UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Paulo André S. Giacomin

Um Método Interativo de Recuperação de Diagramas Grafcet e de Estados a partir de Diagramas Ladder usando Diagramas Binários de Decisão

VITÓRIA
2005

Livros Grátis

http://www.livrosgratis.com.br

Milhares de livros grátis para download.

Paulo André S. Giacomin

Um Método Interativo de Recuperação de Diagramas Grafcet e de Estados a partir de Diagramas Ladder usando Diagramas Binários de Decisão

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Engenharia Elétrica, na área de concentração em automação, na sub-área de desenvolvimento de sistemas de automação.
Orientador: Prof. Dr. Hans Jorg Andreas Schneebeli

VITÓRIA
2005

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Central da Universidade Federal do Espírito Santo, ES, Brasil)

G429
m

Giacomin, Paulo André Sperandio, 1977-
Um método interativo de recuperação de diagramas grafcet e de estados a partir de diagramas ladder usando diagramas binários de decisão / Paulo André Sperandio Giacomin. – 2005.
152 f. : il.

Orientador: Hans Jorg Andreas Schneebeli.
Dissertação (mestrado) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Engenharia reversa. 2. Controladores programáveis. I. Schneebeli, Hans Jorg Andreas. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 621.3

Paulo André S. Giacomin

Um Método Interativo de Recuperação de Diagramas Grafcet e de Estados a partir de Diagramas Ladder usando Diagramas Binários de Decisão

Dissertação submetida ao programa de Pós-Graduação em Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisição parcial para a obtenção do Grau de Mestre em Engenharia Elétrica - Automação.

Aprovada em 04 de novembro de 2005

COMISSÃO EXAMINADORA

Professor Doutor Hans Jorg Andreas Schneebeli
Universidade Federal do Espírito Santo - UFES
Orientador

Professor Doutor José Denti Filho
Universidade Federal do Espírito Santo - UFES

Professor Doutor Jun Okamoto Junior
Universidade de São Paulo - USP

Professor Doutor Paulo Faria Santos Amaral
Universidade Federal do Espírito Santo - UFES

Aquilo que fui não sou mais, nem sou tudo o que devo ser. Já sou pela graça o que sou,
e ao vê-lo estou certo que vou, na glória de Cristo viver!

Agradecimentos

"porque o Poderoso me fez grandes coisas; e santo é o seu nome. E a sua misericórdia vai de geração em geração sobre os que o temem. Com o seu braço manifestou poder; dissipou os que eram soberbos nos pensamentos de seus corações; depôs dos tronos os poderosos, e elevou os humildes. Aos famintos encheu de bens, e vazios despediu os ricos. Auxiliou a Israel, seu servo, lembrando-se de misericórdia (como falou a nossos pais) para com Abraão e a sua descendência para sempre." Lucas 1:49-55.

Obrigado Poderoso por ter feito grandes coisas na minha vida. Obrigado meu pai, Paulo Dárcio Giacomin, porque enfrentou grandes dificuldades por minha causa e por toda a família, e em nenhum momento nos deixou desamparados, jamais negou uma ajuda sequer, por mais difícil que fosse, demonstrando fortíssimo amor de pai em todas as situações, sempre com muita sinceridade, honestidade, retidão e fidelidade a Deus. Obrigado minha mãe, Nair de Lourdes Sperandio Giacomin, porque me ajudou várias vezes na minha vida.

Agradeço a todos os membros da Igreja Cristã Maranata de Porto Canoa, aos diáconos, obreiros e jovens que são realmente uma família para mim.

Obrigado Doutora Janine, porque diante das dificuldades que passei, as pressões do mestrado e da conclusão da minha graduação, não deixou de demonstrar em nenhum momento grande vontade em me ajudar e me querer bem, preoculpando-se com os mínimos detalhes, com muita ética e profissionalismo, mostrando grande riquesa humana, conseqüência de sua educação e principalmente da fé e do amor que lhe foram dados pelo Senhor.

Obrigado amado Pastor Wilson, porque nunca recebi em minha vida uma assistência tão sincera e verdadeira, com tanta atenção, compreensão, sabedoria e amor, compreendendo os meus problemas, depositando em mim confiança, permitindo que eu realizasse a

obra do Senhor ao seu lado. Tem servido ao Senhor gratuitamente durante anos e anos, alcançando o coração de Deus, dos seus familiares, e de toda a igreja do Senhor.

Obrigado professor Hans, por ter compartilhado parte do seu conhecimento comigo, dando-me oportunidade de crescer, e confiando em meu trabalho e em minha capacidade. Fez todas as coisas buscando excelência na minha formação profissional. Sua garra e persistência em seu trabalho fazem de você um lutador, e mesmo diante do sucateamento do ensino público superior de nosso país, um herói para a sociedade capixaba.

Conteúdo

1 Introdução **25**

1.1 Motivação 26

1.2 Descrição do Funcionamento do Controlador 30

1.2.1 O Diagrama Ladder 31

1.2.2 O Diagrama Grafcet 31

1.2.3 O Diagrama de Blocos 32

1.2.4 Lista de Instruções 32

1.2.5 Texto Estruturado 33

1.2.6 Diagrama de Estados 33

1.2.7 Lógica Temporal 35

1.2.8 Redes de Petri 36

1.3 A Legibilidade dos Diagramas 36

1.4 Definição do Problema 45

1.5 Solução Proposta 46

1.6 Metodologia 47

1.7 Estrutura do Trabalho 48

2 O Método **49**

2.1 Circuitos de Chaveamento 49

2.2 Formalização do Diagrama Ladder 52

2.3 Análise do Diagrama Ladder 56

2.3.1 Malhas Seladas com Selo Direto 58

2.3.2 Malhas Seladas com Selo Indireto 61

2.3.3 Malhas Combinacionais 64

2.4 Formalização do Grafcet 66

2.5 Formalização do Diagrama de Estados 68

2.6 Casos de Mapeamento sem Paralelismo 69

2.7 Formalização do Método Proposto para Diagramas Grafcet 74

2.8 Formalização do Método Proposto para Diagramas de Estados 76

2.9 Casos de Mapeamento com Paralelismo 78

2.10 Formalização do Método Proposto para Detecção de Paralelismo 81

2.11 Conclusão 82

3 Manipulação de Expressões Booleanas **92**

3.1 As Manipulações Algébricas 92

3.2 A Expansão de Shannon 93

3.3 Diagramas Binários de Decisão 94

3.3.1 Representação 95

3.3.2 Operações 99

3.4 Árvores 101

3.4.1 Caminhamento em profundidade 105

3.5 Usando os Diagramas Binários de Decisão 106

3.6 Análise da eficiência 108

3.7 Conclusão 109

4 O Protótipo **110**

4.1 Especificação 110

4.2 Arquitetura 110

4.3 Detalhamento dos Módulos do Sistema 114

4.3.1 O módulo do diagrama ladder 114

4.3.2 O módulo de persistência 115

4.3.3 O módulo de interface do diagrama ladder 115

4.3.4 O módulo de interface com o usuário 117

4.3.5 O módulo de conversão 117

4.3.6 O módulo de gerenciamento 117

4.3.7 O módulo do diagrama grafcet 118

4.3.8 O módulo de interface do diagrama grafcet 118

4.4 Um Exemplo 118

5 Conclusão **129**

A Diagramas de Controle Sequencial **132**

A.1 O Diagrama Ladder 132

A.1.1 Componentes 132

A.1.2 Interpretação 133

A.2 O Diagrama Grafcet 133

A.2.1 Passos e Transições 135

A.2.2 Passagem pelas Transições 136

A.2.3 Regras de Passagem 137

A.2.4 Ações e Saídas 138

A.2.5 Concorrência e Sincronização 138

A.2.6 Interpretação 138

A.2.7 Interpretação 140

A.3 Máquina de Estados 141

B Outros Métodos de Recuperação de Diagramas Grafcet a partir de Diagramas Ladder 145

C O formato do arquivo de entrada 148

Lista de Tabelas

1.1 Classificação dos diagramas e linguagens 30

2.1 Cada linha da tabela é um cubo 56

2.2 Atuadores 79

3.1 Operações 99

Lista de Figuras

1.1 O Controlador e a planta 25

1.2 Exemplo de um diagrama *ladder* 27

1.3 Um sistema de corte de peças 30

1.4 Exemplo de um diagrama *ladder* 31

1.5 Exemplo de um diagrama grafcet 32

1.6 Exemplo de um diagrama de blocos 32

1.7 Exemplo de um programa que utiliza lista de instruções 33

1.8 Exemplo de um programa que utiliza texto estruturado 34

1.9 Exemplo de um diagrama de estados 34

1.10 Exemplo de um programa que utiliza lógica temporal 35

1.11 Exemplo de uma Rede de Petri 36

1.12 Sistema delta de transferência de peças 37

1.13 Diagrama *ladder* do sistema delta 39

1.14 Diagrama grafcet do sistema delta 40

1.15 Verificando a legibilidade do diagrama de blocos 40

1.16 Analisando a legibilidade da lista de instruções 41

1.17 Verificando a legibilidade do texto estruturado 42

1.18 Analisando a legibilidade do diagrama de estados 43

1.19 Analisando a legibilidade da lógica temporal 43

1.20 Analisando a legibilidade das redes de Petri 44

1.21 A planta como parte do sistema 45

2.1 Contatos normalmente abertos e fechados [1] 50

2.2 A função ***and*** implementada com relés [1] 50

2.3 A função ***or*** implementada com relés [1] 51

2.4 A função ***not*** implementada com relés [1] 51

2.5 Uma expressão booleana implementada com relés [1] 52

2.6 Contatos normalmente abertos e fechados [1] 53

2.7 Os elementos básicos do diagrama *ladder* são semelhantes aos dos relés . . 53

2.8 Realimentação direta 57

2.9 Malhas seladas com selo direto 58

2.10 Ilustração do processo de ativação e desativação de malhas seladas 60

2.11 Exemplo de uma malha não selada com realimentação 61

2.12 Modelo geral de uma malha selada com selo direto 61

2.13 Malhas seladas com selo indireto nível um 62

2.14 Selo indireto nível um 63

2.15 Uma função booleana nível zero 64

2.16 Uma função booleana nível um 64

2.17 Uma função booleana nível dois 65

2.18 Modelo geral de uma malha combinacional 65

2.19 Malhas Combinacionais 66

2.20 Um diagrama grafcet 67

2.21 Diagrama de estados 68

2.22 Portão 70

2.23 Diagrama *ladder* 70

2.24 Diagrama de estados 83

2.25 Diagrama grafcet do portão 83

2.26 Sistema $\boldsymbol{\alpha}$ de transferência de peças 84

2.27 Diagrama *ladder* do sistema $\boldsymbol{\alpha}$ 85

2.28 Estado intermediário 86

2.29 Diagrama de estados do sistema $\boldsymbol{\alpha}$ 86

2.30 Diagrama grafcet do sistema $\boldsymbol{\alpha}$ 87

2.31 Sistema $\boldsymbol{\beta}$ de transferência de peças 87

2.32 Diagrama *ladder* dos motores do sistema $\boldsymbol{\beta}$ 88

2.33 Diagrama *ladder* dos pistões do sistema $\boldsymbol{\beta}$ 89

2.34 Diagrama *ladder* da garra do sistema $\boldsymbol{\beta}$ 90

2.35 Diagrama grafcet do sistema $\boldsymbol{\beta}$ 91

3.1 Exemplo de um BDD 95

3.2 Simplificando um BDD [16, 20] 98

3.3 Uma malha de um diagrama *ladder* 101

3.4 Uma pequena árvore gramatical 102

3.5 As três primeiras malhas da figura 2.32 103

3.6 Árvores 104

3.7 A árvore e seus BDDs 106

4.1 Principais módulos do sistema 111

4.2 Módulo de entrada 112

4.3 Módulo de conversão 113

4.4 Módulo de visualização 113

4.5 Módulos do Sistema 115

4.6 Diagrama de classes do módulo do diagrama *ladder* 116

4.7 O Módulo de Conversão 117

4.8 O módulo do diagrama grafcet 119

4.9 Arquivo do diagrama *ladder* no formato XML 120

4.10 Interface do diagrama *ladder* 120

4.11 Confirmação de estado 121

4.12 Estados iniciais 122

4.13 Confirmação de sequenciamento 123

4.14 Nem todas as transições são confirmadas 124

4.15 Mais do que uma transição 125

4.16 Confirmando a ação condicional 126

4.17 Diagrama de estados resultante 127

4.18 Diagrama grafcet gerado pelo protótipo 128

A.1 Uma malha de um diagrama *ladder* 133

A.2 Arquitetura de um PLC 134

A.3 Ciclo 134

A.4 Passos 135

A.5 Arcos direcionados e transições 136

A.6 Sincronização e concorrência 139

A.7 Máquina de estados 141

A.8 Super estado 142

A.9 Junções e disjunções 143

A.10 Paralelismo 144

A.11 Ações 144

C.1 O arquivo de entrada 148

C.2 A árvore DOM 149

C.3 A árvore DOM de uma malha do diagrama *ladder* 150

Simbologia

u_3	Contato normalmente aberto de um diagrama ladder		Arco **or** de um diagrama grafcet
q_2	Contato normalmente fechado de um diagrma ladder		Arco **and** de um diagrama grafcet
q_1	Bobina de um diagrama ladder	OR	Bloco de um diagrama de blocos
	Pistão pneumático		Estado inicial de um diagrama de estados
	Pistão pneumático com plataforma	Abaixando	Estado de um diagrama de estados
4	Passo inicial de um diagrama grafcet		Arco utilizado no diagrama de estados, no diagrama grafcet e nas redes de Petri
4	Passo de um diagrama grafcet	p_0	Lugar (place) de uma rede de Petri
2 Q_2	Passo com ação	p_0	Lugar com um token
u_3	Transição		Esteira rolante
	Contato normalmente fechado de um relé		Contato normalmente aberto de um relé
	Bobina de um relé		Fonte de tensão de corrente contínua
	Relé		Lâmpada
B^1_i	Representação gráfica de uma sub-função em uma malha de um diagrama ladder, também chamada de bloco		Sensor
1	Nó não-terminal de um BDD		Classe
1	Nó terminal de um BDD		Pacote
AND	Nó de uma árvore sintática	módulo	Módulo do protótipo

Notação Matemática

Notação	Descrição
$\boldsymbol{f}$	Função booleana
$\bigcirc \boldsymbol{f}$	$\bigcirc \boldsymbol{f}$ é verdadeiro se e somente se $\boldsymbol{f}$ é verdadeira no próximo estado da seqüencia
$\Box \boldsymbol{f}$	$\Box \boldsymbol{f}$ é verdadeiro se e somente se $\boldsymbol{f}$ é verdadeira em todos os futuros estados da seqüencia
$\Diamond \boldsymbol{f}$	$\Diamond \boldsymbol{f}$ é verdadeiro se e somente se $\boldsymbol{f}$ é verdadeira em algum estado futuro
$\boldsymbol{f_1 U f_2}$	$\boldsymbol{f_1 U f_2}$ é verdadeiro se e somente se $\boldsymbol{f_1}$ é verdadeira em todos os estados até o primeiro estado onde $\boldsymbol{f_2}$ é verdadeira
$\boldsymbol{T}$	True ou verdadeiro
$\boldsymbol{F}$	False ou falso
$\wedge$	E lógico
$\vee$	Ou lógico
$\neg$	Complemento de um valor booleano
$\mathbf{0}$	Zero lógico, equivalente ao valor F
$\mathbf{1}$	Um lógico, equivalente ao valor T
$\overline{\boldsymbol{x}}$	Complemento de $\boldsymbol{x}$
$\cdot$	Multiplicação lógica, equivalente ao E lógico
$+$	Soma lógica, equivalente ao Ou lógico

Notação	Descrição
$y = f(x_1, x_2, \cdots, x_n)$	Função lógica de n variáveis
$f \mid_{x_i=a}$	Restrição de uma função em relação à uma variável, ou seja, $f(x_1, \cdots, a, \cdots, x_n)$
$E = \{e_1, e_2, \cdots, e_n\}$	Conjunto E de n elementos
$\emptyset$	Conjunto vazio
$B : 0,1^n \longmapsto 0,1$	Função que mapeia o espaço booleano de dimensão n no espaço booleano de dimensão 1
$\overrightarrow{a_i}$	Vetor ativador da função booleana i
$\overrightarrow{d_i}$	Vetor desativador da função booleana i
f_i^j	Sub-função j da função booleana que representa a malha i
$f \sqsubseteq g$	A função f é sub-função da função g
$a = (p, t)$	Um arco a que vai do passo p até a transição t
$a = (t, p)$	Um arco a que vai da transição t até o passo p
$p_x \rightarrow p_y$	Os passos p_x e p_y são seqüenciais no sentido indicado por esta flecha
$\delta(a_{cj}) = x$	Função que mapeia uma ação condicional à sua condição
$\circledcirc$	Estado inicial
W_i	Passo ou estado de espera(Wait)
C'	Novo conjunto criado a partir do conjunto C qualquer porém com algumas atualizações
$i \parallel j$	Se e somente se i e j são passos paralelos
$i \shortparallel j$	Se e somente se dois passos i e j poderão tornar-se ativos no mesmo instante de tempo
$i \mid j$	Se e somente se dois passos i e j podem permanecer selados simultaneamente
$\oplus$	Conjunto de malhas intertravadas
Φ	Estados analisados no caso do diagrama de estados e passos analisados no caso do diagrama *grafcet*

Notação	**Descrição**
φ	Novos estados no caso do diagrama de estados e novos passos no caso do diagrama *grafcet*
$\boldsymbol{S_f}$	Conjunto de todos os cubos de um BDD
$\lvert\boldsymbol{S_f}\rvert$	Quantidade de cubos de um BDD

Abreviaturas

Abreviatura	Descrição
PLC	Controlador Lógico Programável
IL	Lista de Instruções
ST	Texto Estruturado
FBD	Diagrama de Blocos de Funções
SFC	Gráfico de Funções sequenciais
PLSL	Linguagem de Especificação de Lógica Programável
XML	Linguagem de Marcação Extensível
BDD	Diagrama Binário de Decisão
DOM	Modelo de Objetos de Documento

Resumo

Apresentamos neste trabalho um novo método de recuperação de diagramas *grafcet* e de estados a partir de diagramas *ladder*. Um método interativo é proposto. Nenhuma modelagem preliminar da planta é necessária, todas as possibilidades de sequenciamento são detectadas e não é necessário descrever toda a planta. O diagrama *ladder* é representado através de um conjunto de BDDs, com os quais descobrimos as expressões de ativação e desativação de cada malha do diagrama *ladder*. A seguir é feita uma análise do diagrama ladder em busca de possíveis eventos de transição. Uma interface gráfica é utilizada para que o especialista confirme os eventos de transição que ocorrem na planta. No final, o diagrama de estados e o diagrama grafcet equivalentes ao diagrama ladder são apresentados ao usuário através de uma interface gráfica.

Abstract

We show a new approach for the recovery of grafcet or state diagrams from ladder diagrams. It is based on the use of Binary Decision Diagrams (BDDs). Analysing the activation and deactivation expressions for each coil and using the knowledge of an expert it is possible, interactively, to build a diagram with the relevant transitions. A graphical interface is used to interact with the expert reducing the burdening and easing the task. At the end, we have a graphical representation of the grafcet or state diagram.

Capítulo 1

Introdução

Hoje em dia é comum o uso de Controladores Lógico Programáveis (Programmable Logic Controller - PLC) na automação das mais variadas plantas industriais, sejam elas químicas, metalúrgicas, mineradoras, refinarias, entre outras, para soluções de problemas de controle seqüencial, caracterizados principalmente por sua natureza digital.

Como podemos ver na figura 1.1, temos em um sistema de controle seqüencial dois sub-sistemas bem definidos: o controlador e a planta. A planta é o sistema que estamos querendo controlar. Dela o controlador recebe informações como valores digitais de variáveis medidas, tais como sinais de sensores de fim de curso, sensores de presença, chaves liga-desliga, e processa essas informações, toma decisões e responde apropriadamente modificando ou não a planta com atuadores adequados. O controlador, por sua vez, poderá também receber informações que não venham diretamente da planta, podendo ter outra origem, como um sistema supervisório ou o operador, por exemplo. Apesar de termos papéis bem definidos entre a planta e o controlador, ambos fazem parte de um mesmo sistema que é formado por dois sub-sistemas reativos que se comunicam. Quase sempre não poderemos compreender um sem que tenhamos que estudar o outro.

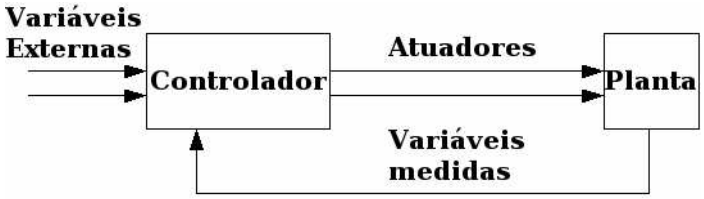


Figura 1.1: O Controlador e a planta

O controlador pode ser implementado com várias ferramentas diferentes, como relés ou contatores, circuitos microcontrolados construídos sob-medida, software executado em um computador que faça uso de um sistema operacional em tempo real ou controladores lógico programáveis ou PLCs. Assim, podemos dizer que os PLCs são uma ferramenta de implementação de controladores digitais, e são, no momento, os mais utilizados.

Existem diversas vantagens em se utilizar um PLC ao invés de utilizarmos outras técnicas, tal como o painel de relés. O PLC é muito mais confiável, estável, consome menos energia, ocupa menos espaço e facilita muito a manutenção da lógica do sistema, uma vez que a mesma é implementada por software, ficando assim separada das ligações elétricas, ao contrário do que acontecia com o painel de relés. Com isto, a redução do custo também acaba sendo uma vantagem na utilização do PLC.

1.1 Motivação

Há alguns anos atrás os profissionais da área de automação programavam os PLCs quase que exclusivamente usando diagramas *ladder*, cuja criação foi baseada no antigo diagrama de contatos. O diagrama *ladder* foi a ferramenta mais utilizada na programação de PLCs porque não haviam muitas alternativas existentes na época, uma vez que esta tecnologia estava apenas começando e o estado da arte ainda era limitado.

Podemos ver um exemplo deste tipo de diagrama na figura 1.2. Nela, o símbolo da primeira malha com endereço $\boldsymbol{x}$ é um exemplo de um contato normalmente aberto, o qual sendo ativado, fecha. O símbolo da primeira malha com endereço $\boldsymbol{sv+}$ é um exemplo de um contato normalmente fechado, o qual sendo ativado, abre. Os círculos presentes em todas as malhas são as bobinas do diagrama *ladder*, equivalentes às antigas bobinas dos relés.

O diagrama *ladder* é basicamente uma adaptação do antigo diagrama de contatos usados na indústria antes do aparecimento dos PLCs. Esta adaptação foi feita devido à abundância de mão-de-obra especializada que existia na época. Uma vez que mão-de-obra especializada era algo escasso naquela época, fizeram o diagrama *ladder* similar a um diagrama de contatos, com o objetivo de aproveitar a mão-de-obra existente. Também existiram outros motivos desta similaridade, pois como dissemos anteriormente, o estado da arte em implementação de controladores digitais naquela época era limitado, o que incentivou uma abordagem similar à que já existia na automação de sistemas de controle seqüencial, que era a utilização de painéis de relés.

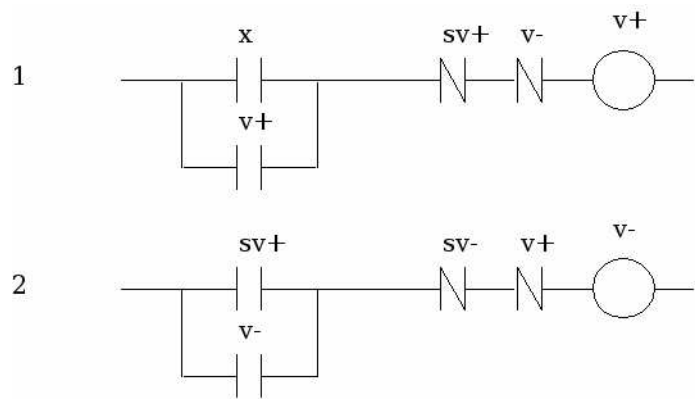


Figura 1.2: Exemplo de um diagrama *ladder*

Entretanto, o diagrama *ladder* não é a melhor linguagem para especificarmos a lógica de um sistema de controle seqüencial. Um dos principais problemas deste diagrama é a legibilidade. Compreender um diagrama ladder de uma planta complexa é algo extremamente difícil, até mesmo para os profissionais especializados. Assim, o diagrama *ladder* era apenas a ferramenta mais apropriada se levassemos em consideração a mão-de-obra existente naquele momento, bem como as limitações impostas pelo estado da arte daquele momento.

No ano de 2003, a norma IEC 6113-1 [26] padronizou 5 linguagens a serem utilizadas na programação de PLCs: o Diagrama *Ladder* (LD), Gráfico de Funções Seqüenciais (SFC - também conhecido como *grafcet*), Texto Estruturado (ST), Diagrama de Blocos de Funções (FBD) e a Lista de Instruções (IL). Além do alcance desta norma, podemos citar outras ferramentas para modelagem de sistemas de controle seqüencial, como o diagrama de estados, as redes de Petri e a linguagem PLSL.

Como os PLCs já existem há algum tempo no mercado e a norma IEC 6113-1 é recente, então os programas de PLCs estão, em sua maioria, escritos através de diagramas *ladder*.

Muitas das plantas existentes na indústria são enormes, e consequentemente a lógica dos seus respectivos controladores. Assim, construir um novo controlador dessas plantas a partir do início usando uma nova linguagem seria um grande desperdício do trabalho que já foi feito, além de novos gastos com mão-de-obra especializada. Vários programas que estão funcionando atualmente já foram exaustivamente testados e estão funcionando há anos. Assim, nós poderíamos optar por refazer toda a programação dos PLCs usando *grafcet* ou então fazer a engenharia reversa dos sistemas já existentes. Caso optássemos por refazer todo o sistema, levaria algum tempo para o programa ser testado, ou seja,

nós perderíamos a confiabilidade que já está implementada nos antigos diagramas *ladder*. Caso seja feita a engenharia reversa, esta confiabilidade poderia em parte ser aproveitada. Como confiabilidade é algo extremamente importante em programas de PLCs de plantas industriais, podemos dizer que a engenharia reversa é atraente. Outro fator motivador da engenharia reversa é a redução de custos que ela pode trazer, uma vez que com ela não é necessário refazer todo o sistema. Pressman [4] faz em seu trabalho uma reflexão da necessidade da engenharia reversa em sistemas antigos e fala da necessidade da automatização do processo de engenharia reversa.

Vários trabalhos na literatura levantaram limitações do uso do diagrama *ladder* na programação de PLCs, tais como os de Zhou e Twiss [32] e Venkatesh et al [13]. Outros trabalhos [2, 10, 11, 12, 29, 30, 31, 33, 34, 35, 36, 37, 38, 39, 40] confirmaram as deficiências do diagrama *ladder*. Dentre algumas destas limitações, destacam-se:

- *complexidade de projeto.* Venkatesh et al [13] apontaram duas características do diagrama *ladder* que influenciam a complexidade de projeto de um programa: a complexidade gráfica e a dificuldade de adaptação à novas especificações. A complexidade gráfica de um diagrama *ladder* aumenta consideravelmente com o aumento do sistema. Além disso, modificações na especificação provocam grandes modificações no diagrama *ladder*;

- *tempo de resposta.* Venkatesh et al [13] fizeram um estudo do tempo de resposta de um diagrama *ladder* e relacionaram-no com a complexidade gráfica do sistema. Quanto maior a complexidade gráfica, maior o tempo de resposta. Assim, a medida que o sistema cresce, ele vai ficando acentuadamente mais lento.

- *simbologia precária.* Venkatesh et al [13] constataram que o diagrama *ladder* não possui simbologia para representar condições, status, atividades, informação, fluxo e recursos de forma explícita;

- *ilegibilidade.* O problema fica ainda mais sério quando a complexidade dos sistemas aumenta, seja por causa do tamanho do sistema ou devido a outros fatores técnicos como problemas de concorrência, etc.;

- *manutenibilidade.* Para sistemas mais complexos, dar manutenção em um programa escrito usando-se o diagrama *ladder* é algo extremamente difícil, até mesmo para os profissionais mais bem preparados;

- *custo.* com o aumento da complexidade dos diagramas *ladder*, o tempo gasto em manutenção aumenta, e consequentemente aumentam também os gastos;

Tipo	Descrição	Exemplos
Linguagens textuais	Utilizam texto para especificar a lógica do controlador	Lista de instruções, texto estruturado e a linguagem PLSL
Diagramas gráficos	Utilizam símbolos gráficos para especificar a lógica do controlador	Diagrama ladder, diagrama grafcet, diagrama de blocos, redes de Petri, diagrama de estados.

Tabela 1.1: Classificação dos diagramas e linguagens

- *falta de informação.* A não ser que a planta seja muito simples, um diagrama *ladder* não possui informações suficientes para que um programador possa extrair o fluxo de controle do processo da planta. Em outras palavras, não é possível prever totalmente o funcionamento de uma planta apenas analisando um diagrama *ladder* relativamente complexo. Nestes casos, precisamos de informação da planta para compreendermos o seu funcionamento.

O mesmo não acontece com o *grafcet* e com o diagrama de estados, ou seja, se implementarmos um sistema utilizando diagrama *grafcet*, o mesmo poderá ser muito mais facilmente compreendido do que se utilizarmos o diagrama *ladder* para implementarmos o mesmo sistema.

Uma vez compreendida a necessidade de fazermos a engenharia reversa dos sistemas já implementados, falaremos agora sobre as linguagens utilizadas na implementação de sistemas de controle seqüencial. Afinal, esta é a natureza dos sistemas que estamos tratando neste trabalho: o controle seqüencial.

1.2 Descrição do Funcionamento do Controlador

Podemos classificar os diagramas e linguagens descritos nesta sessão em diagramas gráficos e linguagens textuais, como mostra a figura 1.1.

Para exemplificar cada um destes diagramas e linguagens, vamos utilizar um exemplo de uma planta simples e a seguir descrevê-la usando cada uma destas ferramentas. Podemos vê-la na figura 1.3. Nela, a esteira A permanece em movimento até que seja

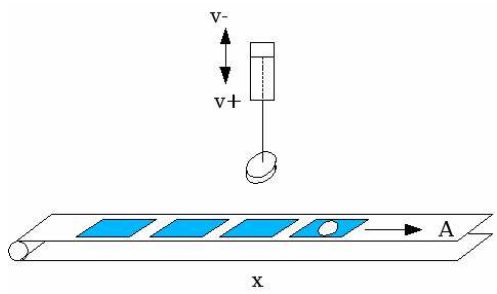


Figura 1.3: Um sistema de corte de peças

detectada a presença de uma folha de aço na posição $\boldsymbol{x}$, quando então a esteira $\boldsymbol{A}$ faz uma pausa de **2** segundos. A seguir o pistão pressiona o molde sobre a folha de aço, cortando-a. Terminado o corte, o pistão volta a subir. É importante dizer que quando o pistão retorna a posição inicial, não encontraremos mais a folha de aço na posição $\boldsymbol{x}$. Para efeito de simplificação, vamos considerar que nosso controlador controla apenas o pistão e que a esteira é controlada por outro controlador.

Uma vez apresentada a planta da figura 1.3, podemos agora utilizar diversas linguagens para implementarmos o controlador dela. Vamos começar pelo diagrama *ladder*.

1.2.1 O Diagrama Ladder

Podemos ver um exemplo deste tipo de diagrama na figura 1.4. Nela, o símbolo da primeira malha com endereço $\boldsymbol{x}$ é um exemplo de um contato normalmente aberto, o qual sendo ativado, fecha. O símbolo da primeira malha com endereço $\boldsymbol{sv+}$ é um exemplo de um contato normalmente fechado, o qual sendo ativado, abre. Os círculos presentes em todas as malhas são as bobinas do diagrama *ladder*.

1.2.2 O Diagrama Grafcet

A figura 1.5 mostra um exemplo de um diagrama *grafcet*. O quadrado com identificação **2** é um exemplo de um passo. As barras horizontais em negrito são as transições, como, por exemplo, $\boldsymbol{x}$. O passo com identificador **1** é um exemplo de um passo inicial. O fluxo de controle inicia-se no passo inicial e, após as condições das transições tornarem-se verdadeiras, os passos anterior e posterior são imediatamente desativado e ativado,

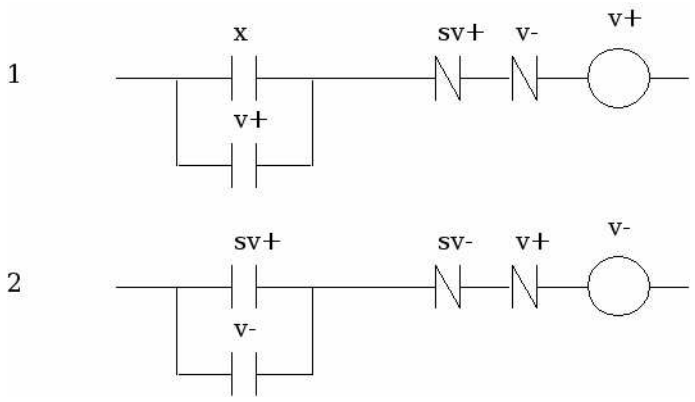


Figura 1.4: Exemplo de um diagrama *ladder*

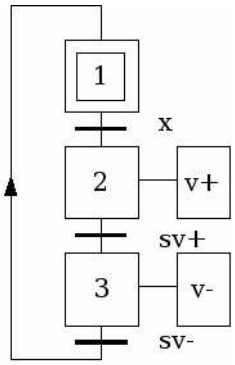


Figura 1.5: Exemplo de um diagrama grafcet

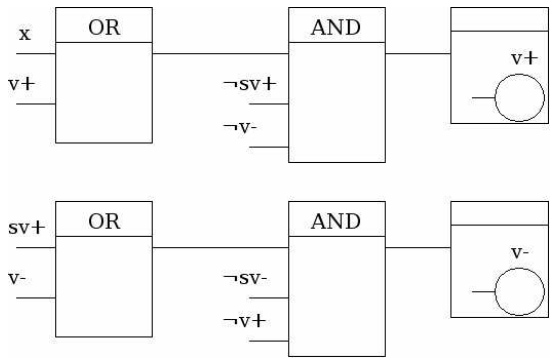


Figura 1.6: Exemplo de um diagrama de blocos

respectivamente. Os passos podem possuir ações. Por exemplo, o passo **2** estando ativo acionará a ação de endereço $\boldsymbol{v+}$, como pode ser visto na figura.

1.2.3 O Diagrama de Blocos

A figura 1.6, mostra um exemplo de um diagrama de blocos. Os retângulos com identificação ***and*** e ***or*** são exemplos de blocos. Nos três primeiros blocos, se $\boldsymbol{x}$ ou $\boldsymbol{v+}$ estiverem ativos, então a saída do bloco ***or*** será ativada e servirá de entrada para o bloco ***and***. Este somente ativará sua saída se os endereços $\boldsymbol{sv+}$ e $\boldsymbol{v-}$ estiverem desativados e se a saída do bloco ***or*** estiver ativa. Uma vez ativada a saída do bloco ***and***, o endereço $\boldsymbol{v+}$ tornar-se-á ativo.

1.2.4 Lista de Instruções

A figura 1.7 mostra um exemplo de um programa que utiliza lista de instruções. Na primeira listagem, o valor do bit endereçado por $\boldsymbol{x}$ é carregado para um registrador, o valor do bit endereçado por $\boldsymbol{v+}$ é armazenado em outro registrador e a seguir é feito um ***or*** entre estes dois valores. Seguindo, o valor negado do bit endereçado por $\boldsymbol{v-}$ é armazenado em um registrador e feito um ***and*** com o resultado do ***or*** anterior. O mesmo é feito com o bit endereçado por $\boldsymbol{sv+}$. No final, o resultado é armazenado no endereço $\boldsymbol{v+}$.

```
LD x
LD v+
AND
LDN v-
AND
LDN sv+
AND
ST v+
```

(a) v+

```
LD x
LD v+
OR
LDN v-
AND
LDN sv+
AND
ST v+
```

(b) v-

Figura 1.7: Exemplo de um programa que utiliza lista de instruções

```
VAR
    presenca: BOOL AT %Ix
    abaixar: BOOL AT %Qv+ := 0
    fim_abaixar: BOOL AT %Isv+
    levantar: BOOL AT %Qv- := 0
    fim_levantar: BOOL AT %Isv-
END_VAR

IF ((presenca OR abaixar) AND (NOT levantar) AND (NOT fim_abaixar)) THEN
    abaixar := TRUE;
ELSE
    abaixar := FALSE;
END_IF

IF ((fim_abaixar OR levantar) AND (NOT fim_levantar) AND (NOT abaixar)) THEN
    levantar := TRUE;
ELSE
    levantar := FALSE;
END_IF
```

Figura 1.8: Exemplo de um programa que utiliza texto estruturado

1.2.5 Texto Estruturado

A figura 1.8 implementa o mesmo controlador dos exemplos anteriores, porém usando texto estruturado. No primeiro bloco são declaradas as variáveis. Para cada uma delas, começamos com o nome da variável seguido do tipo da variável e logo após o endereço de entrada ou saída o qual a ela está relacionado. Terminada a declaração das variáveis entramos em uma cadeia de IFs que implementa uma lógica equivalente à da figura 1.2. Nesta linguagem, toda vez que o fluxo de controle chega ao final do arquivo ele retorna para o início do mesmo. Por isso não há a necessidade de implementarmos um loop principal.

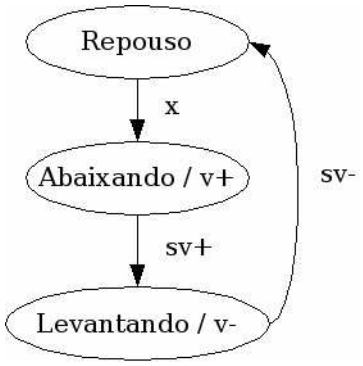


Figura 1.9: Exemplo de um diagrama de estados

1.2.6 Diagrama de Estados

Existem também outros diagramas que não fazem parte da norma IEC 61131 mas que também são amplamente conhecidos, como é o caso do diagrama de estados e da lógica temporal. A figura 1.9 mostra um exemplo de um diagrama de estados. Nela, as elipses representam os estados, os arcos que conectam os estados representam as transições, as quais possuem condições.

Neste exemplo, consideramos que o pistão encontra-se inicialmente em repouso. Após a ativação do sensor de presença $\boldsymbol{x}$ o pistão começa a abaixar o molde através do atuador $\boldsymbol{v+}$. Uma vez que $\boldsymbol{sv+}$ é acionado, o pistão para de abaixar e começa a levantar através do atuador $\boldsymbol{v-}$, até que o sensor $\boldsymbol{sv-}$ seja acionado, quando então o pistão voltará para o estado de repouso.

1.2.7 Lógica Temporal

Existem diversas implementações de lógica temporal na literatura. A figura 1.10 apresenta um exemplo de um programa escrito onde usamos a lógica temporal extendida por Choo et al [43]. Ela é chamada de Programmable Logic Specification Language - PLSL. Nela, alguns operadores lógicos são $\Box(\boldsymbol{sempre})$, $\Diamond(\boldsymbol{algumas\ vezes})$, $\bigcirc$ *(próximo)* e $\boldsymbol{U}(\textit{até})$, sendo que o último é um operador binário e os outros são unários. Formalmente, estes operadores podem ser definidos da seguinte forma:

- $\boldsymbol{f}$ é verdadeira se e somente se $\boldsymbol{f}$ é verdadeira no presente momento;
- $\bigcirc\boldsymbol{f}$ é verdadeiro se e somente se $\boldsymbol{f}$ é verdadeira no próximo estado da seqüência;

- $\Box \boldsymbol{f}$ é verdadeiro se e somente se $\boldsymbol{f}$ é verdadeira em todos os futuros estados da seqüencia;
- $\Diamond \boldsymbol{f}$ é verdadeiro se e somente se $\boldsymbol{f}$ é verdadeira em algum estado futuro;
- $\boldsymbol{f_1 U f_2}$ é verdadeiro se e somente se $\boldsymbol{f_1}$ é verdadeira em todos os estados até o primeiro estado onde $\boldsymbol{f_2}$ é verdadeira.

