**Geração de expressões algébricas
para processos de negócio usando
reduções de digrafos série-paralelo**

Márcio Katsumi Oikawa

TESE APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO
DE
DOUTOR EM CIÊNCIAS

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. João Eduardo Ferreira

São Paulo, julho de 2008

Geração de expressões algébricas de *workflows* usando reduções em digrafos série-paralelo

Este exemplar corresponde à redação
final da tese devidamente corrigida
e defendida por Márcio Katsumi Oikawa
e aprovada pela Comissão Julgadora.

Comissão Julgadora:

Prof. Dr. João Eduardo Ferreira (presidente)	IME-USP
Prof. Dr. Marcelo Finger	IME-USP
Prof. Dra. Marta Lima de Queiros Mattoso	COPPE-UFRJ
Prof. Dr. Duncan Dubugras Alcoba Ruiz	FACIN-PUC-RS
Prof. Dr. Luciano Antônio Digiampietri	EACH-USP

A meus pais e amigos,

Agradecimentos

Agradecer é reconhecer a importância e admirar a ajuda que nos é dada na busca de algum objetivo, do mais simples ao mais grandioso. Na trajetória de todos esses anos de dedicação, seria injusto deixar de compartilhar o resultado final deste trabalho com aqueles que direta ou indiretamente contribuíram com o crescimento deste trabalho e, conseqüentemente, com meu próprio crescimento. Infelizmente, ou felizmente, as minhas limitações de espaço e memória impedem que eu possa citar cada um dos muitos nomes que mereceriam estar presentes nestas linhas. Apesar disso, alguns nomes em especial não podem passar despercebidos por quem lê estes agradecimentos.

Agradeço a Deus acima de tudo, por todas as bênçãos que tenho experimentado por todos os anos, inclusive anteriores a este trabalho. Agradeço também ao Prof. Dr. João Eduardo Ferreira, meu professor e orientador, por sua amizade, seriedade e lealdade nestes longos anos de trabalho. Agradeço também a todos os amigos do laboratório de banco de dados, em especial Pedro Losco Takecian, Kelly Rosa Braghetto e David da Siva Pires, por sua amizade bondade e generosidade, sempre compartilhando momentos bons e ruins com muita alegria. Também agradeço a todos os meus insubstituíveis amigos do Futcerva, para os quais todos os votos de agradecimento são pouco perto de minha real gratidão.

Também não posso deixar de agradecer aos profs. Drs. José de Jesus Pérez-Alcazar, Siang Wun Song, Calton Pu, José Coelho de Pina Jr. por seus valiosos comentários e indicações. Separo uma especial referência aos profs. Drs. Marcelo Finger e Leliane Nunes de Barros, que acompanharam este trabalho desde praticamente seu início, sempre com disposição e boa-vontade nas horas em que as dúvidas eram freqüentes. Além de todos eles, agradeço também ao Ms. Devanir Zuliani, por compartilhar seu conhecimento no enriquecimento deste trabalho.

Não posso deixar de agradecer à minha namorada Andréa, por sua compreensão, paciência e companheirismo. Finalmente, agradeço emocionado à minha família, meus pais e irmãos, que sempre acreditaram e confiaram em mim, certos de que esses sentimentos são recíprocos. Termino agradecendo a todos os demais professores do IME-USP, amigos e aqueles cujo nome não pude citar, mas que sabem ter contribuído para que um dia eu pudesse ter a chance de fazer estes agradecimentos.

Resumo

Modelagem e controle de execução são duas abordagens do gerenciamento de processos de negócio que, embora complementares, têm se desenvolvido independentemente. Por um lado, a modelagem é normalmente conduzida por especialistas de negócio e explora aspectos semânticos do processo. Por outro lado, o controle de execução estuda mecanismos consistentes e eficientes de implementação. Este trabalho apresenta um método algorítmico que relaciona modelagem e controle de execução, por meio da geração de expressões algébricas a partir de digrafos acíclicos. Por hipótese, assumimos que modelos de processos de negócio são formados por estruturas baseadas em grafos, e mecanismos de controle de execução são baseados na interpretação de expressões de álgebra de processos.

Para a geração de expressões algébricas, esta tese apresenta as propriedades topológicas de digrafos série-paralelo e define um sistema de transformação baseado em redução de digrafos. Além disso, um algoritmo de identificação de digrafos série-paralelo e geração de expressões algébricas é apresentado. O texto também discute o tratamento de digrafos que não são série-paralelo e apresenta, para alguns desses casos, soluções baseadas em mudanças topológicas. Finalmente, o algoritmo é ilustrado com o estudo de caso de uma aplicação real.

Palavras-chave: geração automatizada de expressões algébricas, gerenciamento de processos de negócio, álgebra de processos, transformação de digrafos, digrafos série-paralelo.

Abstract

Modeling and execution control are complementary approaches of business process management that have been developed independently. On one hand, modeling is usually performed by business specialists and explores semantical aspects of the business process. On other hand, execution control studies consistent and efficient mechanisms for implementation. This work presents an algorithmic method which joins modeling and execution control through algebraic expression generation from acyclic digraphs. By hypothesis, we assume that business process models are defined by graph structures, and execution control mechanisms are based on interpretation of process algebra expressions.

For algebraic expression generation, this thesis presents the topological properties of series-parallel digraphs and defines a transformation system based on digraph reduction. Therefore, we present an algorithm for identification of series-parallel digraphs and generation of algebraic expressions. This work also discusses the treatment of non-series-parallel digraphs and presents solutions based on topological changing for some cases. Finally, the algorithm is illustrated with a case study based on a real system.

Keywords: automated generation of algebraic expressions, business process management, process algebra, digraph transformation, series-parallel digraphs.

Sumário

Lista de Abreviaturas	xiii
Lista de Símbolos	xv
Lista de Figuras	xvii
Lista de Tabelas	xxi
1 Introdução	1
1.1 Passos de Modelagem de Processos de Negócio	3
1.2 Revisão Histórica e Trabalhos Relacionados	5
1.3 Descrição do Problema e Objetivos	9
1.4 Contribuições	9
1.5 Organização do Trabalho	10
2 Grafos e Digrafos	11
2.1 Introdução	11
2.2 Definições	11
2.2.1 Digrafos Acíclicos	14
2.3 Digrafos Série-Paralelo	16
2.3.1 Definição	16
2.3.2 Árvore Binária de Decomposição	18
2.3.3 Digrafos Proibidos	18
2.3.4 Reconhecimento de SP-Digrafos	22
2.4 Conclusões	23
3 Álgebra de Processos	25

3.1	Introdução	25
3.2	Operadores Básicos	26
3.2.1	Regras de Transição	27
3.3	Paralelismo	28
3.4	Equivalência	29
3.4.1	Equivalência de Rastros	30
3.4.2	Equivalência de Simulação	30
3.4.3	Equivalência de Bissimulação	31
3.5	Concorrência de Termos e Equivalência	32
3.6	Axiomas da álgebra de processos	33
3.7	Deadlock e Encapsulamento	34
3.8	Conclusões	36
4	Sistemas Abstratos de Redução e Transformação de Grafos	37
4.1	Introdução	37
4.2	Sistemas Abstratos de Redução	37
4.3	Transformação de Grafos	40
4.3.1	Pushout Duplo	41
4.3.2	Independência de Derivações	46
4.3.3	Confluência e Terminação	47
4.4	Conclusão	50
5	Digrafos de Expressão	53
5.1	Introdução	53
5.2	Digrafos de Expressões Algébricas	53
5.2.1	Habilitação e Execução	57
5.3	Redução de Digrafos de Expressão	61
5.3.1	Confluência e Terminação	68
5.4	Conflitos Estruturais em SP-Digrafos de Processos	68
5.5	Conclusões	71
6	Geração de Expressões Algébricas	73

6.1	Introdução	73
6.2	Expressões em SP-Digrafos Bem-Formados	73
6.2.1	Tratamento da Entrada de Dados	74
6.2.2	Definição de Dependências	74
6.2.3	Refinamento de Dependências	76
6.2.4	Geração do Digrafo de Expressão	78
6.2.5	Expressões em SP-Digrafos Bem-Formados	80
6.2.6	Identificação de SP-Digrafos e Conflitos	84
6.3	Expressões em Pontes de Wheatstone Isoláveis	85
6.3.1	Casos com presença de Deadlock Total	88
6.3.2	Casos com presença de Deadlock Parcial	91
6.3.3	Casos com Falha de Sincronização	95
6.3.4	Casos com Ausência de Conflitos	101
6.4	Análise de Expressões Resultantes de Transformações	108
6.4.1	Análise de Equivalência de Transformações	109
6.5	Conclusões	113
7	Estudo de Caso	115
7.1	Introdução	115
7.2	DECA Eletrônica	115
7.2.1	DECA Eletrônica - Abertura Inicial	115
7.3	Conclusões	123
8	Conclusões	129
8.1	Principais Contribuições	130
8.2	Limitações do Trabalho	130
8.3	Trabalhos Futuros	131
A	Digrafos com Pontes de Wheatstone Não-Isoláveis	133
A.1	Introdução	133
A.2	Digrafos com Operadores Alternativos	134
A.3	Digrafos com Operadores Paralelos	137

A.4 Conclusões	141
Referências Bibliográficas	143

Lista de Abreviaturas

ACP	Álgebra de Processos Comunicantes (<i>Algebra of Communication Processes</i>);
APB	Álgebra de Processos Básica;
BPEL	<i>Business Process Execution Language</i> ;
BPM	Gerenciamento de processos de negócio (<i>Business Process Management</i>);
CCS	<i>Calculus of Communicating Systems</i> ;
CETESB	Companhia de Tecnologia de Saneamento Ambiental;
CRC	Conselho Regional de Contabilistas;
CSP	<i>Communicating Sequential Processes</i> ;
DECA	(Sistema de) Declaração Cadastral;
EST	Especificação de Sistema de Transição;
ICN	<i>Information Control Nets</i> ;
JUCESP	Junta Comercial do Estado de São Paulo;
LOTOS	<i>Language of Temporal Ordering Specifications</i> ;
NPDL	<i>Navigation Plan Definition Language</i> ;
PFE	Posto Fiscal Eletrônico;
SEFAZ-SP	Secretaria da Fazenda do Estado de São Paulo;
Sp-Digrafos	Grafos dirigidos de topologia série-paralelo;
STR	Sistema de Transições Rotuladas;
WfMC	<i>Workflow Management Coalition</i> .

Lista de Símbolos

Para uniformizar a notação dos símbolos, a lista considera a seguinte convenção:

A	: conjunto qualquer;
$\rightarrow$	: operação binária fechada sobre A ;
D, D_i e D_j	: digrafos acíclicos;
u, v	: vértices de um digrafo;
W	: conjunto de vértices;
m_1, m_2, l, r	: morfismos em grafos;
p_1, p_2	: produções sobre grafos;
t_1, t_2, t_3	: processos (termos da ACP);

Abaixo, a lista de símbolos utilizados neste texto:

$t_1 + t_2$	composição alternativa de t_1 e t_2 ;
$t_1 \cdot t_2$	composição seqüencial de t_1 e t_2 ;
$t_1 \parallel t_2$	composição paralela de t_1 e t_2 ;
$\pi(v)$	rótulo do vértice v em um digrafo de expressão;
$\theta(uv)$	rótulo do arco uv em um digrafo de expressão;
$d_D^+(v), d^+(v)$	grau de saída do vértice v ;
$d_D^-(v), d^-(v)$	grau de entrada do vértice v ;
D_L	digrafo de linha do digrafo D ;
$ D $	número de vértices de D (ordem);
$\ D\ $	número de arcos de D (tamanho);
K_n	grafo completo com n vértices;
$A(D)$	conjunto de arcos do digrafo D ;
$V(D)$	conjunto de vértices do digrafo D ;
$D - D_j$	digrafo formado pelos elementos de D que não estão em D_j ;
$uv, (u, v)$	arco entre os vértices u e v ;
$\mu(u, v)$	número de arcos entre u e v ;
$D_i = L(D)$	D_i é o digrafo de linha de D ;
$N^+(v)$	vizinhança externa do vértice v ;
$N^-(v)$	vizinhança interna do vértice v ;
$d^+(W)$	grau de saída do conjunto de vértices W ;
$d^-(W)$	grau de entrada do conjunto de vértices W ;

$pred(v)$	conjunto de vértices antecessores de v ;
$suc(v)$	conjunto de vértices sucessores de v ;
$m_1(D)$	vértices que são imagem de D_1 pela aplicação de m_1 ;
$D \xrightarrow{l} D_i$	um morfismo l que mapeia vértices de D em D_i ;
$D \rightarrow D_i$	um morfismo arbitrário que mapeia vértices de D em D_i ;
$T(p_1)$	conjunto de rastros de p_1 ;
$termo(D)$	expressão algébrica representada por D ;
$\langle A, \rightarrow \rangle$	sistema abstrato de redução;
$\rightarrow^0$	relação identidade de $\rightarrow$;
$\rightarrow^=$	fecho reflexivo de $\rightarrow$;
$\rightarrow^n$	composição de n operações $\rightarrow$;
$\rightarrow^+$	fecho transitivo de $\rightarrow$;
$\rightarrow^*$	fecho transitivo-reflexivo de $\rightarrow$;
$\leftrightarrow$	fecho simétrico de $\rightarrow$;
$\leftrightarrow^*$	fecho de equivalência de $\rightarrow$;
$D_i \xleftarrow{l} D \xrightarrow{r} D_j$	produção que transforma digrafo D_i no digrafo D_j usando o digrafo D como interface;
$l \circ r$	composição dos morfismos l e r ;
$D \xrightarrow{p_1, m_1} D_i$	derivação direta que transforma o digrafo D no digrafo D_i usando a produção p_1 sobre o morfismo m_1 ;
$D \xrightarrow{p_1} D_i$	derivação direta que transforma o digrafo D no digrafo D_i usando a produção p_1 ;
$D \rightarrow D_i$	morfismo que leva o digrafo D no digrafo D_i ;
$p \leftrightarrow t$	p e q são bissimilares;
$t_1 \xrightarrow{t_3} t_2$	transição de t_1 para t_2 com a execução de um termo t_3 ;
$t_1 \xrightarrow{t_3} \surd$	finalização bem sucedida de t_1 após a execução de um termo t_3 ;
$\eta(t_1, t_2)$	comunicação entre termos t_1 e t_2 ;
δ	<i>deadlock</i> (em ACP);
$\partial_H(t_1)$	aplicação do encapsulamento do conjunto H sobre t_1 ;
$\langle D, l, r \rangle$	<i>pushout</i> ;
$D \cong D_i$	D e D_i são isomórficos;

Lista de Figuras

1.1	Passos de modelagem conceitual de processos de negócio	4
1.2	Ciclo de vida de processos de negócio [1].	8
2.1	Exemplo de grafo	12
2.2	Grafos completos	12
2.3	Exemplo de digrafo	13
2.4	Um digrafo D e seu respectivo digrafo de linha $Q = L(D)$ (adaptado de [2])	14
2.5	Exemplo de sp-digrafo.	17
2.6	Árvore de decomposição do sp-digrafo da Figura 2.5.	18
2.7	Grafos não redutíveis a K_2 [3]	19
2.8	Subdigrafo proibido para a classe de sp-digrafos	19
2.9	Reduções sobre sp-digrafos: (a) redução seqüencial e (b) redução paralela	20
2.10	Padrão estrutural encontrado em pontes de <i>Wheatstone</i>	21
2.11	Disposição estrutural de um subdigrafo com uma ponte de <i>Wheatstone</i>	22
2.12	Alternativa mostrada quando os caminhos a e d não são disjuntos	22
3.1	Equivalência de Rastros	30
3.2	Equivalência de Simulação	31
4.1	Propriedades relacionadas à confluência de sistemas de redução.	40
4.2	Grafo rotulado.	42
4.3	Exemplo de produção.	42
4.4	<i>pushout</i>	43
4.5	<i>Pushout</i>	44
4.6	Derivação direta modelada como <i>pushout</i> duplo	45

4.7	Derivação modelada como <i>pushout</i> duplo.	45
4.8	Independência paralela de derivações diretas (Extraído de [4]).	47
4.9	Independência seqüencial de derivações diretas (Extraído de [4]).	47
4.10	Produções diferentes que atuam sobre o mesmo <i>redex</i> [5].	49
4.11	Par crítico formado pelas produções da Figura 4.10 [5].	49
5.1	Construção de sp-digrafos de expressão	56
5.2	Particionamento de vértices não-terminais	57
5.3	Digrafo de expressão para o termo $\theta(a) + \theta(b)$	58
5.4	Regras de habilitação de vértices e arcos	59
5.5	Seqüência de execuções em digrafos de expressão	60
5.6	Grafo de processos	61
5.7	Redução de termos em digrafos de expressão	63
5.8	Produção seqüencial	65
5.9	Produção alternativa	65
5.10	Produção paralela	66
5.11	Possíveis mapeamentos das produções paralela e seqüencial.	66
5.12	Par crítico formado por produções seqüenciais	67
5.13	Conflitos estruturais em digrafos de expressão	69
5.14	Ausência de sincronização em Processo de Negócio	70
6.1	Grafo de dependências gerado a partir da Tabela 6.1	75
6.2	Grafo de dependências com regras de ramificação e sincronização	78
6.3	Refinamento de dependências múltiplas em pontos de ramificação.	78
6.4	Geração do digrafo de expressão	81
6.5	Geração de expressões algébricas a partir de reduções em sp-digrafos	82
6.6	Ponte de <i>Wheatstone</i>	87
6.7	Digrafo acíclico com pontes de <i>Wheatstone</i> isoláveis	87
6.8	Casos com a presença de <i>deadlock</i> total	89
6.9	Grafo de Processos do Caso 1	90
6.10	Grafo de Processo do Caso 2	90
6.11	Grafo de Processo do Caso 3	91

6.12 Grafo de Processo do Caso 4	91
6.13 Casos com a presença de <i>deadlock</i> parcial	92
6.14 Grafo de Processo do Caso 5	92
6.15 Grafo de Processo do Caso 6	93
6.16 Transformação do Caso 6	94
6.17 Grafo de Processo do Caso 7	94
6.18 Grafo de Processo do Caso 8	95
6.19 Casos com falha de sincronização	96
6.20 Grafo de Processo do Caso 9	97
6.21 Grafo de Processo da Transformação do Caso 9	97
6.22 SP-digrafo resultado da transformação do Caso 9	97
6.23 Grafo de Processo do Caso 10	98
6.24 Grafo de Processo da Transformação do Caso 10	98
6.25 SP-digrafo resultado da transformação do Caso 10	99
6.26 Transformação do Caso 11	99
6.27 SP-digrafo resultado da transformação do Caso 11	100
6.28 Grafo de Processo do Caso 12	100
6.29 Grafo de Processo da Transformação do Caso 12	101
6.30 SP-digrafo resultado da transformação do Caso 12	101
6.31 Casos com ausência de conflitos estruturais	102
6.32 Grafo de Processo do Caso 13	103
6.33 Grafo de Processo da Transformação do Caso 13	104
6.34 SP-digrafo resultado da transformação do Caso 13	105
6.35 Grafo de Processo do Caso 14	105
6.36 Grafo de Processo da Transformação do Caso 14	106
6.37 SP-digrafo resultado da transformação do Caso 14	106
6.38 Transformação do Caso 15	106
6.39 SP-digrafo resultado da transformação do Caso 15	107
6.40 Grafo de Processo do Caso 16	107
6.41 Grafo de Processo da Transformação do Caso 16	108
6.42 SP-digrafo resultado da transformação do Caso 16	108

6.43	Caso 16: (a) Ponte de <i>Wheatstone</i> ; (b) Caso 16 desenhado como ordem topológica. . . .	111
6.44	Caso 16: Possíveis execuções do termo d	111
6.45	Caso 16: Possíveis execuções do rastro representado por d	112
7.1	Fluxograma da DECA Eletrônica (Adaptada de [6])	117
7.2	Digrafo de dependências obtido após fase de refinamento da DECA Eletrônica	124
7.3	Digrafo de Expressão para a DECA Eletrônica.	125
7.4	Reduções sobre o digrafo de expressão da DECA Eletrônica.	126
7.5	Reduções sobre o digrafo de expressão da DECA Eletrônica (continuação).	127
A.1	Digrafo de Fibonacci	134
A.2	Rede de comunicação com operadores alternativos	134
A.3	Composição paralela	137
A.4	Digrafo de Fibonacci	138
A.5	Digrafo de Fibonacci	139
A.6	Digrafo de Fibonacci de expressão com vértices paralelos	139
A.7	Geração de expressões em digrafos de Fibonacci.	140

Lista de Tabelas

3.1	Regras de transição para Álgebra de Processos Básica	28
3.2	Regras de transição para Álgebra de Processos com Paralelismo	28
3.3	Regras de transição para Comunicação em Álgebra de Processos com Paralelismo	28
3.4	Regras de transição para o entrelaçamento à esquerda	29
3.5	Regras de transição para o entrelaçamento com comunicação	29
3.6	Precedência de operadores em álgebra de processos	29
3.7	Axiomas da Álgebra de Processos Básica	33
3.8	Axiomas da Álgebra de Processos com Paralelismo	34
3.9	Regras de transição para Encapsulamento	35
3.10	Axiomas da álgebra de processos com <i>Deadlock</i> e Encapsulamento	35
5.1	Propriedades da ação vazia λ	55
6.1	Conjunto de dependências entre atividades	75
6.2	Conjunto de dependências de ramificação entre atividades	77
6.3	Conjunto de dependências de sincronização entre atividades	77
6.4	Equivalência das transformações	110
7.1	Relação das etapas da DECA Eletrônica	118
7.2	Relação de dependência entre passos de negócio da DECA Eletrônica	121

Capítulo 1

Introdução

A automatização de processos de negócio é uma área de pesquisa que tem despertado grande interesse nos últimos anos, tanto do ponto de vista prático quanto teórico. Os motivos para explicar tal interesse são diversos, seja pela necessidade de aplicações automáticas mais eficientes e confiáveis, por sua aplicabilidade a um conjunto considerável de tarefas em pequenas, médias e grandes corporações, ou mesmo pelo desafio que representa o tratamento de muitos de seus problemas. Podemos dividir essa área em duas grandes linhas de pesquisa: modelagem de processos de negócio e gerenciamento de controle de execução. Embora estejam intimamente relacionados, a modelagem e o controle de execução têm percorrido caminhos de pesquisa independentes nos últimos anos, criando uma considerável lacuna entre ambos.

A modelagem privilegia o conhecimento do negócio e têm como objetivo representar conceitualmente o processo, de forma clara, fácil e concisa. Esse trabalho é desempenhado, normalmente, por especialistas de negócio, com vasto conhecimento dos detalhes que envolvem as atividades que compõem e direcionam o processo de negócio. Os especialistas de negócio não precisam dominar detalhes de implementação do modelo, sendo suficiente conhecer plenamente as características funcionais do processo de negócio e de seus elementos. Por esse motivo, as principais características das ferramentas de modelagem de processos adotadas por especialistas de negócio são a sua simplicidade, facilidade de utilização e aprendizagem, além de um amplo poder gráfico de visualização, que agiliza a construção e compreensão dos modelos.

O controle de execução de processos, por outro lado, privilegia a consistência, eficácia e eficiência da execução dos processos de negócio, idealmente baseado em um modelo de negócios previamente construído. Não é raro, entretanto, encontrar processos de negócio implementados diretamente, ignorando a fase de modelagem. Principalmente em sistemas legados mais antigos, que sofreram pequenas e contínuas modificações durante o tempo, podemos encontrar implementações que não correspondem mais a seus projetos iniciais. Para o controle de execução, a prioridade é o aspecto computacional, de forma que não há exigência na adoção de linguagens gráficas próximas de usuários finais, mas de linguagens robustas, não-ambíguas e de eficiente manipulação computacional.

Este trabalho busca uma maneira de aproximar modelos de processos de negócio e seus respectivos modelos de controle de execução, por meio da aproximação de estruturas gráficas, mais próximas dos especialistas de negócio, e álgebras de processos, mais próximas do processamento computacional. Por hipótese, assumiremos que os modelos de negócio normalmente utilizam estruturas baseadas em grafos,

cuja representação visual é mais apropriada para caracterização de dependências e visualização estrutural do processo. Da mesma forma, assumiremos que o controle da execução de processos é baseado na manipulação de expressões em álgebras de processos, por julgá-las mais adequadas à manipulação computacional. Além desse aspecto, a álgebra de processos permite uma análise formal de propriedades que caracterizam a estruturas, o comportamento e a consistência do fluxo de execução dos processos.

Infelizmente, relacionar estruturas de grafos e expressões algébricas não é um problema simples. Em 2003, Aalst apresentou a limitação da capacidade de mapeamento entre estruturas algébricas e modelos de grafos, apontando problemas de representação de expressões algébricas para configurações particulares de redes de Petri [7]. De fato, não há provas de que o mapeamento de grafos em expressões algébricas seja completo, ou melhor, não há como afirmar que qualquer grafo possui uma expressão algébrica correspondente. Grande parte desse problema se deve às propriedades de cada uma das abordagens, principalmente em relação à representação de dependências e composição de elementos.

Muitos trabalhos exploram mecanismos de representação algébrica de grafos, como as gramáticas de grafos [8], que fornecem fundamentos formais para a definição de sistemas de reescrita de grafos [9] e hiper-grafos [10, 11]. Essas representações definem sistemas de transformação baseados em regras algébricas, as quais permitem classificar conjuntos específicos de grafos por sua aderência a esses sistemas. A transformação de grafos é bastante aplicada em problemas que envolvem a manipulação de expressões lógicas e matemáticas, como por exemplo, encontrar a forma normal de uma expressão [11]. Também há contribuições da utilização de transformação de grafos em diversas aplicações práticas, como a verificação de sistemas [12], a modelagem de linguagens gráficas de especificação de sistemas [13] e a modelagem de mecanismos de controle de acesso em bancos de dados [14].

Podem ser citados, como alguns dos principais problemas relacionados à geração de expressões a partir de especificações de processos de negócio, sistemas de documentação automática e tradução de especificação [15], sistemas de controle de comunicação entre processos [16] ou mesmo sistemas de geração automática de código [17]. Em todos esses casos, é possível identificar a necessidade de relacionar estruturas baseadas em grafos com linguagens algébricas ou de especificação de sistemas.

Este trabalho une modelos de processos baseados em grafos e expressões de álgebra de processos, adaptando transformações topológicas em grafos à composição de expressões algébricas. Para isso, é apresentada uma classe de grafos dirigidos (digrafos) acíclicos para os quais é viável e eficiente a aplicação de reduções topológicas: os grafos dirigidos série-paralelo (ou sp-digrafos). Além disso, é apresentada uma definição de grafos para representação de processos de negócio e uma adaptação das reduções topológicas que possibilita o tratamento de expressões em álgebra de processos. Um algoritmo de geração também é apresentado, bem como suas provas de corretude, confluência e terminação. As provas são baseadas em conceitos usados na abordagem algébrica de transformação de grafos, como produções, derivações diretas e *pushouts*.

Para processos de negócio que não podem ser modelados como sp-digrafos, este trabalho apresenta primeiro um algoritmo de identificação topológica para esses grafos e, em seguida, propõe formas de tratamento de alguns de seus conjuntos. Também são discutidos aspectos relacionados à conservação da equivalência entre expressões nesses casos, considerando os níveis de rastros, simulação e bissimulação.

Também é importante citar que este trabalho constitui uma das fases de modelagem de processos

de negócio descritas em [18] e comentadas na seção seguinte.

1.1 Passos de Modelagem de Processos de Negócio

Como já foi antecipado, este trabalho foi idealizado para tratar o problema de representar um modelo de processo de negócio baseado em grafos por meio de expressões algébricas. Nesta seção, descreveremos brevemente o contexto global que motivou este trabalho. Maiores detalhes podem ser encontrados na referência original [18].

Atualmente, o gerenciamento de processos de negócio têm adotado implementações sem o apoio da modelagem conceitual. Esse panorama têm aumentado a dificuldade de identificação, manutenção e reutilização de regras e processos. Com o objetivo de aproximar a modelagem e a implementação, foram definidos passos de modelagem de processos de negócio, cujas principais características são:

1. divisão da modelagem de processos de negócio em sete passos que caracterizam e organizam a construção do modelo de negócio;
2. geração de grafos de controle de fluxo a partir dos resultados de cada passo;
3. registro das operações de cada passo em tabelas e diagramas.

O processo de negócio pode ser visto com um fluxo de informação formado por ações, que se interrelacionam por meio de regras de composição [19]. As ações representam operações que encapsulam uma transformação de informação com entrada e saída bem-definidos. Enquanto isso, as regras representam meios de composição de ações que avaliam uma informação (produzindo um valor *verdadeiro* ou *falso*) e, a partir dela, direcionam adequadamente o fluxo de execução do processo de negócio. Para as regras, foi adotada a caracterização hierárquica definida pela arquitetura *Riverfish* [20, 21], que as classifica em:

- **restrição:** uma regra declarativa que expressa uma circunstância incondicional que deve ser satisfeita para continuidade do fluxo;
- **condição:** uma regra declarativa que, quando satisfeita, habilita alguma ação;
- **cálculo:** uma regra declarativa que, quando satisfeita, direciona o fluxo para a execução de algum algoritmo;
- **inferência:** uma regra declarativa que, quando satisfeita, incorpora propriedades para uma dada ação.

As regras são utilizadas, principalmente, para caracterizar os pontos de ramificação e sincronização do processo de negócio, reconhecendo os elementos críticos da modelagem do processo. A utilização de regras de ramificação e sincronização confere ao processo de negócio a capacidade de escolher dinamicamente o fluxo de execução baseado na avaliação de variáveis, ou disparar paralelamente várias linhas de execução. Notadamente, essas características aumentam consideravelmente o poder de representação e eficiência de processos de negócio. Mais à frente, no Capítulo 5, veremos com mais detalhes os conceitos relacionados à composição de processos e regras de ramificação e sincronização.

Finalmente, os passos de negócio são mostrados na Figura 1.1. A aplicação dos passos de modelagem delimita as fases nas quais as regras de negócio são definidas ou transformadas, estabelecendo uma divisão procedimental desde a definição dos passos de negócio até a geração de modelos computacionais. Organizado dessa forma, o modelo do processo de negócio facilita a manutenção de regras existentes e a inclusão de novas regras e passos de negócio, sem a obrigação de reconstruir integralmente o modelo.

Figura 1.1: Passos de modelagem conceitual de processos de negócio

Os passos podem ser resumidamente descritos como:

- **Passo 1:** O objetivo dessa fase é dividir o processo de negócio em fases, declarando em alto nível o objetivo de cada uma delas, e classificando-as de acordo com os critérios da arquitetura *Riverfish* (restrição, condição, cálculo ou inferência). A classificação de cada fase não tem a preocupação em identificar operadores, composições, restrições ou regras condicionais, mas somente dividir o processo de negócio em unidades lógicas com objetivo bem definido;
- **Passo 2:** Nesse passo, são analisadas as fases em que o fluxo pode ser interrompido, notadamente fases classificadas como restrições, condições e inferências. Nessas classificações, a continuidade do fluxo depende da satisfação de alguma condição. Quando essa condição não é satisfeita, o fluxo do processo deve ser encaminhado para alguma decisão, como uma nova solicitação de dados, ou a finalização do processo. O objetivo desse passo é complementar as fases de avaliação, quando necessário, para garantir a continuidade do fluxo desde seu início até o seu final;
- **Passo 3:** Esse passo identifica as relações de dependência entre as fases que compõem o processo de negócio. A identificação das dependências é uma das fases mais importantes da definição do processo de negócio, uma vez que ela determina a ordem de execução entre as fases do processo e, indiretamente, relaciona cada fase do processo de negócio com as demais;
- **Passo 4:** Esse passo gera, a partir das informações de dependência entre processos, um grafo de representação do processo. O grafo reúne, na mesma estrutura, as dependências entre fases catalogadas no Passo 3, permitindo uma caracterização global do processo de negócio. Este passo torna explícita a independência entre fases do processo de negócio e permite, ainda, fazer análises de consistência preliminares, como a presença de pontos desconexos no processo de negócio;

- Passo 5: Essa fase utiliza o grafo gerado no Passo 4 para produzir informações estruturais do processo de negócio. Além de identificar o comportamento dos pontos de ramificação e sincronização, esta fase realiza uma análise algébrica que permite verificar se pontos de ramificação e sincronização correspondentes são regidos pelas mesmas regras, o que caracteriza a presença ou ausência de problemas de consistência na execução do processo de negócio;
- Passo 6: Após a geração da estrutura do processo de negócio, o Passo 6 identifica os padrões de controle de fluxo que compõem a definição do processo;
- Passo 7: O Passo 7 utiliza a estrutura gerada no Passo 6 para alimentar ferramentas de análise de processos de negócio. Esta fase é responsável por rotinas de conversão e importação de dados para ferramentas de análise disponíveis no mercado.

Este trabalho atua sobre os Passos 3, 4 e 5, buscando automatizar a interpretação gráfica do processo identificada no Passo 3 e fornecida pelo Passo 4. Ao contrário da classificação manual dos pontos de ramificação e sincronização, este trabalho busca um algoritmo para realizar automaticamente essa operação. O papel do algoritmo será verificar se a composição estrutural pode originar falhas estruturais, e se sua organização permite a geração de expressões algébricas em tempo viável. Na presença de inconsistências insolúveis a intervenção do especialista de negócio é requisitada para revisão do modelo.

Na seção seguinte, é apresentada uma revisão histórica resumida da área de automatização de processos de negócio, com apontamentos de trabalhos relacionados. O objetivo é apresentar a contextualização histórica que conduziu a pesquisa de processos ao atual estágio, embasando a proposta deste trabalho.

1.2 Revisão Histórica e Trabalhos Relacionados

As primeiras referências de iniciativas de automatização de processos vêm da década de 70, por meio da introdução dos Sistemas de Informação de Escritório [22]. O ambiente de escritório convivia com um conjunto considerável de documentos, que seguiam uma determinada e bem-definida rotina desde sua entrada até o seu encaminhamento final. Esses sistemas de informação foram implantados como resposta ao desejo de eliminar o papel e automatizar grande parte do trabalho repetitivo de escritório.

Os Sistemas de Informação de Escritório apresentavam códigos embutidos, em que algumas tarefas eram organizadas de forma sequencial, com dados de entrada previamente definidos. O alto caráter estático e a aplicação de regras bem-definidas incentivou a busca de soluções para automatização desses sistemas.

Surgiram, então, as primeiras implementações e ferramentas de representação para esses sistemas. Nessa época, as redes de Petri já eram ferramentas úteis na representação de sistemas automatizados. O conceito de rede de Petri foi apresentado pela primeira vez por Carl Adam Petri, em 1962, em seu trabalho de doutorado (*Kommunikation mit Automaten*) [23]. Nele, Petri apresentou um novo modelo de representação de fluxos de informação baseado em grafos e estruturas de rede para representar sistemas complexos assíncronos e concorrentes. O modelo de redes de Petri se mostrou amplamente favorável à modelagem por sua capacidade de representação gráfica, além de algumas propriedades semânticas que permitiam estudar o comportamento da rede em situações diversas [24].

A pesquisa na área de processos apresentou uma notável expansão e despertou o interesse de muitos

pesquisadores da época. Um grande otimismo tomou conta da área, que vislumbrava soluções com os altos ganhos de produtividade e eficiência por meio da aplicação da nova tecnologia que surgia [25].

O otimismo inicial, entretanto, não se confirmou. Quando os primeiros sistemas foram implantados, muitos processos se tornaram extremamente intransigentes, forçando a execução de seqüências de tarefas fixas, pré-definidas e pouco flexíveis. Essa realidade retirou grande parte da liberdade dos funcionários, levando-os à insatisfação e posterior rejeição da nova tecnologia. Além disso, os recursos computacionais da época não eram os mais apropriados para muitos dos processos automatizados [22,26]. Dessa forma, esforços bem-sucedidos ocorriam somente em pontos isolados do trabalho de escritório, como o controle e processamento de textos, decepcionando os prognósticos iniciais.

Paralelamente, muitos pesquisadores procuravam melhorar o desenvolvimento da pesquisa na área de grafos e digrafos. A modelagem de algoritmos para resolução de problemas usando estruturas de grafos verificou considerável avanço nessa época. Particularmente, uma área de nosso interesse é a caracterização de classes de digrafos por meio da definição de seus grafos proibidos [3]. Alguns trabalhos adotaram essa linha, na Engenharia, para caracterizar e explorar as propriedades de grafos e digrafos série-paralelo [27,28].

Na década de 80, com o objetivo de melhorar os sistemas automatizados, muita ênfase foi dada à modelagem de fluxos de tarefas. Nesta época, surgiram os primeiros trabalhos sobre redes de Petri coloridas [29,30] e ICNs (*Information Control Nets*) [22]. Ambas as ferramentas tinham o objetivo de construir modelos de representação de controle associados a fluxos de dados. Esses novos modelos permitiam uma maior participação de usuários e projetistas na construção dos fluxos, buscando caminhos para flexibilizar uma ferramenta até então estritamente estática. Nesse período, surgiu a filosofia de que modelos de *workflow* (como passaram a ser chamados) deveriam ser suficientemente operacionais, de modo a permitir sua execução por sistemas computacionais, porém com componentes declarativos que permitissem modelar atividades não-estruturas, normalmente realizadas por seres humanos [31].

Também nessa década, muitos pesquisadores começaram a tratar processos como conjuntos com propriedades algébricas [32–37]. Nessa época, surgiram as primeiras propostas de expressões matemáticas representantes de conjuntos de processos. Tais propostas, após anos de pesquisa e desenvolvimento, dariam origem ao que hoje chamamos de álgebras de processos.

No início da década de 90, o interesse por *workflows* ressurgiu, motivado pela pesquisa de novas tecnologias de comunicação em rede e recursos de *hardware* mais poderosos e de fácil utilização. Além disso, a popularização dos computadores pessoais e o surgimento de uma nova geração de usuários ajudou a diminuir a resistência a sistemas automáticos. A evolução dos recursos computacionais e das interfaces de interação ajudaram a resolver vários problemas, porém explicitaram outros, como o reaproveitamento ou adaptação de sistemas legados a ambientes de maior complexidade e demanda de usuários. Grande parte da pesquisa na área de automatização passou a trabalhar com distribuição e escalonamento de instâncias, usando sistemas de *workflow* como um integrador de sistemas heterogêneos ou legados [38].

Com o desenvolvimento tecnológico, os requisitos necessários para automatização de processos de negócio se tornaram cada vez mais complexos, como transações longas, separação geográfica entre nós de processamento, e atividades com intervenções manuais. Esse panorama motivou cada vez mais a

pesquisa em áreas relacionadas ao estudo e implementação de processos de negócio. Sistemas automatizados começaram a desenvolver, além do caráter técnico, uma importância também gerencial. Além de despertar interesse em áreas ligadas à integração, escalabilidade, eficiência, controle e eficácia de sistemas, os sistemas automatizados também passaram a ser analisados do ponto de vista organizacional, motivando estudos de novas tecnologias gerenciais, aplicação de modelos estratégicos, análise de custo/benefício, impacto, aceitação de funcionários, investimentos e economia.

Uma importante contribuição no sentido de definir claramente o desenvolvimento da área de *workflows* foi a criação da *Workflow Management Coalition* (WfMC), como grupo de referência para a padronização de conceitos e metodologias relacionadas a sistemas de gerenciamento de *workflow*. A WfMC foi fundada por usuários, desenvolvedores, consultores, analistas e pesquisadores universitários interessados na pesquisa e aplicação de *workflows* e processos de negócios em problemas reais. Os principais objetivos da WfMC são aumentar o desenvolvimento e investimento em tecnologia de *workflows*, diminuir os riscos do uso de ferramentas baseadas em *workflows* e expandir o mercado para o uso de *workflows* por meio da conscientização de seus conceitos [39].

A pesquisa na área de *workflows* começava, então, a se desenvolver de maneira mais uniforme e as ferramentas e técnicas de análise ensaiavam uma conversão tecnológica. A ampla aplicabilidade de *workflows* também se expandiu para outras áreas de conhecimento, despertando novos desafios de projeto, modelagem e implementação. Um novo requisito que surgia era a necessidade de modelos mais flexíveis a mudanças ou adaptáveis a diferentes ambientes de *software* e *hardware*.

Problemas com essas características explicitaram cada vez mais o caráter altamente restritivo dos *workflows*, uma vez que sua aplicação bem-sucedida estava muitas vezes restrita a ambientes imutáveis, previsíveis e com baixa necessidade de intervenções humanas. Sendo assim, muitos pesquisadores perceberam a importância do trabalho com processos de negócio, que representam uma visão abstrata do *workflow*, relacionando-se mais com o comportamento da informação do que propriamente com a implementação de sua transformação.

Em [1], os autores definem *Gerenciamento de Processos de Negócio* (BPM) da seguinte maneira: “*Sistemas de apoio à execução de processos de negócio que utilizam métodos, técnicas e software para projetar, implementar, controlar e analisar processos operacionais envolvendo atores humanos, organizações, aplicações, documentos e outras fontes e informação*”.

