l_C. Caso a condição associada à transição não seja especificada, é considerado que ela está associada a uma condição lógica sempre verdadeira. De modo análogo, caso o evento associado à transição não seja especificado, consideramo-la associada a λ, o evento sempre ocorrente. Já as transições temporizadas $t_j \in T_C^D$ estão associadas apenas a um tempo em D através da função l_D.

Uma transição instantânea $t_j \in T_C^0$ poderá disparar apenas se, além de estar habilitada, sua condição associada for verdadeira. A transição disparará de fato quanto seu evento associado ocorrer.

No momento em que uma transição temporizada $t_j \in T_C^D$ fica habilitada, inicia-se um contador. Quando esse contador alcançar o tempo associado à transição, ela irá

disparar automaticamente. Se a transição ficar desabilitada nesse meio tempo, o contador retornará a zero.

As ações são sinais de saída da RPIC, e estão associadas a lugares seguros através da função l_A. Caso um lugar seguro $p_{Ci} \in P_{Cs}$ não tenha uma ação associada, é considerado que $l_A(p_{Ci}) = \emptyset$. Quando um desses lugares muda seu número de fichas de zero para um, todas as suas ações associadas são executadas imediatamente. As ações podem ser do tipo impulsional ou contínua; uma ação impulsional é executada apenas no momento da mudança da marcação do lugar correspondente, enquanto uma ação contínua permanece sendo executada enquanto o lugar correspondente mantiver sua marcação igual a um. Nos grafos das RPICs, as ações impulsionais são representadas por um asterisco na frente do seu nome, e as contínuas apenas pelo nome.

O exemplo da Figura 2-6 ilustra um grafo de uma RPIC. Vale salientar que ambos os lugares dessa rede são seguros, pois suas marcações nunca superarão uma ficha. A transição t_0 está associada à condição c_0 e nenhum evento. Sendo assim, como está habilitada, a transição t_0 disparará assim que a condição c_0 se tornar verdadeira. Ao ser disparada, ela mudará a marcação do lugar p_1 de zero para um, executando a ação impulsional $*A_0$. Já a condição t_1 está associada ao evento $\uparrow S_0$, mas nenhuma condição foi especificada. Nesse caso, a transição irá disparar no momento em que $\uparrow S_0$ ocorrer, caso esteja habilitada. A transição t_2 é uma transição temporizada de 2 segundos. Sendo assim, 2s após sua habilitação, ela disparará, caso mantenha-se habilitada neste tempo. Se a ficha de p_1 for retirada antes desse tempo, o contador retornará a zero.

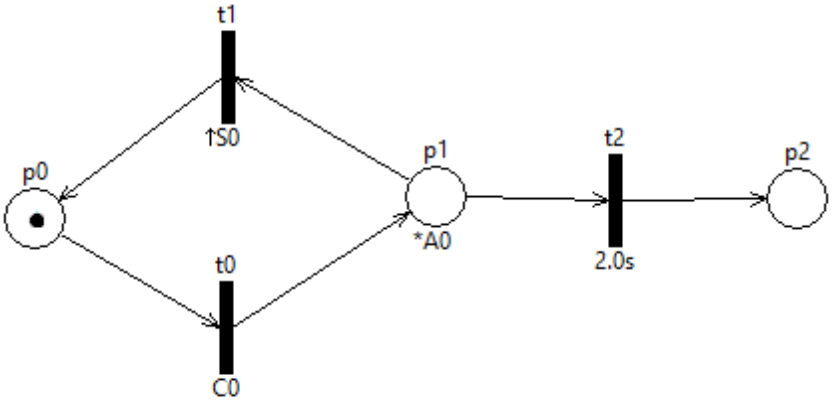


Figura 2-6: Exemplo de grafo de RPIC

Capítulo 3

Implementação de Controladores a Eventos Discretos

Para a operação de máquinas reais, são necessários sistemas de controle. Os primeiros sistemas de controle foram desenvolvidos durante a Revolução Industrial, no final do século XIX. Suas funções eram implementadas por engenhosos dispositivos mecânicos, que automatizavam algumas tarefas críticas e repetitivas das linhas de montagem da época.

Posteriormente, os dispositivos mecânicos foram substituídos por relés e contatores, o que permitiu realizar funções de controle mais complexas e sofisticadas. Mais tarde, com o desenvolvimento da tecnologia dos Circuitos Integrados (CIs), surgiu uma nova geração de sistemas de controle. Comparados com os relés, os CIs eram muito menores, mais rápidos e possuíam uma vida útil muito maior.

Em muitos sistemas de controle, que utilizam relés e CIs, a lógica de controle, ou algoritmo, é definida permanentemente pela interligação elétrica. Sistemas com lógica definida pela interligação elétrica são fáceis de implementar, mas o trabalho de alterar o seu comportamento ou sua lógica é muito difícil e demorado.

No início da década de 1970, com surgimento dos primeiros computadores industriais, esse problema começou a ser resolvido. Devido ao fato de o computador ser programável, ele proporciona uma grande vantagem em comparação com a lógica por interligação elétrica, utilizada nos sistemas com relés e CIs. No entanto, os primeiros

computadores eram grandes, caros, difíceis de programar e muito sensíveis à utilização em ambientes "hostis" encontrados em muitas plantas industriais.

O Controlador Lógico Programável (CLP) foi desenvolvido a partir de uma demanda existente na indústria automobilística norte-americana. Podemos considera-lo como um computador projetado para trabalhar no ambiente industrial, em que os transdutores e atuadores são conectados a robustos cartões de interface. Comparados com um computador de escritório, os primeiros CLPs tinham um conjunto de instruções reduzido, normalmente apenas condições lógicas e não possuíam entradas analógicas, podendo manipular somente aplicações de controle digital (discreto).

Este capítulo está estruturado da seguinte forma: na seção 3.1 serão apresentados os controladores lógicos programáveis; na seção 3.2 serão abordadas as diversas linguagens de programação para os CLPs; por fim, na seção 3.3 será estudada mais a fundo a linguagem Ladder.

3.1 Controladores Lógicos Programáveis

3.1.1 Definição do Controlador Lógico Programável

Um Controlador Lógico Programável é definido pelo IEC (*International Electrotechincal Commision*) como:

"Sistema eletrônico operando digitalmente, projetado para uso em um ambiente industrial, que usa uma memória programável para a armazenagem interna de instruções orientadas para o usuário para implementar funções específicas, tais como lógica, sequencial, temporização, contagem e aritmética, para controlar, através de entradas e saídas digitais ou analógicas, vários tipos de máquinas ou processos. O controlador programável e seus periféricos associados são projetados para serem facilmente integráveis em um sistema de controle industrial e facilmente usados em todas as suas funções previstas."

Em outras palavras, o CLP pode ser visto como um equipamento eletrônico de processamento que possui uma interface amigável com o usuário e tem como função executar o controle de vários tipos e níveis de complexidade.

3.1.2 Utilização dos CLPs

Toda planta industrial necessita de algum tipo de controlador para garantir sua operação segura e economicamente viável, desde o nível mais simples, em que pode ser utilizado para controlar o motor elétrico de um ventilador, ou para regular a temperatura de uma sala, até um grau de complexidade elevado, controlando a planta de um reator nuclear para produção de energia elétrica. Embora existam tamanhos e complexidades diferentes, todos os sistemas de controle podem ser divididos em três partes, com funções bem definidas: os transdutores (sensores), os controladores – como os CLPs – e os atuadores.

- **Sensores/transdutores:** transdutor é um dispositivo que converte uma condição física do elemento sensor em um sinal elétrico para ser utilizado pelo CLP através da conexão às entradas do CLP. Um exemplo típico é um botão de pressão momentânea, em que um sinal elétrico é enviado do botão de pressão ao CLP, indicando sua condição atual (pressionado ou liberado).
- **Atuadores:** sua função é converter o sinal elétrico oriundo do CLP em uma condição física, normalmente ligando ou desligando algum elemento. Os atuadores são conectados às saídas do CLP. Um exemplo típico é fazer o controle do acionamento de um motor utilizando um CLP. Nesse caso, a saída do CLP vai ligar ou desligar a bobina do contator que o comanda.
- **Controladores**: de acordo com os estados das suas entradas, o controlador utiliza um programa de controle para calcular os estados das suas saídas. Os sinais elétricos das saídas são convertidos no processo através dos atuadores. Muitos atuadores geram movimentos, tais como válvulas, motores bombas; outros utilizam energia elétrica ou pneumática. O operador pode interagir com o controlador por meio dos parâmetros de controle. Alguns controladores podem mostrar o estado do processo em uma tela ou em um *display.*

Um sistema de controle típico encontra-se na Figura 3-1. O controlador monitora o status do processo em tempo real através de um número definido de transdutores, que convertem grandezas físicas em sinais elétricos. Após receber os dados de entrada, ele então os processa de acordo com as instruções programadas e os parâmetros configurados, provendo informações sobre seu estado em sua interface. Por fim, provê sinais elétricos para atuadores a partir da lógica realizada.

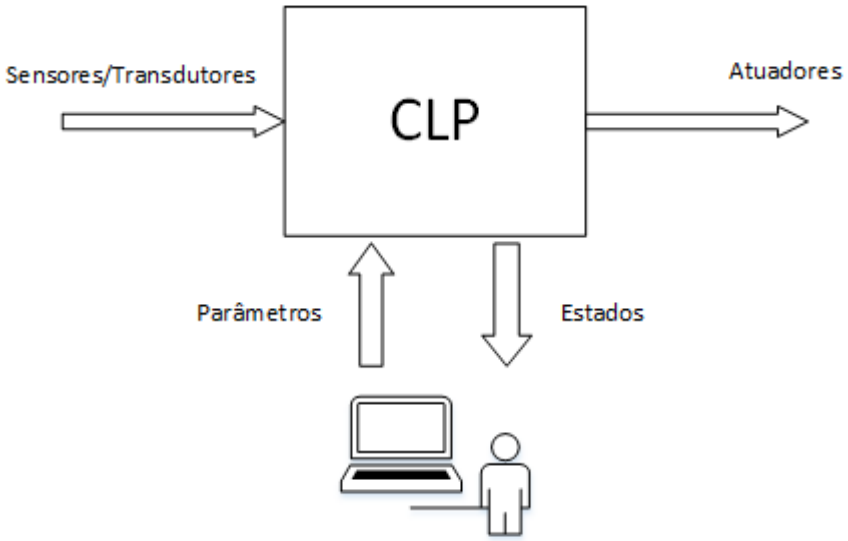


Figura 3-1: Sistema de controle típico utilizando-se um CLP

3.1.3 Aplicações do CLP

O CLP, devido às suas características especiais de projeto, tem um campo de aplicação muito vasto. A constante evolução do hardware e do software é uma necessidade para que o CLP possa atender às demandas dos processos.

É utilizado fundamentalmente nas instalações em que é necessário um processo de manobra, controle e supervisão. Dessa forma, sua aplicação abrange desde processos de fabricação industrial até qualquer processo que envolva transformação de matéria-prima.

As dimensões reduzidas, extrema facilidade de montagem e possibilidade de armazenar os programas que descrevem o processo tornam o CLP ideal para aplicações em processos industriais, como:

- Indústria de plástico;
- Indústria petroquímica;
- Máquinas de embalagens;
- Instalações de ara condicionado e calefação;
- Indústria de açúcar e álcool;
- Papel e celulose;
- Indústrias alimentícias;

- Mineração.

3.1.4 Arquitetura dos CLPs e princípio de funcionamento

O CLP é um equipamento de estado sólido que pode ser programado para executar instruções que controlam dispositivos, máquinas e operações de processos pela implementação de funções específicas, como lógica de controle, sequenciamento, controle de tempo, operações aritméticas, controle estatístico, controle de malha, transmissão de dados, etc.

Os primeiros controladores lógicos programáveis tinham como função primordial somente substituir os relés utilizados na indústria. A sua função era somente realizar operações sequenciais que eram anteriormente implementadas com relés, como, por exemplo, controle liga/desliga de máquinas e processos que necessitavam operações repetitivas. Em um curto tempo, esses controladores tiveram muitas melhorias em relação aos relés, como o uso de menor espaço e energia, indicadores de diagnóstico e ao contrário dos relés, a sua lógica de operação poderia ser mudada sem a necessidade de alteração das conexões físicas dos elementos.

Um controlador lógico programável pode ser dividido em duas partes: uma unidade central de processamento e sistemas de interface de entrada/saída. A Unidade Central de Processamento (ou *Central Processing Unit*, CPU, em inglês), comanda todas as atividades do CLP, sendo formada pelos três elementos mostrados na Figura 3-2.

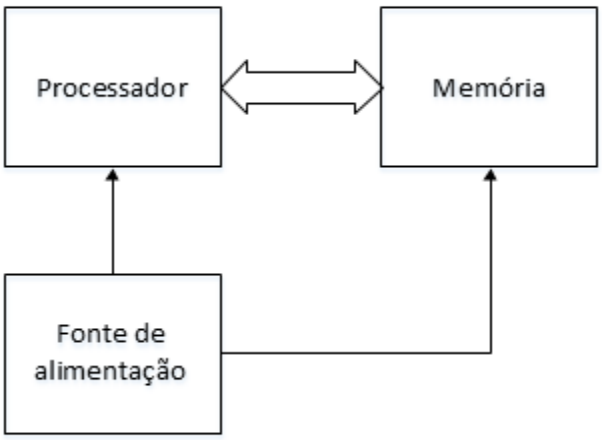


Figura 3-2: Diagrama de bloco dos principais componentes da CPU

Podemos ter um diagrama de blocos simplificado do CLP, como está ilustrado na Figura 3-3, consistindo de:

1. Fonte de alimentação
2. Entradas (analógicas e/ou digitais)
3. Saídas (analógicas e/ou digitais)
4. Unidade Central de Processamento (CPU)
5. Unidade de comunicação

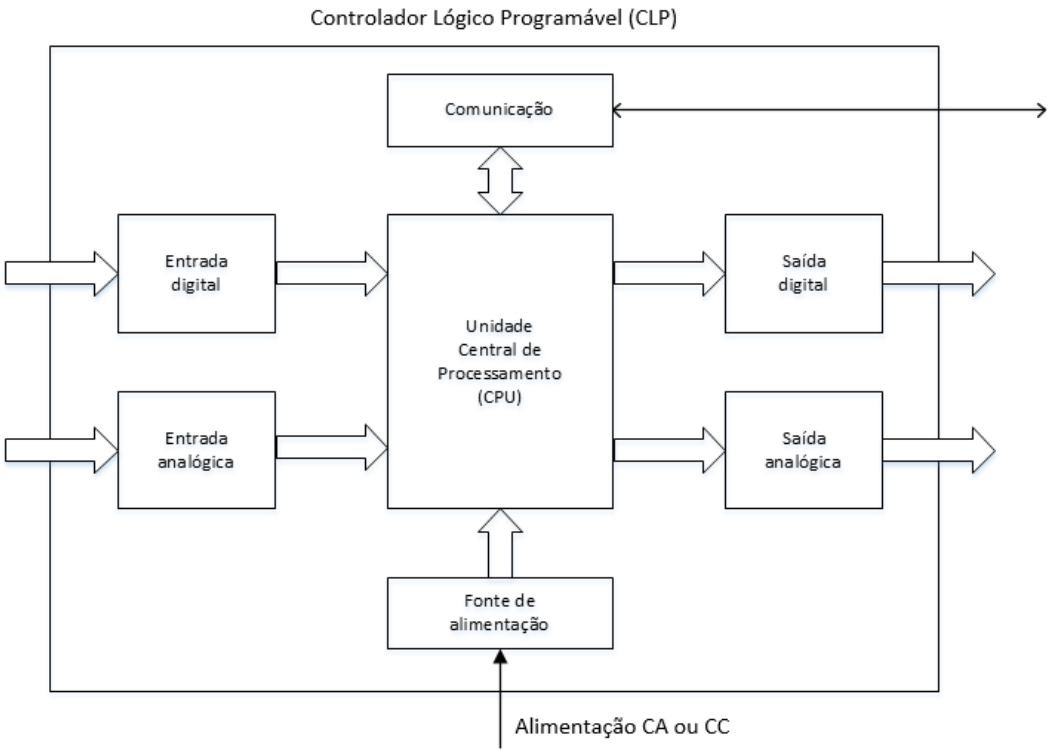


Figura 3-3: Estrutura de um CLP

A fonte de alimentação é responsável pelo fornecimento da energia necessária para a alimentação da CPU e dos módulos de entrada e de saída. Fornece todos os níveis de tensão exigidos para as operações internas do CLP. Convém lembrar que, como geralmente os CLPs são modulares, existem casos em que uma segunda fonte é necessária devido ao aumento de consumo com a expansão dos módulos. Cada fabricante especifica as condições que tornam necessária a segunda fonte. Certos modelos de CLPs são

projetados para operarem com uma tensão de alimentação de 220 V, outros trabalham com tensão de alimentação contínua de 24 V.

As memórias são divididas em duas partes: instruções do programa executivo que controla as atividades da CPU e instruções do programa de aplicação do usuário, essa última parte expansível.

3.2 Linguagens de Programação

3.2.1 Definições básicas

Genericamente falando, linguagem é um meio de transmissão de informação entre dois ou mais elementos com capacidade de se comunicarem. Esses elementos não ficam restritos aos seres humanos, nem mesmo é exclusividade dos seres vivos, já que máquinas podem ser construídas com tal capacidade.

Na área da computação, define-se *instrução* como um comando que permite a um sistema com capacidade computacional realizar determinada operação. *Linguagem de programação* é um conjunto padronizado de instruções que o sistema computacional é capaz de reconhecer.

Programar significa fornecer uma série de instruções a um sistema com capacidade computacional, de maneira que esse seja capaz de comportar-se deterministicamente, executando de forma automática as decisões de controle em função do estado atual, das entradas e das saídas do sistema num dado instante.

3.2.2 A programação em um CLP

Os elementos mais importantes de um CLP são as entradas, as saídas e a memória interna. Somente através de suas entradas o CLP recebe informações do mundo externo. De forma similar, o CLP só pode controlar algum dispositivo se este estiver conectado a uma de suas saídas.

Chamamos de *variáveis* os elementos textuais que permitem acessar diretamente as posições de memória dos CLPs. Uma posição de memória de um CLP é identificada

por três regiões lógicas. A primeira letra identifica se a variável está mapeando uma entrada, saída ou posição interna de memória, conforme a Tabela 3.1.

Tabela 3.1: Mapeamento das posições de memória de um CLP

Primeira letra	Inglês	Português
I	Inputs	Entradas
Q	Outputs	Saídas
M	Memory	Memória

No ato de sua definição, uma variável deve ser associada a uma das posições de memória mostradas na Tabela 3.1, além de ser provida de um nome simbólico e ser indicado o tipo de dado que ela armazenará. Variáveis de entrada e saída só podem ser do tipo booleano. Já as variáveis de memória podem assumir diversos tipos, como inteiros, decimais e temporizadores, dependendo do CLP. Uma vez configuradas as variáveis, resta criar o programa em uma das linguagens de programação disponíveis para o CLP.

A interface que permite a configuração e programação varia de CLP para CLP. No entanto, de maneira geral, todos dividem seus modos de operação em *programação* e *execução*. No modo de programação (Prog), o CLP não executa nenhum programa, isto é, fica aguardando para ser configurado ou receber novos programas ou modificações de programas já instalados. Esse tipo de programação é chamado de *off-line.* Já no modo de execução (Run), o CLP passa a executar o programa do usuário. CLPs de maior porte podem sofrer alterações de programa mesmo durante a execução. Esse tipo de programação é chamado de *on-line*.

O funcionamento do CLP é baseado num sistema microprocessado em que há uma estrutura de software que realiza continuamente ciclos de leitura, chamados de *scan.* O scan é constituído de três processos:

1. Efetua-se a leitura dos dados através dos dispositivos via interface de entrada;
2. Executa-se o programa de controle armazenado na memória;
3. Atualizam-se os dispositivos de saída via interface de saída.

3.2.3 As linguagens de programação

Visando atender aos diversos segmentos da indústria, incluindo seus usuários, e uniformizar as várias metodologias de programação dos controladores industriais, a norma IEC 61131-3 definiu sintática e semanticamente cinco linguagens de programação:

- **Diagrama de Blocos de Funções (*Function Block Diagram*, FBD)**

  É uma das linguagens gráficas de programação cujos elementos são expressos por blocos interligados, semelhantes aos utilizados em eletrônica digital. Essa linguagem permite um desenvolvimento hierárquico e modular do software, uma vez que podem ser construídos blocos de funções mais complexos a partir de outros menores e mais simples.

- **Sequenciamento Gráfico de Funções (*System Function Chart*, SFC)**

  O SFC é uma linguagem gráfica que permite a descrição de ações sequenciais, paralelas e alternativas existentes numa aplicação de controle. Como é descendente direto do *Grafcet*, o SFC fornece os meios para estruturar uma unidade de organização de um programa num conjunto de etapas separadas por transições. A cada etapa está associado um conjunto de ações. A cada transição está associada uma receptividade que terá de ser satisfeita para que a transposição da transição ocorra, e assim o sistema evolua para a etapa seguinte.

- **Lista de instruções (*Instruction List*, IL)**

  Inspirada na linguagem *assembly* e de característica puramente sequencial, é caracterizada por instruções que possuem um operador e, dependendo do tipo de operação, podem incluir um ou mais operandos, separados por vírgulas. É indicado para pequenos CLPs ou para controle de processos simples.

- **Texto estruturado (*Structured Text*, ST)**

  É uma linguagem textual de alto nível e muito poderosa, inspirada na linguagem Pascal, que contém todos os elementos essenciais de uma linguagem de programação moderna, incluindo as instruções condicionais (IF-THEN-ELSE e CASE OF) e instruções de iterações (FOR, WHILE e REPEAT). Como o seu nome sugere, encoraja o desenvolvimento de programação estruturada, sendo excelente para a definição de blocos funcionais complexos.

- **Linguagem Ladder (*Ladder Diagram*, LD)**

  É uma linguagem gráfica baseada na lógica de relés e contatos elétricos para a realização de circuitos de comandos e acionamentos. Por ser a primeira linguagem utilizada pelos fabricantes, é a mais difundida e encontrada em quase todos os CLPs da atual geração.

Uma metodologia sistemática de conversão de Redes de Petri em Diagramas Ladder foi proposta em [1]. Essa metodologia foi uma das principais razões para o desenvolvimento do PETRILab, que conta com o recurso de fazer essa conversão automaticamente. Por esse motivo, entraremos em mais detalhes da linguagem Ladder neste trabalho.

3.3 A Linguagem Ladder

3.3.1 Conceitos básicos

A linguagem Ladder foi a primeira a surgir para CLPs. Para que obtivesse uma aceitação imediata no mercado, seus projetistas consideraram que ela deveria evitar uma mudança de paradigma muito brusca. Considerando que, na época, os técnicos e engenheiros eletricistas eram normalmente os encarregados da manutenção no chão de fábrica, a linguagem Ladder deveria ser algo familiar para esses profissionais. Assim, ela foi desenvolvida com os mesmos conceitos dos diagramas de comandos elétricos que utilizam bobinas e contatos.

3.3.2 Lógica de contatos

O diagrama Ladder segue basicamente a chamada lógica de contatos, que é baseada na condução de corrente elétrica em um circuito com chaves. Uma chave fechada permite a passagem de corrente elétrica, enquanto uma chave aberta não. A saída do circuito é um elemento que deve ser energizado ou não, dependendo do estado das chaves. Um exemplo de circuito com uma chave aberta pode ser visto na Figura 3-4; um exemplo com uma chave fechada é mostrado na Figura 3-5. Nesses circuitos, a saída é uma lâmpada, que pode estar acesa ou apagada.

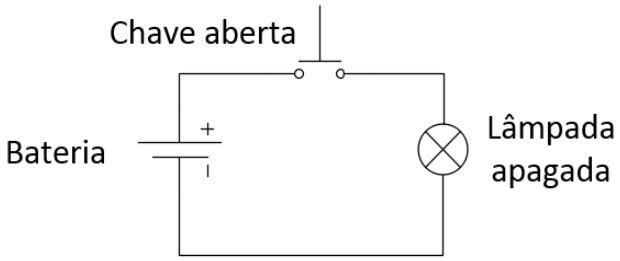


Figura 3-4: Circuito elétrico com chave aberta

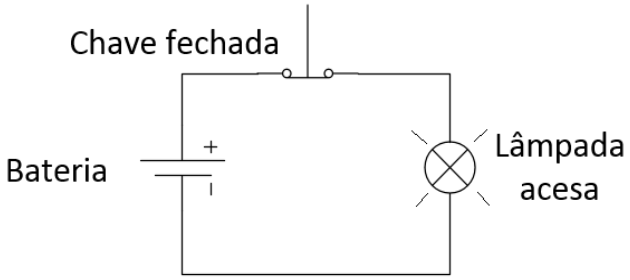


Figura 3-5: Circuito elétrico com chave fechada

3.3.3 Símbolos básicos

A simbologia utilizada em um diagrama Ladder não é única: diferentes fabricantes de CLPs utilizam diferentes símbolos para cada elemento do diagrama. A indústria caminha em direção à adoção da norma IEC 61131-3, que padroniza essa simbologia, mas, como ainda não é obrigatória, grandes fabricantes como Siemens, Allen-Bradley, Schneider Electric ainda não a aderiram. Neste trabalho, no entanto, utilizaremos apenas a simbologia proposta pela norma.