Segundo Choo et al [43], a sintaxe da linguagem PLSL pode ser definida da seguinte forma:

$$(\boldsymbol{estado\ presente\ U\ \ condições})\ ]\ \boldsymbol{próximo\ estado}$$

Um exemplo do padrão acima é a terceira linha da figura 1.10, onde $\boldsymbol{v-}$ está representando o estado atual (levantando), *repouso* está representando o próximo estado e o termo intermediário é a condição de transição.

```
(repouso U ◊((x or v+) and (not v-) and (not sv+) )) ] O v+
v+ U ◊(v- or sv+) ] O v-
v- U ◊(sv- or v+) ] O repouso
```

Figura 1.10: Exemplo de um programa que utiliza lógica temporal

1.2.8 Redes de Petri

A figura 1.11 mostra um exemplo de uma Rede de Petri. Nela as barras são transições, os circulos representam lugares *(places)*. O ponto negro dentro de um lugar, chamado de token, indica que o lugar está ativo. Quando a condição de uma transição for verdadeira, o token passa imediatamente do lugar anterior para o próximo lugar conectado pelo arco.

O token encontra-se inicialmente sobre o passo $\boldsymbol{p_1}$, significando que o pistão está em repouso, até que o sensor $\boldsymbol{x}$ seja acionado, quando então o pistão começará a abaixar. Isto acontecerá até que o sensor de fim de curso $\boldsymbol{sv+}$ seja acionado, quando então o pistão irá parar de abaixar e começará a levantar, até que o sensor $\boldsymbol{sv-}$ seja acionado, quando então o pistão voltará para a posição de repouso. Uma formalização das Redes de Petri pode ser encontrada no trabalho de Moraes e Castrucci [44].

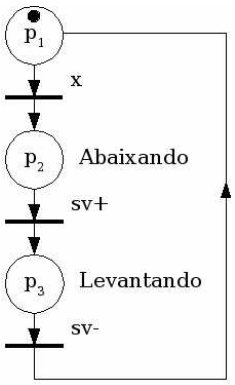


Figura 1.11: Exemplo de uma Rede de Petri

1.3 A Legibilidade dos Diagramas

A fim de que possamos analisar a legibilidade dos diagramas e linguagens utilizados na especificação de controladores de sistemas sequenciais, definimos o conceito de *fluxo de controle do processo da planta*, como sendo a sequencia de possíveis estados que um sistema pode assumir. Dependendo da capacidade ou incapacidade de um diagrama ou linguagem de especificação de controle sequencial de representar explicitamente o fluxo de controle do processo da planta, criamos duas classes de diferentes níveis de legibilidade:

- *Classe A:* Os diagramas e linguagens que se enquadram nesta classe de legibilidade não representam explicitamente o fluxo de controle do processo da planta. Com isto, quando estamos tentando compreender os diagramas e linguagens de especificação de controladores sequenciais que se enquadram dentro desta classe de legibilidade, ficamos fortemente dependentes das informações da planta para compreendermos estes diagramas e linguagens;

- *classe B:* Os diagramas e linguagens que se enquadram nesta classe de legibilidade representam explicitamente o fluxo de controle do processo da planta. Com isto, quando estamos tentando compreender os diagramas e linguagens de especificação de controladores sequenciais que se enquadram dentro desta classe de legibilidade, verificamos que ficamos muito menos dependentes das informações da planta para compreendermos estes diagramas e linguagens do que se estivéssemos utilizando um diagrama ou linguagem com grau de legibilidade *classe A*.

Vamos agora classificar os diagramas e linguagens estudados até aqui. Para isto utilizaremos o exemplo a seguir.

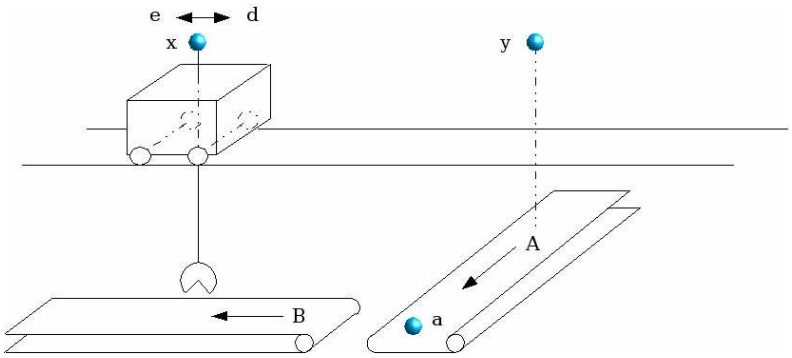
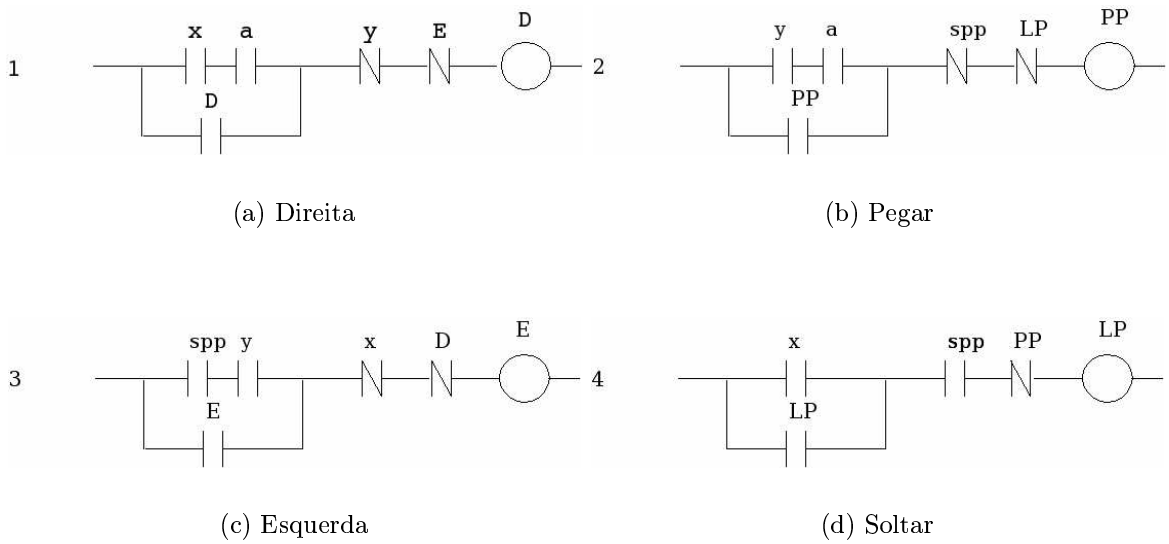


Figura 1.12: Sistema delta de transferência de peças

EXEMPLO: O sistema da figura 1.12 possui uma esteira de chegada de peças, A, além de um carro sobre trilhos com uma garra de pega G e uma esteira de evacuação B. O sistema possui vários sensores de presença, tais como $\boldsymbol{x}$ e $\boldsymbol{y}$, que detectam a posição do carro, e o sensor $\boldsymbol{a}$, que detecta a presença de uma peça sobre a esteira A. O funcionamento consiste em, detectada a presença de uma peça sobre a esteira de chegada, transportar a peça para a esteira de evacuação. Os motores D e E fazem o carro deslocar-se para a direita e para a esquerda, respectivamente. Os atuadores PP e LP fazem a garra pegar e soltar uma peça, respectivamente. O sensor *spp* diz se há uma peça presa pela garra. A figura 1.13 apresenta o diagrama *ladder* deste sistema.

A figura 1.13 apresenta um exemplo de um diagrama *ladder*. Como o leitor pode ver, a malha **1** pode ser desativada através de $\boldsymbol{y}$, e as malhas **2** e **3** podem ser ativadas através de $\boldsymbol{y}$. No entanto, devido a planta que estamos utilizando, a única transição que irá ocorrer será da malha **1** para a malha **2**, mas a transição da malha **1** para a malha **3** nunca irá acontecer. *Não há informação suficiente no diagrama ladder para chegarmos a esta conclusão.* Isto acontece porque o diagrama *ladder* não representa o fluxo de controle do processo da planta explicitamente. Assim, dizemos que o diagrama *ladder* possui um grau de legibilidade *classe A*.

A figura 1.14 apresenta o diagrama *grafcet*, usando a mesma planta do diagrama *ladder* anterior. Como podemos ver, não existe dúvidas que, se o estado atual do sistema em uma determinada situação for o passo **1**, então o próximo passo será o passo **2**. Assim, não precisamos verificar se o passo **3** é seqüencial ao passo **1**, ao contrário do diagrama *ladder*. Isto acontece porque este diagrama representa explicitamente o fluxo de controle do processo da planta através dos arcos direcionados. Assim, dizemos que o diagrama *grafcet* possui um grau de legibilidade *classe B*.



(a) Direita

(b) Pegar

(c) Esquerda

(d) Soltar

Figura 1.13: Diagrama *ladder* do sistema delta

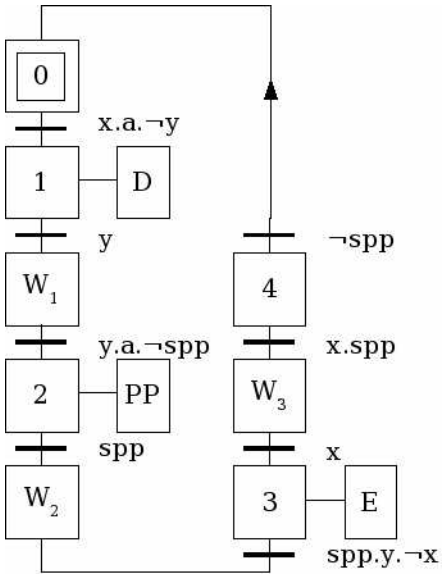


Figura 1.14: Diagrama grafcet do sistema delta

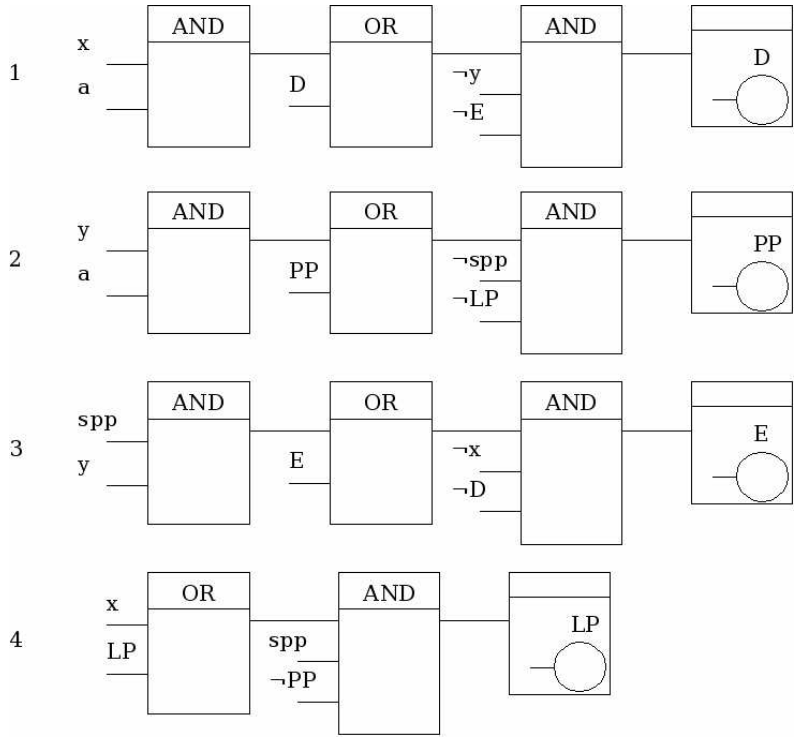


Figura 1.15: Verificando a legibilidade do diagrama de blocos

```
1          2          3          4

LD x       LD y       LD spp     LD x
LD a       LD a       LD y       LD LP
AND        AND        AND        OR
LD D       LD PP      LD E       LD spp
OR         OR         OR         AND
LDN y      LDN spp    LDN x      LDN PP
AND        AND        AND        AND
LDN E      LDN LP     LDN D      ST LP
AND        AND        AND
ST D       ST PP      ST E
```

Figura 1.16: Analisando a legibilidade da lista de instruções

A figura 1.15 apresenta o diagrama de blocos equivalente ao diagrama *ladder* que estamos utilizando nesta sessão. Como podemos ver, o bloco da malha **1** pode ser desativado quando $\boldsymbol{x}$ for acionado. Neste exato momento duas outras malhas deste diagrama de blocos poderão ser ativadas, ou seja, as malhas **2** e **3**. Assim, caímos no mesmo problema do diagrama *ladder*, ou seja, não temos como saber que a malha **1** é seqüencial apenas à malha **2** e não à malha **3**, olhando apenas para o diagrama de blocos. Isto acontece porque o diagrama de blocos não representa explicitamente o fluxo de controle do processo da planta. Isto classifica a sua legibilidade como de *classe A*.

A figura 1.16 a lista de instruções equivalente ao diagrama *ladder* utilizado nesta sessão. Como podemos ver, se $\boldsymbol{y}$ for acionado, então um valor igual a *falso* será armazenado na variável $\boldsymbol{D}$. Isto significa que este evento desativará o endereço $\boldsymbol{D}$. Observando as sublistas de instruções seguintes, vemos que $\boldsymbol{y}$ poderá acionar tanto o endereço $\boldsymbol{PP}$, quanto o endereço $\boldsymbol{E}$. Mas uma vez que não temos como saber, olhando apenas para a lista de instruções, que apenas o endereço $\boldsymbol{PP}$ será ativado após a desativação do endereço $\boldsymbol{D}$. Isto acontece porque a lista de instruções não representa explicitamente o fluxo de controle do processo da planta. Assim, classificamos a legibilidade da lista de instruções como sendo *classe A*.

A figura 1.17 apresenta texto estruturado equivalente ao diagrama *ladder* do exemplo desta sessão. Como podemos ver, se a variável $\boldsymbol{sobreEsteiraA}$ tornar-se verdadeira, então a variável *direita* será desativada. Neste mesmo instante, a variável $\boldsymbol{sobreEsteiraA}$ poderá ativar **2** outras variáveis, ou seja, *pegar* e *esquerda*. Não temos como saber, olhando apenas para o texto estruturado, que apenas a variável *soltar* será ativada. Isto acontece porque o texto estruturado não representa explicitamente o fluxo de controle do processo

```
VAR
     posicaoInicial: BOOL AT %Ix
     esteiraA: BOOL AT %Ia
     direita: BOOL AT %OD
     sobreEsteiraA: BOOL AT %Iy
     esquerda: BOOL AT %OE
     pegar: BOOL AT %OPP
     pegou: BOOL AT %Ispp
     soltar: BOOL AT %OLP
END_VAR

IF (((posicaoInicial AND esteiraA) OR direita) AND (¬sobreEsteiraA) AND (¬esquerda)) THEN
     direita := TRUE;
ELSE
     direita := FALSE;
END_IF

IF (((sobreEsteiraA AND esteiraA) OR (pegar)) AND (¬pegou) AND (¬soltar)) THEN
     pegar := TRUE;
ELSE
     pegar := FALSE;
END_IF

IF (((pegou AND sobreEsteiraA) OR (esquerda)) AND (¬posicaoInicial) AND (¬direita)) THEN
     esquerda := TRUE;
ELSE
     esquerda := FALSE;
END_IF

IF ((posicaoInicial OR soltar) AND (pegou) AND (¬pegar)) THEN
     soltar := TRUE;
ELSE
     soltar := FALSE;
END_IF
```

Figura 1.17: Verificando a legibilidade do texto estruturado

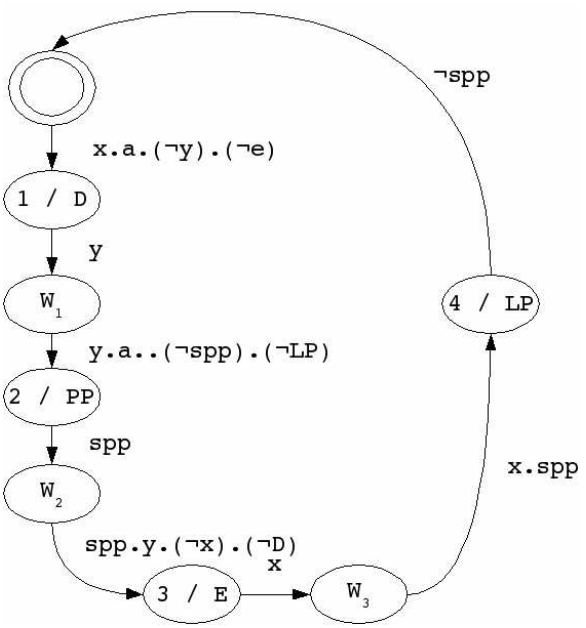


Figura 1.18: Analisando a legibilidade do diagrama de estados

da planta. Assim, dizemos que a legibilidade do texto estruturado se enquadra na *classe A*.

A figura 1.18 implementa um controlador para a mesma planta do exemplo do diagrama *ladder* utilizado nesta sessão. Como podemos ver, se em uma determinada situação o estado ativo for o de número **1**, então não restará dúvidas de que o próximo estado será o estado **2** e não o estado **3**, como poderia acontecer no caso dos diagramas e linguagens de legibilidade *classe A*. Assim, dizemos que o diagrama de estados representa o fluxo de controle do processo da planta explicitamente, e consequentemente possui legibilidade *classe B*.

A figura 1.19 apresenta a lógica temporal equivalente ao diagrama *ladder* utilizado como exemplo nesta sessão. Como podemos ver, olhando apenas para a especificação em lógica temporal podemos concluir que o próximo estado após o estado relativo à variável $\boldsymbol{D}$ será o estado relativo à variável $\boldsymbol{PP}$, ao contrário do que acontece com os diagramas e linguagens de legibilidade *classe A*. Assim, podemos dizer que a lógica temporal definida por Choo et al [43] representa explicitamente o fluxo de controle do processo da planta, e consequentemente possui legibilidade *classe B*.

A figura 1.20 apresenta a especificação de um controlador para a mesma planta do diagrama *ladder* utilizado como exemplo nesta sessão. Como podemos ver, não resta

repouso U ◊((x or a) and (not y) and (not E))] O D
D U ◊ y] O W_1
W_1 U ◊((y or a) and (not spp) and (not LP))] O PP
PP U ◊ spp] O W_2
W_2 U ◊(spp or y and (not x) and (not D))] O E
E U ◊ x] O W_3
W_3 U ◊ (x and spp and (not PP))] O LP
LP U ◊ (not spp)] O repouso

Figura 1.19: Analisando a legibilidade da lógica temporal

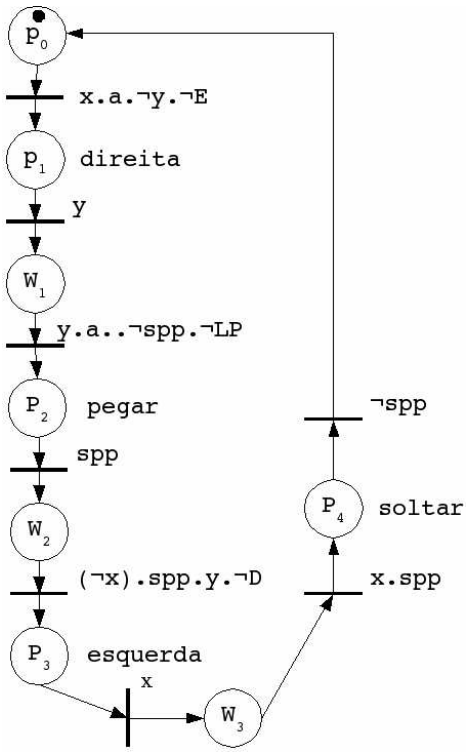


Figura 1.20: Analisando a legibilidade das redes de Petri

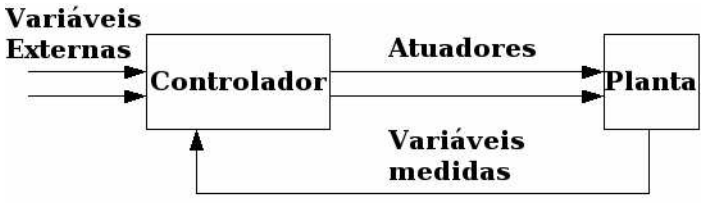


Figura 1.21: A planta como parte do sistema

dúvidas que, olhando apenas para esta rede de Petri, que o passo seguinte ao passo $\boldsymbol{P_1}$ será o passo $\boldsymbol{P_2}$, e não o passo $\boldsymbol{P_3}$. Isto acontece porque a rede de Petri representa o fluxo de controle do processo da planta explicitamente, e consequentemente possui legibilidade *classe B*.

Como podemos ver na figura 1.21, o controlador não é a única peça do sistema, mas também faz parte do sistema a planta que está sendo controlada. Desta forma, *se não utilizarmos informações da planta, não conseguiremos descobrir quais transições poderão acontecer e quais transições não irão acontecer em um diagrama ou linguagem de legibilidade classe A*. Assim, isto faz com que seja extremamente difícil compreender um diagrama ou linguagem desta classe sem que tenhamos informações da planta em mãos. Desta forma, *devemos utilizar informações da planta para analisarmos com êxito diagramas ou linguagens classe A*, como é o caso do diagrama *ladder*.

Com relação aos diagramas e linguagens com legibilidade *classe B*, apesar de todos representarem explicitamente o fluxo de controle do processo da planta, alguns deles possuem mecanismos de representação de paralelismo, como as redes de Petri, os diagramas *grafcet* e a lógica temporal, enquanto que outros como o diagrama de estados, não possuem este mecanismo. Isto acontece porque pela definição do diagrama de estados, pode existir apenas um estado ativo por instante de tempo, enquanto que no diagrama *grafcet*, por exemplo, podem existir dois ou mais passos ativos em um instante de tempo.

1.4 Definição do Problema

Devido a todos os motivos relatados na sessão 1.1, temos agora um problema: dada a especificação de um controlador implementada em um diagrama *ladder*, conseguirmos gerar a partir deste diagrama um diagrama *grafcet* e um diagrama de estados que implemente um controlador equivalente ao controlador especificado a partir do diagrama *ladder*.

Dizemos que dois controladores são equivalentes se, para a mesma planta eles possuem o mesmo funcionamento, ou seja, para um mesmo estado inicial e para uma mesma seqüência de valores de variáveis de entrada eles responderem com a mesma seqüência de valores de variáveis de saída.

Falcione e Krogh[2] e Zanma et al [3] propuseram técnicas de conversão de diagramas *ladder* em diagramas *grafcet*. Já , Lee e Lee [14] propuseram um método de conversão de diagramas *ladder* em redes de Petri. *No entanto, qualquer que seja o método utilizado, ele requererá algum tipo de informação sobre a planta para que a conversão possa ser feita.* O algoritmo de Falcione e Krogh [2] requer esta informação sob a forma de um grafo de simultaneidade. Já o algoritmo de Zanma et al [3] requer informações da planta, que deve ser escrita em lógica temporal. Lee e Lee [14] propõem que esta informação da planta seja dada na forma de uma tabela de estados.

Assim, vamos reformular o nosso problema: precisamos descobrir um método adequado de aquisição de informações da planta para que as mesmas sejam utilizadas no processo de conversão, de tal forma que o usuário possa fazê-la sem que seja necessário descrever completamente a planta. O método não levará em consideração os temporizadores e os blocos avançados do diagrama *ladder*.

1.5 Solução Proposta

Precisamos de um método adequado para entrada de grande quantidade de informações, uma vez que as plantas na indústria são grandes. Também precisamos evitar repetir o processo de conversão, dado a tarefa ser trabalhosa. Temos duas alternativas:

- *Modelagem da planta*: poderíamos utilizar algum tipo de diagrama ou linguagem para descrever previamente o funcionamento da planta e unirmos estas informações com a descrição do controlador para que então pudéssemos converter um diagrama *ladder* em um diagrama *grafcet* e em um diagrama de estados. Levando em consideração o tamanho e a complexidade das plantas existentes na indústria, as quais são o alvo deste trabalho, entrar com estas informações em um sistema de conversão pode requerer um esforço muito grande, consumindo assim tempo e dinheiro.

- *Processo interativo*: Sem que fosse feita uma análise preliminar do diagrama *ladder*, se solicitássemos ao usuário que ele informasse que transições poderiam ocorrer na planta a partir de uma malha de um diagrama *ladder* em uma determinada situação,

deveríamos informar todas as outras malhas do sistema, para que o mesmo selecionasse apenas as malhas em que pudessem ocorrer as transições naquele instante. No entanto, após uma análise das malhas de um diagrama *ladder*, poderíamos filtrar todas as possíveis transições que poderão acontecer em um determinado momento. Assim, uma quantidade muito menor de opções precisaria ser fornecida pelo usuário, uma vez que apenas as malhas que poderão permitir que ocorra uma transição na planta serão apresentadas ao usuário. Além disso, apenas as informações relevantes ao sub-processo do qual estamos extraindo o fluxo de controle da planta serão necessárias, mas não de todo o diagrama *ladder*. Isto também reduz ainda mais a quantidade de opções que serão apresentadas ao usuário. Além disso, estas informações serão solicitadas utilizando uma interface gráfica amigável, com o usuário informando todos os dados necessários ao processo de conversão.

Desta forma, neste trabalho nós utilizamos o método interativo para adicionar informações da planta ao processo de conversão de diagramas *ladder* em diagramas *grafcet* e diagramas de estados, uma vez que ele aparenta requerer do usuário um esforço muito menor.

1.6 Metodologia

Para representarmos o diagrama ladder, usamos uma representação em árvore de cada uma de suas malhas. Podemos fazer cada bobina do diagrama *ladder* representar um estado do diagrama de estados ou um passo do diagrama *grafcet*, após a confirmação do usuário. Associada a cada bobina do diagrama *ladder* temos uma expressão booleana. Uma ligação em paralelo no diagrama *ladder* pode ser representada em sua árvore como um nó do tipo ***or***, enquanto que uma ligação em série pode ser representada como um nó ***and***. Utilizando orientação a objetos, podemos representar os nós desta árvore como objetos polimórficos. Buscamos então utilizar uma ferramenta adequada de manipulação de expressões booleanas, que possam ser construídas a partir destes objetos polimórficos. Como pode ser verificado nos trabalhos de Bryant [16, 20], os BDDs são uma estrutura de dados eficiente de manipulação de expressões booleanas, a qual utilizamos.

Uma vez construídos os BDDs de cada uma das malhas, fazendo a manipulação destes, podemos descobrir as expressões de ativação e desativação de cada uma delas. Assim, fazendo uma análise destas expressões, podemos detectar as possíveis transições que podem acontecer entre cada uma das malhas. Para isto, basta verificar se existe o casamento de

uma expressão de desativação de uma malha com a expressão de ativação de uma outra. Com isto, podemos eliminar uma grande quantidade de possibilidades de casamento, uma vez que se fossem verificadas as combinações entre todas as malhas, o número de possibilidades seria muito maior. Assim, algumas poucas informações precisam ser solicitadas ao usuário. Utilizamos assim um método interativo de obtenção destas informações, fazendo uso de uma interface gráfica amigável.

1.7 Estrutura do Trabalho

Assim, neste trabalho apresentamos uma técnica de engenharia reversa de diagramas *ladder* buscando amenizar a quantidade de informações a serem informadas pelo programador. Desta forma, no capítulo 2 faremos a apresentação de alguns exemplos de plantas que programamos usando o diagrama *ladder*, fazendo uso de um simulador, e depois apresentaremos o diagrama de estados e o diagrama *grafcet* equivalentes. A seguir, faremos uma formalização destes diagramas e formalizaremos o método de engenharia reversa utilizado. No capítulo 3 falaremos sobre as ferramentas que podem ser utilizadas para trabalharmos com expressões booleanas, uma vez que as malhas do diagrama *ladder* podem ser vistas como expressões booleanas. No capítulo 4 faremos a apresentação de um protótipo que implementa o método proposto, com o intuito de validarmos o método. Inicialmente, utilizaremos como exemplo as plantas que podem ser encontradas na bibliografia de Silveira [6], Georgini [7] e Natale [8]. Finalmente, no capítulo 5 mostraremos as conclusões sobre o trabalho.

Capítulo 2

O Método

O objetivo deste capítulo é formalizar um método interativo de recuperação de diagramas *grafcet* e de estados a partir de diagramas *ladder* utilizando teorias matemáticas adequadas já utilizadas na literatura para análise de circuitos de chaveamento.

2.1 Circuitos de Chaveamento

São circuitos cujo comportamento apresenta de alguma forma natureza discreta com apenas dois valores lógicos, ou seja, circuito fechado ou aberto para o caso dos dispositivos lógicos construídos com relés, ou nível de tensão alto ou baixo para o caso de circuitos feitos com transistores ou portas lógicas, por exemplo.

Shannon [45] brilhantemente utilizou a *álgebra booleana* para analisar circuitos de chaveamento. Boole [46] havia criado esta álgebra para trabalhar com raciocínio lógico, formada basicamente pelos elementos $\{\boldsymbol{T}, \boldsymbol{F}, \wedge, \vee, \neg \boldsymbol{x}\}$, onde $\boldsymbol{T}$ significa ***true*** (***verdadeiro***), $\boldsymbol{F}$ significa ***false*** (***falso***), $\wedge$ significa ***and*** (***e***), $\vee$ significa ***or*** (***ou***) e $\neg \boldsymbol{x}$ significa ***not x*** (***não x***). Mais tarde, com a aplicação da álgebra booleana na análise de circuitos digitais, surgiram novas notações, como $\{\mathbf{1}, \mathbf{0}, \cdot, +, \overline{\boldsymbol{x}}\}$ [1], mas sem mudança de significado entre os seus respectivos elementos.

Shannon [45] representou estas operações em termos de circuitos eletromecânicos. Primeiramente, ele configurou os contatos dos relés para conduzir corrente (**1** ou fechado) ou não (**0** ou aberto), quando os mesmos eram energizados ou desenergizados. Com a energização de um relé, os contatos normalmente abertos (**NA**) e os contatos normalmente fechados (**NF**) são automaticamente fechados e abertos, passando de **0** para **1** e de **1** para

0, respectivamente, conforme podemos ver na figura a seguir, extraída do livro de Hill e Peterson [1].

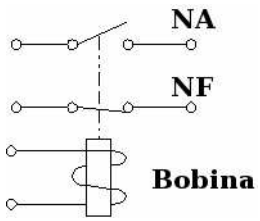


Figura 2.1: Contatos normalmente abertos e fechados [1]

Shannon [45] verificou que uma simples operação ***and*** podia ser utilizada para analisar, por exemplo, dois relés que ligam uma lâmpada. Se apenas um dos relés, ou ambos, estiver desativado, a lâmpada não acenderá, conforme pode ser visto na figura 2.2.

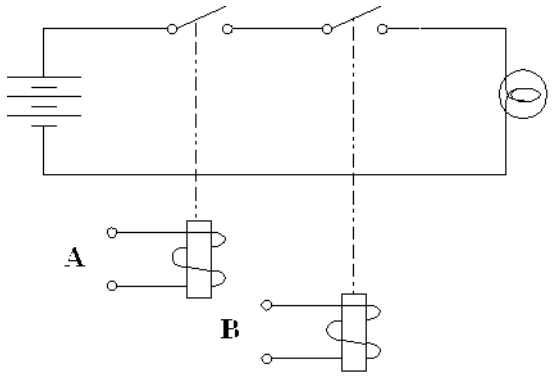


Figura 2.2: A função ***and*** implementada com relés [1]

Da mesma forma, em uma operação ***or*** a lâmpada acenderá se um dos dois relés, ou ambos, fecharem os seus contatos, correspondendo a figura 2.3.

A operação ***not*** é implementada com relés usando um contato normalmente fechado. Assim, a lâmpada acenderá se o relé estiver desativado, e a lâmpada ficará apagada se o relé estiver energizado, conforme podemos ver na figura 2.4.

Shannon [45] também verificou que os contatos dos relés podiam ser combinados para formarem expressões booleanas formadas a partir das operações anteriores. Por exemplo, a figura 2.5 pode ser representada pela expressão booleana $(\boldsymbol{A} \vee \boldsymbol{B}) \wedge (\neg \boldsymbol{C})$.

Seja uma função de chaveamento de $\boldsymbol{n}$ variáveis [1], $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x_1}, \boldsymbol{x_2}, \cdots, \boldsymbol{x_n})$. Shannon [45] também descobriu que escolhendo uma variável $\boldsymbol{x_i}$, podemos reescrever esta função

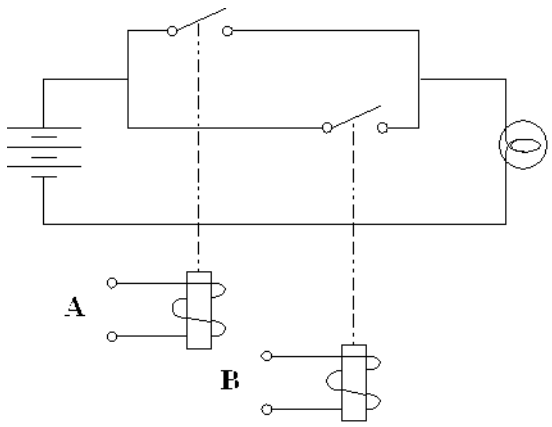


Figura 2.3: A função ***or*** implementada com relés [1]

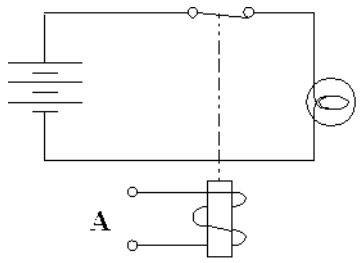


Figura 2.4: A função ***not*** implementada com relés [1]

como $y = x_i \cdot f\mid_{x_i=1} + (\neg x_i) \cdot f\mid_{x_i=0}$. Tomemos como exemplo a expressão booleana relativa ao circuito de relés da figura 2.5, $f(A, B, C) = (A \vee B) \wedge (\neg C)$. Se escolhermos a variável B, podemos reescrever esta expressão como $f(A, B, C) = B \wedge f\mid_{B=1} \vee (\neg B) \wedge f\mid_{B=0}$, onde as novas expressões são $f\mid_{B=1} = f(A, 1, C) = \neg C$ e $f\mid_{B=0} = f(A, 0, C) = A \wedge (\neg C)$. A expansão de Shannon [45] foi utilizada mais tarde por Bryant [16, 20] para construir uma ferramenta computacional chamada Binary Decision Diagram (BDD), que foi utilizada neste trabalho.

Uma vez que os PLCs são uma evolução dos antigos painéis de relés, o diagrama *ladder* herdou os elementos básicos dos relés, ou seja, os contatos normalmente abertos, os contatos normalmente fechados, as bobinas, e as ligações em série e em paralelo. Desta forma, qualquer diagrama de controle usando relés pode ser reescrito utilizando um diagrama *ladder*, sem perda de equivalência, mas apenas de notação simbólica. Assim, podemos dizer que o diagrama *ladder* também é uma ferramenta de implementação de lógica de chaveamento. Como Shannon [45] provou em seu trabalho que a álgebra booleana pode

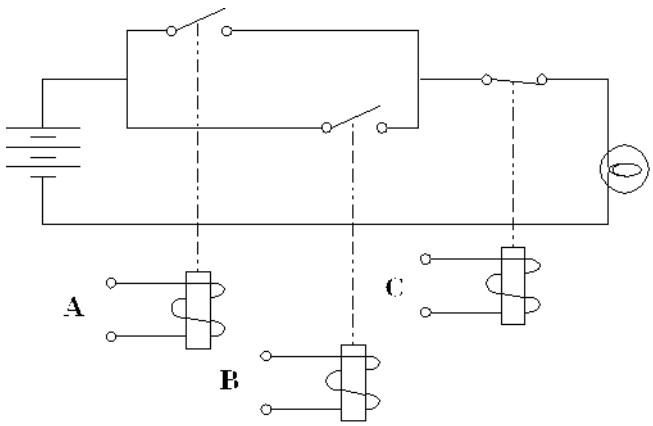


Figura 2.5: Uma expressão booleana implementada com relés [1]

ser utilizada para analisarmos circuitos de chaveamento, tal como os circuitos de relés, podemos utlizar a álgebra booleana para analisarmos diagramas *ladder*.

2.2 Formalização do Diagrama Ladder

Os elementos básicos dos relés foram transportados para o diagrama *ladder* sem mudança de significado lógico, mas apenas de notação simbólica. Os contatos normalmente abertos e normalmente fechados dos relés, como os contatos NA e NF da figura 2.6, foram representados no diagrama *ladder* conforme os contatos NA e NF da figura 2.7.

Da mesma forma que os contatos de um painel de relés podiam estar associados à bobinas de relés ou à qualquer chave vinda da planta, como sensores de fim de curso ou botões, os contatos de um diagrama *ladder* podem estar associados às bobinas ou aos endereços de entrada de um PLC, respectivamente. A relação entre os contatos e as bobinas de um diagrama *ladder* é feita através de endereços internos ou endereços de saída. Assim, formalizamos os elementos básicos de um diagrama *ladder*.

Na figura 2.7 podemos ver os elementos básicos de um diagrama *ladder*, que representam os elementos básicos de um circuito de relés sem perda de significado lógico, mas apenas de notação. A primeira malha apresenta um contato normalmente aberto, enquanto que a segunda um contato normalmente fechado. O círculo representa a bobina, que possue equivalencia lógica com a bobina dos antigos relés.

DEFINIÇÃO: Um diagrama *ladder* é representado por uma tripla $\boldsymbol{\lambda} = (\boldsymbol{R}, \boldsymbol{Q}, \boldsymbol{U})$ onde $\boldsymbol{R}$ é um conjunto de $\boldsymbol{n}$ malhas $\boldsymbol{R} = \{\boldsymbol{r_1}, \boldsymbol{r_2}, ..., \boldsymbol{r_n}\}$, $\boldsymbol{Q}$ é um conjunto de $\boldsymbol{n}$ bobinas

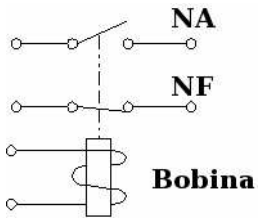


Figura 2.6: Contatos normalmente abertos e fechados [1]

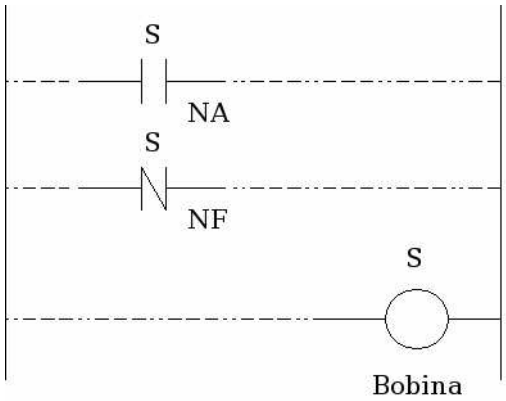


Figura 2.7: Os elementos básicos do diagrama *ladder* são semelhantes aos dos relés

$\boldsymbol{Q = \{q_1, q_2, ..., q_n\}}$, onde cada bobina $\boldsymbol{q_i}$ representa a saída da malha $\boldsymbol{r_i}$, e $\boldsymbol{U}$ é um conjunto de sinais de entrada $\boldsymbol{U = \{u_1, u_2, ..., u_k\}}$, onde $\boldsymbol{k}$ é a quantidade de sinais de entrada de um diagrama *ladder*. Um subconjunto de bobinas de $\boldsymbol{Q}$ pode ser representado por $\boldsymbol{Q_i = \{q_1, q_2, ..., q_p\} \cup \emptyset / p \leq n}$. Um subconjunto de sinais de entrada de $\boldsymbol{U}$ pode ser representado por $\boldsymbol{U_i = \{u_1, u_2, ..., u_q\} \cup \emptyset / q \leq k}$. Cada uma das malhas $\boldsymbol{r_i}$ pode ser representada por uma função $\boldsymbol{q_i = B_i(Q_i, U_i)}$.

Será necessário saber quando uma malha de um diagrama ladder será ativada e quando ela será desativada. Para que isto seja feito, deveremos conhecer os vetores de ativação e desativação de cada bobina do diagrama ladder, que são formados pelos endereços de entrada, ou de bobinas, e que são utilizados pela malha desta bobina para fazer a ativação ou desativação da mesma. Assim, formalizamos estes vetores.

DEFINIÇÃO: Seja uma determinada função booleana $\boldsymbol{q_i = B_i(Q_i, U_i)}$ onde $\boldsymbol{B : 0, 1^{p+q} \longmapsto 0, 1}$. Um vetor pertencente a $\boldsymbol{0, 1^{p+q}}$ que faça com que $\boldsymbol{q_i}$ torne-se verdadeira é chamado de *ativador de* $\boldsymbol{q_i}$, e será representado por $\boldsymbol{\overrightarrow{a_i}}$. Um vetor pertencente a $\boldsymbol{0, 1^{p+q}}$ que faça com que $\boldsymbol{q_i}$ desative-se será chamado de desativador de $\boldsymbol{q_i}$ e será representado

por $\overrightarrow{d_i}$.