A Figura 1.2 exibe o ciclo de vida de *workflows*, que ilustra a relação entre sistemas de gerenciamento de *workflow* e sistemas de gerenciamento de processos de negócio. Na fase de projeto, os processos de negócio são modelados. Em seguida, na fase de configuração, os processos são configurados por um sistema de informação. A fase de implementação sugere a implantação operacional do processo de negócio. A fase de diagnóstico identifica falhas ou alterações na definição inicial do processo de negócio, e habilita o seu reinício.

Analisando esse ciclo, podemos ver que o gerenciamento de *workflows* é responsável por menos da metade do gerenciamento de processos de negócio. Tarefas relacionadas à análise gerencial e validação semântica extrapolam o mérito de *workflows*, sendo realizados por níveis diferentes no processo de negócio. Sendo assim, podemos encarar o processo de negócio como uma extensão conceitual do *workflow*. Dessa forma, no nível de implementação e parte do projeto, podemos estender os conceitos válidos

Figura 1.2: Ciclo de vida de processos de negócio [1].

para *workflows* também para os processos de negócio.

Em 2003, o trabalho [40] apresentou uma nova iniciativa de categorização de componentes para *workflows*: os Padrões de Controle de Fluxo de *Workflow* (*Control-Flow Workflow Patterns*). Os Padrões de *workflow* são um conjunto de padrões construído por meio da análise de sistemas automatizados atualmente em produção. O objetivo desse conjunto de padrões é regular algumas das estruturas básicas que podem ser usadas para compor ou decompor *workflows*. A definição desses componentes visa facilitar a análise individual e coletiva de elementos internos dos *workflows*, de forma que seja possível melhorar técnicas de validação, simulação, análise de integridade e identificação de inconsistências ou exceções. Paralelamente ao estudo e especificação de padrões, a pesquisa em *workflows* e processos de negócio também presenciou o surgimento de problemas envolvendo ambientes dinâmicos de negócio e aplicações em domínios não-tradicionais, como, por exemplo, Ciências Naturais e automatização de ambientes laboratoriais [41]. Weske também discute o problema, enfatizando a necessidade de ferramentas mais flexíveis ou com capacidade de adaptação às mudanças [42]. Nesses ambientes, encontramos pelo menos um dos seguintes problemas: (a) o processo de negócio pode não ser totalmente conhecido, de modo que as especificações construídas são incompletas e necessitam de adaptações constantes; (b) ou o processo de negócio obedece a regras que podem ser alteradas com relevante índice de frequência.

Problemas envolvendo mudanças não são privilégio de aplicações em Ciências Naturais e laboratórios. Não é raro encontrar serviços com essas características em ambientes empresariais. Aplicações com necessidade de baixo tempo de resposta, ambientes móveis, internet e redes de comunicação sem fio são somente alguns dos muitos exemplos de ambientes sujeitos a constantes mudanças na lógica de processamento de informações.

Começaram a aparecer, então, as primeiras contribuições envolvendo a utilização de Planejamento [43,44] na construção dinâmica de *workflows* [45–47]. Os planejadores mais eficientes utilizam estratégia hierárquica [48,49], na qual é necessário que os elementos do processo de negócio mantenham, entre si, níveis explícitos de hierarquia. Mesmo assim, a solução de planejamento não é indicada em todos os casos, pois o resultado fornecido por um planejador é uma instância de execução do *workflow*. A geração de instâncias não é um caminho viável para a construção de especificações de modelos globais, visto que instâncias não capturam o comportamento global do sistema.

Enquanto isso, a pesquisa de linguagens algébricas para tratamento de processos de negócio também evoluiu, incorporando a representação de *deadlocks*, encapsulamento de transições e refinamento de critérios de equivalência entre processos. Muitas abordagens de controle de processos baseados em álgebra também surgiram, como o conceito de Plano de Navegação [50], um arcabouço teórico para

modelagem e controle de processos baseado na arquitetura *Riverfish* e controlado por meio de uma linguagem baseada em álgebra de processos [51].

É cada vez mais evidente que modelos gráficos e algébricos, originados de abordagens consideravelmente distintas, possuem características complementares. Este trabalho visa auxiliar a demanda por algoritmos e métodos que aproximem tais abordagens.

1.3 Descrição do Problema e Objetivos

Este trabalho procura responder à seguinte pergunta:

Dado um conjunto de dependências diretas entre pares de atividades que compõem um processo de negócio, é possível construir, em tempo viável, uma expressão algébrica que represente tal processo?

Em caso negativo, mostrar o porquê. Em caso positivo, apresentar a expressão algébrica correspondente.

Para responder à pergunta acima, algumas considerações são necessárias. A primeira delas se refere à escolha da álgebra que será usada para representar processos. Atualmente, há várias linhas de pesquisa que apontam para linguagens algébricas de representação de processos, como LOTOS [52,53], CSP [54], CCS [32,33], π -calculus [55] e ACP [56]. Cada linguagem apresenta diferentes conceitos, operadores e axiomas, de modo que sua capacidade e complexidade de representação podem apresentar grandes variações. Este trabalho adotará a ACP como linguagem de representação algébrica de processos, por sua simplicidade e pelo seu número considerável de desenvolvimento e publicações durante os últimos 30 anos.

Os objetivos deste trabalho são:

- Encontrar um método de tradução de modelos de processos de negócio baseados em grafos dirigidos acíclicos em expressões algébricas, conservando o maior nível possível de equivalência. Caso esse método não seja possível, ou viável, deseja-se encontrar:
 - (a) uma classe de digrafos acíclicos para os quais essa operação possa ser realizada em tempo polinomial;
 - (b) um algoritmo de classificação que defina, em tempo polinomial, se um digrafo acíclico pertence ou não à classe descrita em (a).

1.4 Contribuições

As principais contribuições deste trabalho são:

- Estudo sobre a influência topológica de digrafos acíclicos na geração de expressões algébricas;
- Definição de um método algorítmico para geração de expressões algébricas;
- Apresentação de fundamentos formais que comprovem a corretude e eficiência desse método algorítmico.

1.5 Organização do Trabalho

Os três próximos capítulos deste trabalho apresentam e discutem conceitos preliminares que serão usados no desenvolvimento do trabalho. O Capítulo 2 apresenta conceitos básicos de Teoria dos Grafos, com especial atenção para digrafos e sp-digrafos. Esse capítulo discute em detalhes as propriedades de sp-digrafos e apresenta um conjunto de reduções que pode ser utilizado como ferramenta de reconhecimento topológico de sp-digrafos. O Capítulo 3 apresenta conceitos básicos da álgebra de processos, como operadores, axiomas, relações de equivalência e representação de *deadlocks*. O Capítulo 4 apresenta a abordagem algébrica de transformação de grafos. Esse capítulo explora o conceito de regras de produção, derivações e *pushouts*, que fundamentam a definição de um sistema de transformação de grafos e permite um estudo algébrico de sua aplicação sobre um conjunto de grafos (ou digrafos).

Após os capítulos iniciais de fundamentos, são apresentados os capítulos que abordam o trabalho propriamente dito. O Capítulo 5 define uma modificação do conceito de digrafos para representação de termos e operadores algébricos usando rótulos de arcos e vértices, respectivamente. Esse digrafo, chamado digrafo de expressão, tem como objetivo a representação de processos de negócio usando grafos dirigidos acíclicos com informação algébrica. Esse capítulo também apresenta um sistema de redução de digrafos que adiciona valor semântico às reduções topológicas de sp-digrafos do Capítulo 2. Esse sistema de redução é usado posteriormente, no Capítulo 6 para construir um algoritmo de geração de expressões algébricas. O algoritmo consegue gerar expressões bissimilares à representação inicial do processo quando o digrafo de entrada é série-paralelo. Para digrafos que não são série-paralelo, é necessário fazer uma análise individual de cada construção, pois na maioria dos casos, elas apresentam problemas de consistência.

Os casos mais simples de digrafos que não são série-paralelo são estudados no Capítulo 6, que apresenta um conjunto de transformações topológicas que mapeiam digrafos que não são série-paralelo em sp-digrafos, permitindo, assim, a aplicação do algoritmo do Capítulo 6. Casos mais complexos não podem ser resolvidos por meio de reduções e são discutidos no Apêndice A.

Um estudo de caso que aborda a utilização do algoritmo de geração de expressões algébricas em um sistema real é apresentado no Capítulo 7. As conclusões deste trabalho são apresentadas no Capítulo 8.

Capítulo 2

Grafos e Digrafos

2.1 Introdução

A Teoria de Grafos é reconhecidamente uma das mais importantes áreas de pesquisa da Ciência da Computação, fornecendo um poderoso instrumento matemático de modelagem de problemas e desenvolvimento de soluções. A capacidade de abstração de grafos permite sua utilização nos mais variados tipos de problemas, mesmo àqueles não-relacionados à área de computação. Grafos podem ser informalmente definidos como conjuntos de elementos, chamados *vértices*, relacionados entre si por meio de arestas. Os grafos dirigidos, ou digrafos, são grafos especiais em que as arestas são direcionadas, permitindo a modelagem de relações de dependência entre vértices. Levando em conta sua capacidade visual de modelagem, a utilização de grafos em fluxogramas e modelos de processos de negócio pode ser considerada natural, de modo que o estudo das propriedades dos grafos auxilia na compreensão do próprio modelo analisado.

Neste trabalho, os grafos serão usados para representar modelos e propriedades de processos de negócio. O objetivo deste capítulo é fornecer uma visão resumida e consistente de conceitos e propriedades de grafos, que serão usados no restante do texto. Muitos dos conceitos e propriedades relacionados aqui serão fundamentais para o completo acompanhamento das estruturas que veremos nos capítulos seguintes. Além disso, o capítulo também apresenta a classe dos digrafos série-paralelo, que será amplamente explorada neste trabalho para caracterizar uma estrutura topológica da qual é possível extrair expressões algébricas de forma eficiente.

O conteúdo deste capítulo é um resumo das definições básicas de grafos apresentadas por Diestel [57], de digrafos, apresentadas por Bang-Jensen e Gutin [2] e uma coletânea de trabalhos sobre digrafos série-paralelo [2, 3, 28, 58–62]. O restante do capítulo é organizado com uma seção de definições (Seção 2.2), uma seção dedicada a digrafos série-paralelo, Seção 2.3, e uma seção com as conclusões (Seção 2.4).

2.2 Definições

Um **grafo** é um par $G = (V, E)$, que consiste num conjunto finito V de elementos chamados **vértices** e um conjunto finito $E \subseteq V^2$, de pares ordenados de vértices distintos, chamados **arestas**. Afim de evitar ambigüidades notacionais, sempre será assumido que $V \cap E = \emptyset$. O conjunto de vértices de G é representado por $V(G)$, assim como o seu conjunto de arestas é representado por $E(G)$. A **ordem** de

um grafo é a contagem de seu número de vértices, representado por $|G|$. O tamanho de um grafo é a contagem de seu número de arestas, representado por $\|G\|$. Muitos grafos são classificados como *finitos*, *infinitos* ou *contáveis* pela análise de sua ordem. O *grafo trivial* é um grafo de ordem 0 ou 1.

A Figura 2.1 apresenta um exemplo de grafo, que chamaremos de G . Neste exemplo, $V(G) = \{a, b, c, d, e\}$ e $E(G) = \{ab, ac, bd, cd, ce, de\}$. A ordem $|G|$ é 5, e seu tamanho $\|G\|$ é 6.

Figura 2.1: Exemplo de grafo

Uma aresta é representada por um par de vértices $e=(v,w)$, ou simplesmente $e = vw$. Em grafos não-dirigidos, como o da Figura 2.1, a ordem dos vértices em um arco é indiferente, ou seja, $e = (v, w) = (w, v)$. Nesse caso, v e w são chamados extremos de e . Um vértice v é incidente a uma aresta e se é um extremo de e . Dois vértices $v, w \in V(G)$ são chamados adjacentes se existe uma aresta vw em $E(G)$. Duas arestas e_1 e e_2 são chamadas adjacentes se possuem um vértice extremo em comum.

Um grafo G é chamado completo quando, para todo par de vértices u e v de $V(G)$, o arco uv está em $E(G)$. Os grafos completos são também chamados **cliques**. As cliques são representadas por K_n , onde n é seu número de vértices. A Figura 2.2 mostra os cliques com 1, 2, 3 e 4 vértices.

Grafos permitem relacionar elementos de um mesmo conjunto por meio de relações matemáticas. É possível, ainda, atribuir a elementos de um grafo algumas características particulares, que o tornam mais apropriado para modelagem de determinados problemas. Desse modo, surgiram extensões do conceito de grafos, como hipergrafos, multigrafos e digrafos dirigidos. Como este trabalho explora bastante propriedades relacionadas a grafos dirigidos, eles serão discutido com um pouco mais de detalhes.

Um **grafo dirigido** ou **digrafo** é definido como um grafo em que todas as arestas são direcionadas. Nesse caso, falamos em arcos, ao invés de arestas. Formalmente, o digrafo $D = (V, A)$ consiste num conjunto finito V de elementos chamados **vértices** e um conjunto finito A , de pares ordenados de vértices distintos, chamados **arcos**. Usaremos a notação $V(D)$ e $A(D)$ para representar, respectivamente, os conjuntos de vértices e arcos de D , quando estes não forem evidentes. Chamamos de **ordem** de D o

Figura 2.2: Grafos completos

número de elementos em $V(D)$, representado por $|D|$, e de **tamanho** o número de elementos de $A(D)$, representado por $\|D\|$.

Para ilustrar melhor, vamos acompanhar a Figura 2.3, que exibe um digrafo D , com estrutura semelhante ao grafo da Figura 2.1. Na figura, $V(D) = \{a, b, c, d, e\}$ e $A(D) = \{(a, b), (c, a), (b, d), (e, d), (d, c), (c, e)\}$. A ordem de D é 5, e seu tamanho é 6. Neste exemplo, ao contrário da Figura 2.1, a ordem dos vértices em cada arco é importante. Dessa maneira, temos que $(a, b) \neq (b, a)$; o mesmo valendo para os demais arcos.

Figura 2.3: Exemplo de digrafo

Em um digrafo, dizemos que o arco (u, v) relaciona os vértices u e v , de modo que u **domina** v , ou v é **dominado** por u . Chamamos essa relação de **dominância** e, em alguns casos, podemos representá-la como uma relação de ordem, afirmando que u é **antecessor** de v , ou v é **sucessor** de u . Graficamente, esta relação é representada por $u \rightarrow v$. Nesse caso, dizemos que u e v são, respectivamente, a **cauda** e a **cabeça** do arco (u, v) . Dois vértices u e v , de D , são ditos adjacentes se, ou $(u, v) \in A(D)$, ou $(v, u) \in A(D)$. O número de arcos entre dois vértices u e v , em D , é representado por $\mu(u, v)$. Se $\mu(u, v) > 1$ para algum par de vértices u, v , dizemos que D é um **multidigrafo**.

Um digrafo H é chamado **subdigrafo** do digrafo D se $V(H) \subseteq V(D)$, $A(H) \subseteq A(D)$ e todo arco em $A(H)$ possui suas extremidades em $V(H)$. Se $V(H) = V(D)$, dizemos que H é um **subdigrafo gerador** de D . Se todo arco de $A(D)$, com extremidades em $V(H)$, também está em $A(H)$, dizemos que H é subdigrafo induzido de D . Se D não contém um digrafo H' como um subdigrafo induzido, dizemos que H' é um **subdigrafo proibido** de D , e D é **livre de H'** .

Um **torneio** é um digrafo em que todo par de vértices u, v de $V(D)$, com $u \neq v$, é adjacente. Em outras palavras, um digrafo D com $V = \{v_1, v_2, \dots, v_n\}$ é um torneio se exatamente um dos arcos $v_i v_j$ ou $v_j v_i$ está em $A(D)$ para todo $i, j \in \{1, 2, \dots, n\}$, com $i \neq j$.

Para um digrafo D , o **digrafo de linha** $Q = L(D)$ é um digrafo com o conjunto de vértices $V(Q) = A(D)$, e com conjunto de arcos $A(Q) = \{ab : a, b \in V(Q), \text{ a cabeça de } a \text{ coincide com a cauda de } b\}$. A Figura 2.4 exibe um exemplo da construção de um digrafo de linha, onde $Q = L(D)$. Note que, para cada arco de D , existe um vértice em Q . Os arcos de Q são construídos unindo vértices em que o início e o final dos rótulos é idêntico.

Outro conceito de grande importância é a **vizinhança**. Vértices adjacentes também são chamados **vizinhos**, que podem ainda ser classificados em internos e externos. O conjunto $N_D^+(v) = \{w \in V - v : vw \in A\}$ é chamado **vizinhança externa** e agrupa os vértices dominados por v . Da mesma

Figura 2.4: Um digrafo D e seu respectivo digrafo de linha $Q = L(D)$ (adaptado de [2])

forma, $N_D^-(v) = \{u \in V - v : uv \in A\}$ é chamado **vizinhança interna** e representa o conjunto dos vértices que dominam v . A **vizinhança** N_D de v é o conjunto dos vértices adjacentes a v , ou seja, $N_D(v) = N_D^-(v) \cup N_D^+(v)$.

O **grau de saída** de um vértice, $d_D^+(v)$, é definido como o número de arcos com cauda em v e o **grau de entrada** $d_D^-(v)$ é o número de arcos com cabeça em v . O grau de um vértice $d_D(v)$ é a soma de seus graus de entrada e saída. Vértices com grau de saída igual a zero são chamados *sorvedouros* e vértices com grau de entrada igual a zero são chamados *fontes*. Vértices que são fonte ou sorvedouro são chamados **terminais**, e todos os demais vértices são chamados **internos**, ou **não-terminais**. O conceito de grau também pode ser estendido para conjuntos de vértices. Seja W um conjunto de vértices, tal que $W \subseteq V$. O grau de saída $d_D^+(W) = |\{uv \in A : u \in W, v \in V - W\}|$ é a soma dos arcos com cauda em W e cabeça em $V - W$. Da mesma maneira, o grau de entrada $d_D^-(W) = |\{uv \in A : u \in V - W, v \in W\}|$ é a soma dos arcos com cauda em $V - W$ e cabeça em W . O grau d_D de W é definido como $d_D(W) = d_D^+(W) + d_D^-(W)$.

Um passeio P , em D , é uma sequência alternada de vértices v_i e arcos a_j , da forma $P = v_1 a_1 v_2 \dots v_{k-1} a_{k-1} v_k$, onde a cauda de a_i é v_i e a cabeça de a_i é v_{i+1} , para todo $i = 1, 2, 3, \dots, k-1$. O passeio P é fechado se $v_1 = v_k$ e aberto caso contrário. O conjunto de vértices $\{v_1, v_2, v_3, \dots, v_k\}$ é denotado por $V(P)$ e o conjunto $\{a_1, a_2, a_3, \dots, a_{k-1}\}$ é denotado por $A(P)$. Dizemos que P é um passeio de v_1 a v_k , ou um **passeio**-(v_1, v_k). Quando P é aberto, v_1 é chamado vértice inicial e v_k , vértice final. Em alguns casos, quando arcos possuem importância secundária no modelo representado pelo digrafo, podemos escrever o passeio como sua sequência de vértices $v_1 v_2 v_3 \dots v_k$. O **comprimento** do passeio é seu número de arcos. Dizemos que o passeio tem comprimento **par (ímpar)** se k é par (ímpar).

Uma **trilha** é um passeio em que todos os arcos são distintos entre si. Um **caminho** é uma trilha com os vértices distintos entre si. Um **caminho**-(v_1, v_k) é um caminho entre v_1 e v_k . O ciclo é um caminho em que $k \geq 3$ e $v_1 = v_k$. O comprimento de trilhas, caminhos e ciclos segue o mesmo critério adotado para passeios.

2.2.1 Digrafos Acíclicos

Digrafos acíclicos podem ser definidos simplesmente como digrafos que não possuem ciclos. Os digrafos acíclicos constituem uma importante classe dentro dos digrafos, pois um digrafo acíclico determina um conjunto parcialmente ordenado, do qual podem ser inferidas importantes propriedades. A primeira delas é a presença de pelo menos um vértice fonte e pelo menos um vértice sorvedouro.

Proposição 2.2.1. (*Presença de Fonte e Sorvedouro [2]*) *Todo digrafo acíclico possui, pelo menos, um vértice com grau de entrada igual a zero e, pelo menos, um vértice com grau de saída igual a zero.*

Demonstração. (*Contradição sobre a ausência de ciclos*). Seja D um digrafo em que todos os vértices possuem grau de saída positivos. Mostraremos que D possui um ciclo. Escolha um vértice em D . Como $d^+(v_1) > 0$, existe um vértice v_2 , tal que $v_1 \rightarrow v_2$. Como $d^+(v_2) > 0$, v_2 domina um vértice v_3 . Procedendo dessa maneira, chegaremos a um caminho $v_1 v_2 v_3 \dots v_k$. Como $V(D)$ é finito, existe $k > 2$, tal que $v_k = v_i$, para algum $1 < i < k$. Claramente, $v_i v_{i+1} \dots v_k$ é um ciclo. ■

Em todo digrafo acíclico, é possível descobrir uma ordem acíclica de seus vértices. Chamamos de ordenação acíclica uma seqüência de vértices $O = v_1 v_2 \dots v_n$ de D , tal que para todo arco $(v_i, v_j) \in A(D)$, v_i vem antes de v_j na seqüência O . Informalmente, uma ordenação acíclica representa uma disposição topológica em que todos os vértices são colocados em seqüência, da esquerda para a direita por exemplo, e todos os arcos de D tem o mesmo sentido (da esquerda para a direita). Dessa forma, do ponto de vista de modelagem, encontrar uma ordenação acíclica significa encontrar um modelo em que os vértices apresentam, entre si, uma relação de ordem total ou parcial.

Proposição 2.2.2. (*Ordenação Acíclica [2]*) *Todo digrafo acíclico possui uma ordenação acíclica.*

Demonstração. Mostraremos uma prova construtiva, descrevendo os passos para obtenção da ordem acíclica de vértices de um digrafo acíclico D .

Escolha um vértice v , com grau de entrada zero (pela Proposição 2.2.1, este vértice existe). Faça $x_1 = v$ e remova x_1 de D . Repetindo sistematicamente a mesma operação, o i -ésimo passo achará um vértice u , para o qual fará $x_i = u$, e removerá x_i em seguida.

Resta mostrar que, se o arco ij está em D , necessariamente $i < j$. Então, suponha que, para algum arco ij em D , $i > j$. Como x_j foi escolhido antes de x_i , x_j não possuía grau de entrada zero no j -ésimo passo, o que é uma contradição.

Com isso, obteremos uma ordenação acíclica $x_1 \dots x_i$, $1 < i < |V|$. ■

Em digrafos acíclicos com uma única fonte e um único sorvedouro, existe uma interessante característica que relaciona seus vértices internos e terminais. Nesses casos, sempre é possível encontrar um caminho entre os terminais. Além disso, todo vértice interno está presente em pelo menos um desses caminhos, como mostra a proposição a seguir.

Proposição 2.2.3. (*Caminhos entre terminais [2]*) *Seja D um digrafo acíclico com uma única fonte x e um único sorvedouro y . Para todo vértice $v \in V(D)$, existe um caminho- (x, v) e um caminho- (v, y) em D .*

Demonstração. Vamos provar que, para um digrafo acíclico D com uma fonte x , existe um caminho- (x, v) para qualquer vértice $v \in V(D)$. A prova para existência de um caminho entre v e o sorvedouro y

é análoga. A prova será feita por indução sobre a existência de caminhos entre vértices. Para o digrafo trivial $V(D) = \{x\}$, existe um caminho entre a fonte x e todos os vértices de $V(D)$, que é o próprio x .

Suponha, agora, o digrafo $D - \{v\}$, que também é acíclico e possui uma única fonte. Seja D' o digrafo formado pela inclusão de v a $D - \{v\}$, além da inclusão de um conjunto qualquer de arcos.

Certamente v será dominado por algum vértice u , pois, caso contrário, v seria uma outra fonte de D' . Por hipótese de indução, para todo $u \in V(D - \{v\})$, existe um caminho (x, u) . Se usarmos este caminho para sair de x e chegar a u , podemos chegar em v percorrendo mais um arco. Portanto, existe um caminho (x, v) em D' . ■

É bastante útil, em determinadas situações, caracterizar a transitividade de um digrafo. Um digrafo D é chamado *transitivo* se, para todos os arcos uv e vw em $A(D)$, com $u \neq w$, o arco uw também está em $A(D)$. A proposição seguinte mostra que todo torneio acíclico também é, necessariamente, transitivo.

Proposição 2.2.4. *Todo torneio acíclico é transitivo.*

Demonstração. A prova dessa proposição será feita por contradição sobre a presença de ciclos. Suponha que D é um torneio acíclico que não é transitivo. Sendo assim, existem três vértices u , v e w em $V(D)$, tal que $uv, vw \in A(D)$, mas uw não. Com isso, o arco wu está em $A(D)$ e participa do ciclo $uvwu$, o que é uma contradição, pois D é acíclico. ■

Dentro do conjunto de digrafos acíclicos, este trabalho está particularmente interessado na classe dos digrafos série-paralelo, que se mostrarão úteis na modelagem de processo de negócio. Esses digrafos, bem como algumas de suas principais propriedades, serão apresentados e discutidos a seguir.

2.3 Digrafos Série-Paralelo

Digrafos série-paralelo, ou **sp-digrafos**, representam uma classe de digrafos amplamente adotada em aplicações envolvendo sistemas e modelos reais. O conceito inicial de sp-digrafos é atribuído a Duffin, como o resultado de seus estudos sobre classes de conexões favoráveis à elaboração de circuitos elétricos [27]. Posteriormente, novas pesquisas exploraram outras aplicações de suas características topológicas, desenvolvendo modelos e algoritmos para o tratamento de outros problemas, como escalonamento de tarefas [63,64] e manipulação de redes PERT (*Program Evaluation and Review Technique*) [65,66], entre outros.

2.3.1 Definição

Na literatura, é possível encontrar definições ligeiramente divergentes para caracterizar sp-digrafos. A caracterização dos sp-digrafos pode ser baseada na topologia de seus vértices ou arcos. Historicamente, a caracterização inicial feita por Duffin foi baseada nas características topológicas dos arcos. Posteriormente, Valdes estendeu esse conceito para vértices [58]. Essa nova definição dividiu o conceito de sp-digrafos, caracterizando duas novas classes de digrafos: os **sp-vértice-digrafos** e os **sp-arco-digrafos**. Este trabalho se restringe à utilização de sp-arco-digrafos, de modo que toda referência

posterir a sp-digrafos estará, na verdade, relacionada a sp-arco-digrafos. Na literatura, essa classe de digrafos pode ainda ser encontrada com outras nomenclaturas, digrafos série-paralelo com dois terminais (*two-terminal series-parallel digraph*), ou ainda digrafos com dois terminais [28].

A definição de sp-digrafos é recursiva e mostrada a seguir:

Definição 2.3.1: (*Digrafos Série-Paralelo (SP-Digrafos)*)

1. Um digrafo formado por dois vértices unidos por um único arco é um sp-digrafo (chamado *sp-digrafo trivial*);
2. Sejam D_1 e D_2 sp-digrafos e, s_i e t_i , respectivamente, os vértices fonte e sorvedouro de D_i ($i = 1, 2$). Também será um sp-digrafo o resultado da aplicação de uma das duas composições a seguir:
 - (a) *Composição paralela:* Faça $s_1 = s_2$, e $t_1 = t_2$;
 - (b) *Composição seqüencial:* Faça $t_1 = s_2$.

Um exemplo de sp-digrafo pode ser visto na Figura 2.5. A composição paralela produz um sp-digrafo criando uma identidade entre as fontes e os sorvedouros de dois sp-digrafos. A composição seqüencial, por outro lado, cria uma identidade entre a fonte de um sp-digrafo e o sorvedouro de outro. Entende-se por criação de identidade uma operação que unifica dois vértices.

Um sp-digrafo possui apenas uma fonte e um sorvedouro. A demonstração desse fato é simples, pois o digrafo trivial possui somente uma fonte e um sorvedouro. Além disso, o resultado de uma composição, seja ela seqüencial ou paralela, também possui uma fonte e um sorvedouro.

Também podemos ver que os sp-digrafos são acíclicos, pois o sp-digrafo trivial é acíclico e as composições não criam ciclos quando aplicadas a digrafos acíclicos.

Figura 2.5: Exemplo de sp-digrafo.

A definição de sp-digrafos permite que dois vértices sejam unidos por vários arcos. Por este motivo, alguns autores usam o termo sp-multidigrafos [2]. Este trabalho, por simplicidade, não fará distinção entre digrafos e multidigrafos.

2.3.2 Árvore Binária de Decomposição

A árvore binária de decomposição é uma estrutura que torna possível observar a sequência de transformações usadas na construção de um determinado sp-digrafo. Em uma árvore binária de decomposição, os nós-folha são formados pelos arcos do sp-digrafo, enquanto os nós internos indicam uma operação de composição (seqüencial ou paralela) que é aplicada sobre as sub-árvores esquerda e direita.

A árvore de decomposição é utilizada em muitos algoritmos de identificação de sp-digrafos e casamento de sub-árvores. Além de sua utilização na otimização de alguns algoritmos de reconhecimento, a árvore de decomposição também serve com certificado da topologia de um sp-digrafo. A Figura 2.6 mostra a árvore binária de decomposição para o sp-digrafo da Figura 2.5.

Figura 2.6: Árvore de decomposição do sp-digrafo da Figura 2.5.

2.3.3 Digrafos Proibidos

É possível caracterizar uma classe de grafos (e particularmente, digrafos) por meio de conjuntos de subgrafos que não são membros dessa mesma classe. Tais (di)grafos são chamados (di)grafos proibidos, e sua caracterização é um objetivo bastante comum em Teoria de Grafos [61]. A caracterização de sub(di)grafos proibidos possui alguns trabalhos importantes, dentre os quais podemos destacar o Teorema de Kuratowski, em relação a grafos planares. Pelo Teorema de Kuratowski, um grafo é planar se, e somente se, não contém um subgrafo que é homomórfico a K_5 ou $K_{3,3}$ [57].

Figura 2.7: Grafos não redutíveis a K_2 [3]

A Figura 2.7 apresenta classes de grafos proibidos para a classe de grafos redutíveis a K_2 , retirados de [3]. Estendendo esse trabalho, podemos mostrar que o subdigrafo proibido, cuja ausência caracteriza a classe de sp-digrafos, é formado por um subdigrafo de interdição¹, ou ponte de *Wheatstone*, que pode ser visto na Figura 2.8. A notação $\bullet \rightsquigarrow \bullet$ indica um caminho entre dois vértices.

Figura 2.8: Subdigrafo proibido para a classe de sp-digrafos: (a) digrafo redutível à ponte de *Wheatstone*; (b) ponte de *Wheatstone*.

Para provar que a ponte de *Wheatstone* é o subdigrafo proibido para a classe de digrafos série-paralelo, precisamos definir as reduções paralela e seqüencial para digrafos. As reduções podem ser vistas como “operações inversas” às composições, apresentadas na definição de sp-digrafos. Tal como na construção de um sp-digrafo, reduções também são definidas como série e paralelo, e aplicadas repetidamente sobre o digrafo (veja a Figura 2.9). O resultado final é um digrafo **reduzido**, no qual nenhuma redução pode ser aplicada.

As reduções em sp-digrafos são definidas da seguinte forma:

Definição 2.3.2: (*Reduções em sp-digrafos* [28]) Seja D um sp-digrafo e $u, v, w \in V(D)$, as reduções são definidas como:

- (a) *redução seqüencial*: substituição de um caminho uvw , tal que $d^+(v) = d^-(v) = 1$, por um arco uw (Figura 2.9(a)).
- (b) *redução paralela*: substituição de um par de arcos paralelos uv por um único arco uv (Figura 2.9(b)).

A proposição a seguir é usada para mostrar que a aplicação de uma redução não exclui uma ponte de *Wheatstone* do digrafo, quando ela existir.

Proposição 2.3.1. *Seja D um digrafo acíclico. Se D possui uma ponte de *Wheatstone*, a aplicação de uma redução sobre D também possuirá uma ponte de *Wheatstone*.*

¹Tradução para o termo *interdictive graph* [28]

Figura 2.9: Reduções sobre sp-digrafos: (a) redução seqüencial e (b) redução paralela

Demonstração. Dividiremos a prova da proposição em duas partes, destinadas a demonstrar individualmente a validade da proposição para as duas possíveis reduções:

Redução Seqüencial: Seja D_1 um digrafo acíclico isomórfico a uma ponte de *Wheatstone* e D um digrafo que possui um subdigrafo acíclico isomórfico a D_1 . Não existe, em D_1 , nenhum vértice $v \in V(D_1)$, tal que $d^+(v) = d^-(v) = 1$, portanto não é possível aplicar uma redução seqüencial sobre D_1 . Sendo assim, a aplicação de uma redução seqüencial sobre D , quando possível, também conserva o subdigrafo isomorfo a D_1 .

Redução Paralela: Seja D_1 um digrafo acíclico isomórfico a uma ponte de *Wheatstone* e D um digrafo acíclico que possui um subdigrafo isomórfico a D_1 . Não existe, em D_1 , nenhum par de vértices $u, v \in V(D_1)$, tal que haja mais de um arco uv , portanto não é possível aplicar uma redução paralela sobre D_1 . Sendo assim, a aplicação de uma redução paralela sobre D , quando possível, conserva o subdigrafo isomorfo a D_1 . ■

Chamamos de subdigrafo **reduzido** D_R o digrafo minimal obtido pela aplicação de reduções paralelas e seqüenciais sobre um digrafo D . Nesse digrafo, todo vértice $v \in V(D_R) : d_{D_R}^+(v) > 0$ e $d_{D_R}^-(v) > 0$, necessariamente apresenta pelo menos uma das características abaixo:

- (a) possui pelo menos dois sucessores distintos;
- (b) possui pelo menos dois antecessores distintos;

Por hipótese, D_R é reduzido. Caso um vértice, digamos v , não apresente nenhuma das características acima, então será possível fazer mais uma redução sobre v e sua vizinhança, o que é uma contradição.

Vamos chamar de **vértice de divergência** um vértice com dois, ou mais, sucessores distintos; e de **vértice de convergência** um vértice com dois, ou mais, predecessores distintos.

Lema 2.3.2. *Seja D_R o subdigrafo reduzido de um digrafo D , acíclico, não-isomórfico ao sp-digrafo trivial, com uma única fonte e um único sorvedouro. Existe, em D_R , um vértice de convergência que é sucessor de um vértice de divergência.*

Demonstração. A demonstração será feita por construção do conjunto de vértices de divergência. Seja S um conjunto formado somente por vértices de grau de saída maior que 1, ou seja, $S = \{v \in V(D_R) : d_{D_R}^+(v) > 1 \text{ e } d_{D_R}^-(v) = 1\}$. Mostraremos que existe um vértice w , sucessor de algum vértice $v \in S$, que é vértice de convergência.

Antes, porém, temos que garantir que S não é vazio. Esta observação pode ser confirmada percorrendo um dos caminhos a partir da fonte do digrafo. Se o vértice fonte possui dois ou mais sucessores,

a fonte já pode ser usada como elemento de S . Caso a fonte possua somente um sucessor, digamos u , certamente u possuirá dois ou mais sucessores, uma vez que o único predecessor de u é a fonte de D_R .

Sendo assim, podemos sempre construir, a partir de S , um conjunto S' , tal que $S' = S \cup \{v \in V(D_R) \setminus V(S) : \exists u \in S, \text{ tal que } uv \in A(D_R)\}$, ou seja, acrescentando um vértice de divergência v em S . Se aumentarmos sistematicamente o conjunto S levando em conta este critério, veremos que, em algum momento encontraremos, em S' , um vértice de convergência, pois o número de vértices é finito e D_R possui somente um sorvedouro. ■

A Proposição 2.3.1 e o Lema 2.3.2 nos ajudarão a provar o teorema a seguir, base fundamental do tratamento de sp-digrafos que será proposto neste trabalho.

Teorema 2.3.3. (*Adaptação de [28]*) *Seja D um digrafo acíclico com uma fonte e um sorvedouro. D é um sp-digrafo se, e somente se, seu digrafo reduzido D_R não possui um subdigrafo com uma ponte de Wheatstone.*

Demonstração. $\longrightarrow$ *Prova por contradição:* Seja D um sp-digrafo que possui uma ponte de *Wheatstone* em um subdigrafo reduzido. A aplicação de sucessivas reduções sobre D leva ao sp-digrafo trivial. Assim, pela proposição 2.3.1, o sp-digrafo trivial também deve possuir uma ponte de *Wheatstone* como subgrafo, o que é uma contradição.

$\longleftarrow$ *Prova indireta* Para provar a suficiência proposta pelo teorema, mostraremos que se D não é sp-digrafo, então D possui um subdigrafo com uma ponte de *Wheatstone*.

Aplicando as reduções sequencial e paralela sobre D , chegaremos em um digrafo D_R que não é isomórfico ao sp-digrafo trivial. Podemos, também, concluir que D_R é acíclico, possui uma fonte e um sorvedouro. Isto porque a aplicação das reduções conserva essas propriedades de D .

De acordo com o Lema 2.3.2, podemos localizar o padrão da Figura 2.10 em D_R . Como D_R possui uma fonte e um sorvedouro, podemos achar os caminhos a , b , c e d em D_R (Figura 2.11). Assuma, nesse caso, que s é o último vértice do caminho a que também está em b , e t é o primeiro vértice que está no caminho c e em d .

Figura 2.10: Padrão estrutural encontrado em pontes de *Wheatstone*

Considerando a situação da Figura 2.11, podemos verificar que:

- (a) a e b são disjuntos, pela escolha de s ;

- (b) c e d são disjuntos, pela escolha de t ;
- (c) a e c , b e c , e b e d são disjuntos, pois D_R não possui ciclos.

Figura 2.11: Disposição estrutural de um subdigrafo com uma ponte de *Wheatstone*

Agora, basta analisar a relação entre os caminhos a e d . Se a e d forem disjuntos, nada resta a fazer, pois já teremos a ponte de *Wheatstone*. Caso a e d não sejam disjuntos, seja w o primeiro vértice que pertença a a e d , teríamos um subdigrafo semelhante ao da Figura 2.12, que também é uma ponte de *Wheatstone*. ■

Figura 2.12: Alternativa mostrada quando os caminhos a e d não são disjuntos

O resultado do Teorema 2.3.3 nos fornece uma importante propriedade, que permite construir um algoritmo rápido de reconhecimento de sp-digrafos. Pelo teorema, um sp-digrafo pode ser caracterizado somente pela ausência de uma ponte de *Wheatstone*. Dessa forma, a aplicação sucessiva das reduções leva até um digrafo reduzido, que conterà uma ponte de *Wheatstone*, ou será isomórfico ao sp-digrafo trivial.

2.3.4 Reconhecimento de SP-Digrafos

Descobrir se um digrafo D é série-paralelo é um problema para o qual se conhece soluções eficientes. Em [28, 58, 60], podemos encontrar exemplos de algoritmos de reconhecimento de sp-digrafos que consomem tempo de execução de ordem linear sobre o número de arcos. Basicamente, as estratégias adotadas em todos os algoritmos são variações da aplicação de reduções sobre o digrafo dado como entrada.

A aplicação sucessiva de reduções sequenciais e paralelas reduz o número de arcos e vértices, até a obtenção de um digrafo reduzido. Caso esse digrafo seja isomórfico ao sp-digrafo trivial, então o digrafo inicial é um sp-digrafo. Caso contrário, o digrafo inicial possui uma ponte de *Wheatstone*.

Lema 2.3.4. *Um digrafo é um série-paralelo se, e somente se, pode ser reduzido ao sp-digrafo trivial por meio de uma seqüência de reduções sequenciais e paralelas.*

Demonstração. $\longrightarrow$ A demonstração será feita por indução sobre o número de arcos. Seja D um sp-digrafo. Se D possui somente um arco, então D é isomórfico ao sp-digrafo trivial e nenhuma redução se faz necessária. Caso contrário, D possui k arcos. Por definição de sp-digrafos, D pode ser construído a partir de dois sp-digrafos D_1 e D_2 , tal que cada um possui, no máximo $k - 1$ arcos. Por hipótese de indução, D_1 e D_2 podem ser reduzidos independentemente ao sp-digrafo trivial por meio de uma seqüência de reduções seqüenciais e paralelas. A aplicação das reduções sobre D_1 e D_2 reduzirá o digrafo D a dois arcos dispostos em série ou paralelo, de modo que a aplicação de mais uma redução originará o sp-digrafo trivial.

$\longleftarrow$ Esta demonstração também será feita por indução sobre o número de arcos. Seja D um digrafo que pode ser reduzido ao sp-digrafo trivial. Se D possui um único arco, então D é sp-digrafo. Caso contrário, D possuirá k arcos. Considere a última redução aplicada a D , que reduz dois arcos a_1 e a_2 , dispostos em série ou em paralelo, em um único arco. Os arcos a_1 e a_2 certamente são resultado da redução de dois subdigrafos D_1 e D_2 . Tanto D_1 quanto D_2 possuem, no máximo, $k - 1$ arcos e, por hipótese de indução, são sp-digrafos. Os vértices fonte e sorvedouro de a_1 (a_2) também são os vértices fonte e sorvedouro de D_1 (D_2), pois as reduções conservam os vértices terminais. Assim, D pode ser construído a partir de uma composição seqüencial de D_1 e D_2 (se a_1 e a_2 estiverem em série), ou paralela, caso contrário. Desse modo, D é sp-digrafo. ■

O Corolário 2.3.5 apresenta uma importante consequência do Lema 2.3.4 acima.