Um dos dois elementos mais básico de um diagrama Ladder é o *contato*, cujo símbolo pode ser visto na Tabela 3.2. Ele funciona como uma chave em um circuito elétrico, com a exceção de que a corrente só flui da esquerda para a direita. Esse comportamento será discutido mais à frente.

Os contatos são associados a variáveis booleanas configuradas previamente no CLP. Um contato normalmente aberto (NA) conduzirá caso sua variável associada tiver valor lógico 1, e não conduzirá caso ela tenha valor lógico 0. Já o contato normalmente

fechado (NF) tem funcionamento contrário: conduz caso sua variável tenha valor lógico 0, e não conduz caso ela tenha valor lógico 1.

Tabela 3.2: Simbologia de contatos

Contato Normalmente Aberto (NA)	Contato Normalmente Fechado (NF)
	

O outro elemento básico de um diagrama Ladder é a chamada *bobina*. Assim como o contato, a bobina deve ser associada a uma variável pré-configurada no CLP. No entanto, diferentemente do contato, ela não lê o valor da variável, e sim, o define. A Tabela 3.3 mostra os símbolos da bobina de acordo com a norma.

Tabela 3.3: Simbologia de bobinas

Bobina	Bobina Negada
	

Uma bobina fica energizada, ou seja, define o nível lógico de sua variável associada como 1, quando há um fluxo de energia virtual chegando nela. Caso contrário, a variável tem nível lógico 0. Apesar da bobina negada ser raramente utilizada - inclusive não estando disponível em alguns fabricantes de CLPs - ela é prevista pela norma. Seu funcionamento é o inverso da bobina normal: fica energizada quando não há fluxo virtual de energia chegando nela, e desenergizada quando há fluxo.

3.3.4 Diagramas de contatos em Ladder

Sendo conhecidos os elementos básicos de um diagrama Ladder, podemos prosseguir com a apresentação do diagrama em si. A função principal de um programa na linguagem Ladder é controlar o acionamento de saídas, dependendo da combinação lógica dos contatos de entrada. O diagrama de contatos Ladder é composto de duas barras verticais que representam os polos positivos e negativos de uma bateria. A linha à esquerda representa o polo positivo, e, a da direita, o negativo. A ideia por trás dessa linguagem é representar um fluxo virtual de eletricidade entre duas barras energizadas.

Conforme dito na seção anterior, as variáveis devem ser previamente criadas no CLP, e associadas à entrada (I), à saída (Q) ou à memória (M) do mesmo. Variáveis associadas à entrada são lidas diretamente do estado dos contatos físicos do CLP (abertos ou fechados); sendo assim, seu valor não pode ser modificado pelo programa, apenas lido. Já as variáveis de saída comandam os relés físicos da saída do CLP: abrindo-os ou fechando-os; sendo assim, seu valor não pode ser lido, apenas modificado. As variáveis de memória, por sua vez, podem ser lidas e modificadas.

Um exemplo simples de diagrama Ladder pode ser visto na Figura 3-6, onde as variáveis X, Y e W são variáveis de entrada, F é variável de saída e Z é variável de memória. A bobina Z só será energizada caso o contato X esteja no nível lógico 1 e o contato Y no nível lógico 0, o que corresponde aos contatos físicos de entrada associados do CLP estarem fechado e aberto, respectivamente. Caso isso ocorra, o contato Z será considerado fechado (conduzindo), e basta que o contato de entrada W se feche para que a bobina de saída F seja energizada. Nesse caso, o relé de saída correspondente no CLP será fechado.

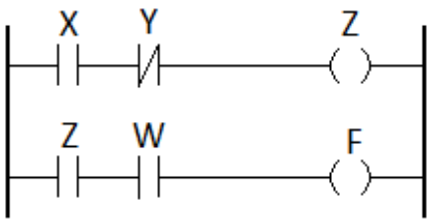


Figura 3-6: Exemplo de diagrama Ladder

As linhas do diagrama são processadas sequencialmente pelo CLP, de cima para baixo, num processo chamado de *varredura*. Um ciclo de varredura completo do diagrama leva de 1 a 20 milissegundos [4]. Em geral, o tempo de varredura é muito menor que o da comutação das entradas do CLP. No entanto, nos casos raros de um contato comutar duas vezes dentro desse intervalo de tempo, o programa pode não perceber a primeira comutação, causando resultados inesperados. Devido ao processo sequencial de varredura, a ordem das linhas no diagrama também deve ser levada em conta no ato da programação.

O diagrama Ladder se assemelha muito a um circuito elétrico, mas não podemos tratá-lo inteiramente como tal. O chamado *fluxo reverso* não é permitido em Ladder, ou seja, o fluxo elétrico virtual deve sempre ir da esquerda para a direita de um contato. A Figura 3-7 ilustra essa situação. Mesmo que os contatos F, D, B e C estejam fechados, a bobina Y não será energizada, pois houve tentativa de fluxo reverso no contato D.

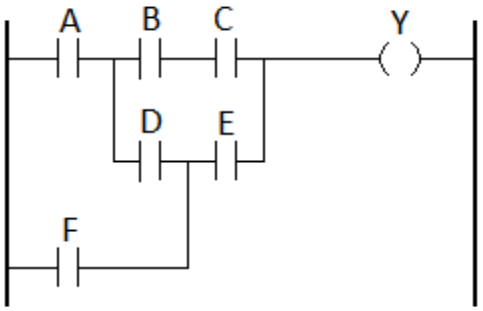


Figura 3-7: Diagrama Ladder com fluxo reverso

3.3.5 Outros elementos de diagramas Ladder

Os contatos e bobinas são os elementos base de qualquer diagrama Ladder. No entanto, muitos CLPs implementam outros elementos e blocos que podem realizar funções mais complexas. No decorrer deste trabalho, utilizaremos alguns desses blocos, sendo assim, explicaremos aqui suas funções e formas de utilização. São eles:

- **Contatos de borda de subida e descida**

  Além dos contatos comuns apresentados previamente, existem dois outros tipos muito utilizados pelos programadores de Ladder: os contatos de *borda de subida* e *borda de descida*. Um contato de borda de subida é fechado (conduz) apenas no ciclo de varredura em que a condição booleana anterior a ele muda de 0 para 1. Nos ciclos seguintes, o contato já estará aberto. O contato de borda de descida é o oposto: será fechado apenas no ciclo de varredura em que a condição booleana anterior a ele mudar de 1 para 0. A Tabela 3.4 mostra a simbologia desse tipo de contato.

Tabela 3.4: Contatos de borda de subida e descida

Contato de Borda de Subida	Contato de Borda de Descida
⊣P⊢	⊣N⊢

- **Bobinas de Set e Reset**

Nas bobinas tradicionais, suas variáveis associadas só permanecem no estado lógico 1 enquanto elas estão energizadas; caso sejam desenergizadas, a variável volta para o estado lógico 0. As bobinas *Set*, por sua vez, colocam a variável no estado lógico 1 ao serem energizadas, e não modificam seu estado lógico ao serem desenergizadas! Por outro lado, a bobina *Reset* modifica a variável para o estado lógico 0 ao ser energizada, e não modifica seu estado ao ser desenergizada. A Tabela 5 mostra a simbologia dessas bobinas.

Tabela 3.5: Bobinas Set e Reset

Bobina Set	Bobina Reset
-(S)-	-(R)-

- **Blocos COMP**

Blocos COMP são utilizados para comparação entre variáveis e números inteiros. As comparações podem ser do tipo $=, >, \geq, <$ *e* $\leq$. Caso a comparação seja verdadeira, o bloco age como um contato fechado, conduzindo; caso contrário, age como um contato aberto. A Figura 3-8 mostra a simbologia de um bloco COMP.

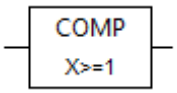


Figura 3-8: Bloco COMP

- **Blocos MOVE**

Os blocos do tipo MOVE são usados para definir o valor de variáveis inteiras. Ao ser energizado, ele define sua variável com o valor configurado. A Figura 3-9

mostra a simbologia de um bloco MOVE. O bloco deve ser inserido no final da linha.

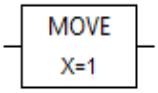
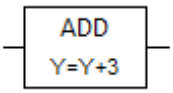
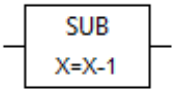


Figura 3-9: Bloco MOVE

- **Blocos ADD e SUB**

  O bloco ADD incrementa o valor de uma variável inteira em uma certa quantia. Já o bloco SUB faz o oposto: subtrai o valor de uma variável inteira em uma certa quantia. A Tabela 3.6 mostra a simbologia de ambos os blocos. Estes blocos devem ser inseridos no final da linha.

Tabela 3.6: Blocos ADD e SUB

Bloco ADD	Bloco SUB
	

- **Bloco TIMER**

  O bloco TIMER representa o temporizador *Timer On Delay* dos diversos fabricantes de CLPs. São associados a ele uma variável e um valor de tempo. Ao ser energizado, o bloco inicia um contador, e quando esse atinge o tempo configurado, a variável configurada assume o valor lógico 1. Se o bloco for desenergizado em qualquer momento, o contador zera e a variável configurada assume o valor lógico 0. A Figura 3-10 mostra a simbologia do bloco TIMER.

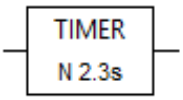


Figura 3-10: Bloco TIMER

Capítulo 4

Conversão de Redes de Petri Interpretadas para Controle em Diagramas Ladder

Vimos que as Redes de Petri Interpretadas para Controle são excelentes modelos para Sistemas a Eventos Discretos na área de automação, tendo alta capacidade de modelagem e controle; no entanto elas são apenas modelos teóricos, não podendo ser utilizadas diretamente no controle de máquinas reais. Os diagramas Ladder, por sua vez, podem ser implementados em qualquer CLP, podendo ser utilizados para o controle de máquinas; no entanto, não é fácil obter um diagrama Ladder diretamente de um SED. Objetivando estabelecer um elo entre teoria e prática, Moreira e Basílio [1] propõem um método sistemático para a conversão de RPICs em diagramas Ladder. Este capítulo destina-se a apresentar esse método, estando estruturado da seguinte forma: na seção 4.1 será mostrada a estrutura geral do método proposto; nas seções de 4.2 a 4.6 serão mostrados os diferentes módulos que compõem o diagrama gerado; na seção 4.7 será tratado sobre a organização do diagrama gerado; por fim, na seção 4.8 será explicitado o tamanho final do diagrama gerado.

4.1 Descrição Geral do Método Proposto

No método proposto por Moreira e Basílio, o diagrama Ladder é dividido em cinco módulos:

- Módulo M1, associado a identificação da ocorrência de eventos externos
- Módulo M2, associado com as condições para o disparo das transições
- Módulo M3, que descreve a evolução das fichas na Rede de Petri
- Módulo M4, que representa a inicialização da Rede de Petri, isto é, define sua marcação inicial
- Módulo M5, que define quais ações serão ativadas no estado atual do sistema

Nas subseções seguintes, serão apresentadas explicações detalhadas sobre cada um dos módulos, e será apresentado um exemplo para ilustrar o método de conversão proposto. O grafo da Rede de Petri para o exemplo é o mostrado na Figura 4-1.

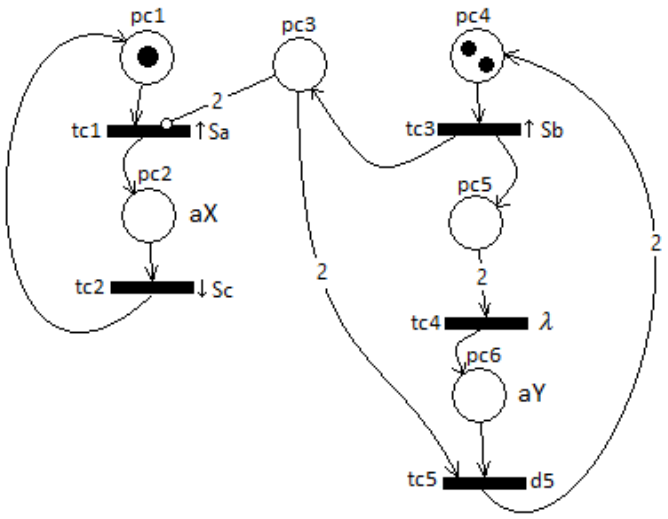


Figura 4-1: RPIC utilizada para ilustrar o método de conversão proposto em [1]

4.2 Módulo dos Eventos Externos

Eventos externos são associados com as bordas de subida ou descida dos sinais dos sensores na RPIC, e podem ser detectados utilizando-se contatos de borda de subida ou de descida. Como já foi dito, esse tipo de contato fecha por apenas um ciclo, quando a

variável muda seu valor de 0 pra 1 (contato P), ou de 1 para 0 (contato N). O evento sempre ocorrente (λ) não é representado no módulo de eventos externos, já que é um evento interno.

Na Rede de Petri da Figura 4-1, existem quatro eventos sincronizando com as transições: dois eventos associados às bordas de subida dos sinais dos sensores, $\uparrow S_a$ e $\uparrow S_b$, um evento associado com a borda de descida do sinal de um sensor, $\downarrow S_c$, e o último evento é o evento sempre ocorrente (λ). Portanto, o módulo de eventos externos para esta RPIC deve ter três linhas, como mostra a Figura 4-2. A primeira e a segunda linha do diagrama Ladder levam em conta os eventos externos associados à borda de subida dos sinais dos sensores $\uparrow S_a$ e $\uparrow S_b$, respectivamente. Quando, por exemplo, S_a muda seu valor de zero para um, o contato P fecha por um ciclo de varredura, energizando a bobina denominada S_{ar}, que representa a borda de subida de S_a, $\uparrow S_a$ (o r vem de *rising*, subida em inglês). A terceira linha leva em conta a borda de descida de S_c. Quando S_c muda seu valor de um para zero, o contato N fecha, energizando a bobina S_{cf}, que corresponde à borda de descida de S_c, $\downarrow S_c$ (o f vem de *falling*, descida em inglês).

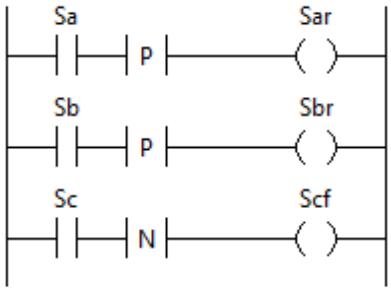


Figura 4-2: Módulo dos eventos externos da RPIC exemplo

4.3 Módulo das Condições de Disparo

O módulo das condições de disparo tem $|T_C|$ linhas, em que $|.|$ denomina cardinalidade. Cada linha descreve as condições para o disparo da transição $t_{Cj} \in T_C$. Uma vez que cada RPIC é uma Rede de Petri Rotulada Estendida, então uma transição t_{Cj} está habilitada se, e somente se

$$x_C(p_{Ci}) \geq Pre_C(P_{Ci}, t_{Cj}), \forall p_{Ci} \in I(t_{Cj}) \quad (7)$$

e

$$x_C(p_{Ci}) < In_C(p_{Ci}, t_{Cj}), \forall In_C(p_{Ci}, t_{Cj}) > 0 \quad (8)$$

Se a transição $t_{Cj} \in T_C^0$, então t_{Cj} estará em condições de disparar se as condições (7) e (8) forem satisfeitas e a condição associada c_j é verdadeira, e será disparada quando o evento associado e_j ocorrer. Por outro lado, se $t_{Cj} \in T_C^D$, então t_{Cj} estará em condições de disparar se as condições (7) e (8) forem satisfeitas, mas é disparada apenas depois de um tempo de atraso d_j.

As condições de habilitação (7) e (8) podem ser facilmente expressas no diagrama Ladder usando instruções de comparação, as quais são conectadas em série com outros elementos que dependem da transição t_{Cj} ser temporizada ou não. Se a transição não for temporizada, a expressão booleana para a condição c_j é implementada com uma associação simples de contatos NA ou NF. Essa associação é conectada em série com o contato NA que representa a borda de subida ou descida do sinal de sensor correspondente que observa e_j. Quando todas as condições para o disparo de $t_{Cj} \in T_C^0$ forem satisfeitas, uma bobina associada com a variável binária B_j é energizada para representar que t_{Cj} está pronta para disparar. É importante salientar que se $c_j = 1$ ou $e_j = \lambda$, então nenhum contato é adicionado para representar a condição ou evento. Por outro lado, se $t_{Cj} \in T_C^D$, então um temporizador com valor preconfigurado igual ao tempo de atraso d_j deve ser utilizado. Passado o tempo de atraso, um temporizador energiza uma bobina TDN_j indicando que o tempo de atraso passou.

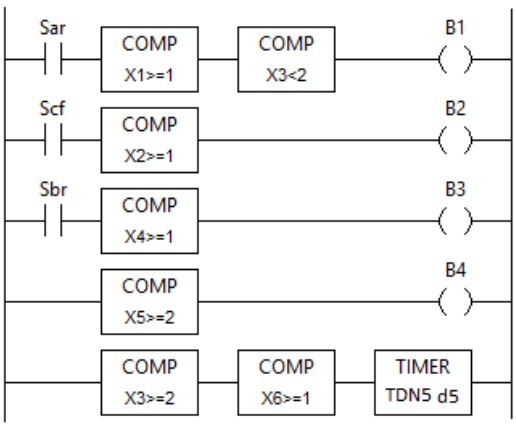


Figura 4-3: Módulo das condições de disparo da RPIC exemplo

A Figura 4-3 mostra o diagrama Ladder para o módulo das condições de disparo da RPIC da Figura 4-1. Perceba que as transições t_{Cj}, para $j = 1, \dots, 4$ não são temporizadas, mas a transição t_{C5} é temporizada. Considere, por exemplo, a transição não temporizada t_{C1}. Perceba que a transição t_{C1} é habilitada se $x_C(p_{C1}) \geq 1$ e $x_C(p_{C3}) < 2$, e t_{C1} dispara quando $\uparrow S_a$ ocorre. As condições de habilitação são representadas na primeira linha da Figura 4-3 com duas instruções de comparação associadas com a marcação dos lugares p_{C1} e p_{C3}, representadas pelas variáveis inteiras X_1 e X_3, respectivamente, e a ocorrência de $\uparrow S_a$ é verificada usando um contato NA associado com a variável binária S_{ar}. As condições para o disparo da transição temporizada t_{C5} são representadas na quinta linha da Figura 4-3. Perceba que o temporizador é energizado apenas depois da habilitação das condições $x_C(p_{C3}) \geq 2$ e $x_C(p_{C6}) \geq 1$ ocorrer, o que é representado no diagrama Ladder por $X_3 \geq 2$ e $X_6 \geq 1$.

Vale notar que, para lugares seguros, é possível substituir os blocos COMP por contatos, pois suas marcações só poderão valer zero ou um. As condições de (7) são então representadas por contatos NF, e as condições de (8) são representadas por contatos NA. Essa simplificação pode permitir o uso do diagrama gerado em CLPs antigos que não possuem blocos COMP.

4.4 Módulo das Dinâmicas da Rede de Petri

Após o disparo de uma transição t_{Cj}, o número de fichas na Rede de Petri precisa ser atualizado. Esse processo é realizado no diagrama Ladder pelo módulo das dinâmicas da Rede de Petri. Este módulo tem $|T_C|$ linhas. Cada linha é associada a uma transição $t_{Cj} \in T_C$ e expressa as mudanças na marcação dos lugares após o disparo de t_{Cj}. Um contato NA, associado com a variável binária B_j ou com a bobina do temporizador TDN_j, é usado para representar que a transição t_{Cj} está pronta para disparar. Funções matemáticas são usadas em série com o contato NA para representar as mudanças nas marcações dos lugares de entrada e saída de t_{Cj}. A função de subtração (SUB) é usada para variáveis associadas com os lugares de entrada p_{Ci} de t_{Cj}, e o peso $Pre_C(p_{Ci}, t_{Cj})$ é subtraído da variável inteira X_i que representa o número de fichas de p_{Ci}. A função de adição (ADD) é usada para variáveis associadas com os lugares de saída p_{Co} de t_{Cj}, e o peso

$Post_C(t_{Cj}, p_{Co})$ é adicionado na variável inteira X_o que representa o número de fichas no lugar de saída p_{Co}.

A Figura 4-4 mostra o diagrama Ladder do módulo das dinâmicas da Rede de Petri mostrada na Figura 4-1. Cada lugar de entrada de uma transição é associado com um bloco SUB e cada lugar de saída é associado com um bloco ADD. Considere, por exemplo, a transição t_{C2}. Ela tem um arco de entrada de peso um vindo de p_{C2}, e um arco de saída de peso um chegando em p_{C1}; desse modo, a segunda linha do módulo das dinâmicas da Rede de Petri conterá um contato da bobina B_2 em série com uma combinação paralela de um bloco SUB removendo uma ficha de p_{C2} com um bloco ADD adicionando uma ficha a p_{C1}.

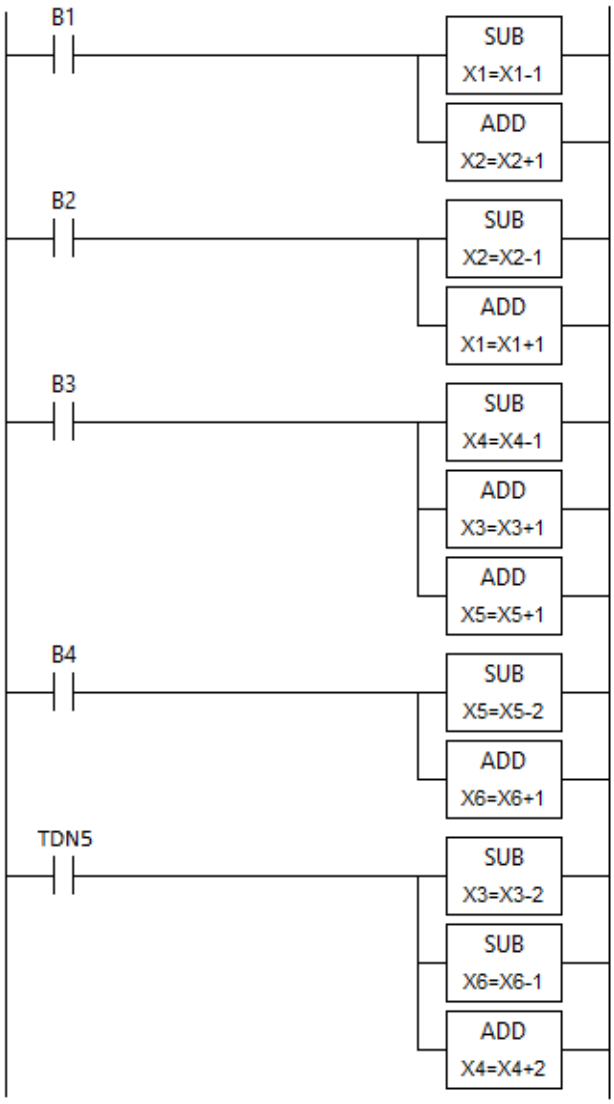


Figura 4-4: Módulo das dinâmicas da Rede de Petri da RPIC exemplo

Caso um determinado lugar seja seguro, sabemos que sua marcação só pode valer zero ou um. Nesse caso, os blocos ADD e SUB podem ser substituídos por bobinas SET e RESET, respectivamente. Essa é outra medida que pode aumentar a compatibilidade com os diversos modelos de CLP.

4.5 Módulo da Inicialização

O módulo da inicialização contém apenas uma linha formada por um contato NF associado com uma variável interna B_0 que, no primeiro ciclo de varredura, energiza logicamente blocos MOVE associados com lugares que têm fichas na marcação inicial. Após o primeiro ciclo de varredura, o contato NF é aberto. Vale salientar que não é necessário definir o valor zero para variáveis associadas a lugares sem nenhuma marcação inicial, já que as variáveis são automaticamente inicializadas com zero.

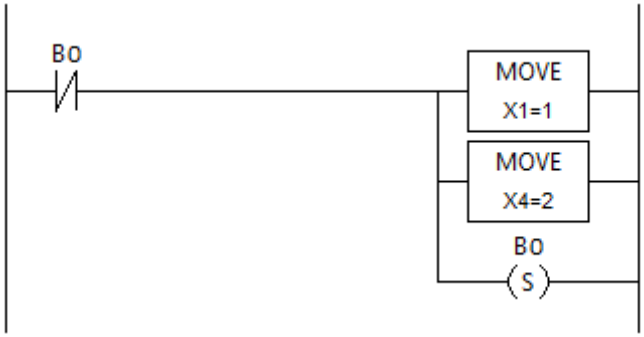


Figura 4-5: Módulo da Inicialização da RPIC exemplo

Novamente, para aumentar a compatibilidade do diagrama Ladder com os diversos modelos de CLP, podemos substituir os blocos MOVE por bobinas SET, caso os lugares sejam seguros.