A figura 2.10 ilustra um exemplo de vetor de ativação $\overrightarrow{a_1}$ e um exemplo de um vetor de desativação $\overrightarrow{d_1}$.

Uma vez que uma função booleana pode ser obtida a partir do ***ou*** lógico de todos os seus vetores, podemos interpretar uma expressão de ativação e de desativação de uma malha como o conjunto de todos os vetores de ativação e desativação desta malha, respectivamente. Com este objetivo, formalizamos estes conjuntos.

DEFINIÇÃO: Chamaremos de A_i o conjunto de todos os vetores $\overrightarrow{a_j}$, $j \in 1, 2, \cdots, n$, que ativam a bobina q_i, onde n é a quantidade de vetores que ativam a bobina q_i. Também chamaremos de D_i o conjunto de todos os vetores $\overrightarrow{d_j}$, $j \in 1, 2, \cdots, m$, que desativam a bobina q_i, onde m é a quantidade de vetores que desativam a bobina q_i.

Ainda tomando como exemplo a figura 2.10, temos que o conjunto de todos os vetores de ativação da malha 1 é $A_1 = \{\{xa\bar{b}\bar{Y}\}, \{xabQ_0\bar{Y}\}\}$ e o conjunto de todos os vetores de desativação da malha 1 é $D_1 = \{Y\}$.

Podemos representar genericamente a maior parte das malhas utilizadas nos casos reais, a fim de que possamos generalizar a análise que sobre elas será realizada. Com este objetivo formalizamos o conceito de região de uma malha de um diagrama *ladder*.

DEFINIÇÃO: Uma região, ou bloco, de uma malha r_i é uma função booleana $f_i^j = B_i^j(Q_i, U_i)$, tal que a malha r_i representada por $q_i = B_i(Q_i, U_i)$ possa ser representada por $q_i = B_k(Q_k, U_k, f_i^j)$. E ainda, usaremos a notação $f_i^j \sqsubseteq q_i$ para dizermos que f_i^j é subfunção de q_i.

Desta forma, quando nos referirmos a um bloco de uma malha de um diagrama *ladder*, estaremos simplesmente nos referindo a representação gráfica de uma subfunção da função da malha do diagrama *ladder*. Consequentemente, poderemos efetuar as mesmas manipulações lógicas de funções booleanas sobre os blocos de um diagrama *ladder*. Por exemplo, se temos um bloco B_i^1 e outro bloco B_i^2, poderemos construir dois outros blocos fazendo $B_i^1 \vee B_i^2$ e $\neg B_i^1$.

Utilizando os blocos de uma malha, poderemos construir as expressões de ativação e desativação de cada uma delas. Comparando a expressão de desativação de uma malha com a expressão de ativação de outra malha, poderemos detectar o seqüenciamento que poderá ocorrer entre elas. No entanto, quase sempre estas expressões serão diferentes, mas ainda assim poderá haver uma relação entre elas. Desta forma, definimos esta relação.

a	b	$\boldsymbol{a \wedge b}$
0	0	0
0	1	0
1	0	0
1	1	1

Tabela 2.1: Cada linha da tabela é um cubo

DEFINIÇÃO: Sejam duas funções booleanas $\boldsymbol{f : 0,1^m \longmapsto 0,1}$ e $\boldsymbol{g : 0,1^n \longmapsto 0,1}$. Dizemos que $\boldsymbol{f}$ é mais restrita do que $\boldsymbol{g}$ se e somente se $\boldsymbol{g \sqsubseteq f}$ e $\boldsymbol{m > n}$.

Para compararmos a expressão de desativação de uma malha com a expressão de ativação de uma outra, deveremos comparar os vetores destas expressões. Freqüentemente, um vetor de ativação de uma expressão booleana é chamado de *cubo* desta expressão. Assim, podemos dizer que todo vetor de ativação $\overrightarrow{\boldsymbol{a_i}}$ é um cubo da referida função booleana de ativação. Também podemos dizer que todo o vetor de desativação $\overrightarrow{\boldsymbol{d_i}}$ é um cubo da referida função booleana de desativação.

No entanto, devemos tomar cuidado quando nos referirmos a um vetor de ativação de uma expressão e quando nos referirmos a um cubo de uma malha, pois um cubo de uma malha é um vetor de ativação da mesma, mas um cubo da expressão de desativação de uma malha não é um cubo desta malha, mas sim da expressão de desativação da mesma.

DEFINIÇÃO: Seja uma dada função booleana $\boldsymbol{f_i = B_i(Q_i, U_i)}$ definida no espaço $\boldsymbol{B : 0,1^m \longmapsto 0,1}$. Poderão existir $\boldsymbol{n}$ vetores $\boldsymbol{v_k}$ do espaço $\boldsymbol{0,1^m}$ que façam com que a função $\boldsymbol{f_i}$ torne-se verdadeira. Dizemos que cada vetor $\boldsymbol{v_k}$ é um cubo da função $\boldsymbol{f_i}$.

Obviamente, cada cubo $\boldsymbol{v_k}$ pode ser representado por uma função booleana $\boldsymbol{f_k = B_k(Q_k, U_k)}$ definida no espaço $\boldsymbol{B : 0,1^m \longmapsto 0,1}$. Assim, podemos dizer que a função $\boldsymbol{f_i}$ pode ser escrita como uma soma de seus cubos $\boldsymbol{f_i = f_1 \vee f_2 \vee ... \vee f_n}$.

Explicando, dada uma função booleana qualquer, se levarmos em consideração sua tabela verdade, diremos que cada linha da tabela é um cubo da função dada. Por exemplo, na tabela 2.1, a linha **1** e a linha **2** são cubos da função booleana $\boldsymbol{a \wedge b}$.

No entanto, esta é apenas uma das formas de representar uma função booleana, que também pode ser chamada de soma de produtos ou soma de *mintermos*. Assim, todo *mintermo* é um cubo. Uma outra forma de representar uma função booleana se dá através do produto de somas, ou produto de *maxtermos*. Geralmente, um *maxtermo* não é um cubo.

DEFINIÇÃO: Seja uma função booleana $\boldsymbol{f_i = B_i(Q_i, U_i)}$ e $\boldsymbol{f_j = B_j(Q_j, U_j)}$. Seja ainda $\boldsymbol{f_m}$ um cubo de $\boldsymbol{f_i}$ e $\boldsymbol{f_n}$ um cubo de $\boldsymbol{f_j}$. Dizemos que $\boldsymbol{f_m}$ é um sub-cubo de $\boldsymbol{f_n}$, se e somente se a função $\boldsymbol{f_m}$ é sub-função da função $\boldsymbol{f_n}$.

2.3 Análise do Diagrama Ladder

Uma vez compreendidas as definições do diagrama *ladder*, resta agora classificarmos as malhas de um diagrama *ladder* de acordo com a lógica digital que elas implementam. Isto é importante porque dependendo do tipo da malha do diagrama *ladder* que estivermos utilizando, teremos uma forma diferente de manipulação das expressões booleanas equivalentes. Existem duas categorias gerais de circuitos digitais, os circuitos *combinacionais* e os circuitos *seqüenciais* [1].

No circuito *combinacional*, dado um instante de tempo qualquer, a saída depende exclusivamente das variáveis de entrada para este instante de tempo. No circuito *seqüencial*, dado um instante de tempo qualquer, a saída não depende apenas das variáveis de entrada para este instante de tempo, mas também do passado do sistema. Neste caso, precisamos memorizar os estados anteriores do sistema para que possamos utilizá-los juntamente com as variáveis de entrada para obtermos a variável de saída.

No caso dos circuitos digitais, uma forma muito comum de memorizarmos o estado das informações anteriores do sistema se dá através da utilização de dispositivos eletrônicos chamados flip-flops, que são capazes de armazenar o valor de uma variável booleana. Estes dispositivos eletrônicos são utilizados principalmente na construção de memórias, microprocessadores e microcontroladores, além de outros circuitos integrados. No caso de circuitos de relés e diagramas *ladder*, a forma de memorizarmos o estado das informações anteriores do sistema se dá através da realimentação dos contatos dos relés ou bobinas, respectivamente. A forma mais simples de realimentação é a realimentação direta, mostrada na figura 2.8, mas a realimentação também pode ser indireta de $\boldsymbol{n}$ níveis, passando por $\boldsymbol{n}$ circuitos combinacionais até fechar o ciclo de realimentação. A variável utilizada para implementar a realimentação é chamada de selo e geralmente implementa um estado do sistema.

Assim, classificamos as malhas de um diagrama ladder em 2 tipos: malhas combinacionais ou malhas seladas. As malhas seladas podem ainda ser classificadas como de selo direto ou de selo indireto. Falaremos sobre cada uma delas a seguir, no entanto levaremos em consideração apenas as malhas combinacionais e as malhas seladas com selo direto,

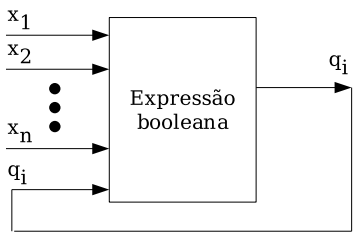


Figura 2.8: Realimentação direta

uma vez que a grande maioria das malhas de diagramas *ladder* existentes se enquadram dentro desta classificação.

Embora a compreensão do que seja, bem como o conceito, de malha combinacional seja mais simples que a compreensão e os conceitos de malha selada com selo direto ou indireto, simplificamos muito a definição de malha combinacional se definirmos primeiro as malhas seladas com selo direto ou indireto. Assim, apresentamos primeiro as definições de malha selada.

2.3.1 Malhas Seladas com Selo Direto

Na maior parte dos casos, um diagrama *ladder* que busque prover o sequenciamento de eventos da planta é implementado usando malhas seladas com selo direto. Cada bobina de uma malha selada é utilizada para representar um determinado estado do sistema.

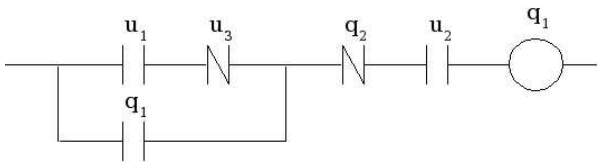
Como podemos ver, a bobina $\boldsymbol{q_1}$ da figura 2.9 somente tornar-se-á ativa se a função

$$\boldsymbol{q_1 = u_1 \wedge \neg u_3 \wedge \neg q_2 \wedge u_2} \tag{2.1}$$

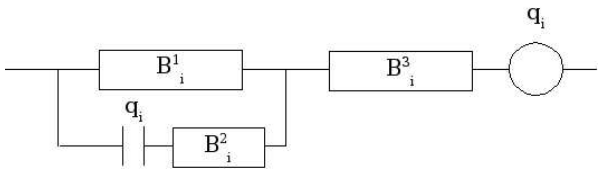
tornar-se verdadeira, ou seja, seus contatos tornarem-se condutores, conforme mostra a figura 2.10b. Uma vez estabelecida a estabilidade da malha e cessada a condição de ativação, ela ficará como mostra a figura 2.10c. Estando na situação da figura 2.10c, a malha somente se desativará se a expressão abaixo tornar-se verdadeira, retornando à situação inicial da figura 2.10a.

$$\boldsymbol{q_2 \vee \neg u_2} \tag{2.2}$$

Uma propriedade interessante da bobina $\boldsymbol{q_1}$ é que após ela tornar-se ativa, ou seja, após $\boldsymbol{u_1 \wedge u_3 \wedge \neg q_2 \wedge u_2}$ tornar-se verdadeiro, mesmo que logo a seguir $\boldsymbol{u_1 \wedge \neg u_3}$



(a) Representação convencional



(b) Representação em blocos

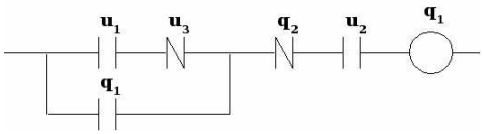
Figura 2.9: Malhas seladas com selo direto

torne-se falso , a bobina q_1 continuará sendo ativada. As malhas caracterizadas por esta propriedade serão chamadas de malhas seladas. Elas são importantes porque elas geralmente implementam estados do sistema. Evidentemente, nem toda bobina q_i que possue um contato normalmente aberto, de endereço q_i, em sua malha, poderá ser considerada selada. Um exemplo é a malha da figura 2.11, pois ela possui um contato normalmente aberto com o mesmo endereço da bobina da referida malha, e no entanto este mesmo contato não desempenha qualquer função de selo, muito pelo contrário, ele faz com que a bobina q_i nunca torne-se ativa. Desta forma, surge a necessidade de formalizarmos a definição de malha selada.

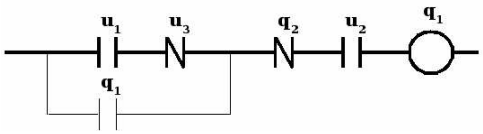
DEFINIÇÃO: Seja uma função booleana $g = B(x_1, x_2, \cdots, x_i, \cdots, x_n)$. Dizemos que a variável de entrada x_i é relevante para a função g se $B\mid_{x_i=1} \neq B\mid_{x_i=0}$. Chamaremos de malha selada toda malha representada pela função $q_i = B_i(Q_i, U_i)$ que possua pelo menos uma subfunção $f_i(y_1, y_2, \cdots, y_k)$, onde $k > 0$, tal que a decomposição $q_i = B_k(Q_i, U_i, f_i)$ seja possível, e que f_i seja relevante para a ativação de q_i, mas não seja relevante para a desativação de q_i.

Como o leitor já deve ter percebido, definimos o que é uma malha selada, mas não definimos se ela possui selo direto ou selo indireto. Fizemos isso com o intuito de aproveitarmos a definição acima na formalização de novos conceitos, como a definição a seguir.

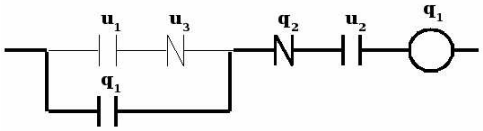
DEFINIÇÃO: Seja uma malha r_i representada pela função $q_i = B_i(x_1, x_2, \cdots, x_n)$.



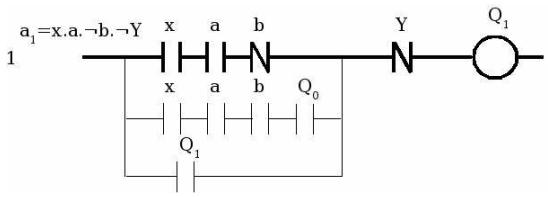
(a) Malha desativada



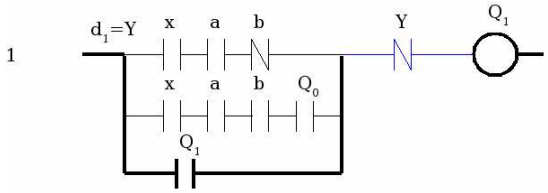
(b) Malha em ativação



(c) Malha em condição estável



(d) Ativação



(e) Desativação

Figura 2.10: Ilustração do processo de ativação e desativação de malhas seladas

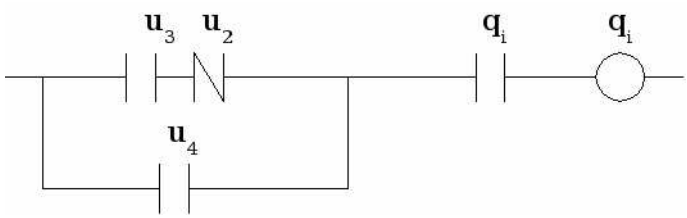


Figura 2.11: Exemplo de uma malha não selada com realimentação

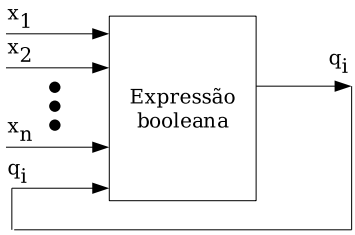


Figura 2.12: Modelo geral de uma malha selada com selo direto

Dizemos que a malha $\boldsymbol{r_i}$ é uma malha selada com selo direto se $\boldsymbol{r_i}$ é uma malha selada e se $\exists \boldsymbol{j} \in \{\boldsymbol{1, 2, \cdots, n}\}$ tal que $\boldsymbol{x_j = q_i}$.

A figura 2.12 ilustra o enunciado.

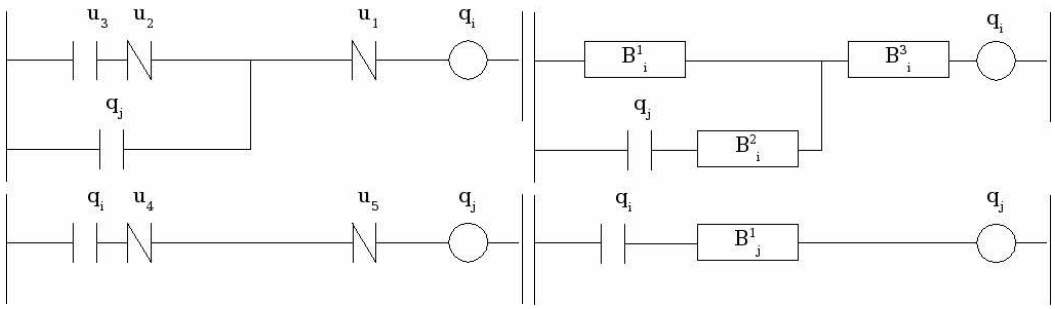
2.3.2 Malhas Seladas com Selo Indireto

As malhas deste tipo também podem ser utilizadas para implementar um estado do sistema, embora sejam bem menos utilizadas do que as malhas seladas com selo direto, uma vez que a utilização das malhas seladas com selo indireto dificultam a legibilidade do diagrama *ladder*. Podemos ver um exemplo deste tipo de malha na figura 2.13.

Como o leitor pode ver, a bobina $\boldsymbol{q_i}$ da figura 2.14a somente ativar-se-á se a expressão

$$\boldsymbol{B_i^1 \wedge B_i^3} \tag{2.3}$$

for verdadeira, como mostra a figura 2.14b. A bobina $\boldsymbol{q_i}$ somente permanecerá ativa se a expressão $\boldsymbol{B_j^1 \wedge B_i^2 \wedge B_i^3}$ for verdadeira, como mostra a figura 2.14c. A seguir, a bobina $\boldsymbol{q_i}$ será desativada se a expressão $\boldsymbol{\neg B_i^2 \vee \neg B_i^3 \vee \neg B_j^1}$ tornar-se verdadeira, como mostra a figura 2.14d.



(a) Representação convencional (b) Representação em blocos

Figura 2.13: Malhas seladas com selo indireto nível um

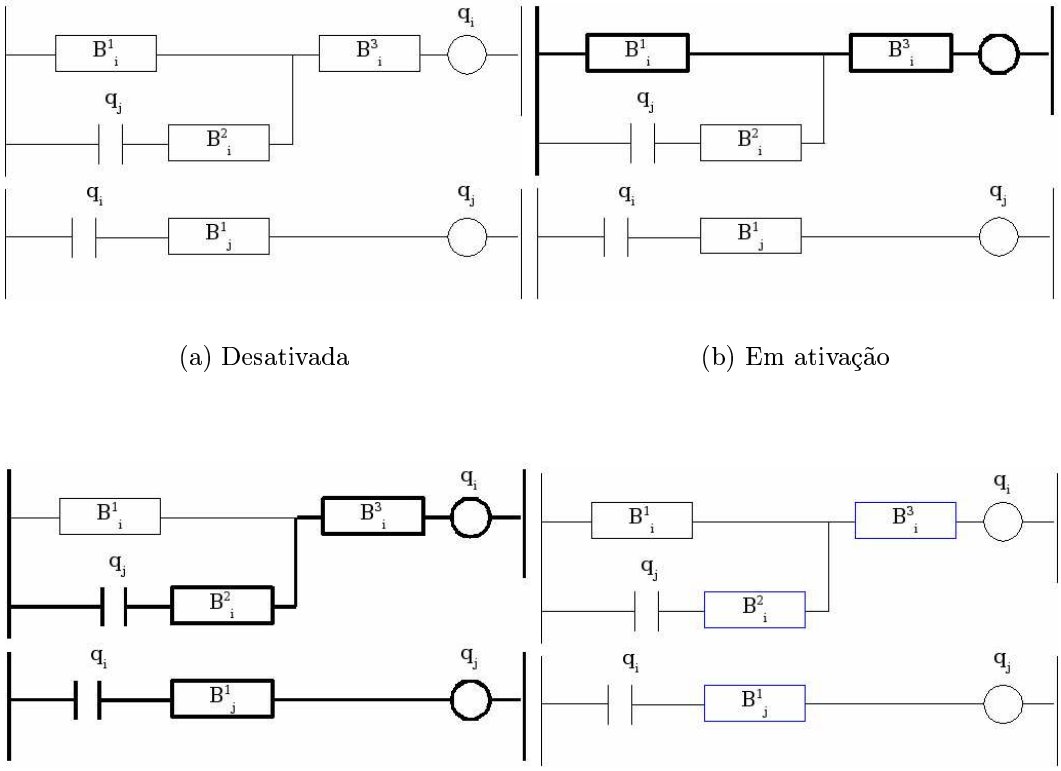
A figura 2.14 apresenta um exemplo de uma malha selada com selo indireto de nível um, uma vez que existe apenas uma malha intermediária entre a bobina que está sendo selada e o seu selo. Semelhantemente, poderíamos ter uma malha selada com selo indireto nível $\boldsymbol{n}$, onde teríamos $\boldsymbol{n}$ malhas intermediárias entre a bobina que está sendo selada e o seu selo.

Antes de formalizarmos o conceito de malha selada com selo indireto, vamos primeiramente fazer uma observação. Imagine que tenhamos uma função booleana $\boldsymbol{q_i} = \boldsymbol{B_i(u_1, u_2, u_3, u_4)}$. Podemos implementar esta função usando um diagrama ladder de várias formas diferentes, como mostram as figuras 2.15, 2.16 e 2.17. Na figura 2.15, implementamos a função booleana $\boldsymbol{q_i} = \boldsymbol{B_i(u_1, u_2, u_3, u_4)}$ de forma direta, pois como podemos ver, não existe nenhuma malha auxiliar entre a bobina $\boldsymbol{q_i}$ e o seu selo. Dizemos assim que esta implementação possui nível zero.

Na figura 2.16, implementamos a mesma função usando a malha de partida e uma malha auxiliar. Como podemos ver, existe uma malha entre a bobina $\boldsymbol{q_i}$ e o contato de endereço $\boldsymbol{q_j}$. Dizemos assim que esta implementação possui nível um, pois existe **1** malha entre $\boldsymbol{q_i}$ e o contato $\boldsymbol{q_j}$. Assim, surge a definição.

DEFINIÇÃO: Seja uma função $\boldsymbol{q_i} = \boldsymbol{B_i(u_1, u_2, \ldots, u_k, \cdots, u_n)}$, de uma malha de um diagrama *ladder*, onde $\boldsymbol{n}$ é o número de endereços do PLC utilizado pelos contatos da malha $\boldsymbol{i}$, e $\boldsymbol{u_k}$ a variável que implemente a função de selo da bobina $\boldsymbol{q_i}$, mas não de endereço $\boldsymbol{q_i}$. Definimos a distância entre $\boldsymbol{q_i}$ e $\boldsymbol{u_k}$ como o número de malhas que existe entre $\boldsymbol{q_i}$ e $\boldsymbol{u_k}$. Se a distância entre $\boldsymbol{q_i}$ e $\boldsymbol{u_k}$ é $\boldsymbol{n}$, então $\boldsymbol{u_k}$ é um selo de nível $\boldsymbol{n}$.

Assim, na figura 2.17, implementamos a função $\boldsymbol{q_i} = \boldsymbol{B_i(u_1, u_2, u_3, u_4)}$ usando a



(a) Desativada

(b) Em ativação

(c) Estável

(d) Em desativação

Figura 2.14: Selo indireto nível um

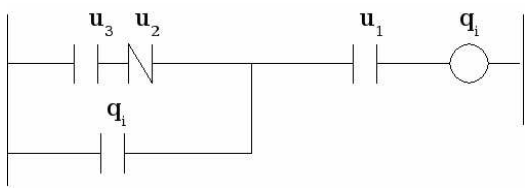


Figura 2.15: Uma função booleana nível zero

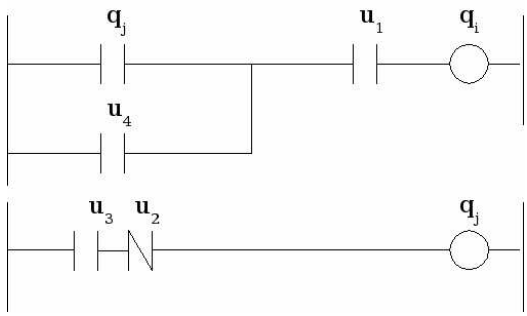


Figura 2.16: Uma função booleana nível um

malha de partida e mais duas malhas intermediárias. Como o leitor pode ver, a distância entre a bobina q_i e o contato q_k é dois, uma vez que existem duas malhas entre a bobina q_i e o contato q_k. Se uma função fosse implementada com n malhas intermediárias, diríamos que sua implementação seria de nível n. Agora podemos formalizar o conceito de malha selada com selo indireto.

DEFINIÇÃO: Seja uma malha de um diagrama ladder r_i representada pela função booleana $q_i = B_i(x_1, x_2, \cdots, x_n)$. Dizemos que a malha r_i do diagrama ladder, com selo x_j, é uma malha selada com selo indireto nível n se a malha r_i é uma malha selada e se a distância entre a bobina q_i e o selo x_j é igual a n.

2.3.3 Malhas Combinacionais

A principal característca deste tipo de malha é que a ativação ou desativação da bobina depende exclusivamente da combinação dos valores das variáveis de entrada, não sendo necessário conhecer os valores anteriores das variáveis de entrada e da própria bobina, conforme pode ser visto na figura 2.18.

Podemos ver na figura 2.19 um exemplo de uma malha combinacional representada sob

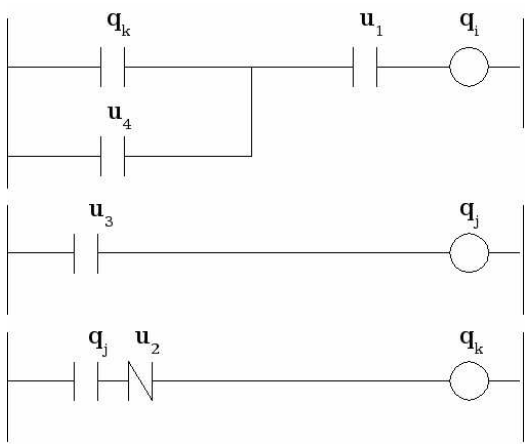


Figura 2.17: Uma função booleana nível dois

Figura 2.18: Modelo geral de uma malha combinacional

duas formas diferentes, ou seja, usando a representação convencional e a representação em blocos. Uma vez que os valores de saída de uma malha combinacional dependem apenas da combinação dos valores das variáveis de entrada, então podemos representar toda malha combinacional utilizando um único bloco, como mostra a figura 2.19b.

Assim, a bobina $\boldsymbol{q_i}$ de uma malha combinacional somente tornar-se-á ativa se a expressão

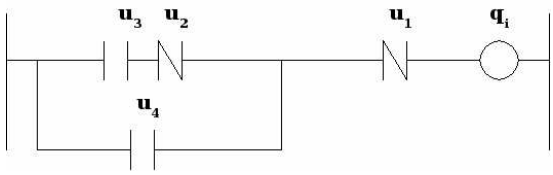
$$\boldsymbol{B_i^1} \tag{2.4}$$

tornar-se verdadeira. Da mesma forma, a bobina $\boldsymbol{q_i}$ de uma malha combinacional somente desativar-se-á se a expressão

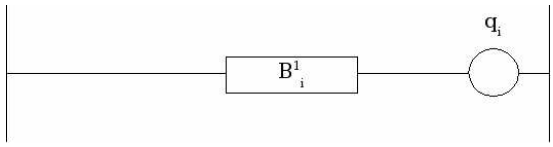
$$\neg \boldsymbol{B_i^1} \tag{2.5}$$

tornar-se verdadeira.

As malhas combinacionais são importantes porque quase sempre elas representam uma ação condicional de algum estado do sistema. No entanto, as vezes elas também



(a) Representação convencional



(b) Representação em blocos

Figura 2.19: Malhas Combinacionais

implementam estados, mas com menor freqüência. Assim, vamos formalizar o conceito de malha combinacional.

DEFINIÇÃO: Seja uma malha r_i de um diagrama ladder representada pela função $q_i = B_i(x_1, x_2, \cdots, x_n)$. Dizemos que a malha r_i é uma malha combinacional se ela não é uma malha selada.

2.4 Formalização do Grafcet

Como podemos ver na figura 2.20, os elementos básicos deste tipo de diagrama são os arcos direcionados, que quando são ascendentes precisam de representação explícita do seu sentido, os passos, que na figura são representados pelos números **2**, **3** e **4**, o passo inicial, representado na figura pelo número **1** e as transições, que possuem um nome, como por exemplo t_1, e uma condição necessária para que a transição possa ser efetuada.

DEFINIÇÃO: Dizemos que um diagrama *grafcet* é formado por um conjunto de passos $P = \{p_1, p_2, \cdots, p_m\}$, por um conjunto de transições $T = \{t_1, t_2, \cdots, t_n\}$ e por um conjunto de arcos $A = \{a_1, a_2, \cdots, a_k\}$ onde ou $a_i = (p_j, t_w)$ ou $a_i = (t_w, p_j)$. Se $\exists t_i \in T$, $a_i, a_j \in A$ tal que $a_i = (p_x, t_i)$ e $a_j = (t_i, p_y)$, então $p_x \rightarrow p_y$.

A definição acima diz basicamente que um diagrama *grafcet* é formado por um conjunto

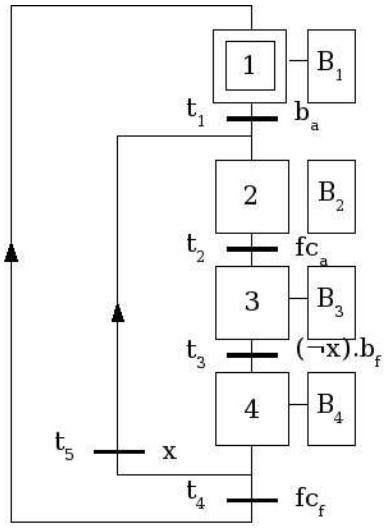


Figura 2.20: Um diagrama grafcet

de passos, um conjunto de transições e por um conjunto de arcos que ligam ou um passo a uma transição ou uma transição a um passo. Ela também diz que se dois passos são ligados por uma transição, então estes dois passos são seqüenciais.

Na figura 2.20 temos que $p_1 \rightarrow p_2$, $p_2 \rightarrow p_3$, $p_3 \rightarrow p_4$, $p_4 \rightarrow p_1$ e $p_4 \rightarrow p_2$.

DEFINIÇÃO: Seja um passo $p_i \in P$. Dizemos que $Ac_i = a_{c1}, a_{c2}, \cdots, a_{cz}$ é o conjunto de todas as ações que estão relacionadas ao passo i, tal que estas ações poderão ser condicionais ou não. Representaremos uma condição de uma ação condicional da seguinte forma: se $\delta(a_{cj}) = x$, então x é a expressão que representa a condição desta ação condicional.

2.5 Formalização do Diagrama de Estados

Como podemos ver na figura 2.24, os elementos básicos deste tipo de diagrama são os estados, como por exemplo *Abrindo*, os arcos direcionados que associam um estado de origem a um estado de destino, e as condições necessárias para que ocorra a transição entre dois estados. Somente um estado pode estar ativo por instante de tempo.

DEFINIÇÃO: Dizemos que um diagrama de estados é formado por uma quádrupla $(E, T, \delta : ExE \longmapsto T, \circledcirc)$ onde $E = \{e_1, e_2, ..., e_m\}$ é um conjunto de estados, $T = \{t_1, t_2, ..., t_n\}$ é um conjunto de transições, $\delta : ExE \longmapsto T$ é uma função que diz

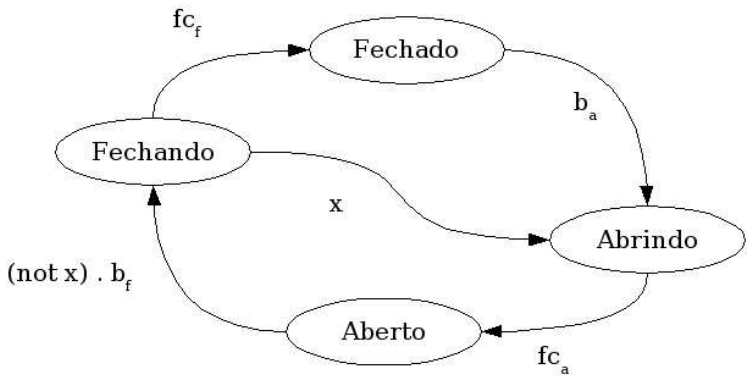


Figura 2.21: Diagrama de estados

qual é a transição entre dois estados e ⊚ é o estado inicial do sistema. Caso dois estados não possuam transição entre si, então a função $\boldsymbol{\delta}$ retornará $\boldsymbol{t_0}$. Caso $\exists \boldsymbol{t_k} \in \boldsymbol{T}$ tal que $\boldsymbol{\delta(e_i, e_j) = t_k}$, então $\boldsymbol{e_i \rightarrow e_j}$.

Uma vez que consideramos que cada malha selada representa um estado, ou um passo (no caso dos diagramas *grafcet*), então precisamos agora apenas descobrir as transições entre estas malhas para que possamos construir um diagrama de estados ou um diagrama *grafcet*. A ferramenta mais adequada a ser utilizada dependerá da planta que estivermos utilizando, ou seja, para plantas onde não há possibilidade de execução de tarefas simultâneas, tanto o diagrama de estados como o diagrama *grafcet* podem ser utilizados, mas quando houver possibilidade de paralelismo, deveremos optar pelo diagrama *grafcet*, uma vez que o diagrama de estados pode ter no máximo um estado ativo por instante de tempo, tornando o diagrama muito complexo para este caso.

2.6 Casos de Mapeamento sem Paralelismo

A fim de explicarmos de forma clara o método proposto de extração de diagramas de estados a partir de diagramas *ladder*, vamos primeiramente exemplificar o processo utilizando casos simples, isto é, que não possuem sub-processos simultâneos.

EXEMPLO: Imaginemos, por exemplo, um sistema de controle de um portão como o que é mostrado na figura 2.22. Neste sistema, as esferas simbolizam os sensores de fim de curso do portão, enquanto o sensor retangular é um sensor de emergência. O sistema funciona da seguinte forma: consideramos inicialmente que o portão está fechado, com o sensor fim de curso $\boldsymbol{fc_f}$ acionado. O motorista que desejar passar pelo portão deverá

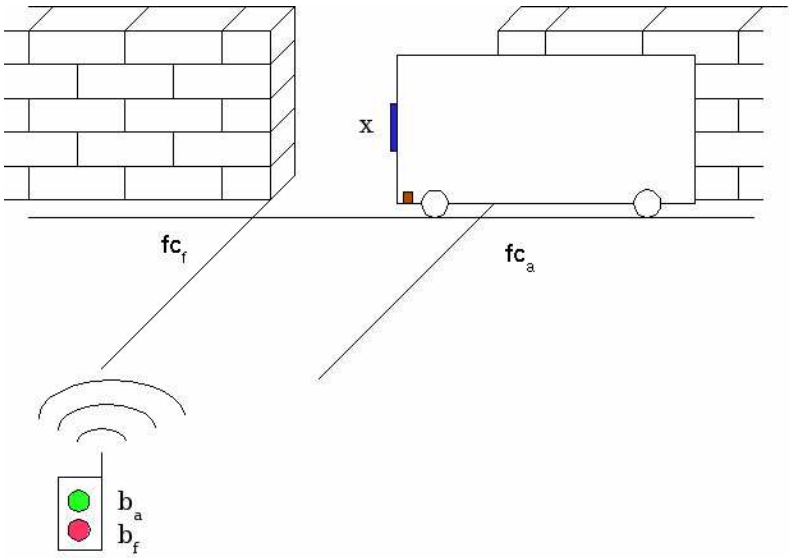


Figura 2.22: Portão

portar um controle remoto, como o que é mostrado na figura. A seguir, o motorista aciona o botão $\boldsymbol{b_a}$, informando ao sistema que ele deseja que o portão abra. Assim, o motor será acionado até que o sensor fim de curso $\boldsymbol{fc_a}$ seja acionado. O veículo então irá passar pelo portão até que o veículo esteja totalmente no outro lado. Por conseguinte, o motorista deverá acionar o botão $\boldsymbol{b_f}$, informando ao sistema que ele deseja que o portão feche. O portão então fechará até que o sensor fim de curso $\boldsymbol{fc_f}$ seja acionado, ou por algum motivo o sensor de emergência $\boldsymbol{x}$ entre em operação. Caso $\boldsymbol{x}$ seja acionado, então o portão deverá parar imediatamente e começar a abrir novamente. Em nenhum momento os motores de abertura e fechamento do portão deverão estar ativos simultaneamente, pois isto possivelmente implicaria em quebras de equipamentos ou até mesmo em risco de incêndio.

Para o sistema da figura 2.22, temos o diagrama *ladder* da figura 2.23 e seu respectivo diagrama de estados na figura 2.24. Assim, temos dois espaços definidos: o espaço do diagrama *ladder* e o espaço do diagrama de estados. Faremos o mapeamento do espaço do diagrama *ladder* para o espaço do diagrama de estados, ou seja, mudaremos a forma de se representar o controlador do processo, mas em ambos os espaços o processo que está sendo controlado é o mesmo, ou seja, teremos dois espaços diferentes, mas equivalentes. Precisamos agora desenvolver um método de engenharia reversa, o qual dado um diagrama *ladder* como o da figura 2.23, consiga gerar o diagrama de estados da figura 2.24. Descrever este método sem uma formalização matemática poderá causar ambiguidades. Desta forma, vamos tentar apresentar o raciocínio através de uma formalização

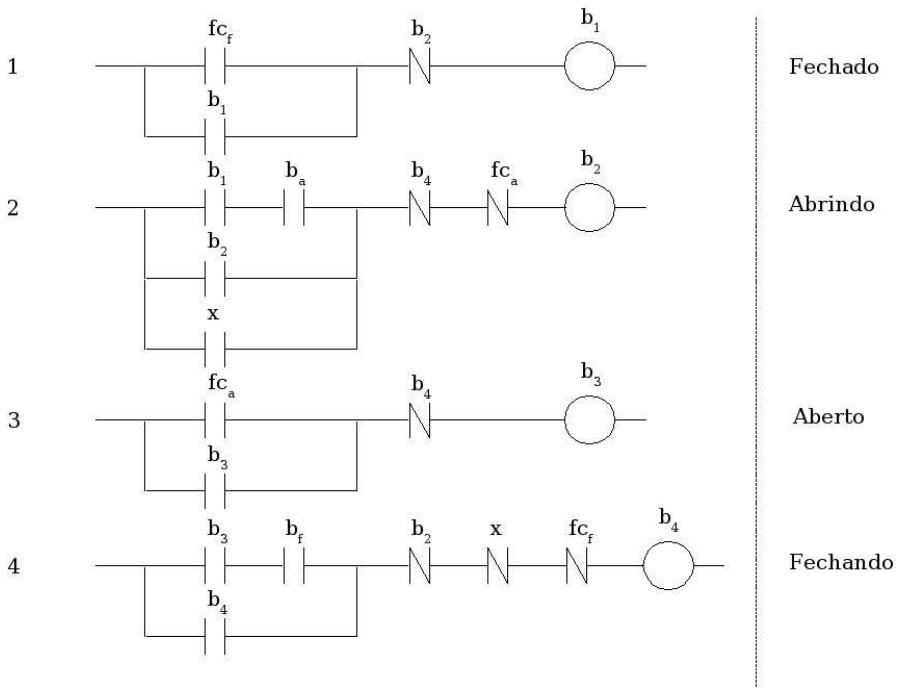


Figura 2.23: Diagrama *ladder*

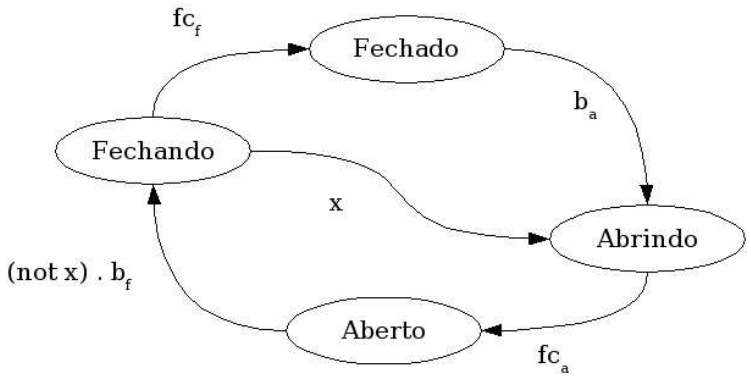


Figura 2.24: Diagrama de estados

matemática sempre que possível.