Corolário 2.3.5. *Seja D um sp-digrafo. A aplicação de uma redução sobre D gera um digrafo que também é série-paralelo.*

Demonstração. Seja D' o digrafo obtido por meio de uma redução (seqüencial ou paralela) sobre D . Suponha que D' não seja sp-digrafo. Pelo Lema 2.3.4, D' não pode ser reduzido ao sp-digrafo trivial, o que é uma contradição, pois D é sp-digrafo. ■

Os resultados do Teorema 2.3.3, Lema 2.3.4 e Corolário 2.3.5 formam a base para a elaboração de um algoritmo eficiente de reconhecimento de sp-digrafos. O lema 2.3.4, entretanto, não garante que a aplicação de uma seqüência arbitrária de reduções levará um sp-digrafo ao sp-digrafo trivial, mas que é possível, por meio de uma seqüência bem escolhida, chegar ao modelo trivial de sp-digrafo. Felizmente, podemos usar o fato de que o sistema de redução que estamos adotando possui a propriedade Church-Rosser, que garante a preservação da redutibilidade, seja qual for a seqüência de reduções usada.

2.4 Conclusões

Grafos e digrafos são ferramentas de modelagem extremamente úteis e poderosas, com grande aplicabilidade em várias categorias de problemas da Ciência da Computação. Além do seu aspecto visual, que facilita a compreensão dos modelos, a Teoria de Grafos é amplamente estudada há décadas, e conta com um número considerável de trabalhos que exploram suas propriedades matemáticas e apresentam algoritmos capazes de resolver total ou parcialmente muitos de seus problemas.

Grafos dirigidos são representantes de um tipo de grafo no qual os arcos são direcionados. Digrafos são muito úteis na definição de modelos em que os vértices apresentam algum tipo de ordenação ou dependência. Os sp-digrafos formam uma classe de digrafos amplamente usada em pesquisa de escalonamento de tarefas, modelagem de circuitos e processos de negócio. A identificação de sp-digrafos pode ser feita rapidamente e seu tratamento é considerado simples, o que o credencia como interessante ferramenta de modelagem. Nos capítulos seguintes, veremos como aproveitar a topologia de sp-digrafos para geração de expressões algébricas a partir de estruturas de grafos.

Capítulo 3

Álgebra de Processos

3.1 Introdução

A álgebra de processos é um conjunto de ferramentas formais para a representação de dados e processos, com ênfase em processos executados concorrentemente [36, 56]. A principal motivação do estudo de aspectos algébricos de processos é o desafio de representar sistemas usando expressões matemáticas, facilitando a identificação de propriedades internas e derivando, a partir dessas, outras propriedades formais. O estudo algébrico do comportamento de processos é ainda uma área em desenvolvimento, contando com linhas de pesquisa que priorizam diferentes aspectos de representação, como π -Calculus [55], LOTOS (*Language of Temporal Ordering Specifications*) [52, 53] e a álgebra de processos propriamente dita [37, 56].

O termo *álgebra de processos* foi usado pela primeira vez por Bergstra e Klop, em [67]. Seus fundamentos, entretanto, tiveram origem em linhas de trabalho independentes, conduzidas por Milner [32, 33] e Hoare [54]. O desenvolvimento da álgebra de processos passou pela definição de operadores básicos, operadores de entrelaçamento, abstração de subconjuntos de expressões, análise de equivalência e, finalmente, a definição de estados de *deadlock* e operações de encapsulamento. Após esse avanço, Bergstra e Klop [68–70] definiram as bases formais da ACP (*Algebra of Communication Processes*), originando um poderoso arcabouço formal para definição de processos de negócio e suas propriedades. Por todas essas características, este trabalho adotará a ACP como sua linguagem de referência, de modo que qualquer menção ao termo “álgebra de processos” estará diretamente ligada à ACP.

Este capítulo apresenta algumas definições e propriedades básicas da ACP, que fundamentarão os capítulos relacionados à geração de expressões algébricas. A maioria das definições apresentadas foram extraídas de [56, 71, 72]. A Seção 3.2 apresenta os operadores básicos da ACP, complementados posteriormente pelos operadores de paralelismo, na Seção 3.3. Regras e critérios de equivalência são discutidos na Seção 3.4 e a Seção 3.5 discute o comportamento da equivalência de processos sobre ações decomponíveis. A Seção 3.6 apresenta os principais axiomas da ACP. O tratamento de *deadlocks* e encapsulamento é abordado na Seção 3.7. Finalmente, as conclusões do capítulo podem ser vistas na Seção 3.8.

3.2 Operadores Básicos

Nesta seção, serão apresentados os operadores básicos da ACP.

Álgebra de Processos Básica

A Álgebra de Processos Básica (APB) é o conjunto de todas as definições que envolvem os elementos básicos da álgebra de processos, tais como ação atômica, composição alternativa e composição sequencial.

Definição 3.2.3: (*Assinatura*) Uma assinatura Σ consiste num conjunto finito de símbolos de função (operadores), no qual cada símbolo de função $f \in \Sigma$ possui aridade $ar(f)$, sendo esta a sua quantidade de parâmetros.

Um símbolo de função com aridade 0 é chamado **constante**. Símbolos de função com aridade 1 são chamados **unários** e símbolos com aridade 2 são chamados **binários**.

Definição 3.2.4: (*Termo*) Seja Σ uma assinatura. O conjunto $\mathbb{T}(\Sigma)$ de termos (abertos) é o menor conjunto de Σ que satisfaça as seguintes condições:

- cada variável está em $\mathbb{T}(\Sigma)$;
- se $f \in \Sigma$ e $t_1, \dots, t_{ar(f)} \in \mathbb{T}(\Sigma)$, então $f(t_1, \dots, t_{ar(f)}) \in \mathbb{T}(\Sigma)$.

Um termo que não contém variáveis é chamado *termo fechado*. O conjunto dos termos fechados é denotado por $\mathcal{T}(\Sigma)$.

Usaremos a notação $o_1 \equiv o_2$ para representar a equivalência sintática entre os objetos o_1 e o_2 . Para funções unárias f e números naturais k , f^k representa k aplicações de f . Por exemplo, $f^0(t) \equiv t$ e $f^{k+1}(t) \equiv f(f^k(t))$.

Considere um conjunto A , não-vazio, de ações atômicas, que representam um comportamento definido e indivisível. Cada ação $a \in A$ é uma constante que pode ser executada e finalizada com sucesso. Considere, ainda, um conjunto S de estados que podem ser observados num processo de negócio.

Definição 3.2.5: (*Sistema de Transições Rotuladas [56]*) Uma transição é uma tripla (s, a, s') com $a \in A$, ou um par (s, P) com P um predicado, onde $s, s' \in S$. Um **sistema de transições rotuladas** (STR) é um conjunto possivelmente infinito de transições. Um STR possui ramificação finita se cada um de seus estados possui somente transições que levam a estados com ramificações finitas.

Por conveniência, uma transição (s, a, s') é geralmente denotada por $s \xrightarrow{a} s'$. Ela indica que um estado s pode evoluir para um estado s' por meio da execução de uma ação a . Além disso, uma transição (s, P) é geralmente denotada por sP e indica que um predicado P é válido para s .

Definição 3.2.6: (*Processo*) Um grafo de processos (ou simplesmente processo) p é um STR em que um estado s é tomado como raiz. Se o STR contém uma transição $s \xrightarrow{a} s'$, então $p \xrightarrow{a} p'$, onde p'

tem estado raiz em s' . Além disso, se o STR possuir uma transição sP , então pP .

- Um processo p_0 é finalizável se existem, a partir de p_0 , somente seqüências finitas $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_k} p_k$;
- Um processo p_0 é regular se existe somente um processo p_k , tal que $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_k} p_k$.

Definição 3.2.7: (*Especificação de Sistema de Transição (EST)*) Uma regra de transição ρ é uma expressão da forma $\frac{H}{\pi}$, com H um conjunto de expressões $t \xrightarrow{a} t'$ e tP , com $t, t' \in \mathbb{T}(\Sigma)$, chamado de premissas de ρ , e π uma expressão $t \xrightarrow{a} t'$ ou tP , com $t, t' \in \mathbb{T}(\Sigma)$, chamado conclusão de ρ . O lado esquerdo de π é chamado fonte de ρ . Uma regra de transição é fechada se não contém variáveis.

Uma especificação de sistema de transição é um conjunto de regras de transição.

O predicado $\xrightarrow{a} \sqrt{}$ representa a finalização bem sucedida após a execução da ação a .

Considere também um operador binário $+$, que será chamado *composição alternativa*, cuja semântica é dada a seguir:

Definição 3.2.8: (*Composição Alternativa*) Dados dois termos fechados t_1 e t_2 , que representam respectivamente os processos p_1 e p_2 . A composição alternativa de t_1 e t_2 é o termo fechado $t_1 + t_2$, que representa o processo que executa exclusivamente p_1 ou p_2 .

O grafo de processos de $t_1 + t_2$ é obtido pela junção das raízes dos grafos de p_1 e p_2 .

Definição 3.2.9: (*Composição Seqüencial*) Dados dois termos fechados t_1 e t_2 , que representam respectivamente os processos p_1 e p_2 , a composição seqüencial de t_1 e t_2 , que é o termo fechado $t_1 \cdot t_2$ representa o processo que executa p_1 e, em seguida, p_2 .

O grafo de processos de $t_1 \cdot t_2$ é obtido pela substituição de cada finalização bem-sucedida $s \xrightarrow{a} \sqrt{}$ em p_1 por uma transição $s \xrightarrow{a} s'$, em que s' é a raiz de p_2 .

3.2.1 Regras de Transição

A EST de APB fornece, para cada termo básico de processo, um grafo de processos. As regras de transição são mostradas na Tabela 3.1. Para todas as regras de transição deste capítulo, $v \in A$ é uma ação atômica, enquanto x e y são termos algébricos.

- a primeira regra de transição diz que cada ação atômica v pode finalizar com sucesso, executando a si mesma;
- as quatro regras seguintes expressam que $t + t'$ executa, ou t , ou t' ;
- as últimas duas regras expressam que $t.t'$ executa t' após a finalização bem-sucedida de t .

Tabela 3.1: Regras de transição para Álgebra de Processos Básica

$$\begin{array}{c}
\frac{}{v \xrightarrow{v} \sqrt{}} \\
\\
\frac{x \xrightarrow{v} \sqrt{}}{x+y \xrightarrow{v} \sqrt{}} \quad \frac{x \xrightarrow{v} x'}{x+y \xrightarrow{v} x'} \quad \frac{y \xrightarrow{v} \sqrt{}}{x+y \xrightarrow{v} \sqrt{}} \quad \frac{y \xrightarrow{v} y'}{x+y \xrightarrow{v} y'} \\
\\
\frac{x \xrightarrow{v} \sqrt{}}{x \cdot y \xrightarrow{v} y} \quad \frac{x \xrightarrow{v} x'}{x \cdot y \xrightarrow{v} x' \cdot y}
\end{array}$$

3.3 Paralelismo

Os operadores de representação de paralelismo não pertenciam à proposta inicial de álgebra de processos. O paralelismo só foi incorporado após os trabalhos de Milner [32], que sugeriram um operador binário de *entrelaçamento*, que toma duas ações como parâmetro e as executa paralelamente. O paralelismo entre duas atividades, s e t , é representado pelo operador de *entrelaçamento* ($s \parallel t$), indicando que a transição pode ser iniciada em s ($s \xrightarrow{a} s'$ ou $s \xrightarrow{a} \sqrt{}$) ou t ($t \xrightarrow{a} t'$ ou $t \xrightarrow{a} \sqrt{}$). As regras de transição podem ser vistas na Tabela 3.2.

Tabela 3.2: Regras de transição para Álgebra de Processos com Paralelismo

$$\begin{array}{c}
\frac{x \xrightarrow{v} \sqrt{}}{x \parallel y \xrightarrow{v} y} \quad \frac{x \xrightarrow{v} x'}{x \parallel y \xrightarrow{v} x' \parallel y} \\
\\
\frac{y \xrightarrow{v} \sqrt{}}{x \parallel y \xrightarrow{v} x} \quad \frac{y \xrightarrow{v} y'}{x \parallel y \xrightarrow{v} x \parallel y'}
\end{array}$$

Além de permitir o início da execução pelas ações s e t , $s \parallel t$ pode executar uma comunicação entre transições iniciais de s e t . Para isso, assume-se uma função de comunicação $\eta : A \times A \rightarrow A$, que produz uma comunicação $\eta(a, b)$ para cada par de ações a e b . Assume-se também que a η é comutativa e associativa.

Com isso, $s \parallel t$ ganha mais quatro regras de transição, mostradas na Tabela 3.3.

Moller provou que não existe uma axiomatização finita correta e completa que incorpore uma extensão da APB com o operador $\parallel$ [56]. Para preencher essa lacuna, dois novos operadores foram propostos: o *entrelaçamento à esquerda* ($\ll$) e o *entrelaçamento com comunicação* ($\mid$). Juntos, esses operadores podem reproduzir o comportamento do operador de entrelaçamento original ($\parallel$) e ainda apresentam uma axiomatização completa para ACP.

Tabela 3.3: Regras de transição para Comunicação em Álgebra de Processos com Paralelismo

$$\begin{array}{c}
\frac{x \xrightarrow{v} \sqrt{}}{x \parallel y \xrightarrow{\eta(v, w)} \sqrt{}} \quad \frac{x \xrightarrow{v} \sqrt{}}{x \parallel y \xrightarrow{\eta(v, w)} y'} \\
\\
\frac{x \xrightarrow{v} x'}{x \parallel y \xrightarrow{\eta(v, w)} x'} \quad \frac{x \xrightarrow{v} x'}{x \parallel y \xrightarrow{\eta(v, w)} x' \parallel y'}
\end{array}$$

O *entrelaçamento à esquerda* é um operador binário, que toma o primeiro argumento como sua transição inicial e, a partir daí, apresenta o mesmo comportamento do operador $\parallel$. As regras de transição que expressam esse comportamento são mostradas na Tabela 3.4:

Tabela 3.4: Regras de transição para o entrelaçamento à esquerda

$$\frac{x \xrightarrow{v} \surd}{x \parallel y \xrightarrow{v} y} \quad \frac{x \xrightarrow{v} x'}{x \parallel y \xrightarrow{v} x' \parallel y}$$

O *entrelaçamento com comunicação* também é um operador binário, que toma dois termos como argumentos e executa, a comunicação entre as transições iniciais deles. A partir disso, o entrelaçamento com comunicação apresenta o mesmo comportamento do operador $\parallel$. Suas regras de transição podem ser vistas na Tabela 3.5.

Tabela 3.5: Regras de transição para o entrelaçamento com comunicação

$$\frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} \surd}{x|y \xrightarrow{\eta(v,w)} \surd} \quad \frac{x \xrightarrow{v} \surd \quad y \xrightarrow{w} y'}{x|y \xrightarrow{\eta(v,w)} y'}$$

$$\frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} \surd}{x|y \xrightarrow{\eta(v,w)} x'} \quad \frac{x \xrightarrow{v} x' \quad y \xrightarrow{w} y'}{x|y \xrightarrow{\eta(v,w)} x' \parallel y'}$$

As relações de precedência entre esses operadores vistos até o momento estão relacionadas na Tabela 3.6. Casos com precedência de mesmo nível evoluem da esquerda para a direita.

Tabela 3.6: Precedência de operadores em álgebra de processos

Precedência	Operador	Descrição	Exemplo
1	()	Agrupamento	(a + b).c
2	.	Composição seqüencial	a.b
3	$\parallel$ $\ll$	Composição paralela	a $\parallel$ b
4	+	Composição alternativa	a + b

Com a apresentação dos operadores de entrelaçamento à esquerda e entrelaçamento com comunicação, encerra-se o conjunto de operadores da álgebra de processos que serão considerados nesse trabalho. Serão apresentados, agora, aspectos relacionados à equivalência entre termos de processos, usados como referência para a definição dos axiomas que fundamentam o comportamento das expressões em ACP.

3.4 Equivalência

Na álgebra de processos, assim como na análise formal de sistemas, de forma geral, a comparação entre sistemas é uma operação extremamente importante. Por meio de comparações, é possível associar implementações com sua representação abstrata, ou mostrar que um sistema é subconjunto de outro, ou ainda mostrar que um sistema pode ser substituído por outro. Essa informação pode ser útil na construção de sistemas maiores e complexos.

Figura 3.1: Equivalência de Rastros

Nesta seção, veremos três relações de equivalência, que usam critérios com níveis variados de exigência e podem ser usados para comparar sistemas representados por grafos ou expressões algébricas.

3.4.1 Equivalência de Rastros

Em álgebra de processos, chamamos de **rastro** uma seqüência finita de ações que podem ser observadas em um determinado sistema. A definição original de rastro foi dada por Hoare [54], para comparar sistemas modelados em CCS, e usada em muitos trabalhos posteriores, por exemplo a análise global de comparações [73].

Formalmente, podemos definir rastro da seguinte forma:

Definição 3.4.10: (*Rastro*) Dados dois processos p e q , o rastro de p a q é uma seqüência de transições $\sigma \in T(\Sigma^*)$, tal que $p \xrightarrow{\sigma} q$.

Definição 3.4.11: (*Equivalência de Rastros*) Seja $T(p)$ e $T(q)$ os conjuntos de todos os rastros de p e q , respectivamente. Dois processos, p e q , são equivalentes em rastros se têm o mesmo conjunto de rastros, isto é, $T(p) = T(q)$.

No exemplo mostrado na Figura 3.1, temos dois processos, p e q , nos quais a ramificação é alternativa. Considerando a , b e c ações atômicas, nota-se, por meio do grafo de processo que ambas as expressões produzem exatamente o mesmo conjunto de rastros, ou seja, $T(p) = T(q) = \{ab, ac\}$. Sendo assim, podemos afirmar que p e q são equivalentes por rastros.

Um bom método para avaliar a equivalência por rastros de duas expressões é traçar uma árvore de processo de cada uma e comparar seus caminhos entre a raiz e cada folha.

3.4.2 Equivalência de Simulação

Dado um conjunto S de sistemas, e um conjunto de ações A , dizemos que o sistema p simula o sistema q se existe uma relação binária $R \subseteq S \times S$ que satisfaça:

1. $p \mathcal{R} q$;

Figura 3.2: Equivalência de Simulação

2. Se $p\mathcal{R}q$ e $p \xrightarrow{\alpha} p'$, então existe algum q' , tal que $q \xrightarrow{\alpha} q'$, e $p'\mathcal{R}q'$.

O objetivo da equivalência por simulação é indicar que um processo p pode simular o comportamento de um processo q . Para isso, devem existir duas relações $\mathcal{R}$ e $\mathcal{S}$, tais que $p\mathcal{R}q$ e $q\mathcal{S}p$.

O exemplo mostrado na Figura 3.2 apresenta um exemplo de dois sistemas equivalentes por simulação. O sistema p possui quatro estados, s_1 , s_2 , s_3 e s_4 , com as regras de transição $R_p = \{s_1 \xrightarrow{a} s_2, s_2 \xrightarrow{b} s_3, s_1 \xrightarrow{a} s_4\}$. Da mesma forma o sistema q possui três estados, r_1 , r_2 e r_3 , com as regras de transição $R_q = \{r_1 \xrightarrow{a} r_2, r_2 \xrightarrow{b} r_3\}$.

Mapeando as raízes de ambos os sistemas, temos que $s_1\mathcal{R}r_1$. A partir de então, constroem-se dois mapeamentos de p em q , a fim de provar que $p\mathcal{R}_1q$ e $p\mathcal{R}_2q$. Para $p\mathcal{R}_1q$, podemos usar o mapeamento $\mathcal{R}_1 = \{\{s_1, r_1\}, \{s_2, r_2\}, \{s_3, r_3\}, \{s_4, r_2\}\}$. Note que, para o estado s_4 , não faz diferença que o estado r_2 de q ainda possua transições. Isso porque a função de mapeamento $\mathcal{R}_1$ tem domínio somente em p .

Por outro lado, o mapeamento $\mathcal{R}_2$ pode ser $\mathcal{R}_2 = \{\{r_1, s_1\}, \{r_2, s_2\}, \{r_3, s_3\}\}$. Portanto, p e q são equivalentes por simulação.

Já o exemplo da Figura 3.1 mostra dois sistemas que não são equivalentes por simulação. Ao construir os mapeamentos, pode-se constatar que existe um mapeamento entre q e p , porém não existe mapeamento entre p e q . Um outro detalhe interessante que pode ser mostrado é que sistemas equivalentes por simulação geram os mesmos rastros, ou seja, também são equivalentes por rastros. O contrário não é verdade, como vimos no exemplo da Figura 3.1.

3.4.3 Equivalência de Bissimulação

Na *equivalência de simulação*, dois processos p e q são ditos equivalentes se existem duas relações $\mathcal{R}$ e $\mathcal{S}$, tais que $p\mathcal{R}q$ e $q\mathcal{S}p$. Na equivalência por bissimulação, a relação $\mathcal{S}$ é oposta a $\mathcal{R}$, ou seja $\mathcal{S} = \mathcal{R}^{-1}$. Isso torna o critério de equivalência muito mais restritivo, exigindo que os processos, além de rastros e simulação, também apresentem a mesma estrutura de dependências e ramificação. A definição formal da bissimulação é dada a seguir.

Definição 3.4.12: (*Bissimulação*) A **bissimulação** é uma relação binária $\mathcal{B}$, tal que:

- se $p\mathcal{B}q$ e $p \xrightarrow{a} p'$, então $q \xrightarrow{a} q'$ com $p'\mathcal{B}q'$;
- se $p\mathcal{B}q$ e $q \xrightarrow{a} q'$, então $p \xrightarrow{a} p'$ com $p'\mathcal{B}q'$;
- se $p\mathcal{B}q$ e $p \xrightarrow{a} \perp$ então $q \xrightarrow{a} \perp$;

- se $p\mathcal{B}q$ e $q \xrightarrow{a} \surd$ então $p \xrightarrow{a} \surd$;

Dois processos p e q são bissimilares, $p \leftrightarrow q$, se existe uma relação de bissimulação $\mathcal{B}$, tal que $p\mathcal{B}q$.

A exemplo de rastros e simulação, a bissimulação também é uma relação de equivalência:

- $\leftrightarrow$ é reflexiva, pois $p \leftrightarrow p$;
- $\leftrightarrow$ é simétrica, pois se $p \leftrightarrow q$, então $q \leftrightarrow p$;
- $\leftrightarrow$ é transitiva, pois se $p \leftrightarrow q$ e $q \leftrightarrow r$, então $p \leftrightarrow r$.

A equivalência por bissimulação é considerada o nível mais exigente de avaliação de equivalência de processos. As equivalências por rastros, simulação e bissimulação, nessa ordem, apresentam níveis progressivos de exigência, de modo que dois processos bissimilares são também equivalentes por simulação, e estes, por sua vez, também são equivalentes por rastros [71, 72].

3.5 Concorrência de Termos e Equivalência

A modelagem de processos de negócio procura, na medida do possível, fazer uso de execuções paralelas entre ações ou conjuntos de ações. Esse caminho de modelagem minimiza o número de dependências e otimiza o tempo de execução, na medida em que aponta regiões do processo que podem ser executadas de modo independente. A álgebra de processos traduz esse mecanismo de modelagem para a composição paralela de ações atômicas ou termos. Entretanto, a composição paralela de termos ou ações apresenta uma deficiência que não pode ser ignorada. Essa indesejável deficiência possui caráter semântico e está relacionada ao entrelaçamento em relação ao refinamento de termos. Considerando as regras de composição paralela e a álgebra de processos sem comunicação, cada expressão apresenta um conjunto de rastros que pode ser obtido pelo entrelaçamento dos elementos compostos pelos operadores paralelos. Chamamos esse mecanismo de semântica de entrelaçamento. Por exemplo, a expressão

$$E : p = a \parallel b \cdot c$$

pode evoluir para $b \cdot c$, após a execução de a ; ou para $a \parallel c$, após a execução de b . Assim, é possível mostrar que E é equivalente, por bissimulação, à seguinte expressão:

$$p' = a \cdot b \cdot c + b \cdot a \cdot c + b \cdot c \cdot a$$

A expressão é resultado do entrelaçamento das ações do termo da esquerda (a) com as ações do termo da direita ($b \cdot c$).

Considere que, no lugar de ações, há termos e um termo pode ser substituído por uma seqüência finita de subtermos (encadeados pelo operador “.”). O raciocínio a seguir mostra que, em Álgebra de Processos, o refinamento de termos em ações atômicas não é fechado em relação à semântica de entrelaçamento. Considerando a expressão

$$q = a \parallel d,$$

nota-se que os termos a e d são executados concorrentemente. Agora, vamos trocar q por uma expressão equivalente bissimilar q' .

$$q' = a \cdot d + d \cdot a,$$

Se o termo d for refinado para $b \cdot c$ na expressão q , teremos que $p = q$. Em contrapartida, A expressão q'' , resultado do refinamento de d em q' não é equivalente a p' .

$$q'' = a \cdot b \cdot c + b \cdot c \cdot a,$$

Dessa maneira, podemos afirmar que a semântica de entrelaçamento vale somente quando falamos de ações atômicas.

Se, ao contrário, a composição envolvida for alternativa, o problema já não ocorre, pois nesse caso não há entrelaçamento entre as ações (ou termos).

3.6 Axiomas da álgebra de processos

A bissimulação, como relação de equivalência, sustenta métodos de comparação e tratamento de processos. A identificação da bissimulação entre dois processos pode ser feita por meio do mapeamento entre seus estados correspondentes. Entretanto, essa operação pode ser, por vezes, custosa, de modo que é extremamente desejável encontrar outros caminhos para estabelecer algebricamente a relação de equivalência entre dois processos.

A axiomatização é um processo de desenvolvimento de axiomas, que são aplicados na álgebra de processos para estabelecer relações de equivalência entre processos sem a necessidade de construção de seus grafos. Na apresentação dos axiomas deste capítulo, u , v e w representam ações atômicas, enquanto x , y e z representam termos algébricos.

Os axiomas da álgebra de processos básica são mostrados na Tabela 3.7.

Tabela 3.7: Axiomas da Álgebra de Processos Básica

$$\begin{aligned} \text{A1: } & x + y = y + x \\ \text{A2: } & (x + y) + z = x + (y + z) \\ \text{A3: } & x + x = x \\ \text{A4: } & (x + y) \cdot z = x \cdot z + y \cdot z \\ \text{A5: } & (x \cdot y) \cdot z = x \cdot (y \cdot z) \end{aligned}$$

Assim, a álgebra de processos básica apresenta as seguintes propriedades para operações de composição:

- *A1 (comutatividade sobre +)*: a composição alternativa entre dois termos é comutativa;
- *A2 (associatividade sobre +)*: a composição alternativa é associativa;
- *A3 (idempotência sobre +)*: a escolha alternativa entre termos iguais é suprimida ao próprio termo;
- *A4 (distributividade à direita sobre ·)*: a composição alternativa é distributiva à direita em relação à composição seqüencial;

- *A5 (associatividade sobre $\cdot$):* a composição seqüencial é associativa.

Para a álgebra de processos com paralelismo, tem-se os axiomas mostrados na Tabela 3.8, que apresenta as seguintes propriedades para operações de composição:

Tabela 3.8: Axiomas da Álgebra de Processos com Paralelismo

I1:	$x \parallel y = (x \parallel y + y \parallel x) + x y$
IE2:	$v \parallel y = v \cdot y$
IE3:	$(v \cdot x) \parallel y = v \cdot (x \parallel y)$
IE4:	$(x + y) \parallel z = x \parallel z + y \parallel z$
IC5:	$v w = \eta(v, w)$
IC6:	$v (w \cdot y) = \eta(v, w) \cdot y$
IC7:	$(v \cdot x) w = \eta(v, w) \cdot x$
IC8:	$(v \cdot x) (w \cdot y) = \eta(v, w) \cdot (x \parallel y)$
IC9:	$(x + y) z = x z + y z$
IC10:	$x (y + z) = x y + x z$

- I1 define o axioma para o entrelaçamento, segundo o qual cada transição do termo $x \parallel y$ tem sua primeira transição em x (representado por $x \parallel y$) ou y (representado por $y \parallel x$), ou ainda a comunicação entre as transições iniciais de x e y ;
- IE2 e IE3 são conseqüências diretas da aplicação do operador $\parallel$;
- IE4 *distributividade à direita para $\parallel$* : a escolha alternativa local entre os termos x e y na expressão $(x + y) \parallel z$ leva à escolha global entre os termos $x \parallel z$ e $y \parallel z$;
- Os axiomas IC5-IC8 são conseqüências diretas da semântica do operador $|$, que toma como transição inicial a comunicação entre v e w ;
- IC9 *distributividade à direita para $|$* : a escolha alternativa local entre os termos x e y na expressão $(x + y)|z$ leva à escolha global entre $s|u$ e $t|u$;
- IC10 *distributividade à esquerda para $|$* : a escolha alternativa local entre os termos x e y na expressão $x|(y + z)$ leva à escolha global entre $x|y$ ou $x|z$.

3.7 Deadlock e Encapsulamento

Outro importante fundamento relacionado à comunicação entre termos é a representação de situações em que a comunicação não é possível. Na álgebra de processos, isso é feito por meio da constante δ , chamada de constante de *deadlock*. A inclusão da idéia de *deadlock* na álgebra de processos foi introduzida por Milner [32] e representa um complemento da definição de comunicação entre termos de um processo.

Sendo assim, a função η passa a ser definida como $\eta : A \times A \rightarrow A \cup \{\delta\}$. Essa extensão permite, agora, expressar explicitamente que duas ações, por exemplo a e b , não se comunicam, de modo que $\eta(a, b) \triangleq \delta$, onde a relação $\triangleq$ indica que a função $\eta(a, b)$ é mapeada para uma classe de equivalência em que não ocorre comunicação entre ações (δ).

Além do *deadlock*, o conceito de encapsulamento representa um conjunto de ações atômicas que não apresentam comunicação entre si. O operador de encapsulamento ∂_H é um operador unário que atribui δ a todos os elementos do conjunto H de ações atômicas. As regras de transição de ∂ são mostradas na tabela 3.9.

Tabela 3.9: Regras de transição para Encapsulamento

$$\frac{x \xrightarrow{v} \surd}{\partial_H(x) \xrightarrow{v} \surd} \quad v \notin H \qquad \frac{x \xrightarrow{v} x'}{\partial_H(x) \xrightarrow{v} \partial_H(x')} \quad v \notin H$$

Os axiomas envolvendo *Deadlock* e Encapsulamento são apresentados na Tabela 3.10.

Tabela 3.10: Axiomas da álgebra de processos com *Deadlock* e Encapsulamento

A6:	$x + \delta = x$
A7:	$\delta \cdot x = \delta$
D1:	$v \notin H \quad \partial_H(v) = v$
D2:	$v \in H \quad \partial_H(v) = \delta$
D3:	$\partial_H(\delta) = \delta$
D4:	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$
D5:	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$
CM11:	$\delta \parallel x = \delta$
CM12:	$\delta x = \delta$
CM13:	$x \delta = \delta$

Em relação ao *deadlock* e encapsulamento, a álgebra de processos apresenta como propriedades:

- A6: O caminho alternativo entre um *deadlock* e um termo x sempre privilegia x ;
- A7: Composições seqüenciais iniciadas por *deadlock* sempre resultam em *deadlock*;
- D1-D3: Um termo só produz uma transição válida se não estiver no encapsulamento ∂_H ;
- D4-D5: O encapsulamento é uma função fechada sobre a composição alternativa e seqüencial;
- CM11: A presença de um *deadlock* no início de um entrelaçamento à esquerda produz um *deadlock*;
- CM12-CM13: A comunicação de qualquer termo com um *deadlock* resulta em um *deadlock*.

3.8 Conclusões

A álgebra de processos é um instrumento formal que permite representar o comportamento de sistemas de forma algébrica. Expressões algébricas de processos facilitam a análise do sistema por meio do estudo de suas propriedades matemáticas. Além disso, a álgebra de processos fornece um caminho para relacionar processos diferentes usando análise de equivalência. Uma outra característica relevante das expressões algébricas é a sua representação textual, que pode ser organizada de forma unidirecional, facilitando a leitura por parte de sistemas computacionais. Todas essas características credenciam o estudo algébrico como base para sistemas de controle de execução de processos de negócio.

Capítulo 4

Sistemas Abstratos de Redução e Transformação de Grafos

4.1 Introdução

Sistemas abstratos de redução fundamentam muitos métodos formais de modelagem, verificação e validação de estruturas, como transformação de grafos, álgebras de processos e gramáticas. Um sistema abstrato define conjuntos de elementos para os quais é possível definir relações de transformação. Além disso, o comportamento das relações é utilizado para avaliar propriedades importantes como confluência e terminação.

O objetivo deste capítulo é introduzir os conceitos básicos de sistemas abstratos de redução e transformação de grafos, que serão necessários para fundamentar a definição do sistema de redução usado no Capítulo 5 para gerar expressões algébricas. Este capítulo se divide em duas partes: sistemas abstratos de redução e transformação de grafos. A Seção 4.2 apresenta as principais características de um sistema abstrato de redução, bem como suas propriedades mais importantes. A Seção 4.3 apresenta uma abordagem algébrica associada a transformação de grafos, que utiliza como base formal os conceitos de sistemas abstratos de redução. Transformação de grafos é uma área com grandes esforços de pesquisa nas três últimas décadas, sendo aplicada na construção e geração de modelos de programação [74, 75], métodos de verificação formal de sistemas [12, 76, 77] e modelagem de sistemas biológicos [78].

4.2 Sistemas Abstratos de Redução

Os sistemas abstratos de redução consistem em um conjunto de objetos, uma relação entre objetos e um conjunto de regras associadas à relação. A relação é chamado *passo de redução*, que pode representar uma transformação entre objetos, ou mesmo a execução de alguma operação sobre objetos. Os sistemas abstratos de redução foram estudados por Newman [79], e propagados por Rosen [80], Huet [81] e Klop [82]. Os sistemas abstratos de redução representam a parte formal básica para sistemas de regras algébricas, e possuem duas grandes vantagens: a definição e utilização de regras baseadas na estrutura dos objetos, e a sua aplicação em vários sistemas formais de reescrita. O rigor em relação ao aspecto estrutural de cada objeto permite a exploração consistente de propriedades dos objetos e fundamenta um sistema no qual é possível elaborar provas matemáticas baseadas nessas propriedades. Do ponto de vista de aplicação, os sistemas abstratos podem ser adequados para o estudo de sistemas de reescrita

de termos, grafos, cadeias de caracteres e sistemas de reescrita condicionais.

Nesta seção, são apresentados resumidamente alguns conceitos sobre os sistemas abstratos de redução. A intenção, neste caso, não é apresentar detalhes sobre os sistemas abstratos de redução, em si, mas fundamentar um conjunto de conceitos que vai ser estendido e explorado na seção seguinte. Maiores detalhes sobre sistemas abstratos de redução podem ser encontrados em [83,84].

Definição 4.2.13: (*Sistema Abstrato de Redução*) Um sistema abstrato de redução $\langle A, \rightarrow \rangle$ consiste num conjunto A e uma relação binária $\rightarrow$ sobre A .

Em um sistema abstrato de redução, é útil verificar algumas propriedades relacionadas à relação $\rightarrow$, que são apresentadas a seguir:

SAR1: A *identidade* sobre A é uma relação $\rightarrow^0 = \{\langle a, a \rangle | a \in A\}$;

SAR2: Um *fecho reflexivo* de $\rightarrow$ é uma relação $\rightarrow^= \Rightarrow \cup \rightarrow^0$;

SAR3: Para todo $n > 0$, então um *composição* n de $\rightarrow$ é uma relação $\rightarrow^n \Rightarrow \rightarrow \circ \rightarrow^{n-1}$;

SAR4: O *fecho transitivo* de $\rightarrow$ é a relação $\rightarrow^+ = \cup_{n>0} \rightarrow^n$;

SAR5: O *fecho transitivo-reflexivo* de $\rightarrow$ é a relação $\rightarrow^* \Rightarrow \rightarrow^+ \cup \rightarrow^0$;

SAR6: O *fecho simétrico* de $\rightarrow$ é a relação $\leftrightarrow \Rightarrow \rightarrow \cup \leftarrow$;

SAR7: O *fecho de equivalência* de $\rightarrow$ (ou *conversibilidade* sobre $\rightarrow$), representado por $\leftrightarrow^*$, é o fecho transitivo-reflexivo de $\leftrightarrow$.

As propriedades apresentadas estabelecem algumas características gerais associadas à relação $\rightarrow$. A primeira propriedade (SAR1) estabelece que qualquer elemento de A pode se relacionar consigo mesmo por meio de $\rightarrow$. A propriedade SAR1 fundamenta a definição do fecho reflexivo sobre $\rightarrow$ em SAR2. Uma composição sobre $\rightarrow$ é definida como a aplicação sucessiva de $\rightarrow$ sobre um elemento de A (SAR3), que é usado para definir o fecho transitivo de $\rightarrow$ (SAR4). As propriedades SAR4 e SAR2 fornecem a base para definição do fecho transitivo-reflexivo (SAR5). Se $\rightarrow$ for simétrica, é possível definir o fecho simétrico em SAR6. Finalmente, o fecho de equivalência é definido, em SAR7, como o fecho transitivo-reflexivo do fecho simétrico de $\rightarrow$.

Entre as propriedades apresentadas em $\langle A, \rightarrow \rangle$, algumas merecem destaque. As propriedades a seguir definem a presença ou não de terminação e confluência em um sistema de redução. Essas duas propriedades permitem analisar se as reduções podem gerar seqüências finitas ou infinitas. Quando finitas, também fornecem ferramentas para avaliar se convergem ou não para a mesma estrutura. Em muitos casos, a presença ou não dessas propriedades é um fator decisivo para avaliar a viabilidade de um sistema de redução.

Definição 4.2.14: (*Propriedades de Terminação e Confluência*)

Uma relação $\rightarrow$ é

- TC1 *finalizável* (*fortemente normalizável* ou *noetheriana*), se não houver uma seqüência infinita da forma $a_1 \rightarrow a_2 \rightarrow \dots$;
- TC2 *normalizável* (ou *fracamente normalizável*) se cada elemento de A possui uma forma normal;
- TC3 *Church-Rosser*, se para todo a e b com $a \leftrightarrow^* b$, existe algum c , tal que $a \rightarrow^* c \leftarrow^* b$;
- TC4 *confluente*, se para todo a , b e c com $b \leftarrow^* a \rightarrow^* c$, existe algum d tal que $b \rightarrow^* d \leftarrow^* c$;
- TC5 *localmente confluente* (ou *fracamente confluente*), se para todo a , b e c com $b \leftarrow a \rightarrow c$, existe algum d tal que $b \rightarrow^* d \leftarrow^* c$;
- TC6 *sub-comutativa* se para todo a , b e c com $b \leftarrow a \rightarrow c$, existe algum d tal que $b \rightarrow^= d \leftarrow^= c$;
- TC7 *convergente*, se é finita e confluente.

A terminação (TC1) avalia se toda instância de um sistema de redução tem sempre um número finito de passos, ou não. Em outras palavras, dado o sistema de redução $\langle A, \rightarrow \rangle$. Se for possível mostrar que, para qualquer elemento de A , existe um número finito de aplicações de $\rightarrow$, então $\langle A, \rightarrow \rangle$ é finalizável. A terminação é uma propriedade extremamente desejável em sistemas de redução, pois garante que qualquer objeto pode ser reduzido a uma forma minimal simplesmente pela aplicação sucessiva da regra de redução. Mais que isso, algumas propriedades que são geralmente indecidíveis se tornam decidíveis na presença de terminação [11].

Se a forma minimal é única, ela é chamada forma normal (TC2). Para garantir a presença de uma forma normal, é necessário que, além de finalizável, o sistema apresente a propriedade *Church-Rosser* (TC3), segundo a qual qualquer seqüência de transformações realizada a partir de um elemento pode convergir para um elemento comum. Podemos ver a ilustração dessa propriedade na Figura 4.1(a). Linhas pontilhadas com rótulo “*” representam a existência de uma seqüência de reduções de um fecho transitivo-reflexivo que leva um objeto a outro. Linhas pontilhadas com rótulo “=” representam a existência de um conjunto de reduções de um fecho reflexivo que leva um objeto a outro. Uma outra maneira de caracterizar a propriedade *Church-Rosser* se dá pela definição de confluência (TC4), cuja ilustração pode ser vista na Figura 4.1(b). Em [11], o autor mostra que as propriedades TC3 e TC4 são equivalentes para sistemas de redução.

Uma outra propriedade útil é a confluência local. Se, na confluência, quaisquer duas seqüências de aplicações de $\rightarrow$ convergem para o mesmo elemento, a confluência local assume que quaisquer duas aplicações diretas de $\rightarrow$ sobre um elemento de A produzem resultados que convergem para o mesmo elemento. Em [79], *Newman* provou que a confluência local e a confluência são equivalentes na presença de terminação, estabelecendo um resultado até hoje conhecido por *Lema de Newman*.