4.6 Módulo das Ações

O número de linhas no módulo das ações é igual ao número de lugares com ações associadas na RPIC. Ações impulsionais devem ser executadas apenas quando a marcação de um lugar muda de zero para um. Para implementar esse comportamento, uma instrução de comparação em série com um contato P é usado para verificar se a

marcação do lugar que tem uma ação associada muda seu valor lógico de zero para um. Se esse for o caso, uma bobina é logicamente energizada e a ação impulsional é executada.

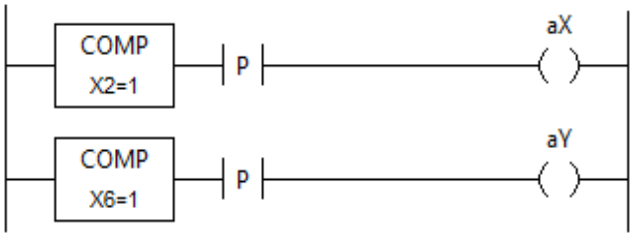


Figura 4-6: Módulo das ações da RPIC exemplo

O módulo das ações para o exemplo da Figura 4-1 é mostrado na Figura 4-6. Perceba que o diagrama Ladder do módulo das ações tem apenas duas linhas, já que a RPIC tem apenas dois lugares com ações associadas.

Para ações contínuas, a linha correspondente será apenas um bloco COMP com a bobina de saída em série, pois ela precisa estar energizada durante todo o tempo em que a marcação do lugar associado for um. Caso a ação aX do exemplo fosse uma ação contínua, o diagrama correspondente seria o da Figura 4-7.

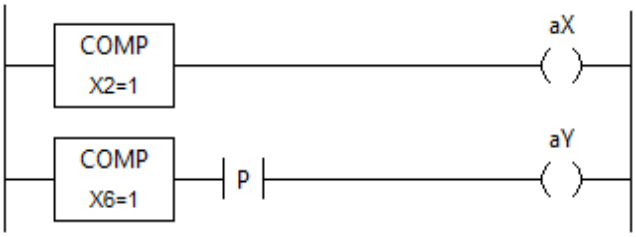


Figura 4-7: Módulo das ações com ação contínua

Vale salientar que os blocos COMP deste módulo sempre podem ser substituídos por contatos NF, pois foi definido anteriormente que as ações só podem estar associadas a lugares seguros.

4.7 A Organização do Diagrama Ladder

Os cinco módulos devem ser implementados na mesma ordem apresentada neste capítulo, ou seja: (i) módulo dos eventos externos; (ii) módulo das condições de disparo; (iii) módulo das dinâmicas da Rede de Petri; (iv) módulo da inicialização; e (v) módulo das ações.

A ordem dos módulos do código Ladder é importante para evitar o efeito avalanche [1], e também para garantir que as ações definidas na marcação inicial sejam executadas. O efeito avalanche é evitado porque as condições para o disparo de todas as transições são verificadas primeiro no módulo das condições de disparo, e, somente depois disso, a evolução das fichas são dadas no módulo das dinâmicas da Rede de Petri. Esse esquema de implementação garante que cada marcação da RPIC (mesmo marcações instáveis) continuam inalteradas por pelo menos um ciclo de varredura na sua implementação em Ladder. Portanto, apenas transições habilitadas poderão disparar quando o evento associado ocorrer. Outro benefício da organização da estrutura dos módulos proposta em [1] é que ela permite as ações associadas aos lugares seguros de marcações instáveis da RPIC serem executadas corretamente.

É importante notar que as ações associadas aos lugares seguros que tem uma ficha na marcação inicial devem ser executadas. Para garantir esse comportamento, o módulo de inicialização é implementado depois dos módulos de condições de disparo e dinâmicas da Rede de Petri. Portanto, se um lugar seguro com ações associadas é marcado com uma ficha no módulo de inicialização, as ações associadas são executadas como descrito no módulo das ações.

Por fim, vale ressaltar que, apesar do método proposto, em geral, gerar um código Ladder maior do que outros métodos propostos na literatura, ele garante que o comportamento esperado da RPIC será executado por sua implementação em Ladder.

4.8 Tamanho do Diagrama Ladder

Considerando que existem l eventos externos distintos associados com a borda de subida ou descida de sinais de sensores, então o máximo número de linhas no diagrama Ladder obtido por este método é $(l + 2|T_C| + 1 + |P_C|)$. Apesar do número de linhas poder ser menor, o diagrama Ladder proposto permite uma completa visualização da estrutura da Rede de Petri, e imita seu comportamento. Assim, qualquer modificação no Controlador a Eventos Discretos descrito pela RPIC pode ser facilmente implementada no diagrama Ladder existente.

Capítulo 5

O PETRILab

Apresentamos todos os conceitos necessários para o entendimento do funcionamento do PETRILab. Vamos neste capítulo apresentar o programa em si, sua arquitetura básica, sua interface, seus recursos e sua forma de utilização. Este capítulo está organizado da seguinte forma: na seção 5.1 será apresentada uma breve história do programa; na seção 5.2 será mostrada uma visão geral de seus recursos; a seção 5.3 contém o guia de utilização do usuário; por fim, na seção 5.4, é exibida uma lista de teclas de atalho do programa.

5.1 Breve História

A elaboração do PETRILab começou no segundo semestre de 2013, como um projeto de iniciação científica do Laboratório de Controle e Automação (LCA) da Universidade Federal do Rio de Janeiro (UFRJ). A proposta era que fosse desenvolvido um meio automático de realizar a conversão proposta no Capítulo 4 deste trabalho: a conversão de uma RPIC em um diagrama Ladder.

A linguagem de programação escolhida para a criação do programa foi o Python, por ser uma linguagem orientada a objetos, simples e robusta. Em suas primeiras versões, o PETRILab não contava com interface gráfica; a modelagem de Redes de Petri era feita por linha de comando, assim como a execução de eventos e mudança de condições. Uma pequena parte do código da classe de RPIC criada no Python pode ser vista na Figura 5-1. Não é o objetivo deste trabalho explicar o código completo do programa – já que o

programa completo tem mais de 3.000 linhas. No entanto, o Apêndice A explica a arquitetura básica do programa, para permitir futuro desenvolvimento por outra pessoa.

```
class CIPN():
    def __init__(self, Pc, Tc, Prec, Postc, x0c, Inc, C, Ec, lc, D, ld, A, la, parent=None):
        self.parent = parent
        self.x0c = x0c
        self.Pc, self.Tc = Pc, Tc
        self.Prec, self.Postc, self.Inc = Prec, Postc, Inc
        self.C, self.Ec, self.lc, self.D = C, Ec, lc, D
        self.ld, self.A, self.la = ld, A, la
        self.x, self.toFire = x0c, [False]*Tc
        self.timers = []
        for t in range(Tc):
            self.timers.append([])
        self.eimpulse = [1]*len(Ec)
    def setImpulse(self, e, x):
        self.eimpulse[e]=x
    def isFireable(self, t):
        for i in range(self.Pc):
            if self.x[i]<self.Prec[i][t]:
                return False
            if self.Inc[i][t]:
                if self.x[i]>=self.Inc[i][t]:
                    return False
        if not self.C[self.lc[t][0]] or not self.Ec[self.lc[t][1]]:
            return False
        return True
    def setFireable(self):
```

Figura 5-1: Parte do código do PETRILab

Na versão inicial, para criar uma RPIC era necessário criar um objeto da classe CIPN, que tem métodos para a exibição, execução de eventos e mudança de condições da rede. A Figura 5-2 mostra como era feita a criação de uma rede simples com: dois lugares p_1 e p_2; uma transição t_1; um arco de p_1 a t_1 e outro de t_1 a p_2, ambos com peso 1; um evento e_1 sincronizado com a transição t_1; e marcação inicial [2 0]. Ao disparar o evento de índice 0 (e_1), vemos que a marcação da rede muda para [1 1], como esperado.

Posteriormente, mesmo ainda não tendo sido feita uma interface gráfica, foi implementada a função de conversão em diagramas Ladder de acordo com o método proposto no Capítulo 4 deste trabalho. Um trecho do código de conversão pode ser visto na Figura 5-3; ele converte a rede em objetos do diagrama – criados em um outro módulo –, passando como argumento os rótulos correspondentes.

O diagrama Ladder podia então ser exibido "desenhado em texto", conforme mostra a Figura 5-4. A formatação era precária, e o texto gerado não muito intuitivo, mas o diagrama correspondia ao método de conversão proposto. Na figura, [ ] representa um contato; ( ) representa uma bobina, que terá um S dentro se for do tipo Set; < > representa um bloco de função; e || representa itens paralelos.

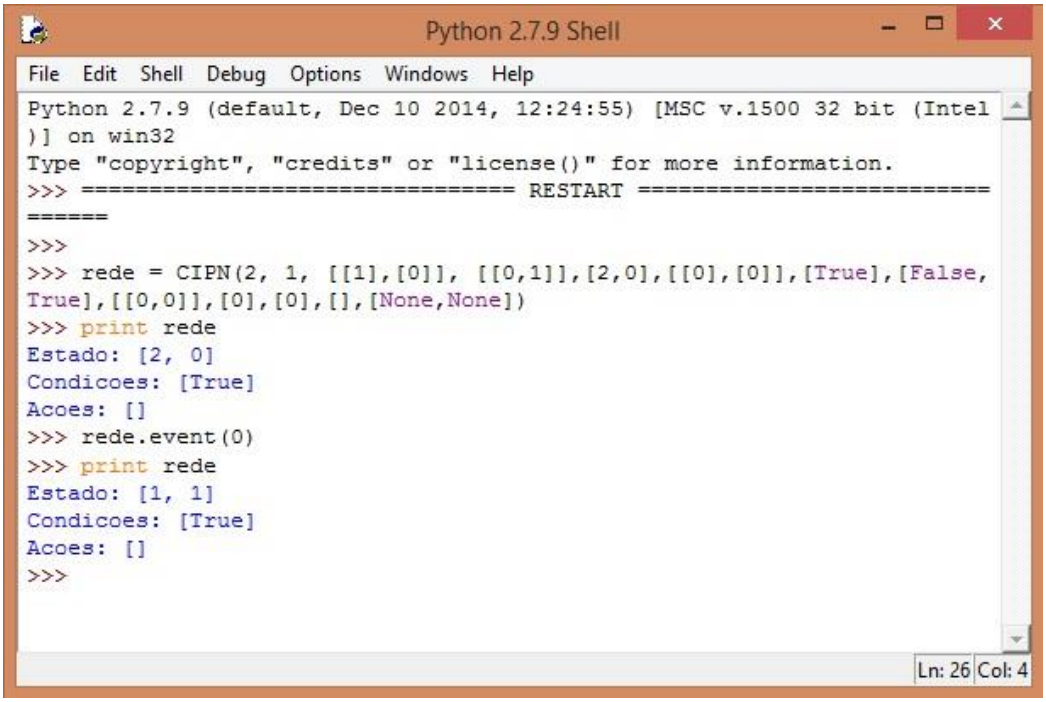


Figura 5-2: Criação de RPIC em uma versão antiga do PETRILab

```
def convert(pn, labels=None):
    # Modulo de eventos externos
    m1 = Ladder()
    l = Linha()
    for i in range(len(pn.Ec)-1):
        if True in map(lambda x: x[1]==i, pn.Ic):
            if labels:
                l += Contato(labels[2][i], 1)
            else:
                l += Contato('S'+str(i), 1)
            if pn.eimpulse[i] == 1:
                l += Borda(1)
                if labels:
                    l += Bobina(labels[2][i]+'r', 1)
                else:
                    l += Bobina('S'+str(i)+'r', 1)
            else:
                l += Borda(0)
```

Figura 5-3: Trecho de código de conversão de RPIC em diagramas Ladder

```
>>> ladder = convert(rede)
>>> for i in ladder:
        print i


|--[S0]----[P]----(S0r)--|
|--[S0r]----<COMP X0>=1>----(B0)--|
|--[B0]----(<SUB X0-=1>||<ADD X1+=1>)--|
|--[*B*]----(<MOVE X0=2>||(S-B))--|

>>>
```

Figura 5-4: Ladder gerado em forma textual

Em meados de 2014 o programa começou a ganhar uma interface gráfica, que foi finalizada no final do ano. Ele será apresentado em detalhes nas seções seguintes.

5.2 Visão Geral

O PETRILab é um programa de modelagem e simulação de Redes de Petri Interpretadas para Controle, que conta com o recurso de conversão automática em diagramas Ladder. Para utilizá-lo, o usuário deve ter a versão 2 do Python instalada em seu computador, que pode ser baixada gratuitamente em http://www.python.org/. Atenção: o programa não é compatível com a versão 3 do Python, pois diversas rotinas são alteradas em relação à versão 2. O site ainda desenvolve as duas versões em separado, basta baixar o lançamento mais recente da segunda versão.

Algumas de suas principais características são:

- Interface gráfica simples e intuitiva
- Simulação completa da rede, com o controle da execução de eventos e mudança de condições
- Suporte a transições temporizadas
- Indicadores de execução de ações impulsionais
- Proteção contra travamentos em loopings infinitos
- Conversão instantânea de RPICs em diagramas Ladder
- Possibilidade de salvar imagens da RPIC e do diagrama
- Teclas de atalho para facilitar a inserção e edição de RPICs

Com essas características, o programa se mostra uma plataforma completa para a modelagem de RPICs. A interface gráfica, combinada com teclas de atalho, permite que

as redes sejam desenhadas muito rapidamente. A rede criada através da linha de comando na Figura 5-2 pode ser desenhada facilmente com alguns cliques, resultando na rede mostrada na Figura 5-5.

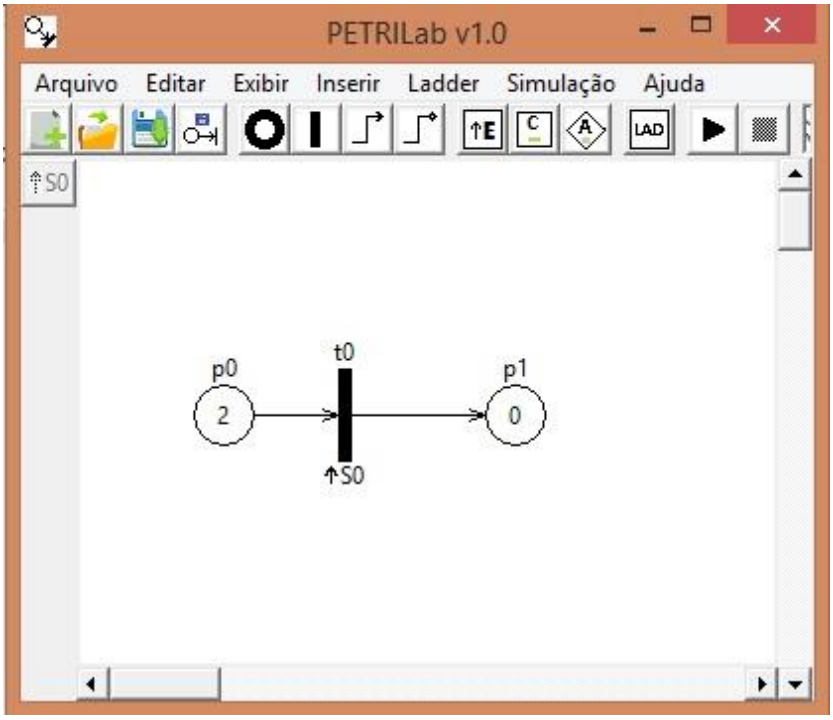


Figura 5-5: Rede de Petri da Figura 5-2 desenhada

Clicando no botão *LAD*, mostrado na barra de ferramentas da Figura 5-5, ou apertando a tecla de atalho <*G*>, uma nova janela com o diagrama é imediatamente exibida, como mostra a Figura 5-6. Podemos ver que o diagrama corresponde ao gerado textualmente na Figura 5-4, ambos de acordo com o método de conversão proposto.

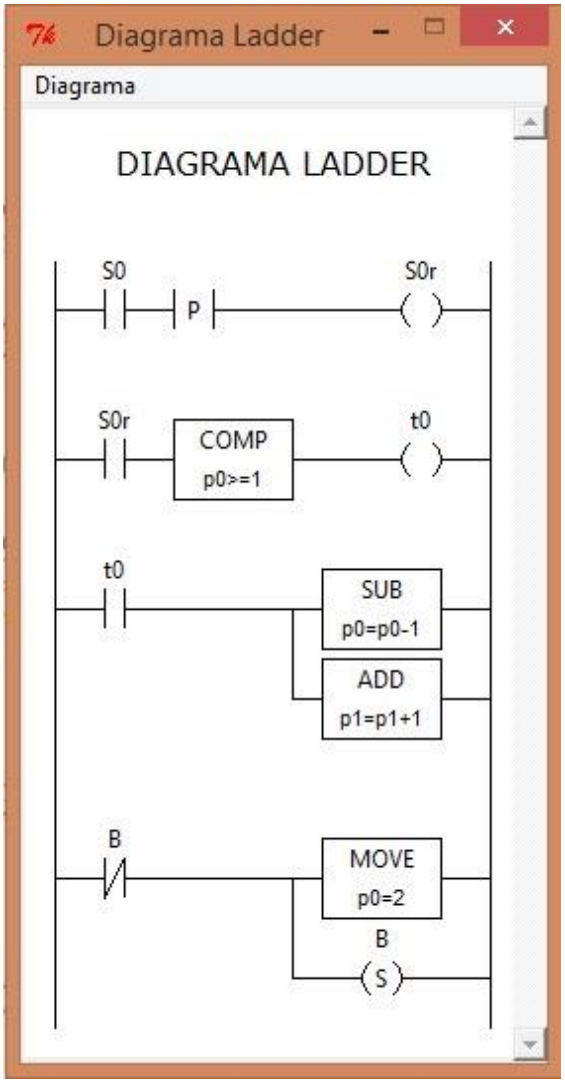


Figura 5-6: Diagrama Ladder gerado a partir da RPIC da Figura 5-5

5.3 Guia do Usuário

5.3.1 Download e instalação

O projeto do PETRILab está hospedado no site do Sourceforge [5], sob o endereço https://sourceforge.net/projects/petrilab/. Na página inicial do projeto existe um botão de Download, que aponta para a versão mais recente do programa, como mostra a Figura 5-7. Clique no botão para baixá-la.

Figura 5-7: Página do PETRILab no Sourceforge

O arquivo está compactado no formato *.zip*, e é necessário descompactá-lo. Se o sistema não tiver um descompactador nativo, é necessário baixar um, como o WinRAR [6]. A extração do arquivo é ilustrada na Figura 5-8.

Após a extração, será criada uma pasta contendo os arquivos do programa. Para executá-lo, basta abrir o arquivo *petrilab.pyw*, como mostrado na Figura 5-9. Pode ser conveniente criar um atalho para esse arquivo em sua pasta de preferência.

Figura 5-8: Extração do PETRILab

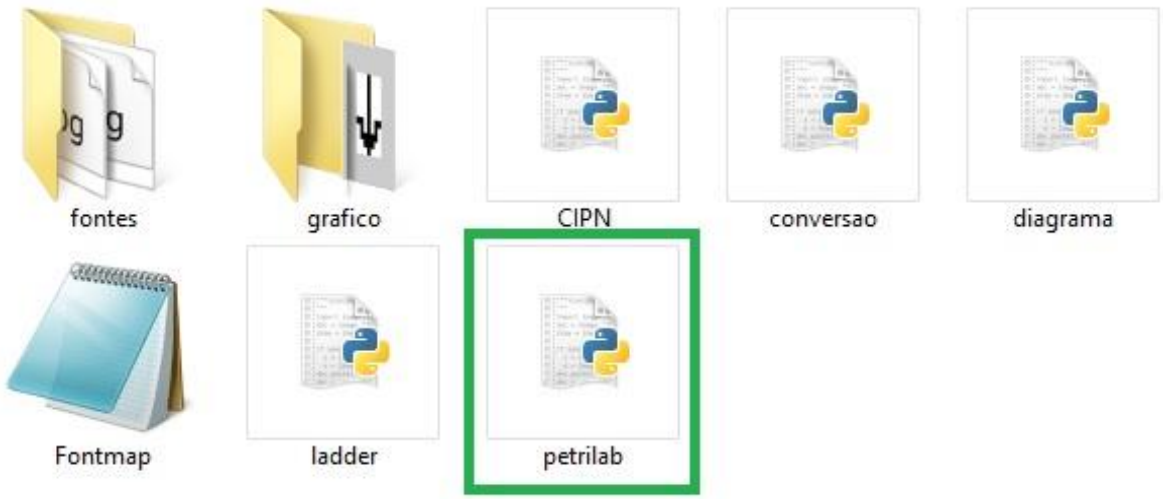


Figura 5-9: Execução do PETRILab

5.3.2 Interface

O PETRILab conta com uma interface gráfica simples e intuitiva, como exibido na Figura 5-10.

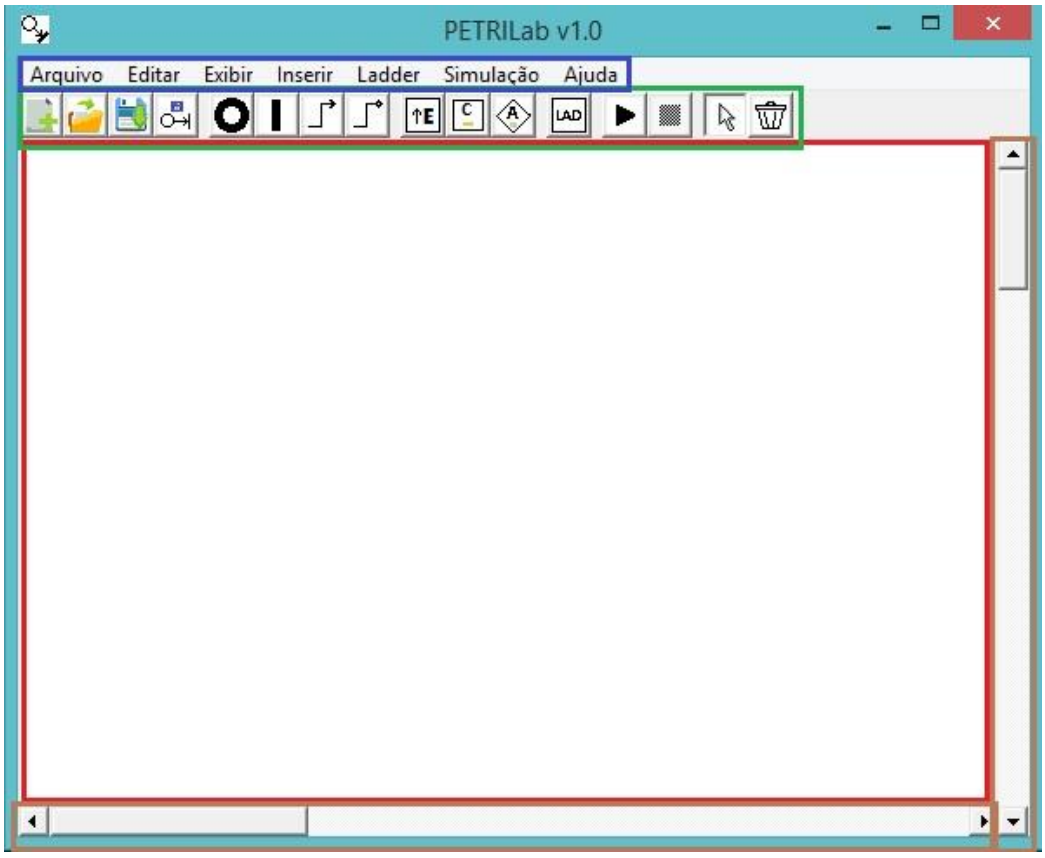


Figura 5-10: Interface do programa

Na Figura 5-10, a área marcada em vermelho é a área de desenho, onde serão inseridos os lugares, transições e arcos da Rede de Petri. Essa área pode ser estendida através das barras de rolagem marcadas em marrom. A barra de ferramentas, marcada em verde, contém botões para acesso rápido de funções, como gerenciamento do arquivo, inserção de elementos, geração de Ladder, simulação e ferramentas de edição. Por fim, a área em azul mostra os menus do programa, onde é possível acessar todas as funções presentes nele.

5.3.3 Inserir elementos

a) Lugar

Para inserir um lugar na rede, clique no botão marcado na Figura 5-11. Alternativamente, vá em *Inserir→Lugar (L)*, ou use a tecla de atalho <*L*>. Em seguida, clique em um ponto da área de desenho. Em qualquer momento o usuário pode apertar a tecla <*Esc*> para cancelar a inserção.

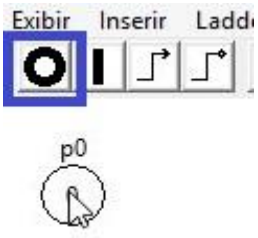


Figura 5-11: Inserção de lugar

b) Transição

Para inserir uma transição na rede, clique no botão marcado na Figura 5-12. Alternativamente, vá em *Inserir→Transição (T)*, ou use a tecla de atalho <*T*>. Em seguida, clique em um ponto da área de desenho. Em qualquer momento o usuário pode apertar a tecla <*Esc*> para cancelar a inserção.

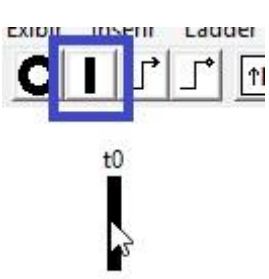


Figura 5-12: Inserção de transição

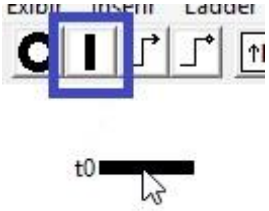


Figura 5-13: Inserção de transição rotacionada

Uma transição pode ser rotacionada no ato de inserção, bastando para isso que o usuário aperte <*Clique-Direito*> em seu mouse. A Figura 5-13 ilustra uma transição rotacionada sendo inserida.

c) Arco

Um arco pode ser inserido de um lugar a uma transição, ou de uma transição a um lugar. Para inserir um arco, clique no botão marcado na Figura 5-14. Alternativamente, vá em *Inserir→Arco*, ou use a tecla de atalho <*A*>. O cursor se transformará na cruz destacada na Figura 5-14. Ao clicar em um lugar ou transição, o cursor se transformará em uma cruz menor, mostrado na Figura 5-15, indicando que a origem do arco já foi definida, restando informar seu destino. Em qualquer momento o usuário pode apertar a tecla <*Esc*> para cancelar a inserção.

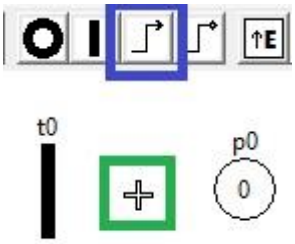


Figura 5-14: Inserção de arco

Figura 5-15: Origem do arco definida

Para definir o destino do arco, basta clicar no lugar ou transição de destino desejado, como mostrado na Figura 5-16. O programa reajusta automaticamente a posição exata do arco para melhor exibição.

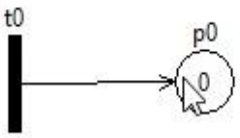


Figura 5-16: Arco inserido

O usuário pode também adicionar diversos segmentos ao arco antes de definir seu destino. Para isso, basta clicar em qualquer lugar da área de desenho, como ilustrado na Figura 5-17. O cursor se manterá como uma cruz pequena. Para finalizar o arco, basta clicar em um lugar ou transição, como mostrado na Figura 5-18.

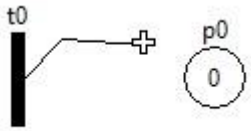


Figura 5-17: Arco segmentado sendo inserido

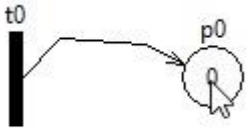


Figura 5-18: Arco segmentado finalizado

d) Arco inibidor

Um arco inibidor pode ser inserido apenas de um lugar a uma transição. Para inseri-lo, clique no botão marcado na Figura 5-19. Alternativamente, vá em *Inserir→Arco Inibidor*, ou use a tecla de atalho <*I*>. O cursor se transformará na cruz destacada na Figura 5-19. Ao clicar em um lugar, o cursor se transformará em uma cruz menor, mostrado na Figura 5-20, indicando que a origem do arco já foi definida, restando informar seu destino. Em qualquer momento o usuário pode apertar a tecla <*Esc*> para cancelar a inserção.

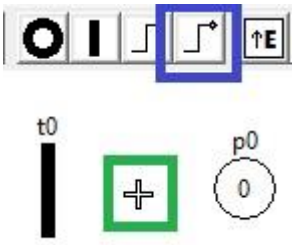


Figura 5-19: Inserção de arco inibidor

Figura 5-20: Origem do arco inibidor definida

Para definir o destino do arco, basta clicar na transição de destino desejada, como mostrado na Figura 5-21. O programa reajusta automaticamente a posição exata do arco para melhor exibição.

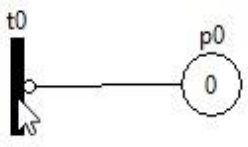


Figura 5-21: Arco inibidor inserido

O usuário pode também adicionar diversos segmentos ao arco antes de definir seu destino. Para isso, basta clicar em qualquer lugar da área de desenho, como ilustrado na Figura 5-22. O cursor se manterá como uma cruz pequena. Para finalizar o arco, basta clicar em uma transição, como mostrado na Figura 5-23.

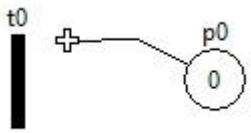


Figura 5-22: Arco inibidor segmentado

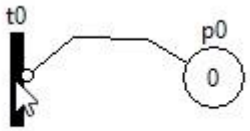


Figura 5-23: Arco inibidor segmentado finalizado

e) Evento

Para adicionar um evento, basta clicar no botão marcado na Figura 5-24, ou ir em *Inserir→Evento*, ou ainda apertar a tecla de atalho <*E*>. Um botão de evento será inserido automaticamente no canto esquerdo do programa, como mostra a Figura 5-24. Para associar eventos às transições, veja a seção 5.3.4b.

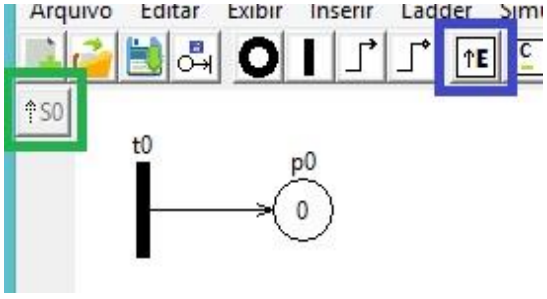


Figura 5-24: Inserção de evento

f) Condição

Para adicionar uma condição, basta clicar no botão marcado na Figura 5-25, ou ir em *Inserir→Condição*, ou ainda apertar a tecla de atalho <*C*>. Um botão de condição será inserido automaticamente na parte superior da área de desenho, como mostra a Figura 5-25. Para associar condições às transições, veja a seção 5.3.4b.

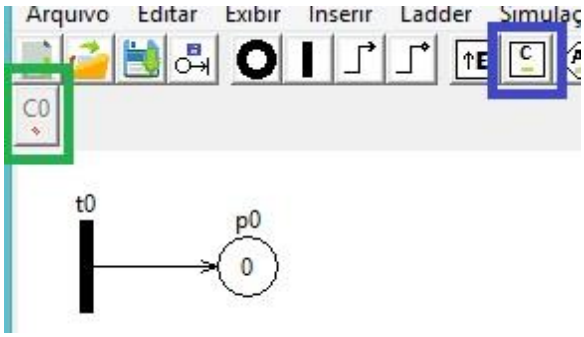


Figura 5-25: Inserção de condição

g) Ação impulsional

Para adicionar uma ação, basta clicar no botão marcado na Figura 5-26, ou ir em *Inserir→Ação*, ou ainda apertar a tecla de atalho <*K*>. Um indicador de ação impulsional será inserido automaticamente no canto direito do programa, como mostra a Figura 5-26. Para associar ações impulsionais aos lugares, veja a seção 5.3.4a.

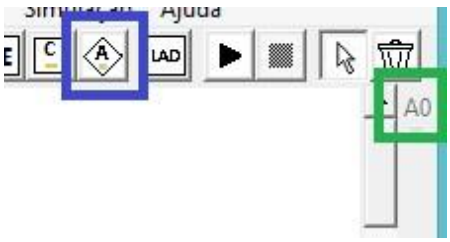


Figura 5-26: Inserção de ação impulsional

5.3.4 Editar elementos

a) Lugares

Para editar um lugar, basta apertar <*Duplo-Clique-Esquerdo*> com o mouse em cima do lugar desejado. A interface mostrada na Figura 5-27 irá aparecer; nela é possível alterar o rótulo do lugar, alterar sua marcação inicial, ou associar a ele ações impulsionais já

criadas. O lugar será então alterado conforme as modificações realizadas, como mostra a Figura 5-28. Caso seja de interesse a geração do diagrama Ladder da rede, recomenda-se que o rótulo dos lugares tenha, no máximo, três dígitos, para que ele não extravase o desenho dos blocos de funções.

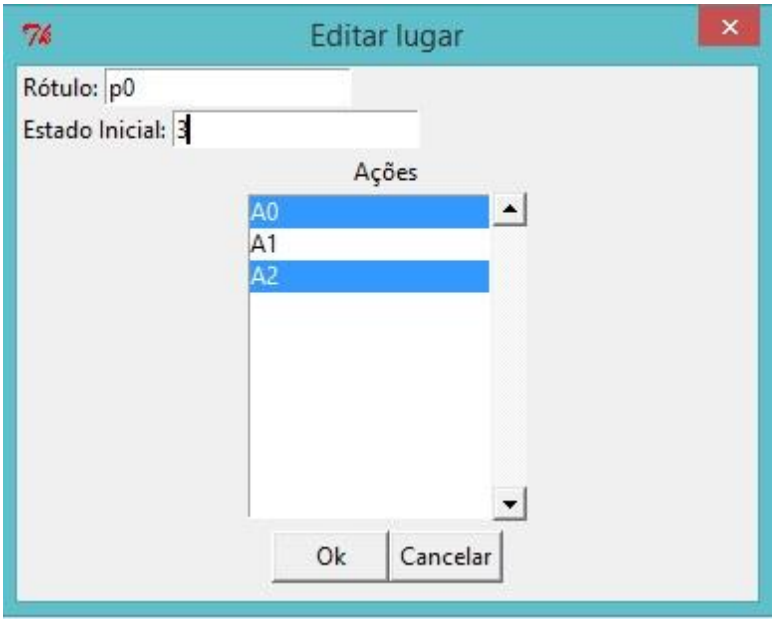


Figura 5-27: Edição de lugar

Figura 5-28: Lugar editado

b) Transições

Para editar uma transição, basta apertar <*Duplo-Clique-Esquerdo*> com o mouse em cima da transição desejada. A interface mostrada na Figura 5-29 irá aparecer; nela é possível alterar o rótulo da transição, o evento associado, a condição associada ou o atraso associado. A transição será, então, alterada conforme as modificações realizadas, como

mostra a Figura 5-30. Caso seja de interesse a geração do diagrama Ladder da rede, recomenda-se que o rótulo das transições tenha, no máximo, três dígitos.

Figura 5-29: Edição de transição

Figura 5-30: Transição editada

Uma transição já inserida também pode ser rotacionada, basta que o usuário aperte *<Clique-Direito>* com o ponteiro do mouse em cima da transição, como ilustrado na Figura 5-31.

Figura 5-31: Rotação de transição

c) Arcos e arcos inibidores

Para editar qualquer tipo de arco, basta apertar <*Clique-Duplo*> com o ponteiro do mouse em cima do arco correspondente. É possível, então, editar seu peso, como mostra a Figura 5-32.

Figura 5-32: Edição de arco

O peso do arco será então mostrado no meio do arco correspondente. Um algoritmo garante que o peso esteja sempre no meio do arco, mesmo que ele seja segmentado, como ilustrado na Figura 5-33.

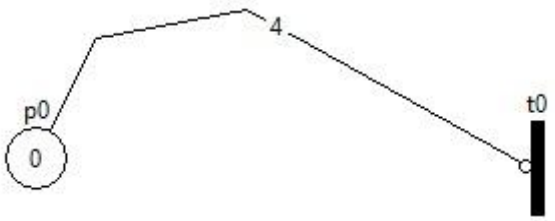


Figura 5-33: Arco editado

d) Eventos, condições e ações

Para editar eventos, ações e condições, basta apertar <*Clique-Direito*> com o mouse em cima do botão correspondente. É possível então editar seus rótulos, e, para o caso de eventos, o seu tipo de borda, como ilustrado da Figura 5-34 a Figura 5-36.

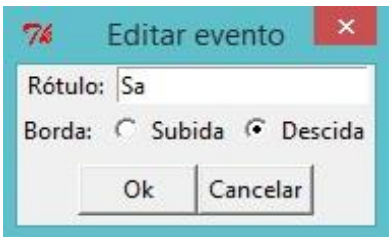


Figura 5-34: Edição de evento

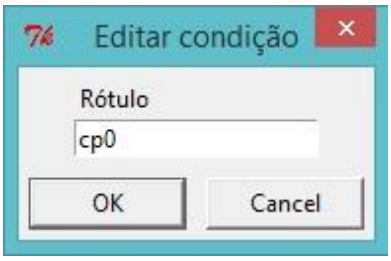


Figura 5-35: Edição de condição

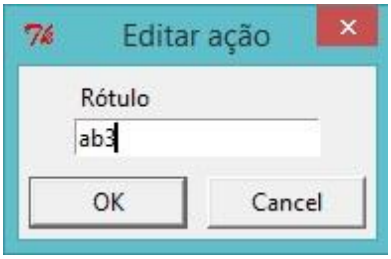


Figura 5-36: Edição de ação

5.3.5 Mover elementos

Para mover um lugar ou uma transição, basta segurar <*Clique-Esquerdo*> no mouse com o ponteiro em cima deles, e então mover para onde desejar. Quando o elemento estiver no local desejado, basta soltar o botão. Quando movemos um lugar ou transição, os arcos se ajustam automaticamente; no entanto, arcos segmentados só movimentam seu primeiro e último segmentos. Esse comportamento é ilustrado nas Figura 5-37 (a) e (b).

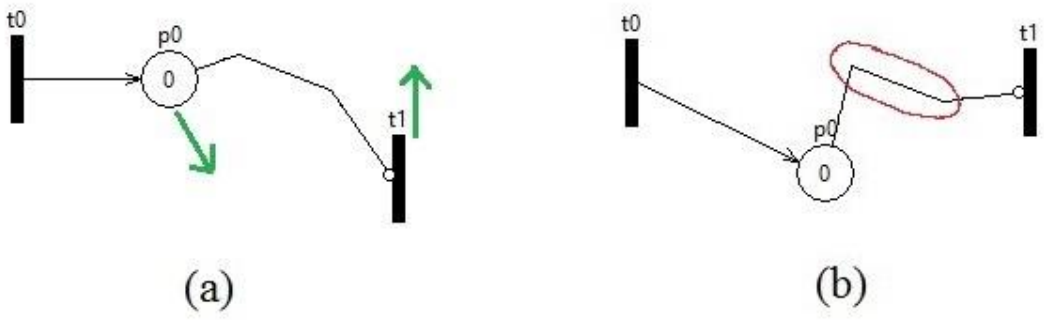


Figura 5-37: Movendo lugares e transições

5.3.6 Remover elementos

a) Lugares, transições, arcos e arcos inibidores

Para remover elementos presentes na área de desenho, é necessário entrar no modo de remoção, clicando no botão mostrado na Figura 5-38 (a) ou apertando a tecla de atalho <*X*>. O cursor se tornará uma mão, indicando o modo de remoção. Para deletar um elemento, basta então apertar <*Clique-Esquerdo*> com o ponteiro do mouse em cima dele; todos os arcos de saída ou chegada do elemento removido também serão removidos, com ilustram a Figura 5-38 (b).

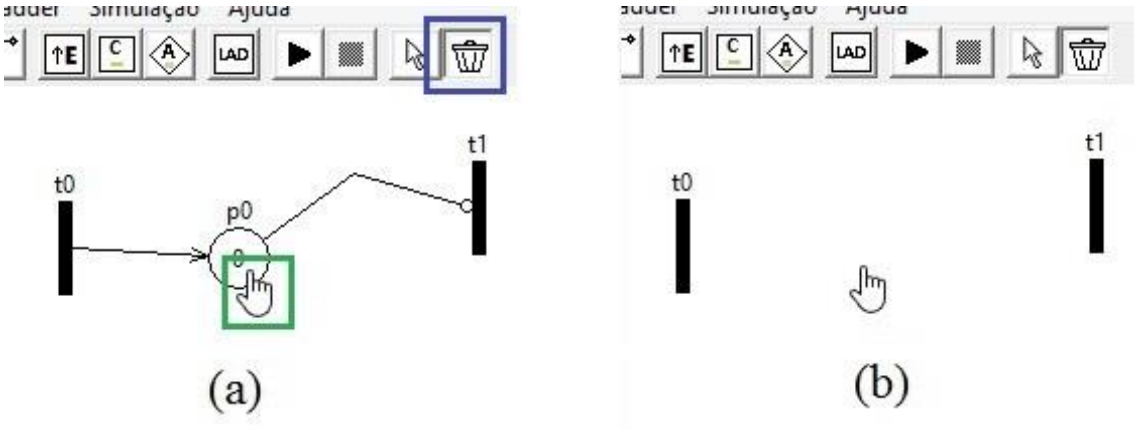


Figura 5-38: Remoção de elementos da área de desenho

Para voltar ao modo de seleção, basta clicar no ícone de seleção marcado na Figura 5-39; o cursor então voltará a ser uma seta. Alternativamente, é possível apertar a tecla de atalho <*Q*>, ou a tecla <*Esc*>.

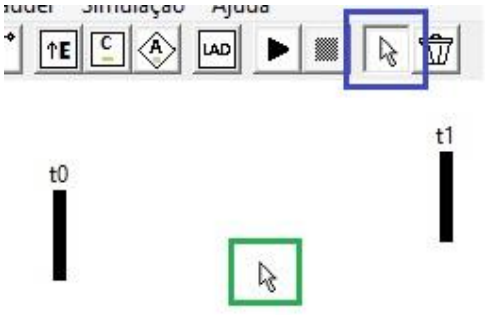


Figura 5-39: Retornando ao modo de seleção

b) Eventos, condições e ações

Para remover eventos, condições ou ações, basta apertar <*Clique-Direito*> sobre o botão correspondente; ele desaparecerá, como ilustrado nas Figura 5-40 (a) e (b). Qualquer lugar ou transição associado com o item deletado será automaticamente desassociado.

Figura 5-40 Remoção de eventos, condições e ações

5.3.7 Exibir ou ocultar rótulos

Pode de interesse do usuário ocultar os rótulos de lugares e transições, deixando apenas a mostra os eventos, condições e ações associados. Para fazer isso, basta ir em *Exibir→Ocultar Rótulos*, como mostra a Figura 5-41. Os rótulos serão, então, ocultados, como pode ser visto na Figura 5-42; para exibi-los novamente, basta desmarcar a opção.

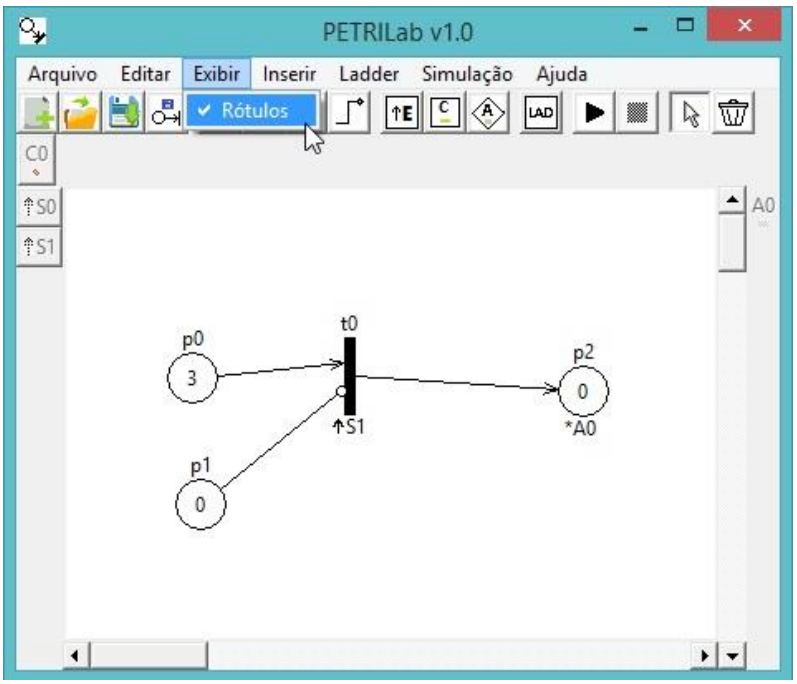


Figura 5-41: Ocultando rótulos

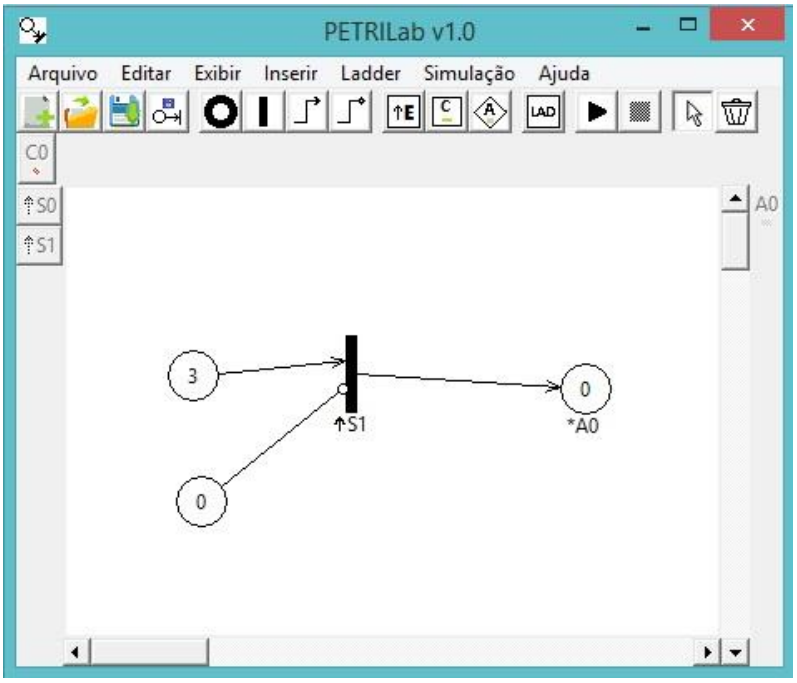


Figura 5-42: Rótulos ocultos

5.3.8 Simular a Rede de Petri

Uma vez modelada, a RPIC pode ser simulada de acordo com a dinâmica explicada na seção 2.4. Para entrar no modo de simulação, basta clicar no botão mostrado na Figura 5-43 (a), ou apertar a tecla de atalho <*S*>. Como pode ser visto na Figura 5-43 (b), todos os botões de edição se tornam desabilitados, enquanto os botões de eventos, condições, e os indicadores de ações se tornam habilitados. Para voltar ao modo de edição, basta clicar no botão destacado na Figura 5-43 (b) ou apertar a tecla de atalho <*Z*>.

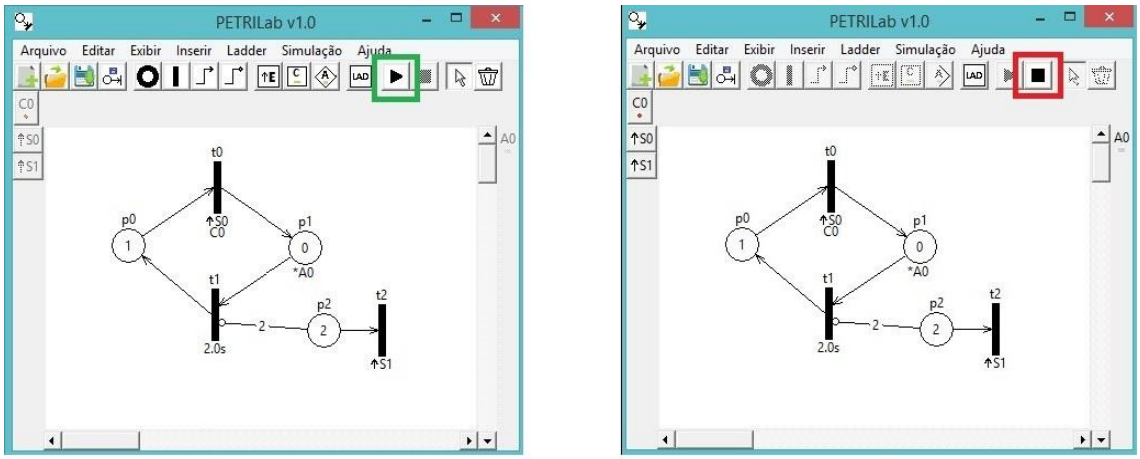


Figura 5-43: Modo de simulação

O modo de simulação só poderá ser iniciado caso haja ao menos um lugar, uma transição e um arco na rede. Além disso, todas as transições da rede precisam de algum limitador para a sua ocorrência, seja ele um arco de entrada, um evento ou uma condição; isso é feito para evitar disparos infinitos de uma transição. O programa conta também com um sistema de prevenção de *loopings infinitos:* caso sejam executadas mais de 2000 iterações na rede sem a intervenção do usuário, a janela mostrada na Figura 5-44 será mostrada, e o usuário pode optar por interromper a simulação.

Figura 5-44: Proteção contra loopings infinitos

Para acionar um evento, basta clicar em seu botão correspondente na barra à esquerda da área de desenho. Para alternar o estado de uma condição, basta clicar em seu botão correspondente na barra superior à área de desenho; uma luz no botão indica se o estado lógico atual é 1 ou 0 – verde ou vermelho – como ilustrado nas Figura 5-45 (a) e (b). A rede responderá de acordo, evoluindo conforme explicado na seção 2.4.

Figura 5-45: Alternando entre estado lógico de condições

Quando um lugar associado a uma ação impulsional muda sua marcação de 0 para 1, o indicador da ação associada pisca com uma luz verde durante 1 segundo, indicando que a ação foi executada, como mostram as Figura 5-46 (a) e (b).

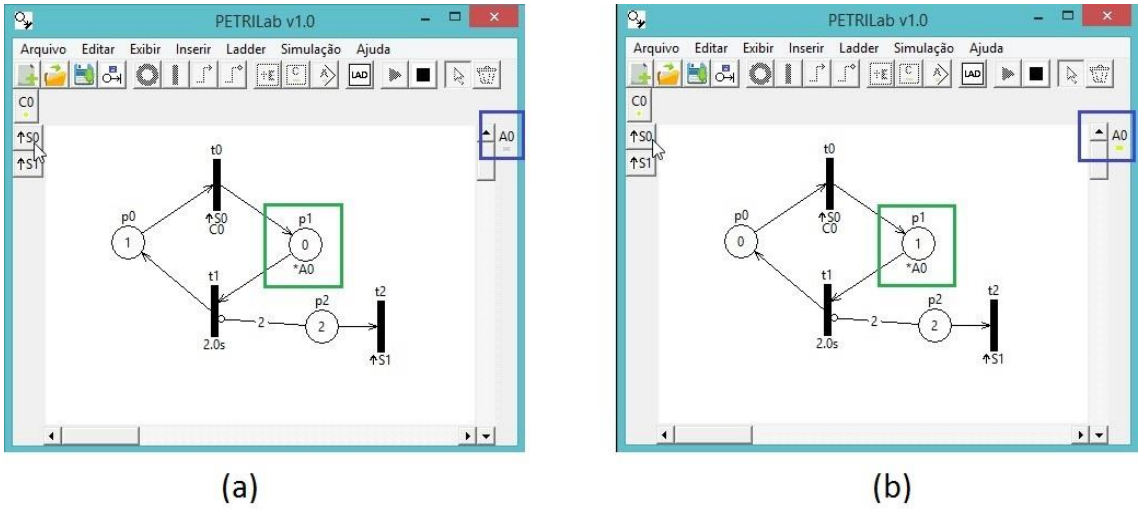


Figura 5-46: Execução de ações impulsionais

5.3.9 Conversão RPIC-LADDER

Uma das funções inovadoras do PETRILab é a conversão automática de uma Rede de Petri Interpretada para Controle em um diagrama Ladder. Essa conversão pode ser feita instantaneamente clicando-se no botão mostrado na Figura 5-47, ou apertando-se a tecla de atalho <*G*>.

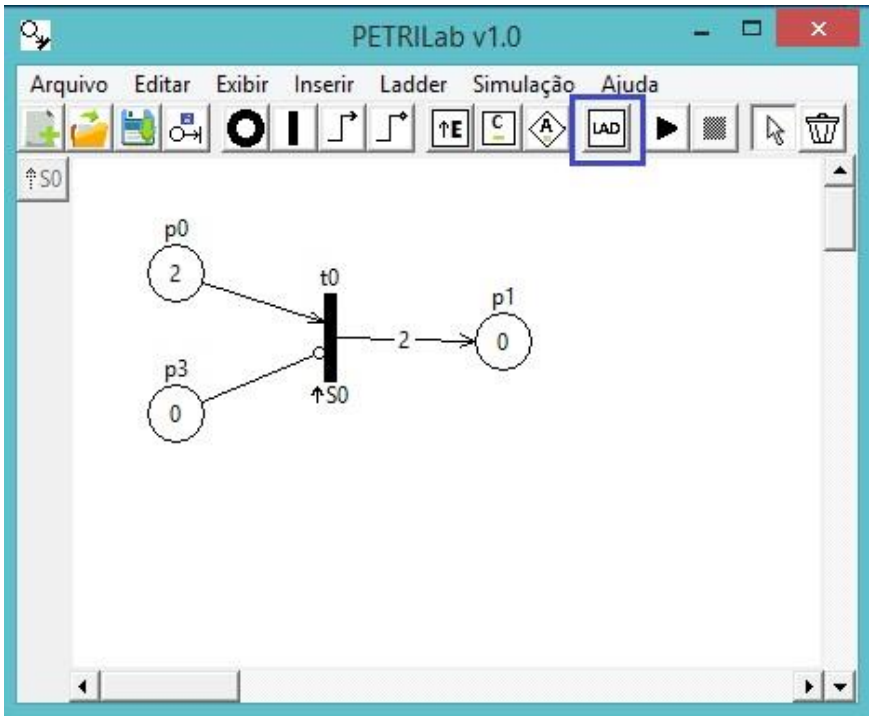


Figura 5-47: Geração de diagrama Ladder

O programa então abrirá uma janela com o diagrama gerado, de acordo com os passos propostos na Seção Capítulo 4 deste trabalho. A Figura 5-48 mostra o diagrama gerado para a rede da Figura 5-47.

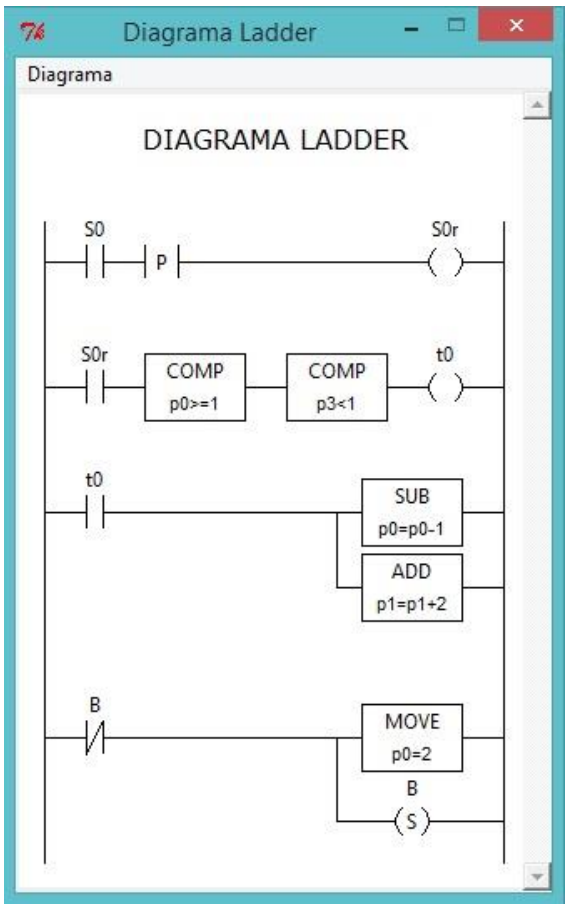


Figura 5-48: Diagrama Ladder gerado a partir da RPIC da Figura 5-47

5.3.10 Salvar imagem da RPIC e do LADDER

a) Grafo da RPIC

O grafo da RPIC modelada pode ser salvo em formato *encapsulated postscript (eps)* clicando-se no botão mostrado na Figura 5-49, ou no menu *Arquivo→Salvar Imagem do Grafo*. Caso o arquivo da rede já tenha sido salvo, um arquivo chamado *RPIC.eps* será gerado na mesma pasta dele; caso contrário, será gerado na pasta do PETRILab.

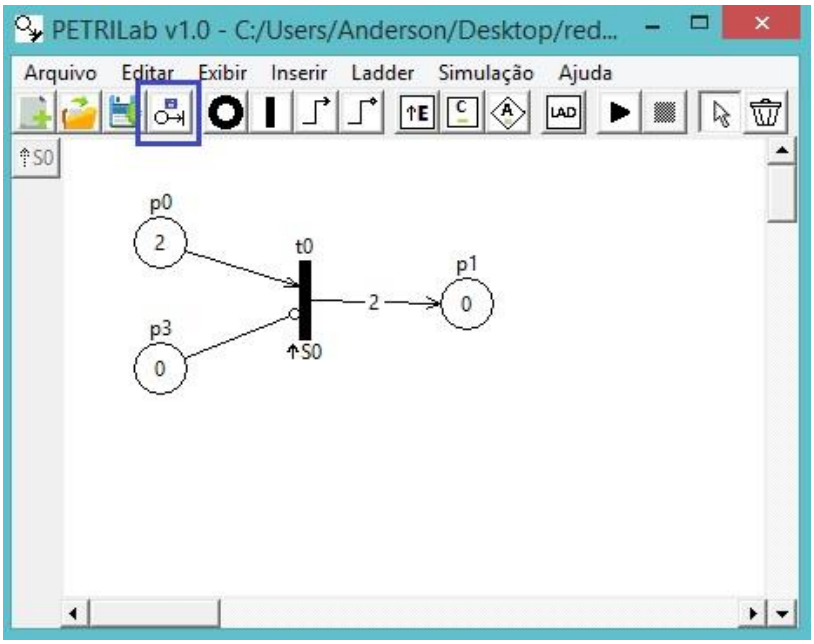


Figura 5-49: Salvando imagem do grafo

O arquivo *eps* pode então ser aberto em algum programa compatível, como o *Adobe Illustrator*, conforme mostra a Figura 5-50.

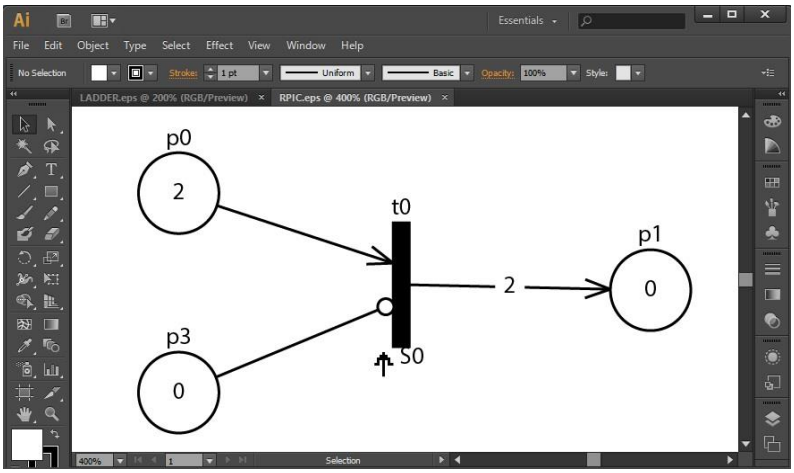


Figura 5-50: Grafo aberta no Adobe Illustrator

b) Diagrama Ladder

Para salvar uma imagem do diagrama Ladder em formato *eps*, basta ir ao menu *Diagrama→Salvar*, como mostra a Figura 5-51.

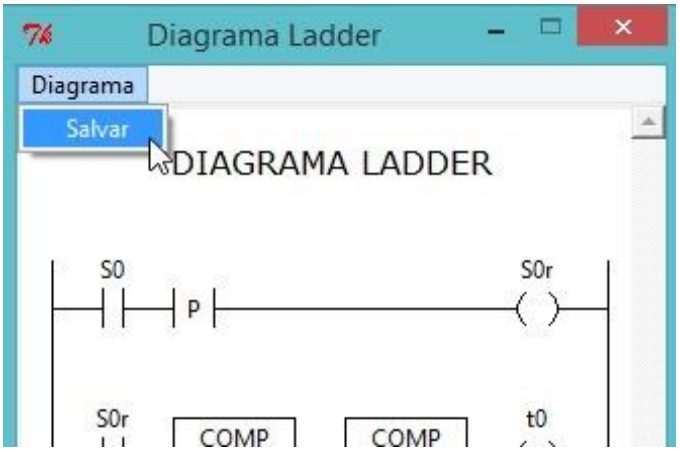


Figura 5-51: Salvando diagrama Ladder

A Figura 5-52 mostra o arquivo *eps* correspondente ao diagrama Ladder da figura 73.

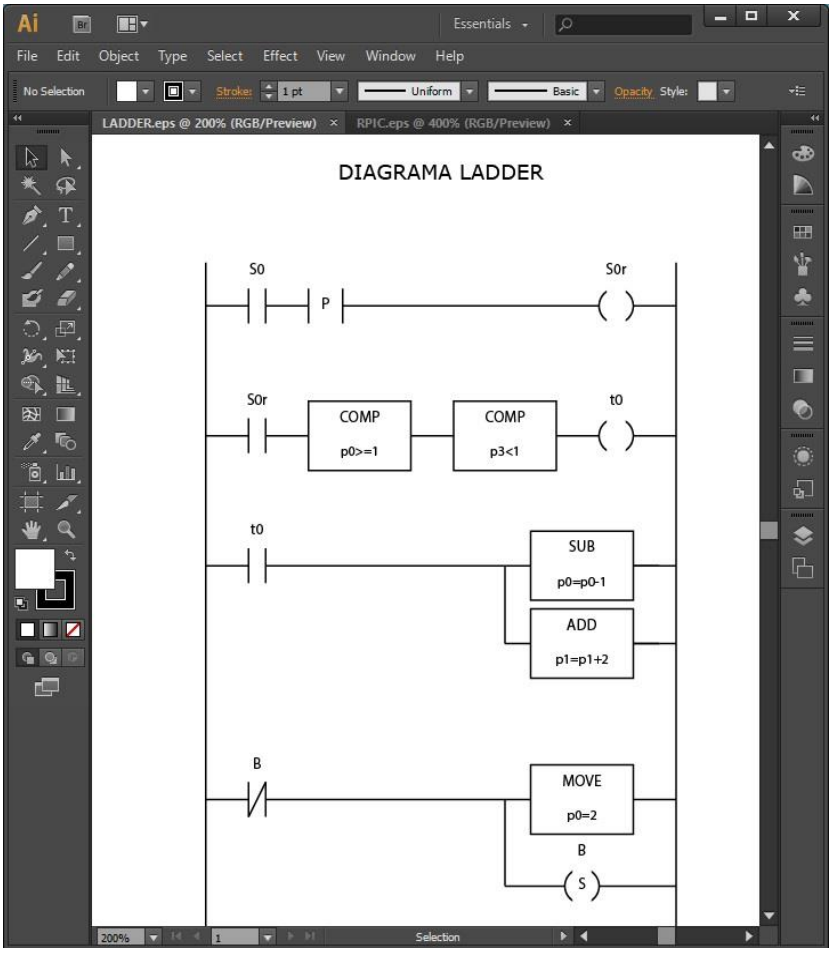


Figura 5-52: Diagrama Ladder aberto no Adobe Illustrator

5.3.11 Criar um Novo Arquivo, Abrir um Arquivo Salvo e Salvar Arquivo Editado

O PETRILab salva os arquivos de projeto com a extensão *.pl;* os arquivos são extremamente leves, dificilmente superando 20kB. As funções *Novo*, *Abrir* e *Salvar* podem ser executadas clicando-se nos botões mostrados na Figura 5-53 – que estão nessa mesma ordem – ou através do menu *Arquivo*.

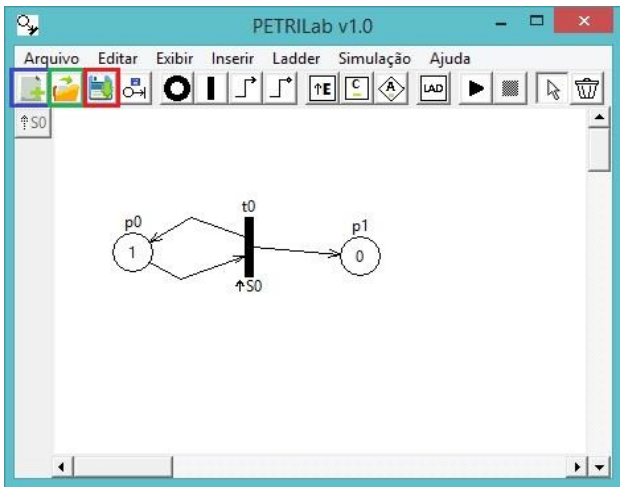


Figura 5-53: Novo, Abrir e Salvar

5.3.12 Ajuda e suporte

O usuário pode ter acesso a este guia através do menu *Ajuda→Guia do Usuário*, que abre no navegador um link para download do documento em *pdf* diretamente da página oficial do PETRILab no Sourceforge.

Para reportar bugs ou dar sugestões de melhorias, o usuário pode entrar no menu *Ajuda→Informar Bugs/Sugestões*, que abrirá no navegador a página de *Tickets* do PETRILab no Sourceforge. Alternativamente, os bugs e sugestões podem ser enviados por e-mail para *petrilab10@gmail.com;* no caso de bugs, é recomendado que o usuário anexe o arquivo *.pl* do projeto em que o bug ocorreu.

5.4 Lista de Teclas de Atalho

Tabela 5.1: Lista de teclas de atalho

Tecla de Atalho	Função
<CTRL-A>	Abrir arquivo
<CTRL-S>	Salvar arquivo
<CTRL-N>	Novo arquivo
<Esc>	Cancelar ação atual
<A>	Inserir arco
<C>	Inserir condição
<E>	Inserir evento
<I>	Inserir arco inibidor
<G>	Gerar diagrama Ladder
<K>	Inserir ação impulsional
<L>	Inserir lugar
<Q>	Modo de seleção
<S>	Iniciar simulação
<T>	Inserir transição
<X>	Modo de remoção
<Z>	Parar simulação

Capítulo 6
Projeto de um Controlador a Eventos Discretos usando o PETRILab

Com o PETRILab devidamente apresentado, iremos ilustrar a sua utilização com o projeto de um controlador a eventos discretos. Para tanto, modelaremos o sistema por uma RPIC, e, em seguida, utilizando o programa, realizaremos sua conversão em diagrama Ladder. O diagrama será então inserido em um CLP, para que seja feito o controle do SED.

O capítulo está estruturado da seguinte forma: na seção 6.1 é descrito o sistema que servirá de exemplo; a seção 6.2 trata da sua modelagem em RPIC e conversão em Ladder; na seção 6.3 é mostrado como o sistema foi implementado em um CLP; na seção 6.4 é descrita a montagem em bancada; por fim, na seção 6.5 são mostrados os resultados.

6.1 Sistema de Exemplo

O sistema escolhido para servir de exemplo foi extraído de [2]. Trata-se de um motor de indução trifásico. Sabe-se que a corrente de partida de um motor desse tipo pode ser muito elevada, e, dependendo da carga a ele acoplada, pode chegar a danificar seus enrolamentos. Por esse motivo, é comum utilizar-se algum método diferente da partida direta. Dentre esses métodos, se destaca o da partida Y-Δ, devido a sua simplicidade e

baixo custo. O método consiste em inicialmente alimentar o motor com a tensão de fase (ligação Y), e, após o sistema atingir regime permanente, mudar a alimentação para a tensão de linha (ligação Δ); dessa forma, a corrente de partida fica reduzida a 1/3 da corrente de partida direta.

Neste exemplo, a troca das ligações será feita através de relés eletromecânicos. A ligação a ser montada em bancada é a da Figura 6-1. Os relés que fazem as ligações de Y e Δ são K1 e K4, respectivamente. É desejado também que o motor possa ter partida em dois sentidos diferentes, o que pode ser realizado escolhendo-se entre os relés K2 e K3, que ligam o motor a rede no esquema ABC ou ACB.

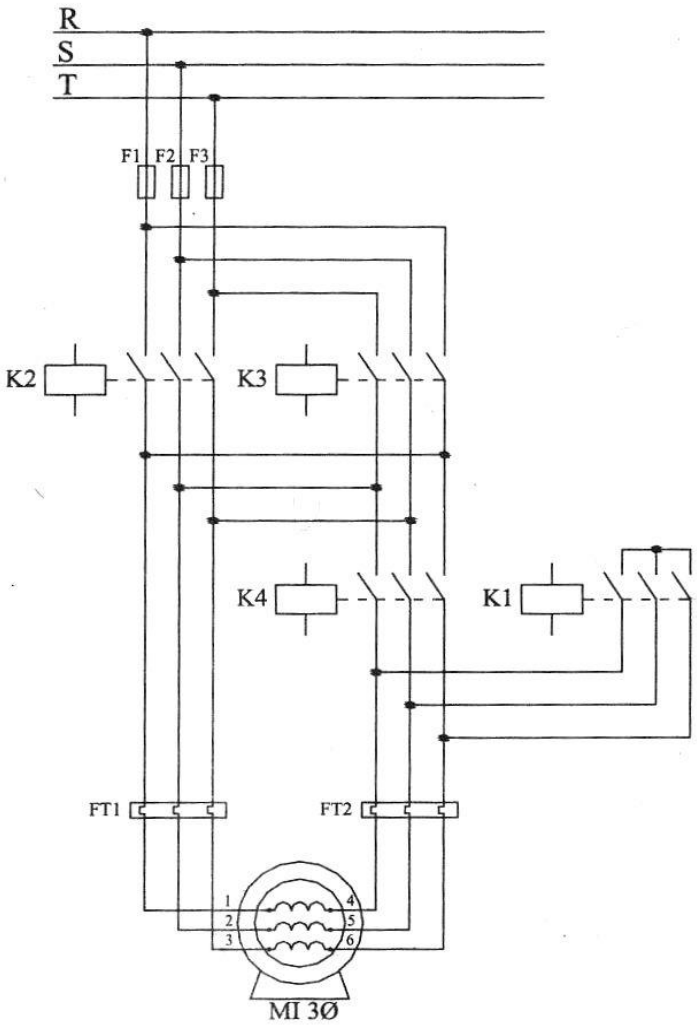


Figura 6-1: Esquema de partida Y- Δ em dois sentidos

O sistema de controle do motor deve obedecer aos seguintes critérios:

- Deve haver um botão S1 para partida em um sentido, um botão S2 para partida em outro sentido e um botão S0 para parada;
- Não deve ser possível reverter o motor com ele girando;
- O motor deve partir em ligação Y e alternar para Δ após 2s;
- Ao apertar-se a chave de parada, nenhum botão terá efeito por 5s, para garantir a parada completa do motor.

6.2 Modelagem da RPIC e Geração do Ladder

Para dar a partida do motor em um determinado sentido, respeitando-se a partida em Y-Δ, o par de chaves (K1, K2) deve ser fechado, e, em seguida, (K4, K2). Para o outro sentido, o par (K1, K3) deve ser fechado, e, em seguida, (K4, K3). Para o parar o motor, podemos abrir todas as chaves do sistema. O conjunto de estados do sistema pode então ser dado por:

$$X = \{\emptyset, (K1, K2), (K1, K3), (K4, K2), (K4, K3)\} \quad (9)$$

O conjunto da Equação (9) é um conjunto contável, portanto os estados do sistema são discretos. Sabemos também que os estados do sistema só devem mudar quando os relés forem acionados pelo sistema de controle, ou seja, o sistema é dirigido por eventos. Sendo assim, temos um SED, e podemos modelá-lo utilizando uma RPIC.

O sistema proposto é bem simples, e sua modelagem em RPIC bem intuitiva: o sistema deve iniciar com apenas uma ficha em um lugar onde todas as chaves estão abertas; ao apertar-se um dos botões de partida, uma transição deve levar um estado do sistema a um lugar com ações correspondentes aos relés de algum sentido e da ligação em Y; após 2s, uma transição deve levar a ficha para um lugar que tenha a ação correspondente à Δ; ao apertar-se a chave S0, em qualquer situação de movimento, a ficha deve ir a um lugar sem nenhuma ação; somente após 5s, o sistema deve voltar ao estado inicial. Para a partida no outro sentido de rotação, a rede deve ser espelhada.

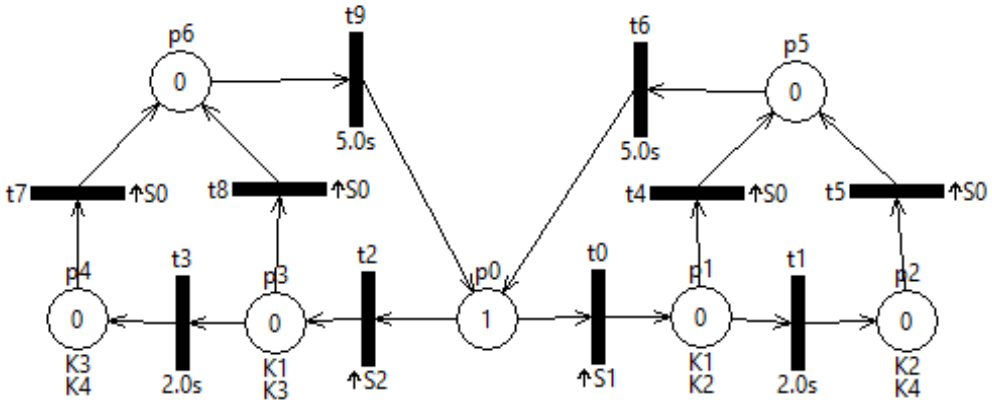


Figura 6-2: RPIC correspondente ao sistema de acionamento do motor de indução

A Figura 6-2 mostra a RPIC modelada para o sistema utilizando o PETRILab, após as ações terem sido alteradas de impulsionais para contínuas. Apesar do programa trabalhar apenas com ações impulsionais, elas foram modificadas para contínuas para facilitar o controle do sistema. Os eventos $\uparrow S0$, $\uparrow S1$ e $\uparrow S2$ representam o apertar dos botões de parada e partida nos dois sentidos, respectivamente. As ações K1, K2, K3 e K4 representam a energização das bobinas dos relés eletromecânicos K1, K2, K3 e K4 da Figura 6-1, respectivamente. Os lugares são explicados na Tabela 6.1.

Tabela 6.1: Significado dos lugares da RPIC da Figura 6-2

$p0$	Motor parado aguardando comando
$p1$	Motor girando para o sentido 1 em Y
$p2$	Motor girando para o sentido 1 em Δ
$p3$	Motor girando para o sentido 2 em Y
$p4$	Motor girando para o sentido 2 em Δ
$p5$	Motor desacelerando, girando no sentido 1
$p6$	Motor desacelerando, girando no sentido 2

Nota-se que os lugares $p5$ e $p6$ da rede são redundantes, e podem ser substituídos por um único lugar. No entanto, esse detalhe só foi percebido após a implementação do controlador, portanto iremos nos referir a rede não simplificada da Figura 6-2 no restante deste trabalho.

Uma vez modelada a RPIC, basta apenas um clique para o programa gerar o diagrama Ladder correspondente, como explica a seção 5.3.9. O diagrama é mostrado na Figura 6-3. Novamente, vale salientar que o programa gera o módulo das ações considerando-as impulsionais. Para transformar o módulo em ações contínuas, basta apagar o bloco P de cada linha dele, como foi explicado na seção 4.6.

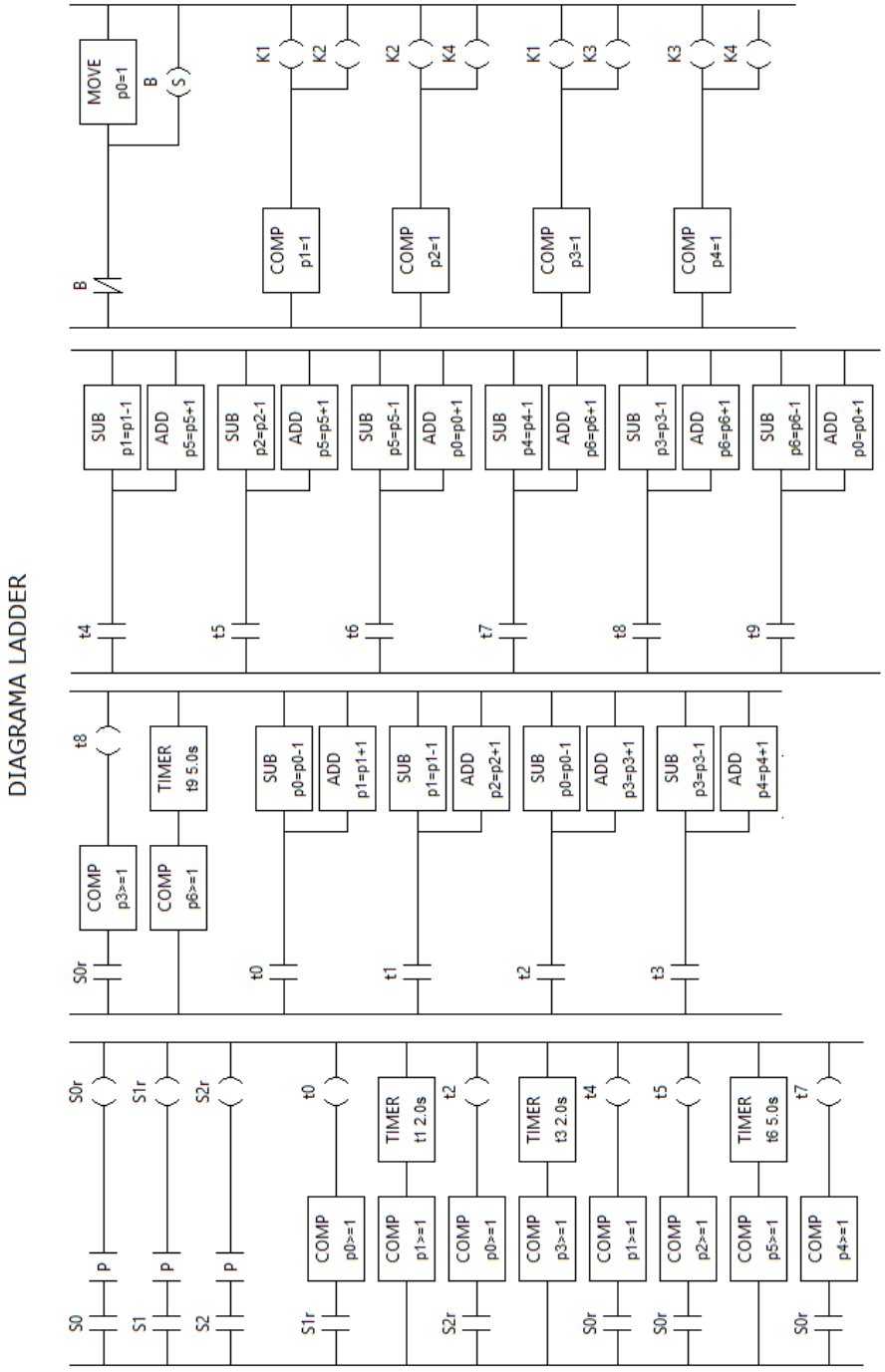


Figura 6-3: Diagrama Ladder gerado pelo PETRILab

6.3 Implementação no CLP

Com o diagrama Ladder gerado, podemos partir para a sua implementação em um CLP. O CLP escolhido para o controle do sistema foi o S7-1200, da Siemens, que possui 14 entradas digitais e 10 saídas a relé, estando acoplado a um painel que permite acesso com cabos banana a todas suas entradas e saídas, o que facilita a montagem do sistema. O conjunto é mostrado na Figura 6-4.

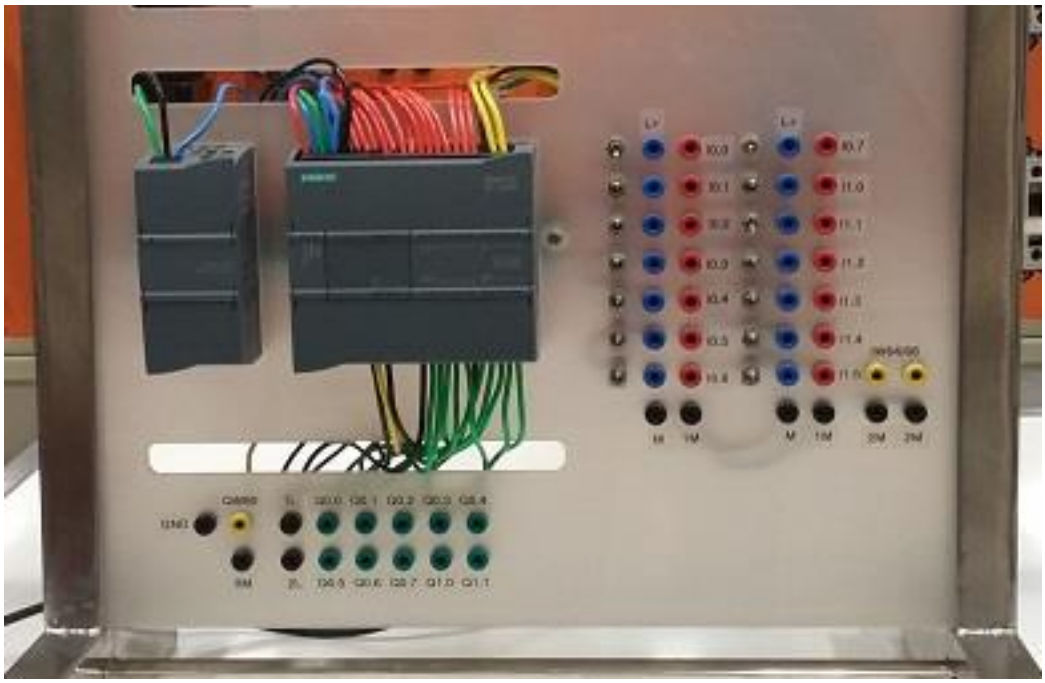


Figura 6-4: CLP Siemens S7-1200 acoplado em painel

A programação do CLP foi feita em um PC, utilizando o software STEP7, da Siemens. Ele foi conectado ao PC por meio de um cabo Ethernet, e configurado seguindo-se os mesmos passos propostos em [7] para a configuração do Siemens S7-300.

Com as configurações iniciais feitas, podemos partir para a criação das variáveis (associadas a *tags* no STEP7), e inserção do Ladder.

Inicialmente, tentamos usar exatamente o Ladder da Figura 6-3 no CLP para realizarmos os testes. No entanto, ocorreram comportamentos inesperados com relação ao uso de variáveis inteiras: elas inicializavam com valores não nulos, e tentativas de reset no início do diagrama não surtiam efeito. Como a RPIC deste exemplo é segura, isto é, todos os seus lugares só assumem marcação um ou zero, podemos utilizar variáveis

booleanas no lugar de inteiras para representar a marcação dos estados. Assim, os blocos "COMP ≥", "$COMP =$" podem ser substituídos por contatos NA; blocos "ADD" e "MOVE" por bobinas SET; e blocos "SUB" por bobinas RESET, como descrito nas seções 4.3, 4.4 e 4.5.

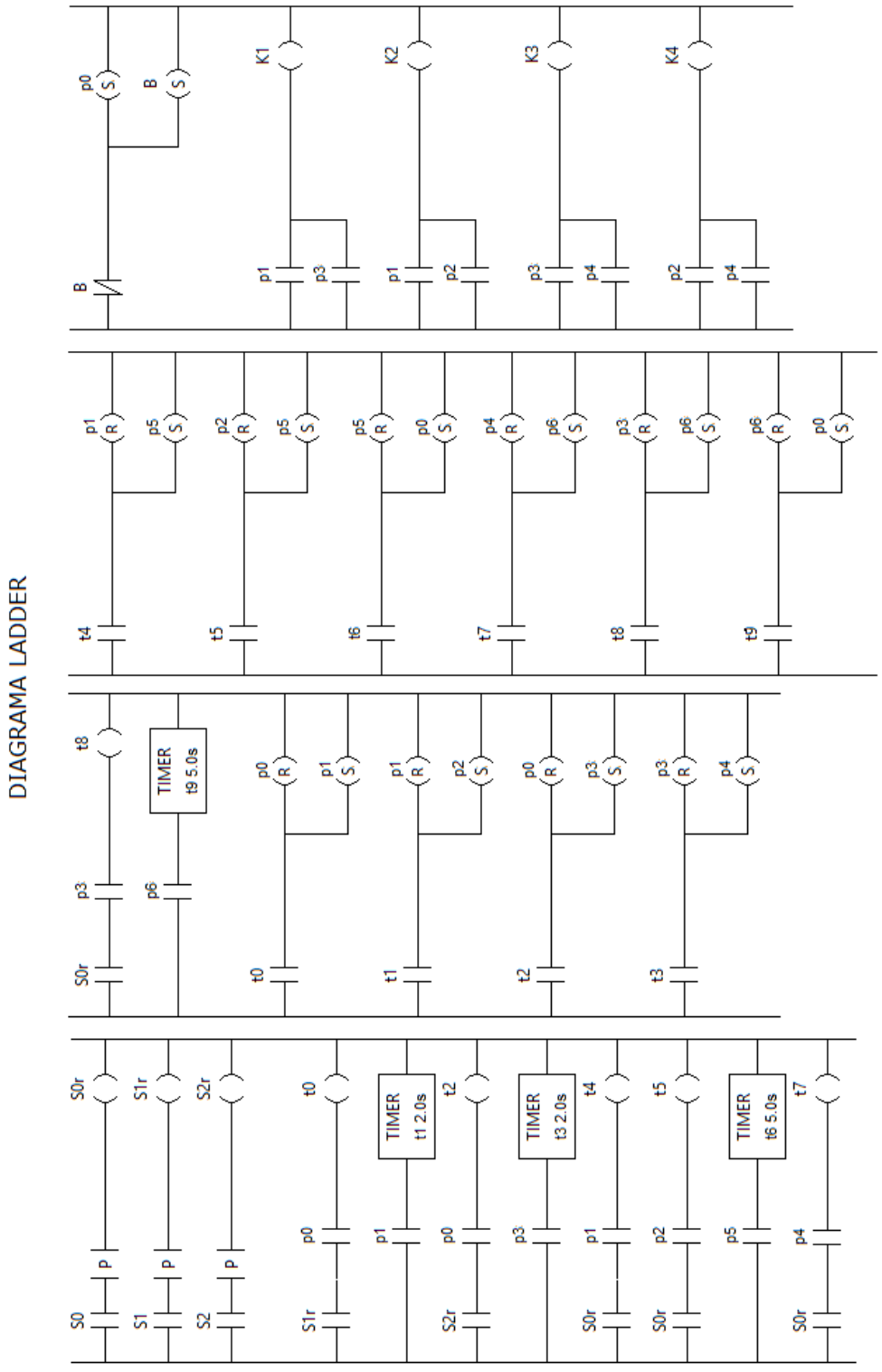


Figura 6-5: Diagrama Ladder com simplificação de blocos

Além desse problema, verificamos também que não é possível utilizar duas bobinas de variáveis iguais ao longo do diagrama. Por esse motivo, no módulo das ações, todos os lugares que acionam determinada bobina foram colocados em paralelo com ela, de forma que só haja uma instancia de cada bobina. A Figura 6-5 mostra o diagrama com os ajustes feitos.

Sendo assim, foram criadas as variáveis (tags) mostradas no Apêndice B. Para criar as tags no STEP7, basta clicar no item *PLC Tags* do painel *Project Tree*, à esquerda da interface, como mostrado na Figura 6-6. Note que as variáveis de entrada são do tipo I, as de memória do tipo M e as de saída do tipo Q, como explicado na seção 3.2.2.

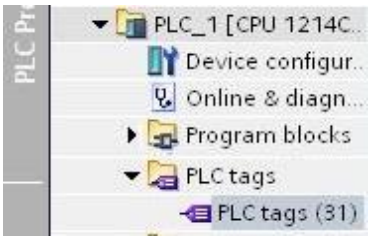


Figura 6-6: Inserindo tags no STEP7

Em seguida, foi inserido o diagrama Ladder mostrado no Apêndice C, que corresponde ao diagrama da Figura 6-5. Para inserir os componentes do diagrama Ladder no STEP7, basta clicar no item *Main [OB1]* do painel *Project Tree*, como mostrado na Figura 6-7. A janela principal se transformará numa janela de edição, com diversos elementos de diagramas Ladder disponíveis para inserção.

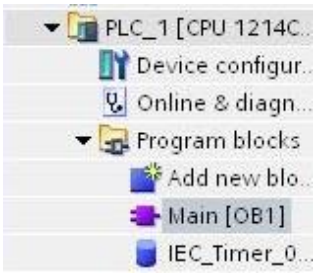


Figura 6-7: Opção de inserção do diagrama Ladder no STEP7

Após a inserção do Ladder no STEP7, basta clicar no botão marcado na Figura 6-8 para baixar o programa para o CLP.

Figura 6-8: Baixando o programa para o CLP

Feito isso, a janela da Figura 6-9 será mostrada; basta clicar em *Load* e o programa será baixado. Em seguida, a tela da Figura 6-10 será exibida; é necessário marcar a caixa *Start All* e, então, clicar em *Finish*, para que o CLP entre em modo de execução. Após esses passos, o CLP está pronto para operar o sistema.

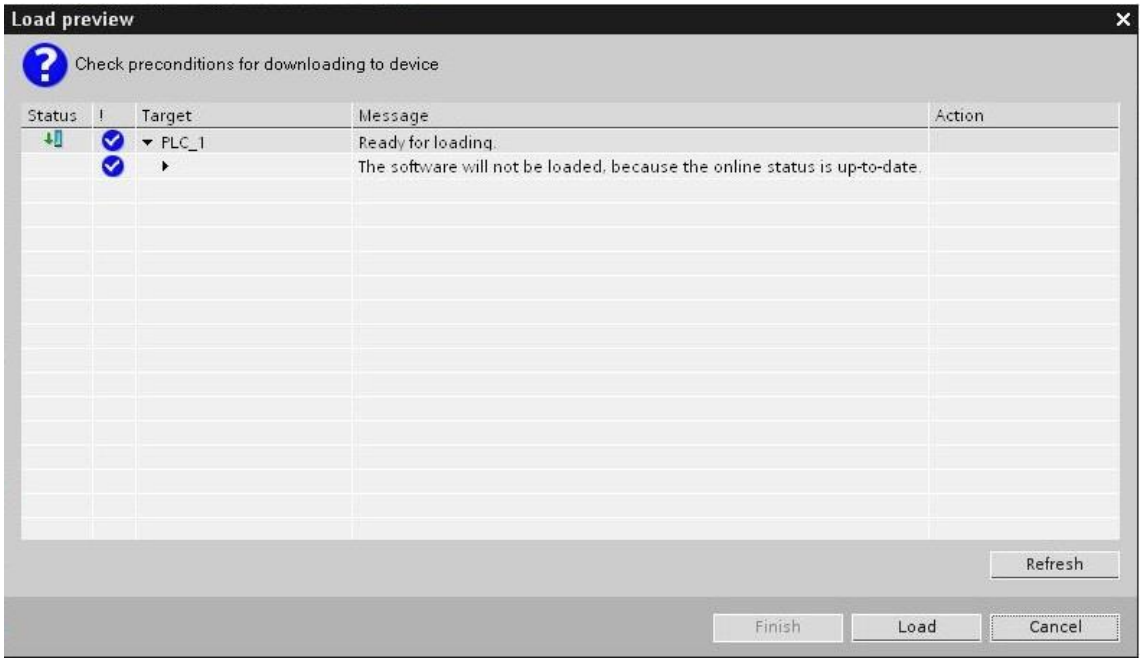


Figura 6-9: Tela de download do programa para o CLP

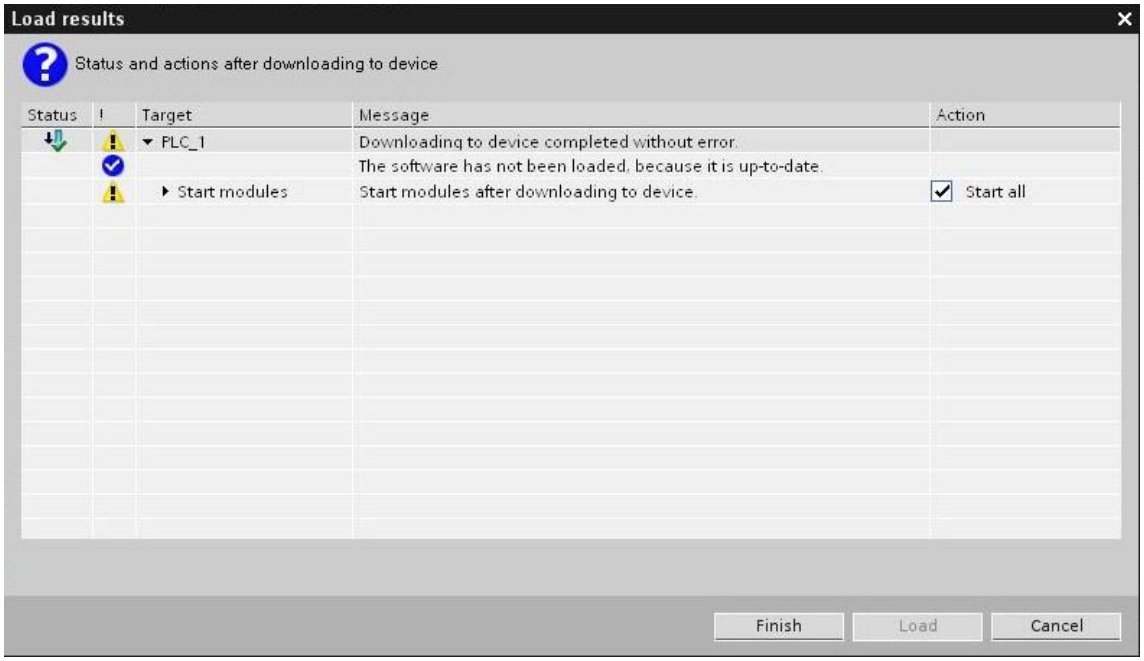


Figura 6-10: Tela de execução do CLP

6.4 Montagem em Bancada

O CLP já programado foi então levado para uma bancada contendo diversos relés eletromecânicos, fusíveis, LEDs e botoeiras com os terminais acessíveis, como mostra a Figura 6-11. O motor de indução a ser controlado encontra-se debaixo da bancada, com os terminais dos dois lados de cada bobina também acessíveis, como mostra a Figura 6-12.

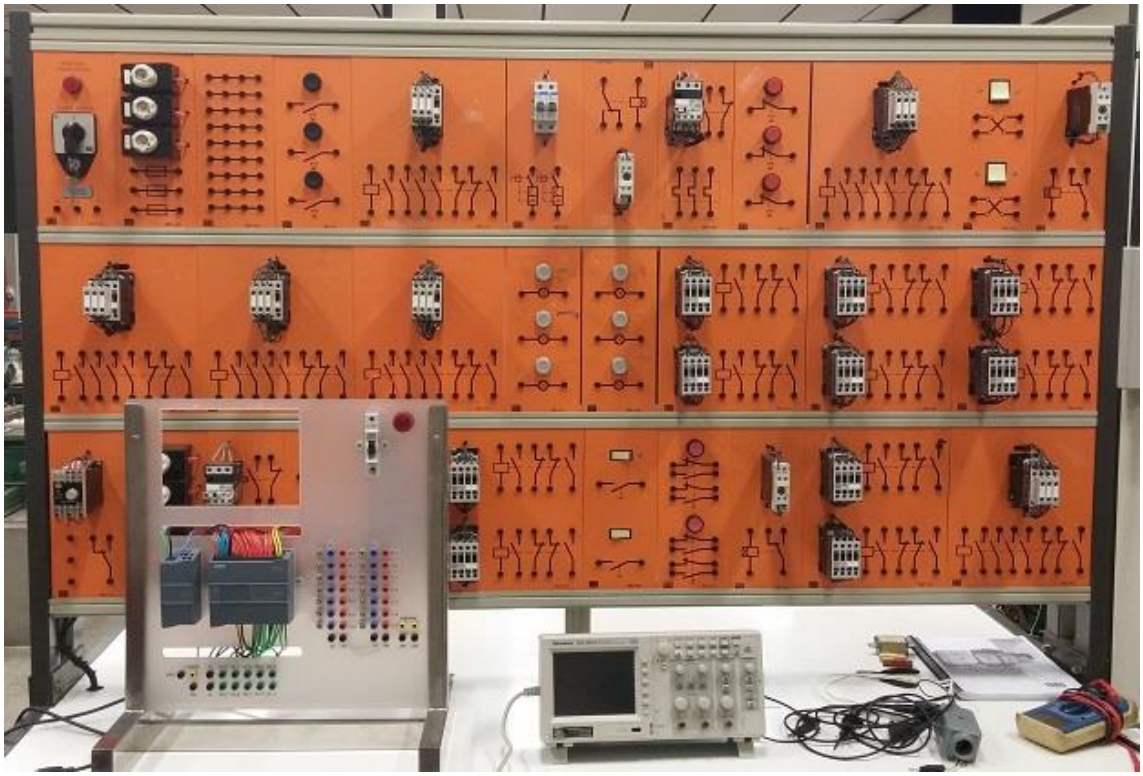


Figura 6-11: Bancada de controle do motor

Figura 6-12: Motor de indução trifásico com os terminais das bobinas acessíveis

As conexões dos relés com os motores foi feita de forma idêntica à mostrada na Figura 6-1, exceto que não foram utilizados relés térmicos, e foram conectados LEDs em cada relé eletromecânico, para indicar qual relé está fechado no momento. As bobinas dos relés eletromecânicos foram conectadas por meio dos relés do CLP à fonte trifásica. As botoeiras do painel foram conectadas de forma a fechar uma tensão de 24 V, gerada pelo próprio CLP, nas entradas digitais do mesmo. A montagem final do painel ficou como na Figura 6-13.

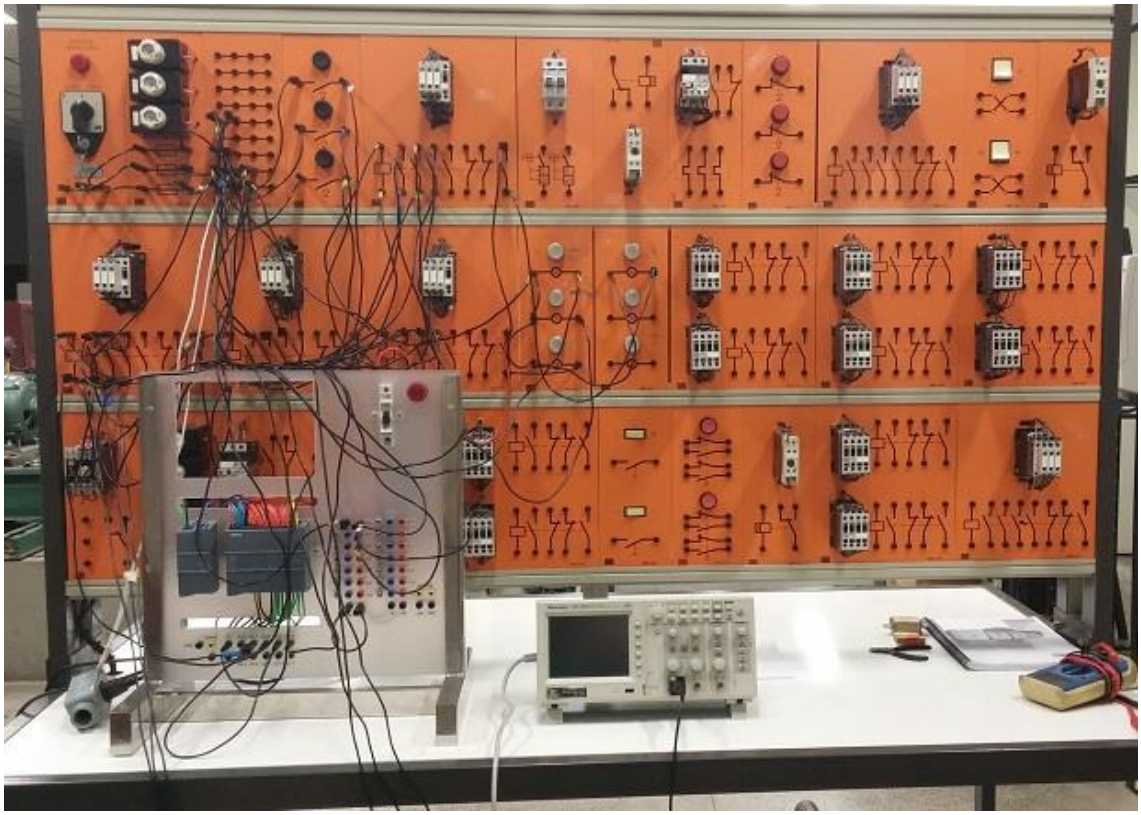


Figura 6-13: Bancada com fios conectados

6.5 Resultados

Com todo sistema montado e energizado, iniciaram-se os testes. Quando apertamos a botoeira correspondente a S1, observamos que o motor partiu para o sentido anti-horário, e os LEDs correspondentes a K1 e K2 se acenderam, como mostra a Figura 6-14. Após 2s, observamos que os LEDs mudaram para K2 e K4, como mostra a Figura 6-15. Ao apertarmos a chave S0 em qualquer um desses momentos, o motor começa a desacelerar, e nenhum botão funciona durante 5s, como esperado.

Figura 6-14: LEDs de K1 e K2 acesos

Figura 6-15: LEDs de K2 e K4 acesos

Apertando a botoeira correspondente a S2, observamos que o motor partiu para o sentido horário, e os LEDs correspondentes a K1 e K3 se acenderam, como mostra a Figura 6-16. Após 2s, observamos que os LEDs mudaram para K3 e K4, como mostra a Figura 6-17. Ao apertarmos a chave S0 em qualquer um desses momentos, o motor começa a desacelerar, e nenhum botão funciona durante 5s, como esperado.

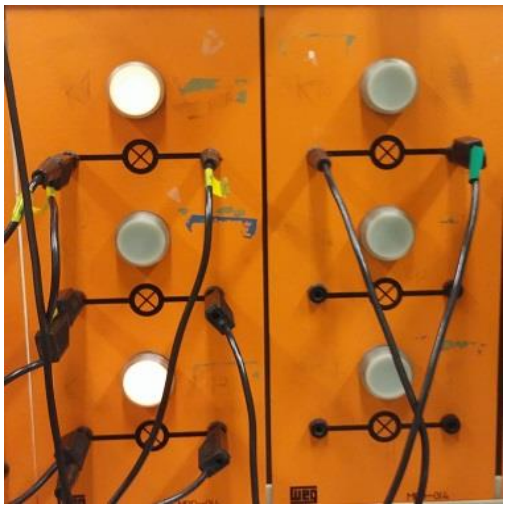


Figura 6-16: LEDs de K1 e K3 acesos

Figura 6-17: LEDs de K3 e K4 acesos

Além da verificação dos LEDs, é conveniente medir a tensão que está sendo aplicada às bobinas do motor, para garantir que o esquema Y-Δ está montado corretamente. Um osciloscópio foi utilizado para medir a tensão em uma das bobinas do motor durante sua operação. O resultado está disposto na Figura 6-18.

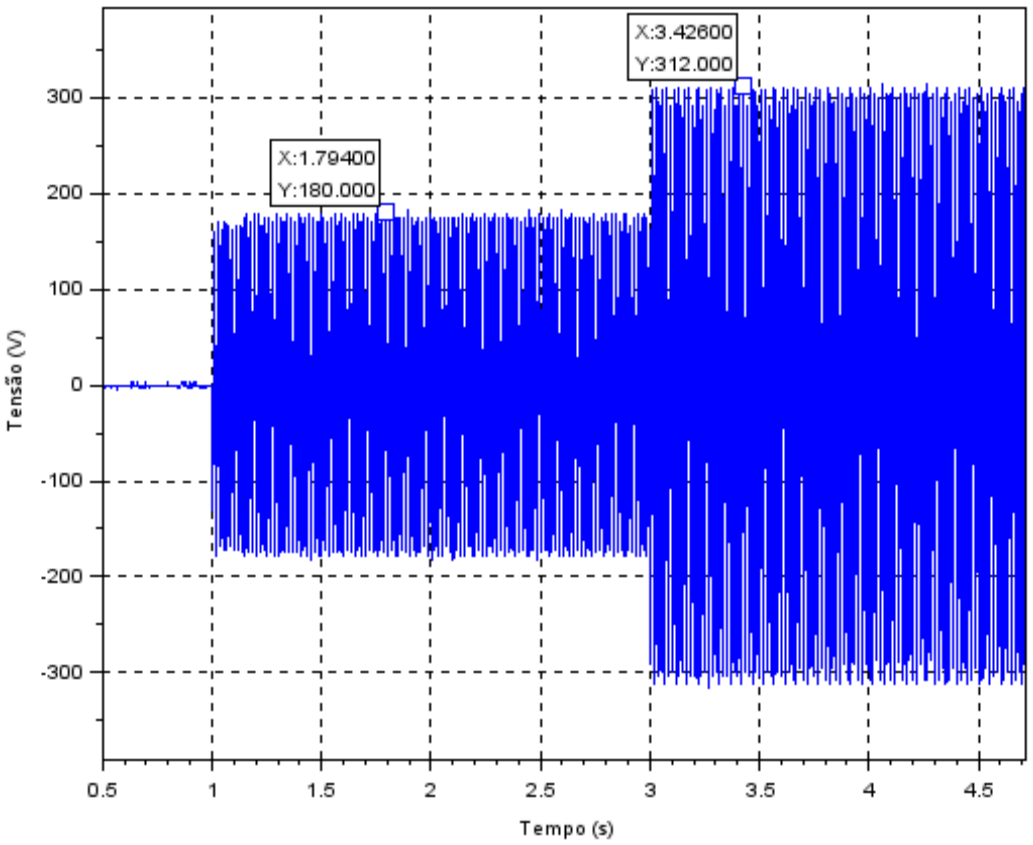


Figura 6-18: Tensão em uma das bobinas do motor de indução

Podemos verificar que quando a botoeira de partida é apertada, uma tensão de $180\ V_p = 127\ V_{rms}$ é aplicada sobre o motor; após exatos 2s, a tensão sobe para $312\ V_p = 221\ V_{rms}$, o que corresponde a uma partida Y-Δ para uma tensão de linha de 220 V.

Capítulo 7

Conclusão e Versões Futuras

Os Sistemas a Eventos Discretos estão presentes em qualquer setor da indústria, sendo de extrema importância o seu estudo e modelagem. Dentre os modelos possíveis, as Redes de Petri Interpretadas para Controle se destacam por sua simplicidade e intuitividade

O CLP é um aparelho versátil e robusto, capaz de controlar sistemas de qualquer porte através de sensores e atuadores. Sua principal linguagem de programação é o Ladder, uma linguagem visual que simula a lógica de contatos elétricos.

A criação do método sistemático proposto em [1] para a conversão de RPICs em diagramas Ladder, permitiu que modelos teóricos possam ser utilizados diretamente na prática para o controle de DES por meio de um CLP. Nesse contexto, o PETRILab se destaca por fazer essa conversão computacionalmente, com apenas um clique. Além da conversão instantânea de RPIC para Ladder, o programa se destaca pela facilidade de inserção e capacidade de simulação da rede, podendo servir também para estudos e testes.

Apesar do PETRILab cumprir com todos os requisitos propostos em sua criação, existem recursos e melhorias que poderiam facilitar ainda mais a modelagem, simulação ou conversão de RPICs, que possivelmente serão incluídos em versões futuras. Podemos citar alguns deles:

- Possibilidade de selecionar itens – únicos ou múltiplos – para então movê-los ou removê-los
- Funções Desfazer e Refazer
- Possibilidade de salvar imagens dos grafos e diagramas em outros formatos que não o *eps*

- Tradução da interface para o inglês

Outros recursos e correções de bugs também podem ser sugeridos através do menu *Ajuda* do programa, como foi dito na seção 5.3.12.

Referências Bibliográficas

[1] M. V. Moreira e J. C. Basílio, "Bridging the Gap Between Design and Implementation of Discrete-Event Controllers," *IEEE Transactions on Automation Science and Engineering*, pp. 48-65, 2013.

[2] R. M. Stephan, Acionamento, Comando e Controle de Máquinas Elétricas, 1ª ed., Rio de Janeiro: Ciência Moderna, 2013.

[3] C. G. Cassandras e S. Lafortune, Introduction to Discrete Event Systems, 2ª ed., New York, NY: Springer, 2008.

[4] C. M. Franchi e V. L. A. d. Camargo, Controladores Lógicos Programáveis, 1ª ed., São Paulo: Érica, 2008.

[5] "Sourceforge," [Online]. Available: http://www.sourceforge.net/. [Acesso em 23 02 2015].

[6] "WinRAR," [Online]. Available: http://www.win-rar.com/. [Acesso em 23 02 2015].

[7] F. P. L. Junior, "Automação de uma Planta Mecatrônica Modelada por uma Rede de Petri Interpretada para Controle," Rio de Janeiro, 2014, pp. 33-48.

Apêndice A

Manual Técnico do PETRILab

Este capítulo tem como intuito apresentar a arquitetura base do PETRILab, de modo a permitir a implementação de novas funções ou correção de *bugs* por futuros desenvolvedores. O programa não é trivial, e é todo orientado a objetos, portanto o desenvolvedor deve estar bem familiarizado com esse tipo de programação. Também é recomendado um amplo conhecimento do módulo de interface gráfica do Pyhon, o *Tkinter*. A estrutura de cada módulo será explicada, assim como a relação entre eles.

O PETRILab consiste basicamente de 5 módulos, cada um contido em um arquivo separado no diretório principal. São eles:

- O módulo *CIPN*, que contém a classe da RPIC em si, e classes para temporizadores
- O módulo *ladder*, que contém as classes correspondentes a cada elemento do diagrama Ladder
- O módulo *conversao*, responsável por fazer a conversão de uma RPIC em um diagrama Ladder
- O módulo *diagrama*, responsável por desenhar o diagrama gerado na área de desenho
- O módulo *petrilab*, que contém toda a interface gráfica do programa

Este capítulo está estruturado da seguinte forma: Na seção A.1 será apresentado o módulo *CIPN.py*; na seção A.2 será apresentado o módulo *ladder*; na seção A.3 será apresentado o módulo *conversao*; na seção A.4 será apresentado o módulo *diagrama*; por fim, na seção A.5 será apresentado o módulo *petrilab*.

A.1 Módulo CIPN

O módulo CIPN está contido no arquivo *CIPN.py* no diretório principal do PETRILab. Sua principal classe é a classe CIPN, que cria um objeto correspondente a uma RPIC. Como mostra a Figura A-1, seus argumentos de criação são basicamente os da RPIC apresentada na seção 2.4, com exceção de *parent*, que, em caso de uso da simulação da interface gráfica, precisa conter uma referência ao objeto *Programa* do programa principal.

```
class CIPN():
    def __init__(self, Pc, Tc, Prec, Postc, x0c, Inc, C, Ec, lc, D, ld, A, la, parent=None):
        self.parent = parent
        self.x0c = x0c
        self.Pc, self.Tc = Pc, Tc
        self.Prec, self.Postc, self.Inc = Prec, Postc, Inc
        self.C, self.Ec, self.lc, self.D = C, Ec, lc, D
        self.ld, self.A, self.la = ld, A, la
        self.x, self.toFire = x0c, [False]*Tc
        self.timers = []
        for t in range(Tc):
            self.timers.append([])
        self.eimpulse = [1]*len(Ec)

    def setImpulse(self, e, x):...
    def isFireable(self, t):...
    def setFireable(self):...
    def setCondition(self, c, b):...
    def initialize(self):...
    def fire(self, t):...
    def event(self, e):...
    def run(self):...
    def check_timers(self, t):...
    def __str__(self):...
    def __repr__(self):...
```

Figura A-1: Classe CIPN

A função *initialize()* serve apenas para verificar se alguma ação deve ser acionada assim que a rede é criada. Já a função *setImpulse()* tem como objetivo definir o tipo de borda de um determinado evento: subida ou descida.

Tendo sido criado o objeto da rede, o estado da rede evoluirá na ocorrência de eventos ou mudança de condições, que são representados pela ocorrência das funções *event()* e *setCondition()*, respectivamente. Basicamente, na ocorrência de uma das duas funções supracitadas, o programa modifica uma *tag* correspondendo a ocorrência de um evento *e* ou uma condição *c*, e chama a rotina *setFireable()* para definir quais transições estão aptas a disparar após essa mudança, utilizando a função *isFireable()* para os testá-las individualmente. Feito isso, o programa chama a função *fire()* para realizar o disparo

das transições, modificando o vetor de estados x, e para acionar as ações que devem ser ativadas. A Figura A-2 mostra a função *isFireable()* expandida. Note que a função é bem simples, sendo puramente um teste das condições de habilitação e disparo de uma transição *t*.

```
def isFireable(self, t):
    for i in range(self.Pc):
        if self.x[i]<self.Prec[i][t]:
            return False
        if self.Inc[i][t]:
            if self.x[i]>=self.Inc[i][t]:
                return False
    if not self.C[self.lc[t][0]] or not self.Ec[self.lc[t][1]]:
        return False
    return True
```

Figura A-2: Função isFireable() da classe CIPN

Em certos casos, as transições de uma rede podem ser disparadas em sequência. Isso ocorre quando elas não estão associadas a eventos, e foram habilitadas depois do disparo de outra transição. Por esse motivo, o programa não pode executar apenas um ciclo de evolução de estados, sendo necessário checar repetidamente se existem transições a serem disparadas após a ocorrência de um evento. Isso é alcançado através da função *run()*, a função que de fato é chamada quando um evento ocorre na interface. A função pode ser vista na Figura A-3.

```
def run(self):
    contador=0
    while self.toFire != [0]*self.Tc:
        contador += 1
        if contador > 2000:
            abortar = askquestion('Looping infinito detectado',
                                  'usuário.
                                  'Deseja ab
            if abortar=='yes':
                self.parent.parar()
                contador=0
                return
            else:
                contador=0
        for t in range(self.Tc):
            if self.toFire[t] and self.D[self.ld[t]]==0:
                self.fire(t)
            elif self.toFire[t] and self.D[self.ld[t]] !=0:
                self.timers[t]+=[Delay(self, t)]
                self.timers[t][-1].start()
```

Figura A-3: Função run() da classe CIPN

Para a simulação dos temporizadores, são criados objetos da classe *Delay*. Essa classe é uma subclasse de *Thread*, a classe do Python que gerencia novas tarefas. Sendo assim, o programa se torna multitarefa, ficando muito mais propenso a travamentos. É sempre boa prática finalizar as tarefas não utilizadas para evitar esse problema. A classe *TimerAct*, também derivada de *Thread*, é apenas um delay para controlar o tempo que a luz indicadora de ação acionada fica acesa na interface gráfica.

A.2 Módulo ladder

O módulo *ladder* é o responsável por criar as classes dos elementos do diagrama Ladder, e está contido no arquivo *ladder.py* no diretório principal. Os objetos dessa classe são criados com parâmetros referentes ao tipo de elemento e os rótulos necessários para sua exibição.

Todos os objetos das classes desse módulo contém a função *draw()*, que tem como argumentos um objeto de *Canvas* – correspondente a uma área de desenho no módulo de interface gráfica do Python – e as coordenadas *x* e *y* que o objeto deve ser desenhado. A Figura A-4 mostra essa função expandida para um objeto correspondente a um bloco COMP.

```
class Comp:
    def __init__(self, l, t, q):...
    def __str__(self):...
    def draw(self, c, xy):
        R = 20
        c.create_rectangle(xy[0],xy[1]-R,xy[0]+3*R,xy[1]+R)
        c.create_text(xy[0]+1.5*R,xy[1]-R/2., text=self.s)
        c.create_text(xy[0]+1.5*R,xy[1]+R/2., text=self.s2, font=('Purisa',8))
        return [xy[0]+3*R,xy[1]]
    def size(self):...
```

Figura A-4: Classe COMP do módulo ladder

A função *size()* retorna as dimensões do bloco, para posterior utilização pelo módulo *diagrama*, responsável pelo deseho. A função *__str__()* representava os objetos na forma de texto, sendo utilizada em versões anteriores do PETRILab, e pode ser ignorada ou apagada.

A.3 Módulo conversao

O módulo conversão é responsável por realizar a conversão de RPIC em diagrama Ladder proposta no Capítulo 4, e está contido no arquivo *conversao.py*.. Ele consiste basicamente de uma única função, *convert()*, que leva como argumentos um objeto CIPN do módulo CIPN, e uma lista com os rótulos dos elementos, no formato mostrado no comentário acima da função. A Figura A-5 mostra esse comentário, além da primeira parte da função, que faz a criação do *Módulo dos Eventos Externos*.

```
# labels = [[plabels],[tlabels],[elabels],[clabels],[alabels]]

def convert(pn, labels=None):
    # Modulo de eventos externos
    m1 = Ladder()
    l = Linha()
    for i in range(len(pn.Ec)-1):
        if True in map(lambda x: x[1]==i, pn.lc):
            if labels:
                l += Contato(labels[2][i], 1)
            else:
                l += Contato('S'+str(i), 1)
            if pn.eimpulse[i] == 1:
                l += Borda(1)
                if labels:
                    l += Bobina(labels[2][i]+'r', 1)
                else:
                    l += Bobina('S'+str(i)+'r', 1)
            else:
                l += Borda(0)
                if labels:
                    l += Bobina(labels[2][i]+'f', 1)
                else:
                    l += Bobina('S'+str(i)+'f', 1)
            m1 += l
            l = Linha()
```

Figura A-5: Função *convert()* do módulo *conversao*

Note que a conversão segue exatamente o esquema proposto no Capítulo 4. Primeiramente, o programa cria uma instância de um diagrama Ladder e de uma linha. Para cada evento da rede, o programa checa se ele está associado a alguma transição, para então começar a inserção de elementos em sua linha correspondente. Um contato é então inserido, seguido de um contato de borda, e finalizando com uma bobina.

O teste *if labels*, realizado diversas vezes nessa função pode confundir o desenvolvedor. Ocorre que, originalmente, existia a opção de não fornecer nenhum rótulo dos elementos à função *convert()*; nesse caso, a função gerava automaticamente rótulos com numeração ordenada para eles. Com a criação da interface gráfica, esse caso nunca irá ocorrer, portanto os testes não são necessários.

Todos os outros módulos do diagrama a ser construído seguem o mesmo esquema de criação: criam-se as linhas necessárias, e adicionam-se os elementos, todos provenientes do módulo *ladder*. A função retorna então uma lista contento as linhas de todos os módulos do diagrama convertidos.

A.4 Módulo *diagrama*

O módulo *diagrama* é responsável por fazer o desenho do diagrama Ladder gerado em um *Canvas*. e está contido no arquivo *diagrama.py*. Ele contém uma classe *Diagrama*, que ao ser inicializada, cria uma janela e um *Canvas* através do módulo de interface *Tkinter*; e uma função *gerar()*, que é a responsável por fazer de fato o desenho do diagrama nesse *Canvas.*

A função *gerar()* leva como argumentos um objeto de RPIC, um objeto da classe *Programa*, presente na interface gráfica, e os rótulos dos elementos do diagrama Ladder. Internamente ele chama a função *convert()*, do módulo *conversao* apresentado na seção A.3, que retorna uma lista com os elementos do diagrama Ladder de cada módulo criado. A Figura A-6 mostra o trecho da função que desenha o Módulo da Inicialização.

```
# Desenho do Módulo 4

if len(b[3].x[0].x[1].e)>1:
    xy = app.xy
    xy = b[3].x[0].x[0].draw(app.c, xy)
    app.c.create_line(xy[0], xy[1], larg+5-b[3].x[0].x[1].size()[0], xy[1])
    xy = [larg+5-b[3].x[0].x[1].size()[0], xy[1]]
    xy = b[3].x[0].x[1].draw(app.c, xy)
    app.xy=xy
    if b[4].x:
        app.newline(70)
```

Figura A-6: Trecho da função de desenho do diagrama Ladder

Na Figura A-6, *b* representa a lista com os elementos de cada módulo. Sendo assim, *b[3]*, o quarto elemento da lista, corresponde ao objeto da classe *Ladder* – do módulo *ladder* – correspondente ao Módulo da Inicialização. A propriedade *x* dessa classe armazena as linhas do módulo, então *x[0]* acessa sua primeira linha – um objeto da classe *Linha*, do módulo *ladder* – que também tem uma propriedade *x* que armazena seus elementos. Acessando então o item *x[1]* dessa linha, obtemos o segundo elemento da linha, que, no caso do Módulo da Inicialização, é uma associação em paralelo, representada por um objeto da classe *Paralelo* do módulo *ladder*. A propriedade *e* desse objeto armazena os elementos associados em paralelo. Portanto, a função o teste *len(b[3].x[0].x[1].e)>1* testa se a quantidade de elementos em paralelo no Módulo da Inicialização é maior que um, pois se ela fosse igual a um – a bobina Set que está sempre presente – não haveria necessidade do desenho desse módulo!

As linhas restantes desse trecho de código basicamente chamam as funções *draw()* dos elementos do módulo, além de desenhar a linha que os une, através da função de desenhar linha do módulo *Tkinter*. A variável *xy* contém uma tupla que representa a posição atual de desenho do módulo no *Canvas*.

O restante do código segue o mesmo esquema do trecho mostrado, sendo necessária uma boa análise para entender o que de fato está sendo testado ou inserido. Vale notar que números os inteiros com valores aleatórios distribuídos pelo código representam as dimensões de alguns componentes, e foram descobertos a partir de tentativa e erro.

A.5 Módulo petrilab

O módulo *petrilab*, contido no arquivo *petrilab.pyw* é responsável pela criação da interface gráfica do programa. É um módulo extenso, contendo pouco mais de 2.400 linhas de código; por esse motivo, recomenda-se o uso de alguma interface de desenvolvimento que permita a minimização de funções e classes, de forma a facilitar a navegação pelo código.

O módulo implementa diversas classes, que contém instruções para o desenho, movimentação e remoção de diversos itens no *Canvas* principal. A Figura A-7 mostra a classe *Lugar*, com a função *mover()* expandida. Essa função supostamente é chamada ao

mover-se o mouse enquanto o botão esquerdo está pressionado sobre um lugar, portanto ela será chamada diversas vezes durante uma única movimentação. As linhas antes de código antes do *if* servem apenas para determinar a posição atual do ponteiro do mouse, levando em conta as barras de rolagem *sh* e *sv* da janela principal. As funções obscuras utilizadas para essa parte podem ser consultadas em alguma documentação do *Tk*, programa que originou o módulo *Tkinter* do Python. Em seguida, o programa apenas move os objetos relativos ao lugar no *Canvas*, chama a função *update_moving()* – que serve pra mover os textos das ações impulsionais associadas ao lugar – e a função *redraw()* de todos os arcos da rede, que os redesenha para acompanhar a movimentação do lugar.

```
class Lugar:
    def __init__(self, parent):...
    def follow(self, e,add=0):...
    def mover(self,add=0,xs=None,ys=None):
        self.parent.saved=False
        parent=self.parent
        a, b = self.parent.sh.get()
        perch = a/(1-b+a)
        a, b = self.parent.sv.get()
        percv = a/(1-b+a)
        xb = perch*(CANVASSIZEX-self.parent.c.winfo_width())
        yb = percv*(CANVASSIZEY-self.parent.c.winfo_height())
        x,y = parent.winfo_pointerx(), parent.winfo_pointery()
        x,y = x-parent.canv.winfo_rootx(), y-parent.canv.winfo_rooty()
        if xs is not None and ys is not None:
            x,y = xs+15, ys+15
            xb,yb = 0,0
        self.canvas.coords(self.oval, (x-15+xb,y-15+yb+add,x+15+xb, y+15+yb+add))
        self.canvas.coords(self.elabel, (x+xb, y+yb+add))
        self.canvas.coords(self.label, (x+xb, y-16+yb+add))
        self.update_moving()
        for i in self.parent.arcos+self.parent.inibidores:
            i.redraw()
```

Figura A-7: Classe *Lugar* do módulo *petrilab*

O programa principal, no entanto, está todo contido na classe *Programa*, que deriva da classe *Tk*, ou seja, representa uma janela. A método *__init__()* desse programa contém diversas linhas que contém: a definição das propriedades da janela, dos quadros de divisão de região dos botões (*Frames*) e das variáveis de controle; a criação de todos os menus *drop-down* e botões; e a associação das teclas de atalho à funções da classe. A Figura A-8 mostra um trecho que associa teclas de atalho a algumas funções.

```
# Teclas de atalho
self.bind('l', lambda x: self.lugar())
self.bind('t', lambda x: self.transicao())
self.bind('a', lambda x: self.arco())
self.bind('i', lambda x: self.inibidor())
self.bind('e', lambda x: self.evento())
self.bind('c', lambda x: self.condicao())
self.bind('k', lambda x: self.acao())
self.bind('g', lambda x: self.ladder())
self.bind('s', lambda x: self.simular())
```

Figura A-8: Trecho da criação de teclas de atalho da classe *Programa*

Como fica evidente na Figura A-8, a classe *Programa* tem diversas funções que implementam todas as ações de inserção e edição na interface principal. Tomemos como exemplo a função *inita()*, mostrada na Figura A-9. Ela é chamada ao clicar-se em qualquer lugar da área de desenho após clicar-se no botão de *Inserir Arco*, e leva como argumento uma tupla com as coordenadas atuais do mouse. As primeiras linhas do código desassociam o <*Duplo-Clique*>, <*Clique-Direito*> e <*Clique-Arrastado*> das suas funções originais de edição, para evitar edições acidentais ao inserir-se um arco. Em seguida, o programa chama uma função do *Canvas*, que detecta se o usuário clicou em cima de algum item. Em caso negativo, a variável *obj* conterá *None*, e a função terminará. Em caso positivo, a função então testa se o objeto clicado pertence a um lugar ou transição da rede; se pertencer, ele primeiramente testa se o arco é inibidor – terminando a função em caso positivo –, e, em seguida, armazena nas variáveis *ainit* e *arcstart* o objeto clicado. Por fim, ele muda o cursor para uma cruz menor, e associa o clique do mouse à definição do fim do arco, dada pela função *enda()*.

Muitas outras funções diferentes da do exemplo mostrado estão presentes nessa classe, mas não é o objetivo deste trabalho detalha-las. No entanto, todas elas seguem uma linha lógica não muito complicada, e o desenvolvedor será capaz de entende-las após estar familiarizado com o programa.

```
def inita(self, e):
    self.c.unbind('<Double-Button-1>')
    self.c.unbind('<Button-3>')
    self.unbind('<Button-3>')
    self.unbind('<Double-Button-1>')
    self.c.unbind('<B1-Motion>')
    obj = self.c.find_withtag(CURRENT)
    if obj:
        for i in self.transicoes+self.lugares:
            if obj[0] in i.items:
                if self.inserindoin and isinstance(i, Transicao):
                    return
                self.arcstart = i
                self.ainit=i
                self.configure(cursor='plus')
                self.c.bind('<Button-1>', self.enda)
                return
```

Figura A-9: Função *inita()* da classe *Programa*

Apêndice B

Variáveis criadas no STEP7

PLC tags

	Name	Data type	Address
1	s0	Bool	%I0.0
2	s0aux	Bool	%M0.0
3	s1	Bool	%I0.1
4	s1aux	Bool	%M0.1
5	s2	Bool	%I0.2
6	s2aux	Bool	%M0.2
7	s0r	Bool	%M0.3
8	s1r	Bool	%M0.4
9	s2r	Bool	%M0.5
10	p0	Bool	%M0.7
11	p1	Bool	%M2.4
12	p2	Bool	%M2.5
13	p3	Bool	%M2.6
14	p4	Bool	%M2.7
15	p5	Bool	%M3.3
16	p6	Bool	%M3.4
17	t0	Bool	%M1.0
18	t1	Bool	%M1.1
19	t2	Bool	%M1.2
20	t3	Bool	%M1.3
21	t4	Bool	%M1.4
22	t5	Bool	%M1.5
23	t6	Bool	%M1.6
24	t7	Bool	%M1.7
25	t8	Bool	%M2.0
26	t9	Bool	%M2.1
27	B	Bool	%M2.2
28	k1	Bool	%Q0.1
29	k2	Bool	%Q0.2
30	k3	Bool	%Q0.3
31	k4	Bool	%Q0.4

Figura B-1: Variáveis criadas no STEP7

Apêndice C

Diagrama Ladder Inserido no STEP7

Figura C-1: Módulo dos eventos externos

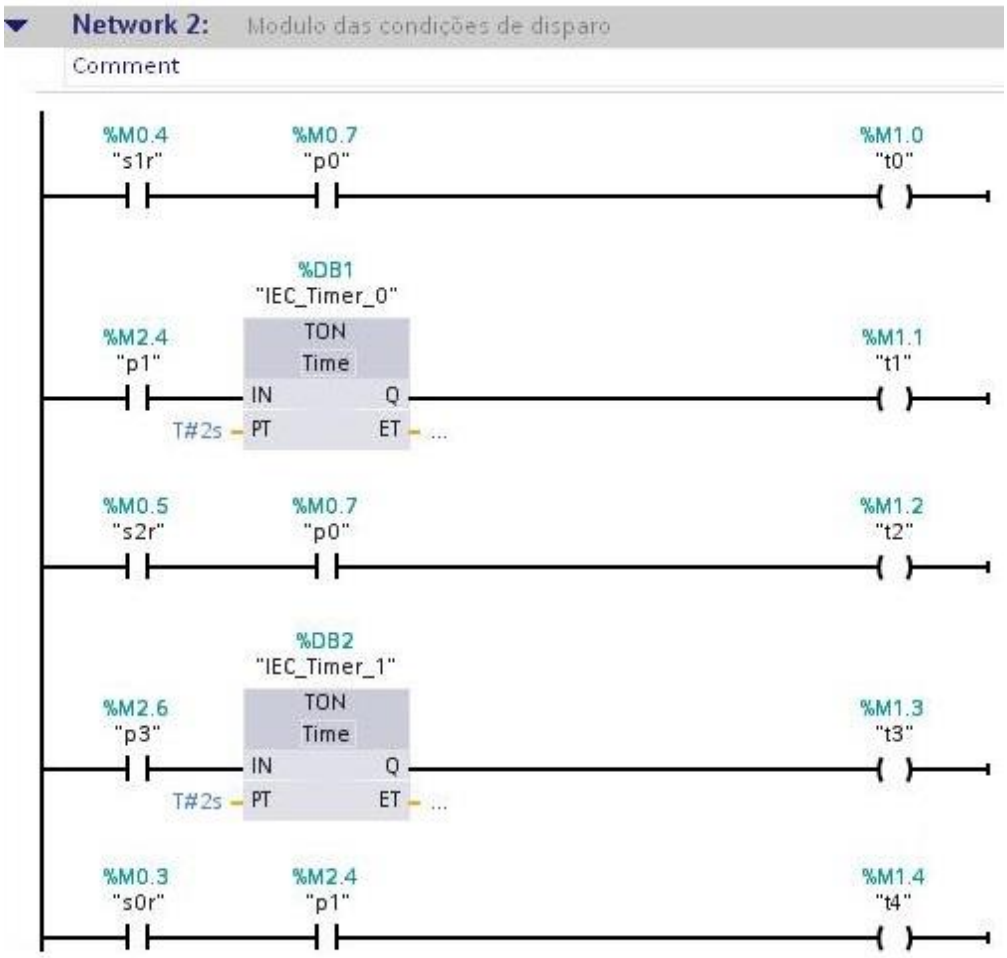


Figura C-2: Módulo das condições de disparo (1)

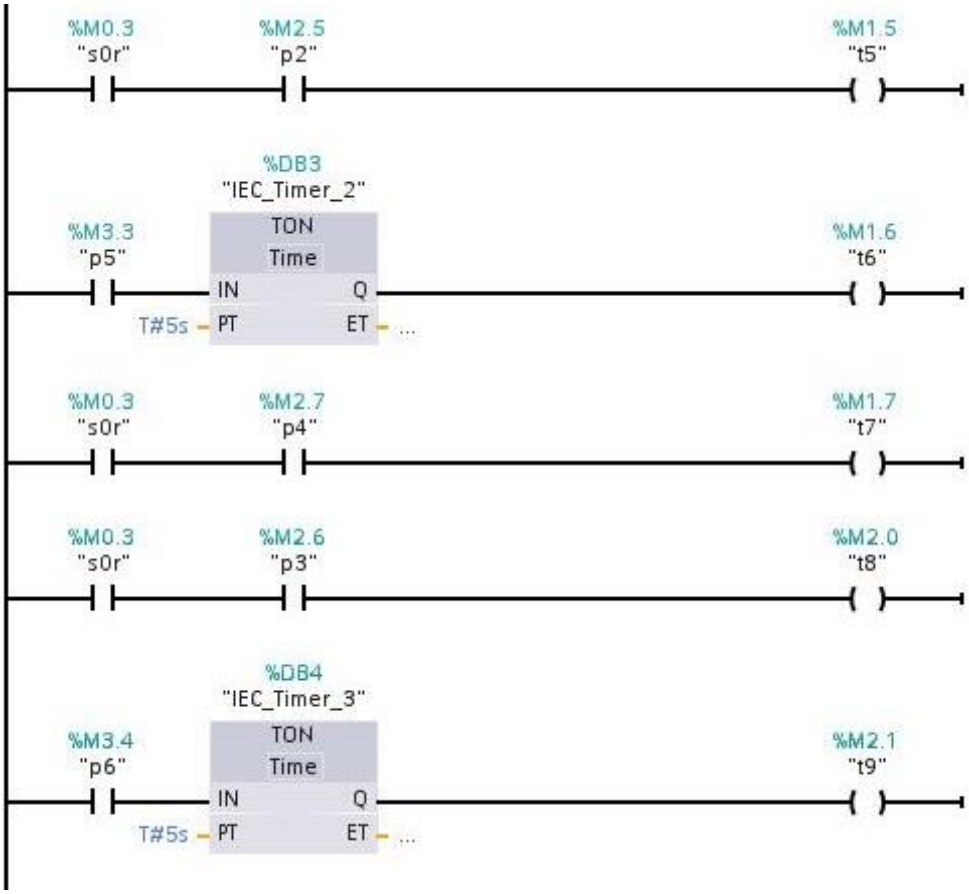


Figura C-3: Módulo das condições de disparo (2)

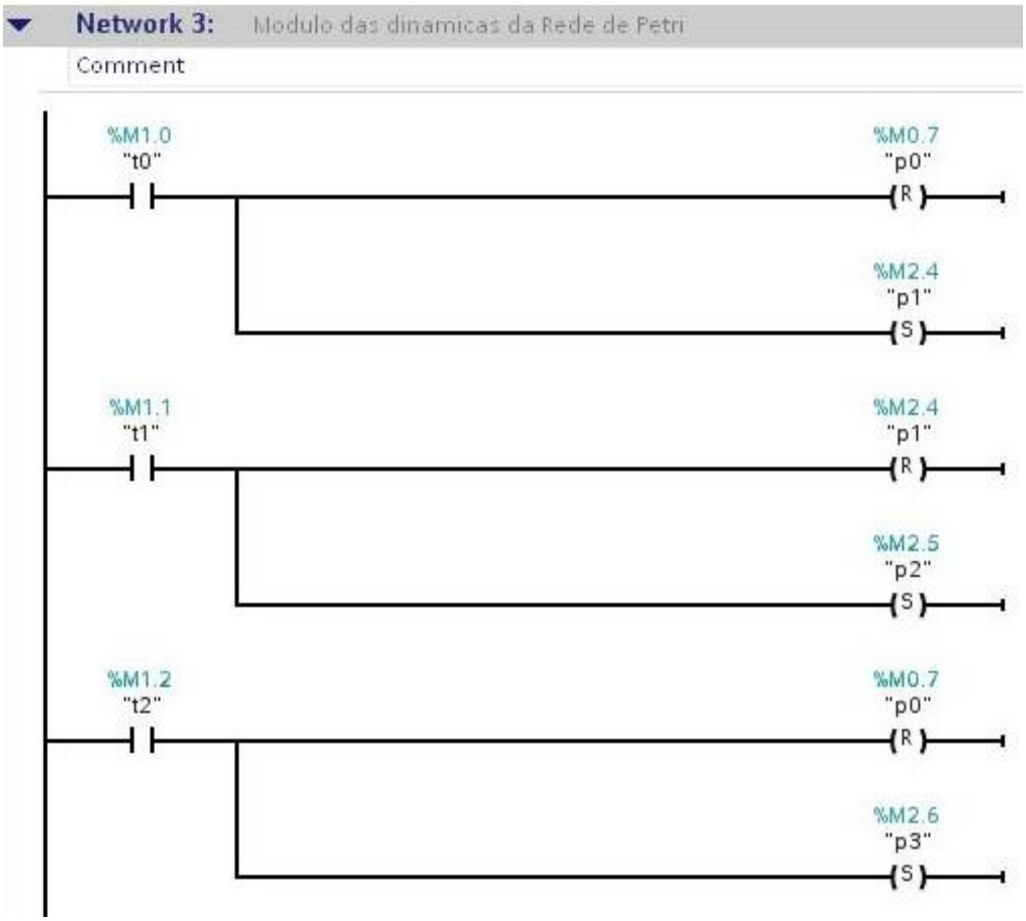


Figura C-4: Módulo das dinâmicas da Rede de Petri (1)

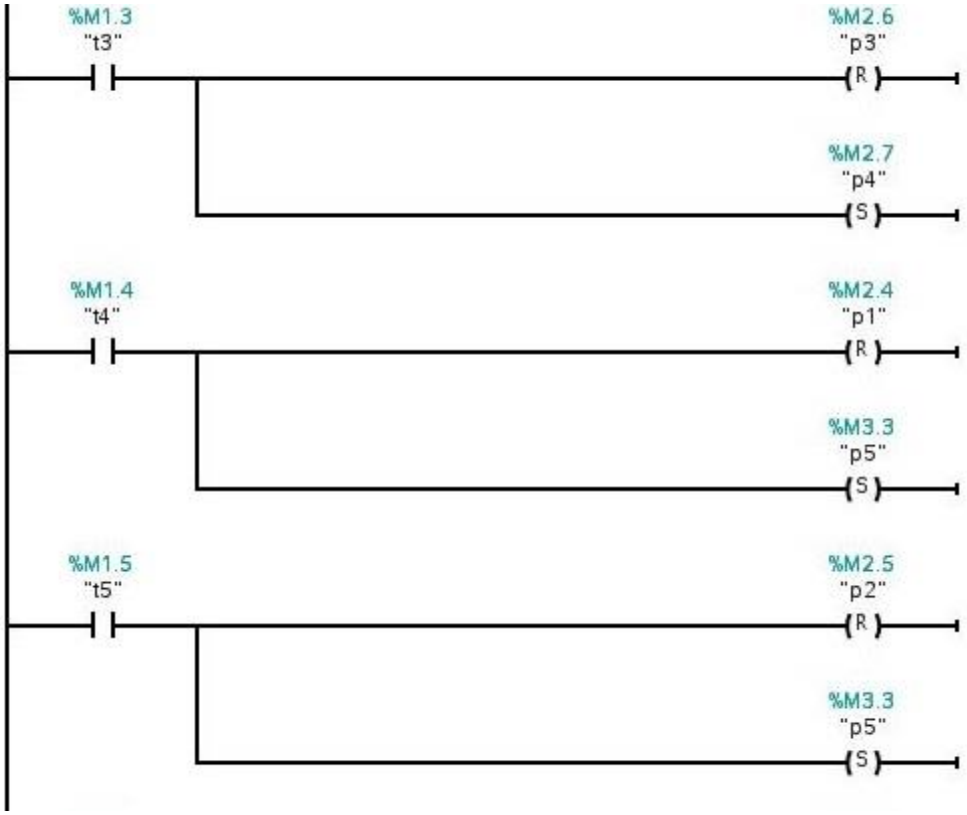


Figura C-5: Módulo das dinâmicas da Rede de Petri (2)

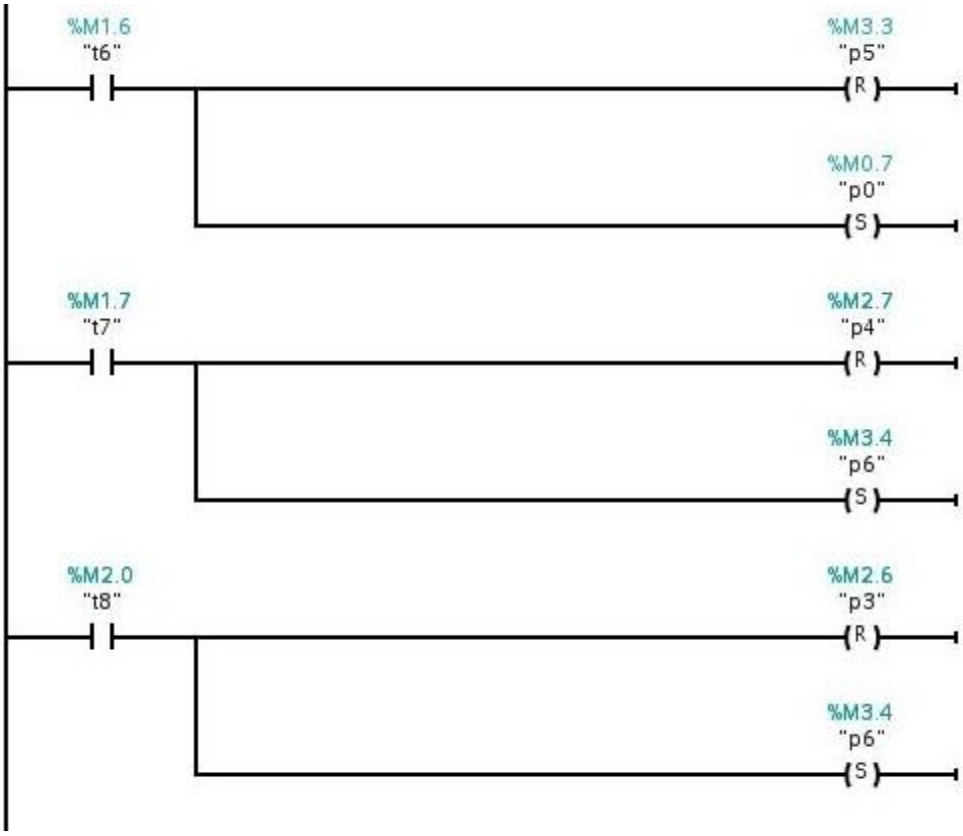


Figura C-6: Módulo das dinâmicas da Rede de Petri (3)

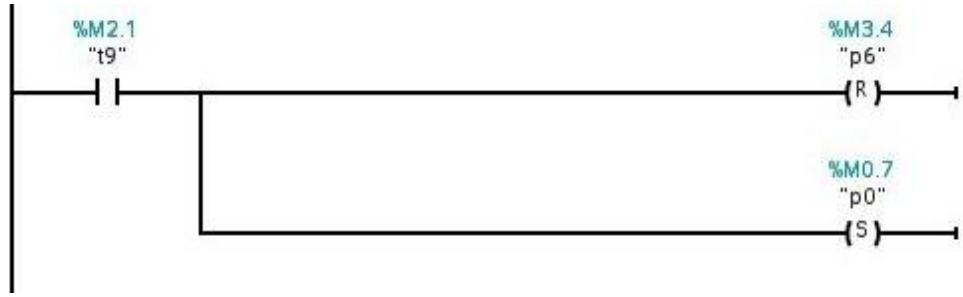


Figura C-7: Módulo das dinâmicas da Rede de Petri (4)

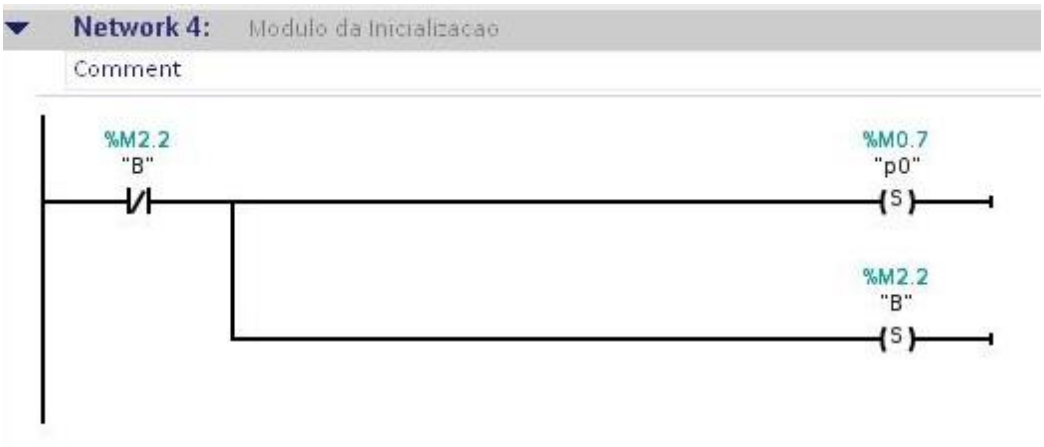


Figura C-8: Módulo da inicialização

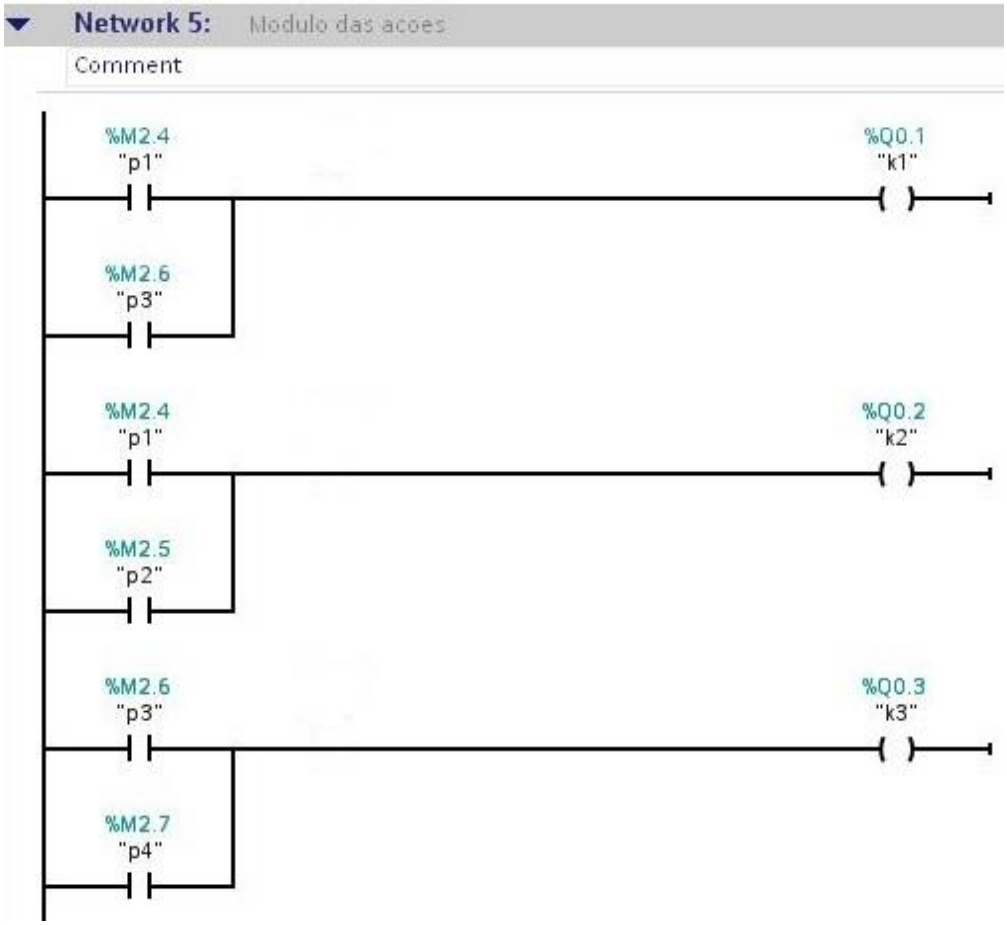


Figura C-9: Módulo das ações (1)

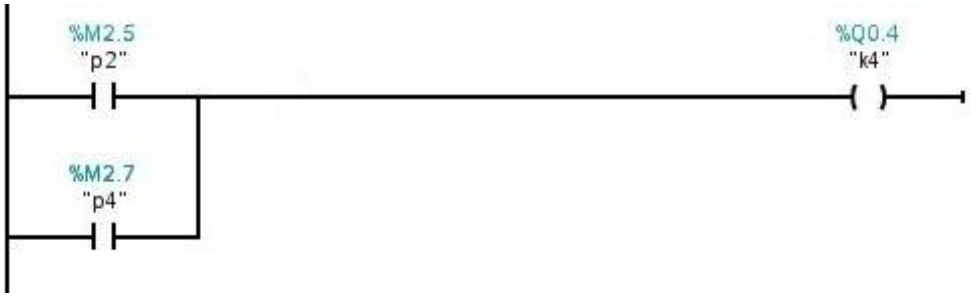


Figura C-10: Módulo das ações (2)