Como o leitor pode ver na figura 2.24, a condição necessária para que o sistema passe do estado *Abrindo* para o estado *Aberto* é $\boldsymbol{fc_a}$. Se analisarmos o diagrama *ladder* veremos que a função de desativação da malha 2 é $\boldsymbol{b_4} \vee \boldsymbol{fc_a}$, que é exatamente a negação de $\boldsymbol{B_2^3}$. Assim, $\boldsymbol{fc_a}$ é subcubo da função de desativação da malha 2. O leitor também poderá ver que $\boldsymbol{fc_a}$ é subcubo de $\boldsymbol{B_3^1}$. Logo, $\boldsymbol{fc_a}$ é subcubo da função de ativação da malha 3. Olhando para a figura 2.24 o leitor poderá verificar que a malha 3 (*Aberto*) é seqüencial à malha 2 (*Abrindo*), ou seja, a transição entre dois estados do diagrama de estados pode ser implementada no diagrama *ladder* fazendo uma condição ser subcubo da condição de desativação de uma malha e subcubo da condição de ativação da malha seguinte.

Assim, podemos dizer que $\boldsymbol{fc_a} \sqsubseteq \boldsymbol{D_2}$ e $\boldsymbol{fc_a} \sqsubseteq \boldsymbol{B_3^1}$. Como $\boldsymbol{D_2}$ é uma condição de desativação da malha *2* e $\boldsymbol{B_3^1}$ é condição de ativação da malha *3*, então pode existir uma situação em que o sistema desative a malha 2 e ative a malha 3 através de $\boldsymbol{fc_a}$, que é exatamente o que acontece no diagrama de estados da figura 2.24. Desta forma, ao analisarmos um diagrama *ladder*, se encontrarmos alguma função $\boldsymbol{g}$ tal que $\boldsymbol{g} \sqsubseteq \boldsymbol{D_i}$ e $\boldsymbol{g} \sqsubseteq \boldsymbol{B_j^1}$, então muito provavelmente o estado referente à malha j é seqüencial ao estado referente à malha i. Podemos utilizar este padrão como uma heurística para tentarmos descobrir o fluxo de controle do processo da planta de um diagrama *ladder*.

Assim, se conseguirmos recuperar corretamente todos os estados de um diagrama *ladder*, bem como o fluxo de controle do processo da planta, poderemos construir um diagrama de estados a partir de um diagrama *ladder*. Usemos então este raciocínio sobre o diagrama *ladder* da figura 2.23. Vamos atribuir os estados ***Fechado***, ***Abrindo***, ***Aberto*** e ***Fechando*** às malhas $\mathbf{1, 2, 3}$ e $\mathbf{4}$, respectivamente. Como $\boldsymbol{b_2} \sqsubseteq \boldsymbol{D_1}$ e $\boldsymbol{b_2} \sqsubseteq \boldsymbol{B_2^1}$, então $\mathbf{1} \rightarrow \mathbf{2}$ e $\boldsymbol{\delta(1,2) = b_2}$. Da mesma forma, como $\boldsymbol{x} \sqsubseteq \boldsymbol{D_4}$ e $\boldsymbol{x} \sqsubseteq \boldsymbol{B_2^1}$ então $\mathbf{4} \rightarrow \mathbf{2}$ e $\boldsymbol{\delta(4,2) = x}$. Semelhantemente, como $\boldsymbol{b_4} \sqsubseteq \boldsymbol{D_3}$ e $\boldsymbol{b_4} \sqsubseteq \boldsymbol{B_4^1}$ então $\mathbf{3} \rightarrow \mathbf{4}$ e $\boldsymbol{\delta(3,4) = b_4}$. Finalizando, como $\boldsymbol{fc_f} \sqsubseteq \boldsymbol{D_4}$ e $\boldsymbol{fc_f} \sqsubseteq \boldsymbol{B_1^1}$, então $\mathbf{4} \rightarrow \mathbf{1}$ e $\boldsymbol{\delta(4,1) = fc_f}$. Resumindo, temos $\mathbf{1} \rightarrow \mathbf{2}$, $\mathbf{2} \rightarrow \mathbf{3}$, $\mathbf{3} \rightarrow \mathbf{4}$, $\mathbf{4} \rightarrow \mathbf{1}$ e $\mathbf{4} \rightarrow \mathbf{2}$, que é exatamente o diagrama de estados da figura 2.24 e também o diagrama *grafcet* da figura 2.25.

No entanto, o exemplo anterior é muito mais simples do que os problemas encontrados nas indústrias. Os diagramas *ladder* utilizados nas indústrias possuem malhas que são muito mais complexas do que as malhas utilizadas no exemplo anterior, além de possuírem um número maior de malhas. Com o objetivo de simplificar um pouco este trabalho, vamos considerar no máximo 1 estado por cada malha selada. As malhas que não possuírem selo serão consultadas ao usuário se as mesmas implementam um estado ou uma ação condicional, por exemplo. Assim, antes de tentarmos descobrir o fluxo de controle da

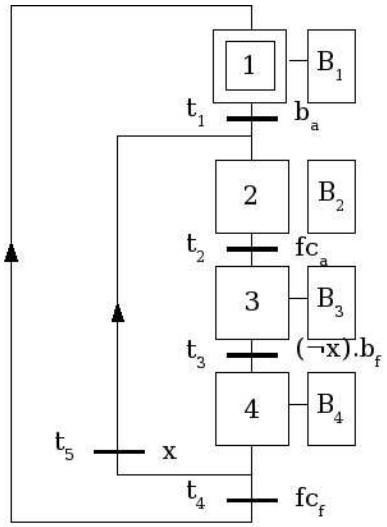


Figura 2.25: Diagrama grafcet do portão

planta, é necessário primeiro descobrir quais são os estados que cada uma das malhas de um diagrama *ladder* pode representar. Vamos então analisar um exemplo usado por Silveira [6], com algumas modificações, de um diagrama *ladder* que possua malhas de maior complexidade.

EXEMPLO: O sistema da figura 2.26 possui duas esteiras de chegada de peças, A e B, além de um carro sobre trilhos com uma garra de pega G e uma esteira de evacuação C. O sistema possui vários sensores de presença, tais como $\boldsymbol{x}$, $\boldsymbol{y}$ e $\boldsymbol{z}$, que detectam a posição do carro, e os sensores $\boldsymbol{a}$ e $\boldsymbol{b}$, que detectam a presença de uma peça sobre as esteiras A e B, respectivamente. O funcionamento consiste em, detectada a presença de uma peça sobre as esteiras de chegada, transportar a peça para a esteira de evacuação. Deve ser dada prioridade a esteira A. Os motores D e E fazem o carro deslocar-se para a direita e para a esquerda, respectivamente. Os atuadores PP e LP fazem a garra pegar e soltar uma peça, respectivamente. O sensor *spp* diz se há uma peça presa pela garra. A figura 2.27 apresenta o diagrama *ladder* deste sistema.

No diagrama *ladder* da figura 2.27 o carro poderá deslocar-se para a direita em 2 situações diferentes: **1*a*)** há uma peça na esteira A **1*b*)** há uma peça na esteira B, mas não na esteira A. Desta forma, teremos 1 estado para a malha 1 e 1 estado para a malha 2. Como o leitor pode ver, a malha 3 não possui selo. Na prática, algumas malhas que não possuem selo podem implementar um estado, enquanto que outras não, implementando apenas ações condicionais. Como não há informação suficiente no diagrama *ladder* para sabermos se a malha implementa um ação condicional ou um estado, então devemos solici-

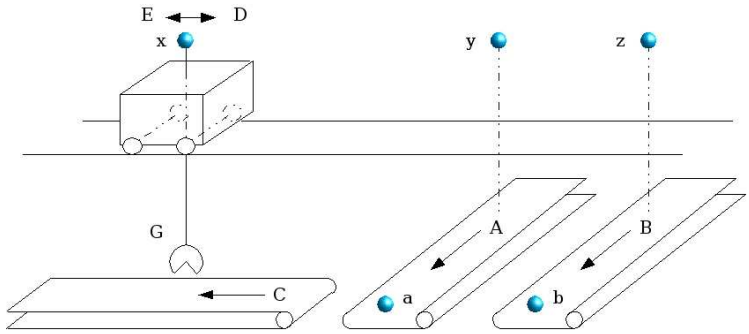


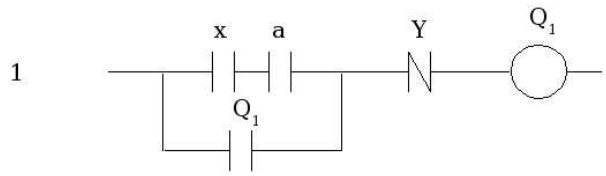
Figura 2.26: Sistema $\boldsymbol{\alpha}$ de transferência de peças

tar ao usuário que ele conceda esta informação. As malhas 5, 6 e 7 também implementam um estado cada.

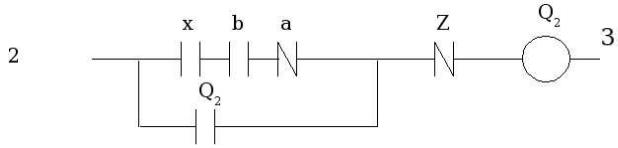
Como $\boldsymbol{D} \sqsubseteq \boldsymbol{D_7}$ e $\boldsymbol{E} \sqsubseteq \boldsymbol{D_3}$, concluímos que as malhas $\mathbf{3}$ e $\mathbf{7}$ nunca poderão estar ativas simultaneamente e portanto fazemos $\boldsymbol{D_7} \leftarrow \boldsymbol{D_7} \mid_{\boldsymbol{D=0}}$ e $\boldsymbol{D_3} \leftarrow \boldsymbol{D_3} \mid_{\boldsymbol{E=0}}$. Temos que $\boldsymbol{D_1} = \neg \boldsymbol{B_1^3} = \neg(\neg \boldsymbol{y}) = \boldsymbol{y}$. Como o leitor pode ver, $\boldsymbol{y} \sqsubseteq \boldsymbol{B_5^1}$ e $\boldsymbol{y} \sqsubseteq \boldsymbol{D_1}$. Isto significa que poderá existir uma situação na planta que faça com que $\mathbf{1} \longrightarrow \mathbf{5}$ através de $\boldsymbol{y}$. Da mesma forma, $\boldsymbol{z} \sqsubseteq \boldsymbol{D_2}$ e $\boldsymbol{z} \sqsubseteq \boldsymbol{B_5^1}$.

Assim, é possível que $\mathbf{2} \longrightarrow \mathbf{5}$. No entanto, $\boldsymbol{B_5^1}$ é mais restritiva do que $\boldsymbol{D_1}$. O mesmo acontece entre $\boldsymbol{B_5^1}$ e $\boldsymbol{D_2}$. Isto poderá fazer com que, caso $\boldsymbol{spp}$ esteja ativo no momento da transição $\mathbf{1} \longrightarrow \mathbf{5}$, então a malha $\mathbf{1}$ será desativada, mas a malha $\mathbf{5}$ não será ativada. Isto pode ser traduzido para o diagrama de estados através de um estado intermediário $\boldsymbol{W_i}$ da seguinte forma: fazemos $\mathbf{1} \longrightarrow \boldsymbol{W_1}$ através de $\boldsymbol{y}$, uma vez que $\boldsymbol{y} \sqsubseteq \boldsymbol{D_1}$, e a seguir fazemos $\boldsymbol{W_1} \longrightarrow \mathbf{5}$, através de $\boldsymbol{y} \wedge \boldsymbol{a} \wedge \boldsymbol{B_1} \wedge (\neg \boldsymbol{spp})$, conforme mostra a figura 2.28. Assim, se no diagrama *ladder* a malha $\mathbf{1}$ for desativada e a malha $\mathbf{5}$ não for ativada, então o estado $\boldsymbol{W_1}$ no diagrama de estados será o estado ativo. Seguindo raciocínio análogo para os estados $\mathbf{2}$ e $\mathbf{5}$, temos que $\mathbf{2} \longrightarrow \boldsymbol{W_2}$ através de $\boldsymbol{z}$ e $\boldsymbol{W_2} \longrightarrow \mathbf{5}$ através de $\boldsymbol{z} \wedge \boldsymbol{b} \wedge \boldsymbol{B_2} \wedge (\neg \boldsymbol{spp})$.

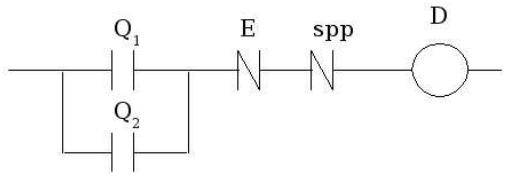
Dando seqüência à resolução do exemplo 2.6, vemos que $\boldsymbol{LP} \sqsubseteq \boldsymbol{D_5}$ e $\boldsymbol{PP} \sqsubseteq \boldsymbol{D_6}$. Logo, a malha $\mathbf{5}$ nunca poderá ser paralela à malha $\mathbf{6}$. Assim, retemos esta informação e fazemos $\boldsymbol{D_5} \leftarrow \boldsymbol{D_5} \mid_{\boldsymbol{LP=0}}$ e $\boldsymbol{D_6} \leftarrow \boldsymbol{D_6} \mid_{\boldsymbol{PP=0}}$. Como o leitor pode ver na figura 2.27, $\boldsymbol{D_5} \sqsubseteq \boldsymbol{B_7^3}$ e $\boldsymbol{D_5} \sqsubseteq \boldsymbol{B_6^3}$. Isto significa que ou $\mathbf{5} \longrightarrow \mathbf{7}$ ou $\mathbf{5} \longrightarrow \mathbf{6}$, ou ambos. Como nós conhecemos a planta com que estamos lidando, nós sabemos que apenas $\mathbf{5} \longrightarrow \mathbf{7}$ e não $\mathbf{5} \longrightarrow \mathbf{6}$. No entanto, não há informação suficiente no diagrama *ladder* para chegarmos a esta conclusão com segurança. Assim, uma forma simples de resolvermos este problema é apresentarmos ao usuário uma interface gráfica que mostre as várias opções de formação do fluxo de controle do processo da planta e deixarmos que ele informe, sem precisar



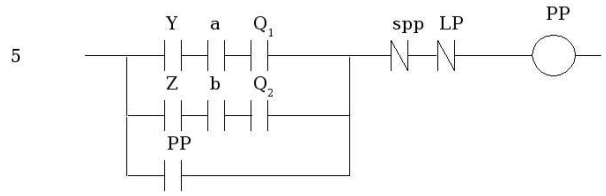
(a) Direita até a esteira A



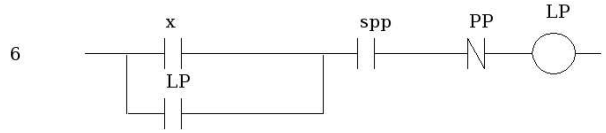
(b) Direita até esteira B



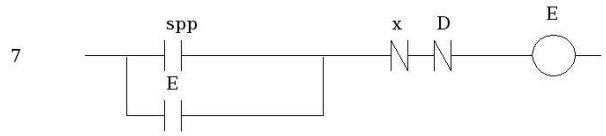
(c) Direita



(d) Pegar



(e) Soltar



(f) Esquerda

Figura 2.27: Diagrama *ladder* do sistema $\boldsymbol{\alpha}$

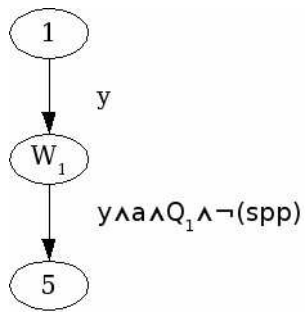


Figura 2.28: Estado intermediário

entrar com grandes quantidades de dados. Mais uma vez, como $\boldsymbol{B_7^1} \wedge \boldsymbol{B_7^3}$ é mais restritivo do que $\boldsymbol{D_5}$, então fazemos $\mathbf{5} \longrightarrow \boldsymbol{W_2}$ através de $\boldsymbol{spp}$ e $\boldsymbol{W_2} \longrightarrow \mathbf{7}$ através de $\boldsymbol{spp} \wedge (\neg \boldsymbol{x})$.

Pode-se ver também que $\boldsymbol{x} \sqsubseteq \boldsymbol{D_7}$, $\boldsymbol{x} \sqsubseteq \boldsymbol{B_6^1}$, $\boldsymbol{D_7} \sqsubseteq \boldsymbol{B_1^1}$ e $\boldsymbol{D_7} \sqsubseteq \boldsymbol{B_2^1}$. Isto corresponderia a $\mathbf{7} \longrightarrow \mathbf{6}$, $\mathbf{7} \longrightarrow \mathbf{1}$ e $\mathbf{7} \longrightarrow \mathbf{2}$. Em um grafo, isto seria equivalente a fazer $\mathbf{6}$, $\mathbf{1}$ e $\mathbf{2}$ paralelos. Mais uma vez não há informação suficiente no diagrama *ladder* para descobrirmos que apenas $\mathbf{7} \longrightarrow \mathbf{6}$, mas $\mathbf{7} \nrightarrow \mathbf{1}$ e $\mathbf{7} \nrightarrow \mathbf{2}$. Assim, mais uma vez requerimos a confirmação do usuário, o qual deverá sinalizar apenas $\mathbf{7} \longrightarrow \mathbf{6}$. Consequentemente, fazemos $\mathbf{7} \longrightarrow \boldsymbol{W_3}$ através de $\boldsymbol{x}$ e $\boldsymbol{W_3} \longrightarrow \mathbf{6}$ através de $(\boldsymbol{x} \wedge \boldsymbol{spp})$. Temos também que $(\neg \boldsymbol{spp}) \sqsubseteq \boldsymbol{D_6}$ e $(\neg \boldsymbol{spp}) \sqsubseteq \boldsymbol{B_3^1}$ e $(\neg \boldsymbol{spp}) \sqsubseteq \boldsymbol{B_5^3}$. Sabemos que com certeza $\mathbf{6} \nrightarrow \mathbf{5}$. Como foi dito anteriormente, a malha $\mathbf{3}$ não implementa um estado, mas uma ação condicional. Desta forma, não podemos fazer $\mathbf{6} \longrightarrow \mathbf{3}$. Assim, não há uma malha $\boldsymbol{\alpha}$ no diagrama *ladder* tal que $\mathbf{6} \longrightarrow \boldsymbol{\alpha}$. Portanto, há apenas uma saída, que é fazer $\mathbf{6} \longrightarrow \circledcirc$.

Uma vez que o grafo formado é cíclico, podemos dizer que ele não possui nem início nem fim. Desta forma, não há como saber qual é o estado inicial do sistema, a menos que haja intervenção do usuário. Assim, *propomos que o usuário informe quais são os estados* $\boldsymbol{\alpha_i}$ *do sistema* tal que $\circledcirc \longrightarrow \boldsymbol{\alpha_i}$. Supondo que o usuário tenha informado os estados $\mathbf{1}$ e $\mathbf{2}$ como estados iniciais do sistema, fazemos $\circledcirc \longrightarrow \mathbf{1}$ através de $\boldsymbol{x} \wedge \boldsymbol{a} \wedge (\neg \boldsymbol{y})$, que é $\boldsymbol{B_1^1} \wedge \boldsymbol{B_1^3}$, e também $\circledcirc \longrightarrow \mathbf{2}$ através de $\boldsymbol{x} \wedge \boldsymbol{b} \wedge (\neg \boldsymbol{a}) \wedge (\neg \boldsymbol{z})$, que é $\boldsymbol{B_2^1} \wedge \boldsymbol{B_2^3}$, além de definitivamente construir o diagrama de estados, como mostra a figura 2.29, e também o diagrama *grafcet*, como mostra a figura 2.30.

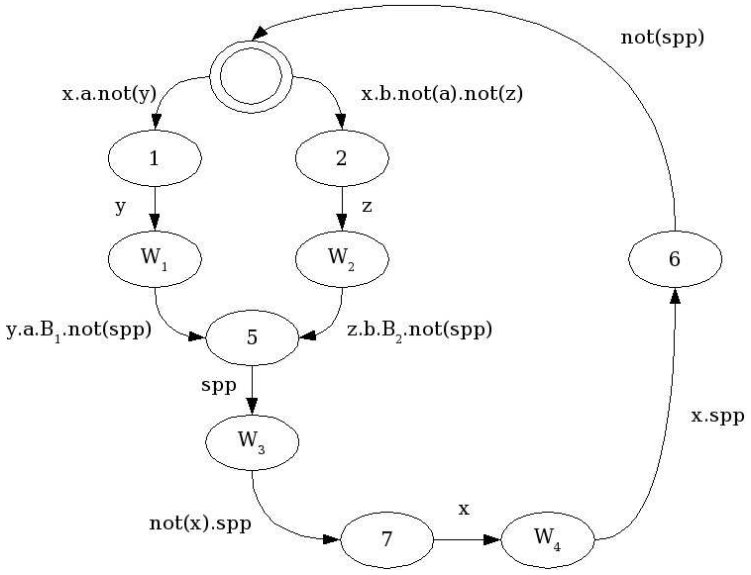


Figura 2.29: Diagrama de estados do sistema $\boldsymbol{\alpha}$

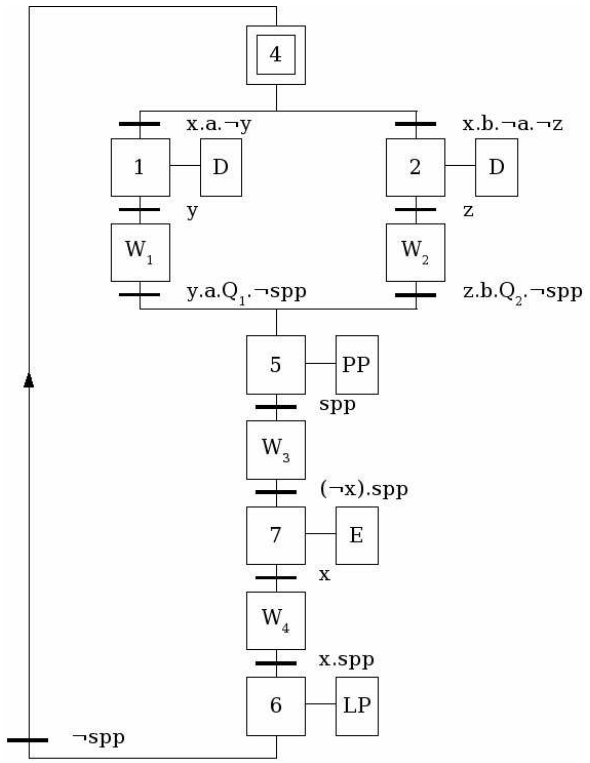


Figura 2.30: Diagrama grafcet do sistema $\boldsymbol{\alpha}$

2.7 Formalização do Método Proposto para Diagramas Grafcet

Baseando-se nos exemplos anteriores, faremos nesta sessão uma formalização do método de engenharia reversa de diagramas *ladder* [1].

1. Conforme definição 2.2, temos inicialmente um diagrama *ladder* $\boldsymbol{\lambda} = (\boldsymbol{R}, \boldsymbol{Q}, \boldsymbol{U})$, onde $\boldsymbol{R}$ é um conjunto de malhas $\boldsymbol{R} = \{\boldsymbol{r_1}, \boldsymbol{r_2}, ..., \boldsymbol{r_n}\}$, $\boldsymbol{Q}$ é um conjunto de bobinas $\boldsymbol{Q} = \{\boldsymbol{q_1}, \boldsymbol{q_2}, ..., \boldsymbol{q_n}\}$, onde cada bobina $\boldsymbol{q_i}$ representa a saída da malha $\boldsymbol{r_i}$, e $\boldsymbol{U}$ é um conjunto de sinais de entrada $\boldsymbol{U} = \{\boldsymbol{u_1}, \boldsymbol{u_2}, ..., \boldsymbol{u_k}\}$, onde $\boldsymbol{k}$ é a quantidade de sinais de entrada de um diagrama *ladder*;

2. $\forall \boldsymbol{p_i} \in \boldsymbol{P}$ será associado uma função de ativação e uma função de desativação, que serão construídas da seguinte forma: a ativação de $\boldsymbol{p_i}$ deverá ser igual a $(\boldsymbol{B_i^1} \vee (\boldsymbol{q_i} \wedge \boldsymbol{B_i^2})) \wedge \boldsymbol{B_i^3}$ e a desativação de $\boldsymbol{p_i}$ deverá ser igual a $\neg(\boldsymbol{B_i^2} \wedge \boldsymbol{B_i^3})$;

3. devemos fazer uma análise identificando os intertravamentos sobre todas as malhas $\boldsymbol{R}$ do sistema $\boldsymbol{\lambda}$. Assim, suponha que existam duas submalhas $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{R}$ tal que $\boldsymbol{q_x} \sqsubseteq \boldsymbol{D_y}$ e $\boldsymbol{q_y} \sqsubseteq \boldsymbol{D_x}$. Logo, elas nunca poderão estar ativas simultaneamente. Se definirmos uma estrutura $(\boldsymbol{x}, \boldsymbol{y})$ representando que as malhas $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{R}$ estão intertravadas, podemos então construir um conjunto $\oplus = \{(\boldsymbol{a}, \boldsymbol{b}), (\boldsymbol{c}, \boldsymbol{d}), \cdots, (\boldsymbol{x}, \boldsymbol{y})\}$ de malhas intertravadas. Uma vez retida esta informação, fazemos $\forall(\boldsymbol{x}, \boldsymbol{y}) \in \oplus$, $\boldsymbol{B_y^{3'}} \leftarrow \boldsymbol{B_y^3} \mid_{\boldsymbol{q_x}=\boldsymbol{0}}$ e $\boldsymbol{B_x^{3'}} \leftarrow \boldsymbol{B_x^3} \mid_{\boldsymbol{q_y}=\boldsymbol{0}}$. Também fazemos $\boldsymbol{r'_y} \leftarrow (\boldsymbol{B_y^1} \vee \boldsymbol{B_y^2}) \wedge \boldsymbol{B_y^{3'}}$ e $\boldsymbol{r'_x} \leftarrow (\boldsymbol{B_x^1} \vee \boldsymbol{B_x^2}) \wedge \boldsymbol{B_x^{3'}}$. Assim, atualizamos o conjunto de malhas do sistema $\boldsymbol{\lambda}$, fazendo $\boldsymbol{R'} \leftarrow \boldsymbol{R} - \{\boldsymbol{r_x}, \boldsymbol{r_y}\} \cup \{\boldsymbol{r'_x}, \boldsymbol{r'_y}\}$. A seguir, fazemos $\boldsymbol{\lambda'} \leftarrow (\boldsymbol{R'}, \boldsymbol{Q}, \boldsymbol{U})$;

4. como explicamos no exemplo anterior, o usuário deverá informar quais são as malhas $\boldsymbol{r_i} \in \boldsymbol{R}$ tais que $\circledcirc \rightarrow \boldsymbol{r_i}$, ou seja, quais são as malhas que representam o estado que vêem após o estado inicial. Seja, $\{\boldsymbol{r_1}, \boldsymbol{r_2}, \cdots, \boldsymbol{r_k}\}$ o conjunto destas malhas. A seguir, para cada malha $\boldsymbol{r_i}$ deverá ser criado um passo $\boldsymbol{p_i}$ e um arco $\circledcirc \rightarrow \boldsymbol{p_i}$ através de $\boldsymbol{t_i} = (\boldsymbol{B_i^1} \vee (\boldsymbol{q_i} \wedge \boldsymbol{B_i^2})) \wedge \boldsymbol{B_i^3}$. Assim, fazemos $\forall \boldsymbol{i}\ \boldsymbol{A} \leftarrow \boldsymbol{A} \cup \{(\circledcirc, \boldsymbol{t_i}), (\boldsymbol{t_i}, \boldsymbol{p_i})\}$, $\boldsymbol{P} \leftarrow \boldsymbol{P} \cup \{\boldsymbol{p_i}\}$, $\boldsymbol{T} \leftarrow \boldsymbol{T} \cup \{\boldsymbol{t_i}\}$;

5. agora está tudo pronto para começarmos a extrairmos o fluxo de controle do processo da planta. Primeiramente, dividimos o conjunto $\boldsymbol{P}$ em dois novos conjuntos $\boldsymbol{\Phi}$ e $\boldsymbol{\varphi}$ de passos analisados e novos passos, respectivamente. Assim, $\forall \boldsymbol{p_i}$ tal que $\circledcirc \rightarrow \boldsymbol{p_i}$,

[1] Caso o leitor deseje saber mais sobre especificação formal de sistemas, uma referência introdutória sobre o assunto é o livro de Pressman [4].

se $p_i \notin \Phi$, então fazemos $\varphi \leftarrow \varphi \cup \{p_i\}$. Agora, enquanto φ é não vazio, permanecemos no passo 6;

6. $a \leftarrow first(\varphi)$, onde a função $first$ retorna o primeiro elemento do conjunto. Depois, fazemos $\varphi \leftarrow \varphi - \{a\}$. A seguir, $\forall r_i \in R$, verificamos quais são as malhas $r_j \in R$ tais que $\exists f$ tal que $f \sqsubseteq D_i$ e $f \sqsubseteq (B_i^1 \vee (q_i \wedge B_i^2)) \wedge B_i^3$. Para todas as malhas $r_j \in R$ confirmadas pelo usuário, então possívelmente $p_i \rightarrow p_j$. No entanto, devemos verificar se $f \neq (B_i^1 \vee (q_i \wedge B_i^2)) \wedge B_i^3$. Caso afirmativo, então não poderemos fazer $p_i \rightarrow p_j$ diretamente, mas deverá existir um estado intermediário W_k tal que $p_i \rightarrow W_k$ através de f, e $W_k \rightarrow p_j$ através de $(B_i^1 \vee (q_i \wedge B_i^2)) \wedge B_i^3$. Assim, faremos $t_a = f$, $t_b = (B_i^1 \vee (q_i \wedge B_i^2)) \wedge B_i^3$, $A \leftarrow A \cup \{(p_i, f), (f, W_k), (W_k, (B_i^1 \vee (q_i \wedge B_i^2)) \wedge B_i^3), ((B_i^1 \vee (q_i \wedge B_i^2)) \wedge B_i^3, p_j)\}$ e $T \leftarrow T \cup \{t_a, t_b\}$. Caso negativo, então fazemos $p_i \rightarrow p_j$ diretamente através de f. Em ambos os casos afirmativo ou negativo, se $p_j \notin \Phi$, então fazemos $\varphi \leftarrow \varphi \cup \{p_j\}$.

O esforço computacional deste método é mais acentuado durante os passos **3**, **4** e **6**. Durante o passo **4**, para cada uma das n malhas do diagrama ladder, verificamos se ela possui intertravamento com as outras $n-1$ malhas. Chamemos de k_1 a ordem de complexidade de comparação entre cada par de malhas. Assim, a complexidade computacional deste passo será da ordem $k_1 \cdot O(n^2)$.

Durante o passo **3**, processaremos todas as n malhas para encontrarmos suas expressões de ativação e desativação. Assim, o tempo de resposta deste passo será da ordem de $O(n) \cdot k_2$, onde k_2 será a ordem do tempo necessário para o sistema construir as expressões de ativação e desativação.

Durante o passo **6**, o sistema irá interagir com o usuário para cada uma das n malhas do sistema. Para cada uma delas, o sistema analisará as outras $n - 1$ malhas a fim encontrar as possíveis transições a serem confirmadas. Uma vez que o sistema interage com o usuário n vezes, o tempo de resposta para cada uma destas n interações será proporcional a $O(n) \cdot k_3$, onde k_3 é ordem de complexidade de comparação entre cada par de malhas.

Com este pré-processamento, o sistema apresenta ao usuário um número reduzido de possíveis transições a serem confirmadas. Além disso, em um diagrama ladder de n malhas, no pior caso, o usuário precisaria interagir com o sistema n vezes. No entanto, as plantas industriais de grande porte são formadas por sub-sistemas menores. Na prática, isto faz com que o usuário interaja com o sistema apenas m vezes, onde m é o número de malhas do sub-sistema em questão.

2.8 Formalização do Método Proposto para Diagramas de Estados

A formalização do método proposto para diagramas de estado é ligeiramente diferente da que é feita para diagramas *grafcet*. Isto acontece porque na formalização de um diagrama *grafcet*, não podemos criar um arco entre dois elementos do mesmo tipo, tal como entre passo e passo ou entre transição e transição. Isto não acontece com a formalização do digrama de estados, uma vez que associamos os estados à seus próximos estados diretamente.

1. Conforme definição 2.2, temos inicialmente um diagrama *ladder* $\boldsymbol{\lambda} = (\boldsymbol{R}, \boldsymbol{Q}, \boldsymbol{U})$, onde $\boldsymbol{R}$ é um conjunto de malhas $\boldsymbol{R} = \{\boldsymbol{r_1}, \boldsymbol{r_2}, ..., \boldsymbol{r_n}\}$, $\boldsymbol{Q}$ é um conjunto de bobinas $\boldsymbol{Q} = \{\boldsymbol{q_1}, \boldsymbol{q_2}, ..., \boldsymbol{q_n}\}$, onde cada bobina $\boldsymbol{q_i}$ representa a saída da malha $\boldsymbol{r_i}$, e $\boldsymbol{U}$ é um conjunto de sinais de entrada $\boldsymbol{U} = \{\boldsymbol{u_1}, \boldsymbol{u_2}, ..., \boldsymbol{u_k}\}$, onde $\boldsymbol{k}$ é a quantidade de sinais de entrada de um diagrama *ladder*;

2. $\forall \boldsymbol{e_i} \in \boldsymbol{E}$ será associado uma função de ativação e uma função de desativação, que serão construídas da seguinte forma: a ativação de $\boldsymbol{e_i}$ deverá ser igual a $(\boldsymbol{B_i^1} \vee (\boldsymbol{q_i} \wedge \boldsymbol{B_i^2})) \wedge \boldsymbol{B_i^3}$ e a desativação de $\boldsymbol{p_i}$ deverá ser igual a $\neg(\boldsymbol{B_i^2} \wedge \boldsymbol{B_i^3})$;

3. devemos fazer uma análise identificando os intertravamentos sobre todas as malhas $\boldsymbol{R}$ do sistema $\boldsymbol{\lambda}$. Assim, suponha que existam duas submalhas $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{R}$ tal que $\boldsymbol{q_x} \sqsubseteq \boldsymbol{D_y}$ e $\boldsymbol{q_y} \sqsubseteq \boldsymbol{D_x}$. Logo, elas nunca poderão estar ativas simultaneamente. Se definirmos uma estrutura $(\boldsymbol{x}, \boldsymbol{y})$ representando que as malhas $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{R}$ estão intertravadas, podemos então construir um conjunto $\oplus = \{(\boldsymbol{a}, \boldsymbol{b}), (\boldsymbol{c}, \boldsymbol{d}), \cdots, (\boldsymbol{x}, \boldsymbol{y})\}$ de malhas intertravadas. Uma vez retida esta informação, fazemos $\forall(\boldsymbol{x}, \boldsymbol{y}) \in \oplus$, $\boldsymbol{B_y^{3'}} \leftarrow \boldsymbol{B_y^3}\,|_{\boldsymbol{q_x}=\boldsymbol{0}}$ e $\boldsymbol{B_x^{3'}} \leftarrow \boldsymbol{B_x^3}\,|_{\boldsymbol{q_y}=\boldsymbol{0}}$. Também fazemos $\boldsymbol{r'_y} \leftarrow (\boldsymbol{B_y^1} \vee \boldsymbol{B_y^2}) \wedge \boldsymbol{B_y^{3'}}$ e $\boldsymbol{r'_x} \leftarrow (\boldsymbol{B_x^1} \vee \boldsymbol{B_x^2}) \wedge \boldsymbol{B_x^{3'}}$. Assim, atualizamos o conjunto de malhas do sistema $\boldsymbol{\lambda}$, fazendo $\boldsymbol{R'} \leftarrow \boldsymbol{R} - \{\boldsymbol{r_x}, \boldsymbol{r_y}\} \cup \{\boldsymbol{r'_x}, \boldsymbol{r'_y}\}$. A seguir, fazemos $\boldsymbol{\lambda'} \leftarrow (\boldsymbol{R'}, \boldsymbol{Q}, \boldsymbol{U})$;

4. como explicamos no exemplo anterior, o usuário deverá informar quais são as malhas $\boldsymbol{r_i} \in \boldsymbol{R}$ tais que $\circledcirc \rightarrow \boldsymbol{r_i}$, ou seja, quais são as malhas que representam o estado que vêem após o estado inicial. Seja, $\{\boldsymbol{r_1}, \boldsymbol{r_2}, \cdots, \boldsymbol{r_k}\}$ o conjunto destas malhas. A seguir, para cada malha $\boldsymbol{r_i}$ deverá ser criado um estado $\boldsymbol{e_i}$ tal que $\circledcirc \rightarrow \boldsymbol{e_i}$ e $\boldsymbol{\delta}(\circledcirc, \boldsymbol{e_i}) = \boldsymbol{t_i} = (\boldsymbol{B_i^1} \vee (\boldsymbol{q_i} \wedge \boldsymbol{B_i^2})) \wedge \boldsymbol{B_i^3}$. Assim, fazemos $\forall \boldsymbol{i}\ \boldsymbol{T} \leftarrow \boldsymbol{T} \cup \{\boldsymbol{t_i}\}$;

5. agora está tudo pronto para começarmos a extrairmos o fluxo de controle do processo da planta. Primeiramente, dividimos o conjunto $\boldsymbol{E}$ em dois novos conjuntos $\boldsymbol{\Phi}$

e φ de estados analisados e novos estados, respectivamente. Assim, $\forall e_i$ tal que $\circledcirc \rightarrow e_i$, se $e_i \notin \Phi$, então fazemos $\varphi \leftarrow \varphi \cup \{e_i\}$. Agora, enquanto φ é não vazio, permanecemos no passo 6;

6. $a \leftarrow first(\varphi)$, onde a função $first$ retorna o primeiro elemento do conjunto. Depois, fazemos $\varphi \leftarrow \varphi - \{a\}$. A seguir, $\forall r_i \in R$, verificamos quais são as malhas $r_j \in R$ tais que $\exists f$ tal que $f \sqsubseteq D_i$ e $f \sqsubseteq (B_i^1 \vee (q_i \wedge B_i^2)) \wedge B_i^3$. Para todas as malhas $r_j \in R$ confirmadas pelo usuário, então possívelmente $e_i \rightarrow e_j$. No entanto, devemos verificar se $f \neq (B_i^1 \vee (q_i \wedge B_i^2)) \wedge B_i^3$. Caso afirmativo, então não poderemos fazer $e_i \rightarrow e_j$ diretamente, mas deverá existir um estado intermediário W_k tal que $e_i \rightarrow W_k$ e $\delta(e_i, W_k = t_a = f$, e $W_k \rightarrow e_j$ e $\delta(W_k, e_j) = t_b = (B_i^1 \vee (q_i \wedge B_i^2)) \wedge B_i^3$. Assim, faremos $T \leftarrow T \cup \{t_a, t_b\}$. Caso negativo, então fazemos $e_i \rightarrow e_j$ diretamente através de f. Em ambos os casos afirmativo ou negativo, se $e_j \notin \Phi$, então fazemos $\varphi \leftarrow \varphi \cup \{e_j\}$.

A análise da eficiência deste método é idêntica ao do método anterior.

2.9 Casos de Mapeamento com Paralelismo

Frequentemente existem situações em que dois ou mais sub-processos do processo que está sendo controlado acontecem simultaneamente, fazendo com que mais de uma malha selada permaneça ativa no mesmo instante de tempo. Vejamos um exemplo.