Lema 4.2.1. (*Lema de Newman* [79]) *Uma relação finalizável é confluente se, e somente se, é localmente confluente.*

O resultado de *Newman* permite garantir a confluência analisando somente a confluência local e a terminação de um sistema de transformação. Um caso particular de confluência local é a sub-comutatividade (TC6), que pode ser vista na Figura 4.1(c). A presença de confluência e terminação garantem que o sistema de redução é convergente (TC7).

(a) propriedade Church-Rosser

(b) confluência

(c) confluência local

(d) sub-comutatividade

Figura 4.1: Propriedades relacionadas à confluência de sistemas de redução.

Os sistemas abstratos de redução constituem a base formal de muitos sistemas de reescrita de objetos, entre os quais a Álgebra de Processos e a Transformação de Grafos. A seguir, apresentamos uma das aplicações de sistemas abstratos de redução, cujo objetivo é fornecer uma abordagem algébrica para transformações em estruturas de grafos. A transformação de grafos é um dos instrumentos formais que serão usados para a caracterização de um sistema de redução, que será definido posteriormente, no Capítulo 5.

4.3 Transformação de Grafos

A abordagem algébrica de transformação de grafos é um método algébrico que relaciona regras de transformação topológica e propriedades algébricas sobre os grafos. O conceito de transformação de grafos é uma extensão do conceito de *gramáticas de grafos*, introduzido por Ehrig, Pfender e Schneider no início da década de 70 [8]. A gramática de grafos, por sua vez, é uma aplicação da Teoria de Categorias,

usada em Álgebra para definir classes de objetos que se relacionam a partir de morfismos e operações de composição. O objetivo inicial das gramáticas foi generalizar gramáticas de Chomsky, modelando *strings* por meio de grafos. Os fundamentos da transformação de grafos mostraram-se extremamente úteis como ferramentas de modelagem, de modo que sua utilização foi amplamente adotada em outras linhas de pesquisa, como análise de paralelismo de processos, avaliação de expressões funcionais, mecanismos de sincronização, sistemas distribuídos, sistemas orientados a objetos, implementação de tipos abstratos de dados e substituição de hiper-arcos livres de contexto [4].

Esta seção apresenta os conceitos da abordagem algébrica para transformação de grafos juntamente com exemplos, visando facilitar a visualização e aplicabilidade dos modelos. A abordagem algébrica adotada neste trabalho é baseada em *pushouts* duplos, que trabalham sobre morfismos totais e geralmente injetores. Morfismos parciais podem ser tratados de duas formas: por meio de adaptações que os transformem em totais [85], ou por meio de *pushouts* simples [86]. *Pushouts* simples são uma extensão de *pushouts* duplos usada exatamente para manipulação de morfismos parciais. A discussão sobre *pushouts* simples foge dos objetivos deste trabalho e, por isso, não serão abordados, mas maiores informações sobre essa técnica podem ser obtidos em [86, 87]. As definições apresentadas nesta seção são uma compilação de resultados apresentados em [4, 5, 10, 88]

4.3.1 Pushout Duplo

Um dos primeiros conceitos algébricos utilizados em transformação de grafos foi o *pushout* duplo, introduzido por Ehrig, em [8]. O termo *pushout* é usado para identificar uma estrutura algébrica que define um passo de derivação. A representação de um *pushout* se dá pela junção de grafos (*gluing*).

Antes de apresentar a definição de *pushout* duplo, é necessário definir alguns conceitos preliminares, como *grafos rotulados* e *produções*.

Definição 4.3.15: (*Grafo Rotulado*) Dados dois alfabetos fixos Ω_V e Ω_A , para rótulos de vértices e arcos, respectivamente, um *grafo rotulado* é uma tupla

$$G = (V, A, l_v, l_a),$$

onde V é o conjunto de vértices, A é o conjunto de arcos e as funções $l_v : V \rightarrow \Omega_V$ e $l_a : A \rightarrow \Omega_A$ são funções de rotulação sobre vértices e arcos.

Um *morfismo* $f : G \rightarrow G'$ é um par $f := (f_v : V(G) \rightarrow V(G'), f_a : A(G) \rightarrow A(G'))$ de funções que preserve os rótulos, ou seja, $l_v(G') \circ f_v = l_v(G)$ e $l_a(G') \circ f_a = l_a(G)$. Um morfismo f é chamado *isomorfismo* se as funções f_v e f_a são bijetoras. Neste caso, G e H são chamados *isomórficos*, e sua relação é representada por $G \cong H$.

A Figura 4.2 apresenta um grafo rotulado G . Neste caso, G é dirigido e possui quatro vértices, identificados pelos números 1, 2, 3 e 4, ou seja $V(G) = \{u, v, w, x\}$. A função l_v é responsável pela atribuição de rótulos nos vértices. Neste exemplo, $l_v(u) = 3$, $l_v(v) = 1$, $l_v(w) = 2$ e $l_v(x) = 4$. A função l_a mapeia os arcos de $A(G) = \{(v, w), (v, u), (w, u), (u, x)\}$ para os rótulos b, a, c e d , respectivamente.

O conjunto de todos os grafos rotulados define uma categoria de objetos. O próximo passo é definir

Figura 4.2: Grafo rotulado.

operações que relacionam esses objetos, indicando operações que causam mudanças estruturais. Isso pode ser obtido por meio de uma *produção*.

Definição 4.3.16: (*Produção*) Uma produção $p : (L \xleftarrow{l} K \xrightarrow{r} R)$ é composta por um *nome de produção* p e um par de morfismos de grafos $l : K \rightarrow L$ e $r : K \rightarrow R$. Os grafos L , K e R são chamados, respectivamente, de *lado esquerdo*, *interface* e *lado direito*.

Duas produções $p : (L \xleftarrow{l} K \xrightarrow{r} R)$ e $p' : (L' \xleftarrow{l'} K' \xrightarrow{r'} R')$ são chamadas *isomórficas por extensão* se existem três isomorfismos $L \xrightarrow{\phi^L} L'$, $K \xrightarrow{\phi^K} K'$ e $R \xrightarrow{\phi^R} R'$, tal que $\phi^L \circ l = l' \circ \phi^K$ e $\phi^K \circ r = r' \circ \phi^K$.

Quando não houver ambigüidades, uma produção $p : (L \xleftarrow{l} K \xrightarrow{r} R)$ pode ser identificada somente por p , ou também por $L \xleftarrow{l} K \xrightarrow{r} R$.

Figura 4.3: Exemplo de produção.

A produção abstrai o conceito de transformação, em que um grafo de interface K é a referência para a transformação do grafo da esquerda L no grafo da direita R . Para isso, utiliza-se os morfismos l e r . De fato, todos os elementos de K devem estar tanto em L quanto em R . Na aplicação da produção, todos os elementos de L que não estão em K (ou seja, $L - K$) devem ser removidos. Da mesma forma, todos os elementos de R que não estão em K (ou seja, $R - K$) devem ser criados. A Figura 4.3 apresenta

uma produção que transforma o grafo $L = (\{1, 2, 3\}, \{(1, 2), (2, 3)\})$, removendo o vértice 3 e seu arco incidente $(2, 3)$, e adicionado um arco reflexivo para os vértices restantes.

Chamamos de *gramática de grafos* $\mathcal{G}$ um par $\mathcal{G} = \langle (p : L \xrightarrow{l} K \xrightarrow{r} R)_{p \in P}, G_0 \rangle$, em que o primeiro componente é uma família de produções no conjunto P e G_0 é um grafo inicial. Quando o grafo G_0 é ignorado, considerando apenas a família de produções, $\mathcal{G}$ é chamado *sistema de transformação de grafos*.

Definição 4.3.17: (*Pushout*) Dados uma categoria de grafos $\mathcal{C}$, os grafos A, B, C e $D \in \mathcal{C}$; e duas funções $b : A \rightarrow B$ e $c : A \rightarrow C$, uma tripla $\langle D, g : B \rightarrow D, f : C \rightarrow D \rangle$ é chamada *pushout* de $\langle b, c \rangle$ se

- $g \circ b = f \circ c$ (comutatividade);
- Para todos os objetos D' e funções $g' : B \rightarrow D'$ e $f' : C \rightarrow D'$, com $g' \circ b = f' \circ c$, existe somente um $h : D \rightarrow D'$ tal que $h \circ g = g'$ e $h \circ f = f'$.

A Teoria de Categorias é uma generalização da Teoria de Conjuntos, onde são estudados objetos e morfismos entre objetos. Objetos podem ser entendidos como conjuntos estruturados e morfismos como funções de associação entre objetos. A única operação exigida em uma categoria é a composição de morfismos, que permite relacionar objetos por meio de seqüências de morfismos sobre eles. São exemplos de categorias amplamente utilizadas na Matemática os conjuntos e grupos. Quando falamos em uma categoria $\mathcal{C}$, ela representa o conjunto de digrafos, que podem ser relacionados por composições de morfismos. Maiores detalhes sobre a Teoria de Categorias podem ser encontrados em [89, 90].

Figura 4.4: *pushout*

Na Figura 4.4, D é chamado *objeto do pushout* de $\langle b, c \rangle$. Dadas as produções $b : A \rightarrow B$ e $g : B \rightarrow D$, o *complemento do pushout* de $\langle b, g \rangle$ é a tripla $\langle C, c : A \rightarrow C, f : C \rightarrow D \rangle$, tal que $\langle D, g, f \rangle$ é o *pushout* de $\langle b, c \rangle$. No caso, C é chamado *objeto do complemento do pushout* de $\langle b, c \rangle$.

O *pushout* estabelece a aplicação de dois morfismos sobre grafos. No exemplo da Figura 4.5, vemos um grafo A , que “produz” um grafo B por meio da transformação b e um grafo C por meio de uma transformação c . O *pushout* de $\langle b, c \rangle$ é a tripla formada por $\langle D, g, f \rangle$. Nesse exemplo, b adiciona um arco reflexivo ao vértice 2, e c adiciona um arco reflexivo ao vértice 3. Da mesma forma, f também adiciona um arco reflexivo no vértice 2 de C e g adiciona um arco reflexivo ao vértice 3, em B . Note que o *pushout* deste exemplo se encaixa nas propriedades fundamental e de comutatividade.

Figura 4.5: *Pushout*

A garantia de existência do *pushout* pode ser avaliada pela observação da seguinte proposição [4].

Proposição 4.3.1. (*Existência e unicidade de complementos de pushout*)

Sejam $b : A \rightarrow B$ e $g : B \rightarrow D$ dois morfismos. Então existe um complemento do pushout $\langle C, c : A \rightarrow C, f : C \rightarrow D \rangle$ se, e somente se as seguintes condições são satisfeitas:

- *Condição de pendência:* Nenhum arco de $D - g(B)$ é incidente a qualquer vértice de $V(B - b(A))$;
- *Condição de Identificação:* Não existe nenhum $x, y \in V(B) \cup A(B)$ tal que $g(x) = g(y)$ e $y \notin b(V(A) \cup A(A))$.

A aplicação de uma produção sobre um grafo, gerando um *pushout* é chamada derivação direta. A derivação direta é o conceito fundamental para definição de um sistema de transformação de grafos.

Definição 4.3.18: (*Derivação Direta*) Dado um grafo G , uma produção $p : (L \xleftarrow{l} K \xrightarrow{r} R)$ e um mapeamento (morfismo) $m : L \rightarrow G$, uma derivação direta de G para H usando p é uma produção que pode ser construída esquematicamente como na Figura 4.6. Nesse caso, o grafo D é chamado *grafo de contexto*, e serve para modelar o mapeamento entre o grafo de interface e os grafos G e H . A derivação direta desse caso é representada por $G \xrightarrow{p, m} H$, ou simplesmente por $G \xrightarrow{p} H$.

Dessa maneira, a construção do *pushout* pode ser interpretada, informalmente, da seguinte forma. Para aplicar uma produção p em um grafo G , é necessário achar uma ocorrência do seu grafo da esquerda (L), o casamento de $m : L \rightarrow G$. A seguir, remove-se de G todos os itens de L que não estão em K .

$$\begin{array}{ccccc}
p : L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
\downarrow m & & \downarrow d & & \downarrow m^* \\
p^* : G & \xleftarrow{l^*} & D & \xrightarrow{r^*} & H
\end{array}$$

Figura 4.6: Derivação direta modelada como *pushout* duplo

Para isso, deve-se encontrar um grafo D , tal que existam morfismos d e l^* que formam um *pushout* sobre D . Finalmente, adiciona-se, em D , os elementos que estão em R , mas não estão em K .

Figura 4.7: Derivação modelada como *pushout* duplo.

No exemplo, mostrado na Figura 4.7, o objetivo é definir uma transformação do grafo G no grafo H , usando a produção $p : L \leftarrow K \rightarrow R$. K é o grafo de interface que define o que vai ser removido do grafo L e o que vai ser adicionado no grafo R . A aplicação de p sobre G depende da ocorrência de um casamento $m : L \rightarrow G$, que determina, em G , um subdigrafo isomórfico a L . Note que o morfismo m , com $m(1) = 2, m(2) = 2, m(3) = 3$ não pode ser aplicado, pois a remoção de $m(3)$ fere a condição de pendência. Sendo assim, o casamento m usado foi $m(1) = 1, m(2) = 3, m(3) = 4$.

Uma vez definido m , remove-se de G todos os elementos de $m(L) - m(l(K))$, gerando o grafo D . Finalizando, o grafo H é produzido por meio da construção do *pushout* e $\langle H, m^*, r^* \rangle$.

O método é chamado *pushout* duplo porque toda derivação direta produz um estrutura semelhante à mostrada na Figura 4.6, onde é possível observar dois *pushouts*, ligados pelos grafos de interface K e

D.

Aplicando o conceito de *pushout* para derivações, temos um importante resultado, relacionado à comutatividade entre aplicações de derivação.

Teorema 4.3.2. *Comutatividade [4] Se $H_1 \xleftarrow{p_1, m_1} G \xrightarrow{p_2, m_2} H_2$ são derivações diretas independentes, então existe um H , tal que $H_1 \xrightarrow{p_3} H \xleftarrow{p_4} H_2$.*

A definição do *pushout* duplo, produção e derivação criam e fundamentam um sistema de redução. Para analisar as propriedades relacionadas à terminação e confluência, vale a pena aprofundar alguns aspectos que estudam o comportamento de conjuntos de derivações diretas. A seguir, são apresentadas as definições de derivações sequenciais e independência entre derivações.

Definição 4.3.19: (*Derivações Sequenciais*) Uma derivação sequencial é uma sequência de derivações $\xi = \{G_0 \xrightarrow{p_0} G_1 \xrightarrow{p_1} \dots \xrightarrow{p_{n-1}} G_n\}$, onde p_i é uma produção, para $0 \leq i < n$. Nesse caso, a derivação é escrita $G_0 \Rightarrow^* G_n$. Se $G \Rightarrow^* H$ é uma derivação, então os grafos G e H são chamados, respectivamente, de grafos *inicial* e *final* da derivação.

4.3.2 Independência de Derivações

Um aspecto extremamente importante no estudo de derivações, que fornece fundamentação para prova de várias propriedades importantes, é o conceito de independência entre derivações. Afirmar que duas derivações são independentes significa dizer que a aplicação de uma delas não exerce influência sobre a aplicação da outra. A independência de derivações pode ser interpretada de duas formas diferentes.

- independência paralela, quando duas derivações diretas do tipo $H_1 \xleftarrow{p_1, m_1} G \xrightarrow{p_2, m_2} H_2$ não são conflitantes;
- independência sequencial, quando duas derivações diretas consecutivas $G \xrightarrow{p_1, m_1} H_1 \xrightarrow{p_2, m_2} H_2$ não apresentam dependência de causalidade;

Definição 4.3.20: (*Independência Paralela*) Sejam $G \xrightarrow{p_1, m_1} H_1$ e $G \xrightarrow{p_2, m_2} H_2$ duas derivações diretas do grafo G . Tais derivações são ditas paralelamente independentes se $m_1(L_1) \cap m_2(L_2) \subseteq m_1(l_1(K_1)) \cap m_2(l_2(K_2))$.

Uma das consequências da definição acima é a existência de dois morfismos $L_1 \xrightarrow{k_2} D_2$ e $L_2 \xrightarrow{k_1} D_1$, tal que $l_2^* \circ k_2 = m_1$ e $l_1^* \circ k_1 = m_2$, que podem ser vistos na Figura 4.8.

Intuitivamente, duas derivações diretas são chamadas *paralelamente independentes* se qualquer uma delas pode ser aplicada após a outra. Isso significa dizer que a aplicação de uma não produz mudanças sobre a região do grafo em que a outra será aplicada. Dessa forma, a produção $G \xrightarrow{p_1, m_1} H_1$ não pode remover nenhum elemento necessário à aplicação da produção $G \xrightarrow{p_2, m_2} H_2$. Isso é representado pela exigência de que os elementos comuns mapeados de L_1 e L_2 em G também devem estar presentes em K_1 e K_2 .

Figura 4.8: Independência paralela de derivações diretas (Extraído de [4]).

Figura 4.9: Independência sequencial de derivações diretas (Extraído de [4]).

Definição 4.3.21: (*Independência Sequencial*) Uma dada derivação de dois passos $G \xRightarrow{p_1, m_1} H_1 \xRightarrow{p_2, m_2} H_2$ é chamada sequencialmente independente se e somente se $m_1^*(R_1) \cap m_2(L_2) \subseteq m_1^*(r_1(K_1)) \cap m_2(l_2(K_2))$.

A proposta da independência sequencial é semelhante à independência paralela, cujo procedimento é verificar se uma alteração na ordem de aplicação das produções mantém os mesmos resultados. Neste caso, a avaliação se dá pela análise do grafo H_2 , gerado pela sequência de derivações $G \xRightarrow{p_1, m_1} H_1 \xRightarrow{p_2, m_2} H_2$. Se for possível aplicar a regra p_2 antes de p_1 , obtendo uma sequência $G \xRightarrow{p_2, m_2} H_3 \xRightarrow{p_1, m_1} H_2$, então p_1 e p_2 são sequencialmente independentes (Figura 4.9).

4.3.3 Confluência e Terminação

A análise de independência entre regras de derivação fundamenta duas importantes propriedades: confluência e terminação. Os conceitos de confluência e terminação já foram discutidos anteriormente para sistemas abstratos de redução. A idéia conceitual dessas propriedades é preservada, porém a sua aplicação em sistemas de transformação de grafos depende, em alguns casos, de adaptações.

Um dos métodos mais comumente utilizados para avaliação de confluência em sistema de reescrita de termos é a análise de pares críticos. Para explicar rapidamente o conceito de pares críticos, vamos analisar o seguinte exemplo, adaptado de [83].

Suponha que exista uma máquina de café que aceite dois tipos de moedas: uma branca ($\circ$) e uma preta ($\bullet$). Considere, ainda, que a máquina seja capaz de interpretar algumas sequências de moedas, obedecendo às seguintes regras:

$$\begin{array}{lcl} \bullet\circ & \rightarrow & \circ\circ\circ\bullet \\ \circ\bullet & \rightarrow & \bullet \end{array}$$

Considere, agora, que a sequência de moedas $\bullet\circ\bullet$ foi enviada para a máquina de café. A aplicação da primeira regra sobre essa sequência produz uma nova sequência $\circ\circ\circ\bullet\bullet$. A aplicação da segunda regra produz a sequência $\bullet\bullet$.

Como podemos observar, duas regras diferentes podem ser aplicadas sobre a mesma sequência de moedas, produzindo resultados diferentes. Em sistema de reescrita de termos, dizemos que as duas regras produziram um *par crítico* $\langle \circ\circ\circ\bullet\bullet, \bullet\bullet \rangle$. Este par crítico é dito *combinável* (*joinable*), pois a aplicação sucessiva de regras sobre a primeira sequência converge para a segunda.

$$\circ\circ\circ\bullet\bullet \rightarrow \circ\circ\bullet\bullet \rightarrow \circ\bullet\bullet \rightarrow \bullet\bullet$$

Um dos principais teoremas de confluência de sistemas de reescrita de termos afirma que, se todos os pares críticos forem combináveis, o sistema é confluente. Vamos utilizar o mesmo raciocínio, direcionando o mesmo conceito para transformação de grafos. Em transformação de grafos, um par crítico é definido da seguinte maneira:

Definição 4.3.22: (*Par Crítico*) Sejam duas produções $p_i = L_i \leftarrow K_i \rightarrow R_i$, para $i = 1, 2$. Um par de derivações diretas da forma $H_1 \xleftarrow{p_1, m_1} G \xrightarrow{p_2, m_2} H_2$ é um par crítico se $m_1(L_1) \cap m_2(L_2) \neq m_1(l_1(K_1)) \cap m_2(l_2(K_2))$. Além disso, quando $p_1 = p_2$, exige-se que $m_1 \neq m_2$.

Definição 4.3.23: (*Par crítico combinável*) Um par crítico $T_1 \leftarrow S \Rightarrow T_2$ é combinável se existir um grafo X , tal que $T_1 \Rightarrow^* X_1 \cong X_2^* \leftarrow T_2$.

Em [10], há um importante resultado, que mostra que um par crítico combinável não é uma condição suficiente para garantir a confluência. A Figura 4.10 ilustra um exemplo dessa situação. No exemplo, temos duas produções p_1 e p_2 , que possuem o mesmo lado esquerdo, indicando que qualquer uma das produções pode ocorrer sobre o mesmo subdigrafo. Isso claramente produz um par crítico, mostrado na Figura 4.11, que leva o digrafo a duas diferentes produções. Note que este par crítico é combinável, pois seus resultados são isomórficos. Apesar disso, não há confluência, pois p_1 e p_2 realizam mudanças em vértices diferentes.

De forma genérica, classificar todos os pares críticos como combináveis não é garantia de confluência porque as derivações $S \Rightarrow T_1 \Rightarrow^* X_1$ e $S \Rightarrow T_2 \Rightarrow^* X_2$ podem ser isomórficas mapeando o mesmo vértice

Figura 4.10: Produções diferentes que atuam sobre o mesmo *redex* [5].

de S para vértices diferentes de X_1 e X_2 .

Figura 4.11: Par crítico formado pelas produções da Figura 4.10 [5].

Se, entretanto, for possível garantir que as produções $S \Rightarrow T_1 \Rightarrow^* X_1$ e $S \Rightarrow T_2 \Rightarrow^* X_2$, com $X_1 \cong X_2$ constroem morfismos que conservam o mapeamento dos vértices de S entre X_1 e X_2 , então o *Lema do Par Crítico* é válido.

Lema 4.3.3. (*Lema do Par Crítico* [10]) *Um sistema de redução de grafos é localmente conflúente se todos os seus pares críticos são combináveis.*

Uma consequência direta do Lema 4.3.3 é que um sistema de redução sem pares críticos também é conflúente. Para provar isso, vamos utilizar um resultado obtido por Ehrig, em [91].

Teorema 4.3.4. *Teorema da Comutação* [91] *Sejam $H_1 \xleftarrow{p_1, m_1} G \xrightarrow{p_2, m_2} H_2$ derivações diretas obtidas das produções $p_i = (L_i \leftarrow K_i \rightarrow R_i)$, para $i = 1, 2$. Se $m_1(L_1) \cap m_2(L_2) = m_1(l_1(K_1)) \cap m_2(l_2(K_2))$, então existe um grafo M , tal que $H_1 \xrightarrow{p_2} M \xleftarrow{p_1} H_2$.*

Demonstração. (Contradição sobre igualdade de M_1 e M_2) Mostraremos que $H_1 \xrightarrow{p_2} M \xleftarrow{p_1} H_2$ por contradição sobre o resultado das derivações diretas sobre H_1 e H_2 .

Assuma a existência das derivações diretas $H_1 \xrightarrow{p_2} M_1$ e $H_2 \xrightarrow{p_1} M_2$, e assumamos ainda que $M_1 \neq M_2$. Então $H_1 \xrightarrow{p_2} M_1$ e $H_2 \xrightarrow{p_1} M_2$ não são paralelamente independentes. Isso é uma contradição, pois, por hipótese, $m_1(L_1) \cap m_2(L_2) \subseteq m_1(l_1(K_1)) \cap m_2(l_2(K_2))$. ■

Corolário 4.3.5. *Sistemas de transformação de grafos sem pares críticos são confluentes.*

Demonstração. (Contradição sobre o Teorema da Comutação) Seja um sistema sem pares críticos que não seja confluente. Então, para qualquer par de derivações diretas $H_1 \xleftarrow{p_1} D \xrightarrow{p_2} H_2$, obtidas pela aplicação das produções $p_i = L_i \leftarrow K_i \rightarrow R_i$, para $i = 1, 2$. Como $m_1(L_1) \cap m_2(L_2) = m_1(l_1(K_1)) \cap m_2(l_2(K_2))$, então, pelo Teorema da Comutação, existe um par de derivações diretas da forma $H_1 \xrightarrow{p_1} M \xleftarrow{p_2} H_2$, o que é uma contradição, pois o sistema não é confluente. ■

Corolário 4.3.6. *Derivações paralelamente independentes não geram pares críticos.*

Demonstração. Sejam G , H_1 e H_2 grafos e duas derivações $G \xrightarrow{p_1, m_1} H_1$ e $G \xrightarrow{p_2, m_2} H_2$ paralelamente independentes, tal que H_1 e H_2 formam um par crítico. Assuma que existe um *pushout* duplo para cada derivação, como na Figura 4.8. Então $m_1(L_1) \cap m_2(L_2) \neq m_1(l_1(K_1)) \cap m_2(l_2(K_2))$, pela Definição 4.3.3 e existe algum elemento em $m_1(L_1) \cap m_2(L_2)$ que não está em $m_1(l_1(K_1)) \cap m_2(l_2(K_2))$, o que é uma contradição, pois pela Definição 4.3.2, $m_1(L_1) \cap m_2(L_2)$.

A ausência de pares críticos é uma característica de sistemas de redução com produções independentes. Dessa forma, podemos associar diretamente a presença de pares críticos à independência entre produções e, conseqüentemente, à confluência do sistema global.

Se, para a confluência, temos alguns resultados que nos ajudam a caracterizar sistemas de transformação de grafos, o mesmo não pode ser dito em relação à terminação. Mesmo em sistemas de reescrita mais simples, como *strings* e termos, a terminação não é uma propriedade de fácil verificação. Isso pode ser ilustrado com um exemplo simples.

No exemplo da máquina de café, apresentado anteriormente, nota-se que existem duas produções. A primeira delas aumenta o número de moedas, enquanto a outra o diminui. Em sistemas como esse, é difícil garantir que exista um número finito de derivações para cada seqüência de moedas, pois a primeira produção, ao aumentar o número de moedas, pode aumentar também o número de posições onde novas regras (ou talvez a mesma) possam ser aplicadas. O sistema do café que apresentamos nessa subseção é finalizável, pois a aplicação da primeira regra não cria novos pontos de aplicação, mas a verificação da terminação de sistemas não é simples.

Transferindo o raciocínio para grafos, a definição de um método para avaliar a terminação de um sistema é ainda um problema aberto. Em geral, a terminação é uma propriedade de difícil verificação e indecidível [92–94]. Apesar disso, alguns trabalhos têm concentrado esforços para isolar situações em que é possível afirmar que o sistema é finalizável [95, 96].

4.4 Conclusão

A transformação de grafos é uma abordagem algébrica que procura explorar propriedades algébricas a partir de mudanças estruturais e semânticas em grafos. As mudanças estruturais ocorrem por meio de alterações topológicas na estrutura do grafo, enquanto as semânticas ocorrem por mapeamentos entre rótulos de vértices e arcos. A base formal usada vem de conceitos relacionados a sistemas abstratos de redução, onde operações são definidas sobre conjuntos de objetos.

A utilização de *pushouts* duplos permite analisar a relação entre transformações de um sistema, identificando propriedades importantes como confluência e terminação. Outro conceito importante introduzido no capítulo é o par crítico, que aponta transformações conflitantes e fundamentam métodos de avaliação de confluência.

Os conceitos apresentados neste capítulo serão usados para definir um sistema de redução para geração de expressões algébricas de digrafos acíclicos. Além disso, será possível provar que o sistema é correto, finalizável e confluyente, podendo ser adotado para qualquer digrafo acíclico.

Capítulo 5

Digrafos de Expressão

5.1 Introdução

Como visto nos Capítulos 2, a classe dos sp-digrafos possui propriedades topológicas extremamente favoráveis à aplicação de técnicas de redução. Além disso, é possível caracterizar algebricamente as reduções topológicas sobre digrafos usando *pushouts* de transformação de grafos. Este capítulo apresenta uma forma de associar reduções topológicas de digrafos a transformação de digrafos compostos por termos algébricos em ACP. A representação de expressões algébricas em grafos ocorrerá por meio da construção de digrafos de expressão, que serão definidos na Seção 5.2. Em seguida, na Seção 5.3, são apresentadas as operações de redução sobre digrafos de expressão, seus aspectos algébricos e seus *pushouts*. Também são discutidas algumas propriedades sobre o sistema de redução, como confluência e terminação. O tratamento de algumas construções que geram conflitos é abordado na Seção 5.4 e, finalmente, as conclusões do capítulo são apresentadas na Seção 5.5.

5.2 Digrafos de Expressões Algébricas

A utilização de digrafos, segundo sua definição original, não é capaz de representar completamente o comportamento de processos de negócio. Em um processo de negócio, é comum encontrar subconjuntos de atividades que podem ser executados, ou paralelamente, ou alternativamente mediante a satisfação de uma condição pré-definida. Não é possível representar tais situações em digrafos comuns, pois vértices e arcos são capazes de representar somente precedências e dependências. Para modelagem de processos de negócio, também se faz necessário representar regras de habilitação e execução. Para cobrir essa necessidade, este trabalho estenderá o conceito de digrafos, agregando regras semânticas a seus elementos, permitindo identificar situações nas quais o fluxo de execução possa seguir caminhos paralelos ou alternativos, além do tradicional encadeamento seqüencial de atividades.

As modificações que possibilitarão ao digrafo representar processos de negócio são duas funções de rotulação, uma para vértices e outra para arcos. Os rótulos de arcos e vértices são construções de álgebra de processos. Dessa forma, cada arco será rotulado com um termo algébrico, enquanto cada vértice será rotulado com um operador de composição algébrica. A composição algébrica pode ser

seqüencial, paralela ou alternativa, de maneira que será possível representar, além de dependências e precedências, as regras que regem o comportamento do processo de negócio.

Claramente, a ambigüidade de interpretação das regras de execução paralela e alternativa ocorrem nos pontos de ramificação e sincronização do processo. A mudança proposta por este trabalho é semelhante à abordagem adotada por digrafos E/OU [62, 97], muito usados em modelos de resolução de problemas e modelos de processos de produção e manufatura. Digrafos E/OU também são bastante conhecidos e possuem algoritmos eficientes para problemas de busca, identificação de caminhos críticos e fluxos máximos aditivos [98]. A principal diferença é que digrafos E/OU foram modelados para representação de expressões lógicas conjuntivas e disjuntivas, não possuindo instrumentos para representação algébrica de composição de atividades em processos de negócio. Uma outra diferença marcante é que digrafos E/OU são modelados a partir de uma expressão lógica válida, de modo que sempre podem ser modeladas em forma de árvore, o que não ocorre com processos de negócio. A seguir, a definição de digrafos de expressão.

Definição 5.2.24: (*Digrafo de Expressão Algébrica*) Um digrafo de expressão algébrica, ou simplesmente **digrafo de expressão**, é um digrafo acíclico, com uma fonte e um sorvedouro, $D = (V, A, \pi, \theta)$, em que:

- $V = V_t \cup V_o$ é o conjunto de vértices, tal que V_t é o conjunto de vértices terminais, e V_o o de vértices não-terminais;
- $A = V \times V$ é o conjunto de arcos;
- $\pi : V_o \rightarrow \{+, \parallel, \cdot, | \parallel\}$ é uma função de rotulação que mapeia cada vértice não-terminal em um operador algébrico de composição;
- $\theta : A \rightarrow \mathcal{T}(\Sigma^*) \cup \{\lambda\}$ é uma função de rotulação que mapeia cada arco em termos algébricos;

V_t é o conjunto de vértices terminais, ou seja, vértices fonte ($\{v \in V(D) : d^-(v) = 0\}$) ou sorvedouro ($\{v \in V(D) : d^+(v) = 0\}$). Todos os demais vértices, ditos não-terminais, formam o conjunto V_o . A função π mapeia cada vértice não-terminal a um operador algébrico de composição. Este trabalho considera os operadores de composição seqüencial ($\cdot$), alternativa ($+$) e paralela ($\parallel$). A função θ mapeia todo arco em $A(D)$ num termo fechado ou numa ação vazia, representada por λ .

A ação vazia λ tem como objetivo representar ações neutras, sem participação funcional na dinâmica de execução do processo de negócio modelado pelo grafo de expressão. Sendo assim, λ é uma ação de execução atômica e instantânea, que não consome tempo e possui as propriedades descritas na Tabela 5.1 (considere $a \in \mathcal{T}(\Sigma^*) \cup \{\lambda\}$):

Como qualquer outra ação atômica, λ obedece aos axiomas da álgebra de processos, apresentadas no Capítulo 3. Além disso, a ação λ é o elemento neutro, que será adotado para a composição seqüencial (P1) e a composição paralela (P4). Aplicando os axiomas apresentados no Capítulo 3, é possível mostrar que a composição paralela (P4) é uma consequência direta de P2 e P3, pois

Tabela 5.1: Propriedades da ação vazia λ

$$\begin{aligned}
\text{P1: } & a.\lambda = \lambda.a = a \\
\text{P2: } & a|\lambda = \lambda|a = a \\
\text{P3: } & a \parallel \lambda = \lambda \parallel a = a \\
\text{P4: } & a \parallel \lambda = \lambda \parallel a = a
\end{aligned}$$

$$\begin{aligned}
a \parallel \lambda &= a \parallel \lambda + \lambda \parallel a + a|\lambda \quad (\text{por I1}) \\
&= a + a + a \quad (\text{por P2 e P3}) \\
&= a \quad (\text{por A3})
\end{aligned}$$

Para a composição alternativa entre um termo, digamos a , e λ , o comportamento esperado é uma decisão entre duas transições, uma em que o termo inicial é a e outro, não. Sendo assim, a expressão

$$a + \lambda$$

indica que, ou o termo a será executado, ou nenhuma ação será executada. Nesse caso, λ não pode ser retirado da expressão, pois sua retirada modifica a semântica da expressão. Esse comportamento foi definido em conformidade com os axiomas da APB.

Dado um digrafo de expressão D , $\text{termo}(D)$ representa a expressão algébrica equivalente a D . Um morfismo $f : D \rightarrow D'$ é um par $f := (f_v : V(D) \rightarrow V(D'), f_a : A(D) \rightarrow A(D'))$ de funções que preserva os rótulos de vértices e arcos. Dessa maneira, $\pi(D') \circ f_v = \pi(D)$ e $\theta(D') \circ f_a = \theta(D)$. No caso da função θ , assumimos que $\theta(D) = \theta(D')$ significa que os rótulos dos arcos de D são bissimilares aos rótulos de seus respectivos arcos em D' . Chamamos f de isomorfismo se as funções f_v e f_a são bijetoras. Neste caso, D e D' são chamados *isomórficos*, e sua relação é representada por $G \cong H$.

Uma vez definido o conceito de digrafo de expressão, vamos agora definir um sp-digrafo de expressão. A importância do sp-digrafo de expressão é a caracterização topológica de um digrafo capaz de representar uma expressão algébrica com alto nível de equivalência (exceto em casos de conflito estrutural, como veremos mais adiante).

Definição 5.2.25: (*SP-Digrafo de Expressão Algébrica*) O sp-digrafo de expressão é um digrafo de expressão construído recursivamente a partir das seguintes regras:

- Um digrafo formado por dois vértices unidos por um único arco é um sp-digrafo de expressão, chamado sp-digrafo de expressão trivial;
- Sejam D_1 e D_2 sp-digrafos de expressão, e sejam s_i e t_i , respectivamente, os vértices fonte e sorvedouro de D_i ($i = 1, 2$). Também será um sp-digrafo de expressão o resultado a aplicação de uma das composições a seguir:

(a) composição seqüencial:

$$* \quad V(D) \leftarrow V(D_1) \cup V(D_2);$$

- * $A(D) \leftarrow A(D_1) \cup A(D_2)$;
- * criar identidade entre t_1 e s_2 ;
- * $\pi(t_1) = \pi(s_2) = \cdot$.

(b) composição paralela:

- * $V(D) \leftarrow V(D_1) \cup V(D_2) \cup \{s, t\}$;
- * $A(D) \leftarrow A(D_1) \cup A(D_2) \cup \{(s, s_1), (t_1, t)\}$;
- * criar identidade entre s_1 e s_2 , assim como t_1 e t_2 ;
- * $\pi(s) = \pi(t) = \parallel$;

(c) composição alternativa:

- * $V(D) \leftarrow V(D_1) \cup V(D_2) \cup \{s, t\}$;
- * $A(D) \leftarrow A(D_1) \cup A(D_2) \cup \{(s, s_1), (t_1, t)\}$;
- * criar identidade entre s_1 e s_2 , assim como t_1 e t_2 ;
- * $\pi(s) = \pi(t) = +$;

Figura 5.1: Construção de sp-digrafos de expressão: (a) sp-digrafos de expressão D_1 e D_2 ; (b) Composição seqüencial de D_1 e D_2 ; (c) Composição paralela de D_1 e D_2 ; (d) Composição alternativa de D_1 e D_2 .

A Figura 5.1 ilustra as operações de composição que podem originar sp-digrafos de expressão. A mudança, em relação à definição original de sp-digrafos, é que arcos e vértices são rotulados a partir das funções θ e π , respectivamente. Os rótulos dos arcos são mapeados para termos, que representam expressões algébricas, ou para λ . Os rótulos dos vértices não-terminais são mapeados para operadores algébricos de composição.

Vértices não-terminais com grau de entrada e saída iguais a um são mapeados para o operador de composição seqüencial, ou seja, se $v \in V : d^-(v) = d^+(v) = 1$, então $\pi(v) = \cdot$. Já os demais vértices não-terminais, são mapeados, ou para o operador de composição alternativa (+), ou para o de composição paralela ($\parallel$).

Figura 5.2: Particionamento de vértices não-terminais

Reforçando a nomenclatura que foi usada na Seção 2.3, vértices com grau de saída maior que um são chamados **vértices de ramificação**, enquanto vértices com grau de entrada maior que um são chamados **vértices de sincronização**.

Vértices com grau de entrada e saída maiores que 1 são, ao mesmo tempo, vértices de ramificação e sincronização. Para esses vértices, tanto a regra de ramificação, quanto a regra de sincronização seguem o valor de π . Por exemplo, se $\pi = \text{"||"}$, o vértice sincronizará paralelamente seus arcos de entrada e ramificará paralelamente seus arcos de saída. Se, entretanto, as regras de ramificação e sincronização forem diferentes, é necessário particionar o vértice, como mostrado na Figura 5.2

A divisão de vértices pode ser feita da seguinte maneira. Seja D um digrafo de expressão e $v \in V$ um vértice com graus de entrada e saída maiores que 1, tal que $d^-(v) > 1$ e $d^+(v) > 1$. Para realizar a partição de v , vamos construir um novo digrafo D_P , onde $V(D_P) = V(D) \setminus \{v\} \cup \{v_1, v_2\}$ e $A(D_P) = A(D) \cup \{(v_1, v_2)\}$. Sejam x e y vértices de D ; para cada arco xv em $A(D)$, existe um arco xv_1 em $A(D_P)$. Para todo arco vy em $A(D)$, existe um arco v_2y em $A(D_P)$. Todos os demais arcos de D estão em D_P . Finalmente, $\theta(v_1v_2) = \lambda$.

Os vértices terminais não podem ser vértices de ramificação, nem sincronização. Dessa forma, o vértice fonte terá grau de saída com valor igual a 1, assim como o grau de entrada do vértice sorvedouro.

Outro aspecto que deve ser levado em consideração é a combinação de vértices de ramificação (rótulo "+") e de sincronização (rótulo "||"), e vice-versa. Na seção seguinte, veremos que esta construção nem sempre gera estruturas e mapeamentos válidos de processos de negócio, o que pode caracterizar a presença de erros na modelagem do processo de negócio e produzir situações inesperadas durante a execução.

5.2.1 Habilitação e Execução

O digrafo de expressões representa não somente a organização do processo de negócio do ponto de vista topológico, mas também o esquema de execução de suas instâncias. A execução das instâncias deve seguir regras representadas pelos operadores algébricos mapeados em cada vértice. Cada termo algébrico, representado por um arco, é relacionado com os termos sucessores ou predecessores por meio de interpretação da regra de composição do vértice de sua vizinhança comum.

A visualização do mecanismo de execução de cada sistema está vinculada aos conceitos de **habi-**

Figura 5.3: Digrafo de expressão para o termo $\theta(a) + \theta(b)$

litação e execução de termos. Um termo pode ser *executado* somente se estiver *habilitado*. A execução de um termo representa a realização de suas ações internas, obedecendo às regras e a ordem de composição dadas por seus operadores.