EXEMPLO: O sistema da figura 2.31 possui duas esteiras de chegada A e B, uma garra de pega G, alojada em um carro sobre trilhos, dois cilindros pneumáticos P e V de liberação de peças e uma esteira de evacuação C das mesmas. A tabela 2.2 resume os atuadores do sistema. Seu funcionamento consiste em verificar a presença de peça em uma das esteiras de chegada, que será então pega pela garra e transportada até a bandeja do cilindro V já previamente na posição alta. A seguir, o cilindro V desce a peça até o nível do cilindro P que, então, evacua a peça pela esteira C. O diagrama *ladder* deve possuir um sistema de prioridade de forma a não acumular peças em uma esteira.

Como o leitor pode ver na figura 2.31, estando o carro ou na posição y ou na posição z, uma vez confirmado que a peça que estava sobre a esteira foi pega pela garra do carro, dois sub-processos serão executados simultaneamente: o primeiro deles consiste em movimentar o carro até a posição x. O segundo consiste em levantar o pistão v. Com isto, teríamos dois estados do diagrama de estados ativos simultaneamente, o que

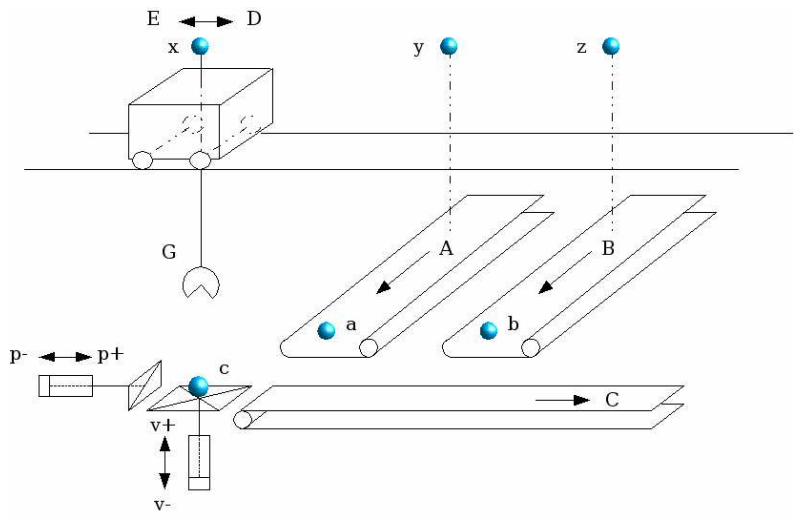


Figura 2.31: Sistema $\boldsymbol{\beta}$ de transferência de peças

seria uma contradição, pois em um diagrama de estados podemos ter apenas um único estado ativo por instante de tempo. Assim, surge a necessidade de utilizarmos o diagrama *grafcet* para estes tipos de sistemas, bem como adaptar o método utilizado para casos de mapeamento sem paralelismo para esta nova situação. Assim, definiremos a notação para passos paralelos e a seguir veremos um exemplo de resolução de sistemas deste tipo.

DEFINIÇÃO: Dois passos $\boldsymbol{i}$ e $\boldsymbol{j}$ são paralelos se e somente se $\boldsymbol{i} \parallel \boldsymbol{j}$. Dois passos $\boldsymbol{i}$ e $\boldsymbol{j}$ poderão tornar-se ativos no mesmo instante de tempo se e somente se $\boldsymbol{i} \shortparallel \boldsymbol{j}$. Dois passos $\boldsymbol{i}$ e $\boldsymbol{j}$ podem permanecer selados simultaneamente se e somente se $\boldsymbol{i} \mid \boldsymbol{j}$.

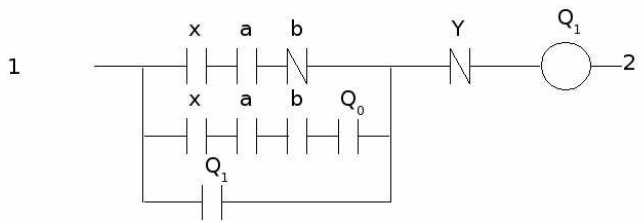
Ainda mais, sejam duas malhas diferentes entre si, $\boldsymbol{q_i} = \boldsymbol{B_i}(\boldsymbol{Q_i}, \boldsymbol{U_i})$ e $\boldsymbol{q_j} = \boldsymbol{B_j}(\boldsymbol{Q_j}, \boldsymbol{U_j})$, onde cada uma delas implementa apenas um passo. Se $\exists \boldsymbol{f}$ tal que $\boldsymbol{f} \sqsubseteq \boldsymbol{B_i^3}$ e $(\neg \boldsymbol{f}) \sqsubseteq \boldsymbol{B_j^3}$, então $\boldsymbol{i} \nparallel \boldsymbol{j}$. Semelhantemente, se $\exists \boldsymbol{f}$ tal que $\boldsymbol{f} \sqsubseteq \boldsymbol{B_i^1}$ e $(\neg \boldsymbol{f}) \sqsubseteq \boldsymbol{B_j^1}$, então $\boldsymbol{i} \nmid \boldsymbol{j}$. Analogamente, se $\exists \boldsymbol{f}$ tal que $\boldsymbol{f} \sqsubseteq \boldsymbol{B_i^2}$ e $(\neg \boldsymbol{f}) \sqsubseteq \boldsymbol{B_j^2}$, então $\boldsymbol{i} \nmid \boldsymbol{j}$.

O diagrama *ladder* da figura 2.31 está representado nas figuras 2.32, 2.33 e 2.34.

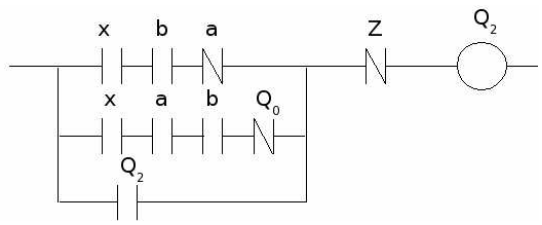
Antes de extrairmos o fluxo de controle do processo, precisamos primeiramente descobrir quais são os intertravamentos existentes no sistema. Uma vez que $\boldsymbol{q_{10}} \sqsubseteq \boldsymbol{D_3}$ e $\boldsymbol{q_3} \sqsubseteq \boldsymbol{D_{10}}$, então as malhas $\mathbf{3}$ e $\mathbf{10}$ estão intertravadas. Assim, fazemos $(\mathbf{3}, \mathbf{10}) \in \oplus$ e $\boldsymbol{D'_3} \leftarrow \boldsymbol{D_3} \mid_{\boldsymbol{q_{10}}=\mathbf{0}}$ e $\boldsymbol{D'_{10}} \leftarrow \boldsymbol{D_{10}} \mid_{\boldsymbol{q_3}=\mathbf{0}}$. Da mesma forma, $\boldsymbol{q_6} \sqsubseteq \boldsymbol{D_8}$ e $\boldsymbol{q_8} \sqsubseteq \boldsymbol{D_6}$. Assim, fazemos $(\mathbf{6}, \mathbf{8}) \in \oplus$, $\boldsymbol{D'_8} \leftarrow \boldsymbol{D_8} \mid_{\boldsymbol{q_6}=\mathbf{0}}$ e $\boldsymbol{D'_6} \leftarrow \boldsymbol{D_6} \mid_{\boldsymbol{q_8}=\mathbf{0}}$. Semelhantemente, $\boldsymbol{q_9} \sqsubseteq \boldsymbol{D_{11}}$ e $\boldsymbol{q_{11}} \sqsubseteq \boldsymbol{D_9}$. Consequentemente, fazemos $(\mathbf{9}, \mathbf{11}) \in \oplus$, $\boldsymbol{D'_9} \leftarrow \boldsymbol{D_9} \mid_{\boldsymbol{q_{11}}=\mathbf{0}}$ e $\boldsymbol{D'_{11}} \leftarrow \boldsymbol{D_{11}} \mid_{\boldsymbol{q_9}=\mathbf{0}}$. Do mesmo modo, $\boldsymbol{q_5} \sqsubseteq \boldsymbol{D_7}$ e $\boldsymbol{q_7} \sqsubseteq \boldsymbol{D_5}$. Logo, fazemos $(\mathbf{5}, \mathbf{7}) \in \oplus$,

Atuador	Descrição
D	Motor que aciona o carro para a direita
E	Motor que aciona o carro para a esquerda
PP	Atuador que faz a garra pegar uma peça
LP	Atuador que faz a garra soltar uma peça
V+	Eletroválvula que comanda o avanço de V
V-	Eletroválvula que comanda o recuo de V
P+	Eletroválvula que comanda o avanço de P
P-	Eletroválvula que comanda o recuo de P
x	Sensor de presença no carro na posição de repouso
y	Sensor de presença do carro sobre a esteira A
z	Sensor de presença do carro sobre a esteira B
a	Sensor de presença de peça na esteira A
b	Sensor de presença de peça na esteira B
c	Sensor de presença sobre a plataforma
spp	Sensor de peça pega pela garra
sv+	Sensor que indica o máximo avanço do cilindro V
sv-	Sensor de posição de recuo total do cilindro V
sp+	Sensor que indica máximo avanço do cilindro P
sp-	Sensor de posição de recuo total do cilindro P

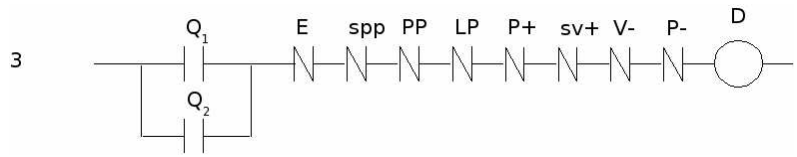
Tabela 2.2: Atuadores



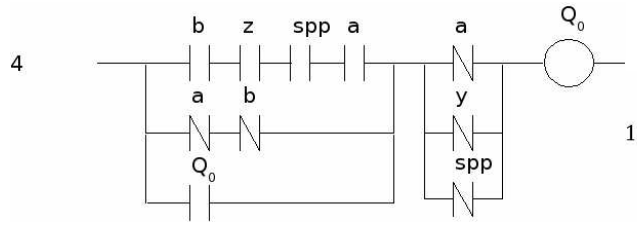
(a) Direita até a esteira A



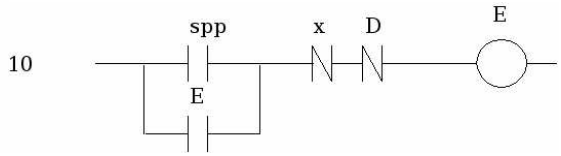
(b) Direita até a esteira B



(c) Direita

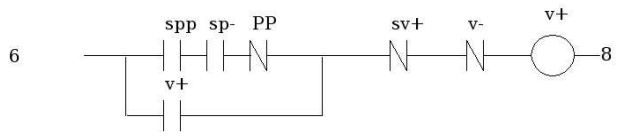


(d) Vez de A

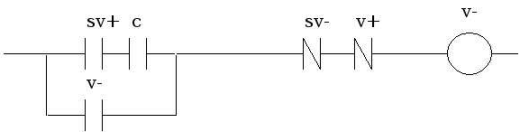


(e) Esquerda

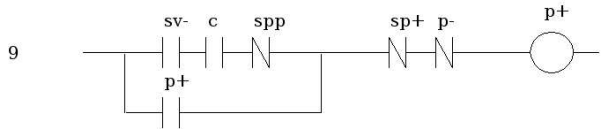
Figura 2.32: Diagrama *ladder* dos motores do sistema $\boldsymbol{\beta}$



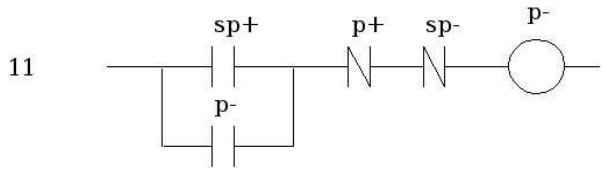
(a) Levantar



(b) Abaixar

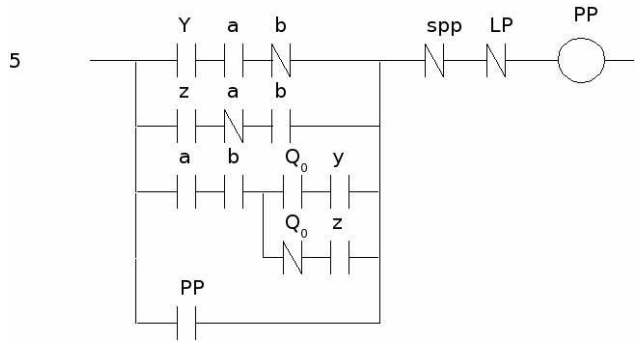


(c) Empurrar

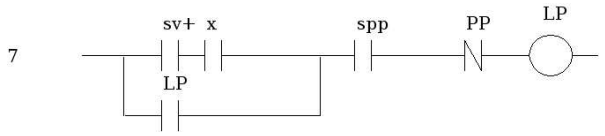


(d) Recuar

Figura 2.33: Diagrama *ladder* dos pistões do sistema $\boldsymbol{\beta}$



(a) Pegar



(b) Soltar

Figura 2.34: Diagrama *ladder* da garra do sistema $\boldsymbol{\beta}$

$\boldsymbol{D'_5 \leftarrow D_5 \mid_{q_7=0}}$ e $\boldsymbol{D'_7 \leftarrow D_7 \mid_{q_5}= 0}$.

Uma vez que $\exists \boldsymbol{f}$ tal que $\boldsymbol{f \sqsubseteq B^1_7}$ e $\boldsymbol{(\neg f) \sqsubseteq B^3_6}$, então segundo a definição 2.9 concluímos que $\mathbf{7} \nparallel \mathbf{6}$. Da mesma forma, uma vez que $\exists \boldsymbol{f}$ tal que $\boldsymbol{f \sqsubseteq B^1_7}$ e $\boldsymbol{(\neg f) \sqsubseteq B^3_{10}}$, então $\mathbf{7} \nparallel \mathbf{10}$. Assim, fazemos $\mathbf{(7,6)}$ e $\mathbf{(7,10)} \in \oplus$. Da mesma forma, $\mathbf{8} \nparallel \mathbf{9}$, e assim fazemos $\mathbf{(8,9)} \in \oplus$.

O próximo passo consiste em obter do usuário a indicação das malhas ou submalhas que seguem o passo inicial do sistema, ou seja, descobrir quais são as malhas $\boldsymbol{x}$ tais que $\circledcirc \rightarrow \boldsymbol{x}$. Assim, temos que $\circledcirc \rightarrow \mathbf{1}$ através de $\boldsymbol{B^1_1 \wedge B^3_1}$ e $\circledcirc \rightarrow \mathbf{2}$ através de $\boldsymbol{B^1_2 \wedge B^3_2}$. Se não levarmos em consideração o selo dessas duas malhas, e se nós chamarmos o conjunto de todos os cubos da malha $\mathbf{1}$ e $\boldsymbol{C_1}$ e da malha $\mathbf{2}$ de $\boldsymbol{C_2}$, veremos que $\forall \boldsymbol{c_i} \in \boldsymbol{C_1}(\exists \boldsymbol{c_j} \in \boldsymbol{C_2})$ tal que $\exists \boldsymbol{f}$ onde $\boldsymbol{f \sqsubseteq c_i}$ e $\boldsymbol{(\neg f) \sqsubseteq c_j}$. Isto implica fazermos $\mathbf{(1,2)} \in \boldsymbol{\eta}$ e, consequentemente, $\mathbf{1} \nparallel \mathbf{2}$. Desta forma, fazemos $\boldsymbol{(\circledcirc, B^1_1 \wedge B^3_1)}$ e $\boldsymbol{(\circledcirc, B^1_2 \wedge B^3_2)} \in \boldsymbol{A}$.

Começamos então a extrair o fluxo de controle do diagrama *ladder*. Como $\boldsymbol{y \sqsubseteq D_1}$ e $\boldsymbol{y \sqsubseteq B^1_5}$, então possivelmente $\mathbf{1} \rightarrow \mathbf{5}$. Após a confirmação do usuário, como $\boldsymbol{y \neq B^1_5 \wedge B^3_5}$, então fazemos $\mathbf{1} \rightarrow \boldsymbol{W_3}$ através de $\boldsymbol{y}$ e $\boldsymbol{W_3} \rightarrow \mathbf{5}$ através de $\boldsymbol{B^1_5 \wedge B^3_5}$. Seguindo raciocínio análogo, temos que $\mathbf{2} \rightarrow \boldsymbol{W_4}$ através de $\boldsymbol{z}$ e $\boldsymbol{W_4} \rightarrow \mathbf{5}$ através de $\boldsymbol{B^1_5 \wedge B^3_5}$.

Como $\boldsymbol{spp \sqsubseteq D_5}$ e $\boldsymbol{spp \sqsubseteq B^1_8 \wedge B^3_8}$ e também $\boldsymbol{spp \sqsubseteq B^1_9 \wedge B^3_9}$ então, após a confirmação do usuário, como $\boldsymbol{spp \neq B^1_8 \wedge B^3_8}$, fazemos $\mathbf{5} \rightarrow \boldsymbol{W_6}$ através de $\boldsymbol{spp}$ e $\boldsymbol{W_6} \rightarrow \mathbf{8}$ através de $\boldsymbol{B^1_8 \wedge B^3_8}$. Seguindo raciocínio análogo, temos que $\mathbf{5} \rightarrow \boldsymbol{W_7}$ através de $\boldsymbol{spp}$ e $\boldsymbol{W_7} \rightarrow \mathbf{9}$ através de $\boldsymbol{B^1_9 \wedge B^3_9}$. Se não levarmos em consideração o selo das malhas $\mathbf{8}$ e $\mathbf{9}$, e se nós chamarmos o conjunto de todos os cubos da malha $\mathbf{8}$ de $\boldsymbol{C_8}$ e da malha $\mathbf{9}$ de $\boldsymbol{C_9}$, veremos que $\forall \boldsymbol{c_i} \in \boldsymbol{C_8}(\neg \exists \boldsymbol{c_j} \in \boldsymbol{C_9})$ tal que $\exists \boldsymbol{f}$ onde $\boldsymbol{f \sqsubseteq c_i}$ e $\boldsymbol{(\neg f) \sqsubseteq c_j}$. Isto significa que é possível que estas duas malhas sejam paralelas. Podemos usar este conceito de *sub-função de exclusão mútua* para alertarmos o usuário de que o que ele está fazendo não é permitido, quando a exclusão mútua for detectada. Em um diagrama *grafcet*, isto é implementado através da estrutura $\boldsymbol{and}$, a qual possui vários arcos partindo de uma mesma transição para vários passos diferentes.

Como $\boldsymbol{x \sqsubseteq D_8}$ e $\boldsymbol{x \sqsubseteq B^1_{12} \wedge B^3_{12}}$, e como $\boldsymbol{x \neq B^1_{12} \wedge B^3_{12}}$, então temos que $\mathbf{8} \rightarrow \boldsymbol{W_{10}}$ através de $\boldsymbol{x}$ e $\boldsymbol{W_{10}} \rightarrow \mathbf{12}$ através de $\boldsymbol{B^1_{12} \wedge B^3_{12}}$. Seguindo raciocínio análogo, temos que $\mathbf{9} \rightarrow \boldsymbol{W_{11}}$ através de $\boldsymbol{sv+}$ e $\boldsymbol{W_{11}} \rightarrow \mathbf{12}$ através de $\boldsymbol{B^1_{12} \wedge B^3_{12}}$. Após a confirmação do usuário, fazemos ambos $\boldsymbol{W_{10}}$ e $\boldsymbol{W_{11}}$ convergirem para o mesmo passo $\mathbf{12}$, através da mesma transição, concluindo assim o paralelismo. Os demais passos podem ser obtidos de forma similar ao caso de mapeamento sem paralelismo, como fizemos anteriormente, resultando no diagrama *grafcet* da figura 2.35. Uma exceção é a malha $\boldsymbol{B_0}$, a qual não possui nenhum passo relativo a ela neste diagrama. Isto significa que ele pertence a uma

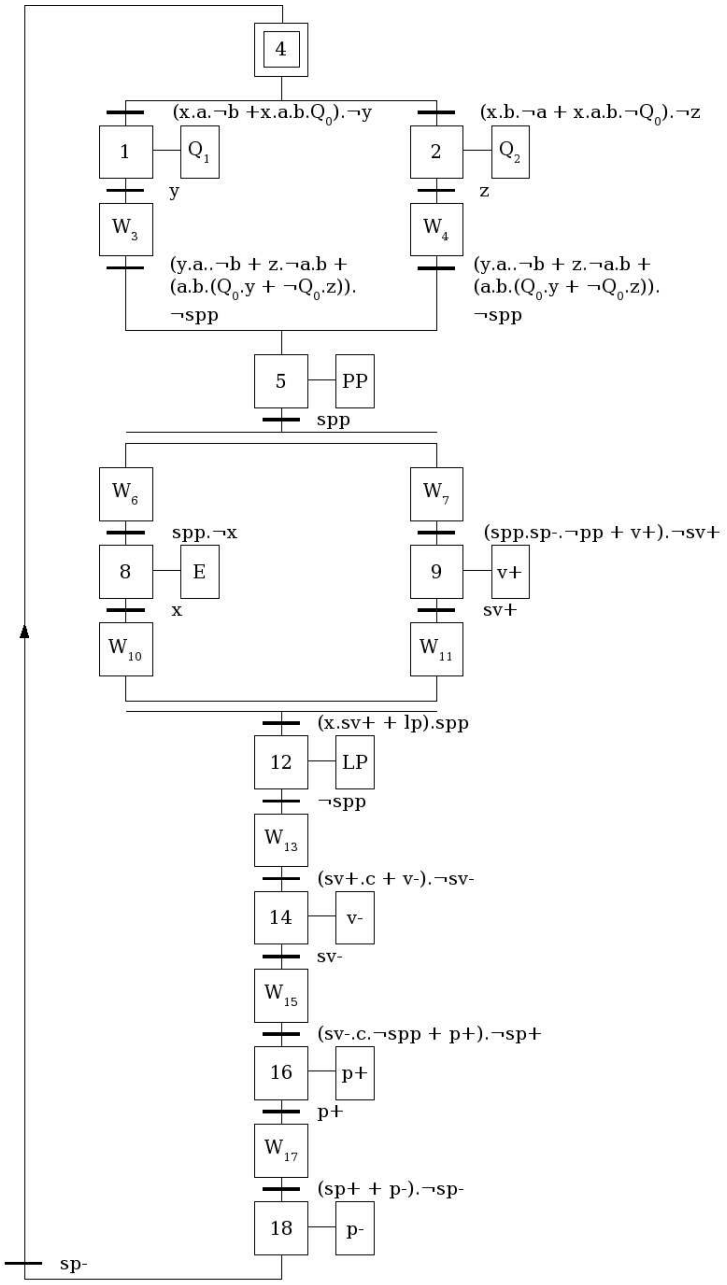


Figura 2.35: Diagrama grafcet do sistema $\boldsymbol{\beta}$

outra máquina de estados. Como esta outra máquina de estados é bastante simples, seu respectivo diagrama grafcet pode ser obtido através do método de extração para casos de mapeamento sem paralelismo descrito anteriormente.

2.10 Formalização do Método Proposto para Detecção de Paralelismo

Uma vez que fizemos a engenharia reversa do diagrama *ladder* sistematicamente, podemos agora fazer uma formalização do mesmo. Evidentemente, não mencionaremos mais o diagrama de estados, uma vez que o diagrama *grafcet* é mais apropriado para representação de sistemas que possuam paralelismo, tendo em vista que um diagrama de estados não pode ter mais do que um estado ativo por instante de tempo.

> O método proposto é semelhante ao método de extração para casos de mapeamento sem paralelismo. A mudança consiste em, detectada um passo $\boldsymbol{p_k}$ possuindo uma mesma transição $\boldsymbol{t} \in \boldsymbol{T}$ saindo para dois passos diferentes $\boldsymbol{p_i}$ e $\boldsymbol{p_j}$, chamamos o conjunto de todos os cubos da malha $\boldsymbol{i}$ de $\boldsymbol{C_i}$ e o conjunto de todos os cubos da malha $\boldsymbol{j}$ de $\boldsymbol{C_j}$. A seguir, se não levarmos em consideração o selo das malhas $\boldsymbol{i}$ e $\boldsymbol{j}$, verificamos que se $\forall \boldsymbol{c_i} \in \boldsymbol{C_i}(\exists \boldsymbol{c_j} \in \boldsymbol{C_j})$ tal que $\exists \boldsymbol{f}$ onde $\boldsymbol{f} \sqsubseteq \boldsymbol{c_i}$ e $(\neg \boldsymbol{f}) \sqsubseteq \boldsymbol{c_j}$, então o usuário deverá ser avisado que as malhas $\boldsymbol{i}$ e $\boldsymbol{j}$ não poderão ser paralelas. Caso contrário, ele deverá confirmar o paralelismo.

O custo computacional do passo acima é proporcional a combinação do número de malhas, $\mathbf{2}$ a $\mathbf{2}$, representada por $\boldsymbol{C_2^n}$, vezes a quantidade de cubos de cada uma delas, vezes uma constante, ou seja, $\boldsymbol{C_2^n} \cdot |\boldsymbol{C_i}| \cdot |\boldsymbol{C_j}| \cdot \boldsymbol{K}$, onde $\boldsymbol{K}$ é o tempo gasto para verificar se $\boldsymbol{f}$ é sub-função do cubo $\boldsymbol{c_i}$ mais o tempo gasto para verificar se $\neg \boldsymbol{f}$ é sub-função de $\boldsymbol{c_j}$. Esta complexidade não é crítica para a quantidade de cubos $\boldsymbol{c_i}$ e $\boldsymbol{c_j}$.

2.11 Conclusão

O método formalizado neste capítulo cobre todos os casos de recuperação de diagramas de estado e de diagramas *grafcet* a partir de diagramas *ladder* formado por malhas de selo direto e por malhas combinacionais. Tendo em vista que o selo direto é muito mais

comum do que o selo indireto, podemos dizer que este método cobre a maior parte dos casos. Embora não tenhamos preparado este método considerando os selos indiretos, trabalhos futuros poderão estender facilmente o método proposto para este caso.

Capítulo 3

Manipulação de Expressões Booleanas

No capítulo anterior, utilizamos álgebra booleana para analisarmos o diagrama *ladder*. Para realizarmos esta análise, precisamos efetuar diversas manipulações algébricas próprias da álgebra booleana sobre a lógica de chaveamento das malhas deste diagrama. Assim, para construirmos um protótipo capaz de analisar um diagrama *ladder*, precisamos utilizar uma ferramenta de manipulação de expressões booleanas. Neste capítulo, estudaremos algumas das diversas ferramentas que podem ser utilizadas para manipulação de expressões booleanas com o objetivo de escolhermos a mais apropriada para o nosso trabalho.

3.1 As Manipulações Algébricas

No capítulo anterior fizemos diversos tipos de operações sobre expressões booleanas, tais como:

- encontrar as expressões de ativação e desativação de uma malha de um diagrama *ladder*
- encontrar todos os cubos de uma expressão booleana;
- verificar se existe um cubo ou sub-cubo de uma função booleana $\boldsymbol{f_1}$ que seja cubo ou sub-cubo de uma função booleana $\boldsymbol{f_2}$;
- dada uma função booleana $\boldsymbol{f_1}$ e uma subfunção de $\boldsymbol{f_1}$ chamada $\boldsymbol{f_2}$, achar uma nova função $\boldsymbol{f_3}$ obtida a partir de $\boldsymbol{f_1}$ fazendo $\boldsymbol{f_2}$ verdadeira ou fazendo $\boldsymbol{f_2}$ falsa;

- dadas duas funções booleanas f_1 e f_2, descobrir se há um sub-cubo g comum a ambas f_1 e f_2. Caso afirmativo, informar qual é o sub-cubo g;

Existem diversas ferramentas na literatura candidatas a desempenhar estas operações, como por exemplo o diagrama de decisões binárias (BDDs) e a álgebra computacional. O BDD possui características muito interessantes, como a canonicidade, ou seja, duas funções booleanas são equivalentes se e somente se elas possuem o mesmo BDD, e a eficiência, conseqüência da capacidade dos BDDs de reduzir uma função booleana a uma forma compacta de representação utilizando grafos direcionados. Estes são algums dos motivos que fazem do BDD a solução adotada pela indústria na síntese de circuitos VLSI há décadas.

A álgebra computacional também foi utilizada em aplicações não-acadêmicas, como simulação e diagnóstico de circuitos elétricos, e em aplicações acadêmicas, como na teoria de controle, dentre outros. Apesar da álgebra computacional ser capaz de simplificar uma função booleana, ela não a reduz a uma forma canônica. Isto dificulta muito a verificação de equivalência entre funções booleanas, capacidade esta necessária para realizarmos diversas manipulações algébricas neste trabalho. Assim, utilizamos os BDDs, cuja concepção e construção está baseada na expansão de Shannon.

3.2 A Expansão de Shannon

No cálculo infinitesimal qualquer função contínua, onde suas derivadas também sejam contínuas, pode ser expandida em uma série de Taylor. Temos uma expansão similar no cálculo proposicional, chamada de expansão de Shannon [45].

TEOREMA: Seja $f : B^n \mapsto B$ uma função booleana de n variáveis. Para as funções g e h $\in B^{n-1} \mapsto B$ tal que $g(x_1, x_2, \cdots, x_n) = f(0, x_2, \cdots, x_n)$ e $h(x_1, \cdots, x_n) = f(1, x_2, \cdots, x_n)$, temos

$$f = \overline{x_1} \cdot g + x_1 \cdot h$$

Como consequencia da expansão de Shannon para a primeira variável de uma função, temos as expansões 3.1, 3.2 e 3.3 que são respectivamente a expansão de Shannon para qualquer variável , a expansão dual de Shannon e a expansão de Shannon com relação ao ou-exclusivo [42].

COROLÁRIO: Para cada função $\boldsymbol{f : B^n \mapsto B}$ e $\forall \boldsymbol{i} \in \{\boldsymbol{0}, \cdots, \boldsymbol{n}\}$ as seguintes afirmações são verdadeiras

$$\boldsymbol{f(x_1, \cdots, x_n) = x_i \cdot f(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n) + \overline{x_i} \cdot f(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n)} \tag{3.1}$$

$$\boldsymbol{f(x_1, \cdots, x_n) = (x_i + f(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n)) \cdot (\overline{x_i} + f(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n))} \tag{3.2}$$

$$\boldsymbol{f(x_1, \cdots, x_n) = x_i \cdot f(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n) \oplus \overline{x_i} \cdot f(x_1, \cdots, x_{i-1}, 0, x_{i+1}, \cdots, x_n)} \tag{3.3}$$

onde a equação 3.1 é conhecida como expansão de Shannon, a equação 3.2 como expansão dual de Shannon e a equação 3.3 como expansão de Shannon com respeito ao ou-exclusivo.

Embora estejamos falando das várias versões da expansão de Shannon, para nós a mais importante e a única que estamos utilizando neste trabalho é a expansão 3.1, pois ela foi utilizada na formalização e construção dos BDDs [16, 20], que foram utilizados neste trabalho.

3.3 Diagramas Binários de Decisão

São conhecidos na literatura como BDDs, resultado da abreviação de Binary Decision Diagrams. Eles são uma estrutura de dados utilizadas para representar, de forma compacta e única, expressões booleanas, juntamente com um conjunto de algoritmos de manipulação destas estruturas de dados [16, 20]. Por exemplo, seja a função booleana $\boldsymbol{f(x_1, x_2, x_4) = x_1 \cdot (x_2 + \overline{x_2} \cdot x_4) + \overline{x_1} \cdot x_4}$. Esta função pode ser representada por um BDD, conforme a figura 3.1.

Foram utilizados por Jacobi [23] para fazer decomposição lógica, por Lai et al [24] para decomposição de funções lógicas aplicadas à síntese de dispositivos SPGA, por Scholl [21] para fazer simulação funcional, por Kuo et al [22] para fazer o particionamento lógico de circuitos seqüenciais, por Yuan et al [17] para fazer a modelagem de técnicas de simulação, tais como *Constraining e input biasing*, por Thacker e Myers [18] para fazer a síntese de circuitos temporizados e por Sztipanovits e Misra [19] para fazer o diagnóstico de sistemas de eventos discretos, entre outros.

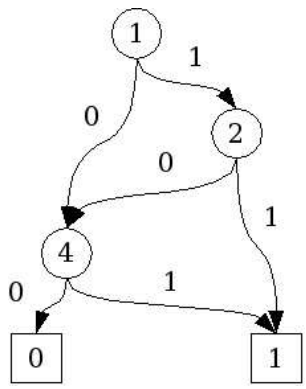


Figura 3.1: Exemplo de um BDD

3.3.1 Representação

Os BDDs são representados graficamente como *grafos direcionados* formados por arcos direcionados, nós terminais e nós não terminais [16, 20]. Os nós terminais representam os valores lógicos **0** e **1**. Os nós não terminais representam as variáveis da função booleana que está sendo representada pelo BDD. Todo nó não terminal possui dois arcos de saída, um para o valor lógico **0** e o outro para o valor lógico **1**.

Navegando pelo BDD, quando partirmos de um nó não terminal através de um arco direcionado, assumimos que o valor do arco direcionado seguido será atribuido à variável relativa a este nó. Ao terminarmos de navegar pelo BDD, o nó terminal encontrado representará o valor da função booleana para os valores de variáveis que foram assumidos durante o percurso. Um conjunto de definições, lemas e teoremas formalizam os BDDS [16, 20].

DEFINIÇÃO: Um grafo de uma função é um grafo direcionado, possuindo uma raiz e um conjunto de vértices $\boldsymbol{V}$ contendo dois tipos de vértices. Um vértice não-terminal $\boldsymbol{v}$ possui como atributos um argumento $\boldsymbol{index(v)} \in \{\mathbf{1}, \mathbf{2}, \cdots, \boldsymbol{n}\}$ e dois filhos $\boldsymbol{low(v), high(v)} \in \boldsymbol{V}$. Um vértice terminal possui como atributo um valor $\boldsymbol{value(v)} \in \{\mathbf{0}, \mathbf{1}\}$.

Exemplificando, na figura 3.1 os retângulos são os vértices terminais e os círculos são os vértices não terminais.

DEFINIÇÃO: Um grafo $\boldsymbol{G}$ de uma função, tendo como raiz um vértice $\boldsymbol{v}$ descreve uma função $\boldsymbol{f_v}$ definida recursivamente como: **1)** Se $\boldsymbol{v}$ é um vértice terminal, então: $\boldsymbol{a}$**)** Se $\boldsymbol{value(v)} = \mathbf{1}$, então $\boldsymbol{f_v} = \mathbf{1}$. $\boldsymbol{b}$**)** Se $\boldsymbol{value(v)} = \mathbf{0}$, então $\boldsymbol{f_v} = \mathbf{0}$. **2)** Se $\boldsymbol{v}$ é um

vértice não-terminal com $\boldsymbol{index(v) = i}$, então $\boldsymbol{f_v}$ é a função

$$\boldsymbol{f_v(x_1, \cdots, x_n) = \overline{x_i} \cdot f_{low(v)}(x_1, \cdots, x_n) + x_i \cdot f_{high(v)}(x_1, \cdots, x_n)}$$

Como podemos ver, podemos construir uma função booleana recursivamente a partir de um BDD. Uma característica dos BDDs é a capacidade de representar compactamente uma função booleana, ao contrário do que acontece quando utilizamos outras formas de representação, como a soma de produtos ou o produto de somas.

No caso do produto de somas, cada termo do produto é chamado de *maxtermo*, somas canônicas ou somas padrões. No caso da soma de produtos, cada termo da soma é chamado de *mintermo*, produtos canônicos ou produtos padrões. Podemos encontrar todos os *mintermos* de uma função encontrando todos os caminhos de um BDD que saem da raiz e chegam até o nó terminal de valor **1**. Embora isto seja possível, é preferível preservar a representação da função usando BDDs sempre que possível, uma vez que esta representação é mais compacta para funções maiores.

Para que possamos fazer com que o BDD seja uma estrutura compacta, é necessário que façamos uma série de simplificações em sua estrutura, eliminando assim informações redundantes. Para descobrirmos que informações redundantes são estas, é definido o conceito de isomorfismo a seguir .

DEFINIÇÃO: Dois grafos $\boldsymbol{G}$ e $\boldsymbol{G'}$ são isomórficos se existe uma função $\boldsymbol{\sigma}$, de um para um, a partir dos vértices de $\boldsymbol{G}$ para os vértices de $\boldsymbol{G'}$ tal que para qualquer vértice $\boldsymbol{v}$, se $\boldsymbol{\sigma(v) = v'}$, então ou $\boldsymbol{v}$ e $\boldsymbol{v'}$ são vértices terminais com $\boldsymbol{value(v) = value(v')}$, ou ambos $\boldsymbol{v}$ e $\boldsymbol{v'}$ são não terminais com $\boldsymbol{index(v) = index(v')}$, $\boldsymbol{\sigma(low(v)) = low(v')}$ e $\boldsymbol{\sigma(high(v)) = high(v')}$.

Uma vez descobertos dois sub-grafos isomórficos em um BDD, podemos chegar a conclusão que um deles é redundante. Com isso retiramos o sub-grafo redundante do BDD e adaptamos a nova estrutura ao sub-grafo que ficou. No entanto, não definimos ainda o conceito de sub-grafo. Assim, surge a necessidade da definição a seguir.

DEFINIÇÃO: Para qualquer vértice $\boldsymbol{v}$ em um grafo $\boldsymbol{G}$, o sub-grafo cuja raiz é $\boldsymbol{v}$ é definido como o grafo consistindo de $\boldsymbol{v}$ e todos os seus descendentes.

Agora que formalizamos o conceito de sub-grafo, podemos exemplificar o processo de simplificação de um BDD. Vejamos a figura 3.2. Na figura 3.2a podemos ver o BDD ainda não simplificado. Como o leitor pode ver, os dois nós de índice **3** são isomórficos, pois $\boldsymbol{index(3) = index(3')}$, $\boldsymbol{\sigma(low(3)) = low(3')}$ e $\boldsymbol{\sigma(high(3)) = high(3')}$. Assim,

eliminamos um dos sub-grafos isomórficos, como $\mathbf{3'}$, e fazemos todas as arestas dos pais de $\mathbf{3'}$ apontarem agora para o sub-grafo isomórifico restante, como mostra a figura 3.2c.

Como podemos ver na figura 3.2c, o nó $(\mathbf{3,3})$ é um nó redundante, uma vez que independentemente do seu valor, ele terá por filho o nó $(\mathbf{1,2})$. Assim, eliminamos o nó $(\mathbf{3,3})$ do BDD e fazemos todas as arestas dos pais do nó $(\mathbf{3,3})$ apontarem para o filho do nó $(\mathbf{3,3})$, como mostra a figura 3.2d. Com isso, eliminamos as informações redundantes do BDD. É importante dizer que o grafo resultante possui algumas propriedades importantes. A definição a seguir visa estudar estas propriedades [16, 20].

DEFINIÇÃO: Se $\boldsymbol{G}$ é isomórfico a $\boldsymbol{G'}$ através de $\boldsymbol{\sigma}$, então para qualquer vértice $\boldsymbol{v}$ em $\boldsymbol{G}$, o sub-grafo cuja raiz é $\boldsymbol{v}$ é isomórfico ao sub-grafo cuja raiz é $\boldsymbol{\sigma(v)}$.

A definição acima diz basicamente que se dois grafos são isomórficos, então seus sub-grafos equivalentes também são isomórficos.

DEFINIÇÃO: Um grafo $\boldsymbol{G}$ é dito reduzido se ele não contém vértice $\boldsymbol{v}$ tal que $\boldsymbol{low(v) = high(v)}$, nem contém vértices distintos $\boldsymbol{v}$ e $\boldsymbol{v'}$ tal que os sub-grafos cujas raízes são $\boldsymbol{v}$ e $\boldsymbol{v'}$ são isomórficos.

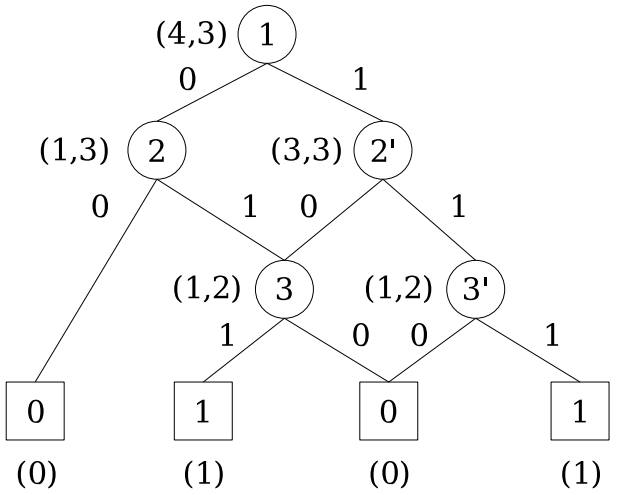
A definição acima formaliza o conceito de grafo reduzido. Esta é uma importante propriedade porque com ela podemos mostrar que a representação de um BDD é canônica, ou seja, existe um único BDD para um determinado conjunto de funções booleanas equivalentes.