A habilitação representa um estado do sistema, onde é permitida a execução de um conjunto de termos. Afirmar que um termo está habilitado significa dizer que o estado atual do sistema permite a execução desse termo. A habilitação de um termo depende da execução dos termos anteriores e da observação das regras de execução determinadas pelo rótulo de cada vértice.

Vale a pena fazer uma breve ressalva sobre a relação entre os operadores internos aos termos e os operadores mapeados para vértices do digrafo. Embora usados em estruturas com características diferentes (termos algébricos e vértices de grafos), seu objetivo é o mesmo: a representação de uma composição entre dois termos algébricos. Para os termos, essa representação segue as regras já definidas em álgebra de processos. Para grafos, a representação se adequa às características conceituais da estrutura de grafos, principalmente em relação ao comportamento de ramificações e sincronizações.

Vamos ilustrar esta relação com um exemplo. Sejam a e b dois arcos de um digrafo de expressão, e $\theta(a)$ e $\theta(b)$ seus respectivos termos. o termo $\theta(a) + \theta(b)$ representa a execução alternativa de $\theta(a)$ ou $\theta(b)$. No digrafo de expressão, $\theta(a) + \theta(b)$ é representado por uma ramificação alternativa, seguida de uma sincronização alternativa, como ilustrado na Figura 5.3.

Note que, no termo $\theta(a) + \theta(b)$, o operador de composição alternativa, $+$, dita, ao mesmo tempo, as regras de ramificação e sincronização dos termos $\theta(a)$ e $\theta(b)$. No digrafo, entretanto, a representação de $\theta(a) + \theta(b)$ precisa da representação explícita do operador “ $+$ ” em dois pontos: na ramificação e na sincronização. Isso ocorre porque a estrutura do digrafo exige explicitamente os pontos de ramificação e sincronização, ao passo que, na expressão algébrica, os pontos de ramificação e sincronização são definidos implicitamente pelos parâmetros do operador de composição.

Apesar dessa diferença de interpretação, que está mais relacionada às características de cada estrutura de modelagem (digrafo ou álgebra), enfatizamos que o objetivo é utilizar os conceitos envolvidos em ambas para representar uma expressão algébrica por meio do digrafo e vice-versa. Foram utilizados operadores algébricos como rótulos de vértices porque, de fato, os vértices originarão uma expressão algébrica após operações de redução.

Sejam a , b e c arcos de um digrafo de expressão, e $\theta(a)$, $\theta(b)$, $\theta(c)$ seus respectivos termos; e seja v um vértice não-terminal. Indicamos a habilitação de uma ação a partir das seguintes situações:

- se $\pi(v) = \cdot$, $\theta(b)$ é habilitado após a execução do termo $\theta(a)$ que domina v (ver Figura 5.4(a));
- se v é vértice de sincronização e $\pi(v) = +$, $\theta(b)$ é habilitado após a execução de algum termo

que domina v . Neste caso, a execução de $\theta(a)$ ou $\theta(c)$ é suficiente para habilitar $\theta(b)$ (ver Figura 5.4(b));

- se v é vértice de sincronização e $\pi(v) = “\parallel”$, $\theta(b)$ é habilitado somente após a execução de todos os termos que dominam v . Neste caso, a habilitação de $\theta(b)$ exige a execução tanto de $\theta(a)$, quanto de $\theta(c)$ (ver Figura 5.4(c));
- se v é vértice de ramificação e $\pi(v) = “+”$, todos os seus termos dominados serão habilitados após a execução do termo dominante de v , porém somente um dos termos habilitados poderá ser executado. Neste caso, a habilitação de $\theta(b)$ e $\theta(c)$ ocorre após a execução de $\theta(a)$. A execução de um dos termos habilitados desabilita os demais (ver Figura 5.4(d));
- se v é vértice de ramificação e $\pi(v) = “\parallel”$, todos os seus termos dominados serão habilitados após a execução do termo dominante de v . Ao contrário do caso anterior, todos os termos habilitados deverão ser executados. Neste caso, a execução de $\theta(a)$ habilita a execução paralela e independente de $\theta(b)$ e $\theta(c)$.

Figura 5.4: Regras de habilitação de vértices e arcos

O último caso que merece comentário diz respeito a termos ligados ao vértice fonte. Se um termo tem seu arco com cauda no vértice fonte, ele estará habilitado no início de cada instância de execução.

Obedecendo às regras de habilitação e execução, o digrafo de expressão pode ser visto como um sistema de estados e transições. Por meio da observação de uma dada sequência de execução de termos (representados nos arcos), é possível montar a árvore de processos e encontrar, conseqüentemente, todos os rastros do sistema. O exemplo da Figura 5.5 acompanha a evolução do sistema de acordo com a execução de termos. Os elementos pontilhados (arcos e vértices) representam pontos do processo que não podem mais ser habilitados a partir do estado atual.

No estado inicial, S_0 , somente o termo a está habilitado. A execução de a leva o sistema ao estado S_1 , onde os termos b e c estão habilitados, porém somente um deles será executado (ramificação alternativa). A execução de c leva ao estado S_2 , e a execução de b leva ao estado S_3 . Caso c seja executado, o termo seguinte é d , que leva também ao estado S_3 . Em S_3 , só o termo e pode ser executado, levando ao estado S_4 . A composição paralela em S_4 pode levar o fluxo para os estados S_5 , pela execução de f , ou S_7 , pela

Figura 5.5: Seqüência de execuções em digrafos de expressão

execução de g . Em S_5 ou S_7 , o fluxo aguarda a execução do outro ramo paralelo, levando ao estado S_6 , onde o termo h está habilitado. Após a execução de h , o fluxo finaliza no estado S_8 .

O grafo de processo da Figura 5.5 é exibido na Figura 5.6. Nela, é possível ver de forma mais clara a evolução dos estados em relação à seqüência de execução dos termos. Na Figura 5.6(a), o grafo de processos ganha a forma de uma árvore, onde cada caminho entre a raiz e uma das folhas determina um rastro válido para o sistema. Uma versão simplificada do mesmo grafo é mostrada na Figura 5.6(b), onde os estados de sincronização são mais explícitos. A grafo de processo simplificado tem o mesmo objetivo do grafo original, porém representa cada estado uma única vez, permitindo visualizar a interação entre os estados do processo de negócio. Nesse caso, os rastros são definidos por todos os caminhos possíveis entre os vértices fonte (S_0) e sorvedouro (S_8).

Figura 5.6: Grafo de processos: em (a), grafo de processos da Figura 5.5; em (b), versão simplificada do grafo de processos.

5.3 Redução de Digrafos de Expressão

Este trabalho definirá, agora, um sistema de redução de digrafos de expressão baseado em conceitos da abordagem algébrica para transformação de grafos, apresentada no Capítulo 4.

Seja $\mathbb{T}(\Sigma)$ o conjunto de termos abertos sobre a assinatura Σ . Uma substituição é um mapeamento $\sigma : \mathbb{T}(\Sigma) \rightarrow \mathbb{T}(\Sigma)$, tal que $\sigma(c) = c$ para toda constante c , e $\sigma(f(t_1, \dots, t_n)) = f(\sigma(t_1), \dots, \sigma(t_n))$ para todo termo $f(t_1, \dots, t_n)$.

Uma regra de reescrita de termos é um par $\langle l, r \rangle$ de termos, representado por $l \rightarrow r$, tal que

- (a) l não é variável;
- (b) todas as variáveis em r estão também em l .

Embora, pela restrição (a), l não possa ser uma variável, l pode ser uma função sobre Σ , cujos argumentos são variáveis. Essa restrição é utilizada em sistemas de reescrita de termos para garantir a *terminação* do sistema de regras [99].

Um Sistema de Reescrita de Termos $\langle \Sigma, \mathcal{R} \rangle$ consiste em um conjunto Σ de símbolos de função e um conjunto $\mathcal{R}$ de regras sobre $\mathbb{T}(\Sigma)$.

Definição 5.3.26: (*Reescrita de Termos*) A relação de reescrita $l \rightarrow r$ sobre $\mathbb{T}(\Sigma)$, induzida pelo STR $\mathcal{R}$ é definido como:

$t \rightarrow u$, se existe uma regra $l \rightarrow r$ em $\mathcal{R}$ e uma substituição σ , tal que:

1. $\sigma(l)$ é um subtermo de t ;
2. u é obtido de t pela substituição da ocorrência de $\sigma(l)$ por $\sigma(r)$.

O objetivo, agora, é estender o conceito de reescrita de termos para digrafos de expressão.

Definição 5.3.27: (*Redex*)

Dado um digrafo de expressão D , um par de vértices $\langle u : v \rangle$, com $u, v \in V$ formam um *redex* se u é fonte e v é sorvedouro de um subdigrafo que satisfaz uma das seguintes condições:

- (a) existe um vértice w , tal que $\{(u, w), (w, v)\} \subseteq A(D)$ e $d^+(w) = d^-(w) = 1$. Chamamos esta situação de *redex seqüencial*;
- (b) existe mais de um arco entre u e v em D , e $\pi(u) = \pi(v)$. Chamamos esta situação de *redex de ramificação e sincronização*.

Em um SRT, um termo pode ser reescrito por meio de repetidas substituições de uma instância do lado esquerdo de uma equação por sua correspondente instância do lado direito. O lado esquerdo da equação é chamado *redex* e o lado esquerdo *reduct*. A aplicação contínua leva o termo a um outro termo, chamado forma normal, que não contém nenhum *redex*. Tomaremos emprestada a idéia de *redex* [100], adaptando-a para grafos de expressão.

Do ponto de vista de transformação de grafos, o *redex* é equivalente ao morfismo de casamento do lado esquerdo de um *pushout* duplo. O *redex* representa a região em que pode ser aplicada uma transformação sobre o grafo.

Definição 5.3.28: (*Redução de digrafos de Expressão*)

Seja D um digrafo de expressão contendo um *redex* $\langle v_1 : v_2 \rangle$. Então, existe uma produção $D \xrightarrow{p, \langle v_1 : v_2 \rangle} D'$, onde D' é o digrafo de expressão construído a partir de um dos procedimentos a seguir:

1. *Redução Seqüencial:* Na redução seqüencial, existe um vértice w entre u e v , tal que $d^+(w) = d^-(w) = 1$. A redução segue o seguinte procedimento:

- (a) $A(D) \leftarrow A(D) \cup \{(u, v)\}$;

- (b) $\theta((u, v)) = \theta((u, w)) \cdot \theta((w, v))$;
- (c) $A(D) \leftarrow A(D) \setminus \{(u, w), (w, v)\}$.

2. *Redução de Ramificação e Sincronização*: Nesta situação, $\| (uv) \| > 1$. Sejam $a_1, a_2, \dots, a_k$ os arcs entre u e v em D . Então, podemos classificar duas situações:

- *Redução Alternativa*: Nesta situação, $\pi(u) = \pi(v) = +$. Nesta situação, a redução adota o seguinte procedimento:
 - (a) $A(D) \leftarrow A(D) \cup \{a_0 = (u, v)\}$;
 - (b) $\theta(a_0) = \theta(a_1) + \theta(a_2) + \dots + \theta(a_k)$;
 - (c) $A(D) \leftarrow A(D) \setminus \{a_1, a_2, \dots, a_k\}$
- *Redução Paralela*: Nesta situação, $\pi(u) = \pi(v) = \parallel$. Nesta situação, a redução adota o seguinte procedimento:
 - (a) $A(D) \leftarrow A(D) \cup \{a_0 = (u, v)\}$;
 - (b) $\theta(a_0) = \theta(a_1) \parallel \theta(a_2) \parallel \dots \parallel \theta(a_k)$;
 - (c) $A(D) \leftarrow A(D) \setminus \{a_1, a_2, \dots, a_k\}$

A aplicação de uma produção $D \Rightarrow D'$ sobre um *redex* $\langle u : v \rangle$ é chamada *redução*, ou *derivação* de D em D' .

Lema 5.3.1. (*Corretude da Redução de Digrafos de Expressão*) Para todos os digrafos de expressão D e H , $D \Rightarrow H$ implica que $\text{termo}(D)$ é bissimilar ao $\text{termo}(H)$.

Figura 5.7: Redução dos termos E_1 e E_2 usando digrafos de expressão; (a) Redução seqüencial; (b) Redução alternativa; (c) Redução paralela

Demonstração. A igualdade, nesse caso, se refere à equivalência de bissimulação entre os termos gerados pelos digrafos de expressão D e H . Para provar que as expressões de D e H produzem termos bissimilares, vamos dividir a prova em 3 situações, que podem ser acompanhadas na Figura 5.7:

- redução seqüencial: ilustrada pela Figura 5.7(a). Suponha que o digrafo D possua um conjunto de execuções que leva ao estado S_i , que habilita o termo E_1 . A única regra de transição válida, nesse caso, é $S_i \xrightarrow{E_1} S_{i+1}$. Em S_{i+1} , observamos a mesma situação, na qual somente a transição

$S_{i+1} \xrightarrow{E_2} S_{i+2}$ é possível. Pelas regras de transição da Tabela 3.1, $S_i \xrightarrow{E_1} S_{i+1} \xrightarrow{E_2} S_{i+2}$ é bissimilar a $S_i \xrightarrow{E_1 \cdot E_2} S_{i+2}$;

- redução alternativa: ilustrada pela Figura 5.7(b). Suponha que o digrafo D possua um conjunto de execuções que leva ao estado S_i , que pode habilitar, ou o termo E_1 , ou o termo E_2 . Dessa maneira, os únicos rastros possíveis são obtidos pelas regras de transição $S_i \xrightarrow{E_1} S_{i+1}$ e $S_i \xrightarrow{E_2} S_{i+1}$. Dessa maneira, se os termos E_1 e E_2 levam o estado S_i a S_{i+1} , então $S_i \xrightarrow{E_1 + E_2} S_{i+1}$;
- redução paralela: ilustrada pela Figura 5.7(c). Suponha que o digrafo D possua um conjunto de execuções que leva ao estado S_i , que habilita, ao mesmo tempo, os termos E_1 e E_2 , que devem ser executados para alcançar o estado S_{i+1} . As transições possíveis são, ou $S_i \xrightarrow{E_1} S_t \xrightarrow{E_2} S_{i+1}$, ou $S_i \xrightarrow{E_2} S_u \xrightarrow{E_1} S_{i+1}$, ou $S_i \xrightarrow{\gamma(E_1, E_2)} S_{i+1}$, gerando a expressão $E = E_1 \parallel E_2 + E_2 \parallel E_1 + E_1 | E_2$. Pelos axiomas da Tabela 3.8, E é bissimilar a $E_1 \parallel E_2$. Os estados S_t e S_u são estados intermediários usados para representar as transições dos operadores de intercalação.

Sendo assim, atribui-se à função θ as seguintes propriedades sobre um digrafo de expressões:

- $\theta(a) + \theta(b) = \theta(a + b)$;
- $\theta(a) \parallel \theta(b) = \theta(a \parallel b)$;
- $\theta(a) \cdot \theta(b) = \theta(a \cdot b)$.

Visto que a aplicação de uma redução sobre um digrafo de expressão conserva a bissimilaridade, vamos agora mostrar que qualquer seqüência de reduções leva ao mesmo digrafo. Esse fato pode ser provado por meio da análise de confluência, uma vez que a existência de uma forma normal é uma consequência direta da confluência do sistema de redução. Para isso, vamos considerar as produções seqüencial, mostrada na Figura 5.8, alternativa, mostrada na Figura 5.9 e paralela, mostrada na Figura 5.10.

Mostraremos que, para as produções do sistema, todos os pares críticos são combináveis. Os lemas a seguir mostram que o único par crítico existente neste sistema de redução ocorre quando um vértice do digrafo é compartilhado por duas reduções paralelas. Todos os demais pares de reduções são paralelamente independentes entre si.

Os lemas a seguir mostram a independência paralela para as produções paralela, alternativa e seqüencial. Para provar que duas reduções $D \xRightarrow{p_1, m_1} H_1$ e $D \xRightarrow{p_2, m_2} H_2$, com $p_i = L_i \leftarrow K_i \rightarrow R_i$, para $i = 1, 2$, são paralelamente independentes, basta mostrar que $m_1(L_1) \cap m_2(L_2) \subseteq m_1(l_1(K_1)) \cap m_2(l_2(K_2))$.

Lema 5.3.2. *Reduções paralelas são paralelamente independentes entre si. O mesmo vale para reduções alternativas.*

Demonstração. Seja D um digrafo, $p_3 : L_3 \xleftarrow{l_3} K_3 \xrightarrow{r_3} R_3$ uma produção paralela e duas derivações $D \xRightarrow{p_3, m'_3} H_1$ e $D \xRightarrow{p_3, m''_3} H_2$. Mostraremos que, para quaisquer morfismos m'_3 e m''_3 , $m'_3(L_3) \cap m''_3(L_3) \subseteq m'_3(l_3(K_3)) \cap m''_3(l_3(K_3))$.

Há dois casos a considerar. O primeiro se refere à situação em que $m'_3(L_3)$ e $m''_3(L_3)$ não possuem elementos em comum, e o segundo se refere à situação em que ambos possuem pelo menos um elemento em comum.

- Quando $m'_3(L_3)$ e $m''_3(L_3)$ não possuem elementos em comum, então $m'_3(L_3) \cap m''_3(L_3) = \emptyset$ e as derivações $D \xRightarrow{p_3, m'_3} H_1$ e $D \xRightarrow{p_3, m'_3} H_2$ são paralelamente independentes;
- Só existe uma situação em que $m'_3(L_3)$ e $m''_3(L_3)$ possuem algum elemento em comum. Isso ocorre quando $m'_3(L_3) = m''_3(L_3)$. Neste caso, $D \xRightarrow{p_3, m'_3} H_1$ e $D \xRightarrow{p_3, m'_3} H_2$ também são paralelamente independentes, pois

$$m'_3(L_3) \cap m''_3(L_3) = m'_3(L_3) \subseteq m'_3(L_3) = m''_3(l_3(K_3)) = m'_3(l_3(K_3)) \cap m''_3(l_3(K_3))$$

A prova para produções paralelas é equivalente. ■

Figura 5.8: Produção sequencial

Figura 5.9: Produção alternativa

Lema 5.3.3. *Uma redução paralela é paralelamente independente a uma redução alternativa.*

Demonstração. Seja D um digrafo e sejam as produções $p_2 : L_2 \xleftarrow{l_2} K_2 \xrightarrow{r_2} R_2$ alternativa e $p_3 : L_3 \xleftarrow{l_3} K_3 \xrightarrow{r_3} R_3$ paralela.

Figura 5.10: Produção paralela

Sejam quais forem os morfismos m_2 e m_3 , tal que $D \xrightarrow{p_2, m_2} D_2$ e $D \xrightarrow{p_1, m_3} D_3$, $m_2(L_2)$ e $m_3(L_3)$ não podem ter elementos em comum, pois os rótulos de seus vértices não permitem morfismos de L_2 para L_3 , de modo que $m_2(L_2) \cap m_3(L_3) = \emptyset$ e, conseqüentemente, quaisquer derivações de p_2 e p_3 são paralelamente independentes. ■

Lema 5.3.4. *Uma redução seqüencial é paralelamente independente a reduções paralelas e reduções alternativas.*

Figura 5.11: Possíveis mapeamentos das produções paralela e seqüencial.

Demonstração. Vamos provar que a redução seqüencial é paralelamente independente à redução paralela. A prova da independência paralela entre as reduções seqüencial e alternativa pode ser construída de forma equivalente.

Seja D um digrafo e sejam as produções $p_1 : L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1$ seqüencial e $p_3 : L_3 \xleftarrow{l_3} K_3 \xrightarrow{r_3} R_3$ paralela. Temos que mostrar que, para quaisquer morfismos m_1 e m_3 , tal que $D \xrightarrow{p_1, m_1} D_1$ e $D \xrightarrow{p_3, m_3} D_3$, $m_1(L_1) \cap m_3(L_3) \subseteq m_1(l_1(K_1)) \cap m_3(l_3(K_3))$.

Quando $m_1(L_1)$ e $m_3(L_3)$ não tem elementos em comum, p_1 e p_3 são paralelamente independentes, pois $m_1(L_1) \cap m_3(L_3) = \emptyset$.

Os únicos casos em que $m_1(L_1) \cap m_3(L_3) \neq \emptyset$ são mostrados na Figura 5.11, em que temos duas situações de mapeamento para m_1 e m_3 . Em (a), temos o vértice v_1 comum a $m_1(L_1)$ e $m_3(L_3)$ e, em (b), temos o vértice w_2 comum a $m_1(L_1)$ e $m_3(L_3)$. Tanto em (a), quanto em (b), o vértice comum a $m_1(L_1)$ e $m_3(L_3)$ não é removido, de modo que ele faz parte também de $m_1(l_1(K_1))$ e $m_3(l_3(K_3))$.

Sendo assim, p_1 e p_3 são paralelamente independentes. ■

Lema 5.3.5. *Todos os pares críticos do sistema de redução são combináveis.*

Figura 5.12: Par crítico formado por produções sequenciais

Demonstração. Pelos Lemas 5.3.2, 5.3.3 e 5.3.4, temos que a única combinação de derivações que gera pares críticos ocorre quando há duas produções sequenciais.

A Figura 5.12 ilustra a situação em que isso ocorre. No caso, o casamento m_1 associa L_1 aos vértices u , v e w . O casamento m_2 , por outro lado associa L_1 aos vértices v , w e x . As derivações originam o par crítico $\langle E_1, E_2 \rangle$. Os rótulos o_1 e o_2 representam operadores de composição quaisquer (sequencial, paralelo e alternativo). A Figura mostra que a aplicação da derivação $D \xrightarrow{p_1, m_1} E_1$ elimina o vértice v , necessário para a aplicação da derivação $D \xrightarrow{p_1, m_2} E_2$.

Pela definição 4.3.3, a prova para um par crítico combinável é um digrafo F , tal que $E_1 \Rightarrow^* F^* \Leftarrow$

E_2 . A Figura 5.12 mostra que é possível aplicar novamente uma redução sequencial em E_1 e E_2 , resultando, respectivamente, nos digrafos F_1 e F_2 , que são isomórficos.

Dessa forma, temos que $E_1 \xrightarrow{p_1, m_3} F_1 \cong F_2 \xleftarrow{p_1, m_4} E_2$, ou seja, o par crítico $\langle E_1, E_2 \rangle$ é combinável. ■

5.3.1 Confluência e Terminação

Lema 5.3.6. *O sistema de redução em digrafos de expressão é localmente confluyente.*

Demonstração. Os Lema 5.3.5 garante que todos os pares críticos são combináveis. Pelo Lema 4.3.3, o sistema de reduções proposto neste capítulo é localmente confluyente.

Lema 5.3.7. *O sistema de redução em digrafos de expressão é finito.*

Demonstração. Suponha que $\mathcal{G}$ admita uma seqüência infinita de derivações diretas $G_1 \Rightarrow G_2 \Rightarrow \dots$. Então, existe pelo menos uma derivação direta que aumente, ou conserve o número de arcos e vértices, de modo que, para alguma derivação $G_i \Rightarrow G_{i+1}$, $i > 0$, $|V(G_i)| \leq |V(G_{i+1})|$ e $|A(G_i)| \leq |A(G_{i+1})|$. Isso é uma contradição, pois todas as regras de redução de $\mathcal{G}$ diminuem o número de arcos e/ou vértices. ■

Teorema 5.3.8. *O sistema de redução em digrafos de expressão é confluyente.*

Demonstração. Pelo Teorema de Newman [79], todo sistema finalizável localmente confluyente também é confluyente. Pelos Lemas 5.3.7 e 5.3.6, temos que o sistema de redução proposto neste capítulo é finalizável e localmente confluyente, então ele também é confluyente. ■

A análise das propriedades do sistema de redução que foi proposto neste capítulo garante que o sistema é correto, conservando a bissimilaridade de termos durante as transformações e, além disso, é finalizável e confluyente. Sendo assim, seqüências infinitas de reduções não são possíveis e, além disso, qualquer seqüência de reduções aplicada sobre o mesmo digrafo leva ao mesmo digrafo reduzido.

5.4 Conflitos Estruturais em SP-Digrafos de Processos

Como já foi antecipado, os rótulos dos vértices possuem um fator semântico que permite analisar pontos de ramificação e sincronização do digrafo de expressão. Estruturas de ramificação e sincronização são muito comuns em modelos de negócios, motivo pelo qual são implementadas pela maioria das ferramentas de modelagem e estão previstos nos padrões de controle de fluxo de *workflow* [40]. Muitos trabalhos, entretanto, exploram o uso de ramificações e sincronizações de forma independente,

Figura 5.13: Conflitos estruturais em digrafos de expressão. Em (a), a ocorrência de um *deadlock* e, em (b), uma falha de sincronização.

levando a crer que podemos combiná-las livremente sem gerar problemas para a estrutura do processo de negócio. Infelizmente, essa dedução não é verdadeira, pois determinadas combinações podem incorporar inconsistências ou falhas no modelo do processo de negócio.

A inclusão de falhas é resultado da falta de coerência semântica entre vértices de ramificação e sincronização. A coerência semântica significa que vértices de ramificação e sincronização correspondentes devem possuir o mesmo rótulo. Sendo assim, ramificações que têm origem em vértices com rótulos definidos pela composição alternativa (+) devem ser sincronizados por vértices que também são definidos pela composição alternativa (+). O mesmo vale para vértices de ramificação e sincronização definidos por regras de composição paralela. Digrafos com vértices que não possuem coerência semântica apresentam problemas em sua lógica de execução, caracterizando a presença de conflitos estruturais.

Conflito estrutural é o termo adotado por alguns autores para representar relações de ramificação e sincronização que apresentam algum problema de correção ou consistência [101]. A maioria dos modelos de representação de processos de negócio baseados em grafos permitem construções com conflitos, que podem ser classificados em:

1. *deadlock*: caracterizado pela tentativa de sincronização de dois (ou mais) ramos de execução que não são executados juntos na mesma instância;
2. *falha de sincronização*: a execução de dois (ou mais) ramos de execução paralelas de execução que não são sincronizados, mesmo que passem por vértices de sincronização.

A Figura 5.13 (a) ilustra um processo de negócio com *deadlock* e a Figura 5.13 (b), uma falha de sincronização. O *deadlock* se caracteriza pela espera de um recurso que nunca ficará disponível. Na Figura 5.13(a), o termo $\theta(d)$ espera pela finalização dos termos $\theta(b)$ e $\theta(c)$, todavia somente um deles será executado após $\theta(a)$. Por esse motivo, o termo $\theta(d)$ ficará indefinidamente em estado de espera, interrompendo a execução de todos os termos posteriores.

Uma *falha de sincronização* ocorre quando duas (ou mais) ramificações paralelas conservam suas linhas execução após um vértice de sincronização. Na Figura 5.13(b), podemos ver essa situação. A

Figura 5.14: Ausência de sincronização em Processo de Negócio

execução isolada de $\theta(b)$ ou $\theta(c)$ é suficiente para habilitar $\theta(d)$. O vértice de sincronização com operador “+” possivelmente prevê que somente um deles foi habilitado para execução, no entanto o vértice de ramificação “||” habilita ambos os termos. O resultado é que $\theta(d)$ é habilitado e executado duas vezes.

Falhas de sincronização e *deadlocks* têm uma relação próxima, como mostra a Figura 5.14. Nela, é possível acompanhar um exemplo de digrafo de expressão com falha de sincronização. Note que, após a execução do termo $\theta(a)$, ocorre uma ramificação de duas linhas de execução, que são posteriormente sincronizadas antes de $\theta(g)$. Um dos ramos de execução possui falha de sincronização. O vértice **2** sincroniza os termos $\theta(c)$ e $\theta(f)$, vindo do vértice **4**. Sendo assim, $\theta(f)$ será habilitado duas vezes, sendo uma pela execução de $\theta(d)$ e outra pela execução de $\theta(e)$.

O problema ocorre na sincronização do vértice **2**. A interpretação mais simples e imediata sugere que o vértice **2** sincronize as execuções de $\theta(c)$ e da execução de $\theta(f)$ habilitada por $\theta(d)$ ou $\theta(e)$. O termo que não foi aproveitado nessa sincronização fica em modo de espera, aguardando uma nova execução de $\theta(c)$ para seguir em frente, fato que não acontecerá mais na mesma instância.

Pelo exemplo da Figura 5.14, podemos construir uma idéia do quão complexo pode ser avaliar um processo de negócio. A propagação de uma falha de sincronização pode gerar novos conflitos em pontos posteriores do modelo de negócio. Além disso, à medida que aumentam as combinações de estruturas de ramificação e sincronização, também aumenta a complexidade e a dificuldade em controlar a execução de cada instância do processo de negócio.

Apesar dos problemas que podem causar no modelo do processo de negócio, não é raro encontrar conflitos estruturais em aplicações reais. Falhas de sincronização podem ser resultado de problemas na modelagem ou efeitos colaterais de modificações históricas. Normalmente, sua presença não é deliberadamente planejada, uma vez que há formas alternativas de representar ramos de execução assíncronos, evitando, por exemplo, ambigüidades de interpretação e dificuldades no acompanhamento da execução.

Para facilitar este estudo, este trabalho define uma classe de sp-digrafos de expressão que chamaremos **bem-formados**, na qual agruparemos os sp-digrafos de expressão com ausência de conflitos estruturais.

Definição 5.4.29: (*SP-Digrafos Bem-Formados*) Um sp-digrafo de processos bem formado é um sp-digrafo de processos sem conflitos estruturais.

Os sp-digrafos bem-formados representam a classe de grafos para os quais a geração de uma expressão algébrica equivalente é possível e eficiente. Eles servirão como ponto de partida para a elaboração do algoritmo de geração de expressões. Posteriormente, o algoritmo será estendido para classes mais amplas, com conflitos e até digrafos que não são série-paralelo.

5.5 Conclusões

Este capítulo introduziu o conceito de digrafos de expressão. Digrafos de expressão são digrafos acíclicos usados para representar processos de negócio por meio de rótulos algébricos em seus vértices e arcos. Essa rotulação serve para incorporar propriedades algébricas na estrutura do digrafo, de modo que seja possível comparar a especificação do processo de negócio com as propriedades da álgebra de processos.

Este capítulo também apresentou um sistema de redução que conserva a bissimilaridade em relação à interpretação algébrica de vértices e arcos. O grande objetivo do sistema de redução é reduzir topologicamente o digrafo de expressão conservando as relações algébricas de seus arcos e vértices. Outras propriedades importantes provadas foram a confluência e a terminação. A terminação garante que qualquer digrafo pode ser reduzido a uma estrutura minimal com um número finito de reduções e a confluência garante que todas as reduções levam à mesma estrutura minimal.

Uma característica importante que deve ser considerada na análise de um digrafo de expressão é a presença de conflitos estruturais. Os conflitos indicam problemas na consistência do processo de negócio, o que pode acarretar conseqüências indesejáveis, como a paralisação da execução (estado de *deadlock*) ou a execução não planejada de atividades (falhas de sincronização).

Todos esses conceitos fundamentam a base formal para elaboração de um algoritmo de geração de expressões algébricas a partir de digrafos de expressão. O próximo capítulo apresenta a aplicação do sistema de redução para a obtenção desse objetivo.

Capítulo 6

Geração de Expressões Algébricas

6.1 Introdução

O capítulo anterior introduziu o conceito de digrafos de expressão, juntamente com um sistema de redução. Este capítulo explora a aplicação desse sistema de redução em digrafos de expressão, a fim de gerar expressões algébricas por meio de sucessivas reduções sobre digrafos de processos que representam processos de negócio. Aproveitando as propriedades de reduções topológicas de sp-digrafos e adaptando-as a produções apropriadas de transformação de grafos, é possível construir um algoritmo eficiente de geração de expressões algébricas. Além disso, essa abordagem fornece ferramentas formais capazes de mostrar que as expressões geradas são únicas, corretas e bissimilares ao sp-digrafo de expressão original.

Outro assunto abordado neste capítulo também é a extensão do algoritmo de geração de expressões algébricas a digrafos que não são série-paralelo. Essa extensão aumenta também a abrangência de aplicação do sistema de redução, permitindo cobrir parte dos casos em que o processo de negócio não é representado por um sp-digrafo.

A Seção 6.2 apresenta o algoritmo para geração de expressões algébricas a partir de sp-digrafos de expressão bem-formados. A Seção 6.3 apresenta uma série de modificações topológicas que podem ser aplicadas a pontes de *Wheatstone* isoláveis, afim de transformá-las em sp-digrafos. Infelizmente, algumas transformações topológicas provocam perda de níveis de equivalência em relação aos digrafos originais, como discutido na Seção 6.4. Finalmente, as conclusões são mostradas na Seção 6.5.

6.2 Expressões em SP-Digrafos Bem-Formados

Esta seção descreve o algoritmo de geração de expressões algébricas a partir de sp-digrafos de expressão bem-formados. O algoritmo aplica as transformações definidas no sistema de redução proposto no capítulo anterior. Antes, porém, é necessário fazer o tratamento apropriado da entrada do algoritmo, uma vez que, por hipótese, o usuário não fornece um digrafo, mas um conjunto de dependências diretas entre atividades. Essa informação é utilizada para construir um digrafo de dependências e, em seguida, um digrafo de expressão.

Após a construção da entrada, o algoritmo de geração de expressões é apresentado e discutido.

Posteriormente, esse algoritmo é modificado para trabalhar com digrafo de expressão genéricos. Para sp-digrafos de expressão bem-formados, o algoritmo modificado continua apresentando a expressão algébrica resultante. Para sp-digrafos que não são bem-formados, o algoritmo identifica pontos de conflito e, finalmente, para digrafos que não são série-paralelo, o algoritmo devolve o digrafo reduzido com uma ponte de *Wheatstone*.

6.2.1 Tratamento da Entrada de Dados

Como apresentado no Capítulo 1, a entrada de dados do problema abordado neste trabalho é uma descrição de dependências entre fases de um processo de negócio. Quando as fases do processo de negócio possuem um caráter atômico, com entrada e saída bem definidos, é usual chamá-los de *atividades*. Dizer que uma atividade B depende de uma atividade A , representado por $A \rightarrow B$, significa que a atividade B só pode iniciar sua execução após a finalização da atividade A . Isso ocorre principalmente quando a atividade B utiliza, como entrada, alguma informação dada como saída da atividade A .

Para representar as relações de dependência entre atividades, será usada uma tabela com três colunas, em que a atividade da coluna da esquerda é predecessora direta da atividade da coluna central. A coluna da direita apresenta uma ilustração gráfica da dependência entre os elementos das duas outras colunas. A Tabela 6.1 ilustra essa estrutura, que será usada como ponto de partida da geração do digrafo de entrada.

A definição da tabela de dependências, entretanto, não é suficiente para caracterizar o comportamento completo do processo de negócio. Isso porque a representação de uma dependência em pontos de ramificação e sincronização não é capaz de distinguir as regras de execução paralela e alternativa. Sendo assim, faz-se necessário um detalhamento dessas regras, que será dividido em três fases:

- (a) definição das dependências, em que as dependências do processo de negócio são catalogadas individualmente;
- (b) refinamento das dependências, em que são definidas as regras de comportamento entre as atividades que marcam pontos de ramificação e sincronização dos processos;
- (c) geração do digrafo de entrada, em que as regras obtidas de (a) e (b) são unificadas e mapeadas para um digrafo de expressão.

A realização de cada fase pode ser acompanhada por meio de um exemplo, no qual é possível acompanhar detalhadamente cada fase de geração da entrada de dados.

Primeiramente, a definição do processo de negócio deve ser acompanhada por um especialista de negócio, profissional com amplo conhecimento do fluxo de transformação de informação que se deseja modelar. O papel do especialista é fundamental na definição e refinamento das dependências, pois dele será a responsabilidade de especificar a relação individual de cada atividade em relação às demais.

6.2.2 Definição de Dependências

Suponha que a entrada para um modelo de processo de negócio é composta pelo conjunto de dependências diretas da Tabela 6.1. As atividades **A**, **B**, **C**, **D**, **E**, **F**, **G**, **H**, **I**, **J**, **K** e **L** representam

Tabela 6.1: Conjunto de dependências entre atividades

Atividade	Sucessor	Representação Gráfica
A	B	Ⓐ → Ⓑ
B	C	Ⓑ → Ⓒ
B	D	Ⓑ → Ⓓ
C	E	Ⓒ → Ⓔ
C	F	Ⓒ → Ⓕ
D	H	Ⓓ → Ⓗ
D	I	Ⓓ → Ⓘ
E	G	Ⓔ → Ⓖ
F	G	Ⓕ → Ⓖ
G	K	Ⓖ → Ⓚ
H	J	Ⓗ → Ⓙ
I	J	Ⓘ → Ⓙ
J	K	Ⓙ → Ⓚ
K	L	Ⓚ → Ⓛ

partes bem definidas do processo de negócio, que produzem alguma transformação no dado de entrada e disponibilizam uma saída que pode ser utilizada por outra atividade. A coluna da esquerda apresenta cada atividade do processo de negócio; a coluna central apresenta um de seus sucessores diretos, que também é uma atividade; e a coluna da direita apresenta um esboço da representação gráfica da dependência entre as colunas da esquerda e do centro.

Figura 6.1: Grafo de dependências gerado a partir da Tabela 6.1

A junção das dependências da Tabela 6.1 origina o grafo de dependências mostrado na Figura 6.1. Esse grafo de dependências deve ser acíclico, ilustrando claramente a ordem de execução entre as atividades que compõem o processo de negócio, desde a sua atividade inicial **A** até sua atividade final **L**. Um outro fato importante é que algoritmos clássicos de Teoria dos Grafos podem ser utilizados para verificar características topológicas, como a ausência de ciclos e unicidade dos vértices fonte e sorvedouro. Caso o grafo não possua alguma delas, a tabela de dependências deve ser remodelada.

Como já foi antecipado, o grafo de dependências não é capaz de representar completamente o comportamento global do processo de negócio. O modelo do processo de negócio necessita, além da ordem de execução entre as atividades (inferidas por meio das dependências), do comportamento adotado pelo fluxo quando o processo se ramifica, ou sincroniza. Por esse motivo, é necessário revisar a tabela de dependências, classificando todos os pontos de ramificação e sincronização.

6.2.3 Refinamento de Dependências

O refinamento das dependências consiste exatamente nessa classificação. Os pontos de ramificação estão em destaque na Tabela 6.2. Eles consistem em todas as atividades da Tabela 6.1 que possuem mais de uma entrada na coluna da esquerda. No exemplo, os pontos de ramificação são as atividades **B**, **C** e **D**.

Da mesma maneira, os pontos de sincronização são descobertos a partir da observação da coluna central da Tabela 6.1, cujo resultado é mostrado na Tabela 6.3. Neste exemplo, as atividades **G**, **J** e **K** são identificadas por possuírem mais de uma entrada na coluna central.

Tanto ramificações quanto sincronizações são classificadas em duas categorias: alternativas e paralelas. Para ilustrar tais comportamentos, semelhantes às definições vistas na Seção 5.2, considere o subgrafo formado pelos vértices **C**, **E**, **F** e **G**, da Figura 6.1. Se **C** marcar um ponto de ramificação alternativa, somente uma das atividades sucessoras (**E** ou **F**) executará após **C**. Se **C** marcar um ponto de ramificação paralela, tanto **E** quanto **F** executarão. Nesse mesmo exemplo, a atividade **G** marca um ponto de sincronização. Caso **G** seja um ponto de sincronização alternativa, a execução (finalizada) de **E** ou **F** é suficiente para habilitar **G**. Se, entretanto, **G** for um ponto de sincronização paralela, sua habilitação necessitará da execução tanto de **E** quanto de **F**.

É oportuno ressaltar que a classificação do comportamento dos pontos de ramificação e sincronização não é resultado de um processamento algorítmico, mas uma parte fundamental da interação do especialista de negócio na construção da entrada de dados. Como já foi comentado, a entrada de dados consiste na definição das dependências diretas entre as atividades e na classificação dos pontos de ramificação e sincronização. Para definir a classificação de ramificações, o especialista adiciona vértices auxiliares após cada ponto de ramificação. Neste exemplo, os pontos de ramificação são representados pelas atividades **B**, **C** e **D**. O vértice auxiliar será colocado entre o vértice de ramificação e seus sucessores, de modo que as dependências sejam conservadas. O rótulo do vértice auxiliar determina, explicitamente, o comportamento da ramificação. O rótulo “+” indica uma ramificação alternativa, enquanto o rótulo “||” indica uma ramificação paralela. Podemos acompanhar este exemplo na representação gráfica contida na Tabela 6.2.

Do mesmo modo, os pontos de sincronização são também classificados em alternativos e paralelos.

Tabela 6.2: Conjunto de dependências de ramificação entre atividades

Ação	Sucessor	Representação Gráfica
B	C	
B	D	
C	E	
C	F	
D	H	
D	I	

Neste exemplo, os pontos de sincronização **G**, **J** e **K** têm sua classificação mostrada na Tabela 6.3. Incorporando a classificação dos pontos de ramificação e sincronização no digrafo de dependências, obtém-se o digrafo mostrado na Figura 6.2.

Tabela 6.3: Conjunto de dependências de sincronização entre atividades

Ação	Sucessor	Representação Gráfica
E	G	
F	G	
H	J	
I	J	
G	K	
J	K	

Quando os pontos de ramificação (sincronização) representam uma atividade que precede (sucede) mais de duas atividades, a classificação pode ser mais complexa, exigindo a utilização de vários vértices auxiliares. Para ter uma idéia das possibilidades de classificação nessas situações, acompanhe a Figura 6.3, que mostra os possíveis resultados de uma classificação de três dependências envolvendo atividades fictícias x , y , z e w . A Figura 6.3(a) exibe as dependências $x \rightarrow y$, $x \rightarrow z$ e $x \rightarrow w$. Na Figura 6.3(b), as três dependências são compostas alternativamente, de modo que a execução de x habilita apenas uma das três atividades seguintes (y , z ou w). A Figura 6.3(c), por outro lado, ilustra uma composição paralela em que a execução de x é seguida pela execução de y , z e w . As Figuras 6.3(d), 6.3(e) e 6.3(f) mostram uma forma de composição em que duas atividades serão executadas após x . Finalmente, as Figura 6.3(g), 6.3(h) e 6.3(i) exibem composições onde uma, ou duas, atividades são executadas após x .

É do especialista de negócio o papel de definir claramente o comportamento do processo de negócio

Figura 6.2: Grafo de dependências com regras de ramificação e sincronização

Figura 6.3: Refinamento de dependências múltiplas em pontos de ramificação.

nessas situações. A definição das regras de ramificação determina o comportamento do processo de negócio, de modo que seja possível prever todos os rastros possíveis do processo mesmo antes da execução de uma de suas instâncias. Infelizmente, essa definição não pode ser inferida de forma automática, pois cada combinação de regras provoca alterações consideráveis no comportamento do processo de negócio.

6.2.4 Geração do Digrafo de Expressão

Retomando o exemplo, a estrutura montada na fase anterior 6.2 precisa, agora, ser transformada em um digrafo de expressão, que será usado como entrada para os algoritmos que serão apresentados posteriormente neste capítulo. A geração do digrafo de expressão produz um digrafo acíclico com arcos e vértices rotulados. Os arcos são mapeados em atividades do processo de negócio e os vértices são mapeados em operadores algébricos de composição. A seguir, apresentamos as três etapas envolvidas

nessa operação:

1. geração do digrafo de linha;
2. atualização de rótulos dos arcos;
3. atualização de rótulos dos vértices;

A seqüência dessas etapas pode ser visualizada na Figura 6.4. Na Figura 6.4(a), temos o digrafo de dependências gerado na fase anterior. Um digrafo de linha é construído a partir desse digrafo, chegando ao resultado mostrado na Figura 6.4(b). Relembrando a definição do Capítulo 2, a construção de um digrafo de linha D_L , a partir do digrafo inicial D , consiste na criação de um novo digrafo, tal que $V(D_L) = A(D)$ e $A(D_L) = \{ab : a, b \in V(D_L), \text{ e a cabeça de } a \text{ é igual à cauda de } b\}$.

No digrafo de linha, cada arco de D é mapeado para um vértice de D_L . Dessa forma, conseguimos reconstruir as relações do digrafo inicial mapeando as dependências, representadas por arcos em D , em vértices de D_L . As dependências regulam as regras de composição de D , de modo que cada vértice de D_L conterá uma regra de composição que associa duas atividades de D .

Os arcos de D_L representam os elementos inicial e final de cada dependência de D . Nesse caso, podemos classificá-los em duas categorias:

- atividades do processo de negócio;
- regras de ramificação e sincronização.

Cada vértice de D possui pelo menos um arco correspondente em D_L . Idealmente, o digrafo de expressão deve mapear seus arcos em atividades do processo de negócio. Entretanto, na fase de refinamento, vértices artificiais, que determinam as regras de ramificação e sincronização, foram adicionados a D e, conseqüentemente, também possuem correspondência em D_L . O objetivo desses vértices artificiais não é representar atividades físicas que compõem o processo de negócio, mas somente caracterizar um comportamento de execução, que também pode ser observado nas regras de composição dos vértices de D_L . Esses vértices terão um tratamento diferenciado em relação aos vértices de D que representam atividades do processo de negócio.

Para finalizar, é necessário atribuir os rótulos de vértices (π) e arcos (θ) de D_L . Para cada vértice uv de D_L (que representa o arco uv em D), $\pi(uv)$ é definido como:

$$\pi(uv) = \begin{cases} \cdot, & \text{se } d^+(uv) = d^-(uv) = 1 \\ v, & \text{se } d^+(uv) > 1 \\ u, & \text{se } d^-(uv) > 1 \end{cases}$$

É interessante comentar que o rótulo $\pi(uv)$ de cada vértice sempre será um operador de composição da álgebra de processos. Já para os arcos, a rotulação θ de cada arco $((uv)(vw))$ é processada de acordo com a seguinte relação:

$$\theta((uv)(vw)) = \begin{cases} v, & \text{se } v \text{ é atividade} \\ \lambda, & \text{caso contrário} \end{cases}$$

A regra atribui, para cada arco $((uv)(vw))$, o rótulo v , como mostrado na Figura 6.4(c). Rótulos que não representam atividades do processo são mapeados para λ , uma vez que são resultado da adição de vértices artificiais durante o refinamento. As regras de ramificação e sincronização, entretanto, não são perdidas, pois ficam representadas nos vértices, cujo mapeamento representa as regras de composição seqüencial, paralela ou alternativa. Podemos ver o digrafo de expressão final na Figura 6.4(d).

6.2.5 Expressões em SP-Digrafos Bem-Formados

Uma vez gerado o grafo de expressão, pode-se usar o algoritmo de geração de expressões algébricas. Assumiremos, por hipótese, que o digrafo dado como entrada é um sp-digrafo de expressão bem-formado. Posteriormente, discutiremos a utilização do algoritmo para sp-digrafos que não são bem-formados. Para facilitar o entendimento do algoritmo, vamos apresentá-lo informalmente por meio de um exemplo e, em seguida, discutiremos sua eficiência e corretude. Considere o processo de negócio da Figura 6.5 (a), representado pelo sp-digrafo de expressão obtido na fase anterior.

O algoritmo se resume na aplicação contínua de reduções seqüenciais, paralelas e alternativas sobre o digrafo de expressão. Na Figura 6.5(b), podemos ver o resultado da aplicação de reduções seqüenciais sobre o sp-digrafo, seguidas por reduções paralelas e alternativas em 6.5(c). Todas as reduções substituem um conjunto de arcos por um único arco, que seja capaz de representar bissimilarmente o conjunto de arcos anterior por meio de seu rótulo θ . Conforme foi apresentado no Capítulo 5, reduções seqüenciais rotulam os arcos com termos formados a partir de composições seqüenciais ($\cdot$). Enquanto isso, reduções paralelas e alternativas rotulam arcos com termos formados a partir de composições paralelas ($\parallel$) e alternativas ($+$).

O resultado da aplicação contínua de reduções seqüenciais e paralelas sobre o sp-digrafo de expressão é o sp-digrafo trivial. Nas Figuras 6.5(d) e 6.5(e), temos passos intermediários da seqüência de reduções, e, finalmente, em 6.5(f), o sp-digrafo trivial, cujo rótulo do arco possui a expressão algébrica em ACP bissimilar ao sp-digrafo de expressão dado como entrada.

O código para geração de expressões a partir de sp-digrafos de expressão bem-formados é mostrado no Algoritmo 6.1. Antes de iniciar a explicação do algoritmo propriamente dito, vamos detalhar cada uma das funções usadas em seu código:

- $pred(x)$: devolve o conjunto de vértices antecessores do vértice x . Se x for fonte, devolve $\emptyset$;
- $suc(x)$: devolve o conjunto de vértices sucessores do vértice x . Se x for sorvedouro, devolve $\emptyset$;
- $\theta(xy)$: devolve o rótulo do arco xy ;
- $\parallel xy \parallel$: devolve a quantidade de arcos entre os vértices x e y , em $A(D)$;

A prova de corretude do algoritmos segue os seguintes lemas:

Lema 6.2.1. *No início de cada iteração, D é um sp-digrafo.*

Figura 6.4: Geração do digrafo de expressão: (a) digrafo com regras de ramificação e sincronização; (b) digrafo de linha obtido a partir de (a); (c) atualização dos rótulos de arcos do digrafo em (b); (d) digrafo de expressão.

Figura 6.5: Geração de expressões algébricas a partir de reduções em sp-digraphos

Algoritmo 6.1 Algoritmo de geração de expressões algébricas a partir de sp-digrafos bem-formados

Entrada: Um sp-digrafo de expressão bem-formado D

```

1: enquanto  $D$  é redutível faça
2:   para todo  $v \in V(D)$  não-terminal faça
3:     se  $d^-(v) = d^+(v) = 1$  então
4:        $u \leftarrow \text{pred}(v)$ ;
5:        $w \leftarrow \text{suc}(v)$ ;
6:        $A(D) \leftarrow A(D) \cup \{(u, w)\}$ 
7:        $\theta(uw) \leftarrow \theta(uv) \cdot \theta(vw)$ 
8:        $A(D) \leftarrow A(D) \setminus \{(u, v), (v, w)\}$ 
9:        $V(D) \leftarrow V(D) \setminus \{v\}$ 
10:    fim se
11:  fim para
12:
13:  para todo  $v \in V(D)$  faça
14:    para todo  $w \in V(D)$  e  $w \in \text{suc}(v)$  faça
15:      se  $\|vw\| > 1$  então
16:        {Sejam  $a_1, a_2, \dots, a_k$  os arcos  $v \rightarrow w$  em  $A(D)$ .}
17:         $\theta'(vw) \leftarrow \theta(a_1)$ 
18:         $A(D) \leftarrow A(D) \setminus \{a_1\}$ 
19:        para todo  $2 \leq i \leq k$  faça
20:           $\theta'(vw) \leftarrow \theta'(vw) \pi(v) \theta(a_i)$ 
21:           $A(D) \leftarrow A(D) \setminus \{a_i\}$ 
22:        fim para
23:         $A(D) \leftarrow A(D) \cup \{vw\}$ 
24:         $\theta(vw) \leftarrow \theta'(vw)$ 
25:         $\pi(v) \leftarrow \pi(w) \leftarrow \cdot$ 
26:      fim se
27:    fim para
28:  fim para
29: fim enquanto
30: devolve termo( $D$ )

```

Demonstração. O laço entre as linhas 2 e 11 realiza, sobre D , reduções seqüenciais, removendo vértices com grau de entrada e saída iguais a um, e criando um arco entre seus vértices adjacentes. O laço entre as linhas 13 e 28 aplica reduções paralelas sobre D . Pelo corolário 2.3.5, a aplicação de reduções seqüenciais e paralelas preserva a topologia de um sp-digrafo.

O Lema 6.2.1 mostra que as reduções aplicadas sobre o digrafo original não modificam suas propriedades topológicas em relação à classe de sp-digrafos, ou seja, digrafos série-paralelo no início de cada iteração continuam a apresentar topologia série-paralelo ao final de cada iteração. O complemento dessa observação também é verdadeiro, pois digrafos que não são série-paralelo continuam não sendo ao final de cada iteração (Proposição 2.3.1). Essa propriedade ajuda a provar o Lema 6.2.2.

Lema 6.2.2. *Para qualquer entrada válida, o algoritmo sempre finaliza.*

Demonstração. A cada iteração do laço entre as linhas 1 e 29, o sp-digrafo é submetido às reduções, sempre reduzindo o seu número de arcos. Pelo Lema 6.2.1, essa operação não altera a estrutura to-

pológica do sp-digrafo. Quando a aplicação de nenhuma redução é possível, D é minimal em relação às reduções, tornando falsa a condição de manutenção do laço. ■

Em relação às expressões geradas pelas reduções, temos:

Lema 6.2.3. *A cada iteração, a aplicação de reduções seqüenciais, paralelas e alternativas conserva a relação algébrica entre os termos (rótulos dos arcos) do digrafo.*

Demonstração. A prova desse lema é resultado direto da aplicação do Lema 5.3.1

A cada iteração do laço da linha 2, a redução seqüencial troca um par de arcos uv e vw por um arco uw , tal que $\theta(uw) = \theta(uv) \cdot \theta(vw)$.

Para cada iteração do laço da linha 13, a redução paralela reduz arcos paralelos $a_1, a_2, \dots, a_n$, entre vértices u e v , por um único arco uv , tal que $\theta(uv) = \theta(a_1)\pi(v)\theta(a_2)\pi(v) \dots \pi(v)\theta(a_n)$, onde $\pi(v)$ é o operador de composição associado ao vértice v ($\parallel$ ou $+$).

Pelos Lemas 6.2.2 e 6.2.3, o algoritmo realiza reduções seqüenciais e paralelas, sucessivamente, diminuindo gradativamente o número de arcos e mantendo as propriedades de seus termos algébricos. Além disso, o algoritmo tem garantia de finalização. Sendo assim, pode-se afirmar que o algoritmo é correto.

O consumo de tempo desse algoritmo é definido pelos laços das linhas 1, 2, 13, 14 e 19. O laço da linha 2 realiza uma busca sobre o conjunto de vértices, consumindo tempo assintótico da ordem de $O(n)$, onde n é o número de vértices. Os laços das linhas 13 e 14, que rastreiam o conjunto de vértices vizinhos têm, juntos, o consumo de tempo equivalente ao rastreamento do conjunto de arcos do digrafo, consumindo tempo $O(m)$, onde m é o número de arcos do digrafo. Esse consumo de tempo já considera o laço da linha 19, pois ele é responsável pelo tratamento de arcos múltiplos entre dois vértices. Desse modo, conclui-se que o bloco entre as linhas 13 e 28 consome tempo de ordem $O(m)$.

O laço da linha 1 se repete enquanto o digrafo for redutível, ou seja, enquanto for possível aplicar alguma das reduções. Como cada redução diminui o número de arcos em uma unidade, pode-se afirmar que esse laço se repete, no máximo, $O(m)$ vezes. Sendo assim, o consumo de tempo total do algoritmo é de $O(m^2 + mn)$.

6.2.6 Identificação de SP-Digrafos e Conflitos

O algoritmo apresentado nesta seção assume que a entrada é um sp-digrafo de expressão bem-formado. Uma tarefa importante, do ponto de vista de aplicação do algoritmo, é identificar se um digrafo pode ser utilizado como entrada para o algoritmo, ou não. Para isso, é necessário classificar o digrafo em relação a três características: presença de ciclos, presença de um subdigrafo com ponte de *Wheatstone* e presença de conflitos estruturais.

Todo sp-digrafo é acíclico com dois terminais, de modo que o primeiro passo é identificar se o digrafo dado possui tais características. Essa tarefa pode ser resolvida sem grandes problemas por um algoritmo de construção de ordenação acíclica, como o mostrado na Proposição 2.2.2. Esse algoritmo consome tempo linear em relação ao número de vértices [102].

Uma vez caracterizada a ausência de ciclos, o segundo passo é identificar se o digrafo acíclico é sp-digrafo ou não. Essa tarefa pode ser feita por meio de vários algoritmos eficientes baseados em reduções. A lógica desses algoritmos é semelhante à apresentada no algoritmo anterior, com otimizações em relação a estruturas de dados capazes de fazer busca mais eficiente em vértices e arcos.

Uma terceira verificação faz referência à presença de conflitos. Os conflitos podem ser descobertos em um sp-digrafo comparando a função π de seus vértices de ramificação com o correspondente vértice de sincronização.

A verificação da presença de conflitos e pontes de *Wheatstone* podem ser incorporadas ao Algoritmo 6.1, sem alterar decisivamente sua curva assintótica de consumo de tempo. Sendo assim, é possível modelar um algoritmo que recebe, como entrada, qualquer digrafo D e devolve a expressão algébrica equivalente a D , se D é sp-digrafo de expressão bem-formado. Caso contrário, o algoritmo devolve um subdigrafo de D com pontes de *Wheatstone* ou pontos de conflito estrutural.

A implementação das modificações sugeridas leva ao Algoritmo 6.2. A primeira modificação se refere à informação devolvida pelo algoritmo. Enquanto o Algoritmo 6.1 devolvia a expressão algébrica, o Algoritmo 6.2 devolve um digrafo. Essa mudança ocorre porque, em ambos os algoritmos, o resultado é obtido por meio da aplicação de reduções no digrafo de entrada, produzindo um digrafo reduzido. Como, no Algoritmo 6.1, assume-se que o digrafo de entrada é bem-formado, então o digrafo reduzido sempre será isomórfico ao sp-digrafo trivial e apresentará, como rótulo de seu único arco, a expressão algébrica correspondente ao digrafo de entrada D . No Algoritmo 6.2, não há a garantia de que o digrafo de entrada seja bem-formado, de modo que não há como garantir que a construção de uma expressão algébrica ao final da aplicação das reduções. Se D não for série-paralelo, então seu digrafo reduzido apresentará um ponte de *Wheatstone*, e a expressão algébrica não será produzida completamente.

Mesmo em digrafos série-paralelo, há o risco da presença de conflitos. Para esses casos, o Algoritmo 6.2 devolve um digrafo no qual todos os pares de vértices com arcos redundantes são locais de conflitos. Para isso, o algoritmo utiliza uma variável local *booleana*, *possui_conflito*, cujo objetivo é indicar a presença de conflitos estruturais no digrafo de em processo de redução. O teste de verificação de conflitos ocorre nas linhas 19-21, onde as reduções paralelas e alternativas consideram a correspondência entre os vértices de ramificação e sincronização.

O Algoritmo 6.2 também cobre os mesmos casos cobertos pelo Algoritmo 6.1. O Algoritmo 6.2 conserva a mesma estrutura do Algoritmo 6.1, aplicando sucessivas reduções sobre o digrafo de entrada D . Caso D seja um sp-digrafo bem-formado, o resultado final do Algoritmo 6.2 é um digrafo isomórfico ao sp-digrafo trivial, que apresenta, no rótulo de seu único arco, a expressão algébrica equivalente ao digrafo original.

Para casos em que o digrafo não é série-paralelo, o Algoritmo 6.2 devolve um digrafo reduzido, em que se pode identificar uma ou mais pontes de *Wheatstone*.

6.3 Expressões em Pontes de Wheatstone Isoláveis

Embora consideradas indesejáveis, pontes de *Wheatstone* podem estar presentes em processos de negócio. Em aplicações reais, não é desprezível a chance de encontrar modelos de processos de negócio que apresentem essa estrutura, caracterizada por um caminho de comunicação entre ramos paralelos

Algoritmo 6.2 Algoritmo de geração de expressões algébricas a partir de sp-digrafos bem-formados, identificação de conflitos e pontes de *Wheatstone*

Entrada: Um digrafo acíclico de expressão D

```

1:
2:  $possui\_conflito \leftarrow \text{false}$ 
3:
4: enquanto  $D$  é redutível faça
5:   para todo  $v \in V(D)$  não-terminal faça
6:     se  $d^-(v) = d^+(v) = 1$  então
7:        $u \leftarrow \text{pred}(v);$ 
8:        $w \leftarrow \text{suc}(v);$ 
9:        $A(D) \leftarrow A(D) \cup \{(u, w)\}$ 
10:       $\theta(uw) \leftarrow \theta(uv) \cdot \theta(vw)$ 
11:       $A(D) \leftarrow A(D) \setminus \{(u, v), (v, w)\}$ 
12:       $V(D) \leftarrow V(D) \setminus \{v\}$ 
13:     fim se
14:   fim para
15:
16:   para todo  $v \in V(D)$  e  $\neg possui\_conflito$  faça
17:     para todo  $w \in V(D)$  e  $w \in \text{suc}(v)$  faça
18:       se  $\|vw\| > 1$  então
19:         se  $\pi(v) \neq \pi(w)$  então
20:            $possui\_conflito \leftarrow \text{true}$ 
21:         fim se
22:         {Sejam  $a_1, a_2, \dots, a_k$  os arcos  $v \rightarrow w$  em  $A(D)$ .}
23:          $\theta'(vw) \leftarrow \theta(a_1)$ 
24:          $A(D) \leftarrow A(D) \setminus \{a_1\}$ 
25:         para todo  $2 \leq i \leq k$  faça
26:            $\theta'(vw) \leftarrow \theta'(vw) \pi(v) \theta(a_i)$ 
27:            $A(D) \leftarrow A(D) \setminus \{a_i\}$ 
28:         fim para
29:          $A(D) \leftarrow A(D) \cup \{vw\}$ 
30:          $\theta(vw) \leftarrow \theta'(vw)$ 
31:          $\pi(v) \leftarrow \pi(w) \leftarrow \cdot$ 
32:       fim se
33:     fim para
34:   fim para
35: fim enquanto
36: devolve  $D$ 

```

de execução. Baseado nessa informação, este capítulo sugere formas de trabalho com tais estruturas, de modo que, apesar da presença de uma ponte de *Wheatstone*, seja possível encontrar uma expressão algébrica para o digrafo dado como entrada.

Esta seção estuda a obtenção de expressões algébricas a partir de pontes de *Wheatstone* isoláveis. O objetivo é mostrar como obter expressões algébricas de pontes de *Wheatstone* por meio de transformações topológicas, conservando ao máximo a equivalência entre o digrafo e a expressão final. Uma mudança topológica normalmente provoca perda de bissimulação, pois altera a estrutura de ramificação do grafo, mas as mudanças propostas a seguir consideram níveis de equivalência mais relaxados, procurando preservar ao máximo a equivalência da expressão algébrica em relação à ponte de *Wheatstone*.

Figura 6.6: Ponte de *Wheatstone* Figura 6.7: Digrafo acíclico com pontes de *Wheatstone* isoláveis

Chamaremos de pontes de *Wheatstone* isoláveis todos os subgrafos isomórficos ao da Figura 6.6, nos quais os vértices internos (2 e 3) não possuam, no digrafo original, arcos envolvendo vértices externos ao subgrafo. Dessa forma, o subgrafo sempre apresentará quatro vértices, dos quais dois são vértices de ramificação (1 e 2) e dois de sincronização (3 e 4).

Definimos, assim, a transformação de pontes de *Wheatstone* isoláveis, cujo objetivo é encontrar uma mudança topológica que, ao mesmo tempo, transforme a ponte de *Wheatstone* em um sp-digrafo e mantenha a equivalência algébrica dos termos representados por seus arcos e vértices. Dessa maneira, será possível estender a aplicação do Algoritmo 6.1 para um número maior de digrafos, com construções semelhantes à Figura 6.7. Nela, a aplicação sucessiva de reduções paralelas e seqüenciais leva a um digrafo reduzido com pontes de *Wheatstone* isoláveis. Se, de alguma forma, for possível converter as pontes em sp-digrafos, o algoritmo pode ser aplicado novamente, construindo as expressões algébricas.

Para isso, entretanto, a transformação de uma ponte de *Wheatstone* em um sp-digrafo deve considerar outros fatores, além da topologia. A relação entre ações (arcos) e as regras de ramificação e sincronização (vértices) pode representar sistemas com grande complexidade. Sendo assim, cada transformação deve gerar um sp-digrafo observando não somente a disposição topológica, mas o seu impacto sobre a conservação dos termos representados no digrafo original.

A definição da transformação da ponte de *Wheatstone* será orientada pela conservação de equivalência dos termos algébricos dos arcos e vértices. Para isso, será usada uma comparação entre os grafos de processos da ponte de *Wheatstone* e o resultado da transformação. A equivalência entre grafos de processos garante, entre outras coisas, que propriedades de execução entre dois processos sejam as mesmas, de modo que seja possível substituir um processo pelo outro sem perdas. Como vimos no Capítulo 3, há várias formas de avaliar a equivalência entre processos, desde critérios mais rigorosos (como a equivalência por bissimulação) até os mais relaxados (como a equivalência por rastros). A mudança topológica provoca alterações que, em alguns casos, inviabilizam a construção de sp-digrafos bissimilares aos originais; porém, mesmo nesses casos, veremos que é possível construir expressões equivalentes, desde que os critérios de equivalência sejam mais relaxados.

Algumas pontes de *Wheatstone* podem ainda apresentar conflitos (*deadlocks* e falhas de sincronização), dificultando ainda mais a busca por sp-digrafos equivalentes. Algumas combinações de operadores de ramificação e sincronização produzem conflitos, como foi apresentado na Seção 5.4. Nesses casos, cada conflito tem um tratamento específico, que consiste em sua solução, quando possível, ou na remoção dos elementos que o provocam.

A seguir, apresentamos as possíveis disposições de uma ponte de *Wheatstone*. A combinação de seus vértices dá origem a 16 casos, que serão organizados de acordo com o tipo e a presença de conflitos. Para cada caso, são apresentadas uma breve explicação e três figuras:

- (a) o grafo de processo que mostra as possíveis execuções que a ponte de *Wheatstone* pode apresentar;
- (b) um sp-digrafo equivalente à ponte de *Wheatstone* (quando possível);
- (c) o grafo de processos do item (b), que mostra seus rastros de execução e permite a comparação de equivalência com o item (a).

O grafo de processo (a) tem o objetivo de mostrar a estrutura de ramificação que orienta a execução de cada instância do digrafo de expressão com estrutura isomórfica à ponte de *Wheatstone*. Quando for possível a construção de um sp-digrafo equivalente, ele é mostrado por (b) como uma estrutura que pode substituir a ponte de *Wheatstone*. Finalmente, o grafo de expressões (c) mostra o grafo de processo do sp-digrafo de (b), de modo que é possível comparar os níveis de equivalência com o (a). Para cada caso, também é apresentada uma expressão algébrica, que pode ser resume o comportamento do sp-digrafo (b).

A seguir, são apresentadas as 16 configurações que podem ser assumidas por uma ponte de *Wheatstone* isolável.

6.3.1 Casos com presença de Deadlock Total

A principal característica destes casos é a presença de *deadlock* em todos os rastros. Todos os casos ilustrados na Figura 6.8, levam o processo, invariavelmente, a um estado de *deadlock*.

Para cada caso, apresentamos também seu grafo de processo, que mostra explicitamente as observações de execução, ou rastros, que levam ao *deadlock*. No grafo de processo, sub-árvores com arcos pontilhados indicam estados que levam a um *deadlock*. Os sp-digrafos equivalentes não serão mostrados,

Figura 6.8: Casos com a presença de *deadlock* total

pois casos com *deadlock* total não possuem rastros livres de conflitos, de modo que nenhum sp-digrafo possuirá rastros bem-sucedidos. Para esses casos, as expressões sempre finalizam com um predicado de *deadlock* (δ) e são obtidas por meio dos grafos de processos.

Caso 1

A primeira situação analisada é mostrada na Figura 6.8 (a). O Caso 1 apresenta uma sincronização de termos paralelos nos vértices 3 e 4. Sendo assim, 4 espera pela execução dos termos d e e , mas observa a execução de somente um deles. Isso ocorre porque o vértice 2 habilita a execução exclusiva de um dos termos seguintes (c ou d), caracterizando o *deadlock* em 4. Note que, em nenhum rastro observado da raiz até uma das folhas, os termos d e e estão presentes ao mesmo tempo.

A Figura 6.9 apresenta o grafo de processo do Caso 1, no qual é possível observar todas as seqüências de execução que podem ser obtidas a partir dessa estrutura. A expressão equivalente a este caso é obtida por meio da observação dos rastros possíveis sobre o grafo de processos, que resulta em

$$(a \cdot c \parallel b) \cdot e \cdot \delta + (a \cdot d \parallel b) \cdot \delta$$

Caso 2

No Caso 2, mostrado na Figura 6.8 (b), podemos dizer que o *deadlock* é um pouco mais explícito, pois o termo e nunca será executado. O vértice 3, que sincroniza as ações c e b , acusará somente uma delas a cada instância, pois o vértice 1 habilita alternativamente as ações a e b . Dessa maneira, o vértice 3 apresenta um estado de *deadlock*, que é propagado também para 4 inviabilizando todas as seqüências de execução dos caminhos deste caso.

$$a \cdot d \cdot \delta + b \cdot \delta + a \cdot c \cdot \delta$$

No Caso 3, mostrado na Figura 6.8 (c), tem-se uma sincronização de termos paralelos no vértice 4. Apesar da sincronização, nenhum vértice produz uma ramificação paralela, de modo que as ações d e e não serão executadas juntas na mesma instância.

$$a \cdot d \cdot \delta + a \cdot c \cdot e \cdot \delta + b \cdot e \cdot \delta$$

Figura 6.11: Grafo de Processo do Caso 3

Figura 6.12: Grafo de Processo do Caso 4

Caso 4

No Caso 4, mostrado na Figura 6.8 (d), também ocorre uma situação similar ao Caso 3. O vértice 3 realiza a sincronização paralela das ações b e c . Sendo assim, a execução do termo e depende diretamente da execução de b e indiretamente da execução de a . O vértice 1, porém, só habilita um desses termos (a ou b), caracterizando o *deadlock* em 3. Esse *deadlock* é propagado para o vértice 4, de modo que nenhuma seqüência de execução de termos consegue ser finalizada com sucesso.

A Figura 6.12 ilustra a descrição acima por meio do grafo de processo do Caso 4, onde podem ser observadas as suas situações de *deadlock*. A exemplo do Caso 2, o termo e não está presente em nenhum rastro. Sua expressão equivalente é

$$a \cdot (c \parallel d) \cdot \delta + b \cdot \delta$$

6.3.2 Casos com presença de Deadlock Parcial

Em algumas situações, a ponte de *Wheatstone* apresenta *deadlocks*, porém existem rastros que podem representar execuções bem-sucedidas. Nesses casos, dizemos que a ponte de *Whatstone* apresenta *deadlock* parcial, pois, apesar da presença de *deadlocks*, há instâncias que podem finalizar com sucesso. Os quatro casos em que isso ocorre são apresentados na Figura 6.13.

Figura 6.13: Casos com a presença de *deadlock* parcial

Figura 6.14: Grafo de Processo do Caso 5

A transformação, nesse caso, altera topologicamente o digrafo, considerando somente os caminhos válidos, e ignorando situações de *deadlock*. Assumimos, assim, que os caminhos que levam a um conflito não são desejáveis no modelo de negócios, de modo que não serão consideradas no processo de transformação. Para esses casos, a equivalência entre processos também considera somente os caminhos sem conflito. As expressões resultantes da transformação, assim, são consideradas bissimilares somente em relação aos rastros bem-sucedidos.

Caso 5

No Caso 5, mostrado na Figura 6.13(a), ocorre *deadlock* no vértice 3, que sincroniza os termos *b* e *c*. Isso ocorre porque somente um deles é executado em cada instância. Apesar disso, a sequência de termos que não passa por 3 é válida e produz uma instância bem-sucedida.

A Figura 6.14(a) exibe o grafo de processo deste caso, mostrando o caminho válido e os caminhos

Figura 6.15: Grafo de Processo do Caso 6

que apresentam *deadlock*. Note que, para finalizar com sucesso, a ponte de *Wheatstone* necessita da execução de *d* ou *e* antes do vértice 4. O único rastro que apresenta essa possibilidade é que chega na folha mais à esquerda.

Na Figura 6.14(b), vemos o grafo de processo da expressão $a.d$, resultado da transformação, e que representa todos os caminhos válidos. A Figura 6.14(c) exibe o digrafo obtido pelos caminhos válidos deste caso, cuja expressão equivalente é

$$a.d$$

Caso 6

No Caso 6, mostrado na Figura 6.13(b), o vértice 2 define se a execução apresentará um estado de *deadlock*. O vértice de sincronização paralela 4 espera pela execução dos termos *d* e *e*, enquanto o vértice 2 habilita alternativamente *c* ou *d*. Se 2 habilitar o termo *c*, 4 receberá duas vezes o termo *e*, não realizando a sincronização, pois o termo *d* não será executado.

O caminho válido, neste caso, exclui a execução de *c*, considerando a execução paralela dos caminhos $a.d$ e $b.e$ no digrafo.

A Figura 6.15 exibe o grafo de processo deste caso, mostrando o caminho válido e os caminhos que apresentam *deadlock*. Note que o vértice 4 espera pela execução dos termos *d* e *e*. Os rastros em que *d* e *e* ocorrem juntos também são aqueles com execuções bem-sucedidas.

A Figura 6.16(a) mostra o grafo de processos da expressão $a.d \parallel b.e$, que é resultado da transformação sobre o Caso 6 e tem o sp-digrafo correspondente mostrado na Figura 6.16(b).

Figura 6.16: (a) Grafo de Processo da Transformação do Caso 6; (b) SP-digrafo resultado da transformação

Figura 6.17: Grafo de Processo do Caso 7

Caso 7

No Caso 7, mostrado na Figura 6.13(c), o vértice 1 habilita um dos termos, ou a , ou b . Habilitando b , o vértice 3 fica em estado de espera por c para efetuar a sincronização de ambos e dar andamento à execução da instância. O termo c , entretanto, não ficará disponível, uma vez que depende da execução de a , caracterizando o *deadlock*.

Por outro lado, se o termo a for habilitado, existe um caminho com execução válida formado pelos termos a e d . Um detalhe interessante desta estrutura é que o termo c , assim como o termo d , também é habilitado por 2, porém não tem influência na finalização bem-sucedida da instância, ficando, inclusive, bloqueado em 3.

A Figura 6.17(a) exibe o grafo de processo deste caso, mostrando os caminhos válidos e o caminho que apresenta *deadlock*. Semelhante ao Caso 5, a execução de d ou e caracteriza um rastro bem-sucedido. Isso ocorre somente se o termo b não for executado. Um fato curioso nessa situação é que há uma rastro, no qual o termo c executa antes de d . Isso ocorre porque o vértice 2 habilita paralelamente c e d . Para

Figura 6.18: Grafo de Processo do Caso 8

conservar a equivalência de rastros, a expressão algébrica gerada pela transformação considera essa situação, originando a expressão $a \cdot d + a \cdot c \cdot d$, cujo grafo de processo pode ser visto na Figura 6.17(b).

O sp-digrafo de expressão para este caso pode ser visto na Figura 6.17(c).

Caso 8

No Caso 8, mostrado na Figura 6.13(d), o vértice 1 habilita alternativamente os termos a e b . Ao habilitar b , o digrafo apresenta *deadlock* no vértice 4, pois somente o termo e será executada. Se, ao contrário, o vértice 1 habilitar a , a seqüência de termos produzirá uma instância bem-sucedida, que pode ser observada no grafo de processo da Figura 6.18 (a).

Note que um rastro bem-sucedido necessita da execução de d e e , que só ocorre em rastros iniciados pelo termo a . A expressão resultante, $a \cdot (d||c \cdot e)$, e seu grafo de processo podem ser vistos na Figura 6.18(b). O sp-digrafo resultante da transformação é mostrado na Figura 6.18(c).

6.3.3 Casos com Falha de Sincronização

Há casos em que um vértice de ramificação habilita paralelamente vários termos que não são posteriormente sincronizadas, embora a estrutura apresente um vértice de sincronização. Essa situação, chamada *falha de sincronização*, é reconhecidamente um conflito em processos de negócio que pode gerar problemas sérios durante a execução de uma instância.

A exemplo do que foi feito nos casos anteriores, nos quais os digrafos apresentam instâncias com *deadlock*, também consideraremos somente as instâncias válidas para exercer uma transformação. Nos casos em que ocorre a falha de sincronização, mostrados na Figura 6.19, consideraremos que a primeira linha de execução que chega a seu final desabilita as demais linhas de execução pendentes.

Nos quatro casos que serão apresentados a seguir, os vértices de sincronização mais externos (4) são alternativos, necessitando da execução de d ou e finalizar corretamente. Sendo assim, todos os rastros válidos são finalizados em d ou e . Todos os outros termos estejam habilitados após a execução de d ou

Figura 6.19: Casos com falha de sincronização

e , esses serão ignorados, forçando a sincronização sobre o vértice 4. Esse tratamento é realizado para prevenir a propagação da falha de sincronização sobre o restante do processo de negócio.

Caso 9

No Caso 9, ilustrado na Figura 6.19(a), a falha de sincronização ocorre porque os termos a e b iniciam ramos de execução independentes que não são sincronizados, pois o vértice 4 espera a execução de apenas um deles.

O grafo de processo deste caso é mostrado na Figura 6.20. A expressão algébrica que considera todos os casos possíveis para este caso é:

$$a \cdot d + a \cdot c \cdot e + b \cdot e + (a \parallel b) \cdot (d + e) + (a \cdot c \parallel b) \cdot e$$

e seu grafo de processo pode ser visto na Figura 6.21. O sp-digrafo representante da expressão é mostrado na Figura 6.22.

Caso 10

A Figura 6.19(b) apresenta o Caso 10, em que todos os termos podem ser executados e o vértice 4 espera pela finalização do termo d ou do termo e . O grafo de processo deste caso pode ser visto na Figura 6.23.

A expressão resultante da transformação é:

$$(a \cdot c \parallel b) \cdot (d + e) + (a \parallel b) \cdot d + a \cdot c \cdot d + a \cdot d$$

Figura 6.20: Grafo de Processo do Caso 9

$$a.d + a.c.e + b.e + (a||b).(d + e) + (a.c||b).e$$

Figura 6.21: Grafo de Processo da Transformação do Caso 9

Figura 6.22: SP-digrafo resultado da transformação do Caso 9

Figura 6.23: Grafo de Processo do Caso 10

Figura 6.24: Grafo de Processo da Transformação do Caso 10

e o grafo de processo pode ser visto na Figura 6.24 e seu sp-digrafo é mostrado na Figura 6.25.

Caso 11

No Caso 11, ilustrado na Figura 6.19(c), a falha de sincronização ocorre quando o termo a é habilitado, uma vez que o vértice 2 habilita os termos d e c , que podem iniciar seqüências de execução que chegam ao vértice 4. A outra alternativa de execução, que passa por 3 por meio do termo b , não produz conflito. É possível acompanhar o grafo de processo deste caso na Figura 6.26(a).

A transformação, neste caso produz a expressão

$$b.e + a.d + a.c.(d + e)$$

e seu grafo de processo pode ser visto na Figura 6.26(b). O sp-digrafo correspondente pode ser visto na Figura 6.27.

Figura 6.25: SP-digrapho resultado da transformação do Caso 10

Figura 6.26: Grafos de Processo da Transformação do Caso 11: (a) Grafo da ponte de *Wheatstone*; (b) Grafo da expressão resultante.

Figura 6.27: SP-digrafo resultado da transformação do Caso 11

Figura 6.28: Grafo de Processo do Caso 12

Caso 12

No Caso 12, mostrado na Figura 6.19(d), as falhas ocorrem tanto no vértice 3, quanto no vértice 4. Os vértices 1 e 2, de ramificação, habilitam ramos paralelos de execução, que chegam a 3 e 4, que regem regras de sincronização alternativa. O resultado é que, neste caso, todos os termos podem ser executados. O grafo de processo deste caso pode ser visto na Figura 6.28.

A expressão algébrica gerada por essa transformação é:

$$a \cdot d + (a \parallel b) \cdot (d + e) + (a \cdot c \parallel b) \cdot (d + e) + a \cdot c \cdot e + b \cdot e$$

e seu grafo de processo pode ser visto na Figura 6.29 e seu sp-digrafo representante pode ser visto na Figura 6.30.

Figura 6.29: Grafo de Processo da Transformação do Caso 12

Figura 6.30: SP-digrafo resultado da transformação do Caso 12

6.3.4 Casos com Ausência de Conflitos

Finalmente, os últimos quatro casos ilustram digrafos que apresentam ausência de conflitos. Os Casos 13 e 14, na verdade, não são totalmente livres de conflitos. Eles não apresentam *deadlock*, nem falhas de comunicação, entretanto induzem a execução repetida de termos, que ficam internamente bloqueados. Esse fato, porém, não inviabiliza nenhum de seus rastros, que finalizam com sucesso.

Para os Casos 15 e 16, não são observados *deadlocks* em nenhum nível, assim como as falhas de comunicação. Nesses casos, a composição dos termos é perfeita, de modo que todos os seus rastros representam caminhos de execução válidos, sem redundância de execução de termos nem *deadlocks*.

Figura 6.31: Casos com ausência de conflitos estruturais

Vale ressaltar que esses são os únicos casos em que não ocorrem conflitos estruturais em nenhum grau.

Caso 13

A Figura 6.31(a) ilustra o Caso 13, no qual é possível observar que a regra de sincronização do vértice 3 habilita o termo e duas vezes, uma após a execução de c e outra após a execução de b . O vértice 4 sincronizará somente um desses caminhos com o termo d , de forma que o outro caminho ficará em modo de espera. O tratamento desse caso será feito de forma semelhante aos casos com falha de sincronização, considerando que somente um dos caminhos (terminados em c ou b) será aproveitado em 4.

A Figura 6.32 mostra o grafo de processo do Caso 13. A expressão algébrica gerada pela transformação é:

$$(a \cdot (c \| d) \| b) \cdot e + (a \cdot c \| b) \cdot (d \| e) + (a \cdot d \| b \cdot e)$$

e seu grafo de processo pode ser visto na Figura 6.33. Seu sp-digrafo correspondente pode ser visto na Figura 6.34.