DEFINIÇÃO: Para todo vértice $\boldsymbol{v}$ em um grafo reduzido de uma função, o sub-grafo cuja raiz é $\boldsymbol{v}$ é por si só um grafo de uma função reduzida.

A definição acima diz basicamente que se um grafo é reduzido, então todos os seus sub-grafos também são.

DEFINIÇÃO: Para qualquer função booleana $\boldsymbol{f}$, existe um único grafo reduzido desta função, e qualquer outro grafo representando $\boldsymbol{f}$ contém mais vértices.

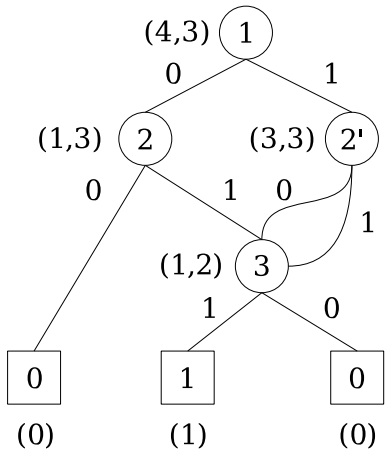
Enfim, chegamos a definição mais importante. Uma vez que sabemos que duas funções booleanas possuem o mesmo BDD, podemos chegar a conclusão que elas são equivalentes. Também podemos concluir a partir da definição acima que um grafo reduzido é uma representação compacta de uma função booleana.



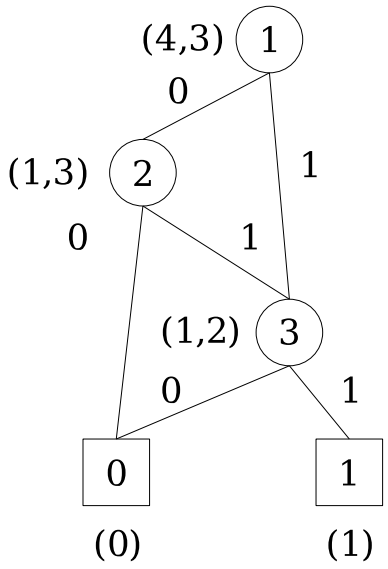
(a) BDD com informação redundante

Chave	**Índice**
(0)	1
(1)	2
(1,2)	3
(1,3)	4
(3,3)	3
(4,3)	6

(b) Chaves e índices dos nós



(c) Removendo sub-grafo isomórfico



(d) Removendo nó redundante

Figura 3.2: Simplificando um BDD [16, 20]

Operação	Resultado	Complexidade
Reduce	G é reduzido à forma canônica	$O(\|G\| \cdot log\|G\|)$
Apply	$f_1 < op > f_2$	$O(\|G_1\| \cdot \|G_2\|)$
Restrict	$f \mid_{x_i=b}$	$O(\|G\| \cdot log\|G\|)$
Compose	$f_1 \mid_{x_i=f_2}$	$O(\|G_1\|^2 \cdot \|G_2\|)$
Satisfy-one	Um cubo de f	$O(n)$
Satisfy-all	Todos os cubos de f	$O(n \cdot \|S_f\|)$
Satisfy-count	$\|S_f\|$	$O(G)$

Tabela 3.1: Operações

3.3.2 Operações

São apresentadas resumidamente na tabela 3.1. Se a função booleana possui n variáveis, então esta função pode ser representada como um sub-set do espaço booleano de dimensão n, pelos valores que fazem com que f venha a possuir valor 1. Este conjunto pode ser representado por

$$S_f = (x_1, \cdots, x_n) | f(x_1, \cdots, x_n) = 1$$

Reduction

O algoritmo ***reduction*** transforma um grafo de uma função arbitrária em um grafo reduzido que representa a mesma função.

Apply

Fornece um método básico para a criação da representação de uma função de acordo com os operadores na expressão booleana ou no circuito lógico. Explicando, se você conhece duas funções booleanas f_1 e f_2 com BDDs G_1 e G_2, respectivamente, então você poderá obter o BDD da função $f_1 \wedge f_2$ chamando a função *apply*, passando como parâmetro os BDDs G_1 e G_2.

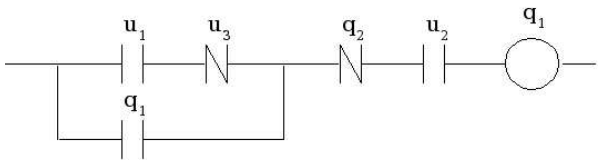


Figura 3.3: Uma malha de um diagrama *ladder*

Restriction

Este algoritmo transforma o BDD que representa a função $\boldsymbol{f}$ em um BDD representando a função $\boldsymbol{f}\mid_{\boldsymbol{x_i=b}}$, para valores $\boldsymbol{i}$ e $\boldsymbol{b}$ dados, onde $\boldsymbol{x_i}$ é uma variável da função $\boldsymbol{f}$ e $\boldsymbol{b}$ é um valor booleano.

Composition

Constrói o grafo de uma função obtida pela composição de duas outras funções. Esta função é basicamente o inverso da função *restriction*. Como mostra a tabela 3.1, dadas duas funções $\boldsymbol{f_1}$ e $\boldsymbol{f_2}$, calculamos $\boldsymbol{f_1}\mid_{\boldsymbol{x_i=f_2}}$.

Satisfy

Considere um cubo um vetor do espaço $\boldsymbol{\{0,1\}^n}$ que faz com que a função $\boldsymbol{f}$ seja verdadeira. As funções $\boldsymbol{satisfy}$ são utilizadas para descobrir o número de elementos de $\boldsymbol{S_f}$, a lista dos elementos de $\boldsymbol{S_f}$ ou apenas um elemento de $\boldsymbol{S_f}$.

3.4 Árvores

Para manipularmos um diagrama *ladder*, precisamos de uma estrutura de dados adequada que o represente, capaz de proporcionar uma implementação eficiente, como a árvore gramatical. Semelhantemente às linguagens de programação, o diagrama *ladder* possui uma gramática que pode ser utilizada em seu processo de compilação. Assim, vamos neste trabalho tratar mais especificamente sobre as gramáticas, suas árvores sintáticas, bem como da análise destas árvores.

Uma gramática descreve a estrutura hierárquica de muitas construções das linguagens de programação [41]. Seja o exemplo da figura 3.3. Ou seja, a malha do diagrama *ladder* é uma ligação em série da bobina $\boldsymbol{q_1}$ com o contato normalmente aberto $\boldsymbol{u_2}$, com o contato normalmente fechado $\boldsymbol{q_2}$ e com uma ligação em paralelo formada por duas ligações, uma ligação em série formada pelo contato normalmente fechado $\boldsymbol{u_3}$ e pelo contato normalmente aberto $\boldsymbol{u_1}$, e pela ligação em série formada apenas pelo contato normalmente aberto $\boldsymbol{q_1}$. Uma vez que ligações em paralelo representam um ***or*** lógico e ligações em série representam um ***and*** lógico, podemos dizer seguramente que uma malha do diagrama *ladder* é formada por uma bobina e uma ligação em série, representada por um ***and*** lógico, que por sua vez pode ser formado por vários contatos $\boldsymbol{NA}$, contatos $\boldsymbol{NF}$, e ligações em paralelo, que uma vez representadas por um ***or*** lógico, podem ser formadas por várias ligações em série. Usando-se a variável ***malha*** para denotar uma malha de um diagrama *ladder*, as variáveis ***or*** e ***and*** para denotar ligações em paralelo e em série, respectivamente, e as variáveis ***bobina***, $\boldsymbol{NA}$ e $\boldsymbol{NF}$ para representar a bobina, os contatos normalmente abertos e fechados, respectivamente, esta regra de construção pode ser expressa como

$$\boldsymbol{malha \rightarrow (bobina, and)}$$

$$\boldsymbol{and \rightarrow (NA \mid NF \mid or)*}$$

$$\boldsymbol{or \rightarrow (and)*}$$

A regra acima é uma produção [41]. Nela, os elementos léxicos tais como $\boldsymbol{NA}$ e $\boldsymbol{NF}$ são chamados de tokens. As variáveis ***and, malha*** e ***or*** que representam seqüências de tokens são chamadas de variáveis não-terminais.

Uma gramática livre de contexto possui quatro componentes [41]:

1. Um conjunto de *tokens*, conhecidos como *símbolos terminais*;
2. um conjunto de *não-terminais*;
3. um conjunto de produções, onde uma produção consiste em um não-terminal, no *lado esquerdo* da produção, uma seta e uma sequência de *tokens* e/ou não-terminais, no *lado direito* da produção;
4. uma designação a um dos não-terminais como o *símbolo de partida*.

Uma árvore gramatical mostra, pictoricamente, como o símbolo de partida de uma gramática deriva uma cadeia de linguagem. Se um não-terminal $\boldsymbol{A}$ possui uma produção

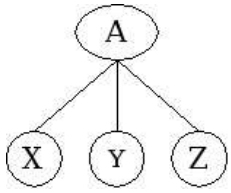


Figura 3.4: Uma pequena árvore gramatical

$\boldsymbol{A \rightarrow XYZ}$, então uma árvore gramatical pode ter um nó interior rotulado $\boldsymbol{A}$, com três filhos rotulados $\boldsymbol{X, Y}$ e $\boldsymbol{Z}$, da esquerda para a direita, como mostra a figura 3.4.

Formalmente, dada uma gramática livre de contexto, uma árvore gramatical possui as seguintes propriedades:

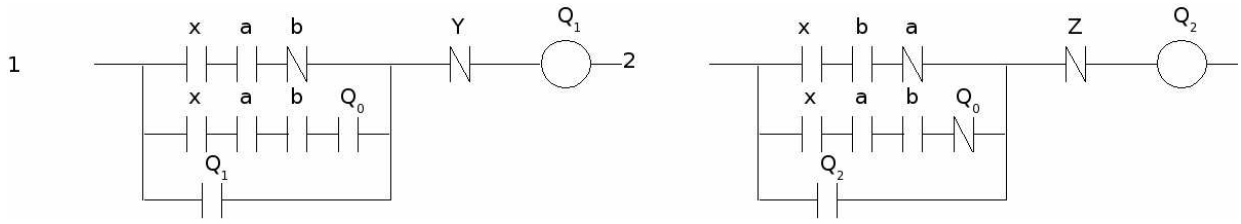
1. A raiz é rotulada pelo símbolo de partida;
2. cada folha é rotulada por um token e/ou por $\boldsymbol{\epsilon}$;
3. cada nó interior é rotulado por um não-terminal;
4. se $\boldsymbol{A}$ é um não-terminal rotulando algum nó interior e $\boldsymbol{X_1, X_2, \cdots, X_n}$ são os rótulos dos filhos daquele nó, da esquerda para a direita, então $\boldsymbol{A \rightarrow X_1, X_2, \cdots, X_n}$ é uma produção. Aqui, $\boldsymbol{X_1, X_2, \cdots, X_n}$ figuram no lugar de símbolos que sejam terminais ou não-terminais. Como um caso especial, se $\boldsymbol{A \rightarrow \epsilon}$, então um nó rotulado $\boldsymbol{A}$ deve possuir um único filho rotulado $\boldsymbol{\epsilon}$.

Sejam as malhas $\mathbf{1, 2}$ e $\mathbf{3}$ da figura 3.5. Estas $\mathbf{3}$ malhas podem ser representadas por árvores, conforme mostra a figura 3.6. Os nós em tom-de-cinza representam que as respectivas variáveis estão negadas. As árvores da figura 3.6 são conhecidas como árvores gramaticais. Nelas, a raíz é rotulada ***malha***, que também é o símbolo de partida da gramática citada anteriormente. Os filhos da raiz são rotulados, da esquerda para a direita, ***bobina*** e ***and***.

O processo de encontrar uma árvore gramatical para uma dada cadeia de tokens é chamado de *análise gramatical* ou *análise sintática* daquela cadeia.

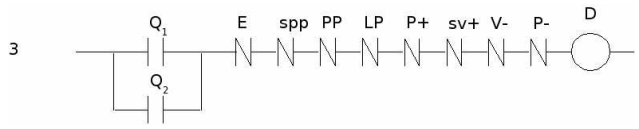
3.4.1 Caminhamento em profundidade

Um caminhamento de uma árvore se inicia à raiz e visita cada nó da mesma em alguma ordem, como por exemplo em profundidade [41]. As regras semânticas a um dado



(a) Direita até a esteira A

(b) Direita até a esteira B



(c) Direita

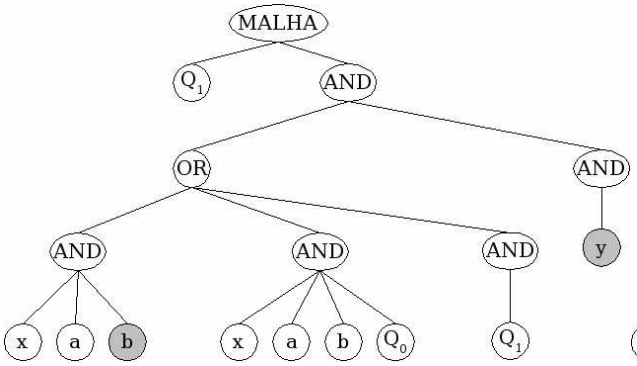
Figura 3.5: As três primeiras malhas da figura 2.32

nó serão avaliadas uma vez que todos os descendentes do mesmo já tenham sido visitados. É chamado de caminhamento em profundidade porque visita um filho não visitado de um nó sempre que o mesmo possua um e, então, tenta visitar nós tão distantes da raiz quanto possível, e mais rapidamente possível.

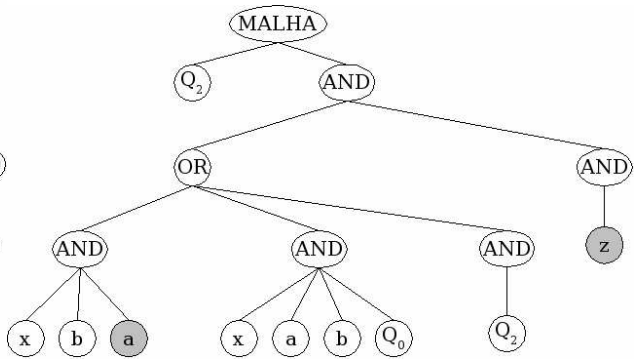
Algorithm 1 Caminhamento em profundidade

```
Procedimento visitar(n:nó);
inicio
    para cada filho m de n, da esquerda para a direita, faça
        visitar(m);
    avaliar as regras semânticas ao nó n
fim
```

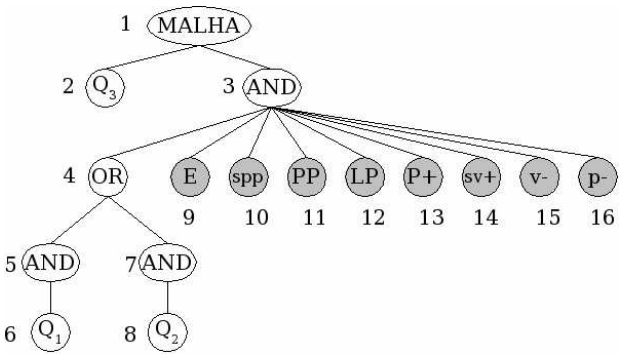
Conforme mostra a figura 3.6c, se visitarmos os nós desta árvore obedecendo o algoritmo em profundidade 1, a ordem com que os nós serão visitados será aquela apresentada pelos números que encontram-se próximos aos nós. Desta forma, podemos reconstruir a malha da figura 3.5. Inicialmente, visitamos o nó **1**. Neste momento, sabemos que estamos construindo uma malha, pois o nó chama-se $\boldsymbol{MALHA}$. Visitamos então o seu primeiro filho, $\boldsymbol{Q_3}$, descobrindo assim o endereço da mesma. A seguir, visitamos o nó **3**, descobrindo que estamos prestes a começar uma ligação em série. Visitanto o nó **4** descobrimos que o primeiro elemento desta ligação em série é uma ligação em paralelo.



(a) Árvore da malha 1

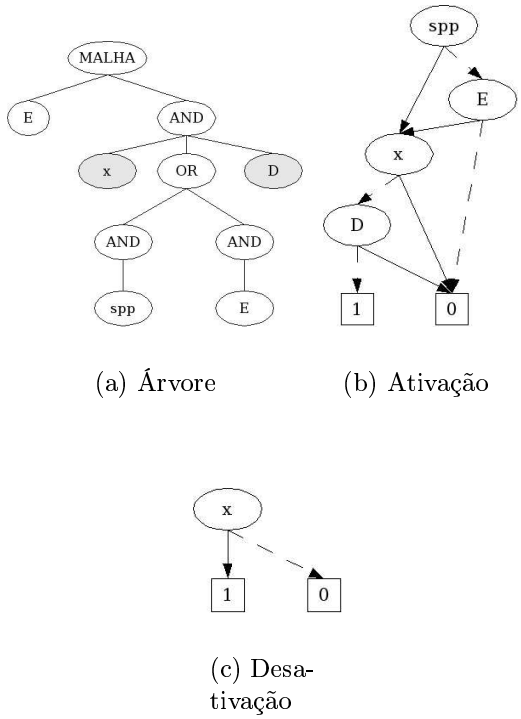


(b) Árvore da malha 2



(c) Árvore da malha 3

Figura 3.6: Árvores



(a) Árvore (b) Ativação

(c) Desativação

Figura 3.7: A árvore e seus BDDs

A seguir, visitamos o nó **5** e assim descobrimos o primeiro ramo desta ligação. No nó **6** ficamos sabendo que esta ligação em série é formada apenas por um elemento, $\boldsymbol{Q_1}$. Sabemos que na prática não existe ligação em série formada apenas por um elemento, mas adotamos esta notação porque ela simplifica muito a gramática utilizada, e consequentemente a construção da árvore sintática e por conseguinte a construção do software. Visitanto os nós **7** e **8** descobrimos o segundo ramo desta ligação em paralelo. A partir do nó **9** terminamos de construir a ligação em série.

3.5 Usando os Diagramas Binários de Decisão

Primeiramente, devemos encontrar as expressões de ativação e desativação de cada uma das malhas. Como conhecemos as árvores sintáticas de cada uma das malhas, podemos utilizá-las para encontrarmos as expressões de ativação destas malhas construindo variáveis nos BDDs a partir das folhas destas árvores sintáticas. A seguir, percorrendo estas árvores recursivamente de baixo para cima, para cada nó ***and*** e ***or*** encontrados construimos um novo BDD formado por um ***and*** e por um ***or*** dos BDDs das sub-árvores,

respectivamente. Seguindo estas regras, a árvore da figura 3.7a foi convertida no BDD de ativação da figura 3.7b.

Encontrada as expressões de ativação destas malhas, para encontrarmos as expressões de desativação das malhas atribuímos inicialmente valor lógico **1** para os selos dessas expressões utilizando a operação *restrict* dos BDDs, criando assim um BDD intermediário chamado $\boldsymbol{BDD_{temp}}$. As expressões de desativação são criadas aplicando a operação *not* dos BDDs sobre o $\boldsymbol{BDD_{temp}}$. Desta forma, com o BDD de ativação da figura 3.7b encontramos o BDD de desativação da figura 3.7c.

Uma vez encontradas as expressões de ativação de cada uma das malhas, com o objetivo de removermos os intertravamentos de uma malha de um diagrama *ladder*, precisamos obter uma nova malha a partir da anterior atribuindo às variáveis que implementam os intertravamentos valor lógico **0**. Isto pode ser feito diretamente através da operação *restrict* dos BDDs [16, 20].

Uma vez removidos os intertravamentos, o próximo passo consiste em detectar através do diagrama *ladder* as transições que podem acontecer na planta. Assim, dado um possível seqüenciamento entre uma malha $\boldsymbol{i}$ e uma malha $\boldsymbol{j}$, devemos verificar se existe um cubo ou sub-cubo da malha $\boldsymbol{i}$ que seja cubo ou sub-cubo da malha $\boldsymbol{j}$. Para isto, basta verificarmos se há um caminho ou sub-caminho no BDD da malha $\boldsymbol{i}$ que seja caminho ou sub-caminho da malha $\boldsymbol{j}$.

Detectados os seqüenciamentos, o próximo passo consiste em construirmos as expressões booleanas das transições do diagrama de estados e do diagrama *grafcet*. Para criarmos as transições do diagrama de estados e do diagrama *grafcet* precisamos extrair uma função booleana a partir de um BDD, o que pode ser feito efetuando a soma lógica de todos os seus cubos. Para isto, é necessário encontrar todos os caminhos que existem entre o nó raiz do BDD e o nó terminal de valor **1** [16, 20].

Durante a análise do seqüenciamento podemos encontrar mais do que uma transição saindo de uma mesma malha. Neste caso, deveremos verificar se existem duas ou mais transições que sejam iguais ou tenham alguma subfunção em comum, pois isto poderá indicar a presença de paralelismo. Podemos verificar se há uma subfunção comum a duas outras funções primeiramente encontrando as variáveis que sejam comuns a ambas as funções fazendo uso dos BDDs das mesmas. Realizando *backtracking* apenas sobre estas variáveis, ao invés de todas as variáveis das duas funções sob análise, tornamos o algoritmo bem mais eficiente. Com este *backtracing* conseguimos analisar todos os cubos que são comuns a ambas as funções. A subfunção que é comum a ambas as funções é dada pela soma destes cubos.

3.6 Análise da eficiência

Como vimos, a complexidade dos passos **3**, **4** e **6** do método proposto para diagramas *grafcet* e de estados são, respectivamente, $\boldsymbol{O(n) \cdot k_2}$, $\boldsymbol{k_1 \cdot O(n^2)}$ e $\boldsymbol{O(n) \cdot k_3}$. $\boldsymbol{k_1}$ é a ordem do tempo gasto para consultarmos se uma variável pertence a um BDD, sendo da ordem de $\boldsymbol{O(n_2)}$, onde $\boldsymbol{n_2}$ é o número de variáveis do BDD [16, 20], mais a ordem do tempo gasto para eliminarmos o intertravamento, $\boldsymbol{O(|G| \cdot log|G|)}$, onde $\boldsymbol{|G|}$ é o número de vértices do BDD. Resumindo, temos para a complexidade do passo **3** uma ordem de $\boldsymbol{\{O(|G| \cdot log|G|) + O(n_2)\} \cdot O(n^2)}$. Na prática, esta ordem de complexidade não é crítica, uma vez que o tamanho dos BDDs a serem manipulados é pequeno, fazendo com que a complexidade $\boldsymbol{O(n^2)}$ se sobressaia sobre as demais, a qual é satisfatória até mesmo para grandes plantas industriais.

Para o exemplo utilizado neste artigo, $\boldsymbol{k_2}$ é a ordem do tempo gasto para calcularmos o BDD de ativação e desativação de uma malha, utilizando operações *apply* e *restrict*. A função *apply* possui complexidade $\boldsymbol{O(|G_1| \cdot |G_2|)}$, onde $\boldsymbol{|G_1|}$ e $\boldsymbol{|G_2|}$ representam o número de nós dos BDDs **1** e **2**, respectivamente [16, 20]. A função *restrict* possui complexidade $\boldsymbol{O(|G| \cdot log|G|)}$ [16, 20], mais uma operação *not*, de complexidade $\boldsymbol{O(|G|)}$. Assim, temos que a complexidade computacional do passo **4** possui ordem de $\boldsymbol{O(n) \cdot \{O(|G| \cdot log|G| + O(|G|) + O(|G_1| \cdot |G_2|)\}}$. Na prática, esta ordem de complexidade também não é crítica pelos mesmos motivos do passo **3**.

Para o passo **6**, $\boldsymbol{k_3}$ é a ordem do tempo gasto para verificarmos se um cubo ou sub-cubo do BDD de desativação da malha $\boldsymbol{i}$ é cubo ou sub-cubo do BDD de ativação da malha $\boldsymbol{j}$. Para isto, encontramos as variáveis que são comuns entre estes dois BDDs, com complexidade $\boldsymbol{O(n_2 \cdot n_3)}$, onde $\boldsymbol{n_2}$ e $\boldsymbol{n_3}$ são o número de variáveis dos BDDs de desativação e ativação, respectivamente, mais a ordem do tempo de casamento entre cada um destes cubos, $\boldsymbol{O(n_4 \cdot log(n_4) \cdot \{|S_{f_1}| + |S_{f_2}|\})}$, onde $\boldsymbol{n_4}$ é o número de variáveis comuns e $\boldsymbol{|S_{f_1}|}$ e $\boldsymbol{|S_{f_2}|}$ são o número de cubos dos BDDs de desativação e ativação, respectivamente [16, 20]. Utilizamos *backtracking* neste algoritmo para aumentar a sua eficiência. Assim, temos que a complexidade computacional do passo **6** possui ordem de $\boldsymbol{O(n) \cdot \{O(n_2 \cdot n_3) + O(n_4 \cdot log(n_4) \cdot \{|S_{f_1}| + |S_{f_2}|\})\}}$. Na prática, esta complexidade também não é crítica, pelos mesmos motivos do passo **3**.

3.7 Conclusão

Utilizando os BDDs é possível efetuar todas as manipulações algébricas sobre as expressões booleanas necessárias neste trabalho com eficiência.

Capítulo 4

O Protótipo

Neste capítulo descrevemos a construção de um protótipo com o intuito de validarmos o método proposto utilizando as técnicas apresentadas nos capítulos anteriores.

4.1 Especificação

Dado um arquivo de entrada no formato XML contendo as informações do diagrama *ladder*, o protótipo deve ser capaz de apresentar graficamente este diagrama ao usuário para que o mesmo possa ser confirmado e, caso necessário, revisado. O protótipo também deve ser capaz de utilizar o método de extração de diagramas *grafcet* e de estados a partir de diagramas *ladder* proposto neste trabalho para analisar o diagrama *ladder* e convertê-lo no diagrama *grafcet* e de estados de forma interativa, utilizando para isto uma interface gráfica amigável. Os diagramas *grafcet* e de estados resultantes do processo de conversão devem ser graficamente apresentados ao usuário.

4.2 Arquitetura

O sistema possui três grandes módulos: o módulo de entrada, o módulo de conversão e o módulo de visualização, conforme podemos ver na figura 4.1. Nela, o módulo *extrator de diagramas de estados a partir de diagramas grafcet sem paralelismo* pode ser considerado um módulo auxiliar, devido a baixa complexidade causada pela semelhança entre o diagrama *grafcet* e o diagrama de estados.

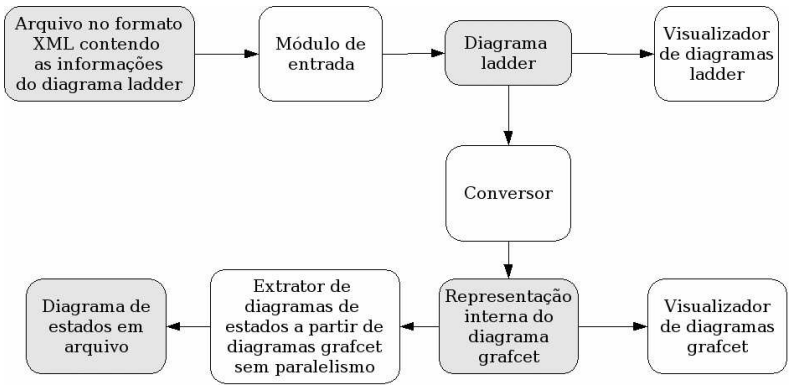


Figura 4.1: Principais módulos do sistema

O módulo de entrada recebe um arquivo no formato XML contendo as informações do diagrama *ladder*. Este arquivo é carregado através de um parser de arquivos XML pertencente a biblioteca *Xerces-J*, implementada com a linguagem *java*, resultando em uma árvore DOM, conforme especificação da W3C (entidade encarregada de padronizar a linguagem XML). A árvore DOM, cujo nome vem do inglês *Document Object Model*, é uma representação em árvore dos elementos do arquivo XML. A seguir, é feita a extração dos endereços de entrada, saída e internos do diagrama *ladder*, das árvores sintáticas de suas malhas e de suas informações gráficas, conforme podemos ver na figura 4.2. Estas informações são representadas pelo sistema usando objetos.

Como já existe um parser disponível, não precisamos desenvolver um programa que faça a análise léxica e sintática do arquivo de entrada, reduzindo assim o esforço para desenvolver o protótipo. Outra vantagem de utilização da linguagem XML é a possibilidade de padronização do arquivo de entrada que ela proporciona, pois atualmente cada fabricante de PLC possui um formato de arquivo para diagramas *ladder*. Há uma descrição do formato do arquivo de entrada no apêndice.

As informações resultantes do módulo de entrada são repassadas para o módulo de visualização e para o módulo conversor de diagramas *ladder* em diagramas *grafcet*. O módulo de visualização de diagramas *ladder* exibe o diagrama na tela do computador utilizando a biblioteca gráfica *Draw2D*, desenvolvida pela IBM como um dos plug-ins que podem ser encontrados na plataforma *Eclipse*, um ambiente gratuito de desenvolvimento de software doado a comunidade de desenvolvimento de software de código aberto.

Uma vez apresentado o diagrama *ladder* ao usuário, este diagrama é processado pelo módulo de conversão de diagramas *ladder* em diagramas *grafcet*, conforme pode ser visto na figura 4.3. Inicialmente, é feito um pré-processamento sobre as árvores sintáticas

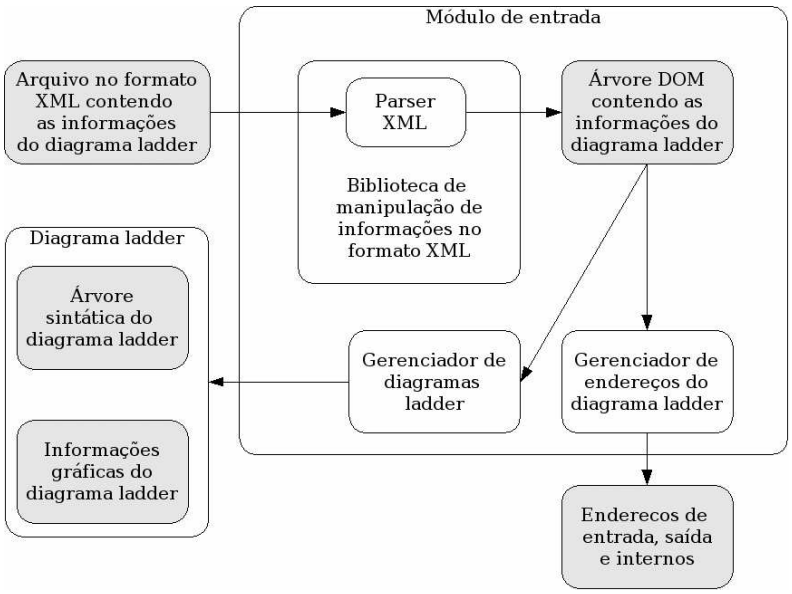


Figura 4.2: Módulo de entrada

do diagrama *ladder* objetivando encontrar os BDDs de ativação e desativação de cada malha do diagrama *ladder*, os intertravamentos, os passos e as ações do diagrama *grafcet*, que são representados na figura 4.3 pelo bloco chamado *informações intermediárias*. A seguir é criada definitivamente a estrutura interna do diagrama *grafcet* utilizando estas informações.

Feita a conversão, o módulo de visualização faz a apresentação do diagrama *grafcet* e do diagrama de estados para o usuário, conforme podemos ver na figura 4.4. Neste módulo é feita uma chamada a uma ferramenta externa ao protótipo, encarregada de fazer a geração de layout de grafos, chamada *neato*. De posse das informações de layout do diagrama *grafcet*, e de suas estruturas internas, é feita a visualização deste diagrama utilizando a biblioteca gráfica *Draw2D*. Ao contrário do diagrama *grafcet*, não desenvolvemos um visualizador de diagramas de estados, mas utilizamos uma ferramenta externa com este fim, chamada *ZGRViewer*, pois além de já estar acessível, é também de fácil utilização.

Com a construção do protótipo foi possível representar o diagrama *ladder* como uma lista de árvores sintáticas relativas as malhas deste diagrama. Navegando recursivamente por estas árvores, podemos construir as expressões de ativação e desativação das malhas do diagrama *ladder* usando BDDs. Como partimos do princípio que cada malha representa no máximo um estado, restou apenas usar as expressões de ativação e desativação para descobrirmos quais são os eventos que podem ocorrer na planta causando a transição entre estes estados. Buscando descobrir uma técnica que pudesse ser utilizada sistema-

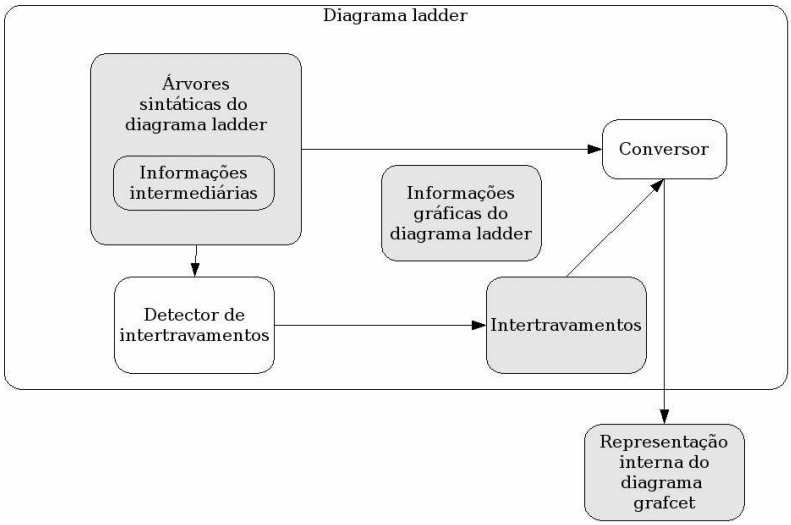


Figura 4.3: Módulo de conversão

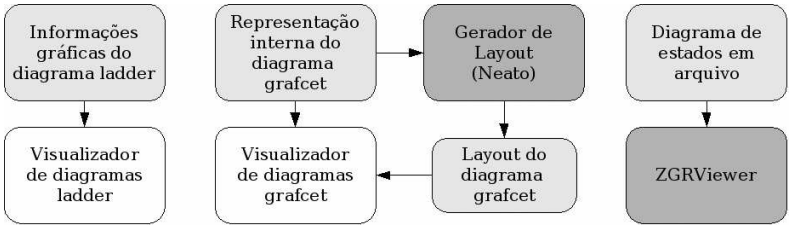


Figura 4.4: Módulo de visualização

ticamente na detecção destes eventos, analizamos uma grande variedade de plantas que foram automatizadas usando o diagrama *ladder*. Usamos para isto as plantas as quais encontramos nos trabalhos de Silveira [6], Georgini [7] e Natale [8]. Constatamos que podemos detectar com segurança todos os eventos possíveis, mas não podemos descobrir sistematicamente quais são os eventos que realmente acontecem na planta, bem como separá-los dos eventos que nunca acontecem na planta. Isto acontece porque não há informação suficiente no diagrama *ladder* para fazermos esta separação. Alguns trabalhos na literatura optaram por acrescentar informação extra ao processo sob a forma de lógica temporal, fazendo assim a modelagem da planta, como foi o caso de Zanma et al [3]. Outros acrescentaram esta informação extra sob a forma de gráficos de simultaneidade, como foi o caso de Falcione e Krog [2]. Diversos outros trabalhos descreveram métodos de conversão entre o diagrama *ladder* e algum outro tipo de diagrama, podendo o mesmo ser uma Rede de Petri, como por exemplo Nakamura et al [10] e Lee et al [14]. Assim, tive-

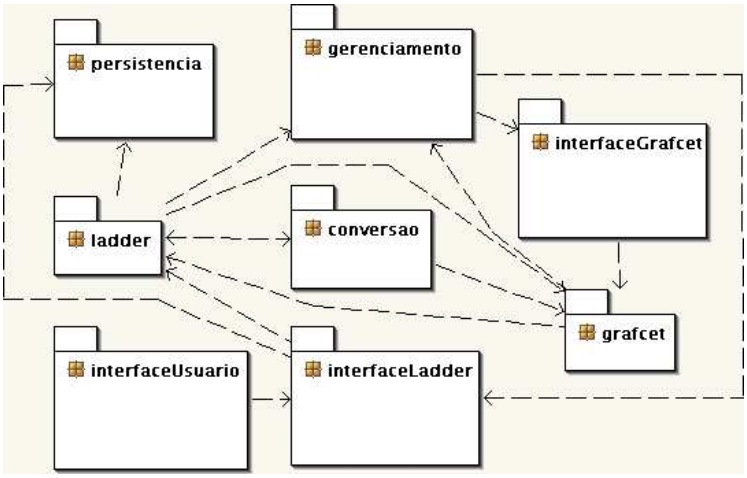


Figura 4.5: Módulos do Sistema

mos que pensar em um método de detecção destes eventos que fosse fácil de ser utilizado por um programador de PLC. Optamos pelo método interativo, onde todos os eventos possíveis são apresentados ao usuário, quando o mesmo apenas faz a separação daqueles que realmente acontecem na planta daqueles que nunca acontecem. Utilizamos para isso uma interface gráfica amigável, onde tudo o que o usuário precisa fazer é pressionar alguns botões confirmando ou não o seqüenciamento entre os estados do diagrama de estados ou entre os passos do diagrama *grafcet*. Com isso o esforço necessário por parte do usuário em recuperar um diagrama de estados ou um diagrama *grafcet* a partir de um diagrama *ladder* acaba sendo muito menor do que se utilizássemos qualquer outro método.

4.3 Detalhamento dos Módulos do Sistema

Os três grandes módulos do sistema, entrada, conversão e visualização, foram divididos em oito sub-módulos, conforme mostra a figura 4.5.

4.3.1 O módulo do diagrama ladder

Este é sem dúvida o principal módulo do sistema. Quem realmente realiza a extração de diagramas *grafcet* a partir de diagramas *ladder* é o próprio módulo do diagrama *ladder*, utilizando para isto o módulo de conversão, bem como os módulos de gerenciamento e o

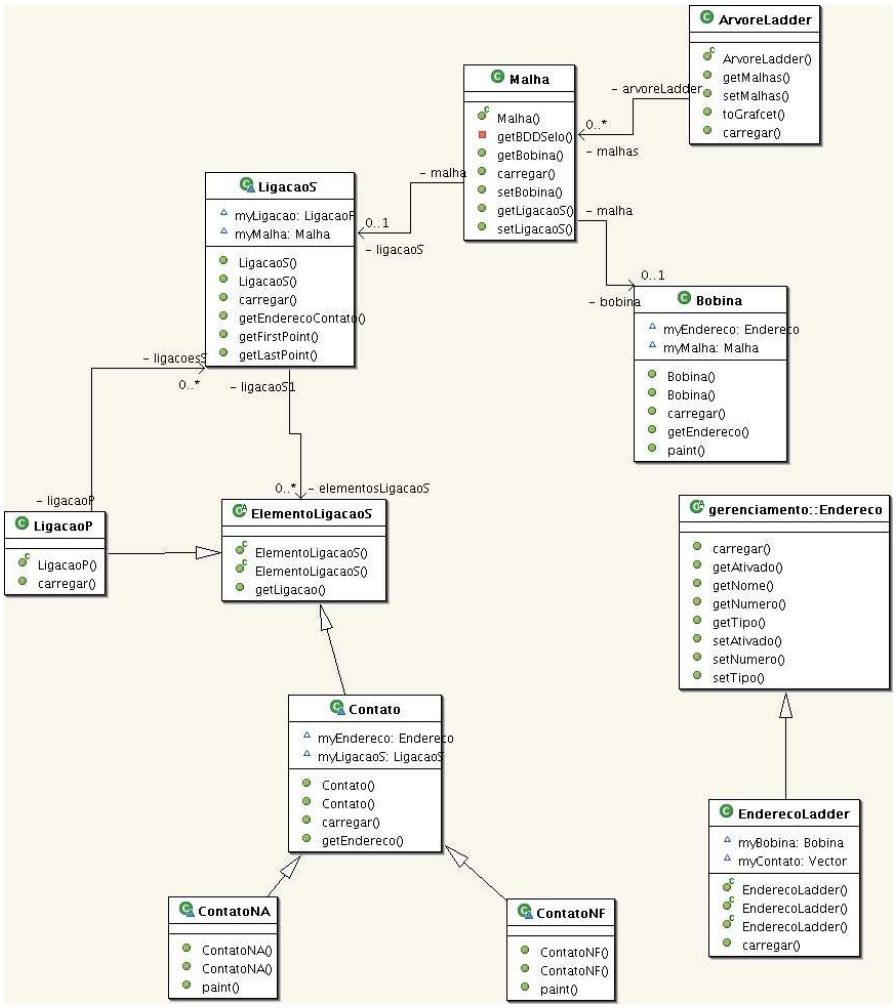


Figura 4.6: Diagrama de classes do módulo do diagrama *ladder*

módulo chamado *grafcet*, o qual implementa o diagrama *grafcet*. A figura 4.6 apresenta o diagrama de classes deste sistema. São ao todo 11 classes e interfaces.

4.3.2 O módulo de persistência

Este módulo é responsável em auxiliar a recuperação do diagrama *ladder*, e sua respectiva representação gráfica, a partir de um arquivo XML. Em outras palavras, o usuário deve informar ao sistema um arquivo XML contendo a representação de um diagrama *ladder* no formato XML. É assim que é feita a edição do diagrama *ladder* usando o sistema. São ao todo 3 classes e interfaces.

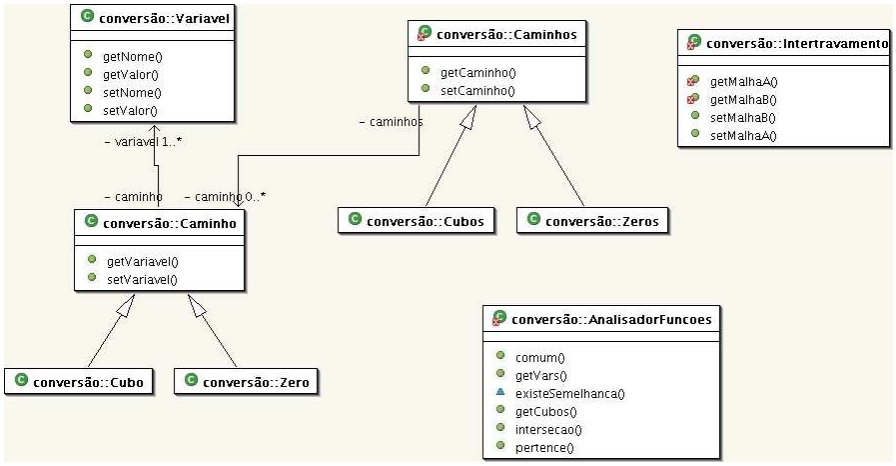


Figura 4.7: O Módulo de Conversão

4.3.3 O módulo de interface do diagrama ladder

Este módulo implementa a representação gráfica do diagrama *ladder*, isto é, enquanto o módulo do diagrama *ladder* armazena informações lógicas sobre o diagrama *ladder*, o módulo de interface do diagrama *ladder* armazena informações gráficas sobre o diagrama *ladder*. São ao todo 12 classes e interfaces.

4.3.4 O módulo de interface com o usuário

Neste módulo estão implementadas as janelas que fazem a interface com o usuário de forma interativa. São ao todo 5 classes e interfaces.

4.3.5 O módulo de conversão

Neste módulo estão implementadas as classes que estão diretamente ligadas ao processo de conversão do diagrama *ladder* para o diagrama de estados, bem como para o diagrama *grafcet*. Ele também é o módulo que mais utiliza o BDD, como mostra a figura 4.7. São ao todo 11 classes e interfaces.

4.3.6 O módulo de gerenciamento

Frequentemente é necessário saber se um determinado objeto já existe. Para que isto seja feito foram criadas classes pertencentes ao módulo gerenciamento, cujo objetivo delas é armazenar e consultar sobre a existência de seus objetos. Assim, se possuímos um objeto da classe Bobina no módulo do diagrama *ladder*, então deverá existir uma classe GerenteBobinas, a qual deverá ser capaz de consultar a existência de uma bobina, ou até mesmo retorná-la, caso ela já exista. Também faz parte deste módulo a classe Endereco, a qual é responsável por representar o endereço de um diagrama *ladder*. São ao todo 18 classes e interfaces.

4.3.7 O módulo do diagrama grafcet

Este é o módulo cujas classes representam o diagrama *grafcet*, conforme pode ser visto na figura 4.8. São ao todo 16 classes e interfaces, onde 13 foram criadas durante a análise e 3 são classes e interfaces de projeto, e por isso não aparecem na figura. Quase todas as classes deste módulo possuem um método de conversão do diagrama *grafcet* para o diagrama de estados. Não foi necessário criar novas classes para o diagrama de estados, e por isto não há um módulo com esta finalidade, pois devido a semelhança do diagrama *grafcet* com o diagrama de estados, toda vez que desejamos exibir o diagrama de estados, fazemos a sua extração a partir do diagrama *grafcet*. Utilizamos a biblioteca Grappa para exibir o diagrama de estados na tela.

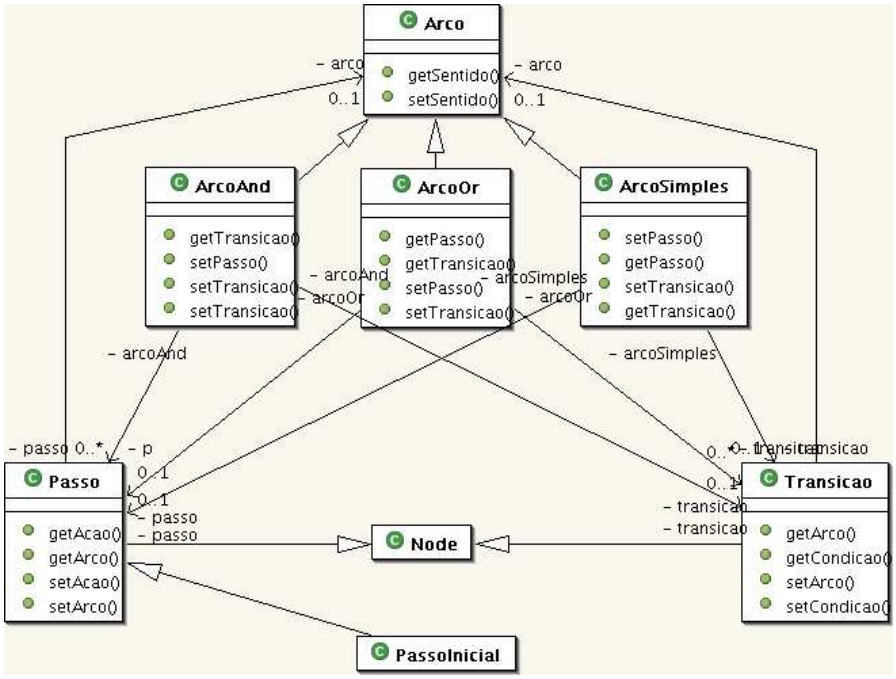
4.3.8 O módulo de interface do diagrama grafcet

Este módulo é responsável por exibir o diagrama *grafcet* na tela do computador. Ele faz uso intenso da biblioteca Draw2D. São ao todo 19 classes e interfaces.

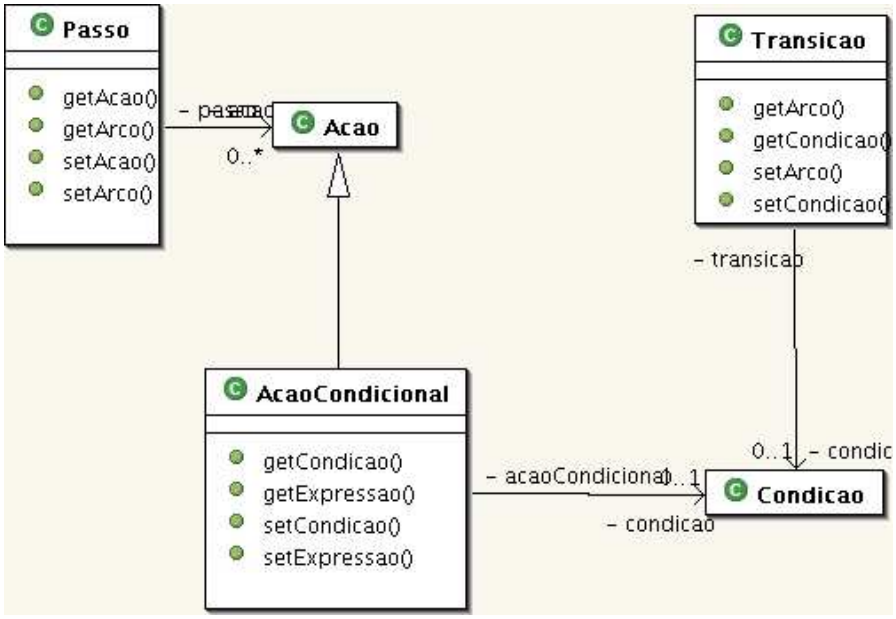
Se levarmos em consideração todos os módulos do sistema totalizamos exatamente 100 classes e interfaces, onde algumas delas chegam a possuir 30 métodos.

4.4 Um Exemplo

Vejamos um exemplo de utilização do protótipo. O usuário inicialmente deverá informar ao sistema o diagrama *ladder*. Isto é feito através da edição de um arquivo XML,



(a) Passos, transições e arcos



(b) Transições, condições e passos

Figura 4.8: O módulo do diagrama grafcet

```
<DiagramaLadder nome = "ladder">

    <ArvoreLadder>

        <Malha id = "1">
            <LigacaoS>
                <LigacaoP>
                    <LigacaoS>
                        <ContatoNA id = "1" idEndereco = "E0" />
                        <ContatoNA id = "2" idEndereco = "E1" />
                        <ContatoNF id = "3" idEndereco = "E2" />
                    </LigacaoS>
                    <LigacaoS>
                        <ContatoNA id = "4" idEndereco = "E0" />
                        <ContatoNA id = "5" idEndereco = "E1" />
                        <ContatoNA id = "6" idEndereco = "E2" />
                        <ContatoNA id = "7" idEndereco = "I0" />
                    </LigacaoS>
                    <LigacaoS>
                        <ContatoNA id = "102" idEndereco = "I1" />
                    </LigacaoS>
                </LigacaoP>
                <ContatoNF id = "8" idEndereco = "E4" />
            </LigacaoS>
```

Figura 4.9: Arquivo do diagrama *ladder* no formato XML

como mostra a figura 4.9. Após feita a edição do arquivo que armazena o diagrama *ladder*, o mesmo é carregado na memória do computador e exibido ao usuário, conforme mostra a figura 4.10.

A seguir, o diagrama *ladder* é carregado para a memória. Uma vez feito isto, o sistema faz a transformação das árvores que representam o diagrama *ladder* em BDDs. Com os BDDs é feito o cálculo das expressões de ativação e desativação de cada estado. Os BDDs que não possuírem selo nem sempre implementam estados. Assim, é necessário confirmar com o usuário se uma malha sem selo é um estado ou se a mesma é uma ação condicional, como mostra na figura 4.11.

Uma vez detectadas as ações condicionais, é necessário saber quais são os estados que vêem após o estado inicial. Como não há informação no diagrama *ladder* suficiente para tirarmos qualquer conclusão sobre o assunto, então uma saída é solicitar que o usuário informe os estados iniciais através da interface gráfica da figura 4.12.

Começamos então a fazer a detecção das possíveis transições de estados, bem como separar as transições que realmente acontecem na planta daquelas que nunca acontecerão. Para fazermos isto utilizamos uma interface gráfica igual a da figura 4.13. Assim, existirão casos em que o sistema detectará mais do que uma transição, mas nem todas acontecem realmente na planta, como pode ser visto na figura 4.14. Também existirão casos em que mais do que uma transição será confirmada, conforme pode ser visto na figura 4.15.

Uma vez confirmadas todas as transições, resta agora confirmar as ações condicionais.

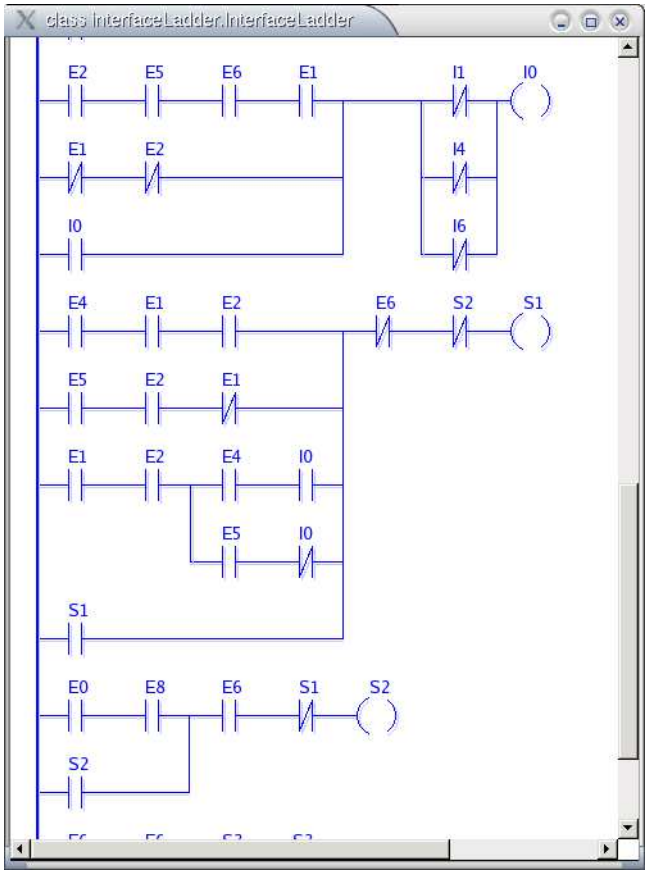


Figura 4.10: Interface do diagrama *ladder*

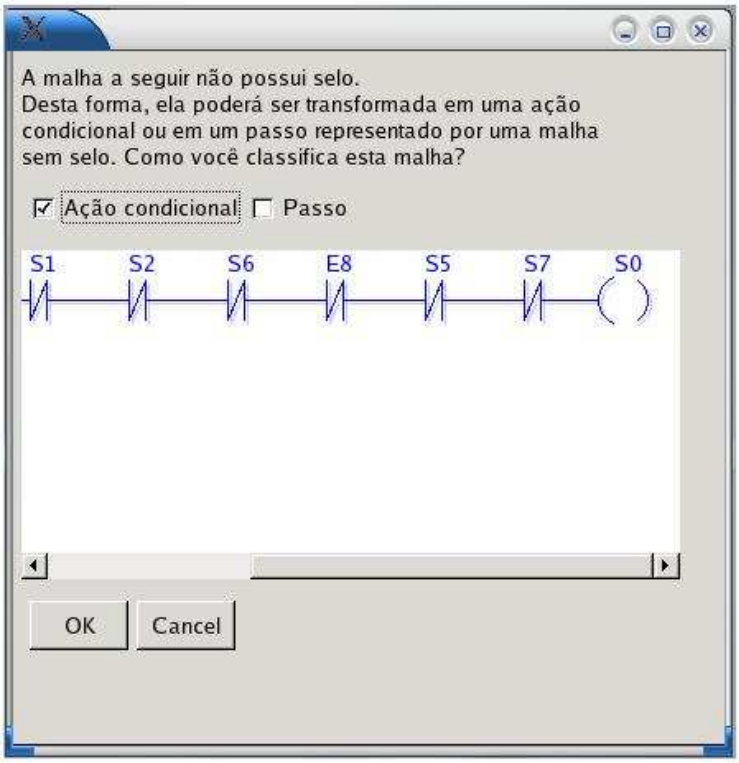


Figura 4.11: Confirmação de estado

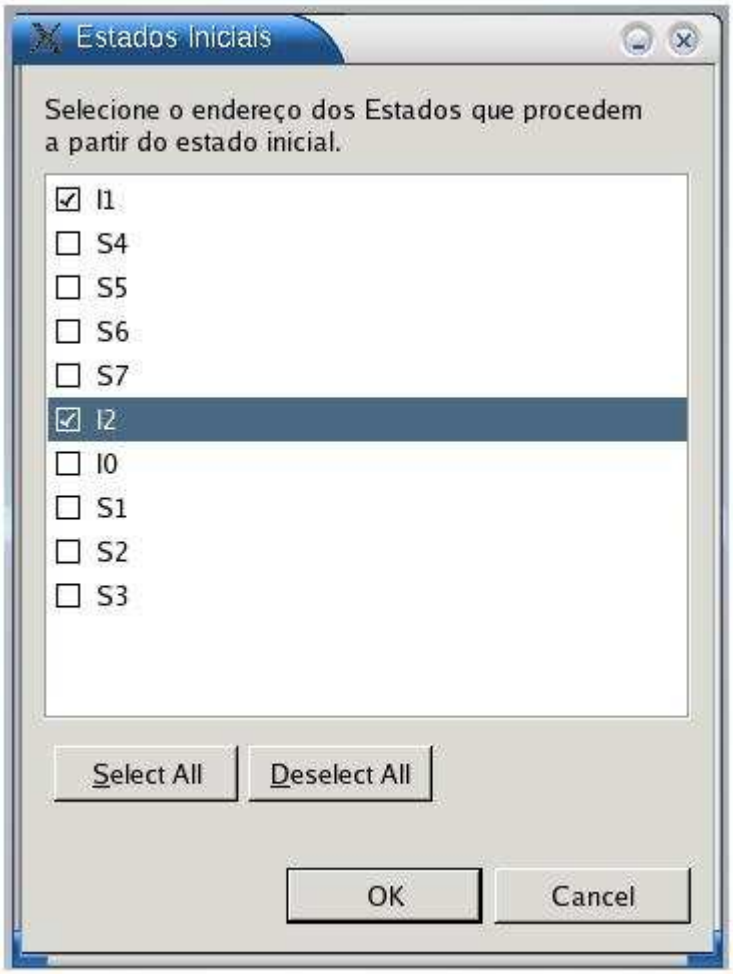


Figura 4.12: Estados iniciais

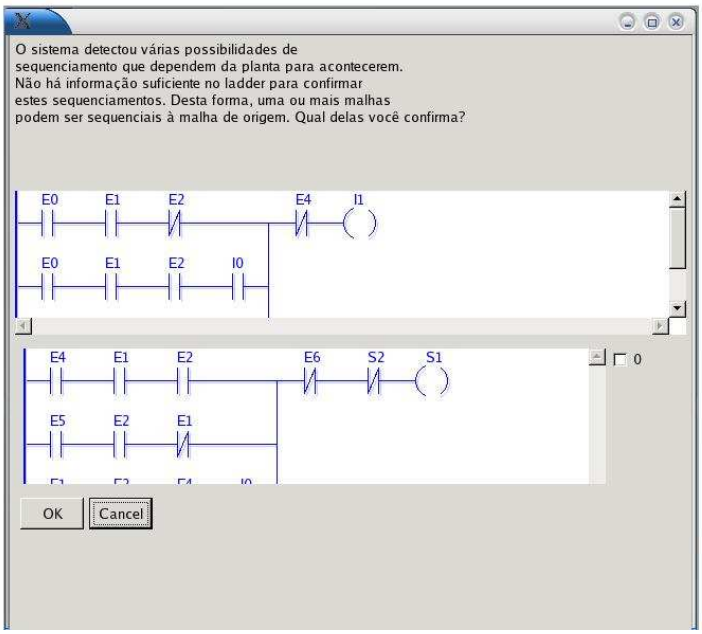


Figura 4.13: Confirmação de sequenciamento

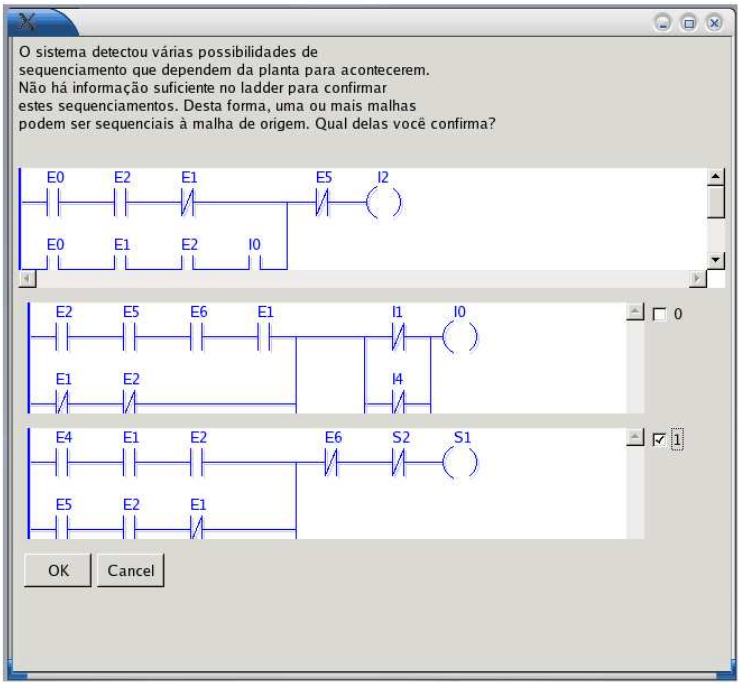


Figura 4.14: Nem todas as transições são confirmadas

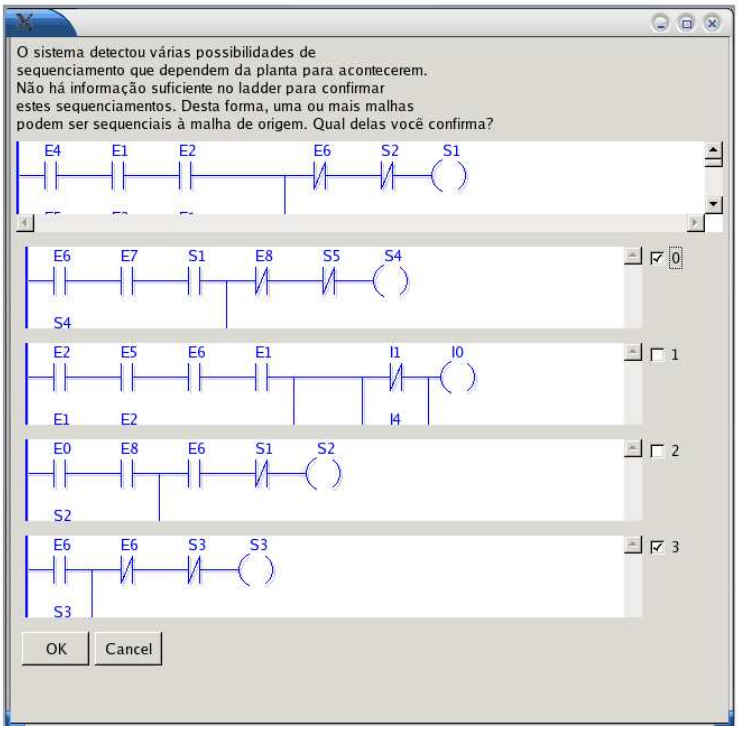


Figura 4.15: Mais do que uma transição

Podemos ver um exemplo de como isto é feito na figura 4.16. Nela, o usuário é questionado se a malha inferior é uma ação condicional da malha superior, uma vez que a malha inferior não possui selo e ainda utiliza um contato normalmente aberto com o endereço da bobina da malha superior.

Finalmente, o sistema imprime o diagrama de estados na tela do computador. Como o grafo gerado ficou muito grande, foi necessário visualizá-lo com uma outra ferramenta: o ZGR Viewer. O resultado pode ser visto na figura 4.17. O sistema também forneceu o diagrama *grafcet*. Não utilizamos a mesma planta para ilustrarmos a apresentação do diagrama de estados e do diagrama *grafcet*, pois apresentamos um diagrama grafcet com paralelismo, e como esta é uma capacidade que os diagramas de estado não representa explicitamente, precisamos utilizar outro exemplo para o diagrama de estados. O diagrama *grafcet* pode ser visto na figura 4.18.

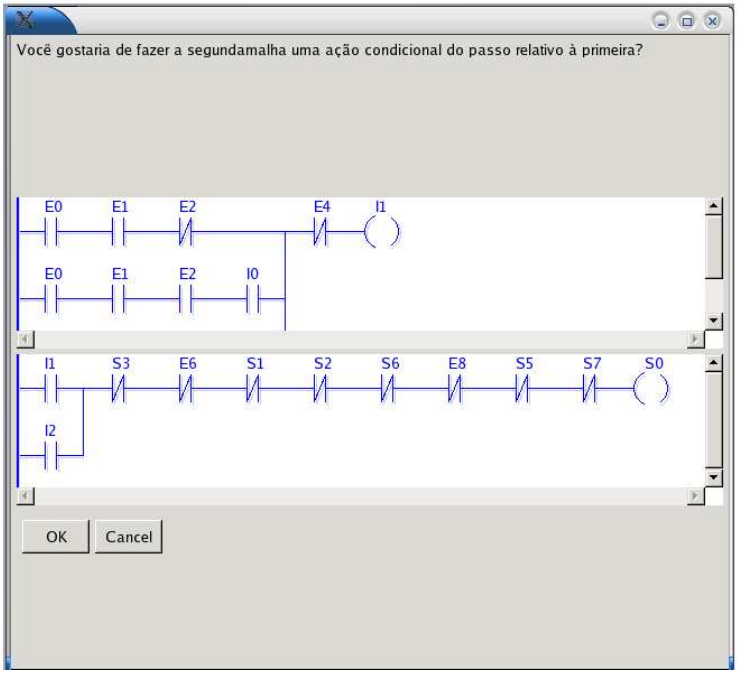


Figura 4.16: Confirmando a ação condicional

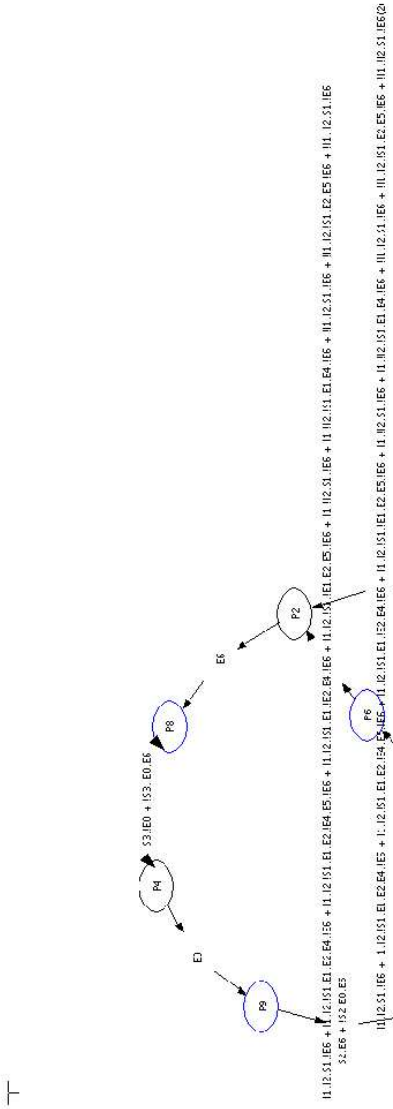


Figura 4.17: Diagrama de estados resultante

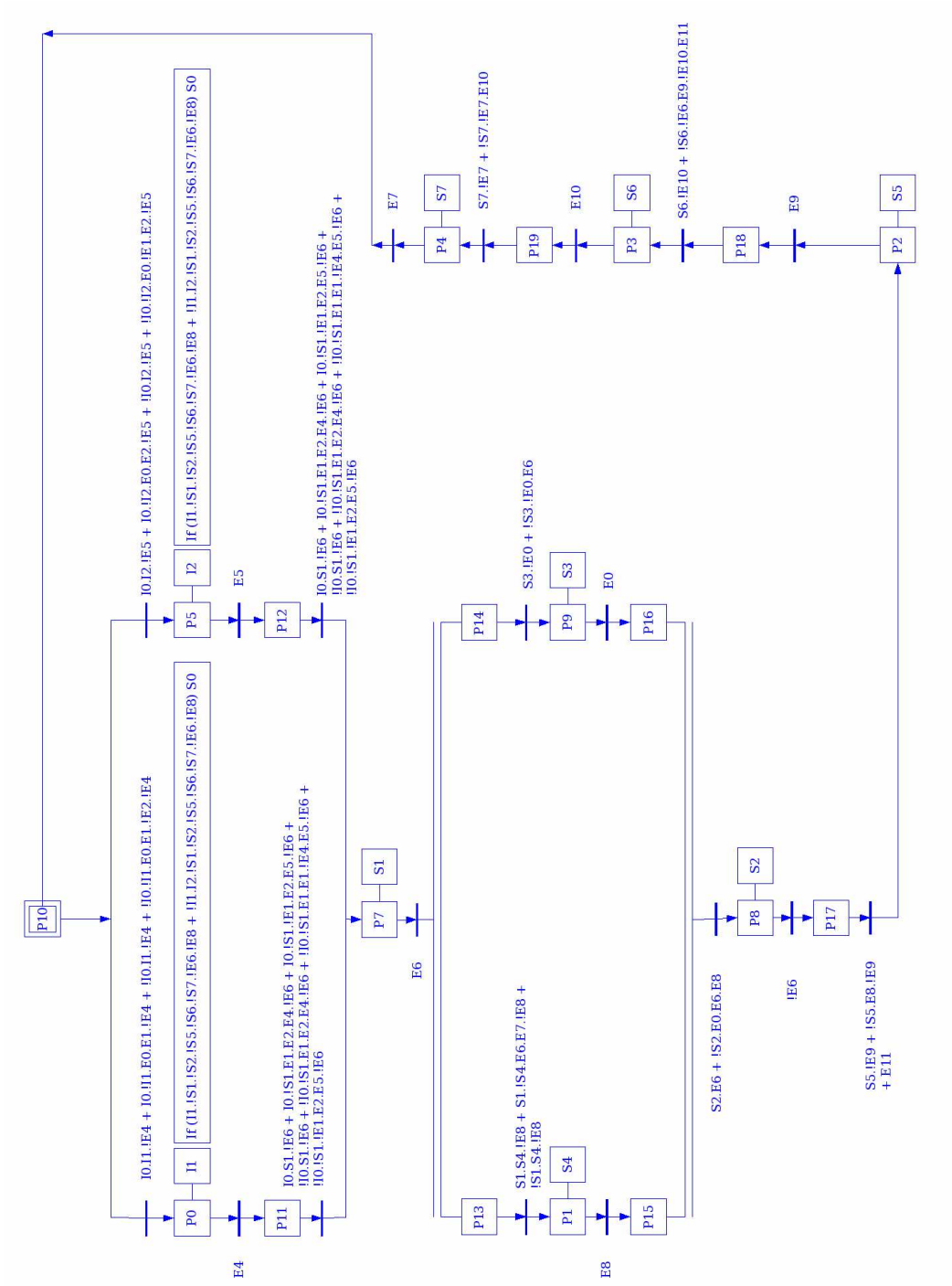


Figura 4.18: Diagrama grafcet gerado pelo protótipo

Capítulo 5

Conclusão

Propomos neste trabalho um novo método de recuperação de diagramas *grafcet* e de estados a partir de diagramas *ladder* usando BDDs. Fazendo uma análise preliminar do diagrama *ladder*, foi possível identificar de forma segura todos os estados, passos e ações condicionais do controlador. O método inovou a forma como as malhas do diagrama *ladder* foram representadas, ou seja, utilizando árvores sintáticas [41], utilizadas previamente na construção de compiladores. Navegando de maneira recursiva e eficiente por estas árvores, foi possível construir os BDDs de ativação de cada uma das malhas. Analisando estas malhas utilizando os seus selos, conseguimos manipular os BDDs de ativação, chegando aos BDDs de desativação destas malhas. Não há nenhuma referência na literatura de algum trabalho que tenha utilizado BDDs na análise de diagramas *ladder*.

Manipulando os BDDs das malhas do diagrama *ladder*, foi possível remover os intertravamentos existentes entre estas malhas, o que simplificou o processo de identificação dos possíveis sequenciamentos que pudessem acontecer na planta. É necessário que o usuário conheça a planta para que possamos solicitar a ele quais são os passos do diagrama *grafcet* que vêem após o passo inicial e quais são os estados do diagrama de estados que vêem após o estado inicial. Uma vez obtida esta informação, foi possível descobrir com segurança as possíveis transições entre os passos do diagrama grafcet e os estados do diagrama de estados comparando os BDDs das expressões de ativação e desativação de cada um deles. Inovação esta não alcançada no trabalho de Falcione e Krogh[2], que para considerar uma malha do diagrama *ladder* sequencial a uma outra, considerou que o endereço da bobina da primeira malha deva existir na segunda, não levantando assim todas as transições possíveis.

O levantamento das possíveis transições existentes no controlador finaliza o método

de extração de diagramas *grafcet* e de estados a partir de diagramas *ladder* para plantas que não possuam eventos simultâneos, ou seja, paralelismo. No entanto, fizemos uma contribuição para os casos em que o paralelismo existe. Analisando todos os cubos de duas malhas quaisquer, foi possível descobrir se elas não podem ser paralelas, eliminando assim uma grande quantidade de possibilidades de passos e estados que pudessem ser considerados paralelos, mas não o seriam por causa da planta que está sendo automatizada. Assim, não foi necessário descrever previamente todos os possíveis casos de paralelismo entre os passos do controlador, simplificando desta forma o processo de conversão. Esta vantagem não pode ser encontrada no método proposto por Falcione e Krogh[2], que necessita que todos os casos de paralelismo precisem ser informados pelo usuário previamente utilizando grafos de simultaneidade.

Assim, fazendo uma análise rigorosa do diagrama *ladder* foi possível eliminar uma grande quantidade de informações que precisassem ser informadas pelo usuário para efetuar a conversão. Ao contrário de todos os outros métodos, nenhuma modelagem preliminar da planta precisou ser feita para utilização do método proposto. Também não foi necessário modelar toda a planta, ao contrário do que acontece com os métodos de Zanma et al [3] e Lee e Lee [14], que utilizaram lógica temporal e tabela de estados para descreverem previamente toda a planta. Além disso, com o método proposto, as poucas informações necessárias ao processo de conversão são solicitadas ao usuário apenas quando elas são realmente necessárias. O estado da arte busca um método para trabalhar com grandes plantas industriais, que são formadas geralmente por vários sub-sistemas, e o método proposto solicita ao usuário apenas as informações do sub-processo que está sob análise. Desta forma, ele é adequado às grandes plantas industriais porque trabalha com redução de escopo. Com a utilização de uma interface gráfica amigável para obter as poucas informações do usuário, o método tornou-se ainda mais atraente.

Sabemos que grandes plantas industriais, quando são automatizadas usando diagramas *ladder*, necessitam de blocos avançados e temporizadores, que não foram tratados neste trabalho. No entanto, após uma análise matemática do desempenho do algoritmo utilizado, concluímos que a eficiência dele não está limitada ao tamanho das plantas que são encontradas na indústria. Quando estes blocos avançados e temporizadores forem tratados em trabalhos futuros, este método poderá ser utilizado na conversão de diagramas *ladder* de grandes dimensões, encontrados na indústria, onde um único diagrama *ladder* pode ter 500 malhas ou mais. Como o método é eficiente e os dados da planta necessários ao processo de conversão são obtidos do usuário com praticidade, o método proposto é uma importante e inédita contribuição para a área de sistemas de automação.

Apêndice A

Diagramas de Controle Sequencial

A.1 O Diagrama Ladder

Apresentamos primeiramente um exemplo do mesmo na figura A.1, formado apenas por uma malha.

A.1.1 Componentes

Uma malha é formada por contatos normalmente abertos, contatos normalmente fechados, ligações em série, ligações em paralelo e uma bobina, que por sua vez poderá ter como endereço um endereço de saída físico ou um endereço usado apenas internamente pelo PLC.

Na figura A.1 $\boldsymbol{u_1}$ e $\boldsymbol{u_3}$ são exemplos de contatos normalmente abertos e fechados, respectivamente. Os contatos normalmente abertos e normalmente fechados também podem possuir endereços internos, endereços de entrada e endereços de saída físicos. Quando o endereço de um contato normalmente aberto torna-se ativo, então o contato normalmente aberto fecha. Quando o endereço é desativado, o contato normalmente aberto abre. Quando o endereço de um contato normalmente fechado torna-se ativo, então o contato normalmente fechado abre. Quando este endereço é desativado, então o contato normalmente fechado fecha.

Na figura A.1, $\boldsymbol{q_1}$ é um exemplo de uma bobina. Neste exemplo, $\boldsymbol{q_1}$ tornar-se-á ativo se a expressão lógica $\boldsymbol{q_1 = (u_1 \cdot u_3 + q_1) \cdot q_2 \cdot u_2}$ tornar-se verdadeira. Como o leitor pode ver, o OU e o E lógicos existes entre $\boldsymbol{u_1 u_3}$ e $\boldsymbol{q_1}$, e entre $\boldsymbol{q_2}$ e $\boldsymbol{u_2}$ são representados na malha do

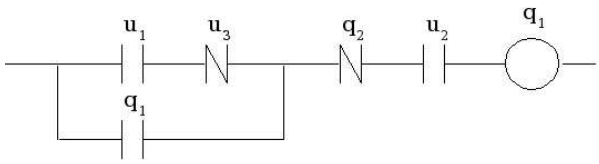


Figura A.1: Uma malha de um diagrama *ladder*

diagrama *ladder* por uma ligação em paralelo e por uma ligação em série, respectivamente. Para obter maiores detalhes, algumas bibliografias são Silveira [6], Georgini [7] e Natale [8].

A.1.2 Interpretação

Alguns trabalhos, como o de Itoh et al[27] e o de Koo e Kwon [28] apresentam a arquitetura e o mecanismo de execução de um diagrama *ladder*. Itoh et al[27] modelou a arquitetura de um PLC em várias partes, como mostra a figura A.2. As interfaces de entrada e saída são geralmente relés ou transistores. O ciclo de funcionamento de um PLC é mostrado na figura A.3. Como o leitor pode ver, inicialmente é feita a leitura dos dispositivos de entrada e seus dados são armazenados na memória. A seguir, é feita uma varredura de todo o diagrama *ladder*, com base nos dados armazenados na memória, onde a CPU analisa cada uma das malhas, verificando se sua respectiva bobina deve estar ativa ou não. Então, a memória é atualizada com os novos valores resultantes do processo de varredura. A seguir, com base na memória, o PLC atualiza o estado dos dispositivos de saída através da interface de saída. Para obter maiores detalhes, o leitor poderá consultar a bibliografia acima citada.

A.2 O Diagrama Grafcet

Segundo David [9], o diagrama *grafcet* é um grafo contendo 2 tipos de nós: passos e transições, sendo que ele deve conter pelo menos um passo e uma transição. Arcos direcionados tanto podem conectar passos à transições quanto transições aos arcos. Como falamos anteriormente, o *grafcet* é muito similar às Redes de Petri: um passo e uma transição do *grafcet* são similares a um *place* e uma transição das Redes de Petri, respectivamente. Uma versão padronizada do *grafcet* foi apresentada pela IEC [25]. Atualmente, está sendo feita a manutenção desta padronização, com previsão para término em 2006.