Caso 14

A Figura 6.31(b) exibe o Caso 14, em que há uma situação de bloqueio sem interferência no resultado final. Na ponte de *Wheatstone*, note que os termos a e b são habilitados pelo vértice 1. Para habilitar o termo d , o vértice 2 deixa o termo c desabilitado, gerando um estado de espera em 3. Essa espera, entretanto, não gera problemas na execução da estrutura, que finaliza com a ação d . Caso o termo c seja habilitado no vértice 2, não há conflitos.

O grafo de processo pode ser visto na Figura 6.35 e a expressão gerada é :

Figura 6.32: Grafo de Processo do Caso 13

Figura 6.33: Grafo de Processo da Transformação do Caso 13

Figura 6.36: Grafo de Processo da Transformação do Caso 14

Figura 6.37: SP-digrafo resultado da transformação do Caso 14

Figura 6.38: (a) Grafo de Processo da Transformação do Caso 15; (b) Grafo da Expressão Resultante da Transformação.

Figura 6.41: Grafo de Processo da Transformação do Caso 16

Figura 6.42: SP-digrafo resultado da transformação do Caso 16

6.4 Análise de Expressões Resultantes de Transformações

O tratamento de pontes de *Wheatstone* representa um passo de grande importância na geração de expressões algébricas. Um grande número de processos de negócios pode, eventualmente, apresentar modelos baseados em grafos com a presença de pontes de *Wheatstone*. Construções que envolvem essas pontes normalmente produzem conflitos, provocados pela combinação de diferentes operadores de composição na mesma estrutura.

Sendo assim, uma das hipóteses deste trabalho é que o número de pontes de *Wheatstone* não deve ser elevado dentro do processo de negócios, pois isso elevaria sensivelmente o número de conflitos estruturais. Por esse motivo, a análise dos digrafos que não pertencem à classe série-paralelo se restringirá somente aos digrafos com pontes de *Wheatstone* isoláveis, como aquelas mostradas até o momento. Usando o Teorema 2.3.3, foi possível mostrar que, invariavelmente, um digrafo que não é série-paralelo possui um subdigrafo que pode ser reduzido a uma ponte de *Wheatstone*. Se, ainda, essa ponte for isolável, é possível, por meio de transformações, aplicar o algoritmo 6.1.

Apesar disso, permanece a dúvida sobre o impacto produzido pelas transformações sobre os termos, uma vez que é necessário fazer alterações topológicas consideráveis sobre o digrafo, que poderiam

modificar a estrutura de ramificação do grafo de processo.

Esta seção mostra que as transformações buscam preservar o maior grau possível de equivalência em relação à ponte de *Wheatstone* original. Os níveis de equivalência variam desde a bissimilaridade até rastros e, em alguns casos, necessitam de condições particulares para que sua geração possa ser considerada viável em relação ao consumo de tempo.

6.4.1 Análise de Equivalência de Transformações

A forma mais natural de classificar um sistema em relação a outro é analisar a equivalência entre ambos. A equivalência de sistemas, como mostrado no Capítulo 3, serve como instrumento de comparação, podendo determinar, por exemplo, se um sistema pode ser simulado ou substituído por outro.

Em STRs, a bissimulação é uma propriedade usada para caracterizar o mais alto nível de equivalência entre processos. Para a álgebra de processos, dois processos p e q são ditos *equivalentes* se p é bissimilar a q . Uma das formas mais usadas de análise de equivalência é a comparação entre grafos de processos. Por meio dela, é possível determinar se dois processos possuem os mesmos rastros (equivalência por rastros) ou até a mesma estrutura de ramificação (equivalência por bissimulação).

Uma das consequências da aplicação de transformações sobre pontes de *Wheatstone* é, em geral, a perda da equivalência por bissimulação. Seja p um processo com grafo isomórfico a uma ponte de *Wheatstone* e q o processo resultante da transformação de p num sp-digrafo. A transformação sobre p altera sua topologia, modificando também sua estrutura de ramificação. Dessa forma, nem todas as suas propriedades topológicas são conservadas. A Tabela 6.4 mostra a equivalência entre os casos observados nas seções anteriores e suas respectivas expressões algébricas.

Embora a mudança topológica provoque mudanças na estrutura de ramificação do digrafo, ainda é possível conseguir expressões bissimilares em alguns casos. Isso pode ser observado nos Casos 2, 3, 5, 6, 8 e 15. Os Casos 1, 4, 7, 9, 10, 11, 12, 13 e 14 perdem a bissimilaridade, mas conservam a equivalência por rastros. O único caso em que não é possível garantir nenhum nível de equivalência é o caso 16, a menos que os termos sejam ações atômicas. Neste caso específico, também é garantida a equivalência por rastros. Os Casos 1-4 sempre geram *deadlocks* em todos os caminhos, de modo que convencionamos atribuir, às pontes de *Wheatstone*, um predicado de *deadlock*. Nos Casos 5-8, há caminhos que podem levar o sistema a um estado de *deadlock*. Nesses casos, se considerarmos a análise de equivalência somente com os caminhos que não geram conflitos, é possível encontrar uma expressão bissimilar. No Caso 15, todos os termos se relacionam por meio de operadores alternativos, formando caminhos sem semântica de concorrência. Nesses casos, também é possível conseguir expressões bissimilares.

Nos demais casos, entretanto, a bissimulação é um critério muito intransigente para caracterizar processos equivalentes. A bissimulação exige que, além dos mesmos caminhos, os grafos de processos de p e q apresentem a mesma estrutura de ramificação. Se relaxarmos o critério de equivalência, é possível mostrar que p e q , embora não-bissimilares, conservam algum grau aceitável de equivalência.

A equivalência de rastros indica que dois processos são capazes de gerar as mesmas observações de execução de ações sem a necessidade de que apresentem a mesma estrutura de ramificação. Pode-se mostrar que as transformações conservam a equivalência por rastros, enumerando os caminhos válidos em cada transformação. Esse foi o instrumento usado para caracterizar cada transformação nas seções

anteriores.

Tabela 6.4: Equivalência das transformações

Caso	Conflito	Nro. Rastros	Nível de Equivalência
Caso 1	<i>deadlock</i>	total	rastros
Caso 2	<i>deadlock</i>	total	bissimilaridade
Caso 3	<i>deadlock</i>	total	bissimilaridade
Caso 4	<i>deadlock</i>	total	rastros
Caso 5	<i>deadlock</i>	parcial	bissimilaridade
Caso 6	<i>deadlock</i>	parcial	bissimilaridade
Caso 7	<i>deadlock</i>	parcial	rastros
Caso 8	<i>deadlock</i>	parcial	bissimilaridade
Caso 9	falha de sincronização	total	rastros
Caso 10	falha de sincronização	total	rastros
Caso 11	falha de sincronização	total	rastros
Caso 12	falha de sincronização	total	rastros
Caso 13	não há	total	rastros
Caso 14	não há	total	rastros
Caso 15	não há	total	bissimilaridade
Caso 16	não há	total	rastros para ações atômicas

Finalmente, o Caso 16 é o mais desafiador de todos. Isso porque a geração de expressões algébricas a partir de digrafos sempre é mais complexa quando existem operadores de composição paralela envolvidos. Os operadores de composição paralela exigem uma análise da semântica de concorrência entre diferentes ramos de execução. Essa análise pode levar a existência de um número grande de rastros. Por esse motivo, no Caso 16, é necessário restringir o tamanho e comportamento dos termos para tornar viável a geração de uma expressão. Se, entretanto, for estritamente necessário montar uma expressão com equivalência completa de rastros, pode-se recorrer a técnicas de análise de rastros, como veremos a seguir, porém com a ressalva de que o consumo de tempo pode tornar a aplicação inviável.

Equivalência de Rastros em Termos Paralelos

Garantir a equivalência de rastros em construções com composições paralelas é uma tarefa difícil. Para ilustrar, usaremos o Caso 16, que apresenta uma característica particular em relação aos demais. Nele, todos os operadores de composição são paralelos, originando um grande conjunto de rastros. Analisando os rastros observados para este caso, chegamos a uma expressão que mantém a equivalência por rastros. Isso é sempre verdade quando os termos representados pelos arcos, ou parte deles, são ações atômicas. Infelizmente, se determinados termos não forem ações atômicas, alguns rastros são perdidos, como o fenômeno descrito na Seção 3.5.

Quando os arcos de um digrafo de expressão representarem termos com composição de ações atômicas, é possível chegar numa expressão com equivalência de rastros, porém, esta operação pode consumir, no pior caso, tempo exponencial em relação ao número de ações do termo.

Para ilustrar essa situação, podemos usar o exemplo da Figura 6.43(a), na qual temos uma ponte de *Wheatstone* idêntica ao Caso 16 visto na seção anterior. Considere que os termos a , b , c , d e e são todos formados por uma única ação atômica de mesmo nome (por simplicidade). A ponte de *Wheatstone* pode ser redesenhada usando sua ordenação topológica, como mostrado na Figura 6.43(b). Esse desenho é útil na visualização da relação entre as execuções paralelas dos termos.

Figura 6.43: Caso 16: (a) Ponte de *Wheatstone*; (b) Caso 16 desenhado como ordem topológica.

Considerando a álgebra de processos sem comunicação, podemos perceber que a execução do termo d ocorre paralelamente à execução da seqüência dos termos c e e . Sendo assim, é razoável admitir que d executará paralelamente a c (ver Figura 6.44(a)), ou paralelamente a e (ver Figura 6.44(b)). A transformação do Caso 16 considera uma alternativa entre os dois casos, gerando dois sp-digrafos que, combinados, resultam na expressão

$$((a.(c \parallel d)) \parallel b) \cdot e + (b \parallel a \cdot c) \cdot (d \parallel e)$$

Figura 6.44: Caso 16: Possíveis execuções do termo d

Essa expressão é equivalente por rastros às construções (a) e (b) da Figura 6.43, como pudemos avaliar na seção anterior, no estudo do Caso 16. Vamos considerar, agora, uma modificação no exemplo, assumindo que d é um rastro, ou seja, uma expressão algébrica formada somente por composições seqüenciais de ações atômicas. Assumindo que d é o rastro $d_1 \cdot d_2 \cdot \dots \cdot d_n$, além das situações vistas nas Figuras 6.44(a) e (b), é necessário considerar que parte do termo d será executado em paralelo com c e o restante em paralelo com e (ver Figura 6.45).

Figura 6.45: Caso 16: Possíveis execuções do rastro representado por d

O número de situações analisadas depende do número de ações atômicas em d . Para cada i , $1 \leq i \leq n - 1$, temos uma partição de d que dá origem a um conjunto de rastros válidos para expressão inteira. Sendo assim, a expressão é formada pela interpretação de $n + 1$ sp-digrafos. Considerando que cada grafo leva um tempo $O(k)$ para gerar a expressão algébrica, temos que o consumo de tempo para gerar um expressão equivalente por rastros é $O(n \cdot k)$ quando d é um rastro.

Infelizmente, o termo d pode ser mais complexo, consistindo num conjunto grande de rastros. O pior caso ocorre quando todos os operadores de composição de d são paralelos. Nesse caso, as n ações de d dariam origem a $k \cdot n!$ rastros, ou seja, a expressão equivalente por rastros teria que analisar $(n + 1) \cdot (k \cdot n!)$ sp-digrafos, consumindo uma função de tempo da ordem de $O(k \cdot (n + 1)!)$.

A conclusão desse caso é que a obtenção de uma expressão que reproduza uma ponte de *Wheatstone* com todos os operadores paralelos pode ser uma tarefa cara se quisermos garantir a equivalência por rastros. Se considerarmos instrumentos algébricos mais poderosos, como a presença de uma função de comunicação, o número de possibilidades aumenta ainda mais.

Métodos alternativos para a geração mais eficiente de expressões algébricas equivalentes podem envolver restrições. Se houver informações adicionais sobre o tempo estimado de execução de cada ação atômica, por exemplo, é possível desconsiderar alguns casos, gerando expressões algébricas menores.

Neste trabalho, embora a garantia de equivalência seja profundamente desejável, o trabalho com expressões extensas não é interessante. Além do consumo de tempo exponencial usado para gerar as expressões em alguns casos, um outro fator não pode ser ignorado: as expressões resultantes de tais processos também são muito extensas. O tamanho das expressões pode tornar seu uso inviável no mecanismo de execução que as utiliza como entrada. Por esse motivo, este trabalho assume restrições que desistem de níveis maiores de equivalência para composições paralelas como o Caso 16, a fim de

conseguir melhor desempenho na produção de expressões.

6.5 Conclusões

Este capítulo apresentou o algoritmo para geração de expressões algébricas a partir de digrafos de expressão. Além do algoritmo, foi mostrado como utilizar os dados de entrada do usuário, na forma de dependências entre atividades, para gerar um digrafo de expressão. O algoritmo gera a expressão algébrica de um sp-digrafo de expressão em tempo $O(m^2)$, onde m é o número de arcos do digrafo. Para digrafos que não são série paralelo, o algoritmo devolve o digrafo reduzido, que possui uma ponte de *Wheatstone*.

Digrafos com pontes de *Wheatstone* não podem ser reduzidos pelo algoritmo de geração de digrafos, mas, em alguns casos, é possível transformar a topologia do digrafo, de modo que ele pertença à classe dos sp-digrafos. Essa mudança topológica pode ser feita em pontes de *Wheatstone* isoláveis e provoca alterações na equivalência dos processos.

A disposição das regras de composição dentro da ponte de *Wheatstone* pode provocar conflitos estruturais. O tratamento dessas situações é feito desconsiderando rastros com conflitos (para *deadlocks*), ou corrigindo o conflito por meio da adaptação de seus rastros (para falhas de sincronização). O tratamento de pontes de *Wheatstone* isoláveis é rápido e não gera impacto significativo no tempo de execução do algoritmo, garantindo a sua viabilidade.

Considerando a presença quase certa de conflitos em pontes de *Wheatstone*, é razoável assumir que construções desse tipo não sejam freqüentes em processos de negócios, ou que representem problemas de modelagem que mereçam atenção.

O tratamento de digrafos com pontes de *Wheatstone* não-isoláveis é um problema de difícil resolução, uma vez que é necessário analisar individualmente cada construção. É possível ter uma idéia da complexidade desses casos tomando como parâmetro o tratamento de pontes de *Wheatstone* isoláveis. Nas pontes isoláveis, formadas por 4 vértices, são possíveis 16 combinações de operadores de composição, das quais 12 apresentam conflitos que influenciam na expressão final, 2 apresentam conflitos não influenciam na expressão final e apenas 2 são livres de conflitos. Um aumento do número de vértices provoca um crescimento exponencial no número de casos que devem ser analisados que, em sua maioria também devem apresentar conflitos. As únicas situações em que é possível modelar um padrão bem-comportado de execução ocorrem quando todos os vértices têm o mesmo operador de composição (“+” ou “||”). Nesses casos, não é possível adotar estratégias de redução (pois a estrutura do digrafo não permite esse tratamento), mas a obtenção de expressões algébricas ainda é possível utilizando outras técnicas, como variações do algoritmo de busca de caminhos em grafos. Embora importante, o tratamento desses casos foge da linha de desenvolvimento deste trabalho, de modo que não será tratado no corpo principal do texto, porém uma discussão mais detalhada pode ser encontrada no Apêndice A.

Capítulo 7

Estudo de Caso

7.1 Introdução

Neste capítulo, ilustraremos a aplicação dos algoritmos descritos nas seções anteriores em um estudo de caso baseado em situações reais. Utilizaremos, como estudo de caso, a DECA Eletrônica [6], o processo de negócio responsável pela abertura de empresas e controle de dados cadastrais de contribuintes da Secretaria da Fazenda do Estado de São Paulo. Este capítulo faz uma breve introdução ao sistema, seus recursos e, em seguida, apresenta a modelagem do processo de negócio, estabelecendo as relações entre suas atividades e aplicando o algoritmo para geração de expressões algébricas em ACP.

A Seção 7.2 apresenta o sistema da DECA Eletrônica e mostra a geração da expressão algébrica que representa seu comportamento. As conclusões deste capítulo são mostradas na Seção 7.3.

7.2 DECA Eletrônica

A **Declaração Cadastral Eletrônica** (DECA Eletrônica) da **Secretaria da Fazenda** do Estado de São Paulo (SEFAZ-SP) é um sistema cadastral de grande porte responsável pelo gerenciamento de informações cadastrais das empresas do estado de São Paulo, respondendo por funções como abertura de empresas, alterações de dados cadastrais e encerramento de atividades. Todos esses serviços estão disponíveis no portal de serviços da SEFAZ-SP (<http://pfe.fazenda.sp.gov.br>), denominado Posto Fiscal Eletrônico (PFE).

Entre todas as funções, o processo de **Abertura Inicial de Empresas** merece especial atenção, por sua significância, abrangência e representatividade em relação aos demais. Por esse motivo, consideraremos esse subsistema em nosso estudo de caso. Além disso, esse exemplo assume pequenas alterações no fluxo cadastral, desconsiderando pontos de ciclo do processo de negócio. Maiores detalhes sobre o processo integral de abertura inicial de empresas da DECA, ou as outras funções, podem ser encontrados em [6].

7.2.1 DECA Eletrônica - Abertura Inicial

Pela opção de abertura inicial do PFE, na DECA Eletrônica, os contribuintes do ICMS (Imposto sobre Circulação de Mercadorias e Prestação de Serviços) podem solicitar abertura inicial de empresas. O processo de abertura inicia por meio do preenchimento e envio de formulário de solicitação. Junto à

solicitação inicial, a empresa deve apresentar registros, alvarás e licenças junto a outros órgãos governamentais, dependendo do ramo de atividade. Na DECA Eletrônica, esses documentos são verificados eletronicamente, por meio de acessos a outros sistemas, externos à SEFAZ-SP.

Os sistemas externos são:

- Junta Comercial do Estado de São Paulo (JUCESP);
- Vigilância Sanitária, órgão ligado à Secretaria de Saúde do Governo de São Paulo, que fornece o “Alvará da Vigilância Sanitária”;
- Companhia de Tecnologia de Saneamento Ambiental (CETESB), órgão ligado à Secretaria do Meio Ambiente do governo de São Paulo, que fornece a “licença de instalação da CETESB”;
- Conselho Regional de Contabilistas (CRC), que detém os dados cadastrais do contabilista responsável pela escrituração fiscal da empresa.

Todas as solicitações cadastrais liberadas para homologação ou rejeição são roteadas para as Unidades Fiscais de jurisdição das empresas solicitantes. Nelas, os auditores fiscais efetuam a homologação ou rejeição. É bom ressaltar que cada Unidade Fiscal jurisdiciona um conjunto de municípios, e seus auditores fiscais possuem poderes legais para homologar ou rejeitar qualquer solicitação cadastral pertencente à sua jurisdição.

Para cada solicitação cadastral recebida pelo sistema, é gerado um Número de Protocolo de acompanhamento, o qual é fornecido ao contribuinte solicitante. Por meio desse número, o contribuinte poderá acompanhar todas as ações efetuadas pelo sistema e, em caso de erros nas informações prestadas, poderá interagir no fluxo de execução do processo para efetuar as devidas correções. A identificação de erros interrompe o fluxo à espera de correções nas informações prestadas. Assim que a devida correção é realizada, o sistema retoma seu fluxo.

A finalização do processo de abertura inicial ocorre com geração de uma *Inscrição Estadual* (IE) ou com a rejeição da solicitação. A IE é gerada automaticamente após a homologação. A partir da geração, a IE funciona como chave identificadora (ID) da empresa para todos os fins junto à SEFAZ. A partir da geração da IE, o Número do Protocolo perde sua função e os acessos às informações cadastrais ocorrem somente após a validação do usuário.

A síntese apresentada até agora fundamenta o fluxograma da Figura 7.1. O fluxograma é dividido em etapas, que podem ser vistas como fases bem-definidas do fluxo. Cada etapa é identificada por um número no canto superior direito de cada retângulo, e sua descrição resumida pode ser vista na Tabela 7.1.

Figura 7.1: Fluxograma da DECA Eletrônica (Adaptada de [6])

Tabela 7.1: Relação das etapas da DECA Eletrônica

Etapa	Descrição
1	<p>Preencher e enviar formulário Abertura Inicial. O formulário fornece informações preliminares, tais como:</p> <ul style="list-style-type: none"> a) CNAE - Classificação Nacional de Atividades Empresariais. Esta classificação é gerenciada pelo Instituto Brasileiro de Geografia e Estatística (IBGE); b) Regime de Apuração - classifica o regime pelo qual a empresa efetuará a apuração e recolhimento do ICMS aos cofres públicos. Existem quatro tipos de regime: Micro-empresa (ME), Empresa de Pequeno Porte Classe A ou B (também conhecido como SIMPLES) e Regime Periódico de Apuração (RPA) c) Tipo Jurídico - enquadramento jurídico da empresa segundo as Leis Comerciais Brasileiras. Exemplo: Sociedade Anônima (SA), Limitada (Ltda), Cooperativa entre outras. d) CEP - Código de Endereçamento Postal dos Correios.
2	<p>Sistema aplica as seguintes consistências nos dados recebidos por meio do primeiro passo.</p> <ul style="list-style-type: none"> a) Verifica a veracidade do CNAE informado; b) Verifica a validade de combinações entre Tipo Jurídico X Regime de Apuração; c) Verifica a existência do CEP informado por meio de acesso a tabela CEP dos Correios.
3	Sistema verifica se o Regime de Apuração informado pertence ao SIMPLES.
4	Se o regime de Apuração pertencer ao SIMPLES, o sistema deverá exigir que o contribuinte solicitante responda algumas perguntas interativamente.
5	Se o Regime de Apuração for diferente do SIMPLES ou a pergunta efetuada no passo 4 obtiver sim como resposta, o sistema gravará os dados recebidos numa base de dados.
6	Com base na CNAE, o sistema deve verificar se para a atividade da empresa é exigida a “Licença CETESB” e/ou “Alvará Vigilância Sanitária”.
7	Se a “licença CETESB” for exigida, o sistema construirá um formulário para captar seus dados.
8	Se o “Alvará Vig. Sanitária” for exigido, o sistema configurará um formulário p/ captar seus dados.
9	As informações referentes ao “Alvará Vigilância Sanitária” e “licença CETESB” devem ser agregadas num único formulário.
10	Todas as demais informações cadastrais da empresa devem ser captadas cada uma de acordo com seu formulário específico. Entretanto, todos os formulários específicos são agregados num único formulário denominado formulário completo.

Etapa	Descrição
11	Contribuinte solicitante preenche e envia o formulário completo.
12	Sistema grava os dados recebidos por meio do formulário completo numa base de dados intermediária denominada base de chegada.
13	Sistema gera o número do protocolo de acompanhamento.
14	Sistema efetua consistência nos dados recebidos por meio do formulário completo.
15 a 22	Caso não seja verificada nenhuma inconsistência nos dados recebidos, o sistema inicia, então, a busca de informações junto aos sistemas da JUCESP, CETESB, Vigilância Sanitária e CRC para efetuar os cruzamentos necessários e obter a veracidade dos dados informados.
23	Depois de efetuadas até três buscas junto aos sistemas externos, efetuados os respectivos cruzamentos, independentemente do resultado obtido, cada uma das solicitações será liberada para homologação ou rejeição.
24	Cada solicitação liberada para homologação ou rejeição será roteada ou encaminhada para as Unidades Fiscais que a SEFAZ mantém em diversas regiões do Estado.
25	Agentes do Fisco de cada Unidade Fiscal homologam ou rejeitam cada uma das solicitações de sua Jurisdição. Uma Jurisdição é composta por diversos Municípios.
26	Se a solicitação for homologada, a Inscrição Estadual será gerada automaticamente pelo sistema e mostrada no protocolo de acompanhamento.

Cada etapa do fluxograma determina uma fase do processo, ou seja, um conjunto global de atividades que é responsável por uma determinada função. Normalmente, esse conjunto pode ser dividido em componentes menores, mais especializados: as atividades atômicas do processo. O conjunto das atividades atômicas representa o modelo mais próximo da implementação, que é usado para identificar e refinar as relações de dependências. Na Tabela 7.2, vemos a divisão de cada etapa da DECA Eletrônica em atividades atômicas e suas respectivas relações de dependência.

Cada dependência é representada por um par de vértices ligados por um arco, em que os vértices representam as atividades e o arco estabelece a seqüência de execução que deve ser obedecida. A partir da definição das dependências, identifica-se os pontos de ramificação e sincronização, atribuindo-lhes vértices artificiais com as regras de ramificação e sincronização adequadas. O resultado do refinamento das dependências constitui a base para a geração do digrafo de expressão, mostrado na Figura 7.2.

Em seguida, gera-se o digrafo de linha correspondente ao da Figura 7.2, que é mostrado na Figura 7.3. O objetivo do digrafo de linha é criar um novo digrafo a partir do original, em que arcos são mapeados para vértices, conservando a mesma estrutura de dependência. A partir de então, são atribuídos os valores dos rótulos dos vértices (π) e dos arcos (θ) sobre o digrafo de linha. O digrafo resultante é o digrafo de expressão da DECA Eletrônica.

Esse digrafo será usado como entrada para a aplicação do Algoritmo 6.1. Algumas características desse digrafo devem ser ressaltadas por sua importância, como por exemplo:

- arcos representam atividades ou termos;
- vértices representam pontos de composição entre atividades ou termos;
- arcos com $(\theta = \lambda)$ representam atividades ou termos vazios.

No digrafo de expressão dado como entrada, arcos rotulados representam atividades do processo de negócio. Arcos sem rótulo possuem $\theta = \lambda$. O algoritmo de geração de expressões algébricas realiza sucessivas reduções sobre o digrafo de expressão, reduzindo gradativamente o digrafo original até a obtenção de um digrafo minimal. O seqüência de reduções pode ser vista nas Figuras 7.4 e 7.5.

O resultado final do algoritmo é um digrafo isomórfico ao sp-digrafo trivial, cujo único arco tem um rótulo com a expressão algébrica equivalente ao digrafo original. Uma implementação deste algoritmo pode ser encontrada no endereço <http://www.ime.usp.br/~koikawa/tese>.

Tabela 7.2: Relação de dependência entre passos de negócio da DECA Eletrônica

Etapa	Identificador	Atividade	Dependência
1	P01	Preencher e enviar formulário 1	P01 → P02 P01 → P03 P01 → P04 P01 → P05
2	P02	Validar CNAE	P02 → P07 P02 → P08
2	P03	Validar combinação CNAE X Tipo Jurídico	P03 → P07 P03 → P08
2	P04	Validar combinação Tipo Jurídico X Regime De Apuração	P04 → P07 P05 → P08
2	P05	Verificar duplicidade do CNPJ	P05 → P06 P05 → P07
3	P06	Verificar se Regime de Apuração escolhido é igual a SIMPLES (ME, EPP-A ou EPP-B)	P06 → P07 P06 → P08
4	P07	Montar Formulário para colher Declaração do SIMPLES	P07 → P08
5	P08	Gravar dados	P08 → P09 P08 → P13 P08 → P14 P08 → P15 P08 → P16 P08 → P17
6	P09	Verificar exigência de CETESB e Vigilância Sanitária	P09 → P10 P09 → P11
7	P10	Montar Formulário CETESB	P10 → P12
8	P11	Montar Formulário Vigilância Sanitária	P11 → P12
9	P12	Agregar Formulários CETESB e Vigilância Sanitária	P12 → P18
10	P13	Montar Formulário CNAE	P13 → P18
10	P14	Montar Formulário Quadro Societário	P14 → P18
10	P15	Montar Formulário Dados do Estabelecimento	P15 → P18
10	P16	Montar Formulário Contabilista	P16 → P18
10	P17	Montar Formulário Regime de Apuração	P17 → P18
11	P18	Contribuinte preenche e envia formulário completo	P18 → P19 P18 → P20
14	P19	Consistir dígito verificador CPFs dos Sócios	P19 → P21
14	P20	Consistir dígito verificador do número do CRC	P20 → P21
12	P21	Gravar dados na base de chegada	P21 → P22

Etapa	Identificador	Atividade	Dependência
13	P22	Sistema gera Protocolo de acompanhamento	P22 → P23 P22 → P27 P22 → P31 P22 → P35
15	P23	Acessar base de dados da JUCESP	P23 → P24
15	P24	Gravar informações de acesso JUCESP no Protocolo	P24 → P25
16	P25	Confrontar dados JUCESP	P25 → P26
16	P26	Gravar estado do cruzamento JUCESP no Protocolo	P26 → P23 P26 → P39
17	P27	Acessar Base de dados da CETESB	P27 → P28
17	P28	Gravar informações de acesso CETESB no Protocolo	P28 → P29
18	P29	Confrontar dados CETESB	P29 → P30
18	P30	Gravar estado do cruzamento CETESB no Protocolo	P30 → P39
19	P31	Acessar Base de dados da Vigilância Sanitária	P31 → P32
19	P32	Gravar informações de acesso Vig. Sanitária no Protocolo	P32 → P33
20	P33	Confrontar dados Vigilância Sanitária	P33 → P34
20	P34	Gravar informações de cruzamento Vig. Sanitária no Protocolo	P34 → P39
21	P35	Acessar Base de dados da CRC	P35 → P36
21	P36	Gravar informações de acesso CRC no Protocolo	P36 → P37
22	P37	Confrontar dados CRC	P37 → P38
22	P38	Gravar informações de cruzamento CRC no Protocolo	P38 → P39
23	P39	Liberar solicitação para homologação	P39 → P40
24	P40	Rotear solicitações p/ unidades Fiscais jurisdição	P40 → P41 P40 → P42 P40 → P43
24	P41	Solicitações retheadas para o PF 01	P41 → P44 P41 → P47
24	P42	Solicitações retheadas para o PF 02	P42 → P44 P42 → P47

Etapa	Identificador	Atividade	Dependência
24	P43	Solicitações reteadas para o PF 100	P43 → P44 P43 → P47
25	P44	Homologar solicitação	P44 → P45
26	P45	Gerar Inscrição estadual	P45 → P46
26	P46	Gravar IE gerada no Protocolo Acompanhamento	P46 → FIM
26	P47	Rejeitar Solicitação	P47 → P48
26	P48	Gravar Rejeição no Protocolo Acompanhamento	P48 → FIM

A partir desse digrafo dado como entrada, o algoritmo realiza uma seqüência de reduções, mostrada pelas Figuras 7.4 e 7.5. As reduções levam à geração da expressão

$$\begin{aligned}
& (P1 \cdot (P2 \parallel P3 \parallel P4 \parallel P5) \cdot P6 \cdot (P7 + P8)) \cdot ((P9 \cdot (P10 \parallel P11) \cdot P12) \\
& \cdot (P13 \parallel P14 \parallel P15 \parallel P16 \parallel P17)) \cdot ((P18 \cdot (P19 \parallel P20) \cdot (P21 \cdot P22)) \cdot ((P23 \cdot P24 \cdot P25 \\
& \cdot P26) \parallel (P27 \cdot P28 \cdot P29 \cdot P30) \parallel (P31 \cdot P32 \cdot P33 \cdot P34) \parallel (P35 \cdot P36 \cdot P37 \cdot P38)) \\
& \cdot ((P39 \cdot P40) \cdot (P41 + P42 + P43) \cdot ((P44 \cdot P45 \cdot P46) + (P47 \cdot P48))))
\end{aligned}$$

7.3 Conclusões

A aplicação das técnicas de geração e redução de digrafos de expressão em aplicações reais mostra-se uma forma viável de obtenção de expressões algébricas. Embora uma parte considerável das aplicações reais apresente um conjunto de dependências que leva a digrafos com pontes de *Wheatstone*, não é esperado que o número de pontes seja elevado, por todos os problemas de modelagem e controle que foram apresentados durante o texto. Há situações práticas em que sistemas apresentam pontes de *Wheatstone* em grande número, porém as regras de composição devem ser sempre as mesmas (paralela ou alternativa) em todos os seus vértices para que não ocorram conflitos.

Neste capítulo, ilustramos a utilização de reduções sobre uma adaptação de sistema real de grande porte, desde a sua definição até a obtenção da expressão algébrica. O exemplo mostrado reforça a idéia de que é viável aplicar reduções sobre digrafos de expressão para caracterizar algebricamente sistemas em produção. Além da geração de um digrafo de expressão e sua correspondente expressão algébrica, a aplicação do método permite também identificar pontos com características estruturais indesejáveis, como pontes de *Wheatstone* isoláveis e não-isoláveis, além de conflitos estruturais (*deadlocks* e falhas de sincronização).

Figura 7.2: Digrafo de dependências obtido após fase de refinamento da DECA Eletrônica

Figura 7.3: Digrafo de Expressão para a DECA Eletrônica.

Figura 7.4: Reduções sobre o digrafo de expressão da DECA Eletrônica.

Figura 7.5: Reduções sobre o digrafo de expressão da DECA Eletrônica (continuação).

Capítulo 8

Conclusões

Este trabalho explorou o problema da geração de expressões algébricas a partir de digrafos acíclicos, que tem despertado interesse de muitos pesquisadores da área de Gerenciamento de Processos de Negócio nos últimos anos. Não é possível afirmar que qualquer modelo de processo de negócio baseado em grafos sempre possui uma expressão algébrica correspondente e equivalente.

Este trabalho apresentou uma classe de digrafos acíclicos que garantidamente possui uma expressão algébrica equivalente (exceto em caso de conflitos estruturais). Para isso, foi necessário modificar a definição clássica de digrafos acíclicos, atribuindo rótulos a vértices e arcos e originando os *digrafos de expressão*. Além dos digrafos de expressão, foi definido um sistema de redução sobre esses digrafos que formalizou operações que, ao mesmo tempo, compõem termos algébricos e diminuem a ordem e o tamanho do digrafo. Essas operações fundamentaram a construção de um algoritmo que, além da geração de expressões algébricas, é capaz também de reconhecer digrafos que não são série-paralelo.

Digrafos de expressão que não são série-paralelo possuem uma ponte de *Wheatstone* em sua estrutura. Isso normalmente também incorpora problemas de consistência, como *deadlocks* e falhas de sincronização. Em aplicações reais, é possível encontrar sistemas cujo modelo apresente tais pontes. Sendo assim, este trabalho também apresentou modificações topológicas que transformam pontes de *Wheatstone* isoláveis em sp-digrafos, permitindo a geração de expressões algébricas, mesmo para esses casos.

Pontes de *Wheatstone* não-isoláveis precisam ser analisadas individualmente, pois sua complexidade não sugere nenhum padrão de comportamento para execução do processo de negócio. Os únicos casos em que é possível determinar um padrão de comportamento ocorre quando todos os vértices possuem o mesmo rótulo. Nesses casos, é possível determinar um algoritmo para geração de expressões, embora nem sempre seja possível garantir que as expressões geradas conservem altos níveis de equivalência.

Os estudos abordados neste trabalho tem considerável potencial de aplicação em sistemas reais cuja execução é controlada por meio da interpretação de expressões algébricas. Algumas das possíveis propostas de utilização podem ser:

- ferramenta de apoio à modelagem de processos de negócio. A análise das dependências e seu tratamento até a geração de uma expressão algébrica podem ser automatizados, de modo que seja

criada uma interessante ferramenta de apoio para modelagem de processos, capaz de identificar rapidamente pontos de inconsistência da especificação do processo de negócio. Esta foi uma das motivações deste trabalho, apresentadas em [18];

- automatização da geração de código de execução a partir da especificação do negócio. A expressão algébrica é praticamente um roteiro de execução das instâncias de um processo de negócio, pois determina a ordem de execução de cada ação em relação às demais. Além da álgebra, outras linguagens de codificação podem ser usadas, desde que possuam operadores de composição bem-definidos e o sistema de redução seja alterado adequadamente;
- validação de modelos, por meio da identificação de regiões com *deadlocks* e falhas de sincronização. A validação dos modelos consiste na verificação da consistência estrutural do processo de negócio. Durante a modelagem, uma ferramenta de validação pode ser útil para identificar possíveis falhas de projeto;
- avaliação de modelos construídos por algoritmos de mineração de dados. O objetivo da mineração de dados é reconstruir os modelos de processos a partir da informação de seus relatórios de execução (*logs*). A análise dos relatórios infere relações de dependência entre atividades do processo de negócio. Em alguma fase da reconstrução do modelo de processo, o algoritmo de geração pode ser útil como ferramenta de apoio para avaliar a presença de conflitos e a topologia do sistema.

8.1 Principais Contribuições

As principais contribuições deste trabalho são:

- (a) Adaptação da definição clássica de digrafos, permitindo a modelagem de processos de negócio incorporando elementos algébricos;
- (b) Definição de um sistema de redução de digrafos de expressão, em que as transformações conservam a bissimilaridade dos termos nas regiões transformadas;
- (c) Um algoritmo para geração de expressões algébricas usando digrafos de expressão;
- (d) Uma discussão sobre o tratamento de digrafos que não são série-paralelo, apontando soluções para digrafos com pontes de *Wheatstone* isoláveis e discutindo os casos mais comuns de pontes de *Wheatstone* não-isoláveis.

8.2 Limitações do Trabalho

As principais limitações do trabalho são o tratamento de dependências indiretas e o tratamento de digrafos que possuem ciclos. Uma das hipóteses do trabalho assume que o especialista de negócio é capaz de fornecer todas as dependências diretas entre atividades que compõem o processo de negócio. Em alguns casos, entretanto, é razoável imaginar que o especialista tenha interesse em representar dependências indiretas. Embora a abordagem deste trabalho apresente essa limitação, ela pode ser contornada, pois a combinação de dependências diretas é capaz de representar também as dependências indiretas. Isso pode ser feito construindo o fecho transitivo do digrafo de dependência (anterior ao

digrafo de expressão), que identifica todas as dependências entre atividades, diretas e indiretas. Além disso, a construção do digrafo de expressão pode ser acompanhada, abrindo a possibilidade de correção do modelo durante a especificação das dependências.

Outra limitação do trabalho é a aplicação de reduções sobre digrafos com ciclos. A presença de ciclos no digrafo pode dificultar sensivelmente a utilização de sistemas de redução, pois a presença de ciclos compromete a terminação do sistema. Com um sistema de redução não-terminável, não há como garantir, em grande parte dos casos, que o sistema é confluyente. Dessa forma, a corretude das reduções também é comprometida. Um outro aspecto que deve ser levado em conta é que a presença de ciclos arbitrários pode também provocar conflitos estruturais, principalmente *deadlocks*. Os *deadlocks* são provocados quando o ciclo necessita de elementos externos para iniciar uma nova iteração, mas estes não estão disponíveis. Acreditamos que é possível gerar expressões algébricas de algumas classes de digrafos com ciclos, porém essa afirmação necessita de um estudo mais profundo do comportamento dos ciclos nas estruturas de composição do digrafo de expressão.

8.3 Trabalhos Futuros

A continuidade da pesquisa apresentada neste trabalho apresenta muitos caminhos. O esforço mais imediato é mostrar que as transformações de pontes de *Wheatstone* podem ser adicionadas ao sistema de redução apresentado no Capítulo 5. A extensão do sistema de redução pode dar origem a um algoritmo de geração de expressões mais poderoso, capaz de trabalhar com um conjunto maior de digrafos. Propriedades como confluência e terminação não parecem ser problemáticas, uma vez que as transformações convergem pontes de *Wheatstone* para sp-digrafos e estes, por sua vez, formam um sistema terminável e confluyente.

Uma outra abordagem que pode ser explorada é a aplicação do sistema de reduções em redes de Petri. A exemplo do digrafo de expressão, as redes de Petri são estruturas baseadas em grafos dirigidos, mas com dois tipos de vértices, chamados lugares e transições. Uma linha de trabalho futuro é a definição de um sistema de redução para redes de Petri. Isso permitirá, em caso de sucesso, a obtenção de expressões ou a identificação de conflitos em sistemas modelados com essa estrutura.

Um caminho de pesquisa que também pode ser trilhado é a busca de algoritmos que direcionem a construção de expressões em pontes de *Wheatstone* com operadores paralelos. Como foi apresentado no Capítulo 6 e reforçado na Apêndice A, a geração de uma expressão equivalente por rastros nesses casos consome tempo exponencial. Exatamente por esse motivo, a transformação desse caso se conforma com um subconjunto de rastros. O problema é que esse subconjunto pode conter rastros ineficientes do ponto de vista de execução. Um exemplo disso seria um rastro em que uma atividade de longa duração é entrelaçada com uma atividade de curta duração. Um caminho de pesquisa seria melhorar a escolha dos rastros dessas pontes de *Wheatstone* a partir de informações sobre as atividades que compõem o processo de negócio, como prioridade, tempo de duração, etc.

Como já foi comentado anteriormente, o tratamento de ciclos é uma limitação que também pode ser melhor trabalhada a partir deste trabalho. A caracterização dos ciclos e a adaptação do sistema de redução podem ser duas importantes contribuições para a extensão do sistema de redução de digrafos de expressão, permitindo cobrir um conjunto maior de modelos de processos de negócio.

Apêndice A

Digrafos com Pontes de Wheatstone Não-Isoláveis

A.1 Introdução

Embora a aplicação de transformações sobre pontes de *Wheatstone* isoláveis permita a construção eficiente de expressões algébricas preservando níveis de equivalência mais relaxados, o mesmo não pode ser dito a respeito de pontes de *Wheatstone* não-isoláveis. Neste capítulo, veremos que a ocorrência de pontes de *Wheatstone* não-isoláveis pode levar ao tratamento de um número exponencial de termos algébricos.

Como vimos anteriormente, a disposição dos operadores de composição nas pontes de *Wheatstone* pode introduzir conflitos estruturais. Em uma ponte de *Wheatstone*, a combinação entre operadores de vértices de ramificação e sincronização normalmente leva à presença de conflitos, exceto em duas situações: quando todos os operadores nos vértices são alternativos (+) e quando são paralelos (||). Não há regra geral para tratamento de casos com conflitos estruturais; nesses casos, é necessário fazer uma análise individual da estrutura do digrafo, pois cada combinação de vértices com operadores de composição paralela e alternativa nos rótulos pode produzir uma estrutura em que o fluxo de execução não possui comportamento uniforme e previsível.

Para os casos sem conflito estrutural, já é possível conseguir estruturas mais bem comportadas, o que permite encontrar mais facilmente algoritmos para a geração de expressões. O gargalo, nesses casos, ocorre quando todos os vértices possuem operadores paralelos, pois o número de rastros converge para um valor exponencial. Para digrafos com operadores alternativos nos vértices, entretanto, o problema é tratável, sendo possível obter uma expressão algébrica em tempo polinomial.

Este capítulo apresenta uma abordagem para digrafos de expressão com pontes de *Wheatstone* não-isoláveis livres de conflitos. O tratamento desses casos têm por objetivo comprovar que o tratamento dessas estruturas não é simples, mesmo quando o digrafo não apresenta conflitos. A Seção A.2 apresenta um estudo sobre pontes de *Wheatstone* formadas por combinações alternativas de termos, apresentando um algoritmo genérico para resolver esse problema. O tratamento de operadores paralelos é apresentado na Seção A.3, que discute a geração de expressões equivalentes por rastros para digrafos de expressão com topologia de Fibonacci. Em seguida, esse tratamento é estendido para digrafos genéricos, porém sem a garantia de equivalência. Finalmente, os resultados são apresentados na Seção A.4.

Figura A.1: Digrafo de Fibonacci

Figura A.2: Rede de comunicação com operadores alternativos

A.2 Digrafos com Operadores Alternativos

Em pontes de *Wheatstone* nas quais todos os vértices de ramificação e sincronização são mapeados para operadores alternativos, é possível construir um algoritmo capaz de gerar a expressão algébrica correspondente por meio do rastreamento de caminhos entre a fonte e o sorvedouro. Isso pode ser obtido eficientemente por meio de uma variação do algoritmo de busca do caminho de menor comprimento entre dois vértices (*shortest path problem* [102]).

Em [103], essa idéia é utilizada pelo autor, que obtém bons resultados na geração de expressões algébricas em digrafos de Fibonacci. Infelizmente, o digrafo de representação de um processo de negócio pode ser bem mais complexa do que um digrafo de Fibonacci, cuja topologia é mostrada na Figura A.1. Um exemplo disso é o digrafo acíclico da Figura A.2, em que temos uma malha de comunicação entre vértices.

Nesta seção, mostraremos como estender a aplicação do algoritmo de [103] para qualquer digrafo acíclico com operadores alternativos, mostrando que é viável obter a expressão algébrica nesses casos. Além da viabilidade, outro aspecto extremamente importante se refere à equivalência. Em tais digrafos, as expressões obtidas são equivalentes por bissimulação aos digrafos de expressão originais.

A entrada do Algoritmo A.1 é um digrafo de expressão qualquer, cujos vértices de ramificação e sincronização são alternativos, e devolve a expressão algébrica correspondente.

A idéia do algoritmo é rastrear todos os possíveis caminhos entre os vértices fonte e sorvedouro. Para digrafos de expressão triviais, o caminho já está definido (linhas 3-5). Para digrafos com número maior de arcos, o algoritmo parte do vértice inicial, digamos u , para todos os seus sucessores v_i , concatena o arco (u, v_i) com os caminhos iniciados em v_i por meio de chamadas recursivas.

Uma característica desse algoritmo é que a entrada não pode ser um multidigrafo, onde é possível

Algoritmo A.1 Algoritmo de geração de expressões com operadores alternativos

```

1: BuscaCaminhos ( $v, u$ )
2:
3: se  $v = u$  então
4:   devolve  $\lambda$ 
5: fim se
6:
7:  $S \leftarrow \text{suc}(v)$ 
8:  $\text{res} \leftarrow "("$ 
9:
10: para todo  $w \in S$  faça
11:    $\text{res} \leftarrow \text{res} + \text{BuscaCaminhos}(w, u) + "+"$ 
12: fim para
13:
14:  $\text{res} \leftarrow \text{res} + ")"$ 
15:
16: devolve  $\text{res}$ 

```

a existência de mais de um arco entre dois vértices. Entretanto, isso não é uma característica que inviabiliza a utilização do algoritmo. Como vimos no Capítulo 5, se dois vértices forem unidos por vários arcos, podemos reduzi-los a um único arco pela aplicação de uma redução alternativa (que preserva também a bissimilaridade do digrafo).

Quanto à complexidade, o Algoritmo A.1 consome tempo quadrático, cujo cálculo veremos a partir de agora. Seja $T(n)$ o tempo consumido para calcular a expressão algébrica de um digrafo de expressão com n vértices. O pior caso se apresenta quando o digrafo é um torneio acíclico, pois é nesse caso que encontraremos a topologia com maior número de caminhos entre os vértices fonte e sorvedouro. A análise da complexidade do algoritmo depende da análise dos vértices e seus sucessores. Para isso, apresentaremos alguns lemas e proposições, e suas respectivas provas, que serão usados para calcular o número de caminhos.

Lema A.2.1. *Seja D um torneio acíclico e t seu vértice sorvedouro. Para todo vértice $v \in V(D)$, $v \neq t$, existe um arco vt .*

Demonstração. Prova por contradição sobre a identificação do vértice sorvedouro. Seja D é um torneio acíclico com vértice sorvedouro t e $vt \notin A$. Como D é um torneio, tv está em D , o que é uma contradição, pois t é sorvedouro e não pode dominar nenhum vértice. ■

Lema A.2.2. *Seja D um torneio acíclico e $u, v, w \in V(D)$. Se $v, w \in \text{suc}(u)$, $v \neq w$, então $|\text{suc}(v)| \neq |\text{suc}(w)|$.*

Demonstração. Prova por contradição sobre Seja D um torneio acíclico, e sejam $u, v, w \in V$, tal que v e w são sucessores de u . Suponha que $|\text{suc}(v)| = |\text{suc}(w)|$. Pela Proposição 2.2.4, todo torneio acíclico é transitivo, de modo que todos os sucessores de v e de w também são sucessores de u . Como v e w tem o mesmo número de sucessores, temos uma duas situações:

- v e w não são adjacentes, o que é uma contradição, pois D é um torneio;
- existe um ciclo em D , o que é uma contradição, pois D é acíclico.

Desse modo, $|suc(v)| \neq |suc(w)|$. ■

Proposição A.2.3. *Seja D um digrafo acíclico e $u, v \in V$ dois de seus vértices, tal que $(u, v) \in A$. Nenhum sucessor de v pode ser predecessor de u .*

Demonstração. A prova desta proposição é feita por contradição. Suponha que exista um vértice $w \in V$, que é sucessor de v e predecessor de u . Então existe, em D , um ciclo $w \dots u \dots v$, o que é uma contradição, pois D é acíclico. ■

Lema A.2.4. *Seja D um torneio acíclico e $u, v \in V(D)$. Se $v \in suc(u)$, então $|suc(v)|$ é, no máximo $|suc(u)| - 1$.*

Demonstração. Pela Proposição A.2.3, todo sucessor de v é, necessariamente, sucessor de u . Sendo assim, os sucessores de v são os mesmos de u , exceto o próprio v ; conseqüentemente $|suc(v)| = |suc(u)| - 1$. ■

Pelos Lemas A.2.2 e A.2.4, todo vértice v de um torneio acíclico tem um número diferente de sucessores, sendo que o maior número pode ser $|suc(v)| - 1$. Sendo assim, o cálculo da função $T(n)$, que calcula o tempo gasto pelo Algoritmo A.1 para gerar as expressões de um torneio acíclico de n vértices é:

$$T(n) = \begin{cases} 0, & \text{se } n = 1 \\ 1 + \sum_{i=1}^{n-1} T(i), & \text{se } n > 1 \end{cases}$$

Sendo assim, para n vértices, $n > 1$, temos:

$$\begin{aligned} T(n) &\leq 1 + T(n-1) + T(n-2) + \dots + T(2) + T(1) \\ &\leq 1 + (n-2) + (n-1) + \dots + 1 + 0 \\ &\leq 1 + (n-2)(n-1)/2 \\ &\leq (n^2 + 3n + 4)/2 \\ &= O(n^2) \end{aligned}$$

Assim, mostramos que é possível conseguir uma expressão algébrica a partir de qualquer digrafo acíclico de expressão com operadores alternativos. A expressão resultante é bissimilar ao digrafo original e pode ser obtida consumindo tempo de ordem assintótica $O(n^2)$.

A.3 Digrafos com Operadores Paralelos

Vimos que é possível encontrar expressões algébricas bissimilares a digrafos de expressão acíclicos quaisquer quando estes são compostos somente por operadores alternativos. Além disso, o consumo de tempo de um algoritmo razoável para resolução do problema não é proibitivo, o que garante a aplicabilidade da solução para casos reais. Quando, entretanto, os vértices de ramificação e sincronização forem mapeados para operadores paralelos, a busca por expressões equivalentes deixa de ser uma tarefa simples.

O operador de composição paralela ($\parallel$), por definição, representa conjuntos de rastros formados pela intercalação entre termos. O caráter associativo dessa composição provoca um aumento significativo do número de rastros possíveis. Essa situação é ilustrada pela Figura A.3, onde as ações atômicas a , b e c são compostas por meio de operadores paralelos. A Figura A.3(a) mostra a representação de intercalação entre as ações a e b , que produz a expressão $E_1 = a \parallel b$ e o conjunto de rastros $R_{E_1} = \{a \cdot b, b \cdot a\}$. Já a Figura A.3(b), mostra a composição entre três ações a , b e c , gerando a expressão $E_2 = (a \parallel b) \parallel c$, cujo conjunto de rastros $R_{E_2} = \{a \cdot b \cdot c, a \cdot c \cdot b, b \cdot a \cdot c, b \cdot c \cdot a, c \cdot a \cdot b, c \cdot b \cdot a\}$.

Figura A.3: Composição paralela: (a) Composição de duas ações atômicas; (b) Composição de três ações atômicas.

A grande dificuldade em gerar expressões equivalentes em composições paralelas está na intercalação de subtermos, que não é uma operação fechada para termos, como foi discutido na Seção 3.5. Essa situação exige que, para obter expressões equivalentes por rastros, todos os arcos do digrafo sejam mapeados exclusivamente para ações atômicas (excluindo termos). Isso ocorre porque não existe uma axiomatização que garanta a bissimilaridade sobre a composição paralela, sendo esse um problema ainda aberto [104]. Em outras palavras, não é possível aplicar reduções, pois essas podem excluir parte dos rastros possíveis da expressão.

Em alguns casos muito particulares, entretanto, é possível obter expressões algébricas equivalentes. Mesmo assim, mostraremos que essa tarefa consome tempo exponencial, em relação ao número de vértices da ponte de *Wheatstone*. A seguir, vamos estudar dois casos, sendo o primeiro a busca de expressões em digrafos de Fibonacci e, em seguida, a generalização para qualquer digrafo acíclico.

Antes de prosseguir, vamos definir o digrafo de Fibonacci. Este trabalho usa o conceito introduzido em [105], e aplicado em [103]. Além disso, vamos considerar que, no digrafo de Fibonacci, todos os termos representados pelos arcos são ações atômicas.

Definição A.3.30: (*Digrafo de Fibonacci*) Seja D um digrafo acíclico com n vértices e uma única

ordem acíclica. Assumindo a ordem acíclica $v_1, v_2, \dots, v_n$, D será um digrafo de Fibonacci se:

1. para todo vértice v_i , $1 < i < n - 1$, existe somente um arco (v_i, v_{i+1}) ;
2. para todo vértice v_i , $1 < i < n - 2$, existe somente um arco (v_i, v_{i+2}) .

A Figura A.1 já havia mostrado um digrafo com topologia de Fibonacci. Se, entretanto, redesenharmos esse digrafo, chegaremos ao digrafo mostrado na Figura A.4. Por convenção, cada vértice v_i possui dois arcos de saída: o arco a_i , que une v_i e v_{i+1} , e o arco b_i , que une v_i e v_{i+2} . A exceção ocorre em v_n , que é sorvedouro e v_{n-1} , que só possui um sucessor, v_n .

Figura A.4: Digrafo de Fibonacci

O digrafo de expressão de Fibonacci, com vértices de ramificação e sincronização paralelos pode ser visto na Figura A.5. Assumindo que cada arco a_i e b_i representa uma ação atômica, mostraremos como é possível obter uma expressão algébrica equivalente por rastros para esse digrafo.

Uma caso particular desse digrafo já foi apresentado no capítulo anterior. Quando $n = 4$, temos uma ponte de *Wheatstone*, cujo tratamento foi abordado no estudo do Caso 16. Seguindo o mesmo raciocínio usado naquela oportunidade, percebe-se que cada ação b_i é executada em paralelo com a seqüência de ações $a_i \cdot a_{i+1}$. Dessa forma, para cada b_i , os rastros possíveis são observados quando b_i é executado em paralelo com a_i ou a_{i+1} .

Assumindo que todo b_i pode ser executado em paralelo com a_i ou em paralelo com a_{i+1} , vamos usar o digrafo da Figura A.4 para construir dois novos digrafos, que chamaremos de D_1 e D_2 . D_1 , D_2 e D são topologicamente idênticos, exceto pelo comportamento do arco b_1 :

- (a) Em D_1 , o arco b_2 é paralelo ao arco a_2 ;
- (b) Em D_2 , o arco b_2 é paralelo ao arco a_3 ,

Como todos os termos associados aos arcos b_2 são ações atômicas, então podemos afirmar que o conjunto de rastros de D é igual ao conjunto de rastros de D_1 e D_2 juntos, ou seja, $T(\text{termo}(D)) = T(\text{termo}(D_1)) \cup T(\text{termo}(D_2))$. Do ponto de vista topológico, o arco b_2 deixa de fazer parte de uma ponte de *Wheatstone* e forma uma composição paralela com a_2 (em D_1) ou com a_3 em D_2 (ver Figura A.6). Repetindo esse mesmo procedimento para os demais b_i 's, obteremos um conjunto de digrafos de expressão $D_1, D_2, \dots, D_k$, de tal modo que $T(\text{termo}(D)) = T(\text{termo}(D_1)) \cup T(\text{termo}(D_2)) \cup \dots \cup T(\text{termo}(D_k))$. Além disso, cada D_j , $1 \leq j \leq k$, é um sp-digrafo de expressão.

Com isso, podemos obter a expressão algébrica equivalente por rastros a D por meio da composição alternativa de das expressões algébricas de todos os D_j . Essa é a idéia que será explorada para a

modelagem do algoritmo de geração de expressões algébricas para digrafos de Fibonacci formado por composições paralelas.

Figura A.5: Digrafo de Fibonacci

Figura A.6: Digrafo de Fibonacci de expressão com vértices paralelos

O Algoritmo A.2 mostra esse procedimento. Cada chamada do algoritmo recebe um digrafo D e um índice inteiro i , que determina qual o subgrafo será usado para o restante do cálculo. Caso o subdigrafo formado pelos vértices $v_i, v_{i+1}, \dots, v_n$ seja um sp-digrafo, então basta aplicar o algoritmo de redução sobre o mesmo. Caso contrário, o subdigrafo é de Fibonacci, e são definidos dois digrafos D_1 e D_2 , que cobrem os casos em que b_i é paralelo a a_i , ou paralelo a a_{i+1} , respectivamente. O algoritmo é, então, chamado recursivamente para D_1 e D_2 , concatenando seus resultados com a expressão algébrica criada até o momento. A Figura A.7(a) mostra a mudança topológica que ocorre em D_1 e D_2 . Em D_1 , o rótulo do arco a_{i+2} vale $\theta(a_{i+2})\|\theta(b_{i+1})$, assim como o rótulo de a_{i+1} em D_2 vale $\theta(a_{i+1})\|\theta(b_{i+1})$ (ver Figura A.7(b)). Considere, no algoritmo, que a_i^1 e b_i^1 são os arcos de D_1 correspondentes aos arcos a_i e b_i de D . Da mesma forma os arcos a_i^2 e b_i^2 são os arcos de D_2 correspondentes aos arcos a_i e b_i de D .

Dessa maneira, o consumo de tempo consiste na geração dos sp-digrafos pela cobertura das execuções de cada ação b_i , para $1 < i < n-2$. A função de consumo de tempo $T(n)$, onde n é o número de vértices é dada a seguir:

Figura A.7: Geração de expressões em digrafos de Fibonacci.

$$\begin{aligned}
T(n) &= O(1) + 2T(n-2) \\
&= O(1) + 4T(n-4) \\
&= \vdots \\
&= O(1) + 2^k T(n-2k) \\
&= \vdots \\
&= O(1) + 2^{n/2} T(0) \\
&= O(1) + 2^{n/2} O(1) \\
&= O(2^n)
\end{aligned}$$

Sendo assim, vemos que o algoritmo consome tempo $O(2^n)$. Levando em conta que o digrafo de Fibonacci é um subgrafo de um torneio acíclico, podemos afirmar que um algoritmo de geração de expressões que receba como entrada um digrafo qualquer (cujos vértices são rotulados com o operador $\parallel$) deve consumir, pelo menos, tempo $O(2^n)$.

Vamos, agora, estender e generalizar o problema a um digrafo acíclico de expressão qualquer com operadores de composição paralela em todos os vértices.

A função *encontra_caminho* (u, v) recebe dois vértices com argumentos de entrada e devolve, quando

Algoritmo A.2 Algoritmo de geração de expressões a partir de digrafos de Fibonacci acíclicos com composição paralela

Entrada: Digrafo de expressão de Fibonacci acíclico cujos vértices operadores de composição paralela.

```

1: BuscaCaminhos ( $D, i$ )
2:
3: se  $n - i = 1$  então
4:   devolve  $\theta(a_i)$ 
5: senão se  $n - i = 2$  então
6:   devolve  $\theta(a_i) \cdot \theta(a_{i+1}) \parallel \theta(b_i)$ 
7: fim se
8:
9:  $D_1 \leftarrow D$ 
10:  $D_2 \leftarrow D$ 
11:
12:  $\theta(a_{i+1}^1) \leftarrow \theta(a_{i+1}^1) + \theta(b_{i+1}^1)$ 
13:  $A(D_1) \leftarrow A(D_1) \setminus \{b_{i+1}^1\}$ 
14:
15:  $\theta(a_{i+2}^2) \leftarrow \theta(a_{i+2}^2) + \theta(b_{i+1}^2)$ 
16:  $\theta A(D_2) \leftarrow A(D_2) \setminus \{b_{i+1}^2\}$ 
17:
18: devolve “ $(\theta(a_i) \cdot \theta(a_{i+1}) \parallel \theta(b_i)) \cdot$ ”
    + BuscaCaminhos ( $D_1, i + 2$ )
    + “ $+(\theta(a'_i) \cdot \theta(a'_{i+1}) \parallel \theta(b'_i)) \cdot$ ”
    + BuscaCaminhos ( $D_2, i + 2$ )

```

existir, um caminho entre u e v que não passa pelo arco uv . Caso esse caminho não exista, o algoritmo devolve um caminho vazio, que, por conveniência, assumiremos possuir tamanho infinito. Essa função pode ser implementada como uma variação do algoritmo de busca em profundidade, com consumo de tempo de $O(m + n)$, onde m é o número de arcos e n o número de vértices.

Se D for um sp-digrafo, então basta aplicar o algoritmo visto no Capítulo 5. Caso contrário, D possui alguma ponte de *Wheatstone*. Nesse caso, a idéia do algoritmo é eliminar, um arco a_i (uv) que une vértices conexos por um outro caminho C_{uv} que não possui a_i . Para isso, criam-se k digrafos isomórficos a $D - \{a_i\}$. Cada um desses digrafos cobre um caso em que o termo $\theta(a_i)$ é executado em paralelo com um arco do caminho C_{uv} . A expressão final é a composição alternativa das chamadas recursivas sobre os digrafos k digrafos.

A expressão devolvida pelo algoritmo representa um subconjunto dos rastros possíveis. Como já foi discutido anteriormente na Seção 6.4, as reduções, quando aplicadas a digrafos com pontes de *Wheatstone* com operadores paralelos nos vértices, não conseguem cobrir a totalidade dos rastros em todos os casos. Apesar disso, tem-se a possibilidade de gerar um subconjunto de rastros válidos sobre a mesma estrutura.

A.4 Conclusões

Neste capítulo, vimos que o tratamento de digrafos com pontes de *Wheatstone* não-isoláveis pode ser complexo e consumir tempo exponencial. Para casos em que as composições são formadas por operadores

Algoritmo A.3 Algoritmo de geração de expressões com composição paralela

```

1: BuscaCaminhos ( $D$ )
2:
3: Executar o Algoritmo 6.2
4:
5: se  $D$  é isomórfico ao digrafo trivial então
6:   devolve  $\theta(a)$ , onde  $a$  é o único arco de  $D$ ;
7: fim se
8:
9: para todo  $uv \in A(D)$  faça
10:    $C_{uv} \leftarrow \text{encontra\_caminho}(u,v)$ 
11: fim para
12:
13: Escolher um arco  $uv$ , tal que  $|C_{uv}|$  é mínimo e maior que 0 e  $|C_{u,v}| \neq \infty$ 
14:
15: para todo arco  $a_i$ ,  $1 \leq i \leq k$ , no caminho  $C_{uv} = a_1 a_2 \dots a_k$  faça
16:    $D_i \leftarrow D$ 
17:    $A(D_i) \leftarrow A(D_i) - \{uv\}$ 
18:    $\theta(a_i) \leftarrow \theta(a_i) \parallel \theta(uv)$ 
19: fim para
20:
21: devolve  $\text{BuscaCaminhos}(D_1) +$ 
    $\text{BuscaCaminhos}(D_2) +$ 
    $\dots +$ 
    $\text{BuscaCaminhos}(D_n)$ 

```

alternativos, é possível obter expressões bissimilares em tempo polinomial. Quando, entretanto, as composições forem paralelas, não é possível construir uma expressão algébrica bissimilar. Além disso, pode-se notar que a construção de expressões equivalentes por rastros é exponencial, mesmo para digrafos com topologia bem comportada, como digrafos de Fibonacci.

Referências Bibliográficas

- [1] AALST, W. M. van der; HOFSTEDE, A. H. ter; WESKE, M. Business process management: A survey. *LNCS Intl. Conf. Business Process Management*, v. 2678, p. 1–12, 2003.
- [2] BANG-JENSEN, J.; GUTIN, G. *Digraphs: Theory, Algorithms and Applications*. 2nd. ed. Heidelberg: Springer-Verlag, 2002. ISBN 978-1-85233-611-0.
- [3] HARARY, F.; KRARUP, J.; SCHWENK, A. Graphs suppressible to an edge. *Canadian Mathematical Bulletin*, v. 15, n. 2, p. 201–204, 1972.
- [4] CORRADINI, A. et al. *Algebraic Approaches to Graph Transformation, Part I: Basic Concepts and Double Pushout Approach*. Corso Italia 40, 56125 Pisa, Italy, 1996.
- [5] PLUMP, D. Confluence of graph transformation revisited. In: *Processes, Terms and Cycles*. [S.l.: s.n.], 2005. p. 280–308.
- [6] ZULIANI, D. *Projeto Conceitual de Workflow*. Dissertação (Mestrado) — Instituto de Pesquisas Tecnológicas - Universidade de São Paulo (IPT-USP), 2006.
- [7] AALST, W. *Pi calculus versus petri nets: Let us eat humble pie rather than further inflate the pi hype*. 2003. Disponível em: <citeseer.ist.psu.edu/vanderaalst03pi.html>.
- [8] EHRIG, H.; PFENDER, M.; SCHNEIDER, H. Graph-grammars: an algebraic approach. In: *IEEE Conf. on Automata and Switching Theory*. [S.l.: s.n.], 1973. p. 167–180.
- [9] BAUDERON, M.; COURCELLE, B. Graph expressions and graph rewritings. *Mathematical Systems*, Springer, New York, v. 20, n. 1, 1987.
- [10] PLUMP, D. Hypergraph rewriting: Critical pairs and undecidability of confluence. In: *Term Graph Rewriting: Theory and Practice*. [S.l.]: John Wiley & Sons Ltd., 1993.
- [11] PLUMP, D. *Term Graph Rewriting*. The Netherlands, 1998.
- [12] FERREIRA, A. P. L.; RIBEIRO, L.; FOSS, L. Formal verification of object-oriented graph grammars specifications. *Electronic Notes in Theoretical Computer Science*, Elsevier Sciences, v. 175, n. 4, p. 101–114, 2007.
- [13] BARDOHL, R.; EHRIG, H.; ERMEL, C. Generic description, behavior and animation of visual modeling languages. In: 5TH INTL. CONF. ON INTEGRATED DESIGN AND PROCESS TECHNOLOGY. *IDPT*. Dallas, 2000.
- [14] KOCH, M.; MANCINI, L. V.; PARISI-PRESICCE, F. A formal model for role-based access control using graph transformation. In: 6TH EUROPEAN SYMPOSIUM ON RESEARCH IN COMPUTER SECURITY. *LNCS*. [S.l.], 2000. p. 122–139.
- [15] WU, Q. et al. Categorization and optimization of synchronization dependencies in business processes. In: 23RD INTL. CONF. ON DATA ENGINEERING. *ICDE*. [S.l.], 2007. p. 306–315.

- [16] SALAÜN, G.; FERRARA, A.; CHIRICHIELLO, A. Negotiation among web services using LOTOS/CADP. *LNCS*, v. 3250, 2004.
- [17] SHINJO, Y.; KUBO, T.; PU, C. Efficient mediators through dynamic code generation: a method and an experiment. In: SYMPOSIUM ON APPLIED COMPUTING. *SAC*. [S.l.], 2003. p. 351–357.
- [18] ZULIANE, D. et al. The riverfish approach to business process modeling: Linking business steps to control-flow patterns. In: 4TH INTL. CONF. ON COLLABORATIVE COMPUTING, 4., 2008, Orlando, USA. *Anais...* Orlando, FL, USA, 2008.
- [19] FERREIRA, J.; TAKAI, O.; PU, C. Integration of collaborative information system in internet applications using riverfish architecture. In: *IEEE Intl. Conf. on Collaborative Computing: Networking, Applications and Worksharing*. San Jose, USA: [s.n.], 2005. v. 1, p. 19–27.
- [20] HALLE, B. V. *Business Rules Applied: Building Better Systems Using the Business Rule Approach*. [S.l.]: Wiley, 2001.
- [21] FERREIRA, J.; TAKAI, O.; PU, C. Integration of business process with autonomous information systems: A case study in government services. In: *IEEE 7th Intl. Conf. on E-Commerce Technology*. [S.l.: s.n.], 2005. p. 471–478.
- [22] ELLIS C. A.; NUTT, G. J. Office information systems and computer science. *ACM Computing Surveys*, v. 12, n. 1, p. 27–60, 1980.
- [23] PETRI, C. *Kommunikation mit Automaten*. Tese (Doutorado) — University of Bonn, Germany, 1962. (In German).
- [24] MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of IEEE*, v. 77, n. 4, p. 541–580, 1989.
- [25] ZISMAN, M. *Representation, Specification and Automation of Office Procedures*. Tese (Doutorado) — Wharton School, University of Penn, 1977.
- [26] MAHLING, D. E.; CRAVEN, N.; W.B. From office automation to intelligent workflow systems. *IEEE Expert*, v. 10, p. 41–47, 1995.
- [27] R.J.DUFFIN. Topology of series-parallel networks. *J. Mathematical Analysis and Applications*, n. 10, p. 303–318, 1965.
- [28] VALDES, J. *Parsing Flowcharts and Series-Parallel Graphs*. [S.l.], 1978. STAN CS-78-682.
- [29] JENSEN, K. Coloured petri nets and the invariant method. *Theoretical Computer Science*, North-Holland, v. 14, p. 317–336, 1981.
- [30] JENSEN, K. *Coloured Petri Nets*. [S.l.]: Springer-Verlag, 1987. 248-299 p. (LNCS Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I, v. 254).
- [31] ELLIS, C.; NUTT, G. *Workflow: The Process Spectrum*. 1996. Disponível em: <citeseer.ist.psu.edu/ellis96workflow.html>.
- [32] MILNER, R. A calculus of communicating systems. *LNCS*, Springer-Verlag, n. 92, 1980.
- [33] MILNER, R. *Communication and Concurrency*. [S.l.]: Prentice Hall, 1989.
- [34] MOLLER, F. The importance of the left merge operator in process algebras. *LNCS*, v. 443, p. 752–764, 1990.
- [35] HOARE, C. *Communicating Sequential Processes*. [S.l.]: Prentice Hall, 1985.

- [36] GLABBEEK, R. van. Bounded nondeterminism and the approximation induction principle in process algebra. *LNCS 247, 4th Annual Symposium on Theoretical Aspects of Computer Science (STACS 87)*, Springer-Verlag, p. 336–347, 1987.
- [37] BAETEN, J. C. M.; WEIJLAND, W. P. *Process Algebra*. [S.l.]: Cambridge, 1991.
- [38] STOHR, E. A.; ZHAO, J. L. Workflow automation: Overview and research issues. *Information Systems Frontiers*, Kluwer Academic, the Netherlands, v. 3, p. 281–296, 2001.
- [39] WORKFLOW Management Coalition. 1993. Disponível em: <<http://www.wfmc.org>>. Acesso em: 20 out 2007.
- [40] AALST, W. van der et al. Workflow patterns. In: *Distributed and Parallel Databases*. [S.l.: s.n.], 2003. v. 14 (3), p. 5–51.
- [41] OIKAWA, M. K. et al. Genflow: Generic flow for integration, management and analysis of molecular biology data. *Genetics and Molecular Biology*, v. 27, n. 4, p. 691–695, 2004.
- [42] WESKE, M. Flexible Modeling and Execution of Workflow Activities. In: *Proc. of the 31st Annual Hawaii International Conference on System Sciences*. [S.l.: s.n.], 1998. p. 713–722.
- [43] GHALLAB, M.; NAU, D.; TRAVERSO, P. *Automated Planning: Theory and Practice*. San Francisco, CA, USA: Morgan-Kaufman, 2004.
- [44] WELD, D. S. An introduction to least commitment planning. *AI Magazine*, v. 15, n. 4, p. 27–61, 1994.
- [45] SIRIN, E. et al. HTN planning for web service composition using shop2. *J. Web Semantics*, v. 1, n. 4, p. 377–396, 2004.
- [46] SIRIN, E.; HENDLER, J.; PARSIA, B. Semi-automatic composition of web services using semantic descriptions. In: WEB SERVICES: MODELING, ARCHITECTURE AND INFRASTRUCTURE WORKSHOP. *WSMAI*. [S.l.], 2003. p. 17–24.
- [47] MEDJAHED, B.; BOUGUETTAYA, A.; ELMAGARMID, A. K. Composing web services on the semantic web. *VLDB Journal*, v. 12, n. 4, p. 333–351, 2003.
- [48] NAU, D. et al. Total-order planning with partially ordered subtasks. In: *17th Intl Conference on Artificial Intelligence*. [S.l.: s.n.], 2001. p. 425–430.
- [49] NAU, D. et al. Shop2: An HTN planning system. *Journal of Artificial Intelligence Research*, n. 20, p. 379–404, 2003.
- [50] FERREIRA, J. E. et al. Large Scale Order Processing through Navigation Plan Concept. In: *IEEE International Conference on Services Computing*. Chicago, USA: [s.n.], 2006.
- [51] BRAGHETTO, K. R. *Padrões de Fluxos de Processos em Banco de Dados Relacionais*. Dissertação (Mestrado) — Universidade de São Paulo, São Paulo, Brasil, Junho 2006. (em Português).
- [52] BOLOGNESI, T.; BRINKSMA, E. Introduction to the iso specification language lotos. *Computer Networks and ISDN Systems*, v. 14, p. 25–59, 1987.
- [53] LOGRIPO, L.; FACI, M.; HAJ-HUSSEIN, M. An introduction to lotos: Learning by examples. *Computer Networks and ISDN Systems*, v. 23, p. 325–342, 1992.
- [54] HOARE, C. *Communicating Sequential Processes*. Eletronic version, 2004. Disponível em: <<http://www.usingcsp.com/cspbook.pdf>>. Acesso em: 20 out 2007.
- [55] MILNER, R. *Communicating and Mobile Systems: the π -Calculus*. [S.l.]: Cambridge, 1999.

- [56] FOKKINK, W. *Introduction to Process Algebra*. [S.l.]: Springer, 2000. (Texts in Theoretical Computer Science).
- [57] DIESTEL, R. *Graph Theory (Electronic Edition)*. 3rd. ed. [S.l.]: Springer-Verlag, 2005.
- [58] VALDES, J.; TARJAN, R.; LAWLER, E. The recognition of series parallel digraphs. *SIAM J. Comput.*, n. 11, p. 298–313, 1982.
- [59] LOVÁSZ, L. Graph minor theory. *The American Mathematical Society*, v. 43, n. 1, p. 75–86, 2005.
- [60] SCHOENMAKERS, B. A new algorithm for the recognition of series parallel graphs. *CWI Report CS-R9504*, 1995.
- [61] B.S.PANDA. The forbidden subgraph characterization of directed vertex graphs. *Discrete Mathematics*, v. 196, p. 239–256, 1999.
- [62] TSAI, J.; LIANG, D.-R.; CHOU, H.-H. *On Recognition of And-Or Series-Parallel Digraphs*. [S.l.], 2000.
- [63] DODIN, B. Bounding the project completion time distribution in pert networks. *Operations Research*, v. 33, n. 4, p. 862–881, 1985.
- [64] ABDEL-WAHAB, H. M.; KAMEDA, T. Scheduling to minimize maximum cumulative cost subject to series-parallel precedence constraints. *Operations Research*, v. 26, n. 1, p. 141–158, 1978.
- [65] RINGER, L. J. A statistical theory for pert in which completion times of activities are interdependent. *Management Science*, v. 17, n. 11, p. 717–723, 1971.
- [66] GHOMI, S. F.; RABBANI, M. A new structural mechanism for reducibility of stochastic pert networks. *European Journal of Operational Research*, n. 145, p. 394–402, 2003.
- [67] BERGSTRA, J.; KLOP, J. *Fixed point semantics in process algebra*. Amsterdam, 1982. Technical Report IW 208.
- [68] BERGSTRA, J.; KLOP, J. Process algebra for synchronous communication. *Information and Control*, v. 60, p. 109–137, 1984.
- [69] BERGSTRA, J.; KLOP, J. The algebra of recursively defined processes and the algebra of regular processes. *ICALP*, p. 82–94, 1984.
- [70] BERGSTRA, J.; KLOP, J. Algebra of communicating processes with abstraction. *Theor. Computer Science*, n. 37, p. 77–121, 1985.
- [71] GLABBEEK, R. J. V. The linear time - branching time spectrum i. the semantics of concrete, sequential processes. In: *Handbook of Process Algebra*. [S.l.]: Elsevier, 2001. p. 3–99.
- [72] PELED, D. A. *Software Reliability Methods*. [S.l.]: Springer, 2001.
- [73] ESCRIG, D. de F.; RODRÍGUEZ, C. G. Process equivalences as global bisimulations. *Journal of Universal Computer Science*, v. 12, n. 11, p. 1521–1550, 2006.
- [74] ANDRIES, M. et al. Graph transformation for specification and programming. *Science of Computer Programming*, Elsevier North-Holland, v. 34, n. 1, p. 1–54, 1999.
- [75] HOFFMANN, B. From graph transformation to rule-based programming with diagrams. *LNCS*, Springer, v. 1779, p. 154–160, 2000.
- [76] KOZIOURA, V. Verification of random graph transformation systems. *Electronic Notes in Theoretical Computer Science*, Elsevier Sciences, v. 175, n. 4, p. 63–72, 2007.

- [77] GÖNCZY, L.; KOVÁČZ, M.; VARRÓ, D. Modeling and verification of reliable messaging by graph transformation systems. *Electronic Notes in Theoretical Computer Science*, Elsevier Sciences, v. 174, n. 4, p. 37–50, 2005.
- [78] ROSSELLÓ, F.; VALIENTE, G. Graph transformation in molecular biology. *LNCS*, v. 3393, p. 116–133, 2005.
- [79] NEWMAN, M. H. A. On theories with a combinatorial definition of “equivalence”. *Annals of Mathematics*, v. 43, p. 223–243, 1942.
- [80] ROSEN, B. Tree-manipulating systems and church-rosser theorems. *Journal of the ACM*, v. 20, p. 160–187, 1973.
- [81] HUET, G. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of ACM*, v. 27, n. 4, p. 797–821, 1980.
- [82] KLOP, J. W.; OOSTROM, V. van; RAAMSDONK, F. van. Combinatory reduction systems: Introduction and survey. *Theoretical Computer Science*, v. 121, n. 1–2, p. 279–308, 1993.
- [83] OHLEBUSCH, E. *Advanced Topics in Term Rewriting*. [S.l.]: Springer, 2002.
- [84] BAADER, F.; NIPKOW, T. *Term Rewriting and All That*. [S.l.]: Cambridge University Press, 1999.
- [85] BURMEISTER, P.; ROSSELLÓ, F.; VALIENTE, G. *Double-Pushout Hypergraph through Free Completions*. [S.l.], 1996.
- [86] LÖWE, M. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, Elsevier Science, v. 109, n. 1-2, p. 183–224, 1993.
- [87] CORRADINI, A. et al. *Algebraic Approaches to Graph Transformation, Part II: Single Pushout Approach and comparison with double pushout approach*. Corso Italia 40, 56125 Pisa, Italy, 1997.
- [88] HABEL, A.; MÜLLER, J.; PLUMP, D. Double-pushout approach with injective matching. *LNCS* 1764, p. 103–117, 2000.
- [89] BARR, M.; WELLS, C. *Category Theory for Computing Science*. [S.l.]: Prentice Hall, 1990.
- [90] MACLANE, S. *Categories for the Working Mathematician*. 2nd. ed. [S.l.]: Springer, 1998. (Graduate Texts in Mathematics).
- [91] EHRIG, H.; KREOWSKI, H.-J. Parallelism of manipulations in multidimensional information structures. In: MATHEMATICAL FOUNDATIONS OF COMPUTER SCIENCE. *LNCS*. [S.l.]: Springer, 1976. v. 45, p. 284–293.
- [92] VARRÓ, D. et al. Termination analysis of model transformations by petri nets. *LNCS*, v. 4178, p. 260–274, 2006.
- [93] EHRIG, H. et al. Termination criteria for model transformation. In: FASE 2005: INTERNATIONAL CONFERENCE ON FUNDAMENTAL APPROACHES TO SOFTWARE ENGINEERING. *LNCS*. Edinburgh, UK: Springer, 2005. v. 3442, p. 49–63.
- [94] PLUMP, D. Termination of graph rewriting is undecidable. *Fundamenta Informaticae*, v. 33, n. 2, p. 201–209, 1998.
- [95] LEVENDOVSKY, T.; PRANGE, U.; EHRIG, H. Termination criteria for DPO transformations with injective matches. *Electronic Notes in Theoretical Computer Science*, v. 175, p. 87–100, 2007.
- [96] BRUGGINK, H. J. S. Towards a systematic method for proving termination of graph transformation systems. *Electronic Notes in Theoretical Computer Science*, v. 213, p. 23–38, 2008.

-
- [97] MAHANTLI, A.; BAGCHI, A. And/or graph heuristic search methods. *Journal of the ACM*, v. 32, n. 1, p. 28–51, 1985.
 - [98] DESMEDT, Y.; WANG, Y. Maximum flows and critical vertices in and/or graphs. *LNCS (COCON 2002)*, n. 2387, p. 238–248, 2002.
 - [99] PLUMP, D. *Evaluation of Functional Expressions by Hypergraph Rewriting*. Tese (Doutorado) — Universität Bremen, 1993.
 - [100] POL, J. C. van de. *Just-in-time: on Strategy Annotations*. [S.l.], 2001.
 - [101] SADIQ, W.; ORLOWSKA, M. E. Applying graph reduction techniques for identifying structural conflicts in process models. *LNCS 1626 (CaiSE'99)*, p. 195–209, 1999.
 - [102] CORMEN, T. H. et al. *Introduction to Algorithms*. [S.l.]: MIT Press, 2001.
 - [103] KORENBLIT, M.; LEVIT, V. E. On algebraic expressions of series-parallel and fibonacci graphs. *LNCS 2731 (DMTCS)*, p. 215–224, 2003.
 - [104] ACETO, L.; FOKKING, W. J. The quest for equational axiomatizations of parallel composition: Status and open problems. *BRICS - Basic Research in Computer Science*, 2005.
 - [105] GOLUMBIC, M.; PERL, Y. Generalized fibonacci maximum path graphs. *Discrete Mathematics*, v. 28, p. 237–245, 1979.