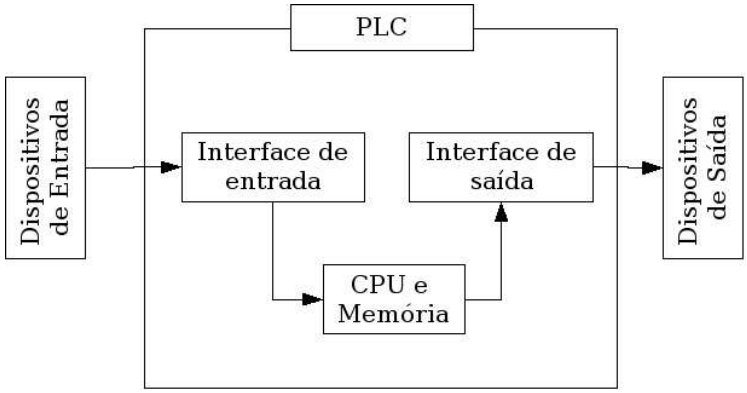


Figura A.2: Arquitetura de um PLC

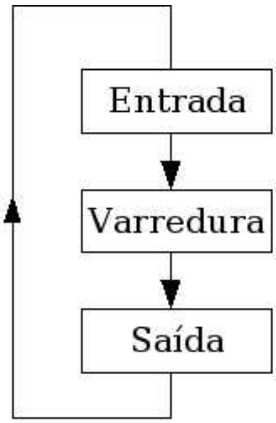


Figura A.3: Ciclo

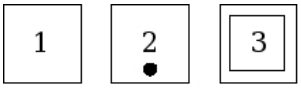


Figura A.4: Passos

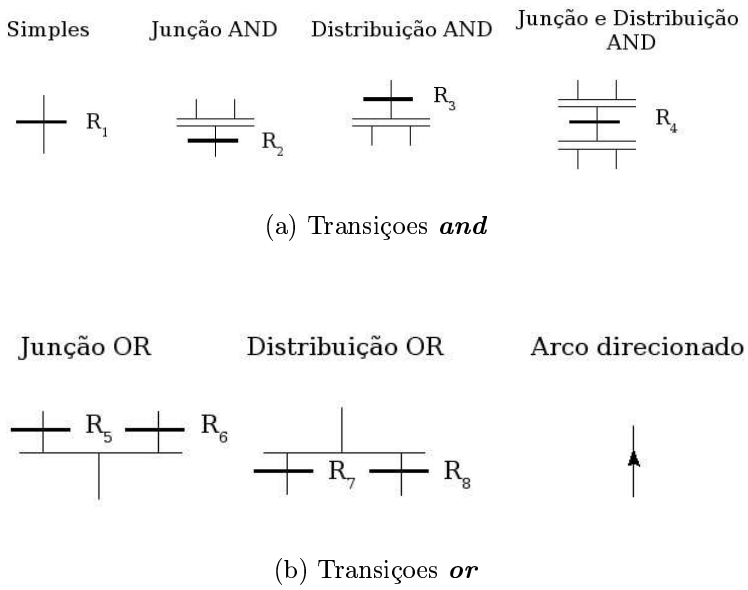
Esta nova versão está sendo chamada de IEC 60848. Infelizmente, não temos em mãos a especificação original, no entanto David [9] fez uma breve apresentação da versão padronizada pela IEC no trabalho dele. Assim, utilizaremos a mesma formalização apresentada por David, tendo em vista que ela está em conformidade com a norma original.

A.2.1 Passos e Transições

Um passo é representado por um quadrado, como mostra a figura A.4. Uma passo pode ter dois estados: ativo ou inativo. O passo 1 da figura A.4 está inativo, enquanto o passo 2 está ativo. O passo 3 é um tipo especial de passo, chamado de passo inicial. O circulo negro do passo 2 da figura é chamado de token.

Uma transição é representada como mostra a figura A.5. A transição é representada por uma barra. Como o leitor pode ver, existem junções e distribuições do tipo ***and*** e do tipo ***or***. As junções e distribuições do tipo ***and*** possuem uma única transição e vários arcos de saída, um para cada passo. As junções e distribuições do tipo ***or*** possuem várias transições e um único arco para um único estado. Toda transição possui uma receptividade $\boldsymbol{R_i}$. A receptividade é uma expressão booleana formada por endereços do PLC de entrada, de saída ou endereços internos. Os arcos direcionados também são apresentados na figura A.5. Um arco vertical que não possui seta possui, por convenção, sentido de cima para baixo. Caso o sentido do arco seja de baixo para cima, então ele obrigatoriamente deverá possuir uma seta indicando este sentido. Um arco direcionado sempre deve partir de um passo para uma transição ou de uma transição para um passo.

Também é importante dizer que um passo pode não possuir transições de entrada e/ou saída. Frequentemente, o passo 3 da figura A.4 não possui transição de entrada, mas apenas de saída. Transições sem passo de entrada e saída são chamadas de transições de nascimento (source) e de falecimento (sink), respectivamente. Por outro lado, um arco direcionado sempre deve ter um nó de chegada (passo ou transição) e um nó de saída (transição ou passo).



(a) Transiçoes ***and***

(b) Transiçoes ***or***

Figura A.5: Arcos direcionados e transições

A.2.2 Passagem pelas Transições

No original esta operação é chamada de *firing of transiction.* Um passo ativo possui um e somente um token. Um passo inativo não possui nenhum token. Todos os passos ativos em um determinado momento definem a situação deste momento. Uma situação corresponde a um estado do sistema. A passagem de uma situação para a outra se dá através da passagem pelas transições do sistema. As entradas do PLC estão associadas com as transições, enquanto que as saídas do PLC estão associadas com os passos.

DEFINIÇÃO: Uma transição permite passagem se ambas as condições são satisfeitas:

1. Se todos os passos que precedem a transição estiverem ativos. Dizemos que neste caso a transição está habilitada;

2. se a receptividade da transição estiver verdadeira.

Quando as duas condições da definição A.2.2 são satisfeitas, então o token do passo que precede a transição passa para o passo que segue a transição. A receptividade é considerada verdadeira em 3 situações diferentes:

1. *A receptividade é uma condição*: neste caso o token do passo que precede a transição passará pela transição tão logo a mesma torne-se verdadeira. Enquanto a transição for falsa, o token não atravessa a transição;

2. *a receptividade é um evento*: neste caso o que é levado em consideração não é o nível lógico do endereço de entrada, mas sim a passagem do nível lógico zero para o nível logico um ou a passagem do nível lógico um para o nível lógico zero do endereço em questão;

3. *a receptividade é um produto entre um evento e uma condição*: por exemplo uma receptividade $\boldsymbol{R_3} = \uparrow \boldsymbol{a} \cdot \boldsymbol{b}$ será verdadeira quando o evento $\uparrow \boldsymbol{a}$ ocorrer e quando a condição $\boldsymbol{b}$ for verdadeira.

É importante dizer que o produto entre um evento e uma condição será sempre um evento. Uma soma entre um evento e uma condição pode ser considerada uma soma entre dois eventos, formando assim um outro evento por extensão.

A.2.3 Regras de Passagem

1. Todas as transições que estiverem permitindo passagem são imediatamente atravessadas;

2. várias transições que estiverem permitindo passagem são simultaneamente atravessadas;

3. quando um passo tiver que permanecer simultaneamente ativo e inativo, ele deverá permanecer ativo.

A.2.4 Ações e Saídas

As ações podem ser classificadas como ações por nível e ações por impulso.

1. *Ações por nível*: pode ser modelada por uma variável booleana, podendo ser condicional ou incondicional;

2. *ações por impulso*: é responsável por alterar o valor de uma variável discreta.

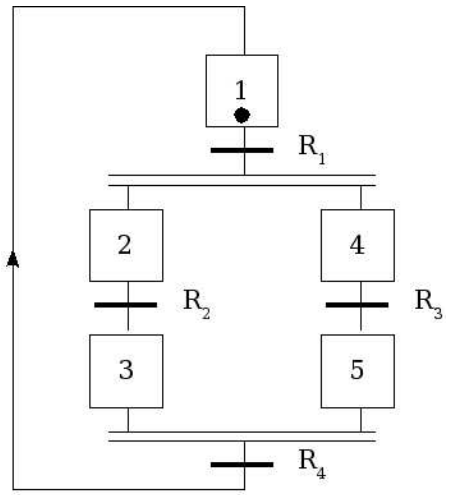


Figura A.6: Sincronização e concorrência

A principal diferença entre uma ação por nível e uma por impulso é que a primeira somente permanecerá ativa se o passo que ela estiver associada for estável, podendo ser ativa por um intervalo de tempo indeterminado, enquanto a segunda possue um tempo de atuação infinitamente pequeno, mesmo que o passo seja instável. Segundo David [9], um passo associado por uma ação por nível e por impulso indicam um estado e uma ordem, respectivamente. Neste trabalho abordaremos apenas as ações por nível.

As saídas poderão ser formadas por ações por nível e possívelmente por ações por impulso, desde que sejam usadas com memória, como por exemplo *abrir válvula* e *fechar válvula,* exemplos de saídas representadas por ações por impulso. Como estamos considerando apenas ações por nível, também utilizaremos apenas saídas por nível.

A.2.5 Concorrência e Sincronização

Vamos ilustrar estes conceitos através de um exemplo. Vejamos a figura A.6. Como o leitor pode ver, a situação inicial do sistema é $\mathbf{1}$, mas após a passagem pela transição $\boldsymbol{R_1}$ a nova situação passa a ser $\mathbf{2, 4}$. Neste exato momento temos um problema de concorrência entre as partes $\mathbf{2}, \boldsymbol{R_2}, \mathbf{3}$ e $\mathbf{4}, \boldsymbol{R_3}, \mathbf{5}$. A transição $\boldsymbol{R_4}$ efetua a sincronização do sistema, uma vez que só existirá passagem por $\boldsymbol{R_4}$ se ambos os estados $\mathbf{3, 5}$ estiverem ativos simultaneamente.

A.2.6 Interpretação

A interpretação de um diagrama *grafcet* é feita obedecendo-se o algoritmo descrito a seguir:

1. Ative os passos iniciais e execute as ações por impulsos a eles associados. Vá para o passo 5;

2. quando um novo evento externo ocorrer, determine o conjunto $\boldsymbol{T_1}$ de todas as transições que permitem passagem por causa da ocorrência deste evento. Se $\boldsymbol{T_1}$ não é um conjunto vazio, vá para o passo 3. Caso contrário, modifique se necessário o estado das ações condicionais associadasa com os passos ativos. Aguarde por um novo evento neste passo 2;

3. passe por todas as transições que estiverem permitindo passagem. Caso a situação permaneça inalterada após esta passagem simultânea pelas transições, vá para o passo 6;

4. execute todas as ações por impulso associadas com os passos que tornaram-se ativos durante a execução do passo 3;

5. determine o conjunto $\boldsymbol{T_2}$ das transições que permitem passagem após a ocorrência de um evento $\boldsymbol{e}$ (sempre ativo). Caso $\boldsymbol{T_2}$ não seja vazio, vá para o passo 3;

6. uma situação estável foi alcançada.
   6.1. determine o conjunto $\boldsymbol{A_0}$ de ações por nível que devem ser desativadas;
   6.2. determine o conjunto $\boldsymbol{A_1}$ de ações por nível que devem ser ativadas.;
   6.3. desative todas as ações que pertencem ao conjunto $\boldsymbol{A_0}$ mas não pertencem ao conjunto $\boldsymbol{A_1}$. Ative todas as ações que pertencem ao conjunto $\boldsymbol{A_1}$. Vá para o passo 2.

Para maiores detalhes sobre o diagrama *grafcet*, o leitor poderá consultar alguns trabalhos como o de David [9] ou a especificação da IEC [25].

A.2.7 Interpretação

A interpretação de um diagrama *grafcet* é feita obedecendo-se o algoritmo descrito a seguir:

1. Ative os passos iniciais e execute as ações por impulsos a eles associados. Vá para o passo 5;

2. quando um novo evento externo ocorrer, determine o conjunto $\boldsymbol{T_1}$ de todas as transições que permitem passagem por causa da ocorrência deste evento. Se $\boldsymbol{T_1}$ não é um conjunto vazio, vá para o passo 3. Caso contrário, modifique se necessário o estado das ações condicionais associadasa com os passos ativos. Aguarde por um novo evento neste passo 2;

3. passe por todas as transições que estiverem permitindo passagem. Caso a situação permaneça inalterada após esta passagem simultânea pelas transições, vá para o passo 6;

4. execute todas as ações por impulso associadas com os passos que tornaram-se ativos durante a execução do passo 3;

5. determine o conjunto $\boldsymbol{T_2}$ das transições que permitem passagem após a ocorrência de um evento $\boldsymbol{e}$ (sempre ativo). Caso $\boldsymbol{T_2}$ não seja vazio, vá para o passo 3;

6. uma situação estável foi alcançada.
   6.1. determine o conjunto $\boldsymbol{A_0}$ de ações por nível que devem ser desativadas;
   6.2. determine o conjunto $\boldsymbol{A_1}$ de ações por nível que devem ser ativadas.;
   6.3. desative todas as ações que pertencem ao conjunto $\boldsymbol{A_0}$ mas não pertencem ao conjunto $\boldsymbol{A_1}$. Ative todas as ações que pertencem ao conjunto $\boldsymbol{A_1}$. Vá para o passo 2.

Para maiores detalhes sobre o diagrama *grafcet*, o leitor poderá consultar alguns trabalhos como o de David [9] ou a especificação da IEC [25].

A.3 Máquina de Estados

A máquina de estados de Harel [5], também conhecida como statechart, é uma formalização utilizada para especificação de sistemas reativos. A figura 2.24 apresenta um exemplo simples de uma máquina de estados de Harel [5].

As setas assinaladas pelas letras $\boldsymbol{\alpha, \beta, \delta, \gamma}$ representam transições, os retângulos com bordas arredondadas representam estados e a letra P representa uma condição para que a transição $\boldsymbol{\gamma}$ leve a situação do sistema de A para C. A máquina de estados de Harel

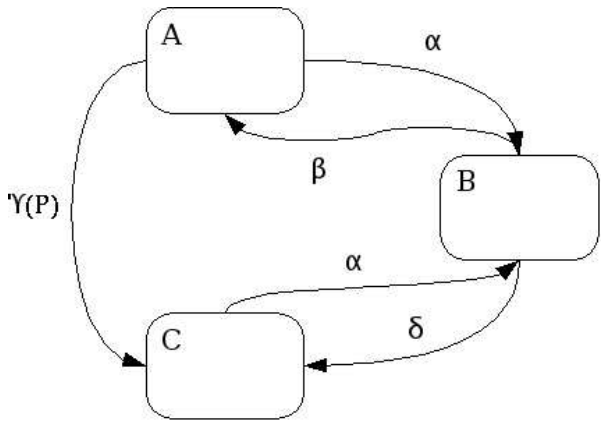


Figura A.7: Máquina de estados

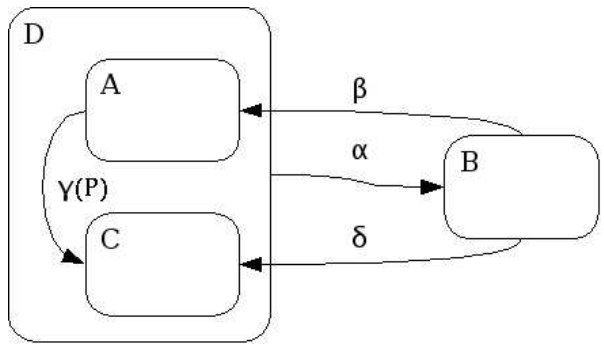


Figura A.8: Super estado

[5] implementa o coneito de super-estado. Como na figura 2.24, a transição $\boldsymbol{\alpha}$ ocorria tanto de $\boldsymbol{A^{\alpha} \rightarrow B}$ quanto de $\boldsymbol{C^{\alpha} \rightarrow B}$, então podemos criar um super estado D, tal que $\boldsymbol{D^{\alpha} \rightarrow B}$, como mostra a figura A.8.

A máquina de estados Harel [5] também possui mecanismos que tornam a manipulação das transições mais econômica e graficamente menos complexa. Vejamos a figura A.9. Entretanto, é importante dizer que estas representações compactas das transições não possuem o mesmo significado das junções e disjunções do diagrama *grafcet*. Caso o leitor deseje implementar paralelismo usando a maquina de estados de Harel [5], a notação da figura A.10 deverá ser utilizada. Nela, a situação do sistema é uma combinação entre os estados *B, C* e *E, F, G*. O pequeno ponto negro presente nos super-estados *A* e *D* indica que os estados *B* e *F* são os estados iniciais do super-estado *Y*.

A máquina de estados de Harel [5] também implementa um conceito chamado *ações*.

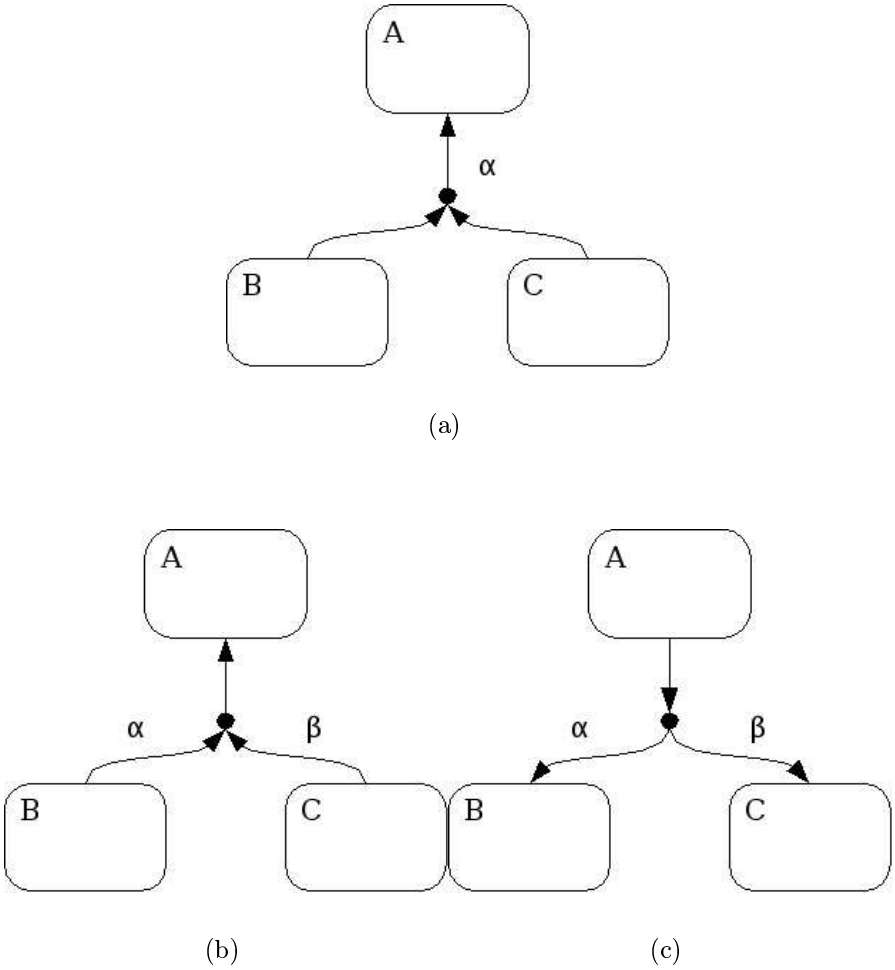


Figura A.9: Junções e disjunções

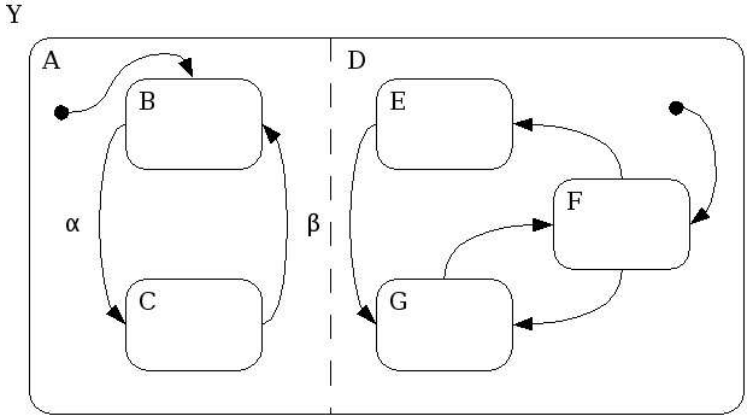


Figura A.10: Paralelismo

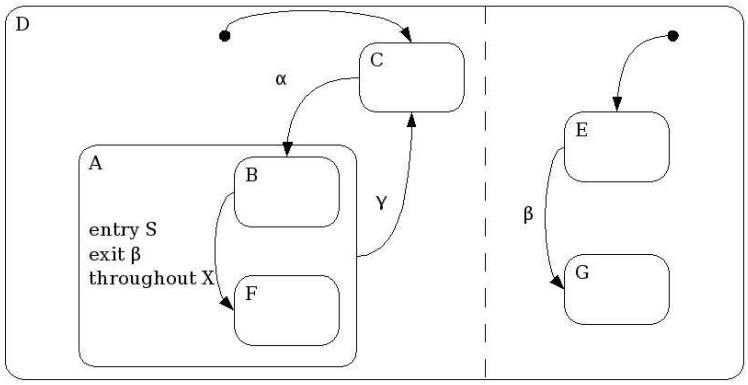


Figura A.11: Ações

Estas podem ocorrer em diferentes instantes, tais como durante a entrada em um estado, durante a saída do estado ou enquanto o estado estiver ativo. Vejamos a figura A.11. Nela, a ação S ocorre no instante da entrada, a ação $\boldsymbol{\beta}$ ocorre durante a saída e a ação X permanece ativa enquanto o estado A estiver ativo.

O leitor poderá consultar o trabalho de Harel [5] para estudar os demais conceitos existentes na literatura. Vamos apresentar apenas estes conceitos pois eles estão diretamente relacionados com nosso trabalho.

Apêndice B

Outros Métodos de Recuperação de Diagramas Grafcet a partir de Diagramas Ladder

Como mensionamos no capítulo 1, surge a necessidade de se extrair informações do fluxo de controle do processo da planta através de um diagrama *ladder*. Já existem algums trabalhos nesta área presentes na literatura. Vamos falar resumidamente sobre alguns deles.

Falcione e Krogh [2] apresentaram um método de engenharia reversa de diagramas *ladder* fazendo uso de algumas ferramentas, tais como o grafo de simultaneidade, o grafo de dependência e o grafo de simultaneidade condensado. Durante a engenharia reversa, o grafo de simultaneidade condensado é decomposto em sub-grafos através de duas decomposições: a decomposição de componentes conectados e a decomposição de elementos totalmente conectados. As malhas particionadas pela primeira decomposição são colocadas em uma única estrutura seqüencial, enquanto que as malhas particionadas pela decomposição de elementos totalmente conectados são colocadas em uma estrutura paralela. Aplicando esta decomposição sobre o grafo de simultaneidade condensado produz-se um ou mais sub-grafos para os quais novos passos no SFC são atribuídos. Estes novos passos são colocados em uma estrutura de um único caminho. Uma vez que nenhum passo desta estrutura pode ficar ativo concorrentemente, o sequenciamento destes passos é possível. A vantagem deste método é a sua simplicidade, uma vez que as operações realizadas sobre o sistema podem ser modelagas graficamente. A desvantagem é que este método requer informações da planta representadas sob a forma de um grafo de simultaneidade.

Zanma et al [3] apresentaram um método de engenharia reversa de diagramas *ladder* usando informações da planta representadas em lógica temporal. Fazendo uma complexa análise do diagrama *ladder* e comparando com as informações da planta, chega-se ao diagrama *grafcet*. A vantagem deste método é que uma vez conhecendo-se toda a informação da planta, pode-se fazer uma análise mais completa durante a engenharia reversa de um diagrama *ladder*, ganhando-se segurança no processo. A desvantagem deste método é que para diagramas *ladder* muito grandes, fazer a modelagem da planta usando-se lógica temporal de forma correta e precisa, sem que novos erros venham a ser introduzidos no sistema, é um novo problema a ser resolvido, quase tão difícil quanto extrair as informações da planta a partir de um diagrama *ladder*.

Nakamura et al [10] propôs um método de engenharia reversa de diagramas *ladder* usando técnicas de programação linear. Basicamente, o sistema é modelado usando-se equações de estado. $\boldsymbol{X_n}$ representa a variável de estado como uma bobina de um diagrama *ladder*. $\boldsymbol{U_n}$ representa a variável de estado da planta como uma entrada para o diagrama *ladder*. $\boldsymbol{N}$ indica o passo final. Temos então

$$\boldsymbol{X_n(x_{1n}, x_{2n}, \cdots, x_{tn})}$$

$$\boldsymbol{U_n(u_{1n}, u_{2n}, \cdots, u_{sn})}$$

onde $\boldsymbol{t}$ e $\boldsymbol{n}$ representam o número de bobinas e endereços de entrada do sistema, respectivamente. Assim, um sistema pode ser modelado da seguinte forma:

$$\boldsymbol{X_1 = f(X_0, U_0)}$$

$$\boldsymbol{U_1 = g(X_1, U_0)}$$

$$\vdots$$

$$\boldsymbol{U_n = g(X_n, U_{n-1})}$$

$$\boldsymbol{X_{n+1} = f(X_n, U_n)}$$

onde $\boldsymbol{f(X_n, U_n)}$ e $\boldsymbol{g(X_n, U_{n-1})}$ são funções lógicas usando expressões variáveis booleanas. $\boldsymbol{f(X_n, U_n)}$ pode ser extraída diretamente a partir do diagrama *ladder*, mas $\boldsymbol{g(X_n, U_{n-1})}$ deve ser extraída pelo usuário. As expressões acima são então transformadas em um sistema de inequações que é solucionado usando programação linear. Encontrar a melhor solução destas inequações equivale a encontrar o melhor caminho de um grafo orientado onde são atribuídos custos às arestas deste grafo. A desvantagem deste método é que ele também precisa que o usuário forneça informações da planta, aqui representadas

pelas funções $\boldsymbol{g(X_i, U_{i-1})}$. O leitor poderá consultar o trabalho de Gerez [15] para obter mais informações sobre o uso de programação linear aplicado a problemas de eletrônica digital.

Shoji et al [11] propôs um método de recuperação do fluxo de controle do processo da planta a partir de um diagrama *ladder*. Inicialmente, é necessário fazer uma separação entre o controle de rotina e o controle que não é de rotina. São exemplos de controle que não são de rotina: parada de emergência, inspeção manual e manutenção. A fim de se extrair o controle que não é de rotina a partir de um diagrama *ladder*, é feita uma análise com base na frequencia dos endereços, ou seja, um endereço que apareça 4 vezes em um diagrama *ladder* terá frequencia 4. Assim, sabendo-se que os endereços de rotina e não-rotina possuem um baixo e um alto indice de frequencia, respectivamente, pode-se extrair o controle de rotina a partir de um diagrama *ladder*.

Alguns métodos também foram propostos para recuperação de redes de Petri a partir de diagramas *ladder*. Podemos citar os trabalhos de Lee e Lee [14] e o trabalho de Lee e Hsu [38]. Uma boa referência sobre uma reflexão dos métodos de recuperação de redes de Petri a partir de diagramas *ladder* é o trabalho de Peng e Zhou [12].

Apêndice C

O formato do arquivo de entrada

O arquivo de entrada pode ser dividido em três partes principais: a parte inicial descreve os endereços do diagrama *ladder*, a parte intermediária descreve as malhas do diagrama *ladder* e a última parte descreve as informações gráficas do diagrama *ladder*, conforme podemos ver na figura C.1.

Figura C.1: O arquivo de entrada

Após a leitura do arquivo de entrada para a memória, é formada uma árvore DOM. As partes principais da árvore DOM são um reflexo das partes principais do arquivo XML de entrada, conforme podemos ver na figura C.2. Como podemos ver, existem três partes principais: a parte dos endereços, a parte que representa as malhas do diagrama *ladder* e a parte que representa as interfaces do diagrama *ladder*.

Agora que conhecemos as partes principais do arquivo de entrada, podemos compreender com mais facilidade o arquivo XML. A parte inicial do arquivo começa com uma TAG sinalizando o início do projeto. A seguir, são descritos os endereços, um a um. Todo

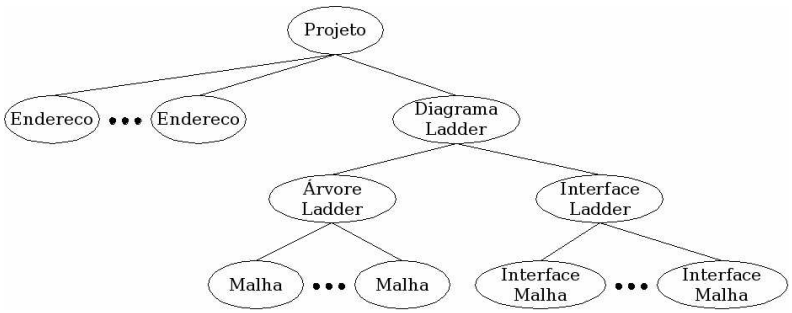


Figura C.2: A árvore DOM

endereço possui três atributos, que são o tipo de endereço, o número do endereço e um flag sinalizando se ele está ativado ou não. Os endereços podem ser internos, representados pela letra I, de saída, representados pela letra S ou de entrada, representados pela letra E.

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE Projeto SYSTEM "Projeto.dtd">

<Projeto>
  <Endereco ativado = "false" tipo = "I" numero = "0"/>
  <Endereco ativado = "false" tipo = "I" numero = "1"/>
  <Endereco ativado = "false" tipo = "I" numero = "2"/>
```

A parte intermediária do arquivo contém a descrição das malhas do diagrama *ladder*. Um conjunto de malhas é chamado de árvore *ladder*. Uma árvore *ladder* juntamente com uma interface *ladder* constituem um diagrama *ladder*. Uma malha é formada por ligações em série, ligações em paralelo, contatos normalmente abertos, normalmente fechados e bobinas.

```
<DiagramaLadder nome = "ladder">
  <ArvoreLadder>

    <Malha id = "1">
      <LigacaoS>
        <LigacaoP>
          <LigacaoS>
```

```
            <ContatoNA id = "1" idEndereco = "E0"/>
            <ContatoNA id = "2" idEndereco = "E1"/>
          </LigacaoS>
          <LigacaoS>
            <ContatoNA id = "3" idEndereco = "I1"/>
          </LigacaoS>
        </LigacaoP>
        <ContatoNF id = "4" idEndereco = "E4"/>
      </LigacaoS>
      <Bobina id = "1" idEndereco = "I1"/>
    </Malha>
  </ArvoreLadder>
```

A forma como estes elementos se relacionam está ilustrada na figura C.3.

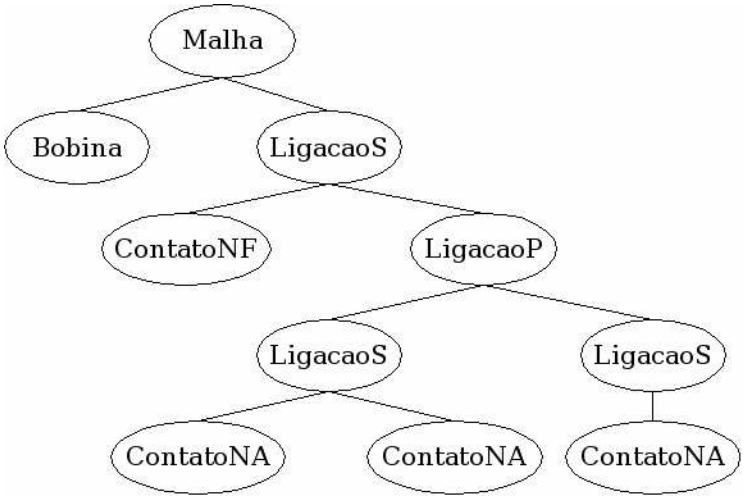


Figura C.3: A árvore DOM de uma malha do diagrama *ladder*

A parte final do arquivo XML, contendo as informações gráficas do diagrama *ladder*, está descrita a seguir. Basicamente, todo elemento da malha de um diagrama *ladder* possui um elemento gráfico na TAG de interface do diagrama *ladder*. A separação da interface gráfica da parte lógica foi feita com o objetivo de melhorar a manutenibilidade, pois futuras modificações na interface do diagrama *ladder* estarão pouco acopladas da parte lógica, e vice-versa.

```
<InterfaceLadder>
```

```
  <InterfaceMalha id = "6">
    <InterfaceLigacaoS>
      <InterfaceLigacaoP>
        <InterfaceLigacaoS>
          <InterfaceContatoNA idContato = "27" />
        </InterfaceLigacaoS>
        <InterfaceLigacaoS>
          <InterfaceContatoNA idContato = "28" />
        </InterfaceLigacaoS>
      </InterfaceLigacaoP>
      <InterfaceContatoNF idContato = "29"/>
      <InterfaceContatoNF idContato = "30"/>
      <InterfaceConectorH idConector = "47"/>
      <InterfaceConectorH idConector = "48"/>
    </InterfaceLigacaoS>
    <InterfaceBobina idBobina = "6"/>
  </InterfaceMalha>
</InterfaceLadder>

</DiagramaLadder>
```

Referências Bibliográficas

[1] Hill, Frederick J.; Peterson, Gerald R.; Introduction to Switching Theory e Logical Design. Páginas 287, 290, 294, 295. Editora John Wiley & Sons, 1981, terceira edição.

[2] Falcione, Albert; Krogh, Bruce H.; Design Recovery for Relay Ladder Logic. IEEE Control Systems, páginas 90-98. Agosto de 1993.

[3] Zanma, Tadanao; Suzuki, Tatsuya; Inaba, Akio; Okuma, Shigeru; Transformation Algorithm from Ladder Diagram to SFC Using Temporal Logic. Electrical Engineering in Japan, volume 129, issue 1, páginas 74-81. 1999.

[4] Pressman, Roger S.; Engenharia de Software. 2002.

[5] Harel, David; Statecharts: A Visual Formalism for Complex Systems. Elsevier Science Publishers. 1984.

[6] Silveira, Paulo Rogério da; Santos, Winderson E. dos; Automação de Controle Discreto. São Paulo. Editora Érica. 1998.

[7] Georgini, Marcelo; Automação Aplicada. Descrição e Implementação de Sistemas Seqüenciais com PLCs. São Paulo. Editora Érica. 2000.

[8] Natale, Ferdinando; Automação Industrial. São Paulo. Editora Érica. 2000.

[9] David, René; Grafcet: A Powerfull Tool for Specification of Logic Controllers. IEEE Transactions on Control Systems Technology. September 1995.

[10] Nakamura, Shogo; Fuji, Yasumasa; Sekiguchi, Takashi; Study on a Transformation Method of Ladder Diagram into Sequential Function Chart on the Basis of Linear Programming Technique. Emerging Technologies and Factory Automation Proceedings. 1997.

[11] Shoji, Norimasa; Ikkai, Yoshitomo; Komoda, Norihisa; An Extraction Method of Control Flow from Ladder Diagram by Judgement of Transfer Frequency. Proceedings on Emerging Technologies and Factory Automation. 1999.

[12] Peng, ShihSen; Zhou, MengChu; Conversion between Ladder Diagrams and PNs in Discrete-Event Control Design - A Survey. IEEE International Conference on Systems, Man, and Cybernetics. 2001.

[13] Venkatesh, Kurapati; Zhou, MengChu; Caudill, J. Reggie; Comparing Ladder Logic Diagrams and Petri Nets for Sequence Controller Design Through a Discrete Manufacturing System. IEEE Transactions on Industrial Eletronics. 1994.

[14] Lee, Gi Bum; Lee, Jin S.; Conversion of Ladder Diagram into Petri Nets. Proceedings of the IASTED International Conference, ROBOTICS and APPLICATIONS. October 28-30, 1999, Santa Barbara, California, USA.

[15] Gerez, Sabih H.; Algorithms for VLSI Design Automation. John Wiley & Sons LTDA. 1999.

[16] Bryant, Randal E.; Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers. Agosto de 1986.

[17] Yuan, J.; Shults, Kurt; Pixley, Carl; Miller, Hillel; Aziz, Adnan; Modeling Design Constraints and Biasing in Simulation Using BDDs. IEEE/ACM International Conference on Computer-Aided Design. Novembro de 1999.

[18] Thacker, Robert A.; Myers, Chris J.; Synthesis of Timed Circuits using BDDs. Proc. International Workshop on Logic Synthesis. 1997.

[19] Sztipanovits, Janos; Misra, Amit; Diagnosis of Discrete Event Systems Using Ordered Binary Decision Diagrams. 7th International Workshop on Principles of Diagnosis. 1996.

[20] Bryant, Randal E.; Symbolic Manipulation of Boolean Functions Using a Graphical Representation. 22nd IEEE Design Automation Conference. 1985.

[21] Scholl, Christoph; Drechsler, Rolf; Becker, Bernd; Functional Simulation using Binary Decision Diagrams. IEEE/ACM Internation Conference on Computer-Aided Design. Novembro de 1997.

[22] Kuo, Ming-Ter; Wang, Yifeng; Cheng, Chung-Kuan; Fujita, Masahiro; BDD-Based Logic Partitioning for Sequential Circuits. Proceedings of the ASP-DAC'97 Design Automation Conference. Janeiro de 1997.

[23] Jacobi, R.; Logic Decomposition with Binary Decision Diagrams. IX Congresso da Sociedade Brasileira de Microeletrônica. Rio de Janeiro. Agosto de 1994.

[24] Lai, Yung-Te; Pedram, Massoud; Vriudhula, B. K; BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis. Proceedings of the 30th International Conference on Design Automation. Julho de 1993.

[25] IEC - International Electrotechnical Commission; Preparation of Function Chart for Control Systems. Publication 848. 1988.

[26] IEC - International Electrotechnical Commission. Norma IEC 61131. 2003.

[27] Itoh, Yoshiaki; MiYazawa, Iko; Sekiguchi, Takashi; Study on Execution Time of Ladder Diagram in Programmable Controller. Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society. 1998.

[28] Koo, Kyeonghoon; Kwon, Wook Hyun; Predicting Execution Time of Relay Ladder Logic for Programmable Logic Controllers. IEEE Conference on Emerging Technologies and Factory Automation. 1996.

[29] Wareham, Robert; Ladder Diagram and Sequential Function Chart: Languages in Programmable Controllers. Fourth Annual Canadian Programmable Control and Automation Technology Conference and Exhibition Conference Proceedings. 1988.

[30] Denavathan, R.; Computer Aided Design of Relay Ladder Diagram from Functional Specifications. 16th Annual Conference of IEEE Industrial Electronics Society. 1990.

[31] Venkateshi, Kurapati; Zhou, MengChu; Caudill, Reggle; Evaluating the Complexity of Petri Nets and Ladder Logic Diagrams for Sequence Controllers Design in Flexible Automation. IEEE Symposium on Emerging Technologies & Factory Automation. 1994.

[32] Zhou, MengChu; Twiss, Edward; A Comparison of Relay Ladder Logic Programming and Petri Net Approach for Sequential Industrial Control Systems. Proceedings of the 4th IEEE Conference on Control Applications. 1995.

[33] Uzam, M.; Jones, A. H.; Ajlouni; Conversion of Petri Net Controllers for Manufecturing Systems into Ladder Logic Diagrams. Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation. 1996.

[34] Davidson, Craig; McWhinnie; Stepping off the Ladder. IEE Review. Setembro de 1997.

[35] Zhou, MengChu; Twiss, Edward; Design of Industrial Automated Systems via Relay Ladder Logic Programming and Petri Nets. IEEE Transactions on Systems, Man., and Cybernetics. Applications and Reviews. Fevereiro de 1998.

[36] Satoh, Takashi; Oshima, Hiroo; Nose, Kazuo; Kumagai, Sadatoshi; Automatic Generation System of Ladder List Program by Petri Net. IEEE International Workshop on Emerging Technologies and Factory Automation. 1992.

[37] Jiménez, Italia; López, Ernesto; Ramírez, Antonio; Synthesis of Ladder Diagrams from Petri Nets Controller Models. Proceedings of the IEEE International Symposium on Intelligent Control. Setembro de 2001.

[38] Lee, Jin-Shyan; Hsu, Pau-Lo; A New Approach to Evaluate Ladder Logic Diagrams and Petri Nets via the IF-THEN Transformation. IEEE International Conference on Systems, Man., and Cybernetics. 2001.

[39] Latha, K.; Umamaheswari, B.; University, Anna; Supervisory Control of an Autometed System with Ladder Logic Programming and Analysis Using Petri Nets. IEEE International Conference on Systems, Man., and Cybernetics. Outubro de 2002.

[40] Peng, Shih Sen; Zhou, Meng Chu; Ladder Diagram and Petri-Net-Based Discrete-Event Control Design Methods. IEEE Transactions on Systems, Man., and Cybernetics. Applications and Reviews. 2004.

[41] Aho, Alfred V.; Sethi, Ravi; Ullman, Jeffrey D.; Compiladores - Princípios, Técnicas e Ferramentas. Editora Guanabara Koogan S.A.. 1995. Tradução do original em inglês Compilers, Principles, Techniques and Tools. Editora Addison-Wesley. 1986.

[42] Meinel, Christoph; Theobald, Thorsten; Algorithms and Data Structures in VLSI Design. Springer-Verlag Berlin Heidelberg. 1998.

[43] Choo, Lim Gek; Devanathan, R.; Choo, Yun Kong; Development of a State Transition Diagram-Based Programmable Logic Controller Using Temporal Operators. Proceedings of the Singapore International Conference on Intelligent Control and Instrumentation. Fevereiro de 1992.

[44] Moraes, Cícero Couto; Castrucci, Plínio de Lauro; Engenharia de Automação Industrial. Editora LTC. 2001.

[45] Shannon, C.E.; Symbolic analysis of relay and switching circuits. Transactions of the American Institute of Electrical Engineers, vol. 57, pp. 713-723. Março de 1938.

[46] Boole, George; An Investigation into the Laws of Thought on wich are Founded the Mathematical Theories of Logic and Probabilities. Cambridge. Macmillan/London. Walton & Maberly. 1854. Reprinted by New York Dover, 1958.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo