COPPE/UFRJ

APLICAÇÕES DE LÓGICAS MODAIS A TEORIA DE GRAFOS E SISTEMAS CONCORRENTES

Luis Menasché Schechter

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Mario Roberto Folhadela Benevides

Rio de Janeiro

Março de 2010

APLICAÇÕES DE LÓGICAS MODAIS A TEORIA DE GRAFOS E SISTEMAS CONCORRENTES

Luis Menasché Schechter

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Mario Roberto Folhadela Benevides, Ph.D.

Prof. Valmir Carneiro Barbosa, Ph.D.

Prof. Severino Collier Coutinho, Ph.D.

Profa. Sheila Regina Murgel Veloso, D.Sc.

Prof. Edward Hermann Haeusler, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2010

Schechter, Luis Menasché

Aplicações de Lógicas Modais a Teoria de Grafos e Sistemas Concorrentes/Luis Menasché Schechter. – Rio de Janeiro: UFRJ/COPPE, 2010.

XVI, 249 p.: il.; 29, 7cm.

Orientador: Mario Roberto Folhadela Benevides

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2010.

Referências Bibliográficas: p. 125 – 131.

1. Lógicas Modais. 2. Teoria de Grafos. 3. Sistemas Concorrentes. 4. Verificação de Modelos. 5. Sistemas Axiomáticos. I. Benevides, Mario Roberto Folhadela. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Esta tese é dedicada a meus pais

Rosa Menasché Schechter

e Marcos Schechter

e aos professores

Mario Benevides

e Severino Collier Coutinho.

Agradecimentos

Gostaria de agradecer em primeiro lugar ao professor Mario Benevides. O professor Mario é meu orientador nesta Tese de Doutorado e já nos conhecemos de longa data. Realizei meu primeiro curso com ele no segundo ano da minha graduação, 2002. A partir daí, durante a graduação, mestrado e doutorado, somaram-se um total de sete cursos com ele. Ter estudado com ele durante estes anos e ter trabalhado com ele durante o doutorado foram, sem dúvida, ótimas experiências.

Outro professor a quem eu também devo grandes agradecimentos é o professor Severino Collier Coutinho. O professor Collier foi meu orientador no Projeto Final de Curso, em um projeto de Iniciação Científica e em minha Dissertação de Mestrado e foi meu professor em seis disciplinas dos meus cursos de graduação e mestrado. Ele sempre foi um professor muito dedicado e sério, com quem também só tive ótimas experiências.

Gostaria de destacar também o apoio que recebi de diversos professores que conheci durante meus cursos de graduação e pós-graduação e em diversos congressos em que estive para apresentar trabalhos. Agradeço a Adriano Cruz, Ageu Pacheco, Andreas Herzig, Celina de Figueiredo, Dov Gabbay, Edward Hermann Haeusler, Elaine Pimentel, Fairouz Kamareddine, Gerson Zaverucha, João Marcos, Johan van Benthem, Luis Fariñas del Cerro, Marcelo Finger, Márcia Cerioli, Mauricio Ayala Rincón, Mauro Rincon, Miguel Jonathan, Odinaldo Rodrigues, Paulo Veloso, Petrucio Viana, Renata de Freitas, Ruy de Queiroz, Sheila Veloso, Sulamita Klein, Susana Scheimberg de Makler, Valmir Barbosa, Wilfrid Hodges e Yde Venema. Muito obrigado a todos.

Agradeço muito a Mariana Sereno por todo carinho, apoio e motivação durante a fase final da elaboração desta tese.

Não poderia também deixar de citar aqui os ótimos amigos que fiz durante a

graduação e pós-graduação na UFRJ, assim como os amigos de tempos anteriores. Graças a eles, estes anos universitários tornaram-se muito mais agradáveis. Em especial, agradeço a Carina Lopes, Carlos Reis, Dan Gandelman, Elias Bareinboim, Fernanda Wanderley, Hugo Cesar Carneiro, Isabella Almeida, João Menezes, Marcelo Alvim, Marcos da Silva Ferreira, Pedro Rocha, Ramon Diacovo, Sergio Warszawski, Thiago Siqueira Batista e Tiago Mota.

Agradeço também a Andre Nudelman, Dario Bialer, Oren Boljover, Sergio Margulies e todos os amigos da ARI pelo apoio constante.

Por último, devo agradecer também ao CNPq pela bolsa concedida a mim para o curso de doutorado e a elaboração desta tese.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

APLICAÇÕES DE LÓGICAS MODAIS A TEORIA DE GRAFOS E SISTEMAS CONCORRENTES

Luis Menasché Schechter

Março/2010

Orientador: Mario Roberto Folhadela Benevides
Programa: Engenharia de Sistemas e Computação

Neste trabalho, desenvolvemos novas aplicações de lógicas modais em duas áreas. Primeiramente, analisamos como descrever e verificar de maneira eficiente conectividade, aciclicidade e as propriedades hamiltoniana e euleriana utilizando lógicas modais. Para esta tarefa, utilizamos lógicas híbridas e lógicas modais graduadas. Ainda no estudo de propriedades de grafos, também determinamos uma condição necessária e suficiente para um grafo ser um produto cartesiano de grafos, obtendo um avanço em relação a um resultado anterior da literatura, que fornecia uma condição necessária, mas não suficiente. Além disto, utilizamos esta caracterização para obter axiomatizações corretas e completas para vários produtos de lógicas modais de dimensões arbitrárias. Este é um segundo avanço em relação a resultados anteriores da literatura, onde a maior parte dos sistemas axiomáticos corretos e completos apresentados se restringe a produtos de um par de lógicas modais.

Posteriormente, consideramos extensões da Lógica Dinâmica Proposicional (PDL) com operadores para a descrição de comportamentos concorrentes. Nosso objetivo é construir lógicas dinâmicas que sejam apropriadas para a descrição e verificação de propriedades de sistemas comunicantes concorrentes, de maneira análoga ao uso de PDL para o caso sequencial. Primeiramente, adicionamos a PDL o operador paralelo de CCS e, posteriormente, também adicionamos a possibilidade de passagem de nomes presente no π-Cálculo. Diferentemente de outras lógicas dinâmicas para sistemas concorrentes apresentadas anteriormente na literatura, nossas lógicas possuem semânticas de Kripke simples, axiomatizações completas e a propriedade do modelo finito.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

APPLICATIONS OF MODAL LOGICS TO GRAPH THEORY AND CONCURRENT SYSTEMS

Luis Menasché Schechter

March/2010

Advisor: Mario Roberto Folhadela Benevides
Department: Systems and Computer Engineering

In the present work, we develop new applications of modal logics in two areas. First, we analyze how to describe and efficiently verify connectivity, acyclicity and the Hamiltonian and Eulerian properties using modal logics. For this task, we use hybrid logics and graded modal logics. Still on the study of graph properties, we also determine a necessary and sufficient condition for a graph to be a cartesian product of graphs, improving on a previous result from the literature, that stated a necessary, but not sufficient, condition. Besides that, we use this characterization to build sound and complete axiomatic systems for many products of modal logics of arbitrary dimensions. This also improves previous results from the literature, where most of the sound and complete axiomatic systems that are presented are for products of a pair of modal logics.

In the second topic, we consider extensions of Propositional Dynamic Logic (PDL) with operators that describe concurrent behavior. Our goal is to build dynamic logics that are suitable for the description and verification of properties of communicating concurrent systems, in a similar way as PDL is used for the sequential case. First, we add to PDL the parallel operator from CCS and then we also add the possibility of name passing from the π-Calculus. Unlike previous dynamic logics for concurrency from the literature, our logics have simple Kripke semantics, complete axiomatizations and the finite model property.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

"Eu quero compartilhar algo com você. As três frases que vão te ajudar durante a vida. Número um: me dê cobertura. Número dois: oh, boa ideia, chefe! Número três: isto já estava assim quando eu cheguei." - Homer Simpson

Este capítulo inicial tem dois objetivos principais: explanar em linhas gerais os temas e objetivos desta tese e traçar um roteiro de como estes temas serão abordados nos capítulos seguintes. O leitor não deve sentir-se incomodado caso não compreenda totalmente alguma parte da terminologia utilizada neste capítulo, pois todos os conceitos serão devidamente explicados mais adiante nesta tese.

1.1 Objetivos

O principal objetivo desta tese é estudar aplicações de lógicas modais para computação. Em particular, a tese se foca em duas aplicações. Primeiramente, consideramos como utilizar lógicas modais para descrever e verificar algumas propriedades da teoria de grafos. Posteriormente, desenvolvemos lógicas dinâmicas para a descrição e verificação de propriedades de sistemas concorrentes.

Grafos estão entre as estruturas mais utilizadas e mais estudadas em Ciência da Computação [1]. Nesta disciplina, muitos conceitos importantes admitem uma representação através de grafos e, em algumas ocasiões, grafos estão presentes no próprio núcleo do modelo de computação utilizado. Isto acontece, por exemplo, na área de sistemas distribuídos [2, 3], onde o modelo subjacente de computação é construído sobre um grafo. Além deste papel central, grafos também são importantes

em sistemas distribuídos como ferramentas para a descrição de problemas de distribuição de recursos, escalonamento de tarefas, detecção de *deadlocks*, entre outros. O caso de sistemas distribuídos é particularmente atraente porque ilustra bem dois níveis diferentes em que propriedades de grafos são descritas. Um é o nível local, que engloba propriedades que são satisfeitas em vértices ou vizinhanças de vértices. O outro nível é o global e consiste de propriedades que são satisfeitas no grafo como um todo, tais como aciclicidade e conectividade.

A teoria de grafos fornece muitas ferramentas para a descrição de tais problemas e apresenta muitos algoritmos eficientes para resolvê-los. Entretanto, existe uma distinção importante entre os dois lados desta questão. No lado da "descrição", grafos fornecem um grande nível de generalidade, permitindo a descrição de problemas muito diferentes no mesmo arcabouço[1] simples. Porém no lado da "solução", cada problema de grafos precisa ser resolvido e cada propriedade de grafos precisa ser testada com um método específico que em geral não pode ser generalizado para outros problemas ou propriedades.

Um arcabouço lógico, por outro lado, pode prover este nível de generalidade. Nesta tese, analisamos como podemos expressar e verificar de maneira eficiente algumas propriedades globais de grafos utilizando lógicas modais. Isto envolve duas questões: a primeira é determinar se cada uma das linguagens modais consideradas possui poder expressivo suficiente para descrever as propriedades de grafos em que estamos interessados; a segunda é quão complexo (computacionalmente) é utilizar estas lógicas para realmente testar se um dado grafo possui uma propriedade desejada.

Ainda no tópico da descrição e verificação de propriedades de grafos, buscamos descrever uma condição necessária e suficiente para que um grafo seja isomorfo ao produto cartesiano de grafos não triviais e verificamos se tal condição pode ser expressa na lógica modal básica ou na lógica híbrida. Determinamos então qual é a complexidade computacional para verificar, utilizando esta condição necessária e suficiente descrita previamente, se um dado grafo finito e conexo é um produto. Finalmente, utilizamos esta caracterização de produtos para descrever sistemas axiomáticos corretos e completos para uma grande classe de produtos de

[1] framework, em inglês

lógicas modais.

Em [4] e [5], três propriedades que são satisfeitas em grafos que são produtos são apresentadas: *comutatividade à esquerda*, *comutatividade à direita* e a *propriedade de Church-Rosser*. No entanto, apesar destas propriedades, em conjunto com a *propriedade de Church-Rosser reversa*, serem necessárias para que um grafo seja um produto, elas não são suficientes. Existem grafos que satisfazem estas quatro propriedades, mas não podem ser decompostos como produto de outros grafos. Nós introduzimos uma nova propriedade, chamada intransitividade, que, em conjunto com as anteriores, forma uma condição necessária e suficiente para que um grafo enumerável e conexo seja um produto.

Nós mostramos que intransitividade não pode ser descrita por uma fórmula na lógica modal básica. De fato, nenhuma condição que seja necessária e suficiente para um grafo ser um produto pode ser descrita por uma fórmula na lógica modal básica. A partir disto, estendemos a linguagem utilizada para uma linguagem híbrida e construímos uma fórmula nesta linguagem que descreve intransitividade.

Produtos de grafos surgem naturalmente como uma possível extensão da semântica de Kripke tradicional para lógicas modais multidimensionais. [4] apresenta uma ampla discussão de lógicas modais multidimensionais e fornece muitos exemplos de produtos de lógicas modais, onde a semântica é construída utilizando-se produtos de grafos. A maior parte dos sistemas axiomáticos corretos e completos para produtos de lógicas modais apresentados na literatura são para produtos de um par de lógicas modais, enquanto que somos capazes, utilizando lógica híbrida, de apresentar axiomatizações corretas e completas para muitos produtos de lógicas modais de dimensões arbitrárias.

No segundo tópico, consideramos extensões da Lógica Dinâmica Proposicional (PDL) com operadores para a descrição de comportamentos concorrentes. O Cálculo de Sistemas Comunicantes (CCS) e o π-Cálculo são álgebras de processos bem conhecidas para a especificação de sistemas concorrentes. Nós construímos lógicas dinâmicas proposicionais em que os programas são descritos, primeiramente, em uma linguagem baseada em CCS e, posteriormente, em uma linguagem baseada no π-Cálculo. O objetivo disto é criar lógicas dinâmicas em que seja simples descrever e verificar propriedades de programas e sistemas concorrentes, comunicantes e não

determinísticos, de uma maneira análoga à que PDL é usada no caso sequencial.

Estas lógicas são relacionadas a outras lógicas para sistemas concorrentes existentes na literatura, como PDL Concorrente (CPDL) [6], CPDL com canais [7], a lógica desenvolvida na tese [8] e PDL com intercalamento[2] [9]. No entanto, as lógicas desenvolvidas nesta tese apresentam vantagens sobre estas outras lógicas, que serão devidamente discutidas posteriormente.

1.2 Roteiro da Tese

No Capítulo 2, serão apresentados alguns conceitos básicos de *lógica modal* que serão importantes no desenvolvimento da tese. Várias lógicas modais são apresentadas neste capítulo, desde a *lógica modal básica* até lógicas com maior poder expressivo com a *lógica híbrida*, a *lógica híbrida com quantificador local*, a *lógica modal graduada*, as *lógicas temporais* CTL e CTL* e a *lógica dinâmica proposicional* (PDL). Todas estas lógicas são fundamentais no desenvolvimento da tese.

No Capítulo 3, serão apresentados conceitos básicos da *teoria de grafos*. Primeiramente, apresentamos as definições básicas a respeito de grafos. Depois, nos detemos em dois tópicos específicos. Em primeiro lugar, discutimos quatro *propriedades globais de grafos* que são muito importantes e muito utilizadas em computação: *conectividade*, *aciclicidade*, a *propriedade hamiltoniana* e a *propriedade euleriana*. Finalmente, na última parte do capítulo, o conceito de *produto cartesiano de grafos* é apresentado.

No Capítulo 4, serão apresentados os conceitos básicos de *álgebras de processos*. Em particular, apresentamos duas álgebras de processos que serão utilizadas no desenvolvimento da tese: o *Cálculo de Sistemas Comunicantes* (CCS) e o π-*Cálculo*. O π-Cálculo é uma extensão de CCS em que os processos podem transmitir nomes. Desta forma, o π-Cálculo consegue lidar com sistemas que possuem algum tipo de *mobilidade*.

No Capítulo 5, analisamos o problema de como descrever e verificar de maneira eficiente as quatro propriedades globais de grafos apresentadas no capítulo 3 utilizando lógicas modais. Para esta tarefa, consideramos as vantagens e desvantagens

[2]interleaving, em inglês

da lógica modal básica, da lógica híbrida, da lógica híbrida com quantificador local, da lógica modal graduada e de uma versão híbrida da lógica temporal CTL*.

No Capítulo 6, determinamos uma condição necessária e suficiente para um grafo ser um produto cartesiano de grafos não triviais, obtendo um avanço em relação a um resultado anterior da literatura, que fornecia uma condição necessária, mas não suficiente. Para isto, descrevemos uma nova propriedade que produtos de grafos devem possuir, chamada intransitividade. Determinamos então qual é a complexidade computacional para verificar se um dado grafo finito e conexo é um produto. Finalmente, utilizamos esta caracterização de produtos para descrever sistemas axiomáticos corretos e completos para uma grande classe de produtos de lógicas modais de dimensões arbitrárias.

No Capítulo 7, consideramos extensões da Lógica Dinâmica Proposicional (PDL) com operadores para a descrição de comportamentos concorrentes. Nosso objetivo é construir lógicas dinâmicas que sejam apropriadas para a descrição e verificação de propriedades de sistemas comunicantes concorrentes, de maneira análoga ao uso de PDL para o caso sequencial. Primeiramente, adicionamos a PDL o operador paralelo de CCS e, posteriormente, também adicionamos a possibilidade de passagem de nomes presente no π-Cálculo.

No Capítulo 8, analisamos os resultados, apresentamos nossas conclusões e delineamos alguns possíveis trabalhos futuros.

Capítulo 2

Conceitos Básicos de Lógica Modal

"Fatos não significam nada. Você pode usar fatos para provar qualquer coisa que não seja nem remotamente verdade!" - Homer Simpson

Neste capítulo, apresentamos diversas lógicas modais que serão utilizadas no decorrer da tese, desde a lógica modal mais básica até lógicas modais com maior poder expressivo. Não serão apresentados todos os detalhes de cada uma destas lógicas, apenas os conceitos e resultados que são de alguma forma importantes para o desenvolvimento da tese.

As lógicas modais permitem exprimir de modo explícito as noções de *necessidade* e *possibilidade*. Elas permitem exprimir que uma dada asserção é necessariamente verdadeira, isto é, não se concebe nenhuma situação em que possa ser falsa, e que uma dada asserção é possivelmente verdadeira, isto é, concebe-se que possam existir situações em que ela é verdadeira podendo, no entanto, existirem também situações em que ela é falsa.

2.1 Lógica Modal Básica

Nesta seção, apresentamos a lógica modal básica, que busca expressar noções de *possibilidade* e *necessidade*.

2.1.1 Linguagem e Semântica

Primeiramente, apresentamos a linguagem e a semântica da lógica modal básica. Em termos gerais, a principal diferença desta lógica para a lógica proposicional con-

siste na presença de dois operadores unários que relativizam o conceito de verdade de uma fórmula ϕ. Estes operadores são conhecidos como *modalidades* e são denotados por $\Diamond$ e $\Box$. A fórmula $\Diamond\phi$ denota que ϕ é *possível* e a fórmula $\Box\phi$ denota que ϕ é *necessário*. No decorrer desta seção, a semântica da lógica será apresentada de maneira formal.

Definição 2.1. *A linguagem da lógica modal básica consiste de um conjunto enumerável Φ de símbolos proposicionais, os operadores booleanos $\neg$, $\wedge$, $\vee$ e $\rightarrow$, as constantes $\top$ e $\bot$ e dois operadores modais (ou modalidades): $\Diamond$ e $\Box$. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid \top \mid \bot \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \Diamond\varphi \mid \Box\varphi,$$

onde $p \in \Phi$.

As fórmulas em lógica modal são avaliadas de acordo com a semântica conhecida como *Semântica de Kripke* ou *Semântica dos Mundos Possíveis*. Esta semântica se baseia em estruturas matemáticas conhecidas como *Estruturas de Kripke* e *Modelos de Kripke*.

Definição 2.2. *Uma* estrutura de Kripke[1] *(ou apenas* estrutura*) é um par $\mathcal{F} = (W, R)$, onde W é um conjunto não vazio de* mundos *(ou* estados*, ou* vértices*) e R é uma relação binária sobre W, isto é, $R \subseteq W \times W$.*

Definição 2.3. *Um* modelo de Kripke *(ou apenas* modelo*) é um par $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, onde $\mathcal{F}$ é uma estrutura e $\mathbf{V}$ é uma função de valoração que mapeia símbolos proposicionais em subconjuntos de W, isto é, $\mathbf{V} : \Phi \mapsto \mathcal{P}(W)$.*

A partir destes dois conceitos, podemos definir a noção semântica de *satisfazibilidade* de uma fórmula da lógica.

Definição 2.4. *Seja $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ um modelo. A noção de* satisfazibilidade *de uma fórmula φ em um modelo $\mathcal{M}$ em um mundo v, denotada por $\mathcal{M}, v \Vdash \varphi$, pode ser definida indutivamente da seguinte maneira:*

1. *$\mathcal{M}, v \Vdash p$ sse $v \in \mathbf{V}(p)$;*

[1] Kripke frame, em inglês

2. $\mathcal{M}, v \Vdash \top$ *sempre;*

3. $\mathcal{M}, v \Vdash \bot$ *nunca;*

4. $\mathcal{M}, v \Vdash \neg\varphi$ *sse* $\mathcal{M}, v \nVdash \varphi$*;*

5. $\mathcal{M}, v \Vdash \varphi_1 \wedge \varphi_2$ *sse* $\mathcal{M}, v \Vdash \varphi_1$ *e* $\mathcal{M}, v \Vdash \varphi_2$*;*

6. $\mathcal{M}, v \Vdash \varphi_1 \vee \varphi_2$ *sse* $\mathcal{M}, v \Vdash \varphi_1$ *ou* $\mathcal{M}, v \Vdash \varphi_2$*;*

7. $\mathcal{M}, v \Vdash \varphi_1 \rightarrow \varphi_2$ *sse se* $\mathcal{M}, v \Vdash \varphi_1$*, então* $\mathcal{M}, v \Vdash \varphi_2$*;*

8. $\mathcal{M}, v \Vdash \Diamond\varphi$ *sse há* $w \in W$ *tal que* vRw *e* $\mathcal{M}, w \Vdash \varphi$*;*

9. $\mathcal{M}, v \Vdash \Box\varphi$ *sse para todo* $w \in W$ *tal que* vRw*,* $\mathcal{M}, w \Vdash \varphi$*.*

A fórmula $\Diamond\varphi$ é satisfeita em um mundo v se existe um mundo w acessível a partir de v através da relação R que satisfaz a fórmula φ. A fórmula $\Box\varphi$ é satisfeita em um mundo v se a fórmula φ é satisfeita em todo mundo w que é acessível a partir de v através da relação R.

A partir da noção de satisfazibilidade acima, podemos observar as seguintes equivalências semânticas: $\varphi \equiv \neg\neg\varphi$, $\bot \equiv \neg\top$, $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv \neg(\varphi_1 \wedge \neg\varphi_2)$ e $\Box\varphi \equiv \neg\Diamond\neg\varphi$. Portanto, em todas as outras lógicas apresentadas nesta tese, iremos considerar como operadores primitivos apenas $\top, \neg, \wedge$ e $\Diamond$, utilizando $\bot, \vee, \rightarrow$ e $\Box$ como operadores derivados a partir destas equivalências semânticas.

Exemplo 2.5. *Seja* $\mathcal{M}$ *o modelo mostrado (sem sua valoração) na figura 2.1. Para ilustrar o uso da lógica, podemos ver que as seguinte fórmulas são satisfeitas no mundo* w *em* $\mathcal{M}$*, supondo que* φ *é satisfeita no mundo* v*:* $\mathcal{M}, w \Vdash \Diamond\varphi$ *e* $\mathcal{M}, w \Vdash \Diamond\Diamond\Diamond\varphi$*.*

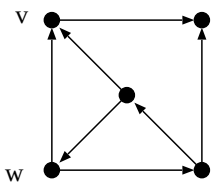

Figura 2.1: Modelo $\mathcal{M}$, onde a fórmula φ é satisfeita no mundo v.

Se $\mathcal{M}, v \Vdash \varphi$ para todo mundo v no modelo $\mathcal{M}$, dizemos que φ é *globalmente satisfeita* em $\mathcal{M}$, notação $\mathcal{M} \Vdash \varphi$. Se φ é globalmente satisfeita em todos os modelos $\mathcal{M}$ de uma estrutura $\mathcal{F}$, dizemos que φ é *válida* em $\mathcal{F}$, notação $\mathcal{F} \Vdash \varphi$. Finalmente, se φ é válida em todas as estruturas $\mathcal{F}$, dizemos que φ é válida, notação $\Vdash \varphi$. Duas fórmulas φ e ψ são *semanticamente equivalentes* se $\Vdash \varphi \leftrightarrow \psi$.

2.1.2 Complexidade Computacional

A complexidade da lógica pode ser medida a partir da complexidade de uma série de problemas computacionais relacionados a verificação de fórmulas da lógica. Entre os problemas computacionais mais importantes, temos o problema da *satisfazibilidade*, o problema da *validade* e o problema da *verificação de modelo*.

Definição 2.6. *O problema da* satisfazibilidade *consiste de, dada uma fórmula ϕ, determinar se existe um modelo $\mathcal{M}$ e um mundo v em $\mathcal{M}$ tais que $\mathcal{M}, v \Vdash \phi$.*

Definição 2.7. *O problema da* validade *consiste de, dada uma fórmula ϕ, determinar se $\mathcal{F} \Vdash \phi$, para todas as estruturas $\mathcal{F}$.*

O problema da satisfazibilidade e o problema da validade são duais um do outro uma vez que ϕ é válida se e somente se $\neg\phi$ não é satisfazível.

Definição 2.8. *O problema da* verificação de modelo *consiste de, dada uma fórmula ϕ e um modelo finito (com número finito de mundos) $\mathcal{M} = (W, R, \mathbf{V})$, determinar o conjunto $S_{\mathcal{M}}(\phi) = \{v \in W : \mathcal{M}, v \Vdash \phi\}$.*

Definição 2.9. *Nós definimos a comprimento de uma fórmula φ, denotado por $|\varphi|$, indutivamente da seguinte forma: $|p| = |\top| = |\bot| = 1$, $|\neg\phi| = |\Diamond\phi| = |\Box\phi| = 1 + |\phi|$ e $|\phi_1 \wedge \phi_2| = |\phi_1 \vee \phi_2| = |\phi_1 \rightarrow \phi_2| = 1 + |\phi_1| + |\phi_2|$. Nas outras lógicas apresentadas nesta tese, regras análogas se aplicam aos novos operadores.*

Definição 2.10. *Seja $\mathcal{M} = (W, R, \mathbf{V})$ um modelo. Seja $|W|$ o número de mundos em W e $|R|$ o número de pares em R. Definimos o tamanho do modelo (ou da estrutura) como $|W| + |R|$.*

Teorema 2.11 ([10])**.** *O problema da satisfazibilidade e o problema da validade para a lógica modal básica são PESPAÇO-Completos no comprimento da fórmula.*

Teorema 2.12 ([11])**.** *O problema da verificação de modelo para a lógica modal básica é polinomial (linear) no produto do tamanho do modelo e do comprimento da fórmula.*

2.1.3 Limites de Expressividade

Os limites do poder expressivo da lógica modal básica são bem conhecidos. Existem uma série de resultados que estabelecem que estruturas de Kripke que são "similares" de diversas maneiras validam as mesmas fórmulas. Podemos então utilizar estes resultados para provar que uma certa propriedade *não pode* ser expressa por nenhuma fórmula da lógica modal básica. Para isto, tomamos duas estruturas que são "similares" e mostramos que em uma a propriedade desejada é satisfeita, enquanto que na outra ela não é. Nós apresentamos dois destes resultados de "similaridade" (mais detalhes sobre eles, assim como outros resultados relacionados, podem ser encontrados em [10]).

Definição 2.13. *Sejam $\mathcal{F} = (W, R)$ e $\mathcal{F}' = (W', R')$ duas estruturas. Uma função $f : W \to W'$ é um* morfismo limitado[2] *de $\mathcal{F}$ para $\mathcal{F}'$ se ela satisfaz as seguintes condições:*

1. *f é um homomorfismo com respeito a R (se wRv, então $f(w)R'f(v)$);*

2. *se $f(w)R'v'$, então há v tal que wRv e $f(v) = v'$.*

Se existe um morfismo limitado sobrejetivo de $\mathcal{F}$ para $\mathcal{F}'$, então dizemos que $\mathcal{F}'$ é uma *imagem mórfica limitada*[3] de $\mathcal{F}$ e utilizamos a notação $\mathcal{F} \Rightarrow \mathcal{F}'$.

Definição 2.14. *Sejam $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ e $\mathcal{M}' = (\mathcal{F}', \mathbf{V}')$ dois modelos. Uma função $f : W \to W'$ é um* morfismo limitado *de $\mathcal{M}$ para $\mathcal{M}'$ se ela satisfaz as seguintes condições:*

1. *f é um morfismo limitado de $\mathcal{F}$ para $\mathcal{F}'$;*

2. *w e $f(w)$ satisfazem os mesmo símbolos proposicionais.*

[2] bounded morphism, em inglês

[3] bounded morphic image, em inglês

Se existe um morfismo limitado sobrejetivo de $\mathcal{M}$ para $\mathcal{M}'$, então dizemos que $\mathcal{M}'$ é uma *imagem mórfica limitada* de $\mathcal{M}$ e utilizamos a notação $\mathcal{M} \Rightarrow \mathcal{M}'$.

Outras definições importantes dizem respeito a coleções de estruturas disjuntas e coleções de modelos disjuntos. Dizemos que duas estruturas $\mathcal{F}_1 = (W_1, R_1)$ e $\mathcal{F}_2 = (W_2, R_2)$ são *estruturas disjuntas* se e somente se $W_1 \cap W_2 = \emptyset$ e dizemos que dois modelos $\mathcal{M}_1 = (\mathcal{F}_1, \mathbf{V}_1)$ e $\mathcal{M}_2 = (\mathcal{F}_2, \mathbf{V}_2)$ são *modelos disjuntos* se e somente se $\mathcal{F}_1$ e $\mathcal{F}_2$ são estruturas disjuntas.

Definição 2.15. *Seja $\mathcal{F}_i = (W_i, R_i)$ uma coleção (finita ou não) de estruturas disjuntas. Sua* união disjunta *é a estrutura $\biguplus \mathcal{F}_i = (W, R)$, onde $W = \bigcup_i W_i$ e $R = \bigcup_i R_i$.*

Definição 2.16. *Seja $\mathcal{M}_i = (\mathcal{F}_i, \mathbf{V}_i)$ uma coleção (finita ou não) de modelos disjuntos. Sua* união disjunta *é o modelo $\biguplus \mathcal{M}_i = (\mathcal{F}, \mathbf{V})$, onde $\mathcal{F}$ é a união disjunta das estruturas $\mathcal{F}_i$ e, para cada símbolo proposicional p, $\mathbf{V}(p) = \bigcup_i \mathbf{V}_i(p)$.*

Teorema 2.17 ([10])**.** *Sejam $\mathcal{M} = (W, R, \mathbf{V})$ e $\mathcal{M}' = (W', R', \mathbf{V}')$ dois modelos tais que $\mathcal{M} \Rightarrow \mathcal{M}'$. Então, $\mathcal{M}, w \Vdash \phi$ se e somente se $\mathcal{M}', f(w) \Vdash \phi$.*

Corolário 2.18 ([10])**.** *Sejam $\mathcal{F} = (W, R)$ e $\mathcal{F}' = (W', R')$ duas estruturas tais que $\mathcal{F} \Rightarrow \mathcal{F}'$. Se $\mathcal{F} \Vdash \phi$, então $\mathcal{F}' \Vdash \phi$.*

Teorema 2.19 ([10])**.** *Seja $\mathcal{M}_i = (W_i, R_i, \mathbf{V}_i)$ uma coleção (finita ou não) de modelos disjuntos e $\biguplus \mathcal{M}_i = (W, R, \mathbf{V})$ sua união disjunta. Então, $\mathcal{M}_i, w \Vdash \phi$ se e somente se $\biguplus \mathcal{M}_i, w \Vdash \phi$.*

Corolário 2.20 ([10])**.** *Seja $\mathcal{F}_i = (W_i, R_i)$ uma coleção (finita ou não) de estruturas disjuntas e $\biguplus \mathcal{F}_i = (W, R)$ sua união disjunta. Se $\mathcal{F}_i \Vdash \phi$ para todo i, então $\biguplus \mathcal{F}_i \Vdash \phi$.*

Como exemplo da aplicação destes resultados, mostramos que a propriedade de uma estrutura ser finita não pode ser expressa na lógica modal básica. Estes resultados serão também utilizados diversas vezes nos próximos capítulos.

Teorema 2.21. *A classe das estruturas finitas não pode ser definida na lógica modal básica.*

Demonstração. A união disjunta de uma coleção infinita de estruturas finitas disjuntas não é finita. Pelo corolário 2.20, como esta propriedade não é preservada por uniões disjuntas, ela não pode ser definida na lógica modal básica. □

2.1.4 Axiomatização

O conjunto de fórmulas válidas da lógica modal básica possui uma axiomatização simples. Consideramos o seguinte conjunto de axiomas e regras, onde p e q são símbolos proposicionais e φ e ψ são fórmulas. Este sistema axiomático é conhecido como **K**, devido ao nome do axioma que lida com as modalidades da lógica.

(LP) Tautologias da Lógica Proposicional

(K) $\vdash \Box(p \to q) \to (\Box p \to \Box q)$

(Du) $\vdash \Box p \leftrightarrow \neg\Diamond\neg p$

(MP) Se $\vdash \varphi$ e $\vdash \varphi \to \psi$, então $\vdash \psi$

(Gen) Se $\vdash \varphi$, então $\vdash \Box\varphi$

(Sub) Se $\vdash \varphi$, então $\vdash \varphi^\sigma$, onde σ substitui uniformemente símbolos proposicionais por fórmulas

Definição 2.22. *Dizemos que uma fórmula ϕ é um* teorema *do sistema axiomático, denotado por $\vdash \phi$, se é possível deduzir ϕ a partir do conjunto de axiomas e regras.*

Definição 2.23 (Correção)**.** *Um sistema axiomático é* correto *se $\vdash \phi$ implica em $\Vdash \phi$, isto é, se todo teorema do sistema axiomático é realmente uma fórmula válida da lógica.*

Definição 2.24 (Completude)**.** *Um sistema axiomático é* completo *se $\Vdash \phi$ implica em $\vdash \phi$, isto é, se toda fórmula válida da lógica pode ser deduzida a partir do sistema axiomático.*

Teorema 2.25 (Correção e Completude [10])**.** *O sistema axiomático* **K** *é correto e completo com relação à lógica modal básica.*

Devido a este teorema, a lógica modal básica também é conhecida como lógica **K**.

2.2 Extensões da Lógica Modal Básica

Nesta seção, apresentamos algumas extensões simples da lógica modal básica.

2.2.1 Lógica Multimodal

Uma possível generalização da lógica modal básica consiste em considerar estruturas de Kripke com mais de uma relação binária e uma linguagem com mais de um par de modalidades. Estas lógicas são conhecidas como lógicas multimodais.

Definição 2.26. *A linguagem da lógica n-modal consiste de um conjunto enumerável Φ de símbolos proposicionais, os operadores booleanos $\neg$ e $\wedge$, a constante $\top$ e n operadores modais (ou modalidades): $\Diamond_i$, $1 \leq i \leq n$. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_1\varphi \mid \ldots \mid \Diamond_n\varphi,$$

onde $p \in \Phi$.

As fórmulas da lógica n-modal são avaliadas em estruturas e modelos de Kripke com n relações binárias.

Definição 2.27. *Uma* n-estrutura *é uma n-upla $\mathcal{F} = (W, R_1, \ldots, R_n)$, onde W é um conjunto não vazio de mundos e R_i é uma relação binária sobre W, isto é, $R_i \subseteq W \times W$, $1 \leq i \leq n$.*

Definição 2.28. *Um* n-modelo *é um par $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, onde $\mathcal{F}$ é uma n-estrutura e $\mathbf{V}$ é uma função de valoração.*

A partir destes dois conceitos, fazemos a seguinte alteração na definição de satisfazibilidade de uma fórmula da lógica.

$$\mathcal{M}, v \Vdash \Diamond_i\varphi \text{ sse há } w \in W \text{ tal que } vR_iw \text{ e } \mathcal{M}, w \Vdash \varphi.$$

As modalidades $\Box_i$ são definidas de forma que $\Box_i\varphi \equiv \neg\Diamond_i\neg\varphi$. As complexidades computacionais da lógica modal básica não se alteram no caso de uma lógica n-modal. O sistema axiomático **K** se estende facilmente para um sistema axiomático correto e completo para a lógica n-modal (denominado sistema $\mathbf{K}_n$). As únicas alterações necessárias são a existência de n axiomas $(\mathbf{K}_i)$ e n regras $(\mathbf{Gen}_i)$ para as modalidades $\Box_i$ e n axiomas $(\mathbf{Du}_i)$ para as modalidade $\Diamond_i$.

2.2.2 Modalidades Transitivas

Uma outra possível extensão da lógica modal básica consiste no uso de modalidades transitivas, que são avaliadas de acordo com o fecho transitivo (ou transitivo-reflexivo) da relação binária da estrutura de Kripke.

Definição 2.29. *A linguagem da lógica modal com modalidades transitivas consiste de um conjunto enumerável* Φ *de símbolos proposicionais, os operadores booleanos* $\neg$ *e* $\wedge$*, a constante* $\top$ *e três operadores modais (ou modalidades):* $\Diamond$*,* $\Diamond^+$ *e* $\Diamond^*$*. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond\varphi \mid \Diamond^+\varphi \mid \Diamond^*\varphi,$$

onde $p \in \Phi$*.*

As fórmulas desta lógica são avaliadas nas estruturas e modelos de Kripke usuais. As seguintes adições são feitas na definição de satisfazibilidade de uma fórmula da lógica, onde R^+ denota o fecho transitivo da relação binária R e R^* denota o fecho transitivo-reflexivo da relação R.

1. $\mathcal{M}, v \Vdash \Diamond^+\varphi$ sse há $w \in W$ tal que vR^+w e $\mathcal{M}, w \Vdash \varphi$;
2. $\mathcal{M}, v \Vdash \Diamond^*\varphi$ sse há $w \in W$ tal que vR^*w e $\mathcal{M}, w \Vdash \varphi$.

As modalidades $\Box^+$ e $\Box^*$ são definidas de forma que $\Box^+\varphi \equiv \neg\Diamond^+\neg\varphi$ e $\Box^*\varphi \equiv \neg\Diamond^*\neg\varphi$. Retornando ao exemplo exibido na figura 2.1, supondo que φ é satisfeita no mundo v em $\mathcal{M}$, então $\mathcal{M}, w \Vdash \Diamond^+\varphi$ e $\mathcal{M}, w \Vdash \Diamond^+\Box\bot$.

A complexidade computacional do problema de verificação de modelo não se altera com a adição das modalidades transitivas. No entanto, há alteração nas complexidades dos problemas de satisfazibilidade e de validade.

Teorema 2.30 ([10])**.** *O problema da satisfazibilidade e o problema da validade para a lógica modal básica com modalidades transitivas são EXPTEMPO-Completos no comprimento da fórmula.*

O sistema axiomático **K** pode ser estendido para incluir as modalidades transitivas. Entretanto, vamos adiar esta discussão até a seção 2.5, quando discutimos a axiomatização da lógica dinâmica proposicional, que também contém modalidades transitivas.

2.2.3 Modalidade Conversa

Uma terceira possível extensão da lógica modal básica consiste no uso da modalidade conversa, que é avaliada de acordo com o inverso R^{-1} da relação binária R da estrutura de Kripke.

Definição 2.31. *A linguagem da lógica modal com modalidade conversa consiste de um conjunto enumerável Φ de símbolos proposicionais, os operadores booleanos $\neg$ e $\wedge$, a constante $\top$ e dois operadores modais (ou modalidades): $\Diamond$ e $\Diamond^{-1}$. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond\varphi \mid \Diamond^{-1}\varphi,$$

onde $p \in \Phi$.

As fórmulas desta lógica são avaliadas nas estruturas e modelos de Kripke usuais. A seguinte adição é feita na definição de satisfazibilidade de uma fórmula da lógica.

$$\mathcal{M}, v \Vdash \Diamond^{-1}\varphi \text{ sse há } w \in W \text{ tal que } wRv \text{ e } \mathcal{M}, w \Vdash \varphi.$$

A modalidade $\Box^{-1}$ é definida de forma que $\Box^{-1}\varphi \equiv \neg\Diamond^{-1}\neg\varphi$. As complexidades computacionais da lógica modal básica não se alteram com a adição da modalidade conversa. O sistema axiomático **K** pode ser estendido para incluir a modalidade conversa. O único axioma que necessita ser adicionado é um axioma que estabelece a inter-relação entre $\Diamond$ e $\Diamond^{-1}$: $p \rightarrow \Box\Diamond^{-1}p \wedge \Box^{-1}\Diamond p$.

A definição de morfismo limitado entre duas estruturas necessita de uma condição extra, devido a presença da modalidade conversa:

$$\text{Se } w'R'f(v), \text{ então há } w \text{ tal que } wRv \text{ e } f(w) = w'.$$

2.2.4 Lógica Modal Graduada

Outra extensão da lógica modal básica consiste no uso de modalidades graduadas. Ao invés de usarmos a modalidade $\Diamond$ que expressa "existe pelo menos um mundo acessível a partir do mundo atual...", definimos modalidades $\Diamond_i$, para todo $i \in \mathbb{N}$. Cada modalidade $\Diamond_i$ expressa "existem pelo menos i mundos distintos acessíveis a partir do atual...".

Definição 2.32. *A linguagem da* lógica modal graduada *consiste de um conjunto enumerável* Φ *de símbolos proposicionais, os operadores booleanos* $\neg$ *e* $\wedge$*, a constante* $\top$ *e os operadores modais (ou modalidades)* $\Diamond_i$*, para todo* $i \in \mathbb{N}$*. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_i\varphi,$$

onde $p \in \Phi$ *e* $i \in \mathbb{N}$*.*

As fórmulas desta lógica são avaliadas nas estruturas e modelos de Kripke usuais. A seguinte alteração é feita na definição de satisfazibilidade de uma fórmula da lógica.

$$\mathcal{M}, v \Vdash \Diamond_i\phi \text{ sse } \#\mathcal{R}(v,\phi) \geq i,$$

onde $\mathcal{R}(v,\phi) = \{w : vRw \text{ e } \mathcal{M}, w \Vdash \phi\}$ e $\#S$ denota a cardinalidade do conjunto S.

A modalidade $\Box_i$ é definida de forma que $\Box_i\varphi \equiv \neg\Diamond_i\neg\varphi$. Além disto, para tornar a linguagem mais elegante, introduzimos a seguinte abreviação: $\blacklozenge_i\varphi = \Diamond_i\varphi \wedge \neg\Diamond_{i+1}\varphi$.

A fórmula $\Diamond_i\varphi$ é satisfeita em um mundo v se existem *pelo menos* i mundos distintos v_k, $1 \leq k \leq i$, tais que vRv_k e φ é satisfeita em v_k. A fórmula $\blacklozenge_i\varphi$ é satisfeita em um mundo v se existem *exatamente* i mundos distintos v_k, $1 \leq k \leq i$, tais que vRv_k e φ é satisfeita em v_k.

Definição 2.33. *Podemos definir o comprimento de uma fórmula* ϕ *na lógica modal graduada de duas maneiras diferentes. Chamamos estas definições de* comprimento padrão*, denotado por* $|\phi|$*, e* comprimento não graduado*, denotado por* $||\phi||$*. Definimos ambos os comprimentos da mesma maneira que na lógica modal básica, exceto com relação a fórmulas da forma* $\phi = \Diamond_i\psi$*, onde o comprimento padrão é definido como* $|\phi| = i + |\psi|$ *e o comprimento não graduado é definido como* $||\phi|| = 1 + ||\psi||$*.*

Teorema 2.34 ([12])**.** *O problema da satisfazibilidade e o problema da validade para a lógica modal graduada são PESPAÇO-Completos no comprimento padrão da fórmula.*

Teorema 2.35 ([13])**.** *O problema da verificação de modelo para a lógica modal graduada é polinomial (linear) no produto do tamanho do modelo e do comprimento não graduado da fórmula.*

2.3 Lógica Híbrida

Nesta seção, apresentamos a lógica híbrida básica, uma extensão da lógica modal básica que possui maior poder de expressão do que a lógica modal básica, sem no entanto apresentar maior complexidade computacional. Boas introduções à lógica híbrida são apresentadas em [14] e [15].

2.3.1 Linguagem e Semântica

A principal diferença da lógica híbrida para a lógica modal básica é a presença de um novo tipo de símbolo atômico: os *nominais*. Nominais se comportam de maneira semelhante a símbolos proposicionais. A diferença chave entre eles está relacionada a suas valorações em um modelo. Enquanto o conjunto $\mathbf{V}(p)$ de um símbolo proposicional p pode ser qualquer elemento de $\mathcal{P}(W)$, o conjunto $\mathbf{V}(i)$ de um nominal precisa ser um conjunto unitário. Desta forma, cada nominal é satisfeito em exatamente um mundo do modelo e, portanto, pode ser usado para referenciar um mundo específico do modelo. A validade de fórmulas híbridas não é preservada nem por uniões disjuntas nem por imagens mórficas limitadas, o que torna o poder expressivo de sua linguagem maior.

Definição 2.36. *A linguagem da lógica híbrida básica consiste de um conjunto enumerável Φ de símbolos proposicionais e um conjunto enumerável $\mathcal{L}$ de nominais tais que $\Phi \cap \mathcal{L} = \emptyset$, os operadores booleanos $\neg$ e $\wedge$, a constante $\top$ e os operadores modais (ou modalidades) $@_i$, para cada nominal i (chamados de* operadores de satisfação*), e $\Diamond$. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid i \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond\varphi \mid @_i\varphi,$$

onde $p \in \Phi$ e $i \in \mathcal{L}$.

A definição de uma estrutura é a mesma da lógica modal básica. A definição de um modelo é um pouco diferente.

Definição 2.37. *Um modelo híbrido é um par $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, onde $\mathcal{F}$ é uma estrutura e $\mathbf{V}$ é uma função de valoração que mapeia símbolos proposicionais em subconjuntos de W, isto é, $\mathbf{V} : \Phi \mapsto \mathcal{P}(W)$, e mapeia nominais em subconjuntos unitários de W,*

isto é, se i é um nominal então $\mathbf{V}(i) = \{v\}$ para algum $v \in W$. Este mundo único que pertence a $\mathbf{V}(i)$ é chamado de denotação *de i sob $\mathbf{V}$. Também podemos dizer que i* denota *ou* nomeia *este mundo único que pertence a $\mathbf{V}(i)$.*

Definição 2.38. *A noção de satisfazibilidade da lógica híbrida é definida adicionando duas cláusulas extras à definição de satisfazibilidade da lógica modal básica:*

1. $\mathcal{M}, v \Vdash i$ *sse* $v \in \mathbf{V}(i)$;
2. $\mathcal{M}, v \Vdash @_i\varphi$ *sse* $\mathcal{M}, d_i \Vdash \varphi$, *onde* d_i *é a denotação de* i *sob* $\mathbf{V}$.

Teorema 2.39. 1. $\mathcal{M}, v \Vdash \neg@_i\phi$ *sse* $\mathcal{M}, v \Vdash @_i\neg\phi$;
2. $\mathcal{M}, v \Vdash @_i(\phi_1 \wedge \phi_2)$ *sse* $\mathcal{M}, v \Vdash @_i\phi_1 \wedge @_i\phi_2$;
3. $\mathcal{M}, v \Vdash @_i(\phi_1 \vee \phi_2)$ *sse* $\mathcal{M}, v \Vdash @_i\phi_1 \vee @_i\phi_2$;
4. $\mathcal{M}, v \Vdash @_i(\phi_1 \rightarrow \phi_2)$ *sse* $\mathcal{M}, v \Vdash @_i\phi_1 \rightarrow @_i\phi_2$.

Demonstração. 1. $\mathcal{M}, v \Vdash \neg@_i\phi$ sse $\mathcal{M}, v \nVdash @_i\phi$ sse $\mathcal{M}, d_i \nVdash \phi$, onde d_i é a denotação de i sob a valoração de $\mathcal{M}$, sse $\mathcal{M}, d_i \Vdash \neg\phi$ sse $\mathcal{M}, v \Vdash @_i\neg\phi$;

2. $\mathcal{M}, v \Vdash @_i(\phi_1 \wedge \phi_2)$ sse $\mathcal{M}, d_i \Vdash \phi_1 \wedge \phi_2$, onde d_i é a denotação de i sob a valoração de $\mathcal{M}$, sse $\mathcal{M}, d_i \Vdash \phi_1$ e $\mathcal{M}, d_i \Vdash \phi_2$ sse $\mathcal{M}, v \Vdash @_i\phi_1$ e $\mathcal{M}, v \Vdash @_i\phi_2$ sse $\mathcal{M}, v \Vdash @_i\phi_1 \wedge @_i\phi_2$;

3. Segue diretamente dos itens anteriores;

4. Segue diretamente dos itens anteriores.

□

Teorema 2.40 ([16])**.** *O problema da satisfazibilidade e o problema da validade para a lógica híbrida são PESPAÇO-Completos no comprimento da fórmula.*

Teorema 2.41 ([17])**.** *O problema da verificação de modelo para a lógica híbrida é polinomial (linear) no produto do tamanho do modelo e do comprimento da fórmula.*

Podemos ver que a lógica híbrida é sem dúvida uma lógica interessante, uma vez que ela possui um poder expressivo maior sem nenhum acréscimo relevante na complexidade computacional em relação à lógica modal básica.

2.3.2 Axiomatização

O conjunto de fórmulas válidas da lógica híbrida possui uma axiomatização que é uma extensão do sistema axiomático **K**, com axiomas para os novos operadores da linguagem. Consideramos o seguinte conjunto de axiomas e regras em adição aos axiomas e regras de **K**, onde p e q são símbolos proposicionais, i e j são nominais e φ é uma fórmula. Este sistema axiomático é denotado por $\mathbf{K}_@$.

($\mathbf{K}_@$) $\vdash @_i(p \to q) \to (@_i p \to @_i q)$

(SD) $\vdash @_i p \leftrightarrow \neg @_i \neg p$

(Ref) $\vdash @_i i$

(Agr) $\vdash @_i @_j p \leftrightarrow @_j p$

(Int) $\vdash i \to (p \leftrightarrow @_i p)$

(Bac) $\vdash \Diamond @_i p \to @_i \varphi$

(Gen@) Se $\vdash \varphi$, então $\vdash @_i \varphi$

(Nam) Se $\vdash @_i \varphi$ e i não ocorre em φ, então $\vdash \varphi$

(BG) Se $\vdash @_i \Diamond j \to @_j \varphi$ e $j \neq i$ não ocorre em φ, então $\vdash @_i \Box \varphi$

Uma coisa que é importante notar a respeito da regra **(Sub)** presente no sistema axiomático **K** (seção 2.1.4) é que, no contexto da lógica híbrida a substituição deve respeitar a diferença entre símbolos proposicionais e nominais. A substituição deve substituir uniformemente nominais por nominais e símbolos proposicionais por fórmulas.

Teorema 2.42 (Correção e Completude [18])**.** *O sistema axiomático* $\mathbf{K}_@$ *é correto e completo com relação à lógica híbrida.*

2.3.3 Quantificador Local

Uma possível extensão da lógica híbrida consiste na adição de uma forma limitada de quantificação sobre mundos de uma estrutura ou modelo. Para isto, adiciona-se à linguagem o quantificador local $\downarrow$.

Definição 2.43. *A linguagem da lógica híbrida com quantificador local consiste de um conjunto enumerável* Φ *de símbolos proposicionais, um conjunto enumerável* $\mathcal{L}$ *de nominais e um conjunto enumerável* $\mathcal{S}$ *de variáveis de estado tais que os conjuntos são disjuntos dois a dois, os operadores booleanos* $\neg$ *e* $\wedge$*, a constante* $\top$*, os operadores modais (ou modalidades)* $@_i$*, para cada nominal* i*,* $@_x$*, para cada variável de estado* x*, e* $\Diamond$ *e o quantificador local* $\downarrow$*. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid i \mid x \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond\varphi \mid @_i\varphi \mid @_x\varphi \mid \downarrow x.\varphi,$$

onde $p \in \Phi$*,* $i \in \mathcal{L}$ *e* $x \in \mathcal{S}$*.*

As definições de uma estrutura e de um modelo são as mesmas da lógica híbrida básica.

Para lidar com as variáveis de estado, precisamos introduzir a noção de *atribuições.*

Definição 2.44. *Uma* atribuição *é uma função* g *que mapeia variáveis de estado para mundos do modelo, isto é,* $g : \mathcal{S} \mapsto W$*. Usamos a notação* $g' = g[v_1/x_1, \ldots, v_n/x_n]$ *para denotar uma atribuição tal que* $g'(x) = g(x)$ *se* $x \notin \{x_1, \ldots, x_n\}$ *e* $g'(x_i) = v_i$*, caso contrário.*

Um detalhe importante é que na lógica híbrida com quantificador local é necessário levar em consideração a atribuição g. Desta forma, escreve-se $\mathcal{M}, g, v \Vdash \varphi$ ao invés de $\mathcal{M}, v \Vdash \varphi$. No entanto, as únicas cláusulas em que a atribuição g é utilizada são exatamente as novas cláusulas definidas abaixo.

Definição 2.45. *A noção de satisfazibilidade da lógica híbrida com quantificador local é definida adicionando três cláusulas extras à definição de satisfazibilidade da lógica híbrida básica.*

1. $\mathcal{M}, g, v \Vdash x$ *sse* $g(x) = v$*;*
2. $\mathcal{M}, g, v \Vdash @_x\phi$ *sse* $\mathcal{M}, g, d \Vdash \phi$*, onde* $d = g(x)$*;*
3. $\mathcal{M}, g, v \Vdash \downarrow x.\phi$ *sse* $\mathcal{M}, g[v/x], v \Vdash \phi$*.*

A fórmula $\downarrow x.\varphi$ denota que, utilizando-se x como nome para o mundo atual (variáveis de estado podem ser pensadas como "nominais sob demanda"), φ é satisfeita. O operador $\downarrow$ é o único operador que liga uma variável. Variáveis livres e ligadas são definidas da forma usual. O único caso que merece menção é que na fórmula $@_x\psi$, x é livre (a não ser que esteja sendo ligada no interior de ψ). Uma *sentença* é uma fórmula sem variáveis livres. Em geral, consideramos apenas fórmulas que são sentenças, porque não desejamos nos preocupar com atribuições. Além disso, variáveis livres podem sempre ser substituídas por nominais.

O quantificador local $\downarrow$, assim como os operadores de satisfação, é dual a si mesmo: $\mathcal{M}, g, v \Vdash \neg \downarrow x.\phi$ sse $\mathcal{M}, g, v \nVdash \downarrow x.\phi$ sse $\mathcal{M}, g[v/x], v \nVdash \phi$ sse $\mathcal{M}, g[v/x], v \Vdash \neg\phi$ sse $\mathcal{M}, g, v \Vdash \downarrow x.\neg\phi$.

Teorema 2.46 ([16])**.** *O problema da satisfazibilidade e o problema da validade para a lógica híbrida com quantificador local $\downarrow$ são* indecidíveis*.*

Este resultado mostra que a inclusão de variáveis de estado e do operador $\downarrow$ transformam uma lógica que tinha a mesma complexidade computacional da lógica modal básica em uma lógica indecidível.

Teorema 2.47 ([17])**.** *O problema da verificação de modelo para a lógica híbrida com quantificador local $\downarrow$ é PESPAÇO-Completo tanto no tamanho do modelo quanto no comprimento da fórmula.*

Em [19], é mostrado que, para uma família de lógicas híbridas com o quantificador local $\downarrow$, existem fragmentos destas lógicas em que as complexidades dos problemas de satisfazibilidade e de verificação de modelo são mais baixas do que as das lógicas completas. Um dos fragmentos discutidos em [19], que é definido usando a noção de fórmulas em forma normal negativa, é um fragmento da lógica híbrida com quantificador local definida nesta seção e a complexidade do problema de verificação de modelo para este fragmento é inferior a complexidade presente no teorema acima. Este resultado será de grande valia em um capítulo posterior da tese.

Definição 2.48. *Uma fórmula da lógica híbrida com quantificador local $\downarrow$ está em* forma normal negativa *(FNN) se o símbolo de negação ($\neg$) aparece apenas imediatamente em frente de símbolos proposicionais, nominais e variáveis de estado.*

Lema 2.49. *Se considerarmos os operadores duais de* $\top, \wedge$ *e* $\Diamond$*, isto é,* $\bot, \vee$ *e* $\Box$ *como operadores primitivos da linguagem, então cada fórmula da lógica híbrida com quantificador local é semanticamente equivalente a uma fórmula em FNN.*

Demonstração. Seja $(\triangle, \triangledown)$ um dos seguintes pares de operadores duais: $(\wedge, \vee)$, $(\Diamond, \Box), (\downarrow x., \downarrow x.)$ e $(@_z, @_z)$, onde x é uma variável de estado e z é um nominal ou uma variável de estado. Usando as equivalências semânticas $\neg\top \equiv \bot$, $\neg\bot \equiv \top, \neg\neg\phi \equiv \phi$ e $\neg \triangle \phi \equiv \triangledown\neg\phi$, podemos empurrar os símbolos de negação para o interior das fórmulas até que eles apareçam apenas imediatamente em frente de símbolos proposicionais, nominais e variáveis de estado. □

Teorema 2.50 ([19])**.** *O problema da verificação de modelo para fórmulas da lógica híbrida com quantificador local* $\downarrow$ *que, quando colocadas em FNN, não possuem nenhuma ocorrência do operador* $\Box$ *é NP-Completo tanto no tamanho do modelo quanto no comprimento da fórmula.*

2.4 Lógicas Temporais

Nesta seção, apresentamos duas lógicas modais pertencentes ao grupo das chamadas *lógicas temporais*. Nestas lógicas, além das modalidades que expressam a existência de mundos satisfazendo certa propriedade, existem também modalidades que expressam a existência de *caminhos* no modelo satisfazendo certa propriedade.

2.4.1 CTL*

A lógica CTL* é uma lógica temporal muito poderosa, que possui dois conjuntos inteiramente independentes de modalidades para discursar sobre mundos e sobre caminhos, respectivamente.

Definição 2.51. *A linguagem da lógica CTL** *consiste de um conjunto enumerável* Φ *de símbolos proposicionais, os operadores booleanos* $\neg$ *e* $\wedge$*, a constante* $\top$ *e os operadores modais (ou modalidades)* **A***,* **E***,* **X***,* **F***,* **G** *e* **U***. As fórmulas são divididas em* fórmulas de mundo S *e* fórmulas de caminho P *definidas pela seguinte indução mútua: definidas da seguinte forma:*

$$S ::= p \mid \top \mid \neg S \mid S_1 \wedge S_2 \mid \mathbf{A}P \mid \mathbf{E}P$$

$$P ::= S \mid \neg P \mid P_1 \wedge P_2 \mid \mathbf{X}P \mid \mathbf{F}P \mid \mathbf{G}P \mid P_1 \mathbf{U} P_2$$

onde $p \in \Phi$. O conjunto das fórmulas bem formadas da lógica CTL* é então o conjunto de todas as fórmulas de mundo *geradas pelas regras acima.*

As definições de uma estrutura e de um modelo são as mesmas da lógica modal básica. A noção de satisfazibilidade é definida a seguir.

Definição 2.52. *Seja $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ um modelo. Também precisamos da notação auxiliar de que se $\pi = \langle s_0, s_1, \ldots \rangle$ é um caminho, nós denotamos por π^i o sufixo de π começando em s_i. A noção de satisfazibilidade de uma fórmula de mundo S em um modelo $\mathcal{M}$ em um mundo v ou de uma fórmula de caminho P em um modelo $\mathcal{M}$ em um caminho π, notação $\mathcal{M}, v \Vdash S$ e $\mathcal{M}, \pi \Vdash P$, pode ser indutivamente definida da seguinte forma:*

1. *$\mathcal{M}, v \Vdash p$ sse $v \in \mathbf{V}(p)$;*
2. *$\mathcal{M}, v \Vdash \top$ sempre;*
3. *$\mathcal{M}, v \Vdash \neg S$ sse $\mathcal{M}, v \nVdash S$;*
4. *$\mathcal{M}, v \Vdash S_1 \wedge S_2$ sse $\mathcal{M}, v \Vdash S_1$ e $\mathcal{M}, v \Vdash S_2$;*
5. *$\mathcal{M}, v \Vdash \mathbf{A}P$ sse para todo caminho π começando em v, $\mathcal{M}, \pi \Vdash P$;*
6. *$\mathcal{M}, v \Vdash \mathbf{E}P$ sse existe um caminho π começando em v tal que $\mathcal{M}, \pi \Vdash P$;*
7. *$\mathcal{M}, \pi \Vdash S$ sse v é o primeiro mundo de π e $\mathcal{M}, v \Vdash S$;*
8. *$\mathcal{M}, \pi \Vdash \neg P$ sse $\mathcal{M}, \pi \nVdash P$;*
9. *$\mathcal{M}, \pi \Vdash P_1 \wedge P_2$ sse $\mathcal{M}, \pi \Vdash P_1$ e $\mathcal{M}, \pi \Vdash P_2$;*
10. *$\mathcal{M}, \pi \Vdash \mathbf{X}P$ sse $\mathcal{M}, \pi^1 \Vdash P$;*
11. *$\mathcal{M}, \pi \Vdash \mathbf{F}P$ sse existe um $k \geq 0$ tal que $\mathcal{M}, \pi^k \Vdash P$;*
12. *$\mathcal{M}, \pi \Vdash \mathbf{G}P$ sse para todo $k \geq 0$, $\mathcal{M}, \pi^k \Vdash P$;*
13. *$\mathcal{M}, \pi \Vdash P_1 \mathbf{U} P_2$ sse existe um $k \geq 0$ tal que $\mathcal{M}, \pi^k \Vdash P_2$ e para todo $0 \leq j < k$, $\mathcal{M}, \pi^j \Vdash P_1$.*

Intuitivamente, devemos pensar em **A** como "para todos os caminhos começando no mundo atual", **E** como "existe um caminho começando no mundo atual tal que...", **X** como "no próximo mundo do caminho atual", **F** como "no mundo atual ou em algum mundo futuro no caminho atual", **G** como "no mundo atual e em todos os mundos futuros no caminho atual" e **U** como "a primeira fórmula é satisfeita no caminho até que a segunda fórmula seja satisfeita neste caminho, e a segunda fórmula é eventualmente satisfeita".

Teorema 2.53 ([20])**.** *O problema da satisfazibilidade e o problema da validade para CTL* são 2EXPTEMPO-Completos no comprimento da fórmula.*

Teorema 2.54 ([11])**.** *O problema da verificação de modelo para CTL* é polinomial (linear) no tamanho do modelo e EXPTEMPO no comprimento da fórmula.*

2.4.2 CTL

A lógica CTL é um subconjunto da lógica CTL*, em que as modalidades de CTL* sempre aparecem em pares **PS**, onde **P** é uma modalidade que quantifica sobre caminhos (**A** ou **E**) e **S** é uma modalidade que quantifica sobre mundos ($\mathbf{X}, \mathbf{F}, \mathbf{G}$ ou **U**). Desta forma, as modalidades de CTL, na prática, são **AX**, **EX**, **AF**, **EF**, **AG**, **EG**, **AU** e **EU**, com o detalhe importante de que as duas últimas modalidades são binárias e não unárias.

A lógica CTL apresenta vantagens em relação à lógica CTL* em termos de complexidade computacional, como é mostrado no final desta seção. O decréscimo em complexidade é especialmente atrativo no problema da verificação de modelos, o que faz a lógica CTL ser uma das mais utilizadas para verificação automática de sistemas.

Definição 2.55. *A linguagem da lógica CTL consiste de um conjunto enumerável* Φ *de símbolos proposicionais, os operadores booleanos* $\neg$ *e* $\wedge$*, a constante* $\top$ *e os operadores modais (ou modalidades)* **EX***,* **EU** *e* **AU***. As fórmulas são definidas da seguinte forma: definidas da seguinte forma:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathbf{EX}\varphi \mid \mathbf{EU}(\varphi_1, \varphi_2) \mid \mathbf{AU}(\varphi_1, \varphi_2),$$

onde $p \in \Phi$*.*

As definições de uma estrutura e de um modelo são as mesmas da lógica modal básica. A noção de satisfazibilidade pode ser inferida a partir das regras da lógica CTL*, uma vez que CTL é um fragmento dela. A partir destas regras, podemos notar que as modalidades ausentes na definição acima podem ser definidas através de equivalências semânticas: $\mathbf{AX}\varphi \equiv \neg\mathbf{EX}\neg\varphi$, $\mathbf{EF}\varphi \equiv \mathbf{EU}(\top, \varphi)$, $\mathbf{AF}\varphi \equiv \mathbf{AU}(\top, \varphi)$, $\mathbf{EG}\varphi \equiv \neg\mathbf{AF}\neg\varphi$ e $\mathbf{AG}\varphi \equiv \neg\mathbf{EF}\neg\varphi$. É importante notar que **EU** e **AU** *não são* interdefiníveis desta forma.

Abaixo, apresentamos uma noção simplificada de satisfazibilidade específica para CTL.

Definição 2.56. *Seja $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ um modelo. A noção de satisfazibilidade de uma fórmula φ em um modelo $\mathcal{M}$ em um mundo v, notação $\mathcal{M}, v \Vdash \varphi$, pode ser indutivamente definida da seguinte forma:*

1. *$\mathcal{M}, v \Vdash p$ sse $v \in \mathbf{V}(p)$;*
2. *$\mathcal{M}, v \Vdash \top$ sempre;*
3. *$\mathcal{M}, v \Vdash \neg\varphi$ sse $\mathcal{M}, v \nVdash \varphi$;*
4. *$\mathcal{M}, v \Vdash \varphi_1 \wedge \varphi_2$ sse $\mathcal{M}, v \Vdash \varphi_1$ e $\mathcal{M}, v \Vdash \varphi_2$;*
5. *$\mathcal{M}, v \Vdash \mathbf{EX}\varphi$ sse há $w \in W$ tal que vRw e $\mathcal{M}, w \Vdash \varphi$;*
6. *$\mathcal{M}, v \Vdash \mathbf{EU}(\varphi_1, \varphi_2)$ sse existe um caminho π começando em v e um $k \geq 0$ tal que $\mathcal{M}, \pi^k \Vdash \varphi_2$ e para todo $0 \leq j < k$, $\mathcal{M}, \pi^j \Vdash P_1$;*
7. *$\mathcal{M}, v \Vdash \mathbf{AU}(\varphi_1, \varphi_2)$ sse para todo caminho π começando em v existe um $k \geq 0$ tal que $\mathcal{M}, \pi^k \Vdash \varphi_2$ e para todo $0 \leq j < k$, $\mathcal{M}, \pi^j \Vdash P_1$.*

Teorema 2.57 ([20])**.** *O problema da satisfazibilidade e o problema da validade para CTL são EXPTEMPO-Completos no comprimento da fórmula.*

Teorema 2.58 ([11])**.** *O problema da verificação de modelo para CTL é polinomial (linear) no produto do tamanho do modelo e do comprimento da fórmula.*

2.5 Lógica Dinâmica Proposicional (PDL)

Nesta seção, apresentamos a lógica dinâmica proposicional. Esta lógica é uma lógica multimodal projetada para a descrição e verificação de propriedades de programas e sistemas sequenciais.

2.5.1 Linguagem e Semântica

A linguagem da lógica dinâmica proposicional (PDL) possui, além de fórmulas, a noção de *programas*. A linguagem possui um conjunto finito de *programas básicos* e programas mais complexos podem ser construídos a partir dos três operadores de linguagens regulares: *concatenação sequencial* (;), *união* ou *escolha não determinística* ($\cup$) e *iteração* ou *estrela de Kleene* (*).

Definição 2.59. *A linguagem da* lógica dinâmica proposicional *(PDL) consiste de um conjunto enumerável* Φ *de símbolos proposicionais, um conjunto finito* Π *de programas básicos, os operadores booleanos* $\neg$ *e* $\wedge$*, a constante* $\top$*, os construtores de programas* ;, $\cup$ *e* * *e um operador modal (ou modalidade)* $\langle\pi\rangle$*, para cada programa* π*. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\pi\rangle\varphi,$$

com

$$\pi ::= a \mid \pi_1;\pi_2 \mid \pi_1 \cup \pi_2 \mid \pi^*,$$

onde $p \in \Phi$ *e* $a \in \Pi$*.*

As fórmulas desta lógica são avaliadas em estruturas e modelos de Kripke conhecidos como estruturas e modelos *regulares*.

Definição 2.60. *Uma* estrutura regular *é uma n-upla* $\mathcal{F} = (W, \{R_a\}_{a\in\Pi})$*, onde* W *é um conjunto não vazio de mundos e* R_a *é uma relação binária sobre* W *para cada programa básico* a*. Além disto, relações binárias* R_π*, para cada programa não básico* π*, são definidas indutivamente utilizando-se as seguintes regras:*

- $R_{\pi_1;\pi_2} = R_{\pi_1} \circ R_{\pi_2}$;
- $R_{\pi_1\cup\pi_2} = R_{\pi_1} \cup R_{\pi_2}$;

- $R_{\pi^*} = R_\pi^*$, *onde* R_π^* *denota o fecho transitivo-reflexivo de* R_π.

Definição 2.61. *Um* modelo regular *é um modelo de Kripke* $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, *onde* $\mathcal{F}$ *é uma estrutura regular.*

A noção de satisfazibilidade de PDL é definida da seguinte forma:

Definição 2.62. *Seja* $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ *um modelo regular. A noção de satisfazibilidade de uma fórmula* φ *em um modelo regular* $\mathcal{M}$ *em um mundo* w, *denotada por* $\mathcal{M}, w \Vdash \varphi$, *pode ser definida indutivamente da seguinte forma:*

1. $\mathcal{M}, w \Vdash p$ *sse* $w \in \mathbf{V}(p)$;
2. $\mathcal{M}, w \Vdash \top$ *sempre;*
3. $\mathcal{M}, w \Vdash \neg\varphi$ *sse* $\mathcal{M}, w \nVdash \varphi$;
4. $\mathcal{M}, w \Vdash \varphi_1 \wedge \varphi_2$ *sse* $\mathcal{M}, w \Vdash \varphi_1$ *e* $\mathcal{M}, w \Vdash \varphi_2$;
5. $\mathcal{M}, w \Vdash \langle\pi\rangle\varphi$ *sse existe* $w' \in W$ *tal que* $wR_\pi w'$ *e* $\mathcal{M}, w' \Vdash \varphi$.

2.5.2 Axiomatização

O conjunto de fórmulas válidas de PDL possui uma axiomatização baseada na axiomatização da lógica multimodal com a adição de axiomas que descrevem o funcionamento dos três operadores de construção de programas ($;$, $\cup$ e *). Consideramos o seguinte conjunto de axiomas e regras, onde p e q são símbolos proposicionais e φ e ψ são fórmulas.

(LP) Tautologias da Lógica Proposicional

(K) $\vdash [\pi](p \to q) \to ([\pi]p \to [\pi]q)$

(Du) $\vdash [\pi]p \leftrightarrow \neg\langle\pi\rangle\neg p$

(Com) $\vdash \langle\pi_1; \pi_2\rangle p \leftrightarrow \langle\pi_1\rangle\langle\pi_2\rangle p$

(Uni) $\vdash \langle\pi_1 \cup \pi_2\rangle p \leftrightarrow \langle\pi_1\rangle p \vee \langle\pi_2\rangle p$

(Rec) $\vdash \langle\pi^*\rangle p \leftrightarrow p \vee \langle\pi\rangle\langle\pi^*\rangle p$

(Ind) $\vdash p \wedge [\pi^*](p \rightarrow [\pi]p) \rightarrow [\pi^*]p$

(MP) Se $\vdash \varphi$ e $\vdash \varphi \rightarrow \psi$, então $\vdash \psi$

(Gen) Se $\vdash \varphi$, então $\vdash [\pi]\varphi$

(Sub) Se $\vdash \varphi$, então $\vdash \varphi^\sigma$, onde σ substitui uniformemente símbolos proposicionais por fórmulas

Teorema 2.63 (Correção e Completude [10])**.** *O sistema axiomático acima é correto e completo com relação a PDL.*

Capítulo 3

Conceitos Básicos de Teoria de Grafos

"Marge, lembre-se, se algo der errado na usina, culpe o cara que não sabe falar inglês." - Homer Simpson

Neste capítulo, apresentamos as definições básicas da teoria de grafos, discutimos quatro propriedades globais de grafos (conectividade, aciclicidade, propriedade hamiltoniana e propriedade euleriana) que serão estudadas sob a ótica das lógicas modais posteriormente e, por último, apresentamos a definição de um produto cartesiano de grafos.

3.1 Definições Básicas

Nesta seção, apresentamos algumas definições básicas da teoria de grafos. Para os leitores com interesse particular em teoria de grafos, [1] fornece uma boa introdução.

Definição 3.1. *Um* grafo não direcionado *G é um par (V, E), onde V é um conjunto finito de* vértices *e E é um conjunto de pares* não ordenados *de vértices. Cada um destes pares é chamado de* aresta. *Se $\langle v_i, v_j \rangle \in E$, dizemos que v_i e v_j são* adjacentes. *O* grau *de um vértice é o número de vértices que são adjacentes a ele. Podemos também escrever $v_i E v_j$ para denotar que $\langle v_i, v_j \rangle \in E$.*

Definição 3.2. *Um* grafo direcionado *G é um par (V, E), onde V é um conjunto (finito ou não) de vértices e E é um conjunto de pares* ordenados *de vértices. Cada um destes pares é chamado de* aresta direcionada *ou apenas aresta. Se $\langle v_i, v_j \rangle \in E$,*

dizemos que v_i *é* adjacente a v_j *e* v_j *é* adjacente a partir de v_i. *O* grau de saída *de um vértice é o número de vértices que são adjacentes a partir dele e o* grau de entrada *é o número de vértices que são adjacentes a ele. Podemos também escrever* $v_i E v_j$ *para denotar que* $\langle v_i, v_j \rangle \in E$.

Nesta tese, iremos utilizar quase que na totalidade das vezes grafos direcionados. Desta forma, quando escrevemos apenas *grafo*, estamos nos referindo a um grafo direcionado.

Podemos notar que uma estrutura de Kripke é um grafo com conjunto de arestas R. Além disso, um grafo pode ser utilizado como uma estrutura de Kripke para a avaliação de fórmulas em uma linguagem modal. Isto já mostra porque lógicas modais são uma escolha natural para o estudo de propriedades de grafos. Desta forma, no resto desta tese, os termos *grafo* e *estrutura* (de Kripke) são considerados equivalentes.

A partir disto, escrevemos $G \Vdash \phi$ para denotar que a fórmula ϕ é válida em G quando G é utilizado como uma estrutura de Kripke da lógica.

Definição 3.3. *Um* multigrafo *(direcionado)* G *é uma* n*-upla* $(V, E_1, \ldots, E_n)$*, onde* V *é um conjunto finito de vértices e* E_i*,* $1 \leq i \leq n$*, são conjuntos de pares ordenados de vértices, chamados de arestas (ou* i*-arestas, se a identificação do conjunto ao qual ela pertence for relevante). Podemos, analogamente aos casos anteriores, definir* i*-adjacência e* i*-graus de entrada e saída.*

Da mesma maneira que no caso básico, a definição de um multigrafo é equivalente à definição de uma n-estrutura de Kripke. Abusando da notação, nós usamos nesta tese o termo *grafo* para se referir tanto a um grafo propriamente dito quanto a um multigrafo. Isto não é incomum na literatura de lógica modal.

Definição 3.4. *Um* caminho *em um grafo* G *é uma sequência de vértices* $\langle v_1, v_2, \ldots, v_n \rangle$*,* $n \geq 2$*, onde ao menos dois vértices* v_i *e* v_j*,* $i \neq j$*, são distintos e* $\langle v_i, v_{i+1} \rangle \in E$*, para* $0 < i < n$*. Dizemos que* n *é o comprimento do caminho.*

Definição 3.5. *Um* caminho fechado *é um caminho tal que* $v_1 = v_n$.

Definição 3.6. *Um ciclo é um caminho onde* $v_1 = v_n$ *e* $v_i \neq v_j$*, para* $1 \leq i, j < n$*,* $i \neq j$.

Definição 3.7. *Todo grafo direcionado possui um* grafo não direcionado subjacente*, em que as orientações particulares das arestas não são consideradas. Isto significa que, se $G = (V, E)$ e $\langle v, w\rangle \in E$, então v é adjacente a w e w é adjacente a v no grado não direcionado subjacente a G.*

3.2 Propriedades Globais de Grafos

Nesta seção, definimos as quatro propriedades globais de grafos que serão estudadas posteriormente nesta tese sob o ponto de vista das lógicas modais. Estas propriedades são: conectividade, aciclicidade, propriedade hamiltoniana e propriedade euleriana.

Definição 3.8. *Podemos definir dois níveis de conectividade em um grafo. Primeiramente, um grafo G é* (fracamente) conexo *se e somente se, para quaisquer dois vértices v e w distintos de G, existe um caminho de v para w no grafo não direcionado subjacente a G. No outro nível, um grafo G é* fortemente conexo *se e somente se, para quaisquer dois vértices v e w distintos de G, existe um caminho de v para w no próprio grafo G.*

Definição 3.9. *Um grafo G é* acíclico *se e somente se não existe nenhum caminho em G que é um ciclo, conforme definido na seção anterior.*

Teorema 3.10. *Um grafo G contém um caminho fechado se e somente se ele contém um ciclo.*

Demonstração. A direção da direita para a esquerda é imediata, já que ciclos são um caso particular de caminhos fechados. Para a direção da esquerda para a direita, procedemos por indução no comprimento $n \geq 2$ do caminho fechado. Se G contém um caminho fechado de comprimento $n = 2$, então este caminho fechado é também um ciclo. Suponha agora que, para todo $n < k$, se G contém um caminho fechado de comprimento n, então ele contém um ciclo. Se G contém um caminho fechado $\langle v_1, \ldots, v_k\rangle$ de comprimento k que não é um ciclo, então existem i e j tais que $1 \leq i < j < k$ e $v_i = v_j$. Mas então $\langle v_i, \ldots, v_j\rangle$ é um caminho fechado de comprimento menor do que k. Pela hipótese de indução, G contém um ciclo. □

Definição 3.11. *Um grafo conexo G é* hamiltoniano *se e somente se existe um ciclo em G que passa por todos os seus vértices.*

Definição 3.12. *Um grafo conexo G é* euleriano *se e somente se existe um caminho fechado em G em que toda aresta do grafo aparece exatamente uma vez.*

Teorema 3.13 ([1])**.** *Um grafo conexo G é euleriano se e somente se o grau de entrada de cada um de seus vértices é igual ao seu grau de saída.*

3.3 Produto Cartesiano de Grafos

Nesta seção, definimos a noção de produto cartesiano de grafos, de acordo com [4] e [5].

Definição 3.14. *Dados $n \geq 2$ grafos direcionados $G_i = \langle V_i, E_i \rangle$, $1 \leq i \leq n$, definimos o seu produto G, notação $G = G_1 \times G_2 \times \ldots \times G_n$, como o grafo $G = \langle V_1 \times V_2 \times \ldots \times V_n, \overline{E}_1, \overline{E}_2, \ldots, \overline{E}_n \rangle$, onde para cada i, $1 \leq i \leq n$, $\overline{E}_i$ é uma relação binária em $V_1 \times V_2 \times \ldots \times V_n$ tal que*

$$\langle u_1, \ldots, u_n \rangle \overline{E}_i \langle v_1, \ldots, v_n \rangle \text{ sse } u_i E_i v_i \text{ e } u_k = v_k, \text{ para todo } k \neq i.$$

Um caso particular importante da definição acima diz respeito ao produto de dois grafos. Neste caso, ao invés dos subscritos 1 e 2, é comum usarmos os subscritos h e v. Eles se referem à intuição geométrica de relações de acessibilidade horizontal e vertical.

Definição 3.15. *Dados dois grafos direcionados $G_1 = \langle V_1, E_1 \rangle$ e $G_2 = \langle V_2, E_2 \rangle$, definimos o seu produto G, notação $G = G_1 \times G_2$, como o grafo $G = \langle V_1 \times V_2, E_h, E_v \rangle$, onde para todo $x, u \in V_1$ e $y, v \in V_2$*

1. $\langle x, y \rangle E_h \langle u, v \rangle$ *sse* $x E_1 u$ *e* $y = v$ *e*

2. $\langle x, y \rangle E_v \langle u, v \rangle$ *sse* $y E_2 v$ *e* $x = u$.

Um exemplo de um grafo produto é mostrado na figura 3.1.

Nesta tese, gostaríamos de identificar uma condição necessária e suficiente para um grafo ser o produto de grafos não triviais. Um grafo é dito *trivial* se ele possui

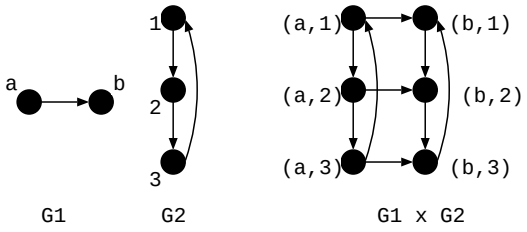

Figura 3.1: Produto de Grafos

apenas um vértice e nenhuma aresta. Todo grafo pode ser descrito como o produto de si mesmo com um grafo trivial.

Em [4] e [5], três propriedades que são satisfeitas em grafos que são produtos são apresentadas. Estas propriedades, em conjunto com a propriedade de Church-Rosser reversa, são necessárias para um grafo ser um produto (figura 3.2). Seja $G = \langle V, \overline{E}_1, \ldots, \overline{E}_n \rangle$ e $1 \leq i, j \leq n$, $i \neq j$. Estas propriedades são descritas da seguinte forma:

1. Comutatividade à Esquerda: $\forall x, y, z \in V(x\overline{E}_j y \wedge y\overline{E}_i z \rightarrow \exists u \in V(x\overline{E}_i u \wedge u\overline{E}_j z))$

2. Comutatividade à Direita: $\forall x, y, z \in V(x\overline{E}_i y \wedge y\overline{E}_j z \rightarrow \exists u \in V(x\overline{E}_j u \wedge u\overline{E}_i z))$

3. Propriedade de Church-Rosser: $\forall x, y, z \in V(x\overline{E}_j y \wedge x\overline{E}_i z \rightarrow \exists u \in V(y\overline{E}_i u \wedge z\overline{E}_j u))$

4. Propriedade de Church-Rosser Reversa: $\forall x, y, z \in V(y\overline{E}_j x \wedge z\overline{E}_i x \rightarrow \exists u \in V(u\overline{E}_i y \wedge u\overline{E}_j z))$

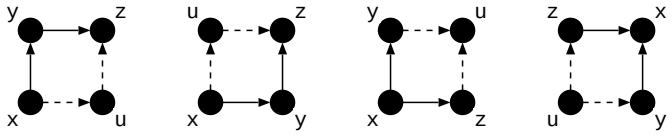

Figura 3.2: Comutatividade à Esquerda e à Direita e Propriedades de Church-Rosser e de Church-Rosser Reversa

Entretanto, embora estas propriedades sejam necessárias para um grafo ser um produto, elas não são suficientes. Existem grafos que satisfazem comutatividade à esquerda e à direita e as propriedades de Church-Rosser e Church-Rosser reversa,

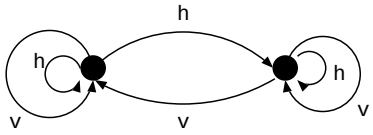

Figura 3.3: Contraexemplo da suficiência das propriedades básicas

mas não podem ser decompostos como um produto de grafos não triviais, como mostra um exemplo de [4] na figura 3.3.

No capítulo 6, iremos acrescentar uma nova propriedade, chamada de *intransitividade*, às quatro propriedades acima e provaremos que a conjunção destas cinco propriedades é uma condição suficiente para que um grafo *conexo* e *enumerável* seja um produto.

Capítulo 4

Conceitos Básicos de Álgebras de Processos

"Cala a boca, Pensamento, ou te enfio uma faca!" - Homer Simpson

Neste capítulo, apresentamos duas álgebras de processos que serão utilizadas no desenvolvimento da tese: o *Cálculo de Sistemas Comunicantes* (CCS) e o π-*Cálculo*. Posteriormente na tese, no capítulo 7, iremos desenvolver extensões da Lógica Dinâmica Proposicional (PDL) com operadores para a descrição de comportamentos concorrentes. Para isto, os programas destas lógicas dinâmicas proposicionais serão descritos em linguagens baseadas em CCS e no π-Cálculo.

4.1 Cálculo de Sistemas Comunicantes (CCS)

O Cálculo de Sistemas Comunicantes (CCS) é uma álgebra de processos bem conhecida, proposta por Robin Milner [21], para a especificação de sistemas comunicantes concorrentes. Ele modela a concorrência e a interação entre processos através de atos individuais de comunicação. Uma especificação CCS é uma descrição do comportamento esperado de um sistema, baseada nos eventos de comunicação que podem ocorrer. Para uma introdução ampla a CCS, [21] pode ser consultado.

4.1.1 Linguagem e Semântica

Em CCS, um par de processos pode se comunicar através de um canal comum e cada ato de comunicação consiste simplesmente de um sinal sendo mandado em

uma ponta do canal e imediatamente sendo recebido na outra.

Seja $\mathcal{N} = \{a, b, c, \ldots\}$ um conjunto de nomes. Cada canal em uma especificação CCS é rotulado por um nome. Os rótulos dos canais também são utilizados para descrever as ações de comunicação (envio e recebimento de sinais) executadas pelos processos, como é mostrado abaixo. Além destas ações de comunicação, CCS possui apenas mais uma ação: a ação silenciosa, denotada por τ, utilizada para representar qualquer ação interna, executada por qualquer dos processos, que não envolva um ato de comunicação (por exemplo, uma atualização de memória).

Existem duas possíveis semânticas para a ação τ em CCS: ela pode ser tratada como sendo observável, da mesma forma que as ações de comunicação, ou ela pode ser tratada como sendo invisível. Nós adotamos a primeira opção, uma vez que ela é mais genérica. Nos formalismos lógicos que apresentamos no capítulo 7, somos capazes de representar a segunda semântica como um caso particular da primeira.

Definição 4.1. *Em nossa apresentação de CCS, especificações de processos podem ser construídas utilizando-se as seguintes operações:*

$$P ::= \alpha \mid \alpha.P \mid \alpha.A \mid P_1 + P_2 \mid P_1|P_2 \mid P\backslash L,$$

com

$$\alpha ::= a \mid \overline{a} \mid \tau,$$

onde $a \in \mathcal{N}$, $L \subseteq \mathcal{N}$ e toda constante A possui uma única equação definidora *$A \stackrel{def}{=} P_A$, onde P_A é uma especificação de processo. Neste trabalho, toda vez que um processo é relacionado a uma constante A através de uma equação definidora, ele será denotado por P_A.*

Originalmente, CCS também define um processo nulo, denotado por **0**. Ele representa o processo que não tem a possibilidade de executar nenhuma ação. Entretanto, devido à sua definição de certa forma frouxa, que falha em diferenciar entre um *deadlock* e uma terminação bem sucedida (ao contrário de outras álgebras de processos, como ACP [22] por exemplo, em que processos em *deadlock* e processos terminados são diferentes), seu uso traria uma séria inconveniência à semântica dos formalismos lógicos iniciais apresentados no capítulo 7: suas semânticas não seriam totalmente composicionais. Isto é mostrado em detalhes no capítulo 7. Devido a isso, excluímos de CCS este processo nulo.

Tabela 4.1: Relações de Transição de CCS

$\alpha \xrightarrow{\alpha} \surd$	$\alpha.P \xrightarrow{\alpha} P$	$\frac{A \overset{def}{=} P_A}{\alpha.A \xrightarrow{\alpha} P_A}$	$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$	$\frac{Q \xrightarrow{\beta} Q'}{P+Q \xrightarrow{\beta} Q'}$
$\frac{P \xrightarrow{\alpha} P'}{P\|Q \xrightarrow{\alpha} P'\|Q}$	$\frac{Q \xrightarrow{\beta} Q'}{P\|Q \xrightarrow{\beta} P\|Q'}$	$\frac{P \xrightarrow{\lambda} P', Q \xrightarrow{\overline{\lambda}} Q'}{P\|Q \xrightarrow{\tau} P'\|Q'}$	$\frac{P \xrightarrow{\alpha} P', \alpha \notin L \cup \overline{L}}{P \backslash L \xrightarrow{\alpha} P' \backslash L}$	

O operador de *prefixo* (.) denota que o processo vai primeiramente executar a ação α e então se comportar como P ou A. O operador de *soma* (ou *escolha não determinística*) (+) denota que o processo irá realizar uma escolha não determinística e se comportar como P_1 ou como P_2. O operador de *composição paralela* (|) denota que os processos P_1 e P_2 podem prosseguir independentemente ou podem comunicar-se através de um canal em comum. Finalmente, o operador de *restrição* ($\backslash$) denota que os canais em L são acessíveis apenas no interior de P. Iteração em CCS pode ser modelada através de equações definidoras recursivas, isto é, equações $A \overset{def}{=} P_A$ onde A ocorre em P_A.

A ação a, chamada *ação de entrada*, denota que o processo recebe um sinal através do canal rotulado por a. A ação $\overline{a}$, chamada *ação de saída*, denota que o processo envia um sinal através do canal rotulado por a. Finalmente, τ denota a ação silenciosa.

Escrevemos $P \xrightarrow{\alpha} P'$ para expressar que o processo P pode executar a ação α e depois disto se comportar como P'. Escrevemos $P \xrightarrow{\alpha} \surd$ para expressar que o processo P termina de maneira bem sucedida após executar a ação α (uma notação emprestada de ACP). Um processo termina apenas quando não existe nenhuma ação possível restante para ele executar. Por exemplo, $\beta \xrightarrow{\beta} \surd$. Quando um processo termina dentro de uma composição paralela, escrevemos P ao invés de $P|\surd$. Também escrevemos $\surd$ ao invés de $\surd \backslash L$ e $\surd|\surd$. Definimos o conjunto $\overline{L}$ como $\overline{L} = \{\overline{a} : a \in L\}$. Na tabela 4.1, apresentamos a semântica para os operadores de CCS baseada nesta notação. Nesta tabela, P, Q e P_A são especificações de processos, enquanto P' e Q' são especificações de processos ou $\surd$.

A partir da tabela 4.1, podemos ver que para descartar completamente o processo nulo $\mathbf{0}$, precisamos também descartar o operador de restrição, uma vez que ele pode ser utilizado para definir um processo com comportamento equivalente ao de $\mathbf{0}$ (por exemplo, $a\backslash\{a\}$). Portanto, em todos os formalismos lógicos do capítulo 7 em que precisarmos descartar o processo $\mathbf{0}$, precisaremos descartar também o operador de

restrição.

Para motivar o uso de CCS, apresentamos um exemplo simples do uso da linguagem abaixo.

Exemplo 4.2 ([21, 23])**.** *Considere uma máquina automática de vendas onde pode-se botar moedas de um ou dois reais e comprar uma barra de chocolate pequena ou grande. Após colocar as moedas, o comprador precisa pressionar o botão pequeno para um chocolate pequeno ou o botão grande para o chocolate grande. A máquina também está programada para desligar-se automaticamente de acordo com um protocolo interno (representado por uma ação τ). Uma especificação CCS descrevendo o comportamento desta máquina é a seguinte:*

$$V = 1r.pequeno.\overline{coleta}.A + 1r.1r.grande.\overline{coleta}.A + 2r.grande.\overline{coleta}.A$$

$$A \stackrel{def}{=} 1r.pequeno.\overline{coleta}.A + 1r.1r.grande.\overline{coleta}.A + 2r.grande.\overline{coleta}.A + \tau$$

Suponhamos agora que um comprador quer usar esta máquina. Podemos descrever o comprador como

$$C = \overline{1r}.\overline{pequeno}.coleta + \overline{1r}.\overline{1r}.\overline{grande}.coleta + \overline{2r}.\overline{grande}.coleta.$$

Note que este comprador não possui um comportamento iterativo. Uma vez que ele obtém o chocolate, suas ações estão encerradas. Agora, se quisermos modelar o processo deste comprador comprando um chocolate nesta máquina, podemos escrever $(V|C)\backslash L$, onde $L = \{1r, 2r, pequeno, grande, coleta\}$.

Definição 4.3. *Seja $\mathcal{P}$ o conjunto de todas as especificações de processos possíveis. Um conjunto $Z \subseteq \mathcal{P} \times \mathcal{P}$ é uma* bissimulação forte *se $(P, Q) \in Z$ implica em*

- *Se $P \stackrel{\alpha}{\rightarrow} P'$ e $P' \in \mathcal{P}$, então existe $Q' \in \mathcal{P}$ tal que $Q \stackrel{\alpha}{\rightarrow} Q'$ e $(P', Q') \in Z$;*
- *Se $Q \stackrel{\alpha}{\rightarrow} Q'$ e $Q' \in \mathcal{P}$, então existe $P' \in \mathcal{P}$ tal que $P \stackrel{\alpha}{\rightarrow} P'$ e $(P', Q') \in Z$;*
- *$P \stackrel{\alpha}{\rightarrow} \surd$ se e somente se $Q \stackrel{\alpha}{\rightarrow} \surd$.*

Definição 4.4. *Duas especificações de processos P e Q são* fortemente bissimilares *(ou simplesmente* bissimilares*), denotado por $P \sim Q$, se existe uma bissimulação forte Z tal que $(P, Q) \in Z$.*

Agora, introduzimos a Lei de Expansão, que é muito importante na definição das semânticas das lógicas apresentadas no capítulo 7 e em suas axiomatizações. Apresentamos um caso particular da Lei de Expansão, que é suficiente para nossas necessidades. O caso mais geral da Lei de Expansão é apresentado em [21].

Definição 4.5. *Dizemos que um processo é* irrestrito *se ele não possui nenhuma ocorrência do operador* $\backslash$.

Teorema 4.6 (Lei de Expansão (EL))**.** *Seja* $P = P_1 \mid P_2$*, onde* P *é irrestrito. Então*

$$P \sim \sum_{P_1 \xrightarrow{\alpha} P_1'} \alpha.(P_1' \mid P_2) + \sum_{P_2 \xrightarrow{\beta} P_2'} \beta.(P_1 \mid P_2') + \sum_{R \in A_\tau} \tau.R,$$

onde $A_\tau = \{(P_1' \mid P_2') : P_1 \xrightarrow{a} P_1'$ *e* $P_2 \xrightarrow{\overline{a}} P_2'$*, para algum* $a \in \mathcal{N}\} \cup \{(P_1' \mid P_2') : P_1 \xrightarrow{\overline{a}} P_1'$ *e* $P_2 \xrightarrow{a} P_2'$*, para algum* $a \in \mathcal{N}\}$*. Denotamos o lado direito desta bissimilaridade por* $Exp(P)$*.*

4.1.2 Sequencias de Ações e Execuções Possíveis

Nesta seção, introduzimos o conceito chave de *execuções finitas possíveis* de um processo. Este conceito desempenha um papel central nas semânticas das lógicas apresentadas no capítulo 7.

Definição 4.7. *Utilizamos a notação* $\overrightarrow{\alpha}$ *para denotar uma sequencia potencialmente infinita de ações* $\alpha_1.\alpha_2.\ .\cdots.\alpha_n(.\cdots)$ *(a sequencia vazia é denotada por* $\overrightarrow{\varepsilon}$*). A sequencia vazia segue a regra* $\overrightarrow{\alpha}.\overrightarrow{\varepsilon} = \overrightarrow{\varepsilon}.\overrightarrow{\alpha} = \overrightarrow{\alpha}$*, para todo* $\overrightarrow{\alpha}$*. Denotamos o* i*-ésimo termo da sequencia* $\overrightarrow{\alpha}$ *por* $(\overrightarrow{\alpha})_i$*.*

Definição 4.8. *Dizemos que uma sequencia finita de ações* $\overrightarrow{\beta}$ *é um* prefixo *de* $\overrightarrow{\alpha}$ *se existe uma sequencia não vazia* $\overrightarrow{\lambda}$ *tal que* $\overrightarrow{\alpha} = \overrightarrow{\beta}.\overrightarrow{\lambda}$*. Se* $\overrightarrow{\beta}$ *é um prefixo de* $\overrightarrow{\alpha}$*, escrevemos* $\overrightarrow{\beta} \subset \overrightarrow{\alpha}$*.*

Definição 4.9. *Para uma sequencia finita de ações* $\overrightarrow{\alpha}$*, escrevemos* $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} P'$ *para expressar que o processo* P *pode executar a sequencia* $\overrightarrow{\alpha}$ *e depois disto se comportar como* P'*. Além disso, escrevemos* $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$ *para expressar que o processo* P *pode terminar de maneira bem sucedida após executar a sequencia* $\overrightarrow{\alpha}$*.*

Definição 4.10. *Definimos o conjunto de execuções finitas possíveis de um processo P, denotado por $\overrightarrow{\mathcal{R}_f}(P)$, como $\overrightarrow{\mathcal{R}_f}(P) = \{\overrightarrow{\alpha} : P \stackrel{\vec{\alpha}}{\Rightarrow} \surd\}$.*

Nas lógicas para sistemas concorrentes que desenvolvemos no capítulo 7, de maneira análoga ao que ocorre em PDL tradicional, utilizamos semânticas que apenas levam em conta as execuções *finitas* possíveis dos processos, isto é, situações em que os processos terminam de forma bem sucedida. Então, apresentamos alguns resultados úteis a respeito de execuções finitas possíveis.

Definição 4.11. *Sejam R e S conjuntos de sequencias finitas de ações. Podemos definir as seguintes operações nestes conjuntos:*

1. $R \circ S = \{\overrightarrow{\alpha}.\overrightarrow{\beta} : \overrightarrow{\alpha} \in R \text{ e } \overrightarrow{\beta} \in S\}$;
2. $R \cup S = \{\overrightarrow{\alpha} : \overrightarrow{\alpha} \in R \text{ ou } \overrightarrow{\alpha} \in S\}$;
3. $R^0 = \{\overrightarrow{\varepsilon}\}$, $R^n = R \circ R^{n-1} (n \geq 1)$;
4. $R^* = \bigcup_{n \in \mathbb{N}} R^n$.

Lema 4.12. *Se $P \sim Q$, então $P \stackrel{\vec{\alpha}}{\Rightarrow} \surd$ se e somente se $Q \stackrel{\vec{\alpha}}{\Rightarrow} \surd$.*

Teorema 4.13. *Se $P \sim Q$, então $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$.*

4.2 CCS Estendido (XCCS)

Conforme foi mencionado na seção anterior, devido à definição de certa forma frouxa da semântica do processo **0**, que falha em diferenciar entre um *deadlock* e uma terminação bem sucedida, seu uso traria uma séria inconveniência à semântica dos formalismos lógicos iniciais apresentados no capítulo 7: suas semânticas não seriam totalmente composicionais. Este problema também afeta a possibilidade de incluirmos o operador de restrição nas linguagens dos programas destas lógicas. Além disto, como será visto no capítulo 7, o tratamento das constantes de CCS e suas equações definidoras dentro dos formalismos lógicos é bastante complexo.

Nesta seção, nosso objetivo é lidar com estes dois problemas. Para isto, estendemos a linguagem de CCS com novos operadores e um novo tipo de ação. Chamamos esta nova álgebra de processos de *CCS estendido* ou XCCS. Devido à definição refinada do processo nulo **0** em XCCS, podemos utilizá-lo em formalismos lógicos

em que os programas são baseados em XCCS, assim como o operador de restrição. Além disso, um dos novos operadores de XCCS, o operador de iteração, nos permite remover as constantes e seu tratamento elaborado da linguagem.

Em CCS, temos o conjunto de ações $\mathcal{A} = \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$. Em XCCS, denotamos este conjunto de ações por $\mathcal{A}_R$, o conjunto de *ações de execução*. Em XCCS, possuímos uma ação extra, além daquelas em $\mathcal{A}_R$, chamada de *ação de terminação* e denotada por END. Um processo em XCCS pode terminar de maneira bem sucedida apenas após executar a ação END e ele sempre termina de maneira bem sucedida após executar tal ação. Se um processo não pode executar nenhuma ação de execução e não pode terminar de maneira bem sucedida, ele é chamado de processo em *deadlock*.

Definição 4.14. *Em XCCS, especificações de processos pode ser construídas utilizando-se as seguintes operações:*

$$P ::= \mathbf{0} \mid END \mid \alpha.P \mid P_1; P_2 \mid P_1 + P_2 \mid P_1 | P_2 \mid P^* \mid P \backslash L,$$

com

$$\alpha ::= a \mid \overline{a} \mid \tau,$$

onde $a \in \mathcal{N}$ *e* $L \subseteq \mathcal{N}$.

0 é o processo nulo. Ele é um processo em *deadlock*, uma vez que ele é incapaz de executar qualquer ação de execução e de terminar de maneira bem sucedida. END é o processo que é incapaz de executar qualquer ação de execução, mas é capaz de terminar de maneira bem sucedida. O operador de *composição sequencial* (;) denota que o processo irá se comportar primeiramente como P_1 e, se e quando P_1 terminar de maneira bem sucedida, irá continuar se comportando como P_2. O operador de *iteração* (*) denota que o processo P é capaz de ser iterado zero ou mais vezes. Na tabela 4.2, apresentamos a semântica dos operadores de XCCS.

A partir da tabela 4.2, não é difícil notar que agora o processo nulo **0** denota apenas um processo em *deadlock*, enquanto o processo END denota terminação de maneira bem sucedida. A situação no CCS tradicional é diferente, uma vez que lá, em uma especificação da forma $\alpha.\mathbf{0}$, **0** está denotando um processo que terminou de forma bem sucedida, enquanto em uma especificação da forma $P + \mathbf{0}$, **0** está denotando um processo em *deadlock*. Já em XCCS, uma especificação da forma

Tabela 4.2: Relações de Transição de XCCS

$\alpha.P \xrightarrow{\alpha} P$	$END \xrightarrow{END} \surd$	$P^* \xrightarrow{END} \surd$	$\frac{P \xrightarrow{\alpha} P'}{P;Q \xrightarrow{\alpha} P';Q}$	$\frac{P \xrightarrow{END} \surd, Q \xrightarrow{\alpha} Q'}{P;Q \xrightarrow{\alpha} Q'}$
$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$	$\frac{Q \xrightarrow{\beta} Q'}{P+Q \xrightarrow{\beta} Q'}$	$\frac{P \xrightarrow{\alpha} P'}{P\|Q \xrightarrow{\alpha} P'\|Q}$	$\frac{Q \xrightarrow{\beta} Q'}{P\|Q \xrightarrow{\beta} P\|Q'}$	$\frac{P \xrightarrow{\lambda} P', Q \xrightarrow{\overline{\lambda}} Q'}{P\|Q \xrightarrow{\tau} P'\|Q'}$
$\frac{P \xrightarrow{\alpha} P'}{P^* \xrightarrow{\alpha} P';P^*}$	$\frac{P \xrightarrow{\alpha} P', \alpha \notin L \cup \overline{L}}{P\backslash L \xrightarrow{\alpha} P'\backslash L}$	$\frac{P \xrightarrow{END} \surd, Q \xrightarrow{END} \surd}{P;Q \xrightarrow{END} \surd}$	$\frac{P \xrightarrow{END} \surd}{P+Q \xrightarrow{END} \surd}$	$\frac{Q \xrightarrow{END} \surd}{P+Q \xrightarrow{END} \surd}$
	$\frac{P \xrightarrow{END} \surd, Q \xrightarrow{END} \surd}{P\|Q \xrightarrow{END} \surd}$		$\frac{P \xrightarrow{END} \surd}{P\backslash L \xrightarrow{END} \surd}$	

$\alpha.\mathbf{0}$ denota que o processo executa a ação α e então entra em *deadlock*, enquanto uma especificação da forma $\alpha.END$ denota que o processo executa a ação α e então termina de maneira bem sucedida.

Em [21] e [24], Milner usa uma construção sintática engenhosa para definir uma forma de composição sequencial. Ela é ligeiramente diferente da forma apresentada na tabela 4.2 e não é um operador primitivo. Ele também utiliza a notação ; para a sua construção, mas nós a denotamos por :, de forma que possamos diferenciar facilmente entre a sua construção e a nossa. A construção de Milner depende de uma série de coisas. Em primeiro lugar, precisamos considerar um novo nome $z \notin \mathcal{N}$. Em segundo lugar, todo processo precisa executar a ação $\overline{z}$ como sua última ação antes da terminação e não pode executar z ou $\overline{z}$ em qualquer outro ponto da execução. Em terceiro lugar, precisamos executar substituições sintáticas de nomes em processos, onde $P[b/a]$ denota a substituição de toda ocorrência de a ($\overline{a}$) em P por b ($\overline{b}$). Então, composição sequencial é definida da seguinte forma:

$$P : Q = (P[a/z] \mid a.Q)\backslash\{a\},$$

onde a precisa ser um nome que não ocorre nem em P nem em Q.

A principal diferença entre as duas formas de composição sequencial é que, como as tabelas 4.1 e 4.2 mostram, $\overrightarrow{\mathcal{R}_f}(P;Q) = \overrightarrow{\mathcal{R}_f}(P) \circ \overrightarrow{\mathcal{R}_f}(Q)$, enquanto $\overrightarrow{\mathcal{R}_f}(P : Q) = \overrightarrow{\mathcal{R}_f}(P) \circ \{\tau\} \circ \overrightarrow{\mathcal{R}_f}(Q)$. Este τ extra estaria presente também nas execuções finitas de um processo P^*, uma vez que utilizamos composição sequencial para definir a semântica do operador de iteração (tabela 4.2). Estes τ's extras aparecendo entre as execuções finitas dos subprocessos seria um complicador para as semânticas das nossas lógicas, uma vez que algumas validades lógicas intuitivas seriam falsas. Uma vez que já estamos introduzindo o processo END para resolver os problemas anteriores com o processo nulo, não há razão porque também não deveríamos usá-lo para

construir uma forma mais simples e mais conveniente de composição sequencial e uma forma simples de iteração, como é feito na tabela 4.2.

Agora, precisamos fazer alguns pequenos ajustes na noção de bissimulação forte e na Lei de Expansão.

Definição 4.15. *Seja $\mathcal{P}$ o conjunto de todas as especificações de processos possíveis. Um conjunto $Z \subseteq \mathcal{P} \times \mathcal{P}$ é uma* bissimulação forte *se $(P, Q) \in Z$ implica, para todo $\alpha \in \mathcal{A}_R$, em*

- *Se $P \xrightarrow{\alpha} P'$ e $P' \in \mathcal{P}$, então existe $Q' \in \mathcal{P}$ tal que $Q \xrightarrow{\alpha} Q'$ e $(P', Q') \in Z$;*
- *Se $Q \xrightarrow{\alpha} Q'$ e $Q' \in \mathcal{P}$, então existe $P' \in \mathcal{P}$ tal que $P \xrightarrow{\alpha} P'$ e $(P', Q') \in Z$;*

e

- *$P \xrightarrow{END} \surd$ se e somente se $Q \xrightarrow{END} \surd$.*

A definição de bissimilaridade forte é análoga à definição 4.4, utilizando a nova noção de bissimulação forte enunciada acima.

Na presença do operador de iteração, uma versão mais fraca da Lei de Expansão é suficiente para as nossas necessidades.

Teorema 4.16 (Lei de Expansão (EL))**.** *Seja $P = P_1 \mid P_2$, onde P é irrestrito* e $\mid$ não ocorre em P_1 e P_2. *Então*

$$P \sim \sum_{P_1 \xrightarrow{\alpha} P_1'} \alpha.(P_1' \mid P_2) + \sum_{P_2 \xrightarrow{\beta} P_2'} \beta.(P_1 \mid P_2') + \sum_{R \in A_\tau} \tau.R + E_P,$$

onde $A_\tau = \{(P_1' \mid P_2') : P_1 \xrightarrow{a} P_1'$ e $P_2 \xrightarrow{\overline{a}} P_2'$, para algum $a \in \mathcal{N}\} \cup \{(P_1' \mid P_2') : P_1 \xrightarrow{\overline{a}} P_1'$ e $P_2 \xrightarrow{a} P_2'$, para algum $a \in \mathcal{N}\}$ e $E_P = END$, se $P_1 \xrightarrow{END} \surd$ e $P_2 \xrightarrow{END} \surd$ ou $E_P = \mathbf{0}$, caso contrário. Novamente, denotamos o lado direito desta bissimilaridade por $Exp(P)$.

Finalmente, devido à presença da ação END, precisamos fazer um pequeno ajuste na definição da composição $R \circ S$ de dois conjuntos R e S de sequencias finitas de ações.

Definição 4.17. *Seja $\natural(\overrightarrow{\alpha}) = \overrightarrow{\lambda}$, se $\overrightarrow{\alpha} = \overrightarrow{\lambda}.END$ e $\natural(\overrightarrow{\alpha}) = \overrightarrow{\alpha}$, caso contrário. Então,*

$$R \circ S = \{\natural(\overrightarrow{\alpha}).\overrightarrow{\beta} : \overrightarrow{\alpha} \in R \text{ e } \overrightarrow{\beta} \in S\}$$

Definição 4.18. *Dizemos que uma relação $\cong$ entre processos é uma* congruência *se ela é uma relação de equivalência e ela é preservada por todos os operadores de XCCS, isto é, se $P \cong Q$, então $\alpha.P \cong \alpha.Q$, $P + R \cong Q + R$ e assim por diante.*

Dizemos que a restrição $\backslash L$ *liga* os nomes $a \in L$. Dizemos que um nome é *ligado* em P se ele ocorre dentro do escopo de uma restrição que o liga. Caso contrário, dizemos que um nome é *livre*.

Definição 4.19. *Uma substituição sintática de um* nome ligado *por um* nome novo *(um nome que não ocorre na especificação de processo) em um conjunto de restrição L e em toda ocorrência do nome dentro do escopo da restrição correspondente $\backslash L$ é chamada de* conversão alfa.

Definição 4.20. Congruência de Restrição*, ou* r-congruência*, denotada por $\equiv_r$, é uma relação entre processos definida pelo seguinte conjunto de axiomas e regras, onde $n(P)$ denota o conjunto de nomes que ocorre em P como parte tanto de ações de entrada quanto de saída.*

1. *É uma congruência;*
2. *É fechada sob conversão alfa;*
3. *Comutatividade de $+$ e $|$;*
4. $\mathbf{0}\backslash L \equiv_r \mathbf{0}$*;*
5. $END\backslash L \equiv_r END$*;*
6. *Se $\alpha \notin L \cup \overline{L}$,* $(\alpha.P)\backslash L \equiv_r \alpha.(P\backslash L)$*;*
7. *Se $\alpha \in L \cup \overline{L}$,* $(\alpha.P)\backslash L \equiv_r \mathbf{0}$*;*
8. $(P;Q)\backslash L \equiv_r (P\backslash L);(Q\backslash L)$*;*
9. $(P + Q)\backslash L \equiv_r (P\backslash L) + (Q\backslash L)$*;*
10. *Se* $n(P) \cap (L \cup \overline{L}) = \emptyset$, $P|(Q\backslash L) \equiv_r (P|Q)\backslash L$*;*
11. $(P^*)\backslash L \equiv_r (P\backslash L)^*$*;*
12. $P\backslash L\backslash M \equiv_r P\backslash(L \cup M)$*;*
13. *Se* $n(P) \cap (L \cup \overline{L}) = \emptyset$, $P\backslash L \equiv_r P$.

Definição 4.21. *Dizemos que um processo está em* forma r-externa *se ele tem a forma $P\backslash L$, onde P é irrestrito.*

Teorema 4.22. *Todo processo é r-congruente a um processo em forma r-externa e todo processo sem ocorrência do operador $|$ é r-congruente a um processo irrestrito.*

Demonstração. A prova segue da definição 4.20. □

Teorema 4.23. *Se $P \equiv_r Q$, então $P \sim Q$.*

Demonstração. A prova segue da tabela 4.2 e da definição 4.4. □

4.3 O π-Cálculo

O π-Cálculo é uma álgebra de processos bem conhecida, proposta por Milner, Parrow e Walker [25]. Ele é uma extensão do CCS de Milner [21] que é capaz de descrever não apenas não determinismo e concorrência, mas também *mobilidade* de processos. Para uma introdução ampla ao π-Cálculo, [26] pode ser consultado.

4.3.1 Linguagem e Semântica

O principal acréscimo da linguagem do π-Cálculo em relação à linguagem do CCS (ou do XCCS) está no fato de que as mensagens enviadas pelos canais entre os processos não são mais meros sinais, mas sim mensagens com conteúdo. No π-Cálculo, o conteúdo de uma mensagem enviada através de um canal é sempre um nome do conjunto $\mathcal{N}$. Uma vez que a linguagem do XCCS é mais apropriada para o nosso objetivo de desenvolver formalismos lógicos para sistemas concorrentes, apresentamos o π-Cálculo partindo desta linguagem.

Definição 4.24. *Em nossa apresentação do π-Cálculo, especificações de processos podem ser construídas utilizando-se as seguintes operações:*

$$P ::= \mathbf{0} \mid END \mid \alpha.P \mid P_1; P_2 \mid P_1 + P_2 \mid P_1|P_2 \mid P^* \mid (\nu a)P,$$

com

$$\alpha ::= a(x) \mid \overline{a}\langle x\rangle \mid \overline{a}\langle \nu x\rangle \mid \tau,$$

onde $a, x \in \mathcal{N}$.

Usualmente o π-Cálculo é apresentado com um operador de replicação (!), que denota a habilidade de um processo de gerar múltiplas cópias de si mesmo. Neste trabalho, nos atemos ao π-Cálculo sem replicação e a questão de se é possível mantermos os resultados que apresentamos no capítulo 7 na presença do operador de replicação permanece em aberto. Deixamos esta questão para um trabalho futuro, conforme explicado no capítulo 8.

A única diferença significativa na sintaxe do π-Cálculo em relação à sintaxe de XCCS, com a exceção da já mencionada diferença de que as mensagens do π-Cálculo possuem conteúdo, está no operador de restrição. No π-Cálculo, o operador de restrição (νa) denota que o canal a só é acessível no interior de P (o escopo de a é P).

A ação $a(x)$, chamada *ação de entrada*, denota que o processo recebe um nome através do canal rotulado por a e o nome x marca, em P, os lugares onde o nome recebido deve ser colocado. As ações $\overline{a}\langle x\rangle$, chamada *ação de saída livre*, e $\overline{a}\langle \nu x\rangle$, chamada *ação de saída ligada*, ambas denotam que o processo envia o nome x através do canal rotulado por a. A diferença entre as duas é que, em $\overline{a}\langle \nu x\rangle$, x é um nome restringido, com esta ação sendo uma abreviação de $(\nu x)\overline{a}\langle x\rangle$. Finalmente, τ denota a ação silenciosa. Definimos a ação de saída ligada como uma uma ação primitiva porque, como é mostrado abaixo, sob certas circunstâncias, a única forma de restrição que é necessária é a oferecida por saídas ligadas.

Dizemos que as ações $a(x)$ e $\overline{a}\langle \nu x\rangle$ e a restrição (νx) *ligam* o nome x, chamando-os de *ligadores*. Dizemos que um nome é *ligado* em P se ele ocorre dentro do escopo de uma ação ou restrição que o liga. Caso contrário, dizemos que o nome é *livre* em P. Denotamos por $f(P)$ o conjunto dos nomes livres em P e por $b(P)$ o conjunto dos nomes ligados em P. De modo análogo, denotamos por $f(\alpha)$ e $b(\alpha)$ os nomes livres e ligados em uma ação α. Nas ações $a(x)$, $\overline{a}\langle x\rangle$ e $\overline{a}\langle \nu x\rangle$, a é chamado de *sujeito*, denotado por $s(\alpha)$, onde α é a ação, e x é chamado de *objeto*, denotado por $o(\alpha)$. A ação τ não possui nem sujeito nem objeto.

Definimos as noções de congruência entre processos e de conversão alfa de forma análoga às definições da seção anterior.

Definição 4.25. Congruência estrutural*, denotada por $\equiv$, é uma relação entre processos definida pelo seguinte conjunto de axiomas e regras:*

1. *É uma congruência;*
2. *É fechada sob conversão alfa;*
3. *Comutatividade de $+$ e $|$;*
4. *Se $a \neq x$, $(\nu x)\overline{a}\langle x\rangle.P \equiv \overline{a}\langle \nu x\rangle.P$;*
5. *$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$;*
6. *Se $x \notin f(P)$, $(\nu x)P \equiv P$.*

Definição 4.26. *Dizemos que um processo P é* limpo *se nenhum nome aparece tanto*

livre quanto ligado em P e nenhum nome é ligado por mais de um ligador. Dizemos que um processo é irrestrito *se ele não possui ocorrências do operador ν. Dizemos que um processo limpo está em* forma ν-prefixo *se ele tem a forma $(\nu x_1)\ldots(\nu x_n)P$, $n \geq 0$, onde P é irrestrito. Finalmente, dizemos que um processo limpo está em* forma ν-padrão *se as únicas ocorrências do operador ν são dentro de prefixos de saída ligada.*

Na tabela 4.3, apresentamos a semântica para os operadores do π-Cálculo. Esta semântica é chamada de *semântica tardia*[1]. Para mais detalhes sobre esta e outras semânticas, [26] pode ser consultado.

$\frac{P' \equiv P, P \xrightarrow{\alpha} Q, Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$	$\alpha.P \xrightarrow{\alpha} P$	$END \overset{END}{\rightarrow} \surd$	$P^* \overset{END}{\rightarrow} \surd$
$\frac{P \xrightarrow{\alpha} P'}{P;Q \xrightarrow{\alpha} P';Q}$	$\frac{P \overset{END}{\rightarrow} \surd, Q \xrightarrow{\alpha} Q'}{P;Q \xrightarrow{\alpha} Q'}$	$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$	$\frac{P \xrightarrow{\alpha} P', b(\alpha) \cap f(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$
$\frac{P \xrightarrow{a(x)} P', Q \xrightarrow{\overline{a}\langle u \rangle} Q'}{P \mid Q \xrightarrow{\tau} P'\{u/x\} \mid Q'}$	$\frac{P \xrightarrow{a(x)} P', Q \xrightarrow{\overline{a}\langle \nu u \rangle} Q'}{P \mid Q \xrightarrow{\tau} (\nu u)(P'\{u/x\} \mid Q')}$	$\frac{P \xrightarrow{\alpha} P'}{P^* \xrightarrow{\alpha} P'; P^*}$	$\frac{P \xrightarrow{\alpha} P', x \notin \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}$
$\frac{P \overset{END}{\rightarrow} \surd, Q \overset{END}{\rightarrow} \surd}{P;Q \overset{END}{\rightarrow} \surd}$	$\frac{P \overset{END}{\rightarrow} \surd}{P+Q \overset{END}{\rightarrow} \surd}$	$\frac{Q \overset{END}{\rightarrow} \surd}{P+Q \overset{END}{\rightarrow} \surd}$	$\frac{P \overset{END}{\rightarrow} \surd, Q \overset{END}{\rightarrow} \surd}{P \mid Q \overset{END}{\rightarrow} \surd}$
$\frac{P \overset{END}{\rightarrow} \surd}{(\nu x)P \overset{END}{\rightarrow} \surd}$			

Tabela 4.3: Relações de Transição do π-Cálculo

Definição 4.27. *Dizemos que um conjunto de nomes é novo para um processo P se nenhum nome deste conjunto ocorre em P.*

Definição 4.28. *Seja $\mathcal{P}$ o conjunto de todas as especificações de processos possíveis. Uma* bissimulação tardia *é uma relação binária* simétrica *$Z \subseteq \mathcal{P} \times \mathcal{P}$ tal que*

1. *Se $(P, Q) \in Z$ e $P \xrightarrow{\alpha} P'$, onde $b(\alpha)$ é novo para P e Q, então*
   (a) *Se $\alpha = a(x)$, então existe $Q' \in \mathcal{P}$ tal que $Q \xrightarrow{\alpha} Q'$ e, para todo $u \in \mathcal{N}$, $(P'\{u/x\}, Q'\{u/x\}) \in Z$;*
   (b) *Se α não é uma ação de entrada, então existe $Q' \in \mathcal{P}$ tal que $Q \xrightarrow{\alpha} Q'$ e $(P', Q') \in Z$;*
2. *Se $(P, Q) \in Z$ e $P \overset{END}{\rightarrow} \surd$, então $Q \overset{END}{\rightarrow} \surd$.*

A razão pela qual as únicas ações de execução α que precisam ser consideradas são as que satisfazem a condição de que $b(\alpha)$ é novo para P e Q é explicada em detalhes em [26].

[1] late semantics, em inglês

Definição 4.29. *Duas especificações de processos P e Q são* tardiamente bissimilares *(ou simplesmente* bissimilares*), denotado por $P \dot{\sim} Q$, se existe uma bissimulação tardia Z tal que $(P, Q) \in Z$. No π-Cálculo, bissimilaridade é uma relação de equivalência, mas não é uma congruência.*

Teorema 4.30. *Se $P \equiv Q$, então $P \dot{\sim} Q$.*

Teorema 4.31 (Lei de Expansão (EL) [26])**.** *Seja $P = P_1 \mid P_2$, onde P é limpo e irrestrito e $\mid$ não ocorre em P_1 e P_2. Então*

$$P \dot{\sim} \sum_{P_1 \xrightarrow{\alpha} P_1'} \alpha.(P_1' \mid P_2) + \sum_{P_2 \xrightarrow{\beta} P_2'} \beta.(P_1 \mid P_2') + \sum_{R \in A_\tau} \tau.R + E_P,$$

onde $A_\tau = \{(P_1'\{u/x\} \mid P_2') : P_1 \overset{a(x)}{\rightarrow} P_1'$ e $P_2 \overset{\overline{a}\langle u\rangle}{\rightarrow} P_2'$, para algum $a \in \mathcal{N}\} \cup \{(P_1' \mid P_2'\{u/x\}) : P_1 \overset{\overline{a}\langle u\rangle}{\rightarrow} P_1'$ e $P_2 \overset{a(x)}{\rightarrow} P_2'$, para algum $a \in \mathcal{N}\}$ e $E_P = END$, se $P_1 \overset{END}{\rightarrow} \surd$ e $P_2 \overset{END}{\rightarrow} \surd$ ou $E_P = \mathbf{0}$, caso contrário. Denotamos o lado direito desta bissimilaridade por $Exp(P)$.

Definição 4.32. ν-bissimilaridade*, denotada por $P \overset{\nu}{\sim} Q$, é uma relação entre processos definida pelo seguinte conjunto de axiomas e regras, onde $x \notin \alpha$ denota que x não é nem o sujeito nem o objeto da ação α:*

1. *Se $P \equiv Q$, então $P \overset{\nu}{\sim} Q$;*
2. *$(\nu x)\mathbf{0} \overset{\nu}{\sim} \mathbf{0}$;*
3. *$(\nu x)END \overset{\nu}{\sim} END$;*
4. *Se $x \notin \alpha$, $(\nu x)\alpha.P \overset{\nu}{\sim} \alpha.(\nu x)P$;*
5. *Se $x = s(\alpha)$, $(\nu x)\alpha.P \overset{\nu}{\sim} \mathbf{0}$;*
6. *$(\nu x)(P; Q) \overset{\nu}{\sim} (\nu x)P; (\nu x)Q$;*
7. *$(\nu x)(P + Q) \overset{\nu}{\sim} (\nu x)P + (\nu x)Q$;*
8. *Se $x \notin f(P)$, $P|(\nu x)Q \overset{\nu}{\sim} (\nu x)(P|Q)$;*
9. *$(\nu x)P^* \overset{\nu}{\sim} ((\nu x)P)^*$.*

Segue da tabela 4.3 e da definição 4.29 que os itens 2 a 9 são realmente bissimilaridades. Logo, a relação de ν-bissimilaridade é um subconjunto da relação de bissimilaridade. ν-bissimilaridade é uma relação conveniente porque possui uma axiomatização simples e é suficiente para as nossas necessidades neste trabalho.

Teorema 4.33. *Todo processo é estruturalmente congruente a um processo limpo. Todo processo limpo é ν-bissimilar a um processo em forma ν-prefixo. Finalmente, todo processo limpo sem ocorrências do operador $\mid$ é ν-bissimilar a um processo em forma ν-padrão.*

4.3.2 Sequencias de Ações e Execuções Possíveis

Podemos definir noções de conversão alfa e de nomes ligados em uma sequencia de ações e de sequencias de ações *limpas*, em analogia com as definições 4.19 e 4.26. Também podemos estender nossa notação e escrever $b(\overrightarrow{\gamma})$ e $f(\overrightarrow{\gamma})$ para os conjuntos de nomes ligados e livres na sequencia $\overrightarrow{\gamma}$. Se uma sequencia de ações $\overrightarrow{\gamma}$ pode ser alfa-convertida em uma sequencia $\overrightarrow{\sigma}$, escrevemos $\overrightarrow{\gamma} \equiv_\alpha \overrightarrow{\sigma}$. Não é difícil perceber que, se $P \stackrel{\vec{\gamma}}{\Rightarrow} \surd$, então $P \stackrel{\vec{\sigma}}{\Rightarrow} \surd$, onde $\overrightarrow{\gamma} \equiv_\alpha \overrightarrow{\sigma}$ e $\overrightarrow{\sigma}$ é limpa. Seja $S(P)$ o conjunto de todas as sequencias limpas $\overrightarrow{\sigma}$ tais que $P \stackrel{\vec{\sigma}}{\Rightarrow} \surd$. Então, também não é difícil perceber que, se estabelecermos a convenção de que $\overrightarrow{\sigma} \equiv_\alpha \overrightarrow{\sigma}$, $\equiv_\alpha$ é uma relação de equivalência para os elementos do conjunto $S(P)$.

Definição 4.34. *Definimos o conjunto de execuções finitas possíveis de um processo P, denotado por $\overrightarrow{\mathcal{R}_f}(P)$, como o conjunto quociente $\overrightarrow{\mathcal{R}_f}(P) = S(P)/\equiv_\alpha$. Se $\overrightarrow{\gamma} \in S(P)$, então $[\overrightarrow{\gamma}] \in \overrightarrow{\mathcal{R}_f}(P)$ denota sua classe de equivalência.*

Definição 4.35. *Sejam $\overrightarrow{\alpha}$ e $\overrightarrow{\sigma}$ duas sequencias de ações e sejam P e Q duas especificações de processos. Se $b(\overrightarrow{\alpha})$ é novo para Q e $b(\overrightarrow{\sigma})$ é novo para P, escrevemos $(\overrightarrow{\alpha}, \overrightarrow{\sigma}) \bowtie (P, Q)$.*

Definição 4.36. *Seja $T = \overrightarrow{\mathcal{R}_f}(P)$ e $U = \overrightarrow{\mathcal{R}_f}(Q)$ e seja $\natural(\overrightarrow{\alpha}) = \overrightarrow{\lambda}$, onde $\overrightarrow{\alpha} = \overrightarrow{\lambda}.END$. Podemos definir as seguintes operações nos conjuntos T e U:*

- $T \circ U = \{[\natural(\overrightarrow{\alpha}).\overrightarrow{\beta}] : [\overrightarrow{\alpha}] \in T, [\overrightarrow{\beta}] \in U \text{ e } (\overrightarrow{\alpha}, \overrightarrow{\beta}) \bowtie (P, Q)\}$;
- $T \cup U = \{[\overrightarrow{\alpha}] : [\overrightarrow{\alpha}] \in T \text{ ou } [\overrightarrow{\alpha}] \in U\}$;
- $R^0 = \{[\overrightarrow{\varepsilon}]\}, R^n = R \circ R^{n-1} (n \geq 1)$ e $R^* = \bigcup_{n \in \mathbb{N}} R^n$.

Lema 4.37. *Se $P \dot{\sim} Q$ e $b(\overrightarrow{\alpha})$ é novo para P e Q, então $P \stackrel{\vec{\alpha}}{\Rightarrow} \surd$ se e somente se $Q \stackrel{\vec{\alpha}}{\Rightarrow} \surd$.*

Teorema 4.38. *Se $P \dot{\sim} Q$, então $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$.*

Capítulo 5

Lógicas Modais Híbridas e Propriedades Globais de Grafos

"Eu não estava mentindo. Estava escrevendo ficção com a boca." - Homer Simpson

Neste capítulo, analisamos o problema de como descrever e verificar de maneira eficiente as quatro propriedades globais de grafos apresentadas no capítulo 3 utilizando lógicas modais. Para esta tarefa, consideramos as vantagens e desvantagens da lógica modal básica, da lógica híbrida, da lógica híbrida com quantificador local, da lógica modal graduada e de uma versão híbrida da lógica temporal CTL*.

O conteúdo deste capítulo tem como base os artigos [27], [28] e [29] (reproduzido no apêndice A). As provas de todos os resultados apresentados neste capítulo podem ser consultadas no apêndice A.

5.1 Introdução

Grafos estão entre as estruturas mais utilizadas e mais estudadas em Ciência da Computação [1]. Nesta disciplina, muitos conceitos importantes admitem uma representação através de grafos e, em algumas ocasiões, grafos estão presentes no próprio núcleo do modelo de computação utilizado. Isto acontece, por exemplo, na área de sistemas distribuídos [2, 3], onde o modelo subjacente de computação é construído sobre um grafo. Além deste papel central, grafos também são importantes em sistemas distribuídos como ferramentas para a descrição de problemas de distribuição de recursos, escalonamento de tarefas, detecção de *deadlocks*, entre outros.

O caso de sistemas distribuídos é particularmente atraente porque ilustra bem dois níveis diferentes em que propriedades de grafos são descritas. Um é o nível local, que engloba propriedades que são satisfeitas em vértices ou vizinhanças de vértices. O outro nível é o global e consiste de propriedades que são satisfeitas no grafo como um todo, tais como aciclicidade e conectividade.

A teoria de grafos fornece muitas ferramentas para a descrição de tais problemas e apresenta muitos algoritmos eficientes para resolvê-los. Entretanto, existe uma distinção importante entre os dois lados desta questão. No lado da "descrição", grafos fornecem um grande nível de generalidade, permitindo a descrição de problemas muito diferentes no mesmo arcabouço[1] simples. Porém no lado da "solução", cada problema de grafos precisa ser resolvido e cada propriedade de grafos precisa ser testada com um método específico que em geral não pode ser generalizado para outros problemas ou propriedades diferentes.

Um arcabouço lógico, por outro lado, pode prover este nível de generalidade. Em uma linguagem intuitiva, isto pode ser colocado da seguinte maneira. Considere uma lógica $\mathbb{L}$ com suas fórmulas e as estruturas onde estas fórmulas são semanticamente avaliadas. Nós precisamos ser capazes de responder as seguintes perguntas:

1. Podemos representar um grafo como uma estrutura para $\mathbb{L}$?

2. Podemos representar as propriedades de grafos que queremos verificar como $\mathbb{L}$-fórmulas?

3. $\mathbb{L}$ possui métodos de inferência decidíveis para verificar se uma fórmula é satisfeita (ou válida) em uma estrutura?

Se as respostas para todas estas perguntas forem positivas, então podemos usar os métodos de inferência da lógica para verificar cada propriedade de grafos que quisermos, desde que esta propriedade possa ser expressa como uma $\mathbb{L}$-fórmula. Naturalmente, ainda há uma quarta pergunta que precisa ser respondida:

4. O método lógico é tão eficiente (em termos de complexidade computacional) em testar uma dada propriedade de grafos quanto o método oriundo da teoria de grafos?

[1] framework, em inglês

Para satisfazer a primeira pergunta, escolhemos trabalhar com a família de lógicas modais. Uma razão muito forte para escolhermos lógicas modais para esta tarefa, ao invés de qualquer outra lógica, está no fato de que as fórmulas das lógicas modais são avaliadas em estruturas que são essencialmente grafos, o que as torna uma escolha muito natural.

Neste capítulo, analisamos como podemos expressar e verificar de maneira eficiente algumas propriedades globais de grafos utilizando lógicas modais. Isto envolve duas questões: a primeira é determinar se cada uma das linguagens modais consideradas possui poder expressivo suficiente para descrever as propriedades de grafos em que estamos interessados; a segunda é quão complexo (computacionalmente) é utilizar estas lógicas para realmente testar se um dado grafo possui uma propriedade desejada.

Um trabalho semelhante é apresentado em [30]. Naquele trabalho, o interesse também estava em como usar lógicas modais para expressar propriedades globais de grafos. Entretanto, em [30], apenas a lógica básica para grafos (apresentada na próxima sessão) era considerada e apenas conectividade e aciclicidade foram analisadas. Além disto, o foco daquele trabalho era em como construir axiomatizações para classes de grafos com estas propriedades globais, enquanto nosso foco está em como encontrar fórmulas que expressem cada uma destas propriedades e que possam ser eficientemente utilizadas para testar se um grafo as satisfaz.

O restante deste capítulo é organizado da seguinte maneira. Na seção 5.2, apresentamos uma lógica modal simples apropriada para a descrição de propriedades de grafos e investigamos se algumas propriedades conhecidas de grafos podem ou não ser expressas por fórmulas em sua linguagem. Analisamos conectividade, aciclicidade, a propriedade hamiltoniana e a propriedade euleriana. Na seção 5.3, estendemos a lógica modal da seção anterior com nominais, obtendo uma lógica híbrida, e a utilizamos para expressar algumas destas propriedades de grafos. Na seção 5.4, introduzimos uma versão híbrida da lógica temporal CTL*, que é uma lógica muito expressiva, e a utilizamos para verificar a propriedade hamiltoniana de uma maneira mais eficiente do que a utilizada na seção anterior. Na seção 5.5, estendemos a lógica híbrida da seção 5.3 com o operador $\downarrow$ e mostramos que podemos verificar a propriedade hamiltoniana com complexidade ótima (NP) nesta lógica.

Na seção 5.6, consideramos a propriedade euleriana de duas maneiras diferentes. Primeiramente, desenvolvemos um método genérico para expressar propriedades relacionadas a arestas em lógicas híbridas e o utilizamos para descrever a propriedade euleriana. Posteriormente, expressamos uma condição necessária e suficiente para que a propriedade euleriana seja satisfeita em um grafo utilizando uma lógica modal graduada.

É importante notar que, como estamos lidando com a questão de verificação computacional, todos os grafos com que trabalhamos neste capítulo são necessariamente finitos.

5.2 Lógica Básica para Grafos

Nesta seção, definimos uma lógica modal com dois operadores modais: $\Diamond$ e $\Diamond^+$. Ela é chamada de *lógica básica para grafos*.

Definição 5.1. *A linguagem da* lógica básica para grafos *é uma linguagem modal que consiste de um conjunto enumerável* Φ *de símbolos proposicionais, os operadores booleanos* $\neg$ *e* $\wedge$*, a constante* $\top$ *e dois operadores modais (ou modalidades):* $\Diamond$ *e* $\Diamond^+$*. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond\varphi \mid \Diamond^+\varphi,$$

onde $p \in \Phi$*.*

A modalidade $\Box^+$ é definida de forma que $\Box^+\varphi \equiv \neg\Diamond^+\neg\varphi$. Além disto, para tornar a linguagem mais elegante, introduzimos também abreviações para o fecho transitivo-reflexivo: $\Diamond^*\varphi = \varphi \vee \Diamond^+\varphi$ e $\Box^*\varphi = \neg\Diamond^*\neg\varphi$.

As fórmulas desta lógica são avaliadas nas estruturas e modelos de Kripke usuais. Conforme vimos no capítulo 3, uma estrutura de Kripke é essencialmente um grafo. Isto confirma a afirmação feita na seção anterior de que lógicas modais são uma escolha muito natural para a presente tarefa.

Para as definições apropriadas de satisfazibilidade das fórmulas desta lógica, o capítulo 2 pode ser consultado.

Neste capítulo, queremos encontrar uma fórmula modal ϕ (para cada propriedade), tal que um grafo G possui a propriedade desejada se e somente se $G \Vdash \phi$.

Para cada lógica modal que consideramos para esta tarefa, existem duas questões envolvidas. A primeira é determinar se a linguagem modal possui poder expressivo suficiente para descrever as propriedades de grafos que queremos. Caso a resposta seja negativa, então precisamos buscar lógicas com maior poder expressivo. Caso a resposta seja positiva e nós consigamos encontrar tal fórmula, então precisamos calcular quão complexo (computacionalmente) é utilizar os mecanismos de inferência da lógica para realmente testar, usando a fórmula que encontramos, se um dado grafo possui a propriedade desejada.

Como para verificar se um dado grafo possui uma propriedade desejada precisamos ser capazes de determinar se $\mathcal{F} \Vdash \phi$, onde $\mathcal{F}$ é uma estrutura de Kripke e ϕ é a fórmula que descreve a propriedade, a complexidade desta verificação não é dada diretamente pelo problema da verificação de modelo, mas sim por um problema relacionado, o *problema da verificação de estrutura*.

Definição 5.2. *O* problema da verificação de estrutura *consiste de, dada uma fórmula ϕ e uma estrutura finita $\mathcal{F}$, determinar se $\mathcal{F} \Vdash \phi$.*

Conforme estudado no capítulo 2, temos as seguintes complexidades computacionais para a lógica básica para grafos.

Teorema 5.3 ([10])**.** *O problema da satisfazibilidade e o problema da validade para a lógica básica para grafos são EXPTEMPO-Completos no comprimento da fórmula.*

Teorema 5.4. *O problema da verificação de modelo para a lógica básica para grafos é polinomial (linear) no produto do tamanho do modelo e do comprimento da fórmula.*

Podemos determinar um limite superior simples para a complexidade do problema da verificação de estrutura baseado na complexidade do problema da verificação de modelo correspondente. Temos que $\mathcal{F} \Vdash \phi$ se e somente se $S_{\mathcal{M}}(\neg\phi) = \emptyset$ para todo modelo $\mathcal{M}$ de $\mathcal{F}$. Então, um método algorítmico de verificar se $\mathcal{F} \Vdash \phi$ consiste em aplicar o algoritmo de verificação de modelo a todos os pares $(\neg\phi, \mathcal{M})$, onde $\mathcal{M}$ é um modelo de $\mathcal{F}$.

Portanto, se FC é a complexidade do problema da verificação de estrutura e MC é a complexidade do problema da verificação de modelo, temos

$$FC = O(2^{|p|\times n} \times MC),$$

onde $|p|$ é o número de símbolos proposicionais distintos que ocorrem em uma dada fórmula ϕ e n é o número de mundos em $\mathcal{F}$. Precisamos aplicar o algoritmo de verificação de modelo para todo modelo $\mathcal{M}$ da estrutura dada $\mathcal{F}$. Cada símbolo proposicional p que aparece em ϕ pode receber 2^n possíveis valorações $\mathbf{V}(p)$.

Teorema 5.5. *O problema da verificação de estrutura para a lógica básica para grafos é polinomial (linear) no comprimento da fórmula e EXPTEMPO no tamanho da estrutura e no número de símbolos proposicionais distintos que ocorrem na fórmula.*

Demonstração. Este resultado segue diretamente da discussão acima. □

Deve-se notar que este cálculo da complexidade do problema da verificação de estrutura é apenas um limite superior geral e que ele pode possivelmente ser reduzido em algumas situações concretas.

Vamos agora analisar os limites do poder expressivo da lógica básica para grafos a partir dos resultados da seção 2.1.3.

Teorema 5.6. *Conectividade fraca e forte não podem ser definidas na lógica básica para grafos.*

Demonstração. A união disjunta de grafos conexos não é um grafo conexo. Pelo corolário 2.20, como conectividade não é preservada por uniões disjuntas, ela não pode ser definida na lógica básica para grafos. □

Teorema 5.7. *Aciclicidade não pode ser definida na lógica básica para grafos.*

Demonstração. Podemos tomar uma estrutura $\mathcal{F} = (W, R)$ onde $W = \mathbb{N}$ e $R = \{\langle i, i+1\rangle, i \in \mathbb{N}\}$ e uma estrutura $\mathcal{F}' = (W', R')$ onde $W' = \{O, E\}$ e $R' = \{\langle O, E\rangle, \langle E, O\rangle\}$. Se definirmos f como $f(i) = E$ se i é par e $f(i) = O$ caso contrário, temos que f é um morfismo limitado sobrejetivo entre $\mathcal{F}$ e $\mathcal{F}'$. Mas $\mathcal{F}$ é acíclico enquanto $\mathcal{F}'$ não é. Logo, pelo corolário 2.18, como aciclicidade não é preservada por imagens mórficas limitadas, ela não pode ser definida na lógica básica para grafos. □

Em princípio, pode parecer que o teorema acima é genérico demais para as nossas necessidades, uma vez que só precisamos ser capazes de definir aciclicidade em estruturas finitas e utilizamos uma estrutura infinita como parte da prova. Entretanto, como o teorema 2.21 mostra, não podemos escrever uma fórmula que descreva

"aciclicidade em estruturas finitas" ou qualquer outra propriedade "em estruturas finitas".

Teorema 5.8. *A classe de grafos hamiltonianos não pode ser definida na lógica básica para grafos.*

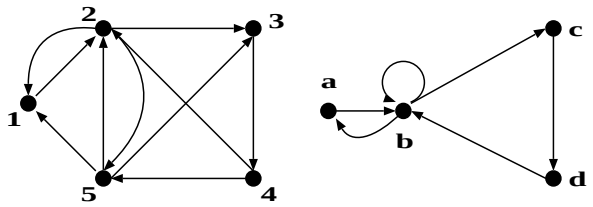

Figura 5.1: O grafo 1,2,3,4,5 é hamiltoniano e o grafo a,b,c,d não é.

Demonstração. Na figura 5.1, seja $f = \{(1, a), (2, b), (3, c), (4, d), (5, b)\}$. É simples mostrar que f é um morfismo limitado sobrejetivo. Pelo corolário 2.18, como a propriedade hamiltoniana não é preservada por imagens mórficas limitadas, ela não pode ser definida na lógica básica para grafos. □

Teorema 5.9. *A classe de grafos eulerianos não pode ser definida na lógica básica para grafos.*

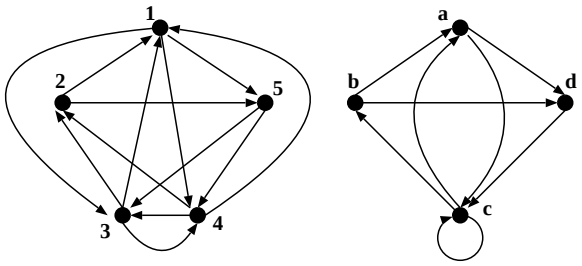

Figura 5.2: O grafo 1,2,3,4,5 é euleriano e o grafo a,b,c,d não é.

Demonstração. Na figura 5.2, seja $f = \{(1, a), (2, b), (3, c), (4, c), (5, d)\}$. É simples mostrar que f é um morfismo limitado sobrejetivo. Pelo corolário 2.18, como a propriedade euleriana não é preservada por imagens mórficas limitadas, ela não pode ser definida na lógica básica para grafos. □

5.3 Lógica Híbrida para Grafos

Como foi mostrado na seção anterior, a linguagem da lógica básica para grafos não possui poder expressivo suficiente para descrever as propriedades que queremos. Para alcançarmos nosso objetivo, precisamos de uma lógica que possua uma linguagem com maior poder expressivo, mas que seja ainda decidível com respeito ao problema de verificação de estrutura, que é utilizado para a verificação das propriedades.

Uma classe interessante de lógicas a serem consideradas é a classe das *lógicas híbridas* [14, 15, 10]. Tais lógicas foram apresentadas no capítulo 2. Uma extensão híbrida da nossa lógica anterior é uma escolha interessante devido a uma combinação de fatores. Sua linguagem possui maior poder expressivo, uma vez que fórmulas híbridas não são invariantes sob uniões disjuntas ou imagens mórficas limitadas [14], mas a lógica não apresenta nenhum incremento relevante em complexidade computacional, na análise de pior caso, em comparação com a lógica anterior, conforme foi mostrado no capítulo 2.

Nesta seção, definimos a *lógica híbrida para grafos* e tentamos expressar, nesta nova lógica, as propriedades de grafos que estamos discutindo.

Definição 5.10. *A linguagem da* lógica híbrida para grafos *é uma linguagem modal que consiste de um conjunto enumerável* Φ *de símbolos proposicionais e um conjunto enumerável* $\mathcal{L}$ *de nominais tais que* $\Phi \cap \mathcal{L} = \emptyset$*, os operadores booleanos* $\neg$ *e* $\wedge$*, a constante* $\top$ *e os operadores modais (ou modalidades)* $@_i$*, para cada nominal* i*,* $\Diamond$ *e* $\Diamond^+$*. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid i \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond\varphi \mid \Diamond^+\varphi \mid @_i\varphi,$$

onde $p \in \Phi$ *e* $i \in \mathcal{L}$*.*

As fórmulas desta lógica são avaliadas nas estruturas e modelos de Kripke híbridos usuais. Para as definições apropriadas de satisfazibilidade das fórmulas desta lógica, o capítulo 2 pode ser consultado.

Conforme estudado no capítulo 2, temos as seguintes complexidades computacionais para a lógica híbrida para grafos.

Teorema 5.11 ([31]). *O problema da satisfazibilidade e o problema da validade para a lógica híbrida para grafos são EXPTEMPO-Completos no comprimento da fórmula.*

Teorema 5.12 ([17]). *O problema da verificação de modelo para a lógica híbrida para grafos é polinomial (linear) no produto do tamanho do modelo e do comprimento da fórmula.*

O limite superior para a complexidade do problema da verificação de estrutura é um pouco diferente no caso de uma lógica híbrida, devido à restrição particular na valoração de nominais. Para lógicas híbridas o limite superior tem a forma

$$FC = O(2^{|p|\times n} \times n^{|i|} \times MC), \tag{5.1}$$

onde $|p|$ é o número de símbolos proposicionais distintos que ocorrem na fórmula dada ϕ, $|i|$ é o número de nominais distintos que ocorrem em ϕ e n é o número de vértices em $\mathcal{F}$. Precisamos aplicar o algoritmo de verificação de modelo a todo modelo $\mathcal{M}$ da estrutura dada $\mathcal{F}$. Cada símbolo proposicional p que aparece em ϕ pode receber 2^n valorações possíveis $\mathbf{V}(p)$, enquanto cada nominal i pode receber apenas n valorações possíveis $\mathbf{V}(i)$.

Teorema 5.13. *O problema da verificação de estrutura para a lógica híbrida para grafos é polinomial (linear) no comprimento da fórmula e EXPTEMPO no tamanho da estrutura, no número de símbolos proposicionais distintos que ocorrem na fórmula e no número de nominais distintos que ocorrem na fórmula.*

Demonstração. Este resultado segue diretamente da discussão acima. □

Podemos notar que a lógica híbrida para grafos é realmente uma escolha muito interessante, uma vez que obtemos um poder expressivo maior sem nenhum acréscimo relevante na complexidade computacional.

Na lógica híbrida para grafos, podemos expressar ao menos duas das propriedades que queremos.

Teorema 5.14. *Seja G um grafo e G' seu grafo não direcionado subjacente. G é* fortemente conexo *se e somente se $G \Vdash \phi$ e* (fracamente) conexo *se e somente se $G' \Vdash \phi$, onde ϕ é a fórmula*

$$\phi = i \vee \Diamond^{+} i.$$

Demonstração. Aqui utilizamos uma fórmula diferente da apresentada nos artigos [27] e [29] e no apêndice A. A fórmula apresentada no teorema acima possui um nominal a menos do que a fórmula presente nas referências citadas acima. Para uma prova do teorema acima, o artigo [32] ou o apêndice B (teorema B.68) podem ser consultados. □

Queremos agora determinar quão complexo é testar se um grafo é conexo utilizando a fórmula ϕ acima. Isto consiste em, dado um grafo G, executar a verificação de estrutura para verificar se $G \Vdash \phi$. Devemos notar primeiramente que não há símbolos proposicionais em ϕ e há apenas um nominal em ϕ. Além disso, o comprimento de ϕ é constante e não depende do tamanho do grafo. Seja CON a complexidade de testar se um grafo é conexo através de uma verificação de estrutura com a fórmula ϕ. Então, levando em consideração estas observações e a fórmula na equação (5.1), temos que

$$CON = O(n \times MC),$$

onde, para a fórmula ϕ, MC é polinomial (de fato, linear) no tamanho do grafo.

Teorema 5.15. *A complexidade para verificar se um grafo é conexo utilizando-se a fórmula ϕ acima é polinomial (quadrática) no tamanho do grafo.*

Demonstração. Este resultado segue diretamente da discussão acima. □

Teorema 5.16. *Um grafo G é* acíclico *se e somente se $G \Vdash \phi$, onde ϕ é a fórmula*

$$\phi = @_i \neg \Diamond^+ i.$$

Vamos agora determinar quão complexo é testar se um grafo é acíclico utilizando a fórmula ϕ acima. Novamente, não há símbolos proposicionais em ϕ e há apenas um nominal em ϕ. Além disso, o comprimento de ϕ é, novamente, constante e não depende do tamanho do grafo. Seja ACY a complexidade de testar se um grafo é acíclico através de uma verificação de estrutura com a fórmula ϕ. Então, levando em consideração estas observações e a fórmula na equação (5.1), temos que

$$ACY = O(n \times MC),$$

onde, para a fórmula ϕ, MC é polinomial (de fato, linear) no tamanho do grafo.

Teorema 5.17. *A complexidade para verificar se um grafo é acíclico utilizando-se a fórmula ϕ acima é polinomial (quadrática) no tamanho do grafo.*

Demonstração. Este resultado segue diretamente da discussão acima. □

Consideramos as complexidades nos teoremas 5.15 e 5.17 satisfatórias. Não é necessário procurar por formas alternativas de expressar estas propriedades nesta lógica ou em outra.

Antes de tentarmos encontrar uma fórmula para descrever grafos hamiltonianos, precisamos considerar algumas questões da teoria de grafos. Na teoria de grafos [1], não há resultado conhecido que enuncie uma condição necessária e suficiente para um grafo ser hamiltoniano. Se pudéssemos encontrar uma fórmula que descrevesse os grafos hamiltonianos sem ter que descrever o ciclo hamiltoniano em si, estaríamos encontrando tal condição necessária e suficiente. Desta forma, o que nossa fórmula faz é inspecionar todos os caminhos do grafo, procurando por um ciclo hamiltoniano.

Seja $\mathcal{L}_n = \{i_1, \ldots, i_n\}$ um conjunto contendo n nominais. Antes de definirmos uma fórmula para a propriedade hamiltoniana, precisamos definir uma fórmula que é globalmente satisfeita em um modelo sob uma valoração $\mathbf{V}$ se e somente se $\mathbf{V}(i_k) \neq \mathbf{V}(i_l)$, para todos $i_k, i_l \in \mathcal{L}_n$ tais que $k \neq l$.

Lema 5.18. *Uma valoração satisfaz $\mathbf{V}(i_k) \neq \mathbf{V}(i_l)$, para todos $i_k, i_l \in \mathcal{L}_n$ tais que $k \neq l$, se e somente se $(\mathcal{F}, \mathbf{V}) \Vdash \psi_n$, onde ψ_n é a fórmula*

$$\psi_n = \bigwedge_{1 \leq k < n} \left(@_{i_k} \bigwedge_{k < l \leq n} \neg i_l \right).$$

Agora definimos um conjunto F de permutações dos nominais de $\mathcal{L}_n$. Este conjunto possui $n!$ elementos. Representamos uma permutação como uma função bijetiva $\sigma : \{1, \ldots, n\} \mapsto \mathcal{L}_n$.

Teorema 5.19. *Um grafo conexo G (com n vértices) é* hamiltoniano *se e somente se $G \Vdash \phi$, onde ϕ é a fórmula*

$$\phi = \psi_n \rightarrow \delta_n,$$

onde ψ_n é a fórmula do lema 5.18 e

$$\delta_n = \bigvee_{\sigma \in F} (\sigma(1) \wedge \Diamond(\sigma(2) \wedge \Diamond(\sigma(3) \ldots (\sigma(n-1) \wedge \Diamond(\sigma(n) \wedge \Diamond\sigma(1)) \ldots).$$

Neste caso, o número de nominais distintos e o comprimento da fórmula estão ambos ligados ao tamanho do grafo. Embora a complexidade do problema da verificação de estrutura seja polinomial no comprimento da fórmula, como a fórmula possui comprimento fatorial no tamanho do grafo, é impraticável realizar uma verificação desta fórmula. No entanto, o comprimento da fórmula não é o único problema. Como existem tantos nominais na fórmula quanto vértices no grafo, é fácil verificar que a complexidade continuaria fatorial mesmo com uma redução no tamanho da fórmula. No caso da propriedade hamiltoniana, precisamos procurar por formas alternativas de expressá-la utilizando outras lógicas. Duas outras tentativas são apresentadas nas próximas duas seções.

A dificuldade em encontrar uma fórmula para descrever a propriedade euleriana é de uma natureza completamente diferente. Aqui, a limitação está na linguagem, não na definição teórica da propriedade. Há um resultado conhecido que enuncia uma condição necessária e suficiente para um grafo ser euleriano (teorema 3.13), então o argumento usado acima para grafos hamiltonianos não se aplica aqui. Entretanto, a linguagem da lógica híbrida para grafos não possui o poder expressivo, ao menos não sem ter que recorrer a uma busca exaustiva através de cada possível caminho no grafo de uma maneira semelhante a utilizada pela fórmula no teorema 5.19, para enunciar condições de cardinalidade a respeito de arestas adjacentes a partir de um vértice ou a um vértice, como é necessário para o teorema 3.13.

A outra maneira de descrever a propriedade euleriana seria encontrar uma fórmula que descreva explicitamente um caminho euleriano no grafo. Entretanto, é muito difícil encontrar tal fórmula, uma vez que a lógica híbrida para grafos e muitas outras lógicas modais não são boas para descrever propriedades relacionadas a arestas. Agora, utilizando nominais, nós temos nomes para vértices, mas ainda não podemos manter um registro de quais arestas estamos usando quando caminhamos em um grafo. Isto sugere que uma possível solução seria encontrar uma maneira de nomear arestas de uma maneira similar ao uso de nominais para nomear vértices. Nós desenvolvemos esta ideia na seção 5.6, onde descrevemos um método para nomear arestas dentro do arcabouço de uma lógica híbrida e o utilizamos para encontrar uma fórmula para a propriedade euleriana.

5.4 A Lógica Temporal CTL* Híbrida

O fato da fórmula introduzida na seção anterior para descrever grafos hamiltonianos possuir comprimento fatorial no tamanho do grafo torna sua verificação impraticável. Obviamente, não podemos ter a expectativa de verificar a propriedade hamiltoniana em tempo polinomial, uma vez que determinar se um grafo é hamiltoniano é um problema NP-Completo [33], mas podemos certamente buscar um modo de verificá-la com uma complexidade inferior a tempo fatorial. Este é o nosso objetivo nesta seção e na próxima.

Em nossa primeira tentativa de escrever uma fórmula curta (com comprimento polinomial no tamanho do grafo) para descrever a classe de grafos hamiltonianos, utilizamos uma extensão híbrida da lógica temporal CTL* [34], a qual denotamos por *CTL* híbrida*. Poderíamos usar o μ-Cálculo híbrido apresentado em [35], uma vez que esta lógica contém CTL* híbrida [36, 35]. Entretanto, não tomamos este caminho, uma vez que CTL* híbrida possui poder expressivo suficiente para as nossas necessidades e suas fórmulas são mais simples de serem compreendidas do que fórmulas do μ-Cálculo híbrido.

Definição 5.20. *A linguagem da* lógica CTL* híbrida *é uma linguagem modal que consiste de um conjunto enumerável Φ de símbolos proposicionais e um conjunto enumerável $\mathcal{L}$ de nominais tais que $\Phi \cap \mathcal{L} = \emptyset$, os operadores booleanos $\neg$ e $\wedge$, a constante $\top$ e os operadores modais (ou modalidades) $@_i$, para cada nominal i,* **A**, **E**, **X**, **F**, **G** *e* **U**. *As fórmulas são divididas em* fórmulas de vértices **S** *e* fórmulas de caminhos **P** *definidas pela seguinte indução mútua:*

$$\mathbf{S} ::= p \mid i \mid \top \mid \neg\mathbf{S} \mid \mathbf{S_1} \wedge \mathbf{S_2} \mid \mathbf{AP} \mid \mathbf{EP} \mid @_i\mathbf{S}$$

$$\mathbf{P} ::= \mathbf{S} \mid \neg\mathbf{P} \mid \mathbf{P_1} \wedge \mathbf{P_2} \mid \mathbf{XP} \mid \mathbf{FP} \mid \mathbf{GP} \mid \mathbf{P_1UP_2}$$

onde $p \in \Phi$ e $i \in \mathcal{L}$. O conjunto das fórmulas bem formadas da lógica CTL* híbrida é então o conjunto de todas as fórmulas de vértices *geradas pelas regras acima.*

As fórmulas desta lógica são avaliadas nas estruturas e modelos de Kripke híbridos usuais. Para as definições apropriadas de satisfazibilidade das fórmulas desta lógica, o capítulo 2 pode ser consultado.

Teorema 5.21 ([20])**.** *O problema da satisfazibilidade e o problema da validade para CTL* são 2EXPTEMPO-Completos no comprimento da fórmula.*

Teorema 5.22 ([36])**.** *Toda fórmula da lógica CTL* híbrida pode ser traduzida em um fórmula do μ-Cálculo híbrido. A complexidade desta tradução é 2EXPTEMPO no comprimento da fórmula original. A fórmula traduzida pode ser escrita com comprimento exponencial com respeito ao comprimento da fórmula original.*

Teorema 5.23 ([35])**.** *O problema da satisfazibilidade e o problema da validade para o μ-Cálculo híbrido são EXPTEMPO-Completos no comprimento da fórmula.*

Teorema 5.24. *O problema da satisfazibilidade e o problema da validade para CTL* híbrida são 2EXPTEMPO-Difíceis no comprimento da fórmula e são decidíveis.*

Demonstração. O limite inferior segue do teorema 5.21 e a decidibilidade segue dos teoremas 5.22 e 5.23. □

Teorema 5.25. *O problema da verificação de modelo para CTL* é polinomial (linear) no tamanho do modelo e EXPTEMPO no comprimento da fórmula.*

Demonstração. Isto segue de uma combinação direta das complexidades dos problemas da verificação de modelo para CTL* e para a lógica híbrida básica apresentadas no capítulo 2. □

O limite superior para a complexidade do problema da verificação de estrutura é o mesmo da seção anterior:

$$FC = O(2^{|p|\times n} \times n^{|i|} \times MC), \tag{5.2}$$

onde $|p|$ é o número de símbolos proposicionais distintos que ocorrem na fórmula dada ϕ, $|i|$ é o número de nominais distintos que ocorrem em ϕ e n é o número de vértices em $\mathcal{F}$.

Teorema 5.26. *O problema da verificação de estrutura para a CTL* híbrida é EXPTEMPO no comprimento da fórmula, no tamanho da estrutura, no número de símbolos proposicionais distintos que ocorrem na fórmula e no número de nominais distintos que ocorrem na fórmula.*

Demonstração. Este resultado segue diretamente da discussão acima. □

Seja G um grafo com n vértices e seja $\mathcal{L}_n = \{i_1, \ldots, i_n\}$. Vamos adicionar um laço a todos os vértices de G. Podemos então definir uma fórmula que é válida se e somente se G é hamiltoniano.

Teorema 5.27. *Um grafo conexo G (com n vértices) é* hamiltoniano *se e somente se $G \Vdash \phi$, onde ϕ é a fórmula*

$$\phi = \psi_n \rightarrow \varepsilon_n,$$

onde ψ_n é a fórmula do lema 5.18 e

$$\varepsilon_n = @_{i_1}\mathbf{E}[\mathbf{XF}i_1 \wedge \mathbf{F}i_2 \wedge \ldots \wedge \mathbf{F}i_n \wedge$$

$$\wedge \mathbf{XG}(i_1 \rightarrow \mathbf{G}i_1) \wedge \mathbf{G}(i_2 \rightarrow \mathbf{XG}\neg i_2) \wedge \ldots \wedge \mathbf{G}(i_n \rightarrow \mathbf{XG}\neg i_n)].$$

Vamos agora determinar quão complexo é testar se um grafo é hamiltoniano utilizando a fórmula ϕ acima. Primeiramente, agora temos uma fórmula com comprimento polinomial (quadrático) no tamanho do grafo. Além disso, não há símbolos proposicionais na fórmula. A partir da estrutura da fórmula, não é difícil perceber que seria perda de tempo verificar valorações que não atribuem diferentes vértices a cada nominal, uma vez que ψ_n é uma precondição de uma implicação. Além disso, a fórmula ε_n é completamente simétrica com respeito aos nominais $i_2, \ldots, i_n$. Portanto, para cada valoração possível de i_1, precisamos apenas verificar uma valoração arbitrária para $i_2, \ldots, i_n$ tal que cada um destes nominais nomeie um vértice distinto. Seja HAM a complexidade de testar se um grafo é hamiltoniano através de uma verificação de estrutura com a fórmula ϕ. Então, levando em consideração estas observações e a fórmula na equação (5.2), temos que

$$HAM = O(n \times MC),$$

onde, para a fórmula ϕ, MC é EXPTEMPO no tamanho do grafo.

Teorema 5.28. *A complexidade para verificar se um grafo é hamiltoniano utilizando-se a fórmula ϕ acima é EXPTEMPO no tamanho do grafo.*

Demonstração. Este resultado segue diretamente da discussão acima. □

5.5 Lógica Híbrida para Grafos com Quantificador Local

Na seção anterior, fomos capazes de reduzir o limite superior da complexidade de testar se um grafo é hamiltoniano utilizando um método baseado em verificação de estrutura de um tempo fatorial para um tempo exponencial.

Nesta seção, descrevemos uma terceira lógica, que é a extensão da lógica híbrida para grafos com a adição do quantificador local $\downarrow$, e a utilizamos para construir uma fórmula que expressa a propriedade hamiltoniana. Com esta fórmula, conseguimos reduzir ainda mais a complexidade de testar se um grafo é hamiltoniano utilizando um método baseado em verificação de estrutura. Isto acontece porque somos capazes, para esta fórmula, de reduzir o problema de verificação de estrutura a um caso particular do problema de verificação de modelo que possui complexidade inferior a do caso geral.

Definição 5.29. *A linguagem da* lógica híbrida para grafos com quantificador local $\downarrow$ *é uma linguagem modal que consiste de um conjunto enumerável* Φ *de símbolos proposicionais, um conjunto enumerável* $\mathcal{L}$ *de nominais e um conjunto enumerável* $\mathcal{S}$ *de variáveis de estado tais que os conjuntos são disjuntos dois a dois, os operadores booleanos* $\neg$ *e* $\wedge$*, a constante* $\top$*, os operadores modais (ou modalidades)* $@_i$*, para cada nominal* i*,* $@_x$*, para cada variável de estado* x*,* $\Diamond$ *e* $\Diamond^+$ *e o quantificador local* $\downarrow$*. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid i \mid x \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond\varphi \mid \Diamond^+\varphi \mid @_i\varphi \mid @_x\varphi \mid \downarrow x.\varphi,$$

onde $p \in \Phi$*,* $i \in \mathcal{L}$ *e* $x \in \mathcal{S}$*.*

As fórmulas desta lógica são avaliadas nas estruturas e modelos de Kripke híbridos usuais. Para as definições apropriadas de satisfazibilidade das fórmulas desta lógica, o capítulo 2 pode ser consultado.

Conforme estudado no capítulo 2, temos as seguintes complexidades computacionais para a lógica híbrida para grafos com quantificador local.

Teorema 5.30 ([16])**.** *O problema da satisfazibilidade e o problema da validade para a lógica híbrida para grafos com quantificador local são* indecidíveis*.*

Teorema 5.31 ([17])**.** *O problema da verificação de modelo para a lógica híbrida para grafos com quantificador local é PESPAÇO-Completo tanto no tamanho do modelo quanto no comprimento da fórmula.*

Teorema 5.32 ([19])**.** *O problema da verificação de modelo para fórmulas da lógica híbrida para grafos com quantificador local que, quando colocadas em forma normal negativa, não possuem nenhuma ocorrência dos operadores* $\Box$, $\Diamond^+$ *e* $\Box^+$ *é NP-Completo tanto no tamanho do modelo quanto no comprimento da fórmula.*

O limite superior para a complexidade do problema da verificação de estrutura é novamente

$$FC = O(2^{|p|\times n} \times n^{|i|} \times MC), \tag{5.3}$$

onde $|p|$ é o número de símbolos proposicionais distintos que ocorrem na fórmula dada ϕ, $|i|$ é o número de nominais distintos que ocorrem em ϕ e n é o número de vértices em $\mathcal{F}$. É importante notar que, se o problema da verificação de modelo pode ser resolvido em espaço polinomial, então o problema da verificação de estrutura também pode ser resolvido em espaço polinomial, uma vez que a verificação de estrutura de uma fórmula em uma estrutura $\mathcal{F}$ é feita através de um número finito de verificações de modelo independentes entre si.

Teorema 5.33. *O problema da verificação de estrutura para a lógica híbrida para grafos com quantificador local é PESPAÇO tanto no tamanho do modelo quanto no comprimento da fórmula.*

Demonstração. Este resultado segue diretamente da discussão acima. □

Teorema 5.34. *Um grafo conexo G (com n vértices) é* hamiltoniano *se e somente se* $G \Vdash \phi$*, onde* ϕ *é a fórmula*

$$\phi = \downarrow x_1.\Diamond \downarrow x_2.(\neg x_1 \wedge \Diamond \downarrow x_3.(\bigwedge_{1\leq k<3} \neg x_k \wedge \Diamond \downarrow x_4.(\ldots \wedge \downarrow x_{n-1}.(\bigwedge_{1\leq k<n-1} \neg x_k \wedge \Diamond \downarrow x_n.$$

$$(\bigwedge_{1\leq k<n} \neg x_k \wedge \Diamond x_1)\ldots).$$

Vamos agora determinar quão complexo é testar se um grafo é hamiltoniano utilizando a fórmula ϕ acima. Primeiramente, temos agora uma fórmula que é uma sentença com comprimento polinomial (quadrático) no tamanho do grafo. Além

disso, não há nem símbolos proposicionais nem nominais na fórmula. Isto significa que a valoração é completamente irrelevante para a satisfação desta sentença. Portanto, o problema da verificação de estrutura é reduzido ao problema de verificação de modelo para um modelo arbitrário da estrutura. Seja HAM a complexidade de testar se um grafo é hamiltoniano através de uma verificação de estrutura com a fórmula ϕ. Então, levando em consideração estas observações e a fórmula na equação (5.3), temos que

$$HAM = MC.$$

Agora, devemos notar que ϕ já se encontra em forma normal negativa e não possui nenhuma ocorrência dos operadores $\Box$, $\Diamond^+$ ou $\Box^+$. Então, a redução de complexidade no problema da verificação de modelo enunciada no teorema 5.32 se aplica e a complexidade da verificação de modelo para esta fórmula está em NP.

Teorema 5.35. *A complexidade para verificar se um grafo é hamiltoniano utilizando-se a fórmula ϕ acima é NP no tamanho do grafo.*

Demonstração. Este resultado segue diretamente da discussão acima. □

A fórmula ϕ acima é uma fórmula "ótima" para descrever a propriedade hamiltoniana, no seguinte sentido: uma vez que o problema de testar se um grafo é hamiltoniano é NP-Completo [33] e o teste que desenvolvemos usando esta fórmula está em NP, é impossível encontrar qualquer outra fórmula, nesta lógica ou em outra, que descreva a propriedade hamiltoniana e possa ser verificada com uma complexidade inferior a de ϕ (assumindo que NP $\neq$ P).

5.6 Propriedades Relativas a Arestas

Como foi mencionado no fim da seção 5.3, a melhor maneira de descrever a propriedade euleriana seria utilizando o teorema 3.13. Entretanto, como também foi mencionado no fim daquela seção, as modalidades $\Diamond$ comuns não possuem o poder expressivo para enunciar condições de cardinalidade a respeito de arestas adjacentes a partir de e a um vértice, como é necessário para o teorema 3.13. Isto continua sendo verdade para os operadores temporais definidos na seção 5.4. Isto ocorre porque todos estes operadores são operadores "existenciais". Tudo que eles podem

fazer é diferenciar entre situações como "há uma aresta" e "não há uma aresta" ou "há um caminho" e "não há um caminho". Precisaríamos de operadores capazes de realizar algum tipo de "contagem" para podermos expressar eficientemente o teorema 3.13. Lógicas modais com este tipo de operador existem na literatura e são chamadas de *lógicas modais graduadas*, tendo sido introduzidas primeiramente em [37]. No capítulo 2, a lógica modal graduada básica é apresentada.

Apresentamos uma lógica modal graduada no final desta seção e a utilizamos para expressar a propriedade euleriana com a ajuda do teorema 3.13. Mas primeiramente, desenvolvemos um método para expressar a propriedade euleriana nas lógicas híbridas já apresentadas nas seções anteriores. Este método poderia ser útil para expressar outras propriedades relativas a arestas para as quais não existe um teorema que enuncia uma condição necessária e suficiente para que esta propriedade seja satisfeita.

5.6.1 Propriedade Euleriana através de Subdivisões de Grafos

Se não usamos o teorema 3.13, então precisamos expressar a propriedade euleriana com uma fórmula que descreva explicitamente o caminho euleriano no grafo. Isto também é uma tarefa difícil, devido as razões expostas no fim da seção 5.3. Precisamos de uma maneira de identificar arestas particulares, mas lógicas híbridas possuem apenas nomes para vértices. Então, primeiramente desenvolvemos um método para nomear arestas *dentro* do formalismo de uma lógica híbrida e posteriormente o utilizamos para descrever a propriedade euleriana.

Definição 5.36. *Seja $\langle v, w\rangle$ uma aresta em um grafo G. Uma* subdivisão de aresta *consiste de adicionar um novo vértice u a G, remover a aresta $\langle v, w\rangle$ e adicionar as arestas $\langle v, u\rangle$ e $\langle u, w\rangle$ a G. Uma* subdivisão *de um grafo G é um grafo G' obtido a partir de G por um número finito de subdivisões de aresta.*

Definição 5.37. *Seja G um grafo. Definimos $G' = \mathcal{E}(G)$ como o grafo obtido a partir de G subdividindo cada aresta de G exatamente uma vez. Dizemos que G' é um $\mathcal{E}$-grafo.*

Logo, se G possui n vértices e m arestas, G' possui $m + n$ vértices. De fato,

se denotarmos por W o conjunto de vértices de G e por W' o conjunto de vértices de G', temos que $W' = W \cup W^*$ ($W \cap W^* = \emptyset$), onde W^* é o conjunto de novos vértices adicionados durante a subdivisão. Além disso, toda aresta de G' possui uma extremidade em W e a outra em W^* e existe um mapeamento bijetivo entre elementos de W^* e arestas de G.

Este mapeamento bijetivo entre o conjunto W^* e as arestas de G é o ponto chave nesta construção. No grafo original G, não podemos identificar arestas particulares utilizando apenas uma linguagem híbrida. Então, se queremos definir uma propriedade em G, descrita utilizando suas arestas, construímos $G' = \mathcal{E}(G)$ e a descrevemos em G', utilizando os elementos de W^*. Estes elementos podem ser identificados por nominais da maneira comum. Isto é o que fazemos para expressar a propriedade euleriana.

Para este método funcionar, precisamos apenas prestar atenção a um detalhe importante. Em $\mathcal{E}$-grafos, é fundamental podermos distinguir se um vértice pertence a W ou a W^*. Logo, ao invés de trabalharmos com um conjunto de nominais $\mathcal{L}$, trabalhamos com dois conjuntos, $\mathcal{L}$ e $\mathcal{L}^*$ ($\mathcal{L} \cap \mathcal{L}^* = \emptyset$). Ao invés de escrevermos $G' = (W', R')$, escrevemos $G' = (W, W^*, R')$, para deixar clara a diferença entre os dois conjuntos de vértices, e definimos valorações $\mathbf{V}$ de forma que $\mathbf{V}(p) \in \mathcal{P}(W \cup W^*)$, se p é um símbolo proposicional, $\mathbf{V}(i) = \{v\}$, tal que $v \in W$, se $i \in \mathcal{L}$ e $\mathbf{V}(j) = \{w\}$, tal que $w \in W^*$, se $j \in \mathcal{L}^*$. Denotamos os nominais em $\mathcal{L}$ por $i_1, i_2, \ldots$ e os nominais em $\mathcal{L}^*$ por $j_1, j_2, \ldots$.

Uma vez que vamos expressar a propriedade euleriana com uma fórmula que descreve explicitamente um caminho euleriano no grafo, podemos aproveitar ideias de três fórmulas apresentadas anteriormente: as fórmulas nos teoremas 5.19, 5.27 e 5.34. Mas a fórmula no primeiro teorema possui comprimento fatorial, então iremos adaptar apenas as fórmulas no segundo e terceiro teoremas para o caso euleriano.

Esta não é uma tarefa difícil. As fórmulas nos teoremas 5.27 e 5.34 expressam que, para um grafo $G = (W, R)$, existe um ciclo que visita cada vértice em W exatamente uma vez. Para expressar a propriedade euleriana, precisamos de fórmulas que verifiquem que, para um grafo G, existe um caminho fechado tal que toda aresta de G aparece exatamente uma vez nele. Isto é equivalente a verificar, em $G' = \mathcal{E}(G)$, se existe um caminho fechado que visita todo vértice de W^* exatamente uma vez.

Primeiramente, adaptamos a fórmula no teorema 5.27 para expressar a propriedade euleriana. Seja $G' = \mathcal{E}(G)$ um $\mathcal{E}$-grafo com n vértices em W e m vértices em W^* e seja $\mathcal{L}_m = \{j_1, \ldots, j_m\} \subset \mathcal{L}^*$. Adicionamos um laço a todos os vértices de G' em W^*. Podemos então definir uma fórmula que é válida se e somente se G é euleriano.

Teorema 5.38. *Um grafo conexo G (com m arestas) é* euleriano *se e somente se $G' \Vdash \phi$, onde $G' = \mathcal{E}(G)$ e ϕ é a fórmula*

$$\phi = \psi_m \rightarrow \varepsilon_m,$$

onde ψ_m é a fórmula no lema 5.18 e

$$\varepsilon_m = @_{j_1}\mathbf{E}[\mathbf{XF}j_1 \wedge \mathbf{F}j_2 \wedge \ldots \wedge \mathbf{F}j_m \wedge$$

$$\wedge \mathbf{XG}(j_1 \rightarrow \mathbf{G}j_1) \wedge \mathbf{G}(j_2 \rightarrow \mathbf{XG}\neg j_2) \wedge \ldots \wedge \mathbf{G}(j_m \rightarrow \mathbf{XG}\neg j_m)].$$

Agora, adaptamos a fórmula no teorema 5.34.

Teorema 5.39. *Um grafo conexo G (com m arestas) é* euleriano *se e somente se $G' \Vdash \phi$, onde $G' = \mathcal{E}(G)$ e ϕ é a fórmula*

$$\phi = @_{j_1} \downarrow x_1.\Diamond\Diamond \downarrow x_2.(\neg x_1 \wedge \Diamond\Diamond \downarrow x_3.(\bigwedge_{1 \leq k < 3} \neg x_k \wedge \Diamond\Diamond \downarrow x_4.(\ldots \wedge \downarrow x_{m-1}.$$

$$(\bigwedge_{1 \leq k < m-1} \neg x_k \wedge \Diamond\Diamond \downarrow x_m.(\bigwedge_{1 \leq k < m} \neg x_k \wedge \Diamond\Diamond x_1)\ldots).$$

Como as fórmulas nos teoremas 5.38 e 5.39 foram adaptadas das fórmulas nos teoremas 5.27 e 5.34, os resultados de complexidade apresentados nos teoremas 5.28 e 5.35, respectivamente, também se aplicam a elas.

5.6.2 Propriedade Euleriana em uma Lógica Modal Graduada

Nesta seção, definimos uma lógica modal graduada que é apropriada para as nossas necessidades e a utilizamos para expressar a propriedade euleriana. Esta lógica é chamada de *lógica graduada para grafos*.

Definição 5.40. *A linguagem da* lógica graduada para grafos *consiste de um conjunto enumerável* Φ *de símbolos proposicionais, os operadores booleanos* $\neg$ *e* $\wedge$*, a constante* $\top$ *e os operadores modais (ou modalidades)* $\Diamond_i$ *e* $\Diamond_i^{-1}$*, para todo* $i \in \mathbb{N}$*. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_i\varphi \mid \Diamond_i^{-1},$$

onde $p \in \Phi$ *e* $i \in \mathbb{N}$*.*

As fórmulas desta lógica são avaliadas nas estruturas e modelos de Kripke usuais. Para as definições apropriadas de satisfazibilidade das fórmulas desta lógica, o capítulo 2 pode ser consultado. Para tornar a linguagem mais elegante, introduzimos a abreviação $\blacklozenge_i\varphi = \Diamond_i\varphi \wedge \neg\Diamond_{i+1}\varphi$.

A fórmula $\Diamond_i\varphi$ é satisfeita em um vértice v se existem *pelo menos* i vértices distintos v_k, $1 \leq k \leq i$, tais que vRv_k e φ é satisfeita em v_k. A fórmula $\Diamond_i^{-1}\varphi$ é satisfeita em um vértice v se existem *pelo menos* i vértices distintos v_k, $1 \leq k \leq i$, tais que v_kRv e φ é satisfeita em v_k. A fórmula $\blacklozenge_i\varphi$ é satisfeita em um vértice v se existem *exatamente* i vértices distintos v_k, $1 \leq k \leq i$, tais que vRv_k e φ é satisfeita em v_k. A fórmula $\blacklozenge_i^{-1}\varphi$ é satisfeita em um vértice v se existem *exatamente* i vértices distintos v_k, $1 \leq k \leq i$, tais que v_kRv e φ é satisfeita em v_k.

Conforme estudado no capítulo 2, temos as seguintes complexidades computacionais para a lógica graduada para grafos.

Teorema 5.41 ([12])**.** *O problema da satisfazibilidade e o problema da validade para a lógica graduada para grafos são PESPAÇO-Completos no comprimento padrão da fórmula.*

Teorema 5.42 ([13])**.** *O problema da verificação de modelo para a lógica graduada para grafos é polinomial (linear) no produto do tamanho do modelo e do comprimento não graduado da fórmula.*

O limite superior para a complexidade do problema da verificação de estrutura é o mesmo da seção 5.2:

$$FC = O(2^{|p|\times n} \times MC), \tag{5.4}$$

onde $|p|$ é o número de símbolos proposicionais distintos que ocorrem na fórmula dada ϕ e n é o número de vértices em $\mathcal{F}$.

Teorema 5.43. *O problema da verificação de estrutura para a lógica graduada para grafos é polinomial (linear) no comprimento não graduado da fórmula e EXP-TEMPO no tamanho da estrutura e no número de símbolos proposicionais distintos que ocorrem na fórmula.*

Demonstração. Este resultado segue diretamente da discussão acima. □

Seja G um grafo com n vértices. Nós usamos a linguagem definida acima para construir uma fórmula que é válida se e somente se G é euleriano.

Teorema 5.44. *Um grafo conexo G (com n vértices) é* euleriano *se e somente se $G \Vdash \phi$, onde ϕ é a fórmula*

$$\phi = \bigvee_{0 \leq k \leq n} (\blacklozenge_k \top \wedge \blacklozenge_k^{-1} \top)$$

Vamos agora determinar quão complexo é testar se um grafo é euleriano utilizando a fórmula ϕ acima. Primeiramente, temos uma fórmula com comprimento não graduado linear no tamanho do grafo. Além disso, não há símbolos proposicionais na fórmula, o que significa que a valoração é completamente irrelevante para a satisfação desta fórmula. Seja EUL a complexidade de testar se um grafo é euleriano através de uma verificação de estrutura com a fórmula ϕ. Então, levando em consideração estas observações e a fórmula na equação (5.4), temos que

$$EUL = MC,$$

onde, para a fórmula ϕ, MC é polinomial (de fato, quadrática) no tamanho do grafo.

Teorema 5.45. *A complexidade para verificar se um grafo é euleriano utilizando-se a fórmula ϕ acima é polinomial (quadrática) no tamanho do grafo.*

Demonstração. Este resultado segue diretamente da discussão acima. □

Capítulo 6

Lógicas Modais Híbridas e Produtos de Grafos

"Você não pode se culpar constantemente. Culpe-se uma vez só e vá em frente." - Homer Simpson

Neste capítulo, analisamos o problema de como descrever uma condição necessária e suficiente para um grafo ser isomorfo a um produto cartesiano de grafos, conforme definido no capítulo 3. Analisamos também como utilizar esta caracterização para obter sistemas axiomáticos corretos e completos para uma série de produtos de lógicas modais.

O conteúdo deste capítulo tem como base os artigos [38] e [32] (reproduzido no apêndice B). As provas de todos os resultados apresentados neste capítulo podem ser consultadas no apêndice B.

6.1 Introdução

O objetivo deste capítulo é estudar algumas questões relativas a produtos de grafos e produtos de lógicas modais. Em particular, queremos definir uma condição que seja necessária e suficiente para um grafo ser um produto não trivial de outros grafos e utilizar esta caracterização para construir sistemas axiomáticos corretos e completos para produtos de lógicas modais. Estamos especialmente interessados em produtos de lógicas modais com dimensão maior do que dois, pois existem poucos resultados a respeito deles na literatura [5].

Então, nossa primeira tarefa neste capítulo é encontrar uma condição necessária e suficiente para um grafo ser isomorfo a um produto cartesiano de grafos não triviais e verificar se esta condição pode ser expressa em uma linguagem modal ou em uma linguagem híbrida.

Em [4] e [5], três propriedades que são satisfeitas em grafos que são produtos são apresentadas: *comutatividade à esquerda*, *comutatividade à direita* e a *propriedade de Church-Rosser*. No entanto, apesar destas propriedades, em conjunto com a *propriedade de Church-Rosser reversa*, serem necessárias para que um grafo seja um produto, elas não são suficientes (como ilustrado em um exemplo em [4]). Existem grafos que satisfazem estas quatro propriedades, mas não podem ser decompostos como produto de outros grafos.

Neste capítulo, introduzimos uma nova propriedade chamada *intransitividade* que, junto com as propriedades acima, forma uma condição necessária e suficiente para um grafo enumerável e (fracamente) conexo ser um produto. A prova da necessidade destas propriedades é simples e é realizada diretamente, sem a necessidade de assumir que o grafo é enumerável ou conexo. Por outro lado, a prova da suficiência é feita em duas etapas. Primeiramente, provamos que se um grafo enumerável e conexo satisfaz as cinco propriedades mencionadas acima, então seus componentes precisam satisfazer um isomorfismo particular. Em seguida, mostramos que se um grafo enumerável e conexo satisfaz intransitividade e seus componentes satisfazem este isomorfismo particular, então o grafo é um produto.

Os limites do poder expressivo de linguagens modais básicas são bem conhecidos. Existem uma série de resultados que garantem que estruturas que são "similares" de algumas maneiras precisam validar as mesmas fórmulas (alguns destes resultados são apresentados no capítulo 2) [10]. Utilizando estes resultados, mostramos que a propriedade de intransitividade não pode ser expressa na linguagem modal básica. De fato, também mostramos que que nenhuma condição que seja necessária e suficiente para um grafo ser um produto pode ser expressa na linguagem modal básica.

Lógicas híbridas são extensões de lógicas modais que permitem referências explícitas a estados individuais de um modelo. Além dos símbolos proposicionais, elas possuem um segundo conjunto de fórmulas atômicas, chamadas *nominais*, que

tem a propriedade de serem satisfeitas em exatamente um estado do modelo (para mais detalhes, consulte o capítulo 2 e as referências [14] e [15]). Utilizando uma linguagem híbrida, somos capazes de construir uma fórmula que descreve intransitividade.

Prosseguimos então determinando a complexidade computacional de testar, para um grafo finito e conexo, se ele é um produto. Para este teste, utilizamos um algoritmo de verificação de modelo para verificar as fórmulas que descrevem cada uma das cinco propriedades que caracterizam um produto: comutatividade à esquerda e à direita, propriedades de Church-Rosser e de Church-Rosser reversa e intransitividade.

Finalmente, usamos esta caracterização de produtos conexos e enumeráveis para construir sistemas axiomáticos corretos e completos para uma classe grande de produtos de lógicas modais.

Produtos de grafos surgem naturalmente como uma possível extensão da semântica de Kripke tradicional para lógicas modais multidimensionais. [4] apresenta uma ampla discussão de lógicas modais multidimensionais e fornece muitos exemplos de produtos de lógicas modais, onde a semântica é construída utilizando-se produtos de grafos. A maior parte dos sistemas axiomáticos corretos e completos para produtos de lógicas modais apresentados na literatura são para produtos de um par de lógicas modais, enquanto que somos capazes, utilizando lógica híbrida, de apresentar axiomatizações corretas e completas para muitos produtos de lógicas modais de dimensões arbitrárias.

O restante deste capítulo é organizado da seguinte maneira. Na seção 6.2, introduzimos uma nova propriedade de grafos chamada intransitividade. Na seção 6.3, apresentamos o conceito de decomposição de grafos e o utilizamos para provar que as cinco propriedades mencionadas acima formam uma condição necessária e suficiente para um grafo conexo e enumerável ser um produto. A seção 6.4 mostra que intransitividade não pode ser expressa em uma linguagem modal básica e que nenhuma condição que seja necessária e suficiente para um grafo ser um produto pode ser expressa em uma linguagem modal básica. Na seção 6.5, estendemos a linguagem modal da seção anterior para uma linguagem híbrida e mostramos que intransitividade pode ser expressa por uma fórmula híbrida. Na seção 6.6, determinamos

a complexidade computacional de testar, através de um algoritmo de verificação de modelo, se um grafo finito e conexo é um produto. Na seção 6.7, apresentamos a noção de produto de lógicas modais e, utilizando uma linguagem híbrida, construímos sistemas axiomáticos corretos e completos para uma grande classe de produtos de lógicas modais.

6.2 Intransitividade

Conforme vimos no capítulo 3, apesar de comutatividade à esquerda e à direita e as propriedades de Church-Rosser e de Church-Rosser reversa serem condições necessárias para um grafo ser um produto, elas não são suficientes: existem grafos que satisfazem estas quatro propriedades mas não podem ser decompostos como um produto de grafos não triviais, como mostra um exemplo de [4] na figura 3.3.

Para obtermos uma condição necessária e suficiente, precisamos adicionar uma quinta propriedade às quatro mencionadas acima. Ela é chamada de *intransitividade*.

Definição 6.1. *Seja* $G = \langle V, \overline{E}_1, \ldots, \overline{E}_n \rangle$ *e* $1 \leq i, j \leq n$, $i \neq j$. *Dizemos que* G *satisfaz* intransitividade *se e somente se toda tripla* $\langle u, v, w \rangle$ *de vértices de* G *que satisfaz as condições*

1. $u \neq v$;
2. $v \neq w$;
3. *existe um caminho não direcionado através de arestas de* $\overline{E}_j$ *de* u *para* v *e*
4. *existe um caminho não direcionado através de arestas de* $\overline{E}_i$ *de* v *para* w

também satisfaz as seguintes condições:

5. $u \neq w$;
6. $\langle u, w \rangle \notin \overline{E}_i$;
7. $\langle u, w \rangle \notin \overline{E}_j$.

Usamos as notações xU_iy e xU_jy para expressar que existe um caminho não direcionado através de arestas de $\overline{E}_i$ ($\overline{E}_j$, respectivamente) de x para y. Intransitividade é ilustrada na figura 6.1.

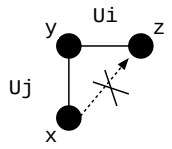

Figura 6.1: Intransitividade

A definição 6.1 lista as três condições (itens 5, 6 e 7) que são necessárias para intransitividade. Entretanto, estas condições podem ser simplificadas, uma vez que, sob as hipóteses na definição 6.1 (itens 1 a 4), a primeira condição (item 5) implica nas outras duas (itens 6 e 7). Suponha que todas as triplas $\langle u, v, w\rangle$ que satisfazem as hipóteses na definição 6.1 também satisfaçam a primeira condição ($u \neq w$). Agora, suponha que existe uma tripla $\langle a, b, c\rangle$ entre elas tal que $\langle a, c\rangle \in \overline{E}_i$ (não satisfaz a segunda condição). Então, aU_jb, bU_ic e $a\overline{E}_ic$. Isto implica que bU_ia. Mas então, $\langle a, b, a\rangle$ é uma tripla que satisfaz as hipóteses na definição 6.1 mas não satisfaz a primeira condição ($u \neq w$), o que é uma contradição à nossa hipótese inicial. Um argumento análogo pode ser feito também para a terceira condição.

Logo, quando precisamos verificar se um grafo satisfaz intransitividade, precisamos apenas verificar a primeira condição na definição 6.1. Por outro lado, quando sabemos que um grafo satisfaz intransitividade, podemos utilizar qualquer uma das três condições de acordo com nossas necessidades.

De acordo com a simplificação acima, intransitividade pode ser descrita da seguinte forma:

$$\forall x, y, z \in V((xU_jy \wedge yU_iz \wedge x \neq y \wedge y \neq z) \rightarrow (x \neq z)).$$

6.3 Decomposição de Grafos

O problema da decomposição de grafos consiste de, dado um grafo, determinar se este grafo pode ser decomposto em um produto de grafos não triviais. Neste capítulo, consideramos uma versão restrita deste problema.

Problema 6.2. *Dado um grafo* $G = \langle V, \overline{E}_1, \ldots, \overline{E}_n\rangle$ *direcionado, enumerável*[1] *e fracamente conexo*[2] *(apenas* conexo *daqui em diante), onde* $\overline{E}_i \neq \emptyset$ *para todo* $1 \leq$

[1]Um grafo é enumerável se seu conjunto de vértices é enumerável.

[2]Um grafo G é fracamente conexo se, para todo par de vértices u e v de G, existe um caminho

$i \leq n$, determinar se G é isomorfo a um produto $G' = G_1 \times G_2 \times \ldots \times G_n$, onde G_i é não trivial para todo $1 \leq i \leq n$.

No caso geral do problema, o grafo não precisaria ser necessariamente enumerável ou conexo.

Hipótese 6.3. *Pelo resto desta seção, todos os grafos G são considerados direcionados, enumeráveis e conexos e são dados na forma $G = \langle V, \overline{E}_1, \ldots, \overline{E}_n \rangle$.*

Observação 6.4. *Denotamos por $\mathcal{V}(G)$ o conjunto de vértices de um grafo G.*

Nesta seção, queremos provar que um grafo G enumerável e conexo é um produto se e somente se ele satisfaz comutatividade à esquerda e à direita, as propriedades de Church-Rosser e Church-Rosser reversa e intransitividade. Começamos pela direção mais simples.

Teorema 6.5. *Se G é um produto, então G satisfaz comutatividade à esquerda e à direita, as propriedades de Church-Rosser e Church-Rosser reversa e intransitividade.*

É interessante notar que, nesta direção da prova, não se faz uso da hipótese 6.3. Isto significa que o teorema 6.5 é verdadeiro para qualquer grafo G. Agora, prosseguimos para provar a outra direção.

Proposição 6.6. *Se um grafo satisfaz comutatividade à esquerda e à direita e as propriedades de Church-Rosser e Church-Rosser reversa, ele também satisfaz as seguintes propriedades:*

1. *Comutatividade Estendida à Esquerda: $\forall x, y, z \in V(xU_j^l y \wedge yU_i^k z \rightarrow \exists u \in V(xU_i^k u \wedge uU_j^l z))$;*
2. *Comutatividade Estendida à Direita: $\forall x, y, z \in V(xU_i^k y \wedge yU_j^l z \rightarrow \exists u \in V(xU_j^l u \wedge uU_i^k z))$.*

onde $uU_i^k v$ e $uU_j^l v$ denotam que existe um caminho não direcionado através de arestas de $\overline{E}_i$ ($\overline{E}_j$, respectivamente) de comprimento k (l) de u para v.

não direcionado de u para v em G.

Definição 6.7. *Seja* $G = \langle V, \overline{E}_1, \ldots, \overline{E}_n \rangle$ *e sejam* $G_k = \langle V, \overline{E}_k \rangle$, $1 \leq k \leq n$, *subgrafos de* G. *Os* k-componentes *são os subgrafos maximais conexos de* G_k, $\{G_k^1, G_k^2, \ldots\}$. *O conjunto de* k*-componentes é enumerável, uma vez que* G *é enumerável.*

Da mesma forma que apresentamos os componentes de dimensão um do grafo na definição acima, também precisamos definir os componentes de *codimensão* um. É interessante notar que no caso particular de produtos bidimensionais, estas definições são equivalentes.

Definição 6.8. *Seja* $G = \langle V, \overline{E}_1, \ldots, \overline{E}_n \rangle$ *e sejam* $\widetilde{G}_k = \langle V, \overline{E}_1, \ldots, \overline{E}_{k-1}, \overline{E}_{k+1}, \ldots, \overline{E}_n \rangle$, $1 \leq k \leq n$, *subgrafos de* G. *Os* $\widetilde{k}$-componentes *são os subgrafos maximais conexos de* $\widetilde{G}_k$, $\{\widetilde{G}_k^1, \widetilde{G}_k^2, \ldots\}$. *O conjunto de* $\widetilde{k}$*-componentes também é enumerável, uma vez que* G *é enumerável.*

Definição 6.9. *Denotamos por* $\overline{E}_{\widetilde{k}}$ *o conjunto de arestas* $\overline{E}_1 \cup \ldots \cup \overline{E}_{k-1} \cup \overline{E}_{k+1} \cup \ldots \cup \overline{E}_n$.

Um aspecto básico a respeito de k-componentes e $\widetilde{k}$-componentes que precisa ser notado é que se estamos em um vértice u em algum k-componente e percorremos um caminho de arestas em $\overline{E}_k$, permanecemos no mesmo k-componente e se estamos em um vértice u em algum $\widetilde{k}$-componente e percorremos um caminho de arestas em $\overline{E}_{\widetilde{k}}$, permanecemos no mesmo $\widetilde{k}$-componente.

Proposição 6.10. *Dois* k*-componentes distintos (ou dois* $\widetilde{k}$*-componentes distintos) não possuem vértices em comum.*

Observação 6.11. *De agora em diante, toda vez que precisarmos considerar um par de* k*-componentes* G_k^i *e* G_k^j *ou um par de* $\widetilde{k}$*-componentes* $\widetilde{G}_k^i$ *e* $\widetilde{G}_k^j$, *os dois componentes do par* não precisam *ser distintos, a não ser que isto seja explicitamente mencionado.*

Definição 6.12. *Para* $1 \leq k, l \leq n$, $k \neq l$, *dizemos que um* k*-componente* G_k^i *é* l-vizinho ao k*-componente* G_k^j *se existe uma aresta* $\langle u, w \rangle \in \overline{E}_l$ *tal que* $u \in G_k^i$ *e* $w \in G_k^j$. *Note que é possível que um* k*-componente seja* l*-vizinho a si mesmo.*

Definição 6.13. *Para* $1 \leq k, l \leq n$, $k \neq l$, *seja* f_{kl}^{ij} *o mapeamento (possivelmente parcial e multivalorado) que associa a cada vértice* $u \in G_k^i$ *o conjunto de vértices* w

tais que $\langle u, w\rangle \in \overline{E}_l$ *e* $w \in G_k^j$*. Dizemos que* f_{kl}^{ij} *é o* mapeamento l-induzido *de* G_k^i *para* G_k^j*.*

Proposição 6.14. *Sejam* G *um grafo que satisfaz comutatividade à esquerda e à direita, as propriedades de Church-Rosser e Church-Rosser reversa e intransitividade e* G_k^i *l-vizinho a* G_k^j*. Então, o mapeamento l-induzido* f_{kl}^{ij} *de* G_k^i *para* G_k^j *é um isomorfismo.*

Definição 6.15. *Se* G_k^i *é l-vizinho a* G_k^j *e o mapeamento l-induzido* f_{kl}^{ij} *é um isomorfismo entre* G_k^i *e* G_k^j*, chamamos* f_{kl}^{ij} *de* isomorfismo l-primitivo*.*

Agora, em um caso como o da proposição acima, em que todos os mapeamentos induzidos entre k-componentes vizinhos são isomorfismos, podemos facilmente estender os isomorfismos além de k-componentes imediatamente vizinhos.

Seja $L = [l_1, \ldots, l_n]$ uma lista finita. Para definir a composição de isomorfismos entre k-componentes, estendemos a notação e escrevemos os mapeamentos como f_{kL}^{ij}. Esta notação pode ser facilmente utilizada para denotar os mapeamentos l-induzidos e os isomorfismos l-primitivos originais, uma vez que neste caso temos apenas que tomar $L = [l]$.

Observação 6.16. *Se todos os elementos no conjunto* $\{f_{kL_1}^{i,i+1}, f_{kL_2}^{i+1,i+2}, \ldots, f_{kL_{j-i}}^{j-1,j}\}$ *são isomorfismos, então*

$$f_{kL}^{ij} = f_{kL_{j-i}}^{j-1,j} \circ \cdots \circ f_{kL_2}^{i+1,i+2} \circ f_{kL_1}^{i,i+1},$$

onde $L = L_1 \circ L_2 \circ \cdots \circ L_{j-i}$*, também é um isomorfismo.*

Observação 6.17. *Se* f_{kL}^{ij} *é um isomorfismo, então seu inverso também é um isomorfismo e é denotado por* $f_{kL'}^{ji}$*, onde* L' *é a lista simétrica a* L*.*

Definição 6.18. *Se* f_{kL}^{ij} *é um isomorfismo primitivo ou é obtido de isomorfismos primitivos utilizando composição e inverso, chamamos* f_{kL}^{ij} *de* isomorfismo ortogonal *ou* O-isomorfismo*.*

Lema 6.19. *Seja* G *um grafo que satisfaz comutatividade à esquerda e à direita, as propriedades de Church-Rosser e Church-Rosser reversa e intransitividade. Então, para todos os pares* G_k^i *e* G_k^j *de k-componentes, existe um O-isomorfismo* f_{kL}^{ij} *entre*

eles. Isto significa que todos os k-componentes são isomorfos entre si e que o conjunto de caminhos não direcionados através de arestas de $\overline{E}_{\widetilde{k}}$ entre G_k^i e G_k^j induz um isomorfismo entre os dois k-componentes.

Definição 6.20. *Seja G um grafo. Se G satisfaz intransitividade e, para todos os pares G_k^i e G_k^j de k-componentes, existe um O-isomorfismo f_{kL}^{ij} entre eles, dizemos que G é* bem comportado.

Lema 6.21. *Todo $\widetilde{k}$-componente contém como subgrafo ao menos um l-componente completo, para todo $l \neq k$.*

Lema 6.22. *Sejam G um grafo bem comportado, G_k^i um k-componente e $\widetilde{G}_k^j$ um $\widetilde{k}$-componente. Então, $\mathcal{V}(G_k^i) \cap \mathcal{V}(\widetilde{G}_k^j)$ é um conjunto unitário.*

Corolário 6.23. *Sejam G um grafo bem comportado, H um subgrafo de G que contém como subgrafo ao menos um k-componente completo e $\widetilde{G}_k^i$ um $\widetilde{k}$-componente. Então, $\mathcal{V}(H) \cap \mathcal{V}(\widetilde{G}_k^i) \neq \emptyset$.*

Corolário 6.24. *Sejam G um grafo bem comportado e $S = \{\widetilde{G}_k^{i_k} : 1 \leq k \leq n\}$ um conjunto que contém um $\widetilde{k}$-componente para cada $1 \leq k \leq n$. Então, $\bigcap\{\mathcal{V}(H) : H \in S\} \neq \emptyset$.*

Lema 6.25. *Seja G um grafo. Se G satisfaz intransitividade e, para todos os pares G_k^i e G_k^j de k-componentes, existe um O-isomorfismo f_{kL}^{ij} entre eles, então G é (isomorfo a) um produto.*

Teorema 6.26. *Se G satisfaz comutatividade à esquerda e à direita, as propriedades de Church-Rosser e Church-Rosser reversa e intransitividade, então G é um produto.*

Demonstração. O teorema segue diretamente dos lemas 6.19 e 6.25. □

Teorema 6.27. *Seja G um grafo. G é um produto se e somente se G satisfaz comutatividade à esquerda e à direita, as propriedades de Church-Rosser e Church-Rosser reversa e intransitividade.*

Demonstração. O teorema segue diretamente dos teoremas 6.5 e 6.26. □

6.4 Limites da Expressividade Modal

Nesta seção, mostramos que a propriedade de intransitividade não pode ser expressa em uma linguagem modal básica. De fato, também mostramos que nenhuma condição que seja necessária e suficiente para um grafo ser um produto pode ser expressa em uma linguagem modal básica. Apesar de termos nos mantido restritos, na seção anterior, a grafos enumeráveis e conexos, esta restrição não é necessária para os resultados de expressividade apresentados nesta seção.

Nesta seção, definimos uma linguagem modal com uma família de operadores modais: $\Diamond_i$, $\Diamond_i^{-1}$ e $\blacklozenge_i$, para todo $1 \leq i \leq n$.

Definição 6.28. *Consideramos uma linguagem multimodal que consiste de um conjunto enumerável* Φ *de símbolos proposicionais, os operadores booleanos* $\neg$ *e* $\wedge$*, a constante* $\top$ *e os operadores modais (ou modalidades)* $\Diamond_i$, $\Diamond_i^{-1}$ *e* $\blacklozenge_i$*, para todo* $1 \leq i \leq n$*. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_i\varphi \mid \Diamond_i^{-1}\varphi \mid \blacklozenge_i\varphi,$$

onde $p \in \Phi$*.*

As fórmulas desta lógica são avaliadas nas n-estruturas e n-modelos de Kripke usuais. É importante notar que as modalidades da linguagem acima *não são* modalidades graduadas, são apenas uma coleção finita de modalidades de uma linguagem multimodal básica. Para as definições apropriadas de satisfazibilidade das fórmulas desta lógica, o capítulo 2 pode ser consultado.

Conforme é mostrado em [4] e [5], um grafo satisfaz comutatividade à esquerda e à direita e a propriedade de Church-Rosser se as fórmulas a seguir são válidas nele para todos os pares $1 \leq i, j \leq n$, $i \neq j$:

1. $com_{ij} = \Diamond_j\Diamond_i p \leftrightarrow \Diamond_i\Diamond_j p$ (comutatividade à esquerda e à direita);

2. $chr_{ij} = \Diamond_i\Box_j p \rightarrow \Box_j\Diamond_i p$ (propriedade de Church-Rosser).

Então, um grafo satisfaz a propriedade de Church-Rosser reversa se a fórmula a seguir, análoga a chr_{ij}, é válida nele para todos os pares $1 \leq i, j \leq n$, $i \neq j$:

3. $rchr_{ij} = \Diamond_i^{-1}\Box_j^{-1}\varphi \rightarrow \Box_j^{-1}\Diamond_i^{-1}\varphi$.

Teorema 6.29. *Nem intransitividade nem sua negação podem ser expressas na linguagem modal básica.*

Demonstração. Na figura 6.2, seja $f = \{(1,a),(2,b),(3,a),(4,b)\}$ e $g = \{(a,A),(b,A)\}$. É simples mostrar que f e g são morfismos limitados sobrejetivos. Também não é difícil ver que o primeiro e terceiro grafos satisfazem intransitividade, enquanto o segundo não satisfaz. Pelo corolário 2.18, uma vez que nem intransitividade nem sua negação são preservadas por imagens mórficas limitadas, elas não podem ser expressas na linguagem modal básica. □

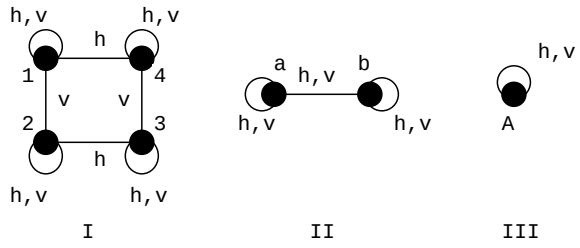

Figura 6.2: O grafo III é uma imagem mórfica limitada do grafo II, que é uma imagem mórfica limitada do grafo I (cada aresta não direcionada representa um par de arestas simétricas)

Teorema 6.30. *Nenhuma condição que seja necessária e suficiente para um grafo* ser *um produto ou para um grafo* não ser *um produto pode ser expressa na linguagem modal básica.*

Demonstração. Tomamos novamente os mesmos morfismos limitados entre os grafos na figura 6.2. Não é difícil ver que o primeiro e o terceiro grafos são produtos, enquanto o segundo não é. Pelo corolário 2.18, uma vez que nem a propriedade de ser um produto nem a propriedade de não ser um produto são preservadas por imagens mórficas limitadas, elas não podem ser expressas na linguagem modal básica. □

6.5 Uma Extensão Híbrida

Como foi mostrado na seção anterior, a linguagem modal básica não possui poder expressivo suficiente para descrever a propriedade que queremos. Para alcançarmos nosso objetivo, precisamos de uma linguagem com maior poder expres-

sivo. Da mesma maneira que fizemos no capítulo anterior, optamos por uma linguagem híbrida. Nesta seção, descrevemos uma linguagem híbrida e a utilizamos para expressar intransitividade.

Definição 6.31. *Consideramos uma linguagem multimodal híbrida que consiste de um conjunto enumerável* Φ *de símbolos proposicionais e um conjunto enumerável* Ω *de nominais tais que* $\Phi \cap \Omega = \emptyset$*, os operadores booleanos* $\neg$ *e* $\wedge$*, a constante* $\top$*, os operadores modais (ou modalidades)* $\Diamond_i$*,* $\Diamond_i^{-1}$ *e* $\blacklozenge_i$*, para todo* $1 \leq i \leq n$*, e os operadores de satisfação* $@_a$*, para todo nominal* a*. As fórmulas são definidas da seguinte forma:*

$$\varphi ::= p \mid a \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_i\varphi \mid \Diamond_i^{-1}\varphi \mid \blacklozenge_i\varphi \mid @_a\varphi,$$

onde $p \in \Phi$ *e* $a \in \Omega$*.*

É comum na literatura o uso das letras i, j e k para denotar nominais. Entretanto, como já estamos utilizando estas letras, neste capítulo, para os múltiplos conjuntos de arestas dos grafos, escolhemos denotar os nominais com as letras a, b e c.

As fórmulas desta lógica são avaliadas nas n-estruturas e n-modelos de Kripke híbridos usuais. Para as definições apropriadas de satisfazibilidade das fórmulas desta lógica, o capítulo 2 pode ser consultado.

Utilizando esta linguagem híbrida, podemos agora expressar intransitividade.

Teorema 6.32. *Um grafo* G *satisfaz intransitividade se e somente se* $G \Vdash int_{ij}$ *para todos os pares* $1 \leq i, j \leq n$*,* $i \neq j$*, onde* int_{ij} *é a fórmula*

$$int_{ij} = (a \wedge \neg b \wedge \blacklozenge_j(b \wedge \neg c \wedge \blacklozenge_i c)) \rightarrow \neg c.$$

É interessante notar que não precisamos utilizar operadores de satisfação para descrever intransitividade. Entretanto, nós temos operadores de satisfação na linguagem porque eles são muito úteis em provas de completude de axiomatizações de lógicas híbridas, como é mostrado na seção 6.7.

6.6 Verificação de Produtos

Nesta seção, analisamos a complexidade computacional de verificar se um grafo finito e conexo é um produto.

Teorema 6.33 ([17])**.** *O problema da verificação de modelo para a lógica apresentada na definição 6.31 é polinomial (linear) no produto do tamanho do modelo e do comprimento da fórmula.*

Assim como no capítulo anterior, podemos determinar um limite superior para a complexidade do problema da verificação de estrutura baseado na complexidade do problema da verificação de modelo correspondente. Temos que $\mathcal{F} \Vdash \phi$ se e somente se $S_{\mathcal{M}}(\neg\phi) = \emptyset$ para todo modelo $\mathcal{M}$ de $\mathcal{F}$. Então, um método algorítmico de verificar se $\mathcal{F} \Vdash \phi$ consiste em aplicar o algoritmo de verificação de modelo a todos os pares $(\neg\phi, \mathcal{M})$, onde $\mathcal{M}$ é um modelo de $\mathcal{F}$.

Portanto, se FC é a complexidade do problema da verificação de estrutura e MC é a complexidade do problema de verificação de modelo, temos

$$FC = O(2^{|p|\times m} \times m^{|a|} \times MC), \tag{6.1}$$

onde $|p|$ é o número de símbolos proposicionais distintos que ocorrem em uma dada fórmula ϕ, $|a|$ é o número de nominais distintos que ocorrem em ϕ e m é o número de vértices em $\mathcal{F}$. A distinção entre símbolos proposicionais e nominais na equação acima surge da restrição especial na valoração de nominais. Precisamos aplicar o algoritmo de verificação de modelo para todo modelo $\mathcal{M}$ da estrutura dada $\mathcal{F}$. Cada símbolo proposicional$p$ que ocorre em ϕ pode receber 2^m possíveis valorações $\mathbf{V}(p)$, enquanto cada nominal a pode receber apenas m possíveis valorações $\mathbf{V}(a)$.

Teorema 6.34. *O problema da verificação de estrutura para a lógica apresentada na definição 6.31 é polinomial (linear) no comprimento da fórmula e EXPTEMPO no tamanho da estrutura, no número de símbolos proposicionais distintos que ocorrem na fórmula e no número de nominais distintos que ocorrem na fórmula.*

Demonstração. Este resultado segue diretamente da discussão acima. □

Deve-se notar que este cálculo da complexidade do problema da verificação de estrutura é apenas um limite superior geral e que ele pode possivelmente ser reduzido em algumas situações concretas.

A partir da equação (6.1), podemos ver que, do ponto de vista da complexidade computacional, é interessante tentar expressar as propriedades desejadas com fórmulas que utilizem apenas nominais e não símbolos proposicionais, uma vez que a

presença de símbolos proposicionais faz com que a verificação da propriedade custe uma quantidade de tempo exponencial no tamanho da estrutura.

Definição 6.35. *Uma* fórmula pura *é uma fórmula sem ocorrência de símbolos proposicionais.*

Então, fórmulas puras são interessantes do ponto de vista da complexidade do problema da verificação de estrutura. Além disso, como é mostrado na seção 6.7, fórmulas puras também possuem vantagens quando usadas como axiomas em um sistema axiomático.

A fórmula no teorema 6.32, que descreve intransitividade, já é pura. Agora, precisamos encontrar fórmulas puras que descrevam comutatividade à esquerda e à direita e as propriedades de Church-Rosser e de Church-Rosser reversa.

Teorema 6.36. *Um grafo G satisfaz comutatividade à esquerda e à direita se e somente se $G \Vdash com^*_{ij}$ para todos os pares $1 \leq i, j \leq n$, $i \neq j$, onde com^*_{ij} é a fórmula*

$$com^*_{ij} = \Diamond_j \Diamond_i a \leftrightarrow \Diamond_i \Diamond_j a.$$

Podemos ver que para comutatividade à esquerda e à direita, a tarefa de encontrar uma fórmula pura que descreva as propriedades foi bastante simples, já que tivemos apenas que substituir o símbolo proposicional p na fórmula original com_{ij} pelo nominal a. Para a propriedade de Church-Rosser, esta tarefa é mais difícil. A simples substituição de p por a na fórmula original chr_{ij} não funciona. Entretanto, é possível descrever a propriedade de Church-Rosser com uma fórmula pura. Em [39], uma fórmula pura para a propriedade de Church-Rosser é apresentada: $chr^*_{ij} = \Diamond_j a \rightarrow \Box_i \Diamond_j \Diamond_i^{-1} a$. Também é mostrado em [39] que não é possível descrever a propriedade de Church-Rosser com uma fórmula pura sem o uso de uma modalidade conversa $\Diamond_i^{-1}$.

Teorema 6.37. *Um grafo G satisfaz a propriedade de Church-Rosser se e somente se $G \Vdash chr^*_{ij}$ para todos os pares $1 \leq i, j \leq n$, $i \neq j$, onde chr^*_{ij} é a fórmula*

$$chr^*_{ij} = \Diamond_j a \rightarrow \Box_i \Diamond_j \Diamond_i^{-1} a.$$

Teorema 6.38. *Um grafo G satisfaz a propriedade de Church-Rosser reversa se e somente se $G \Vdash rchr^*_{ij}$ para todos os pares $1 \leq i, j \leq n$, $i \neq j$, onde $rchr^*_{ij}$ é a fórmula*

$$rchr^*_{ij} = \Diamond_j^{-1} a \rightarrow \Box_i^{-1} \Diamond_j^{-1} \Diamond_i a.$$

Agora, podemos determinar quão complexo é testar se um grafo finito e conexo é um produto. Pelos teoremas 6.27, 6.32, 6.36, 6.37 e 6.38, um grafo finito e conexo G é um produto se e somente se $G \Vdash pro$, onde pro é a fórmula

$$pro = \bigwedge_{1 \leq i,j \leq n, i \neq j} pro_{ij}$$

onde

$$pro_{ij} = com^*_{ij} \wedge chr^*_{ij} \wedge rchr^*_{ij} \wedge int_{ij}$$

Seja $FC(\phi)$ a complexidade da verificação de estrutura para $G \Vdash \phi$. Então, queremos calcular $FC(pro)$. Como existem $O(n^2)$ pares $1 \leq i, j \leq n$, $i \neq j$, temos que

$$FC(pro) = O(n^2) \times FC(pro_{ij}),$$

Primeiramente, devemos notar que não há símbolos proposicionais em pro_{ij} e o comprimento de pro_{ij} é constante e não depende do tamanho do grafo. Além disso, com^*_{ij}, chr^*_{ij} e $rchr_{ij}$ possuem apenas um nominal e int_{ij} possui três nominais distintos. Entretanto, como a verificação de estrutura de uma fórmula ϕ é feita através de verificações de modelo da fórmula $\neg\phi$, podemos reduzir o número de nominais envolvidos no teste utilizando o fato de que valorações de nominais são sempre conjuntos unitários.

$$\neg int_{ij} \equiv \neg(\neg(a \wedge \neg b \wedge \blacklozenge_j(b \wedge \neg c \wedge \blacklozenge_i c)) \vee \neg c) \equiv a \wedge \neg b \wedge \blacklozenge_j(b \wedge \neg c \wedge \blacklozenge_i c) \wedge c$$

Mas uma fórmula $a \wedge c$ em lógica híbrida significa que as valorações de a e c são iguais, então podemos utilizar apenas um destes nominais, substituindo todas as ocorrências do segundo pelo primeiro. Além disso, uma fórmula $a \wedge \neg b$ significa que a e b denotam vértices distintos, então é redundante escrever em outro ponto da fórmula $b \wedge \neg a$. Logo,

$$\neg int_{ij} \equiv a \wedge \neg b \wedge \blacklozenge_j(b \wedge \blacklozenge_i a)$$

Então, podemos realizar o teste com apenas dois nominais. Levando em consideração estas observações e a fórmula na equação (6.1), temos que

$$FC(pro_{ij}) = O(m^2 \times MC),$$

onde, neste caso, MC polinomial (de fato, linear) no tamanho do grafo. Isto implica que

$$FC(pro) = O(n^2 \times m^2 \times MC).$$

Teorema 6.39. *A complexidade para verificar se um grafo finito e conexo é um produto utilizando-se a fórmula pro acima é polinomial (cúbica) no tamanho do grafo e polinomial (quadrática) no número de conjuntos distintos de arestas (número de dimensões).*

Demonstração. Este resultado segue diretamente da discussão acima. □

6.7 Axiomatizações Híbridas de Produtos de Lógicas Modais

Produtos de grafos aparecem naturalmente como uma possível extensão da semântica de Kripke comum para lógicas modais multidimensionais. Nesta seção, apresentamos o conceito de *produto de lógicas modais*, onde a semântica é construída utilizando-se produtos de estruturas de Kripke, e então utilizamos lógica híbrida para construir axiomatizações completas para uma grande classe de produtos de lógicas modais.

Seja Φ um conjunto enumerável de símbolos proposicionais e Ω um conjunto enumerável de nominais tais que $\Phi \cap \Omega = \emptyset$. Para um conjunto finito de operadores modais $\mathcal{O}$, definimos o conjunto de fórmulas híbridas $HFor(\mathcal{O})$ através da regra

$$\varphi ::= p \mid a \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid O\varphi \mid @_a\varphi,$$

onde $p \in \Phi$, $a \in \Omega$ e $O \in \mathcal{O}$. Também definimos o conjunto de fórmulas modais básicas $MFor(\mathcal{O})$ como o subconjunto de $HFor(\mathcal{O})$ que contém apenas fórmulas sem nominais e operadores de satisfação. Quando não é relevante se estamos trabalhando com uma linguagem híbrida ou com uma linguagem modal básica, escrevemos $For(\mathcal{O})$ para o conjunto de fórmulas na linguagem em questão.

Existem duas maneiras comuns de definir uma lógica modal: a maneira "sintática" e a maneira "semântica". Seja $F = For(\mathcal{O})$ o conjunto de todas as fórmulas em uma dada linguagem (seja ela uma linguagem híbrida ou uma linguagem modal básica). A maneira sintática consiste em tomar um conjunto $\mathcal{A} \subseteq F$ de axiomas e um conjunto $\mathcal{R}$ de regras e definir uma lógica modal L como o menor conjunto de fórmulas tal que $\mathcal{A} \subseteq L$ e L é fechado sob as regras em $\mathcal{R}$. A maneira semântica consiste em tomar uma classe $\mathcal{C}$ de estruturas e definir uma lógica modal $L = \mathrm{Log}(F, \mathcal{C})$ como o conjunto

$$\mathrm{Log}(F, \mathcal{C}) = \{\phi \in F : \mathcal{F} \Vdash \phi, \text{ para todo } \mathcal{F} \in \mathcal{C}\}.$$

Também podemos definir os duais dos conjuntos $\mathrm{Log}(F, \mathcal{C})$. Sejam Fr_F a classe de estruturas em que as fórmulas de F são avaliadas e $\Sigma \subseteq F$ um conjunto de fórmulas. Definimos a classe de estruturas $\mathrm{Fr}(F, \Sigma)$ como

$$\mathrm{Fr}(F, \Sigma) = \{\mathcal{F} \in Fr_F : \mathcal{F} \Vdash \phi, \text{ para toda } \phi \in \Sigma\}.$$

Se $L = \mathrm{Log}(F, \mathcal{C})$ é uma lógica definida semanticamente, escrevemos $\Vdash_{\mathcal{C}} \phi$ para denotar que $\phi \in L$.

Suponha que $\nVdash_{\mathcal{C}} \neg\phi$. Isto significa que existe uma estrutura $\mathcal{F} \in \mathcal{C}$ tal que $\mathcal{F} \nVdash \neg\phi$. Isto, por outro lado, significa que existe um modelo $\mathcal{M}$ da estrutura $\mathcal{F}$ e um vértice v tais que $\mathcal{M}, v \Vdash \phi$. Então, ϕ é satisfeita em um vértice de um modelo de uma estrutura em $\mathcal{C}$.

Definição 6.40. *Dizemos que uma fórmula ϕ é* $\mathcal{C}$-satisfazível *se ela é satisfeita em um vértice de um modelo de uma estrutura em $\mathcal{C}$ (de maneira equivalente, se $\nVdash_{\mathcal{C}} \neg\phi$). Dizemos que um conjunto Ψ de fórmulas é $\mathcal{C}$-satisfazível se existe um vértice de um modelo de uma estrutura em $\mathcal{C}$ que satisfaz todas as fórmulas $\phi \in \Psi$.*

Definição 6.41. *Uma lógica modal L é chamada* Kripke-completa *se $L = Log(F, \mathcal{C})$ para alguma classe $\mathcal{C}$ de estruturas. Neste caso, dizemos que L é caracterizada (ou determinada) por $\mathcal{C}$.*

Seja $MFor_1 = MFor(\{\Diamond\})$ o conjunto de fórmulas na linguagem monomodal básica. Como um exemplo trivial de uma lógica modal que é um subconjunto de $MFor_1$ e pode ser definida desta maneira semântica, o conjunto $\mathbf{K} = \{\phi \in MFor_1 :$

$\Vdash \phi\}$ de todas as fórmulas válidas contidas em $MFor_1$ é uma lógica modal, uma vez que $\mathbf{K} = \mathrm{Log}(MFor_1, \mathcal{C})$, onde $\mathcal{C}$ neste caso é a classe de *todas* as estruturas. O próprio conjunto $MFor_1$ também é uma lógica modal, uma vez que $MFor_1 = \mathrm{Log}(MFor_1, \emptyset)$.

A notação **K** é usual na literatura de lógica modal para a lógica na linguagem monomodal básica determinada pela classe de todas as estruturas. Notações comuns para outras lógicas modais que utilizamos nesta seção são **K4**, **K5**, **T**, **B**, **D**, **Alt**, **B4** e **S5** para, respectivamente, as lógicas na linguagem monomodal básica determinadas pelas classes de estruturas transitivas, estruturas euclidianas, estruturas reflexivas, estruturas simétricas, estruturas seriais, estruturas funcionais, estruturas transitivas e simétricas e estruturas transitivas, simétricas e reflexivas[3].

Sejam $MFor_n = MFor(\{\Diamond_1, \ldots, \Diamond_n\})$, $HFor_1 = HFor(\{\Diamond\})$ e $HFor_n = HFor(\{\Diamond_1, \ldots, \Diamond_n\})$ os conjuntos de fórmulas nas linguagens multimodal básica, monomodal híbrida e multimodal híbrida, respectivamente. Se $L = \mathrm{Log}(MFor_1, \mathcal{C})$, então denotamos por $L(n), L(@)$ e $L(n, @)$ os conjuntos $\mathrm{Log}(MFor_n, \mathcal{C})$, $\mathrm{Log}(HFor_1, \mathcal{C})$ e $\mathrm{Log}(HFor_n, \mathcal{C})$, respectivamente. Então, temos $\mathbf{K4}(@)$, $\mathbf{S5}(n)$ e assim por diante.

Um *produto de lógicas modais* é uma lógica multimodal que é definida semanticamente da seguinte forma.

Definição 6.42. *Seja $\{L_i : 1 \leq i \leq n\}$ um conjunto finito de lógicas modais Kripke-completas definidas sobre os conjuntos $For(\mathcal{O}_i)$, respectivamente, tais que os conjuntos de modalidades $\mathcal{O}_i$, $1 \leq i \leq n$, são disjuntos dois a dois (neste capítulo, consideramos que cada L_i é monomodal, possuindo apenas uma modalidade $\Diamond_i$). Seja $F = For(\cup_i \mathcal{O}_i)$. Consideramos que $For(\mathcal{O}_i)$, $1 \leq i \leq n$, são ou todos conjuntos de fórmulas modais básicas ou todos conjuntos de fórmulas híbridas. Então, o produto de $L_1, \ldots, L_n$ é a lógica multimodal definida como*

$$L_1 \times \cdots \times L_n = Log(F, \mathcal{C}),$$

onde

$$\mathcal{C} = \{\mathcal{F}_1 \times \cdots \times \mathcal{F}_n : \mathcal{F}_i \in Fr(L_i), \text{ para todo } 1 \leq i \leq n\}.$$

Definição 6.43. *Denotamos por L^n o produto $L \times \cdots \times L$, onde L ocorre n vezes.*

[3]Mais detalhes sobre estas classes de estruturas podem ser encontrados em [4] e [40].

Por exemplo, $\mathbf{K} \times \mathbf{K}$ é a lógica modal determinada pela classe de todos os produtos de estruturas da forma $\mathcal{F}_1 \times \mathcal{F}_2$, $\mathbf{K4} \times \mathbf{S5}$ é a lógica modal determinada por todos os produtos de estruturas da forma $\mathcal{F}_1 \times \mathcal{F}_2$ onde $\mathcal{F}_1$ é transitivo e $\mathcal{F}_2$ é transitivo, simétrico e reflexivo e $\mathbf{S5}^n$ é a lógica modal determinada pela classe de todos os produtos de estruturas da forma $\mathcal{F}_1 \times \cdots \times \mathcal{F}_n$ onde cada $\mathcal{F}_i$, $1 \leq i \leq n$ é transitivo, simétrico e reflexivo.

É interessante notar que os produtos de lógicas modais, por serem definidos sobre lógicas modais Kripke-completas, são também Kripke-completos por definição.

Agora, uma questão relevante a respeito de produtos de lógicas modais é o chamado *problema da axiomatização*. Dado o fato de que um produto $L = L_1 \times \cdots \times L_n$ é definido semanticamente, é também possível encontrar uma definição sintática correspondente para L?

De forma a definir melhor o problema da axiomatização, precisamos das noções de correção e completude de uma axiomatização. Seja $\mathbb{A} = (\mathcal{A}, \mathcal{R})$ um sistema axiomático com o conjunto de axiomas $\mathcal{A}$ e o conjunto de regras $\mathcal{R}$. Seja $L_{\mathbb{A}}$ o menor conjunto de fórmulas na linguagem em consideração tal que $\mathcal{A} \subseteq L_{\mathbb{A}}$ e $L_{\mathbb{A}}$ é fechado sob as regras em $\mathcal{R}$. Se $\phi \in L_{\mathbb{A}}$, dizemos que ϕ é um *teorema* de $\mathbb{A}$ e usamos a notação $\vdash_{\mathbb{A}} \phi$.

Definição 6.44. *Dizemos que uma fórmula ϕ é* $\mathbb{A}$-consistente *se $\nvdash_{\mathbb{A}} \neg\phi$. Dizemos que um conjunto finito Ψ_0 de fórmulas é* $\mathbb{A}$-consistente *se $\nvdash_{\mathbb{A}} \neg \bigwedge\{\phi : \phi \in \Psi_0\}$. Finalmente, dizemos que um conjunto Ψ de fórmulas é* $\mathbb{A}$-consistente *se todo subconjunto finito $\Psi_0 \subseteq \Psi$ é $\mathbb{A}$-consistente.*

Definição 6.45. *Se $\mathbb{A} = (\mathcal{A}, \mathcal{R})$ é um sistema axiomático e Σ é um conjunto de fórmulas, denotamos por $\mathbb{A} + \Sigma$ o sistema axiomático $(\mathcal{A} \cup \Sigma, \mathcal{R})$.*

Definição 6.46. *Seja $L = Log(F, \mathcal{C})$ uma lógica modal semanticamente definida e $\mathbb{A}$ um sistema axiomático. Dizemos que $\mathbb{A}$ é* correto *para L se e somente se $\vdash_{\mathbb{A}} \phi$ implica $\Vdash_{\mathcal{C}} \phi$ para toda fórmula ϕ na linguagem em consideração. De maneira equivalente, como pode ser visto diretamente a partir das definições de $\mathbb{A}$-consistência (definição 6.44) e $\mathcal{C}$-satisfazibilidade (definição 6.40), $\mathbb{A}$ é* correto *para L se e somente se toda fórmula $\mathcal{C}$-satisfazível na linguagem em consideração é $\mathbb{A}$-consistente.*

Teorema 6.47. *$\mathbb{A}$ é* correto *para L se e somente se todo conjunto $\mathcal{C}$-satisfazível de fórmulas na linguagem em consideração é $\mathbb{A}$-consistente.*

Definição 6.48. *Seja $L = Log(F, \mathcal{C})$ uma lógica modal semanticamente definida e $\mathbb{A}$ um sistema axiomático. Dizemos que $\mathbb{A}$ é* completo *para L se e somente se $\Vdash_{\mathcal{C}} \phi$ implica $\vdash_{\mathbb{A}} \phi$ para toda fórmula ϕ na linguagem em consideração. De maneira equivalente, $\mathbb{A}$ é* completo *para L se e somente se toda fórmula $\mathbb{A}$-consistente na linguagem em consideração é $\mathcal{C}$-satisfazível.*

Em geral, completude não implica automaticamente que todo conjunto $\mathbb{A}$-consistente de fórmulas na linguagem em consideração é $\mathcal{C}$-satisfazível. Isto é diferente do caso de correção, onde podíamos saltar de fórmulas para conjuntos de fórmulas (teorema 6.47).

Definição 6.49. *Seja $L = Log(F, \mathcal{C})$ uma lógica modal semanticamente definida e $\mathbb{A}$ um sistema axiomático. Dizemos que $\mathbb{A}$ é* fortemente completo *para L se e somente se todo conjunto $\mathbb{A}$-consistente de fórmulas na linguagem em consideração é $\mathcal{C}$-satisfazível. Completude forte implica completude, mas, em geral, completude não implica completude forte.*

Então, para um produto $L = L_1 \times \cdots \times L_n$, se possuímos um sistema axiomático correto e completo $\mathbb{A}_i$ para cada lógica L_i, $1 \leq i \leq n$, existe uma maneira de combiná-los de forma a obter um sistema axiomático correto e completo para L?

Na linguagem modal básica, esta questão é mais complexa do que pode parecer à primeira vista e não há um método geral na literatura para gerar um sistema axiomático correto e completo para o produto de lógicas modais a partir de sistemas axiomáticos corretos e completos para cada um dos componentes do produto.

O problema parece vir do fato de que, como o teorema 6.30 mostra, nenhuma condição que seja necessária e suficiente para um grafo ser um produto pode ser expressa em uma linguagem modal básica. Se L_i é sintaticamente definida por um sistema axiomático $\mathbb{A}_i = (\mathcal{A}_i, \mathcal{R}_i)$, for $1 \leq i \leq n$ e $C = \{com_{ij} \wedge chr_{ij} : 1 \leq i, j \leq n, i \neq j\}$, então $[L_1, \ldots, L_n]$ denota a lógica modal sintaticamente definida por $\mathbb{C} = (\cup_i \mathcal{A}_i, \cup_i \mathcal{R}_i) + C$. $[L_1, \ldots, L_n]$ é chamado de *comutador* de $L_1, \ldots, L_n$. Como comutatividade à esquerda e à direita e a propriedade de Church-Rosser são condições necessárias para um grafo ser um produto, temos que $[L_1, \ldots, L_n] \subseteq L_1 \times \cdots \times L_n$. Entretanto, como elas não são suficientes, não temos em geral que $L_1 \times \cdots \times L_n \subseteq [L_1, \ldots, L_n]$. Lógicas $[L_1, \ldots, L_n]$ para as quais esta segunda inclusão

é verdadeira são chamadas de *produto-equivalentes*[4].

A prova, para lógicas L_i, $1 \leq i \leq n$, que $\phi \in L_1 \times \cdots \times L_n$ (que é uma lógica semanticamente definida) implica $\phi \in [L_1, \ldots, L_n]$ (que é uma lógica sintaticamente definida) não é nada a mais do que uma prova de completude para $\mathbb{C}$ com respeito a $L_1 \times \cdots \times L_n$. O método padrão para provas de completude para lógicas modais é a construção dos chamados *modelos canônicos*. Os vértices do modelo canônico são todos os conjuntos maximais consistentes de fórmulas da linguagem em consideração. Como há uma quantidade enumerável de símbolos proposicionais e uma quantidade enumerável de nominais na linguagem, então também há uma quantidade enumerável de fórmulas na linguagem, o que significa que existe uma quantidade enumerável de conjuntos maximais consistentes. Logo, o modelo canônico possui uma estrutura enumerável. A ideia básica da prova de completude usando modelos canônicos é que o modelo canônico deve satisfazer as propriedades semânticas da classe $\mathcal{C}$ de estruturas que caracteriza a lógica. Se isto acontece, podemos então prosseguir com a prova e mostrar que toda fórmula $\mathbb{C}$-consistente é $\mathcal{C}$-satisfazível. Para detalhes sobre provas de completude utilizando modelos canônicos, [10] pode ser consultado.

O que ocorre no caso específico dos comutadores é que, como comutatividade à esquerda e à direita e a propriedade de Church-Rosser não são suficientes para um grafo ser um produto, não é possível garantir que o modelo canônico obtido a partir de $\mathbb{C}$ seja um produto. Então, modelos canônicos são por si só insuficientes para derivar completude. Ou outro método precisa ser utilizado ou, ao menos, alguns passos complementares precisam ser executados para que se alcance a prova de completude. Logo, não há um método geral para provas de completude de axiomatizações de produtos de lógicas modais, mas existem lógicas particulares para as quais existem provas de que elas são produto-equivalentes. Como primeiro passo, $\mathbf{K} \times \mathbf{K}$ é produto-equivalente, isto é, o sistema axiomático de $[\mathbf{K}, \mathbf{K}]$ é completo para $\mathbf{K} \times \mathbf{K}$ [40].

Um resultado interessante, mostrado em [40], informa que para um número arbitrário $n \in \mathbb{N}$, $n > 1$, a lógica $\mathbf{Alt}^n$ é produto-equivalente, isto é, o sistema axiomático do comutador de n cópias de $\mathbf{Alt}$ é completo para $\mathbf{Alt}^n$.

[4]product-matching, em inglês

Outro resultado mostrado em [40] informa que, para todo par L_1 e L_2 de lógicas do grupo das chamadas PTC-lógicas, a lógica $L_1 \times L_2$ é produto-equivalente, isto é, o sistema axiomático de $[L_1, L_2]$ é completo para $L_1 \times L_2$. Também é mostrado em [40] que, em alguns casos em que ao menos uma das lógicas do par não é uma PTC-lógica, $[L_1, L_2]$ não é produto-equivalente. Também existem lógicas para as quais não se sabe se seu comutador é produto-equivalente ou não. Muitas das lógicas mais comuns são, de fato, PTC-lógicas, como **K4**, **T**, **B**, **D** e **S5**. Para a definição formal de uma PTC-lógica e os resultados detalhados, [40] pode ser consultado.

Os resultados de completude para $\mathbf{K} \times \mathbf{K}$ e para produtos de um par de PTC-lógicas não se generalizam para produtos de dimensões maiores, como é mostrado no teorema a seguir, de [41].

Teorema 6.50 ([41])**.** *Seja $n \geq 3$ e seja $L \subseteq MFor_n$ qualquer lógica modal tal que $\mathbf{K}^n \subseteq L \subseteq \mathbf{S5}^n$. Então, L não é finitamente axiomatizável.*

Resumindo, ao trabalharmos com uma linguagem modal básica, não há um método geral para construir axiomatizações completas para produtos de lógicas modais e um grupo significativo de produtos de lógicas com dimensão maior do que dois não pode ser axiomatizado. Como mostramos anteriormente, uma linguagem híbrida oferece a possibilidade de descrever produtos de grafos, então ela também pode ser apropriada para promover uma melhoria nestes resultados a respeito de axiomatizações para produtos de lógicas.

Então, vamos agora considerar lógicas $L_1 \times \cdots \times L_n$ na linguagem híbrida. Primeiramente, $\mathbf{K}(n, @) \subseteq L_1 \times \cdots \times L_n$, então um bom ponto de partida para axiomatizar $L_1 \times \cdots \times L_n$ é obter uma axiomatização completa para $\mathbf{K}(n, @)$. De fato, este ponto de partida vai se mostrar ainda melhor do que o esperado, uma vez que provas de completude para lógicas híbridas podem ser estendidas automaticamente para outras lógicas híbridas sob certas condições, como mostra o teorema 6.52.

Consideramos o conjunto de axiomas e regras mostrado na figura 6.3, onde p e q são símbolos proposicionais, a e b são nominais e φ e ψ são fórmulas.

Teorema 6.51. *O sistema axiomático $\mathbb{A}_{\mathbf{K}(n,@)}$, mostrado na figura 6.3, é correto e fortemente completo para a lógica $\mathbf{K}(n, @)$.*

(LP) Tautologias Proposicionais

(K$_i$) $\vdash \Box_i(p \to q) \to (\Box_i p \to \Box_i q)$

(Du$_i$) $\vdash \Box_i p \leftrightarrow \neg\Diamond_i\neg p$

(K$_@$) $\vdash @_a(p \to q) \to (@_a p \to @_a q)$

(SD) $\vdash @_a p \leftrightarrow \neg@_a\neg p$

(Ref) $\vdash @_a a$

(Agr) $\vdash @_a@_b p \leftrightarrow @_b p$

(Int) $\vdash a \to (p \leftrightarrow @_a p)$

(Back$_i$) $\vdash \Diamond_i@_a p \to @_a p$

(MP) Se $\vdash \varphi$ e $\vdash \varphi \to \psi$, então $\vdash \psi$

(Gen$_i$) Se $\vdash \varphi$, então $\vdash \Box_i\varphi$

(Gen$_@$) Se $\vdash \varphi$, então $\vdash @_a\varphi$

(Nam) Se $\vdash @_a\varphi$ e a não ocorre em φ, então $\vdash \varphi$

(BG$_i$) Se $\vdash @_a\Diamond_i b \to @_b\varphi$ e $b \neq a$ não ocorre em φ, então $\vdash @_a\Box_i\varphi$

(Sub) Se $\vdash \varphi$, então $\vdash \varphi^\sigma$, onde σ substitui uniformemente nominais por nominais e símbolos proposicionais por fórmulas

Figura 6.3: Sistema axiomático $\mathbb{A}_{\mathbf{K}(n,@)}$ para a lógica $\mathbf{K}(n, @)$

Teorema 6.52. *Seja $\mathcal{C}$ a classe de estruturas definida como $\mathcal{C} = \{\mathcal{F} \in Fr_{HFor_n} : \mathcal{F} \Vdash \phi$ para todo $\phi \in P\}$, onde P é um conjunto $\mathbb{A}_{\mathbf{K}(n,@)}$-consistente de fórmulas puras. Então, o sistema axiomático $\mathbb{P} = \mathbb{A}_{\mathbf{K}(n,@)} + P$ é correto e fortemente completo para a lógica $Log(HFor_n, \mathcal{C})$.*

O teorema acima proporciona correção e completude forte automaticamente para uma grande classe de lógicas híbridas multimodais. Então, se temos um conjunto finito P de fórmulas puras que definam a classe de estruturas que são produto, então tudo que precisamos fazer é adicionar as fórmulas deste conjunto como axiomas e obtemos completude para $\mathbf{K}(@)^n$. A partir disto, completude para outras classes mais restritas de estruturas que são produto também podem ser obtidas, desde que estas classes possam ser descritas por fórmulas puras.

O teorema 6.27 descreve uma condição necessária e suficiente para um grafo enumerável e conexo ser um produto e os teoremas 6.32, 6.36, 6.37 e 6.38 mostram que esta condição pode ser expressa por fórmulas puras. Entretanto, não podemos apenas colocar estas fórmulas no teorema 6.52 e obter a completude que queremos, pois existem duas questões que precisam ser abordadas primeiro.

Começamos pela questão mais óbvia. Existem duas hipóteses que precisam ser satisfeitas antes que possamos aplicar o teorema 6.27: o grafo precisa ser conexo e o grafo precisa ser enumerável. Modelos canônicas são enumeráveis, conforme discutido anteriormente, então esta hipótese não é um problema. Entretanto, quando trabalhamos com uma linguagem híbrida, os modelos canônicos não são garantidamente conexos. Então, precisamos nos restringir explicitamente a classe de estruturas conexas e a melhor maneira de fazer isto é encontrar uma fórmula pura que descreva conectividade.

Teorema 6.53. *Um grafo $G = G_1 \times \cdots \times G_n$ é conexo se e somente se G_i é conexo para todo $1 \leq i \leq n$.*

Teorema 6.54. *Um grafo $G = (V, E)$ é conexo se e somente se $G \Vdash a \vee \blacklozenge a$.*

Agora, discutimos a segunda questão que nos impede de aplicar o teorema 6.52. Para isto, precisamos apresentar a noção de *compacidade*.

Definição 6.55 ([42])**.** *Seja $L = Log(F, \mathcal{C})$ uma lógica modal definida semanticamente. Dizemos que L é* compacta *se um conjunto Σ de fórmulas é $\mathcal{C}$-satisfazível se e somente se todo subconjunto finito $\Sigma_0 \subseteq \Sigma$ é $\mathcal{C}$-satisfazível.*

Precisamos adicionar as modalidades $\blacklozenge_i$ e $\Diamond_i^{-1}$ (uma vez que ela é utilizada na definição de $\blacklozenge_i$) à linguagem, uma vez que as fórmulas que descrevem intransitividade e conectividade faz uso delas. Entretanto, quando adicionamos as modalidades $\blacklozenge_i$ à linguagem, as lógicas modais perdem compacidade. Por exemplo, considere o conjunto

$$\Sigma = \{\blacklozenge p, \neg\Diamond p, \neg\Diamond^{-1} p, \neg\Diamond\Diamond p, \neg\Diamond\Diamond^{-1} p, \neg\Diamond^{-1}\Diamond p, \neg\Diamond^{-1}\Diamond^{-1} p, \ldots\}.$$

Todo subconjunto finito $\Sigma_0 \subseteq \Sigma$ é $\mathcal{C}$-satisfazível, onde $\mathcal{C}$ neste caso é a classe de todas as estruturas. Mas Σ não é satisfazível em nenhum vértice de nenhum modelo de nenhuma estrutura.

A prova de completude utilizando modelos canônicos proporciona completude forte como seu resultado (veja os teoremas 6.51 e 6.52, por exemplo). O seguinte teorema mostra que se temos um sistema axiomático correto e fortemente completo para uma lógica $L = \mathrm{Log}(F, \mathcal{C})$, então L é compacta.

Teorema 6.56. *Sejam $L = Log(F, \mathcal{C})$ uma lógica modal definida semanticamente e $\mathbb{A}$ um sistema axiomático. Se $\mathbb{A}$ é correto e fortemente completo para L, então L é compacta.*

Então, se uma lógica L não é compacta, como o resultado final das provas de completude utilizando modelos canônicos é completude forte, tal prova vai falhar para L [10]. Quando a lógica não é compacta, às vezes ainda é possível construir uma prova de completude (mas não de completude forte) utilizando *modelos canônicos finitos* que são construídos a partir de subconjuntos maximais consistentes de um conjunto finito de fórmulas. Entretanto, para uma prova de completude utilizando modelos canônicos finitos ser bem sucedida, a lógica precisa possuir a *propriedade do modelo finito*, isto é, toda fórmula satisfazível precisa ser satisfeita em um vértice de um modelo de uma estrutura *finita*. No nosso caso, como também precisamos que o modelo canônico seja um produto, precisaríamos na verdade que a lógica possuísse a *propriedade do modelo produto finito*.

Definição 6.57. *Um produto de lógicas modais possui a* propriedade do modelo produto finito *se toda fórmula satisfazível é satisfeita em um vértice de um modelo de uma estrutura* finita *que é um produto.*

Como o teorema a seguir, de [41], mostra, muitos produtos de lógicas modais de dimensão maior do que dois não possuem a propriedade do modelo produto finito.

Teorema 6.58 ([41])**.** *Seja $n \geq 3$ e seja $L \subseteq MFor_n$ qualquer lógica modal tal que $\mathbf{K}^n \subseteq L \subseteq \mathbf{S5}^n$. Então, L não possui a propriedade do modelo produto finito.*

Como $MFor_n \subset HFor_n$, o corolário abaixo é imediato.

Corolário 6.59. *Seja $n \geq 3$ e seja $L \subseteq HFor_n$ qualquer lógica modal tal que $\mathbf{K}(@)^n \subseteq L \subseteq \mathbf{S5}(@)^n$. Então, L não possui a propriedade do modelo produto finito.*

Então, precisamos encontrar uma maneira de expressar intransitividade e conectividade sem utilizar as modalidades $\blacklozenge_i$, de forma que tenhamos a compacidade de volta e possamos usar o teorema 6.52 para derivar as provas de completude. Vamos restringir nossa atenção à classe das estruturas transitivas e simétricas. Nesta classe de estruturas, utilizando simetria, temos que $\Diamond_i^{-1}\phi \equiv \Diamond_i\phi$, o que também implica

(utilizando transitividade) que $\blacklozenge_i \phi \equiv \Diamond_i \phi$. Além disso, a propriedade de Church-Rosser e a propriedade de Church-Rosser reversa são equivalentes. Logo, na classe de estruturas transitivas e simétricas, podemos caracterizar a classe de produtos conexos com as fórmulas puras na figura 6.4.

(Con$_i$) $a \vee \Diamond_i a$

(Com$_{ij}$) $\Diamond_j \Diamond_i a \leftrightarrow \Diamond_i \Diamond_j a$

(Chr$_{ij}$) $\Diamond_j a \rightarrow \Box_i \Diamond_j \Diamond_i a$

(Int$_{ij}$) $(a \wedge \neg b \wedge \Diamond_j (b \wedge \neg c \wedge \Diamond_i c)) \rightarrow \neg c$

Figura 6.4: Fórmulas puras que caracterizam a classe de produtos conexos

Agora, para finalmente obter o resultado que estamos buscando, tudo o que falta é caracterizarmos a classe das estruturas transitivas e simétricas utilizando fórmulas puras. Em [10], as fórmulas necessárias são apresentadas. Elas são mostradas na figura 6.5.

($\mathbf{4}_i$) $\Diamond_i \Diamond_i a \rightarrow \Diamond_i a$ (transitividade)

($\mathbf{B}_i$) $a \rightarrow \Box_i \Diamond_i a$ (simetria)

Figura 6.5: Fórmulas puras que caracterizam a classe de estruturas transitivas e simétricas

Teorema 6.60. *Seja P_{B4} o conjunto de fórmulas puras na figura 6.5 e seja $\mathbb{A}_{\mathbf{B4}(n,@)}$ o sistema axiomático $\mathbb{A}_{\mathbf{B4}(n,@)} = \mathbb{A}_{\mathbf{K}(n,@)} + P_{B4}$. O sistema axiomático $\mathbb{A}_{\mathbf{B4}(n,@)}$ é correto e completo para a lógica $\mathbf{B4}(n, @)$.*

Demonstração. Isto segue diretamente do teorema 6.52. □

Definição 6.61. *Seja $L = Log(F, \mathcal{C})$ uma lógica modal definida semanticamente. Então, denotamos por $\mathbf{C}L$ a lógica definida pela classe de estruturas conexas em $\mathcal{C}$.*

Teorema 6.62. *Seja P_{Prod} o conjunto de fórmulas puras na figura 6.4 e seja $\mathbb{P}$ o sistema axiomático $\mathbb{P} = \mathbb{A}_{\mathbf{B4}(n,@)} + P_{Prod}$. O sistema axiomático $\mathbb{P}$ é correto e completo para a lógica $\mathbf{CB4}(@)^n$.*

Demonstração. Isto segue diretamente do teorema 6.52. □

Teorema 6.63. *Seja* $L = L_1 \times \cdots \times L_n$ *um produto de lógicas modais. Se* $\mathbf{CB4}(@) \subseteq L_i$*, para todo* $1 \leq i \leq n$*, e* $\mathbb{A}_i = \mathbb{A}_{\mathbf{B4}(n,@)} + P_i$*, onde* P_i *é um conjunto finito de fórmulas puras, é um sistema axiomático correto e completo para* L_i*, então* $\mathbb{A} = \mathbb{A}_{\mathbf{B4}(n,@)} + \sum_i P_i + P_{Prod}$ *é um sistema axiomático correto e completo para* L*.*

Demonstração. Isto segue diretamente do teorema 6.52. □

Como exemplos de propriedades que podem ser expressas por fórmulas puras, podemos mencionar reflexividade, irreflexividade, densidade, determinismo, universalidade e tricotomia. Muitas destas propriedades não podem ser expressas por fórmulas na linguagem modal básica. Então, o teorema 6.63 proporciona sistemas axiomáticos corretos e completos para uma grande família de produtos de lógicas modais de dimensões arbitrárias, enquanto a maior parte dos resultados apresentados na literatura até o momento lidam apenas com produtos bidimensionais.

Como um comentário final, o que podemos dizer a respeito de produtos de lógicas modais na linguagem modal básica, dados os sistemas axiomáticos que construímos para produtos de lógicas modais na linguagem híbrida? Primeiramente, como conectividade não pode ser expressa na linguagem modal básica (conforme o capítulo anterior, o apêndice A e [29]), é simples ver que $L = \mathbf{C}L$, se L é uma lógica modal na linguagem modal básica. Além disso, como $MFor_n \subset HFor_n$, $L \subset L(@)$. Então, $L_1 \times \cdots \times L_n \subset \mathbf{C}L_1(@) \times \cdots \times \mathbf{C}L_n(@)$, o que significa que se temos um sistema axiomático $\mathbb{A}$ correto e completo para $\mathbf{C}L_1(@) \times \cdots \times \mathbf{C}L_n(@)$, então todas as fórmulas $\phi \in L_1 \times \cdots \times L_n$ são também teoremas de $\mathbb{A}$ ($\vdash_{\mathbb{A}} \phi$). Como exemplo, podemos obter um sistema axiomático correto e completo para $\mathbf{CS5}(@)^n$ utilizando o teorema 6.63. Então, apesar de $\mathbf{S5}^n$ não ser finitamente axiomatizável na linguagem modal básica, todas as suas fórmulas são dedutíveis a partir da axiomatização de $\mathbf{CS5}(@)^n$. De fato, estas fórmulas são exatamente o subconjunto de $\mathbf{CS5}(@)^n$ que contém apenas as fórmulas sem nominais e operadores de satisfação.

Capítulo 7

Lógicas Modais Dinâmicas e Sistemas Concorrentes

"Bart, com US$ 10.000 nós seremos milionários! Nós poderíamos comprar todo tipo de coisas úteis como... amor!" - Homer Simpson

Neste capítulo, consideramos extensões da Lógica Dinâmica Proposicional (PDL) com operadores para a descrição de comportamentos concorrentes. Nosso objetivo é construir lógicas dinâmicas que sejam apropriadas para a descrição e verificação de propriedades de sistemas comunicantes concorrentes, de maneira análoga ao uso de PDL para o caso sequencial.

O conteúdo deste capítulo tem como base os artigos [43] (reproduzido no apêndice C), [44] e [45] (reproduzido no apêndice D). As provas de todos os resultados apresentados neste capítulo podem ser consultadas nos apêndices C e D.

7.1 Introdução

Lógica Dinâmica Proposicional (PDL) [46] possui um papel importante na especificação formal e na verificação de programas e sistemas sequenciais. PDL é uma lógica multimodal com uma modalidade $\langle P \rangle$ para cada programa P. A lógica possui um conjunto de programas básicos e um conjunto de operadores (composição sequencial, iteração e escolha não determinística) que podem ser utilizados para construir programas mais complexos a partir de programas mais simples. Uma semântica de Kripke pode ser fornecida, com uma estrutura $\mathcal{F} = (W, R_P)$, onde W é um conjunto

não vazio de possíveis estados de programas e, para cada programa P, R_P é uma relação binária em W tal que $(s,t) \in R_P$ se e somente se existe uma computação de P começando em s e terminando em t.

O Cálculo de Sistemas Comunicantes (CCS) é uma álgebra de processos bem conhecida, proposta por Robin Milner [21], para a especificação de sistemas comunicantes concorrentes. Ele modela a concorrência e a interação entre processos através de atos individuais de comunicação. Um par de processos podem se comunicar através de um canal em comum e cada ato de comunicação consiste simplesmente de um sinal sendo enviado em uma ponta do canal e imediatamente sendo recebido na outra. Uma especificação CCS é uma descrição do comportamento esperado de um sistema, baseada nos eventos de comunicação que podem ocorrer. Para uma introdução ampla a CCS, [21] pode ser consultado.

O π-Cálculo é uma álgebra de processos bem conhecida, proposta por Milner, Parrow e Walker [25]. Ele é uma extensão do CCS de Milner [21] que é capaz de descrever não apenas não determinismo e concorrência, mas também *mobilidade* de processos. No π-Cálculo, cada ato de comunicação consiste em um nome sendo enviado em uma ponta de um canal e imediatamente sendo recebido na outra. Para uma introdução ampla ao π-Cálculo, [26] pode ser consultado.

Assim como PDL, CCS e o π- Cálculo possuem um conjunto de operadores (prefixo de ação, composição paralela, escolha não determinística e restrição sobre atos de comunicação) que são usados para indutivamente construir especificações de processos a partir de um conjunto de ações básicas.

Neste capítulo, apresentamos lógicas dinâmicas em que os programas são descritos em linguagens baseadas primeiramente em CCS (e XCCS) e, posteriormente, no π-Cálculo sem replicação.

Existem, na literatura, algumas lógicas que fazem uso de CCS ou do π-Cálculo. Entretanto, elas utilizam estas álgebras de processos como uma linguagem para a descrição de estruturas e modelos, enquanto usam lógicas modais tradicionais para a descrição das propriedades destas estruturas e modelos (veja, por exemplo, [47] e [48]). As lógicas que desenvolvemos neste capítulo utilizam CCS e o π-Cálculo de uma maneira distinta. Seus operadores e construções são *adicionados* a uma linguagem modal básica de forma a criar lógicas dinâmicas em que seja simples descrever

e verificar propriedades de programas e sistemas comunicantes, concorrentes e não determinísticos, de uma maneira similar a que PDL é usada no caso sequencial.

Logo, devemos enfatizar que a contribuição do trabalho que desenvolvemos neste capítulo está relacionada ao campo das lógicas dinâmicas e não ao campo das álgebras de processos. Das álgebras de processos, nós apenas tomamos um conjunto de operadores que são apropriados para a descrição de comunicação e concorrência. Nós utilizamos estes operadores porque eles possuem uma teoria bem estabelecida e nós podemos utilizar muitos de seus conceitos e resultados para ajudar-nos a construir nossas lógicas.

Nossas lógicas estão relacionadas a outras lógicas dinâmicas para classes de programas com algum tipo de concorrência, como PDL Concorrente (CPDL) [6], CPDL com canais [7], a lógica desenvolvida na tese [8] e PDL com intercalamento[1] [9], mas possuem algumas vantagens sobre elas. CPDL e PDL com intercalamento lidam com concorrência, mas não oferecem a possibilidade de comunicação entre os componentes de um sistema concorrente. CPDL com canais [7] é capaz de descrever propriedades interessantes de sistemas comunicantes concorrentes, mas ela não possui uma semântica de Kripke simples (de fato, "uma definição formal da semântica de CPDL com canais é consideravelmente complicada" [7]) e seu problema de satisfazibilidade é indecidível (Π^1_1-difícil), o que também implica que ela não possui uma axiomatização completa. A lógica desenvolvida em [8] força uma distinção semântica entre estados *finais* e estados *não finais*, o que também torna sua semântica e sua axiomatização razoavelmente complexas. Por outro lado, somos capazes de construir uma semântica de Kripke simples para nossas lógicas, baseada na ideia de execuções finitas possíveis de processos, construir axiomatizações completas para elas e mostrar que elas possuem a propriedade do modelo finito.

Escolhemos basear nossas lógicas nos mecanismos de comunicação e concorrência de CCS e sua extensão, o π-Cálculo, pela seguinte razão. Utilizando-se uma linguagem pequena como CCS, onde apenas as construções mais básicas estão presentes, podemos estudar em detalhes quais são os problemas que podem surgir quando tentamos utilizar seus operadores para construir uma lógica dinâmica e quais operadores e construções devem ser adicionados ou removidos para resolver estes problemas.

[1] interleaving, em inglês

O restante deste capítulo é organizado da seguinte maneira. Nossa primeira lógica (sCCS-PDL), em conjunto com um sistema axiomático completo, é apresentada na seção 7.2. Nesta lógica, não utilizamos constantes nem o operador de restrição na linguagem dos programas. Na seção 7.3, apresentamos nossa segunda lógica (CCS-PDL), em que permitimos a presença de constantes nos programas. Também apresentamos uma axiomatização completa para esta lógica. A terceira lógica (XCCS-PDL), em conjunto com uma axiomatização completa para ela, é apresentada na seção 7.4. Nesta lógica, os programas são baseados em XCCS, o que nos permite resolver alguns problemas que aparecem nas lógicas anteriores. Finalmente, nossa última lógica (πDL), em que os programas são baseados no π-Cálculo, em conjunto com um sistema axiomático completo, é apresentada na seção 7.5.

7.2 sCCS-PDL

Nesta seção, apresentamos a nossa primeira lógica dinâmica para sistemas concorrentes. Nesta lógica, os programas são construídos a partir da linguagem de CCS sem as contantes e o operador de restrição. Chamamos esta lógica de CSS-PDL reduzida ou sCCS-PDL. Nosso objetivo aqui é introduzir uma lógica simples e discutir algumas das questões a respeito dos axiomas e da interpretação relacional das fórmulas.

7.2.1 Linguagem e Semântica

Nesta seção, apresentamos a sintaxe e a semântica de sCCS-PDL.

Definição 7.1. *A linguagem de sCCS-PDL consiste de um conjunto enumerável Φ de símbolos proposicionais, um conjunto enumerável $\mathcal{N}$ de nomes, a ação silenciosa τ, os operadores booleanos $\neg$ e $\wedge$, os operadores $.$, $+$ e $|$ e uma modalidade $\langle P\rangle$ para cada programa P. As fórmulas são definidas da seguinte maneira:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle P\rangle\varphi,$$

com

$$P ::= \alpha \mid \alpha.P \mid P_1 + P_2 \mid P_1|P_2,$$

onde $p \in \Phi$ e $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$.

Definição 7.2. *Uma estrutura para sCCS-PDL é uma tupla* $\mathcal{F} = (W, \{R_\alpha\})$ *onde*

- W *é um conjunto não vazio de estados;*
- R_α *é uma relação binária para cada ação básica* $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$.

Definição 7.3. *Um modelo para sCCS-PDL é um par* $\mathcal{M} = (\mathcal{F}, \mathbf{V})$*, onde* $\mathcal{F}$ *é uma estrutura para sCCS-PDL e* $\mathbf{V}$ *é uma função de valoração* $\mathbf{V} : \Phi \mapsto 2^W$.

Definimos agora a noção semântica de satisfação para sCCS-PDL da seguinte maneira:

Definição 7.4. *Seja* $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ *um modelo. A noção de* satisfação *de uma fórmula* φ *em um modelo* $\mathcal{M}$ *em um estado* w*, notação* $\mathcal{M}, w \Vdash \varphi$*, pode ser indutivamente definida da seguinte maneira:*

- $\mathcal{M}, w \Vdash p$ *sse* $w \in \mathbf{V}(p)$*;*
- $\mathcal{M}, w \Vdash \top$ *sempre;*
- $\mathcal{M}, w \Vdash \neg\varphi$ *sse* $\mathcal{M}, w \nVdash \varphi$*;*
- $\mathcal{M}, w \Vdash \varphi_1 \wedge \varphi_2$ *sse* $\mathcal{M}, w \Vdash \varphi_1$ *e* $\mathcal{M}, w \Vdash \varphi_2$*;*
- $\mathcal{M}, w \Vdash \langle P \rangle \varphi$ *sse existe um caminho finito* $(v_0, v_1, \ldots, v_n)$*,* $n \geq 1$*, tal que* $v_0 = w$*,* $\mathcal{M}, v_n \Vdash \varphi$ *e existe* $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$ *de comprimento* n *tal que* $(v_{i-1}, v_i) \in R_\beta$ *se e somente se* $(\overrightarrow{\alpha})_i = \beta$*, para todo* $1 \leq i \leq n$*. Dizemos que tal sequência* $\overrightarrow{\alpha}$ casa com *o caminho* $(v_0, \ldots, v_n)$.

Como foi mencionado no capítulo 4, existem duas possíveis semânticas para a ação τ em CCS: ela pode ser considerada como sendo observável ou como sendo invisível. Em nossas lógicas, adotamos a primeira delas, uma vez que somos capazes de representar a segunda semântica como um caso particular da primeira. De fato, para fazer isto, a única coisa que é necessária é forçar, nas estruturas sendo consideradas, que R_τ seja a relação $R_\tau = \{(w, w) : w \in W\}$.

Teorema 7.5. $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$ *se e somente se* $\Vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$.

Corolário 7.6. *Se* $P \sim Q$*, então* $\Vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$.

Demonstração. Isto segue diretamente dos teoremas 4.13 e 7.5. □

Apresentamos agora algumas igualdades entre conjuntos de execuções finitas possíveis que são úteis para a prova de correção de nossa axiomatização e para mostrar porque o processo nulo $\mathbf{0}$ é problemático.

Teorema 7.7. *As seguintes igualdades de conjuntos são verdadeiras:*

1. $\overrightarrow{\mathcal{R}_f}(\alpha) = \{\alpha\}$;
2. $\overrightarrow{\mathcal{R}_f}(\alpha.P) = \overrightarrow{\mathcal{R}_f}(\alpha) \circ \overrightarrow{\mathcal{R}_f}(P)$;
3. $\overrightarrow{\mathcal{R}_f}(P_1 + P_2) = \overrightarrow{\mathcal{R}_f}(P_1) \cup \overrightarrow{\mathcal{R}_f}(P_2)$.

Demonstração. A prova segue diretamente da tabela 4.1. □

Teorema 7.8. *As seguintes fórmulas são válidas:*

1. $\langle \alpha.P \rangle p \leftrightarrow \langle \alpha \rangle \langle P \rangle p$
2. $\langle P_1 + P_2 \rangle p \leftrightarrow \langle P_1 \rangle p \vee \langle P_2 \rangle p$

Agora é possível ver porque, como foi dito no capítulo 4, o uso do processo nulo $\mathbf{0}$ seria inconveniente em nossas lógicas. O problema que surgiria vem do fato de que, como descrito em [21], em uma especificação da forma $\alpha.\mathbf{0}$, $\mathbf{0}$ está denotando um processo que terminou de maneira bem sucedida, enquanto em uma especificação da forma $P + \mathbf{0}$, $\mathbf{0}$ está denotando um processo em *deadlock*. Este papel duplo não pode ser mantido em nossas lógicas sem sacrificar uma propriedade muito desejada em lógicas dinâmicas: a semântica composicional, ilustrada no teorema 7.8.

A semântica composicional é uma consequência direta das igualdades entre conjuntos no teorema 7.7. Mas quando tentamos mantê-las na presença de $\mathbf{0}$, alguns problemas surgem. $\overrightarrow{\mathcal{R}_f}(\alpha.\mathbf{0}) = \{\alpha\}$, uma vez que $\mathbf{0}$ denota terminação bem sucedida neste caso (se $\mathbf{0}$ denotasse *deadlock*, então $\overrightarrow{\mathcal{R}_f}(\alpha.\mathbf{0})$ seria $\emptyset$), e $\overrightarrow{\mathcal{R}_f}(P + \mathbf{0}) = \overrightarrow{\mathcal{R}_f}(P)$, uma vez que $\mathbf{0}$ denota um *deadlock* neste caso (se $\mathbf{0}$ denotasse terminação bem sucedida, então $\overrightarrow{\mathcal{R}_f}(P + \mathbf{0})$ seria $\overrightarrow{\mathcal{R}_f}(P) \cup \{\overrightarrow{\varepsilon}\}$). Para manter a segunda igualdade no teorema 7.7, precisamos ter $\{\alpha\} = \overrightarrow{\mathcal{R}_f}(\alpha.\mathbf{0}) = \overrightarrow{\mathcal{R}_f}(\alpha) \circ \overrightarrow{\mathcal{R}_f}(\mathbf{0})$, o que implica que $\overrightarrow{\mathcal{R}_f}(\mathbf{0}) = \{\overrightarrow{\varepsilon}\}$ (*). Por outro lado, para manter a terceira igualdade, precisamos ter $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(P + \mathbf{0}) = \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(\mathbf{0})$, o que implica que $\overrightarrow{\mathcal{R}_f}(\mathbf{0}) = \emptyset$ (**).

No formalismo lógico, pelo teorema 7.8, (*) implicaria que $\langle \mathbf{0} \rangle \phi$ é semanticamente equivalente a ϕ, enquanto (**) implicaria que $\langle \mathbf{0} \rangle \phi$ é semanticamente equivalente a

$\perp$. O ponto crucial nesta situação é que precisaríamos ou abandonar ao menos uma das igualdades no teorema 7.7, substituindo-a por um par de equações, uma para o caso em que $P \neq \mathbf{0}$ e outra para o caso em que $P = \mathbf{0}$, ou de alguma forma alterar a semântica de modo que o significado de uma subfórmula da forma $\langle \mathbf{0} \rangle \phi$ dependa do contexto em que ela está inserida, sendo algumas vezes equivalente a ϕ e algumas vezes a $\perp$. Ambas as "soluções" iriam comprometer seriamente a composicionalidade da semântica.

Nós resolvemos a questão do processo nulo em nossa terceira lógica, sem introduzir nenhum dos problemas acima. Lá, ao invés de CCS, nós utilizamos XCCS como linguagem para os programas da lógica.

7.2.2 Sistema Axiomático

Consideramos o seguinte conjunto de axiomas e regras, onde p e q são símbolos proposicionais e φ e ψ são fórmulas.

(LP) Tautologias Proposicionais

(K) $\vdash [P](p \to q) \to ([P]p \to [P]q)$

(Du) $\vdash [P]p \leftrightarrow \neg\langle P \rangle \neg p$

(Pr) $\vdash \langle \alpha.P \rangle p \leftrightarrow \langle \alpha \rangle \langle P \rangle p$

(NC) $\vdash \langle P_1 + P_2 \rangle p \leftrightarrow \langle P_1 \rangle p \vee \langle P_2 \rangle p$

(PC) Se a Lei de Expansão pode ser aplicada a P, então $\vdash \langle P \rangle p \leftrightarrow \langle Exp(P) \rangle p$

(Sub) Se $\vdash \varphi$, então $\vdash \varphi^\sigma$, onde σ substitui uniformemente símbolos proposicionais por fórmulas arbitrárias.

(MP) Se $\vdash \varphi$ e $\vdash \varphi \to \psi$, então $\vdash \psi$.

(Gen) Se $\vdash \varphi$, então $\vdash [P]\varphi$.

É importante notar que os teoremas $\vdash \langle P_1 + P_2 \rangle p \leftrightarrow \langle P_2 + P_1 \rangle p$ e $\vdash \langle P_1 | P_2 \rangle p \leftrightarrow \langle P_2 | P_1 \rangle p$, que enunciam a comutatividade dos operadores $+$ e $|$, são deriváveis a partir do sistema axiomático acima.

Teorema 7.9 (Correção e Completude)**.** *O sistema axiomático acima é correto e completo para sCCS-PDL.*

7.3 CCS-PDL

A lógica apresentada nesta seção utiliza os mesmos operadores de CCS utilizados na seção anterior, com o acréscimo de constantes. A lógica desta seção é chamada de CCS-PDL.

7.3.1 Linguagem e Semântica

Nesta seção, apresentamos a sintaxe e semântica de CCS-PDL.

Definição 7.10. *A linguagem de CCS-PDL consiste de um conjunto enumerável* Φ *de símbolos proposicionais, um conjunto enumerável* $\mathcal{N}$ *de nomes, a ação silenciosa* τ*, os operadores booleanos* $\neg$ *e* $\wedge$*, os operadores* $.$*,* $+$ *e* $|$*, um conjunto enumerável* $\mathcal{C}$ *de constantes, tal que cada elemento de* $\mathcal{C}$ *possui uma única equação definidora correspondente, e uma modalidade* $\langle P \rangle$ *para cada programa* P*. As fórmulas são definidas da seguinte maneira:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle P \rangle \varphi,$$

com

$$P ::= \alpha \mid \alpha.P \mid \alpha.A \mid P_1 + P_2 \mid P_1 | P_2,$$

onde $p \in \Phi$*,* $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ *e* $A \in \mathcal{C}$*.*

A presença de constantes na linguagem nos permite escrever especificações que são capazes de iteração como $P = \alpha.A$, com $A \stackrel{def}{=} \alpha.A + \tau$. Entretanto, constantes possuem um poder muito maior do que o de apenas expressar comportamentos iterativos. Com constantes, somos capazes de escrever especificações replicantes, como $P = ((\tau.A) + \tau)|Q$, com $A \stackrel{def}{=} ((\tau.A) + \tau)|Q$. Após a execução de n ações τ, P é capaz de se comportar como n processos Q em paralelo, para qualquer $n \in \mathbb{N}$.

O exemplo acima é um exemplo muito simples de replicação e é fácil perceber que as coisas podem se tornar bastante complexas se começarmos a aninhar processos replicantes.

De forma a manter a lógica simples, isto é, manter a semântica de Kripke simples, a propriedade do modelo finito e uma axiomatização simples e completa, nós restringimos o uso das constantes em CCS-PDL de maneira a evitar processos replicantes (em [36], Dam força uma restrição sintática similar, também com o objetivo de evitar o crescimento ilimitado de processos). A questão de se é possível manter estas propriedades desejáveis da lógica na presença de replicação permanece um problema em aberto e nós o adiamos para um trabalho futuro, conforme é explicado no capítulo 8.

Definição 7.11. *Seja P um processo e $\{A_1, \ldots, A_n\}$ as constantes que ocorrem em P. Definimos $Cons(P)$ como o menor conjunto de constantes tal que $Cons(P) \supseteq \{A_1, \ldots, A_n\}$ e, para toda constante $A_i \in Cons(P)$, se A_k ocorre em P_{A_i}, então $A_k \in Cons(P)$.*

Hipótese 7.12. *Fazemos as seguintes restrições aos programas em CCS-PDL:*

1. *$Cons(P)$ precisa ser um conjunto finito para todo programa P;*
2. *Apenas permitimos equações definidoras que se encaixem em um dos seguintes modelos:*
   - *$A \stackrel{def}{=} P_A$, onde $A \notin Cons(P_A)$, chamadas* equações não recursivas*;*
   - *$A \stackrel{def}{=} \overrightarrow{\alpha}_1.A + \ldots + \overrightarrow{\alpha}_n.A + T_A$, onde $A \notin Cons(T_A)$, chamadas* equações recursivas.

As igualdades de conjuntos do teorema 7.7 permanecem válidas, em conjunto com a igualdade

$$\overrightarrow{\mathcal{R}_f}(\alpha.A) = \overrightarrow{\mathcal{R}_f}(\alpha) \circ \overrightarrow{\mathcal{R}_f}(P_A), \tag{7.1}$$

que também segue da tabela 4.1.

Entretanto, devido à possibilidade de comportamentos iterativos, algumas igualdades de conjuntos podem se apresentar como equações recursivas. Neste caso, é possível obter equações não recursivas equivalentes. Primeiramente, as equações recursivas podem ser reescritas, utilizando-se as igualdades no teorema 7.7 e equação (7.1), como $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(P') \circ \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(Q)$, onde $\overrightarrow{\mathcal{R}_f}(Q)$ não depende de $\overrightarrow{\mathcal{R}_f}(P)$. Agora, como todas as sequencias em $\overrightarrow{\mathcal{R}_f}(P)$, $\overrightarrow{\mathcal{R}_f}(P')$ e $\overrightarrow{\mathcal{R}_f}(Q)$ são finitas e $\overrightarrow{\varepsilon} \notin \overrightarrow{\mathcal{R}_f}(P')$, podemos utilizar um resultado conhecido como Lema de Arden

[49], que diz que se X, A e B são conjuntos de sequencias finitas de caracteres e a sequencia vazia não pertence a A, então a equação $X = A \circ X \cup B$ possui como única solução $X = A^* \circ B$. Portanto, $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}^*(P') \circ \overrightarrow{\mathcal{R}_f}(Q)$.

Definição 7.13. *Dizemos que um processo P é um* nó *se $P = P_A$ para alguma constante A com uma equação definidora recursiva ou se $P = P_1 \mid P_2$ onde P_1 ou P_2 é um nó.*

Definição 7.14. *Chamamos uma sequencia não vazia de ações $\overrightarrow{\alpha}$ um* laço *de um processo P que é um nó se $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} P$. Dizemos que $\overrightarrow{\alpha}$ é um* laço próprio *se $\overrightarrow{\alpha}$ é um laço e não existe $\overrightarrow{\beta} \subset \overrightarrow{\alpha}$, com $\overrightarrow{\alpha} = \overrightarrow{\beta}.\overrightarrow{\lambda}$, tal que $\overrightarrow{\beta}$ e $\overrightarrow{\lambda}$ sejam laços de P. O conjunto de laços de P é denotado por $Lo(P)$ e o conjunto de laços próprios por $PLo(P)$.*

Teorema 7.15. *$\overrightarrow{\alpha} \in Lo(P)$ se e somente se $\overrightarrow{\alpha} = \overrightarrow{\alpha_1}.\cdots.\overrightarrow{\alpha_n}$, $n \geq 1$, onde $\overrightarrow{\alpha_i} \in PLo(P)$, para todo $i \in \{1, \ldots, n\}$.*

Demonstração. A prova segue diretamente da definição 7.14. □

Definição 7.16. *Chamamos uma sequência de ações $\overrightarrow{\alpha}$ uma* ruptura *de um processo P que é um nó se não existe $\overrightarrow{\beta}$ tal que $\overrightarrow{\alpha} \subset \overrightarrow{\beta}$ e $\overrightarrow{\beta}$ é um laço. Dizemos que $\overrightarrow{\alpha}$ é uma* ruptura própria *se $\overrightarrow{\alpha}$ é uma ruptura e não existe $\overrightarrow{\beta} \subset \overrightarrow{\alpha}$, com $\overrightarrow{\alpha} = \overrightarrow{\beta}.\overrightarrow{\lambda}$, tal que $\overrightarrow{\beta}$ é um laço e $\overrightarrow{\lambda}$ é uma ruptura. Finalmente, dizemos que $\overrightarrow{\alpha}$ é uma* ruptura própria mínima *se $\overrightarrow{\alpha}$ é uma ruptura própria e não existe $\overrightarrow{\beta} \subset \overrightarrow{\alpha}$ tal que $\overrightarrow{\beta}$ é uma ruptura própria. O conjunto de rupturas de P é denotado por $Br(P)$, o conjunto de rupturas próprias por $PBr(P)$ e o conjunto de rupturas próprias mínimas por $MPBr(P)$.*

Teorema 7.17. *$\overrightarrow{\alpha} \in PBr(P)$ se e somente se $\overrightarrow{\alpha} = \overrightarrow{\beta}.\overrightarrow{\lambda}$, onde $\overrightarrow{\beta} \in MPBr(P)$.*

Demonstração. A prova segue diretamente da definição 7.16. □

Utilizando os conceitos de laços e rupturas, podemos decompor um processo P que é um nó em duas partes: a *parte em laço*, denotada por L_P, e a *cauda*, denotada por T_P.

$$L_P = \sum\{\overrightarrow{\alpha} : \overrightarrow{\alpha} \in PLo(P)\}.$$

e

$$T_P = \sum\{\overrightarrow{\alpha}.P' : \overrightarrow{\alpha} \in MPBr(P) \text{ e } P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} P'\}$$

Teorema 7.18. *Se P é um nó, então $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}^*(L_P) \circ \overrightarrow{\mathcal{R}_f}(T_P)$.*

Também definimos o processo L'_P, que é capaz de iterar L_P.

$$L'_P = \sum\{\overrightarrow{\alpha}.Z_P : \overrightarrow{\alpha} \in PLo(P)\} + L_P,$$

onde Z_P é uma nova constante com equação definidora $Z_P \stackrel{def}{=} L'_P$.

As noções de estrutura, modelo e satisfação são definidas da mesma forma que nas definições 7.2, 7.3 e 7.4. Não é difícil ver que o teorema 7.5 e o corolário 7.6 permanecem válidos em CCS-PDL.

7.3.2 Sistema Axiomático

O sistema axiomático é similar ao apresentado na seção 7.2.2. Consideramos o seguinte conjunto de axiomas e regras, onde p, q e r são símbolos proposicionais e φ e ψ são fórmulas.

- Os axiomas **(LP)**, **(K)** e **(Du)** e as regras **(Sub)**, **(MP)** e **(Gen)**.
- Axiomas para processos que são nós:

  (Rec) $\vdash \langle P\rangle p \leftrightarrow \langle T_P\rangle p \vee \langle L_P\rangle\langle P\rangle p$

  (FP) $\vdash (r \rightarrow ([T_P]\neg p \wedge [L_P]r)) \wedge [L'_P](r \rightarrow ([T_P]\neg p \wedge [L_P]r)) \rightarrow (r \rightarrow [P]\neg p)$
- Axiomas para processos que não são nós:

  (sCCS) Os axiomas **(Pr)**, **(NC)** e a regra **(PC)**.

  (Cons) $\vdash \langle \alpha.A\rangle p \leftrightarrow \langle\alpha\rangle\langle P_A\rangle p$

Teorema 7.19 (Correção e Completude)**.** *O sistema axiomático acima é correto e completo para CCS-PDL.*

7.4 XCCS-PDL

Como foi mostrado na seção 7.2, o uso do processo nulo **0** de CCS em nossas lógicas iniciais iria trazer uma séria inconveniência para suas semânticas: suas composicionalidades seriam comprometidas. Este problema também afeta a possibilidade de incluirmos o operador de restrição nestas lógicas. Além disso, em

CCS-PDL, precisamos definir dois conjuntos distintos de axiomas, dependendo se o processo em consideração é ou não um nó.

Nesta seção, nosso objetivo é resolver estes dois problemas que ocorrem nas lógicas anteriores. Para isto, nesta seção desenvolvemos uma lógica em que os programas são baseados na linguagem de XCCS. Devido à definição refinada do processo nulo **0** em XCCS, podemos incluí-lo nesta lógica, assim como o operador de restrição. Além disso, um dos novos operadores de XCCS, o operador de iteração, nos permite lidar com todos os tipos de processos com apenas um conjunto de axiomas e descartar as constantes e sua teoria elaborada da linguagem.

7.4.1 Linguagem e Semântica

Nesta seção, apresentamos a sintaxe e a semântica de XCCS-PDL.

Definição 7.20. *A linguagem de XCCS-PDL consiste de um conjunto enumerável* Φ *de símbolos proposicionais, um conjunto enumerável* $\mathcal{N}$ *de nomes, a ação silenciosa* τ*, a ação de terminação* END*, os operadores booleanos* $\neg$ *e* $\wedge$*, os operadores* $.$*,* $;$*,* $+$*,* $|$*,* * *e* $\backslash$*, uma modalidade* $\langle\alpha\rangle$ *para cada* $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ *e uma modalidade* $\langle P\rangle$ *para cada programa* P*, incluindo os programas atômicos* **0** *e* END*. As fórmulas são definidas da seguinte maneira:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\alpha\rangle\varphi \mid \langle P\rangle\varphi,$$

com

$$P ::= \mathbf{0} \mid END \mid \alpha.P \mid P_1; P_2 \mid P_1 + P_2 \mid P_1|P_2 \mid P^* \mid P\backslash L,$$

onde $p \in \Phi$*,* $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ *e* $L \subseteq \mathcal{N}$*.*

Definição 7.21. *Uma estrutura para XCCS-PDL é uma tupla* $\mathcal{F} = (W, \{R_\alpha\}, R_{END})$ *onde*

- W *é um conjunto não vazio de estados;*
- R_α*, para cada* $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ *e* R_{END} *são as relações binárias básicas, onde* $R_{END} = \{(w, w) : w \in W\}$*.*

A noção de modelo é definida da mesma forma que na definição 7.3. Definimos a noção semântica de satisfação para XCCS-PDL da seguinte forma:

Definição 7.22. *Seja $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ um modelo. A noção de satisfação de uma fórmula φ em um modelo $\mathcal{M}$ em um estado w, notação $\mathcal{M}, w \Vdash \varphi$, pode ser indutivamente definida da seguinte maneira:*

- $\mathcal{M}, w \Vdash p$ *sse* $w \in \mathbf{V}(p)$;
- $\mathcal{M}, w \Vdash \top$ *sempre;*
- $\mathcal{M}, w \Vdash \neg\varphi$ *sse* $\mathcal{M}, w \nVdash \varphi$;
- $\mathcal{M}, w \Vdash \varphi_1 \wedge \varphi_2$ *sse* $\mathcal{M}, w \Vdash \varphi_1$ *e* $\mathcal{M}, w \Vdash \varphi_2$;
- $\mathcal{M}, w \Vdash \langle\alpha\rangle\varphi$ *sse existe* $w' \in W$ *tal que* $wR_\alpha w'$ *e* $\mathcal{M}, w' \Vdash \varphi$, *onde* $\alpha \in \mathcal{N} \cup \overline{N} \cup \{\tau\}$;
- $\mathcal{M}, w \Vdash \langle P\rangle\varphi$ *sse existe um caminho finito* $(v_0, v_1, \ldots, v_n)$, $n \geq 1$, *tal que* $v_0 = w$, $\mathcal{M}, v_n \Vdash \varphi$ *e existe* $\overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P)$ *de comprimento* n *tal que* $(v_{i-1}, v_i) \in R_\beta$ *se e somente se* $(\overrightarrow{\alpha})_i = \beta$, *para todo* $1 \leq i \leq n$. *Dizemos que tal sequencia* $\overrightarrow{\alpha}$ casa com *o caminho* $(v_0, \ldots, v_n)$.

Não é difícil ver que que o teorema 7.5 e o corolário 7.6 permanecem válidos em XCCS-PDL.

Teorema 7.23. *As seguintes igualdades de conjuntos são verdadeiras:*

1. $\overrightarrow{\mathcal{R}_f}(\mathbf{0}) = \emptyset$;
2. $\overrightarrow{\mathcal{R}_f}(END) = \{END\}$;
3. $\overrightarrow{\mathcal{R}_f}(\alpha.P) = \overrightarrow{\mathcal{R}_f}(\alpha) \circ \overrightarrow{\mathcal{R}_f}(P)$;
4. $\overrightarrow{\mathcal{R}_f}(P_1; P_2) = \overrightarrow{\mathcal{R}_f}(P_1) \circ \overrightarrow{\mathcal{R}_f}(P_2)$;
5. $\overrightarrow{\mathcal{R}_f}(P_1 + P_2) = \overrightarrow{\mathcal{R}_f}(P_1) \cup \overrightarrow{\mathcal{R}_f}(P_2)$;
6. $\overrightarrow{\mathcal{R}_f}(P^*) = \overrightarrow{\mathcal{R}_f}(P)^*$;
7. $\overrightarrow{\mathcal{R}_f}(P_1|P_2) = \bigcup\{\overrightarrow{\mathcal{R}_f}(\overrightarrow{\alpha}|\overrightarrow{\beta}) : \overrightarrow{\alpha} \in \overrightarrow{\mathcal{R}_f}(P_1)$ *e* $\overrightarrow{\beta} \in \overrightarrow{\mathcal{R}_f}(P_2)\}$;
8. *Se* $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$, *então* $\overrightarrow{\mathcal{R}_f}(P\backslash L) = \overrightarrow{\mathcal{R}_f}(Q\backslash L)$;
9. *Se* $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(A) \circ \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(B)$ *e* $END \notin \overrightarrow{\mathcal{R}_f}(A)$, *então* $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(A)^* \circ \overrightarrow{\mathcal{R}_f}(B)$.

Demonstração. A prova dos primeiros oito itens segue diretamente da tabela 4.2 e do teorema 4.13. O nono item é o Lema de Arden [49] aplicado no nosso contexto. □

7.4.2 Sistema Axiomático

Consideramos o seguinte conjunto de axiomas e regras, onde p e q são símbolos proposicionais e φ e ψ são fórmulas.

(sCCS) Os axiomas **(LP)**, **(K)**, **(Du)**, **(Pr)** e **(NC)** e as regras **(PC)**, **(Sub)**, **(MP)** e **(Gen)**

(0) $\vdash \neg\langle \mathbf{0} \rangle p$

(END) $\vdash \langle END \rangle p \leftrightarrow p$

(SC) $\vdash \langle P_1; P_2 \rangle p \leftrightarrow \langle P_1 \rangle \langle P_2 \rangle p$

(Rec) $\vdash \langle P^* \rangle p \leftrightarrow p \vee \langle P \rangle \langle P^* \rangle p$

(FP) $\vdash p \wedge [P^*](p \rightarrow [P]p) \rightarrow [P^*]p$

(PCSub) Se $\vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$, então $\vdash \langle P|R \rangle p \leftrightarrow \langle Q|R \rangle p$

(RSub) Se $\vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$, então $\vdash \langle P \backslash L \rangle p \leftrightarrow \langle Q \backslash L \rangle p$

(Ard) Se $\vdash \langle P \rangle p \leftrightarrow \langle A; P + B \rangle p$ e $A \overset{END}{\not\rightarrow} \surd$, então $\vdash \langle P \rangle p \leftrightarrow \langle A^*; B \rangle p$

(Con) Se $P \equiv_r Q$, então $\vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$

Definição 7.24. *Definimos a seguinte relação entre processos: $P \leftrightarrow Q$ se e somente se $\vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$.*

Teorema 7.25. *$\leftrightarrow$ é uma congruência.*

Apresentamos agora um teorema chave para a prova de completude da axiomatização acima.

Teorema 7.26. *Seja $P = P_1 \mid P_2$, onde P é irrestrito. Então $P \leftrightarrow \overline{P}$, onde $\overline{P}$ não possui nenhuma ocorrência do operador $|$.*

Demonstração. A prova deste teorema baseia-se na solução de um sistema de congruências. Ela é inspirada no método algébrico de Brzozowski para obter a expressão regular que descreve a linguagem aceita por um autômato finito [50]. Acreditamos que esta é uma aplicação interessante e original da ideia de Brzozowski e que este

método proporciona uma prova elegante para um resultado chave para a completude do sistema axiomático acima. A prova pode ser consultada no apêndice D (teorema D.36). □

Teorema 7.27 (Correção e Completude)**.** *O sistema axiomático acima é correto e completo para XCCS-PDL.*

7.5 πDL

Nesta seção, apresentamos nossa última lógica dinâmica proposicional para sistemas concorrentes. Nela, os programas são construídos em uma linguagem baseada no π-Cálculo sem replicação (definição 4.24). Esta lógica é chamada de πDL.

7.5.1 Linguagem e Semântica

Nesta seção, apresentamos a sintaxe e a semântica de πDL.

Definição 7.28. *A linguagem de πPDL consiste de um conjunto enumerável Φ de símbolos proposicionais, um conjunto enumerável $\mathcal{N}$ de nomes, a ação silenciosa τ, a ação de terminação END, os operadores booleanos $\neg$ e $\wedge$, os operadores $.$, $;$, $+$, $|$, * e ν, uma modalidade $\langle\alpha\rangle$ para cada $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$ e uma modalidade $\langle P\rangle$ para cada programa P, incluindo os programas atômicos $\mathbf{0}$ e END. As fórmulas são definidas da seguinte maneira:com*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\alpha\rangle\varphi \mid \langle P\rangle\varphi,$$

com

$$P ::= \mathbf{0} \mid END \mid \alpha.P \mid P_1; P_2 \mid P_1 + P_2 \mid P_1|P_2 \mid P^* \mid (\nu a)P,$$

e

$$\alpha ::= a(x) \mid \overline{a}\langle x\rangle \mid \overline{a}\langle\nu x\rangle \mid \tau,$$

onde $p \in \Phi$ e $a, x \in \mathcal{N}$.

As noções de estrutura e modelo são definidas da mesma forma que nas definições 7.21 e 7.3.

Definição 7.29. *Definimos o* núcleo *de uma ação α, denotado por $c(\alpha)$, da seguinte maneira: $c(a(x)) = a$, $c(\overline{a}\langle x\rangle) = c(\overline{a}\langle\nu x\rangle) = \overline{a}$ e $c(\alpha) = \alpha$, se $\alpha = \tau$ ou $\alpha = END$.*

A noção semântica de satisfação para πDL é muito semelhante à da definição 7.22. As únicas (e pequenas) diferenças ocorrem nos casos das modalidades, que mostramos abaixo:

- $\mathcal{M}, w \Vdash \langle\kappa\rangle\varphi$ sse existe $w' \in W$ tal que $wR_\kappa w'$ e $\mathcal{M}, w' \Vdash \varphi$, onde $\kappa = a$ ou $\kappa = \overline{a}$, para algum $a \in \mathcal{N}$, ou $\kappa = \tau$;

- $\mathcal{M}, w \Vdash \langle P\rangle\varphi$ sse existe um caminho finito $(v_0, v_1, \ldots, v_n)$, $n \geq 1$, tal que $v_0 = w$, $\mathcal{M}, v_n \Vdash \varphi$ e existe $\overrightarrow{\alpha}$ tal que $[\overrightarrow{\alpha}] \in \overrightarrow{\mathcal{R}_f}(P)$, o comprimento de $\overrightarrow{\alpha}$ é n e $(v_{i-1}, v_i) \in R_\kappa$ se e somente se $c((\overrightarrow{\alpha})_i) = \kappa$, para todo $1 \leq i \leq n$. Dizemos que tal $\overrightarrow{\alpha}$ *casa com* o caminho $(v_0, \ldots, v_n)$.

Não é difícil ver que o teorema 7.5 e o corolário 7.6 permanecem válidos em πDL.

Teorema 7.30. *As seguintes igualdades de conjuntos são verdadeiras:*

1. $\overrightarrow{\mathcal{R}_f}(\mathbf{0}) = \emptyset$;
2. $\overrightarrow{\mathcal{R}_f}(END) = \{[END]\}$;
3. $\overrightarrow{\mathcal{R}_f}(\alpha.P) = \{[\alpha.END]\} \circ \overrightarrow{\mathcal{R}_f}(P)$;
4. $\overrightarrow{\mathcal{R}_f}(P_1; P_2) = \overrightarrow{\mathcal{R}_f}(P_1) \circ \overrightarrow{\mathcal{R}_f}(P_2)$;
5. $\overrightarrow{\mathcal{R}_f}(P_1 + P_2) = \overrightarrow{\mathcal{R}_f}(P_1) \cup \overrightarrow{\mathcal{R}_f}(P_2)$;
6. $\overrightarrow{\mathcal{R}_f}(P^*) = (\overrightarrow{\mathcal{R}_f}(P))^*$;
7. $\overrightarrow{\mathcal{R}_f}(P_1|P_2) = \bigcup\{\overrightarrow{\mathcal{R}_f}(\overrightarrow{\alpha} \mid \overrightarrow{\beta}) : [\overrightarrow{\alpha}] \in \overrightarrow{\mathcal{R}_f}(P_1) \ e\ [\overrightarrow{\beta}] \in \overrightarrow{\mathcal{R}_f}(P_2)\}$;
8. *Se* $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$, *então* $\overrightarrow{\mathcal{R}_f}\ ((\nu x)P) = \overrightarrow{\mathcal{R}_f}((\nu x)Q)$;
9. *Se* $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(A) \circ \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(B)$ *e* $[END] \notin \overrightarrow{\mathcal{R}_f}(A)$, *então* $\overrightarrow{\mathcal{R}_f}(P) = (\overrightarrow{\mathcal{R}_f}(A))^* \circ \overrightarrow{\mathcal{R}_f}(B)$.

7.5.2 Interlúdio: Jogos Simultâneos

O operador de composição paralela (|) possui um papel duplo no π-Cálculo. Ele descreve tanto intercalamento quanto sincronização de processos. No paradigma descrito por van Benthem de *jogos como processos* ([51]), isto poderia ser utilizado para representar jogos simultâneos, onde cada jogador escolhe suas ações sem necessariamente ter conhecimento das ações realizadas pelo outro jogador. Consideramos um exemplo concreto. Seja $\mathcal{M}$ um modelo que representa os possíveis estados do jogo e w_i, $i = 1, 2$, símbolos proposicionais que denotam que o jogador i vence se o

jogo termina em um estado onde w_i é satisfeito. Sejam $\{a, b, c\}$ as possíveis ações para o jogador 1 e $\{d, e, f\}$ as possíveis ações para o jogador 2. Cada jogador precisa executar uma sequência de três ações, sem ter nenhuma informação de quais ações o outro jogador executou ou mesmo de quantas das três ações o outro jogador já executou. Isto significa que as duas sequencias são intercaladas em uma ordem arbitrária. Então, $\mathcal{M}, v \Vdash \langle a.b.b.END + a.b.c.END \mid d.d.d.END\rangle w_1$ significa que se o jogo começa em v, existe uma sequencia intercalada de a, b, b e d, d, d ou a, b, c e d, d, d que leva o jogador 1 a vencer. Por outro lado, $\mathcal{M}, v \Vdash [a.b.b.END \mid d.d.d.END]w_1$ significa que, se o jogo começa em v, não importa em que ordem as seis ações acontecem, se o jogador 1 jogar a, b, b e o jogador 2 jogar d, d, d, o jogador 1 vencerá garantidamente. Isto também pode ser generalizado para jogos com mais de dois jogadores.

O operador $\mid$ também pode descrever sincronização entre processos. Isto nos permite modelar jogos mais ricos, onde podemos ter "rodadas" de jogos simultâneos "cegos" separadas por alguma troca de informação entre os jogadores. Por exemplo, no jogo descrito acima, suponha agora que os dois jogadores selecionam suas ações a partir do mesmo conjunto $\{a, b, c\}$. Para diferenciar entre as sequencias de ações de cada jogador, cada sequencia começa com p_i, $i = 1, 2$. Além disso, cada jogador agora irá executar as três ações da seguinte maneira: um jogador executa duas ações, então informa ao outro jogador uma delas e executa a ação final. Podemos expressar propriedades deste jogo utilizando fórmulas da nossa lógica, de uma maneira análoga ao que fizemos no parágrafo acima. Por exemplo, $\mathcal{M}, v \Vdash [(\nu x)(\nu y)(p_1.b.b.\overline{x}\langle b\rangle.y(w).w.END \mid p_2.a.b.x(z).\overline{y}\langle a\rangle.b.END + p_2.a.b.x(z).\overline{y}\langle b\rangle.b.END)]w_1$ significa que se o jogo começa em v, não importa em que ordem as quatro ações iniciais e as duas ações finais ocorrem, se o jogador 2 começa com a, b e termina com b, então o jogador 1 pode sempre vencer jogando b, b e então finalizando com a ação informada pelo jogador 2.

Esta inter-relação entre intercalamento e sincronização oferecida pelo operador $\mid$ pode estão ser utilizada para descrever um grande grupo de jogos. Um artigo recente ([52]) também trabalha com esta ideia de que operadores de concorrência podem ser utilizados para modelar jogos simultâneos. Os autores utilizam CPDL [6] como uma ferramenta para a construção de uma lógica dinâmica concorrente para jogos.

Entretanto, como CPDL não admite comunicação, a lógica em [52] também possui esta limitação. Como a generalização de CPDL para CPDL com canais tem como desvantagens a perda de decidibilidade e a perda de uma axiomatização completa, nossa lógica pode ser mais apropriada para uma generalização da lógica apresentada em [52] para lidar com jogos simultâneos com comunicação, como ilustramos brevemente.

7.5.3 Sistema Axiomático

Consideramos o seguinte conjunto de axiomas e regras, onde p e q são símbolos proposicionais e φ e ψ são fórmulas.

(LP) Tautologias Proposicionais

(K) $\vdash [P](p \to q) \to ([P]p \to [P]q)$

(Du) $\vdash [P]p \leftrightarrow \neg\langle P\rangle\neg p$

(0) $\vdash \neg\langle \mathbf{0}\rangle p$

(END) $\vdash \langle END\rangle p \leftrightarrow p$

(Pr) $\vdash \langle \alpha.P\rangle p \leftrightarrow \langle c(\alpha)\rangle\langle P\rangle p$

(SC) $\vdash \langle P_1; P_2\rangle p \leftrightarrow \langle P_1\rangle\langle P_2\rangle p$

(NC) $\vdash \langle P_1 + P_2\rangle p \leftrightarrow \langle P_1\rangle p \vee \langle P_2\rangle p$

(Rec) $\vdash \langle P^*\rangle p \leftrightarrow p \vee \langle P\rangle\langle P^*\rangle p$

(FP) $\vdash p \wedge [P^*](p \to [P]p) \to [P^*]p$

(MP) Se $\vdash \varphi$ e $\vdash \varphi \to \psi$, então $\vdash \psi$

(Gen) Se $\vdash \varphi$, então $\vdash [P]\varphi$

(νBi) Se $P \overset{\nu}{\sim} Q$, então $\vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$

(Sub) Se $\vdash \varphi$, então $\vdash \varphi^\sigma$, onde σ substitui uniformemente símbolos proposicionais por fórmulas arbitrárias

(PCSub) Se $\vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$, então $\vdash \langle P|R\rangle p \leftrightarrow \langle Q|R\rangle p$

(RSub) Se $\vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$, então $\vdash \langle(\nu x)P\rangle p \leftrightarrow \langle(\nu x)Q\rangle p$

(Exp) Se a Lei de Expansão pode ser aplicada a P, então $\vdash \langle P\rangle p \leftrightarrow \langle Exp(P)\rangle p$

(Ard) Se $\vdash \langle P\rangle p \leftrightarrow \langle A; P + B\rangle p$ e $A \overset{END}{\not\to} \surd$, então $\vdash \langle P\rangle p \leftrightarrow \langle A^*; B\rangle p$

Definição 7.31. *Definimos a seguinte relação entre processos: $P \leftrightarrow Q$ se e somente se $\vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$.*

Teorema 7.32. *$\leftrightarrow$ é uma congruência.*

Apresentamos agora um teorema chave para a prova de completude da axiomatização acima.

Teorema 7.33. *Seja $P = P_1 \mid P_2$, onde P é irrestrito. Então $P \leftrightarrow \overline{P}$, onde $\overline{P}$ não possui nenhuma ocorrência do operador $\mid$.*

Demonstração. Consulte o teorema 7.26, na seção anterior, e o teorema D.36, no apêndice D. □

Teorema 7.34 (Correção e Completude)**.** *O sistema axiomático acima é correto e completo para πDL.*

Capítulo 8

Conclusões

"A culpa é minha e eu a coloco em quem eu quiser!" - Homer Simpson

Neste capítulo, analisamos os nossos resultados, apresentamos nossas conclusões e delineamos alguns possíveis trabalhos futuros.

8.1 Propriedades Globais de Grafos

Nosso primeiro objetivo nesta tese era expressar, utilizando lógicas modais, algumas propriedades globais de grafos que são centrais para muitas aplicações em ciência da computação. Nosso trabalho voltou-se para como encontrar fórmulas que expressem cada uma das propriedades em que estamos interessados e que possam ser utilizadas de maneira eficiente para testar se um dado grafo as satisfaz.

Neste trabalho, apresentamos vários formalismos, de uma linguagem modal básica até uma linguagem temporal muito poderosa, e os utilizamos para expressar e verificar quatro propriedades de grafos: conectividade, aciclicidade e as propriedades hamiltoniana e euleriana. Seria interessante continuar esta linha de trabalho para tentar expressar algumas outras propriedades de grafos, como planaridade e k-colorabilidade de vértices e arestas.

Este trabalho também expõe uma questão importante. Em alguns casos, lógicas modais comuns, mesmo algumas que são incrivelmente expressivas, não são capazes de expressar algumas propriedades importantes. Isto acontece devido a algumas condições fortes de invariância que estas lógicas satisfazem. Nestes casos, o uso de uma lógica híbrida é uma forma simples de contornar este problema. Lógicas

híbridas possuem condições de invariância consideravelmente mais fracas [14], o que aumenta o número de propriedades que podem ser expressas.

Na seção 5.5, fomos capazes de encontrar uma fórmula (teorema 5.34) em uma linguagem híbrida com o operador $\downarrow$ que expressa a propriedade hamiltoniana e pode ser verificada em tempo NP. Este é um resultado ótimo, uma vez que o problema de determinar se um grafo é hamiltoniano é NP-Completo [33].

O teorema 5.34 também implica que podemos reduzir polinomialmente o problema de determinar se um grafo é hamiltoniano, que é NP-Completo, ao problema da verificação de modelo de uma fórmula da lógica híbrida para grafos com quantificador local $\downarrow$ que não contenha os operadores $\Box$, $\Diamond^+$ e $\Box^+$. Esta é uma prova alternativa de que o problema da verificação de modelo enunciado no teorema 5.32 é NP-Difícil. O prova original de NP-Dificuldade no teorema 5.32, apresentada em [19], utiliza uma redução polinomial diferente, começando não do problema de determinar se um grafo é hamiltoniano, mas do problema de determinar se uma fórmula da lógica proposicional é satisfazível, que também é um problema NP-Completo [33].

De fato, a fórmula que expressa a propriedade hamiltoniana e a fórmula apresentada em [19] para expressar o problema da satisfazibilidade proposicional são bastante semelhantes, consistindo de uma sequência alternante de operadores $\downarrow$ e operadores $\Diamond$. Isto sugere que o fragmento apresentado em [19] pode ser um arcabouço simples para a descrição de problemas NP-Completos. Expressar estes problemas em uma linguagem comum poderia destacar possíveis similaridades subjacentes entre eles. Como a teoria de grafos possui uma rica coleção de problemas NP-Completos, também seria interessante analisar como podemos expressá-los neste fragmento.

Na seção 5.6, utilizamos uma lógica modal graduada para expressar a propriedade euleriana. A fórmula resultante pode ser verificada com uma eficiência bastante boa. De fato, lógicas modais graduadas parecem ser um escolha muito boa para a verificação eficiente de propriedades de grafos que envolvam condições numéricas.

Finalmente, também na seção 5.6, descrevemos uma maneira de nomear arestas em uma linguagem híbrida utilizando subdivisões de grafos. Uma questão em aberto a respeito deste método se refere ao fato de que algumas fórmula satisfeitas em um vértice v em G não são mais satisfeitas no mesmo vértice em $\mathcal{E}(G)$. Por exemplo,

uma fórmula $\Diamond\varphi$ pode ser satisfeita em v em G mas não em $\mathcal{E}(G)$, devido aos novos vértices que são adicionados entre os antigos. Se queremos apenas avaliar fórmulas em $\mathcal{E}(G)$ e podemos esquecer G, como fizemos na seção 5.6, então isto não é um problema. Mas se queremos trabalhar com os dois grafos em paralelo, então seria interessante definir uma tradução $\mathcal{T}$ entre fórmulas de modo que se φ é satisfeita em v em G, então $\mathcal{T}(\varphi)$ é satisfeita em v em $\mathcal{E}(G)$. Também seria interessante estudar quais outras propriedades poderiam ser expressas com lógicas híbridas utilizando esta construção que nos permite nomear não apenas vértices, mas também arestas.

Os resultados obtidos nesta linha de pesquisa foram publicados em artigos nos congressos LSFA 2007 [27] e LSFA 2008 [28] e no periódico *Logic Journal of the IGPL* [29].

8.2 Produtos de Grafos

Ainda no estudo de propriedades de grafos, também buscamos determinar uma condição necessária e suficiente para um grafo ser um produto cartesiano de grafos não triviais. É sabido que comutatividade à esquerda e à direita e as propriedades de Church-Rosser e Church-Rosser reversa são condições necessárias para um grafo ser um produto, mas a sua conjunção não é uma condição suficiente. Nós introduzimos uma nova propriedade chamada intransitividade, que, em conjunto com as propriedades anteriores, forma uma condição necessária e suficiente para um grafo enumerável e conexo ser um produto. Mostramos que a propriedade de intransitividade não pode ser expressa em uma linguagem modal básica. Também mostramos que nenhuma condição que seja necessária e suficiente para um grafo ser um produto pode ser expressa em uma linguagem modal básica. Então, estendemos a linguagem para uma linguagem híbrida e mostramos que em tal linguagem é possível expressar intransitividade.

Também determinamos a complexidade computacional de testar, para um grafo finito e conexo, se ele é um produto. Para este teste, utilizamos um algoritmo de verificação de modelo para verificar as fórmulas que descrevem cada uma das cinco propriedades que caracterizam um produto. Mostramos que este teste pode ser executado em tempo polinomial tanto no tamanho do grafo (tempo cúbico) quanto

no número de dimensões do grafo (tempo quadrático).

Finalmente, utilizamos a caracterização acima para produtos enumeráveis e conexos para construir sistemas axiomáticos corretos e completos para uma grande classe de produtos de lógicas modais. Enquanto a maioria dos sistemas axiomáticos corretos e completos para produtos de lógicas modais apresentados na literatura são para produtos de um par de lógicas modais, somos capazes, utilizando lógicas híbridas, de construir axiomatizações corretas e completas para muitos produtos de dimensões arbitrárias.

A lógica $\mathbf{S5}^n$ está relacionada ao campo de álgebras cilíndricas [53, 54, 55]: as álgebras modais correspondentes a $\mathbf{S5}^n$ são as álgebras cilíndricas representáveis livres de elementos diagonais de dimensão n [41]. Como um trabalho futuro, seria interessante analisar se a axiomatização que podemos construir para $\mathbf{S5}^n$ utilizando os resultados da seção 6.7 poderia ser útil do ponto de vista algébrico.

Em [56], produtos de lógicas são estudados do ponto de vista de lógicas espaciais e topológicas. No último capítulo de [56], o autor desenvolve um trabalho preliminar no possível uso de lógica híbrida para o estudo de lógicas espaciais e topológicas e seus produtos. Como outro trabalho futuro, seria interessante analisar se nossos resultados são úteis ou significativos para o caso de lógicas espaciais e topológicas.

Os resultados obtidos nesta linha de pesquisa foram publicados em um artigo no congresso LSFA 2009 [38] e submetidos em versão mais completa para o periódico *Theoretical Computer Science* [32].

8.3 Lógicas para Sistemas Concorrentes

No segundo tópico do nosso trabalho, apresentamos lógicas dinâmicas proposicionais para sistemas comunicantes concorrentes em que os programas são descritos em linguagens baseadas em CCS e no π-Cálculo sem replicação. Do ponto de vista de lógicas dinâmicas, estas lógicas representam um avanço em relação ao cenário atual, uma vez que lógicas dinâmicas apresentadas anteriormente não podiam lidar de maneira efetiva com concorrência e comunicação simultaneamente. CPDL [6] e PDL com intercalamento[1] [9] lidam com concorrência, mas não oferecem a possibili-

[1] interleaving, em inglês

dade de comunicação entre os componentes de um sistema concorrente. CPDL com canais [7] modela concorrência e comunicação mas possui uma semântica "consideravelmente complicada" [7], é indecidível e não possui uma axiomatização completa. A lógica desenvolvida na tese [8] também possui uma semântica complicada, em que os estados de um modelo de Kripke precisam ser separados em dois conjuntos disjuntos: os estados finais e os estados não finais. Por outro lado, somos capazes de construir uma semântica de Kripke simples para nossas lógicas, baseada na ideia de execuções finitas possíveis de processos, construir axiomatizações completas para elas e mostrar que elas possuem a propriedade do modelo finito.

Também apresentamos um método, em uma linguagem com operadores de iteração (*) e composição sequencial (;), para reescrever qualquer especificação de processo em uma forma sem o operador de composição paralela ($|$) preservando o conjunto de execuções finitas possíveis do processo. Este método é baseado no algoritmo de Brzozowski para encontrar a expressão regular que corresponde a um autômato finito [50]. Acreditamos que esta é uma aplicação interessante e original da ideia de Brzozowski e que ela proporciona uma prova elegante para um resultado chave para a completude das duas últimas axiomatizações.

Deve-se notar, no entanto, que, enquanto o operador $|$ pode ser removido das especificações, na prática pode ser muito complicado descrever um comportamento concorrente complexo sem utilizar este operador desde o princípio. Além disso, mesmo que ambas as especificações, com e sem $|$, possam ser equivalentes, a que possui o operador $|$ será bem mais sucinta.

Como continuação deste trabalho, seria interessante estudar as complexidades do problema da satisfazibilidade e do problema da verificação de modelo para estas lógicas. Também seria interessante desenvolver um provador automático de teoremas e um verificador de modelos para estas lógicas. Isto envolveria, entre outras coisas, um método algorítmico eficiente para lidar com a expansão de processos paralelos e, no caso específico da segunda lógica, para determinar os processos L_P e T_P relacionados a um processo nó P. Gostaríamos também de analisar a questão de processos com capacidade de replicação, que foi deixada de fora do presente trabalho. Seria interessante estudar o que mudaria nas lógicas com a adição do operador de replicação do π-Cálculo (!).

Finalmente, gostaríamos também de estudar em mais detalhes as possíveis conexões entre nossas lógicas e as ideias apresentadas em [52] e analisar como usar nossas lógicas como uma ferramenta para a descrição de jogos simultâneos com comunicação.

Os resultados obtidos nesta linha de pesquisa foram publicados em artigos nos congressos WoLLIC 2008 [43] e M4M 2009 [45] e submetidos para o periódico *ACM Transactions on Computational Logic* [44].

Referências Bibliográficas

[1] BONDY, J. A., MURTY, U. S. R., *Graph Theory with Applications*. New York, Elsevier, 1976.

[2] BARBOSA, V. C., *An Introduction to Distributed Algorithms*. Cambridge, MIT Press, 1996.

[3] LYNCH, N., *Distributed Algorithms*. San Francisco, Morgan Kaufmann Publishers, 1996.

[4] GABBAY, D., KURUCZ, A., WOLTER, F., et al., *Many-dimensional modal logics: theory and applications*. Amsterdam, Elsevier Science, 2003.

[5] KURUCZ, A., "Combining Modal Logics", In: BLACKBURN, P., VAN BENTHEM, J., WOLTER, F. (eds), *Handbook of Modal Logics*, pp. 869–924, 2006.

[6] PELEG, D., "Concurrent Dynamic Logic", *Journal of the Association for Computing Machinery*, v. 34, n. 2, pp. 450–479, 1987.

[7] PELEG, D., "Communication in Concurrent Dynamic Logic", *Journal of Computer and System Sciences*, v. 35, n. 1, pp. 23–58, 1987.

[8] DOS SANTOS, V. L. P., *Concorrência e Sincronização para Lógica Dinâmica de Processos*, Tese de Doutorado, Universidade Federal do Rio de Janeiro, 2005.

[9] MAYER, A. J., STOCKMEYER, L. J., "The Complexity of PDL with Interleaving", *Theoretical Computer Science*, v. 161, n. 1–2, pp. 109–122, 1996.

[10] BLACKBURN, P., DE RIJKE, M., VENEMA, Y., *Modal Logic. Theoretical Tracts in Computer Science*, Cambridge, Cambridge University Press, 2001.

[11] CLARKE, E. M., GRUMBERG, O., PELED, D., *Model Checking*. Cambridge, MIT Press, 2000.

[12] VAN DER HOEK, W., DE RIJKE, M., "Counting Objects", *Journal of Logic and Computation*, v. 5, n. 3, pp. 325–345, 1995.

[13] FERRANTE, A., NAPOLI, M., PARENTE, M., "CTL Model-Checking with Graded Quantifiers". In: *6th International Symposium on Automated Technology for Verification and Analysis*, v. 5311, *LNCS*, pp. 18–32, 2008.

[14] ARECES, C., TEN CATE, B., "Hybrid Logics", In: BLACKBURN, P., VAN BENTHEM, J., WOLTER, F. (eds), *Handbook of Modal Logic*, pp. 821–868, 2006.

[15] BLACKBURN, P., "Representation, Reasoning, and Relational Structures: a Hybrid Logic Manifesto", *Logic Journal of the IGPL*, v. 8, n. 3, pp. 339–365, 2000.

[16] ARECES, C., BLACKBURN, P., MARX, M., "A Road-map on Complexity for Hybrid Logics". In: *Proceedings of the 8th Annual Conference of the EACSL*, v. 1683, *Lecture Notes in Computer Science*, pp. 307–321, 1999.

[17] FRANCESCHET, M., DE RIJKE, M., "Model Checking Hybrid Logics (With An Application to Semistructured Data)", *Journal of Applied Logic*, v. 4, pp. 279–304, 2006.

[18] BLACKBURN, P., TEN CATE, B., "Pure extensions, proof rules, and hybrid axiomatics", *Studia Logica*, v. 84, n. 2, pp. 277–322, 2006.

[19] TEN CATE, B., FRANCESCHET, M., "On the complexity of hybrid logics with binders". In: *Proceedings of the 19th International Workshop of Computer Science Logic*, v. 3634, *Lecture Notes in Computer Science*, pp. 339–354, 2005.

[20] EMERSON, E. A., "Temporal and Modal Logic", In: VAN LEEUWEN, J. (ed), *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pp. 995–1072, 1990.

[21] MILNER, R., *Communication and Concurrency*. Harlow, Prentice Hall, 1989.

[22] FOKKINK, W. J., *Introduction to Process Algebra. Texts in Theoretical Computer Science*, Berlin, Springer, 2000.

[23] STIRLING, C., *Modal and Temporal Properties of Processes. Texts in Computer Science*, New York, Springer, 2001.

[24] MILNER, R., *Communicating and Mobile Systems: the π-Calculus*. Cambridge, Cambridge University Press, 1999.

[25] MILNER, R., PARROW, J., WALKER, D., "A Calculus of Mobile Processes - Part I and Part II", *Information and Computation*, v. 100, n. 1, pp. 1–77, 1992.

[26] PARROW, J., "An Introduction to the π-Calculus", In: BERGSTRA, J. A., PONSE, A., SMOLKA, S. A. (eds), *Handbook of Process Algebra*, pp. 479–543, 2001.

[27] BENEVIDES, M. R. F., SCHECHTER, L. M., "Modal Expressiveness of Graph Properties". In: *Proceedings of the 2nd Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2007)*, v. 205, *Electronic Notes in Theoretical Computer Science*, pp. 31–47, 2008.

[28] SCHECHTER, L. M., "A Logical Approach to Hamiltonian Graphs". In: *Proceedings of the 3rd Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2008)*, v. 247, *Electronic Notes in Theoretical Computer Science*, pp. 123–138, 2009.

[29] BENEVIDES, M. R. F., SCHECHTER, L. M., "Using Modal Logics to Express and Check Global Graph Properties", *Logic Journal of the IGPL*, v. 17, n. 5, pp. 509–537, 2009.

[30] BENEVIDES, M. R. F., "Modal Logics for Finite Graphs", In: QUEIROZ, R. (ed), *Logic for Synchronization and Concurrency*, pp. 239–267, *Trends in Logic*, 2003.

[31] ARECES, C., BLACKBURN, P., MARX, M., "The Computational Complexity of Hybrid Temporal Logics", *Logic Journal of the IGPL*, v. 8, pp. 653–679, 2000.

[32] BENEVIDES, M. R. F., SCHECHTER, L. M., "A Study on Multi-Dimensional Products of Graphs and Hybrid Logics", Submetido para o *Theoretical Computer Science*.

[33] KARP, R., "Reducibility Among Combinatorial Problems", In: *Complexity of Computer Computations*, pp. 85–103, New York, Plenum, 1972.

[34] MOLLER, F., RABINOVICH, A., "On the Expressive Power of CTL*". In: *XIV IEEE Symposium on Logic in Computer Science*, pp. 360–369, 1999.

[35] SATTLER, U., VARDI, M. Y., "The Hybrid μ-Calculus". In: *First International Joint Conference in Automated Reasoning*, pp. 76–91, 2001.

[36] DAM, M., "CTL* and ECTL* as Fragments of the Modal Mu-Calculus", *Theoretical Computer Science*, v. 126, pp. 77–96, 1994.

[37] FINE, K., "In so Many Possible Worlds", *Notre Dame Journal of Formal Logic*, v. 13, n. 4, pp. 516–520, 1972.

[38] BENEVIDES, M. R. F., SCHECHTER, L. M., "Product of Graphs and Hybrid Logic". In: *Proceedings of the 4th Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2009)*, v. 256, *Electronic Notes in Theoretical Computer Science*, pp. 103–118, 2009.

[39] TEN CATE, B., *Model theory for extended modal languages*, Tese de Doutorado, University of Amsterdam, 2005.

[40] GABBAY, D., SHEHTMAN, V., "Products of Modal Logics, Part 1", *Logic Journal of the IGPL*, v. 6, n. 1, pp. 73–146, 1998.

[41] HIRSCH, R., HODKINSON, I., KURUCZ, A., "On Modal Logics between $\mathbf{K} \times \mathbf{K} \times \mathbf{K}$ and $\mathbf{S5} \times \mathbf{S5} \times \mathbf{S5}$", *Journal of Symbolic Logic*, v. 67, n. 1, pp. 221–234, 2002.

[42] ENDERTON, H. B., *A Mathematical Introduction to Logic*. 2001.

[43] BENEVIDES, M. R. F., SCHECHTER, L. M., "A Propositional Dynamic Logic for CCS Programs". In: *Proceedings of the 15th Workshop on Logic, Language, Information and Computation (WoLLIC 2008)*, v. 5110, *Lecture Notes in Artificial Inteligence*, pp. 83–97, 2008.

[44] BENEVIDES, M. R. F., SCHECHTER, L. M., "CCS-Based Dynamic Logics for Communicating Concurrent Programs", `http://arxiv.org/abs/0904.0034`, Submetido para o *ACM Transactions on Computational Logic*.

[45] BENEVIDES, M. R. F., SCHECHTER, L. M., "A Propositional Dynamic Logic for Concurrent Programs Based on the Pi-Calculus". In: *Proceedings of the 6th Workshop on Methods for Modalities (M4M 2009)*, *Electronic Notes in Theoretical Computer Science*, a ser publicado.

[46] FISCHER, M. J., LADNER, R. E., "Propositional Dynamic Logic of Regular Programs", *Journal of Computer and System Sciences*, v. 18, n. 2, pp. 194–211, 1979.

[47] MILNER, R., PARROW, J., WALKER, D., "Modal Logics for Mobile Processes", *Theoretical Computer Science*, v. 114, n. 1, pp. 149–171, 1993.

[48] DAM, M., "Model Checking Mobile Processes", *Information and Computation*, v. 129, n. 1, pp. 35–51, 1996.

[49] ARDEN, D. N., "Delayed logic and finite state machines", In: *Theory of Computing Machine Design*, pp. 1–35, 1960.

[50] BRZOZOWSKI, J. A., "Derivatives of Regular Expressions", *Journal of the ACM*, v. 11, n. 4, pp. 481–494, 1964.

[51] VAN BENTHEM, J., "Extensive Games as Process Models", *Journal of Logic, Language and Information*, v. 11, n. 3, pp. 289–313, 2002.

[52] VAN BENTHEM, J., GHOSH, S., LIU, F., "Modelling simultaneous games in dynamic logic", *Synthese*, v. 165, n. 2, pp. 247–268, 2008.

[53] VENEMA, Y., *Many-Dimensional Modal Logic*, Tese de Doutorado, University of Amsterdam, 1992.

[54] MARX, M., VENEMA, Y., *Multi-dimensional Modal logic*. 1997.

[55] HENKIN, L., MONK, J. D., TARSKI, A., *Cylindric Algebras, Parts I & II*. 1971 & 1985.

[56] SARENAC, D., *Products of Topological Modal Logics*, Tese de Doutorado, Stanford University, 2006.

[57] BRADFIELD, J., STIRLING, C., "Modal Mu Calculi", In: BLACKBURN, P., VAN BENTHEM, J., WOLTER, F. (eds), *Handbook of Modal Logic*, pp. 721–756, 2006.

[58] VENEMA, Y., "Lectures on the Modal Mu-Calculus", `http://staff.science.uva.nl/~yde/teaching/ml/mu/mu.pdf`.

[59] STRECKER, M., "Modeling and Verifying Graph Transformations in Proof Assistants". In: *4th International Workshop on Computing with Terms and Graphs*, pp. 112–124, 2007.

[60] KOVÁCS, M., GÖNCZY, L., VARRÓ, D., "Formal Analysis of BPEL Workflows with Compensation by Model Checking", *International Journal of Computer Systems and Engineering*, a ser publicado.

[61] ARMANDO, A., CARBONE, R., COMPAGNA, L., "LTL Model Checking for Security Protocols". In: *20th IEEE Computer Security Foundations Symposium*, pp. 385–396, 2007.

[62] BALTAZAR, P., CHADHA, R., MATEUS, P., "Quantum computation tree logic – model checking and complete calculus", *International Journal of Quantum Information*, v. 6, n. 2, pp. 219–236, 2008.

[63] GAY, S. J., NAGARAJAN, R., PAPANIKOLAOU, N., "Model-Checking Quantum Protocols", http://www.warwick.ac.uk/~essiai/downloads/gnp.pdf.

[64] BLACKBURN, P., TZAKOVA, M., "Hybrid Languages and Temporal Logic", *Logic Journal of the IGPL*, v. 7, n. 1, pp. 27–54, 1999.

Apêndice A

Using Modal Logics to Express and Check Global Graph Properties

A.1 Introduction[1]

Graphs are among the most frequently used structures in Computer Science [1]. Usually, in this discipline, many important concepts admit a graph representation, and sometimes a graph lies at the very kernel of the model of computation used. This happens, for instance, in the field of distributed systems [2, 3], where the underlying model of computation is built on top of a graph. In addition to this central role, graphs are also important in distributed systems as tools for the description of resource sharing problems, scheduling problems, deadlock issues, and so on. The case of distributed systems is particularly appealing because it illustrates well two different levels at which graph properties have to be described. One is the local level, encompassing properties that hold for vertices or constant-size vertex-neighborhoods. The other level is global and comprises properties that hold for the graph as a whole, as acyclicity and connectivity.

Graph theory provides a lot of tools to describe such problems and presents many efficient algorithmic methods to solve them. However, there is an important

[1]Este capítulo expõe o conteúdo do artigo [29], publicado no Logic Journal of the IGPL, v. 17, n. 5, p. 509-537, 2009.

distinction between the two sides of this matter. In the "description" side, graphs provide a great level of generality, allowing for the description of very different problems in the same simple framework. But in the "solution" side, each graph problem has to be solved and each graph property has to be tested with a specific method that usually does not generalize to other different problems or properties.

A logical framework, on the other hand, may provide this level of generalization. In an intuitive and non-technical language, this can be stated as follows. Consider a logic $\mathbb{L}$ with its formulas and structures where these formulas are semantically evaluated. We need to be able to answer the following questions:

1. Can we encode a graph as a structure for $\mathbb{L}$?
2. Can we encode the graph properties that we want to verify as $\mathbb{L}$-formulas?
3. Does $\mathbb{L}$ has decidable inference methods to check whether a formula is satisfied (or valid) in a structure?

If the answers to all of these questions are positive, then we can use the inference methods of the logic to verify every graph property that we want, provided that we can express it as an $\mathbb{L}$-formula. Of course, there is still a fourth question that has to be answered:

4. Is the logical method as efficient (in terms of computational complexity) in testing a given property as the graph theoretical method?

In order to satisfy the first question, we choose to work with the family of modal logics. A very strong reason to choose modal logics for this task, instead of any other logic, is that modal logic formulas are evaluated in structures that are essentially graphs, which makes it a very natural choice for our work. As the first slogan in the preface of [10] states, "modal languages are simple yet expressive languages for talking about relational structures".

In this work, we analyze how we can express and efficiently check these global graph properties with modal logics. This involves two issues: the first is whether each of the modal languages that we consider has enough expressive power to describe the graph properties that we want; the second is how complex (computationally)

it is to use these logics to actually test whether a given graph has some desired property.

A similar attempt to do this was presented in [30]. That work also tries to use modal logics to express graph properties, but it approaches this issue from a different point of view. The key differences between the two works will be discussed in the last section of this paper.

A *finite directed graph* (from now on called simply a *graph*) G is a pair (W, R), where W is a finite set of vertices and $R \subseteq W \times W$ is a set of ordered pairs of vertices (a binary relation on W), called edges. If $\langle v_i, v_j \rangle \in R$, we say that v_i is *adjacent to* v_j and v_j is *adjacent from* v_i. The *out-degree* of a vertex is the number of vertices adjacent from it and the *in-degree* the number of vertices adjacent to it. The set R of edges can also be written as a relation between two vertices v_i and v_j. We write $v_i R v_j$ to express the fact that v_i is adjacent to v_j.

A *path* in a graph G is a sequence of vertices $\langle v_1, v_2, \ldots, v_n \rangle$, $n \geq 2$, where $\langle v_i, v_{i+1} \rangle \in R$, for $0 < i < n$. We say that n is the length of the path. A *closed path* is a path such that $v_1 = v_n$. A *cycle* is a path where $v_1 = v_n$ and $v_i \neq v_j$, for $1 \leq i, j < n$, $i \neq j$. A graph G is said to be *acyclic* if there is no cycle in it, otherwise it is *cyclic*.

Every directed graph has an *underlying undirected graph*, in which we do not consider the particular orientation of the edges. This means that, if $G = (W, R)$ and $\langle v, w \rangle \in R$, then v is adjacent to w and w is adjacent to v in the underlying undirected graph of G.

The rest of this paper is organized as follows. In section A.2, we present a simple modal logic suited for the description of graph properties. In section A.3, we investigate the issue of whether some well-known graph properties are definable or not in the language presented in the previous section: connectivity, acyclicity and the Eulerian and Hamiltonian properties. In section A.4, we extend the modal logic of the previous sections with nominals, obtaining a hybrid modal logic, and use it to express some of the above properties. In section A.5, we introduce the branching-time temporal logic CTL* with nominals, which is a very expressive logic, and use it to check the Hamiltonian property in a better way than it was done in the basic hybrid logic. In section A.6, we extend the basic hybrid logic of section A.4 with

the $\downarrow$ operator and show that we can check the Hamiltonian property with optimal (NP) complexity in this logic. In section A.7, we tackle the Eulerian property in two different ways. First, we develop a generic method to express edge-related properties in hybrid logics and use it to express the Eulerian property. Second, we express a necessary and sufficient condition for the Eulerian property to hold using a graded modal logic. Finally, in section A.8 we draw our concluding remarks.

A.2 Basic Graph Logic

In this section, we define a modal logic with two modal operators: $\Diamond$ and $\Diamond^+$. We call it *basic graph logic*.

Definition A.1. *The language of the* basic graph logic *is a modal language consisting of a set Φ of countably many proposition symbols (the elements of Φ are denoted by $p_1, p_2, \ldots$), the boolean connectives $\neg$ and $\wedge$ and two modal operators: $\Diamond$ and $\Diamond^+$. The formulas are defined as follows:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond\varphi \mid \Diamond^+\varphi$$

We freely use the standard boolean abbreviations $\vee$, $\rightarrow$, $\leftrightarrow$ and $\bot$ and also the following abbreviations for the duals: $\Box\varphi := \neg\Diamond\neg\varphi$ and $\Box^+\varphi = \neg\Diamond^+\neg\varphi$. Also, in order to make the language more elegant, we introduce some abbreviations for the reflexive and transitive closures: $\Diamond^*\varphi = \varphi \vee \Diamond^+\varphi$ and $\Box^*\varphi = \neg\Diamond^*\neg\varphi$.

We now define the structures in which we evaluate formulas in modal logics: *frames* and *models*.

Definition A.2. *A* frame *for the basic graph logic is a pair $\mathcal{F} = (W, R)$, where W is a non-empty set (finite or not) of vertices and R is a binary relation over W, i.e., $R \subseteq W \times W$.*

As we see, a frame for the basic graph logic is essentially a graph. This confirms our statement in the first section that modal logics are a very natural choice for this work.

Definition A.3. *A* model *for the basic graph logic is a pair $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, where $\mathcal{F}$ is a frame and $\mathbf{V}$ is a valuation function mapping proposition symbols into subsets of W, i.e., $\mathbf{V} : \Phi \mapsto \mathcal{P}(W)$.*

The semantical notion of satisfaction is defined as follows:

Definition A.4. *Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. The notion of* satisfaction *of a formula φ in a model $\mathcal{M}$ at a vertex v, notation $\mathcal{M}, v \Vdash \varphi$, can be inductively defined as follows:*

1. *$\mathcal{M}, v \Vdash p$ iff $v \in \mathbf{V}(p)$;*
2. *$\mathcal{M}, v \Vdash \top$ always;*
3. *$\mathcal{M}, v \Vdash \neg\varphi$ iff $\mathcal{M}, v \nVdash \varphi$;*
4. *$\mathcal{M}, v \Vdash \varphi_1 \wedge \varphi_2$ iff $\mathcal{M}, v \Vdash \varphi_1$ and $\mathcal{M}, v \Vdash \varphi_2$;*
5. *$\mathcal{M}, v \Vdash \Diamond\varphi$ iff there is a $w \in W$ such that vRw and $\mathcal{M}, w \Vdash \varphi$;*
6. *$\mathcal{M}, v \Vdash \Diamond^+\varphi$ iff there is a $w \in W$ such that vR^+w and $\mathcal{M}, w \Vdash \varphi$.*

Here, R^+ denotes the transitive closure of R.

A formula $\Diamond\varphi$ is satisfied at a vertex v if, for some vertex w, vRw and φ is satisfied at w. A formula $\Diamond^+\varphi$ is satisfied at a vertex v if there is a path $\langle v_1, \ldots, v_n\rangle$, $n \geq 2$, such that $v = v_1$, $w = v_n$, v_iRv_{i+1} for $1 \leq i < n$ and φ is satisfied at w.

Example A.5. *Let $\mathcal{M}$ be the model shown (without its valuation) in figure A.1. In order to illustrate the use of the logic, we can see that the following formulas are satisfied at vertex w in $\mathcal{M}$, supposing that φ is satisfied at vertex v in $\mathcal{M}$: $\mathcal{M}, w \Vdash \Diamond\varphi$, $\mathcal{M}, w \Vdash \Diamond\Diamond\Diamond\varphi$, $\mathcal{M}, w \Vdash \Diamond^+\varphi$ and $\mathcal{M}, w \Vdash \Diamond^+\Box\bot$.*

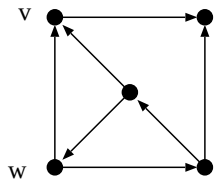

Figure A.1: Model $\mathcal{M}$, where a formula φ is satisfied at vertex v.

If $\mathcal{M}, v \Vdash \varphi$ for every vertex v in a model $\mathcal{M}$, we say that φ is *globally satisfied* in $\mathcal{M}$, notation $\mathcal{M} \Vdash \varphi$. And if φ is globally satisfied in all models $\mathcal{M}$ of a frame $\mathcal{F}$, we say that φ is *valid* in $\mathcal{F}$, notation $\mathcal{F} \Vdash \varphi$.

In this work, we want to find a modal formula ϕ (for each property), such that a graph G has the desired property if and only if $\mathcal{F} \Vdash \phi$, where $\mathcal{F}$ is the frame that represents G.

For each modal logic that we consider for this task, there are two issues involved. The first one is whether the modal language has enough expressive power to describe the graph properties that we want. In case the answer is negative, we need to search for a language with greater expressive power. In case the answer is positive and we are able to find such a formula, then we need to calculate how complex (computationally) it is to use the inference mechanisms of the logic to actually test, using the formula that we found, whether a given graph has the desired property.

The issue of expressive power with respect to the language of the basic graph logic will be addressed in the next section. The issue of the complexity for testing the graph properties involves four basic decision problems.

Definition A.6. *The* satisfiability problem *consists of, given a formula ϕ, determining whether there is a model $\mathcal{M}$ and a vertex v in $\mathcal{M}$ such that $\mathcal{M}, v \Vdash \phi$.*

Definition A.7. *The* validity problem *consists of, given a formula ϕ, determining whether $\mathcal{F} \Vdash \phi$, for all frames $\mathcal{F}$.*

The satisfiability problem and the validity problem are duals to each other.

Definition A.8. *The* model-checking problem *consists of, given a formula ϕ and a finite model $\mathcal{M} = (W, R, \mathbf{V})$, determining the set $S_{\mathcal{M}}(\phi) = \{v \in W : \mathcal{M}, v \Vdash \phi\}$.*

Definition A.9. *The* frame-checking problem *consists of, given a formula ϕ and a finite frame $\mathcal{F}$, determining whether $\mathcal{F} \Vdash \phi$.*

Definition A.10. *We define the length of a formula φ, denoted by $|\varphi|$, inductively in the following way: $|p| = |\top| = 1$, $|\neg\phi| = |\Diamond\phi| = |\Diamond^+\phi| = 1 + |\phi|$ and $|\phi_1 \wedge \phi_2| = 1 + |\phi_1| + |\phi_2|$. In the following logics, analogous rules apply to the new operators.*

Definition A.11. *Let $\mathcal{M} = (W, R, \mathbf{V})$ be a model. Let $|W|$ be the number of vertices in W and $|R|$ the number of pairs in R. We define the size of the model (or the frame, or the graph) as $|W| + |R|$.*

Theorem A.12 ([10])**.** *The satisfiability problem and the validity problem for the basic graph logic are EXPTIME-Complete in the length of the formula.*

Theorem A.13. *The model-checking problem for the basic graph logic is PTIME (linear) in the product of the size of the model and the length of the formula.*

Proof. The basic graph logic is a fragment of CTL, so this result follows from the complexity of the model-checking problem for CTL, presented in [11]. □

We can provide a simple upper bound for the complexity of the frame-checking problem based on the complexity of the correspondent model-checking problem. We have that $\mathcal{F} \Vdash \phi$ if and only if $S_{\mathcal{M}}(\neg\phi) = \emptyset$ for every model $\mathcal{M}$ of $\mathcal{F}$. So, let FC be the complexity of the frame-checking problem and MC be the complexity of the model-checking problem. Then,

$$FC = O(2^{|p|\times n} \times MC),$$

where $|p|$ is the number of distinct proposition symbols that occur in the given formula ϕ and n is the number of vertices in $\mathcal{F}$. We need to apply the model-checking algorithm to every model $\mathcal{M}$ of the given frame $\mathcal{F}$. Every proposition symbol p that appears in ϕ may receive 2^n possible valuations $\mathbf{V}(p)$.

Theorem A.14. *The frame-checking problem for the basic graph logic is PTIME (linear) in the length of the formula and EXPTIME in the size of the frame and in the number of distinct proposition symbols that occur in the formula.*

Proof. This result follows directly from the discussion above. □

It should be noticed that this calculation of the complexity of the frame-checking problem is just a general upper-bound and it can possibly be reduced in some concrete situations.

A.3 Basic Graph Logic Definability

In this section, we investigate whether some well-known global graph properties are definable or not in the language of the basic graph logic. These properties are: connectivity, acyclicity and the Hamiltonian and Eulerian properties.

The limits to the expressive power of basic modal languages are fairly well known. There are a series of standard results that state that frames that are "similar" in a number of ways must agree on the validity of formulas. We can then use these

results to prove that a certain property *cannot* be expressed by any formula in the basic graph logic. To do this, we take two frames that are "similar" and show that in one the desired property holds, while in the other it does not. We present two of these "similarity" results (more details about them and other related results may be found in [10]), and then we prove some theorems for global graph properties using them.

Definition A.15. *Let $\mathcal{F} = (W, R)$ and $\mathcal{F}' = (W', R')$ be two frames. A function $f : W \to W'$ is a* bounded morphism *from $\mathcal{F}$ to $\mathcal{F}'$ if it satisfies the following conditions:*

1. *f is a homomorphism with respect to R (if wRv, then $f(w)R'f(v)$);*

2. *if $f(w)R'v'$, then there is a v such that wRv and $f(v) = v'$.*

If there is a surjective bounded morphism from $\mathcal{F}$ to $\mathcal{F}'$, then we say that $\mathcal{F}'$ is a *bounded morphic image* of $\mathcal{F}$ and use the notation $\mathcal{F} \Rightarrow \mathcal{F}'$.

Definition A.16. *Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ and $\mathcal{M}' = (\mathcal{F}', \mathbf{V}')$ be two models. A function $f : W \to W'$ is a* bounded morphism *from $\mathcal{M}$ to $\mathcal{M}'$ if it satisfies the following conditions:*

1. *f is a bounded morphism from $\mathcal{F}$ to $\mathcal{F}'$;*

2. *w and $f(w)$ satisfy the same proposition symbols.*

If there is a surjective bounded morphism from $\mathcal{M}$ to $\mathcal{M}'$, then we say that $\mathcal{M}'$ is a *bounded morphic image* of $\mathcal{M}$ and use the notation $\mathcal{M} \Rightarrow \mathcal{M}'$.

Other important definitions concern collections of disjoint frames and collections of disjoint models. We say that two frames $\mathcal{F}_1 = (W_1, R_1)$ and $\mathcal{F}_2 = (W_2, R_2)$ are *disjoint frames* if and only if $W_1 \cap W_2 = \emptyset$ and we say that two models $\mathcal{M}_1 = (\mathcal{F}_1, \mathbf{V}_1)$ and $\mathcal{M}_2 = (\mathcal{F}_2, \mathbf{V}_2)$ are *disjoint models* if and only if $\mathcal{F}_1$ and $\mathcal{F}_2$ are disjoint frames.

Definition A.17. *Let $\mathcal{F}_i = (W_i, R_i)$ be a collection (finite or not) of disjoint frames. Their* disjoint union *is the frame $\biguplus \mathcal{F}_i = (W, R)$, where $W = \bigcup_i W_i$ and $R = \bigcup_i R_i$.*

Definition A.18. *Let $\mathcal{M}_i = (\mathcal{F}_i, \mathbf{V}_i)$ be a collection (finite or not) of disjoint models. Their* disjoint union *is the model $\biguplus \mathcal{M}_i = (\mathcal{F}, \mathbf{V})$, where $\mathcal{F}$ is the disjoint union of the frames $\mathcal{F}_i$ and, for each proposition symbol p, $\mathbf{V}(p) = \bigcup_i \mathbf{V}_i(p)$.*

Below are two basic theorems about the definability of properties that are going to be used throughout the next subsections. Their proofs for a basic modal language that contains only $\Diamond$ can be found in [10]. It is not difficult to extend that proof to the language of the basic graph logic, which contains both $\Diamond$ and $\Diamond^+$.

Theorem A.19. *Let $\mathcal{M} = (W, R, \mathbf{V})$ and $\mathcal{M}' = (W', R', \mathbf{V}')$ be two models such that $\mathcal{M} \Rightarrow \mathcal{M}'$. Then, $\mathcal{M}, w \Vdash \phi$ if and only if $\mathcal{M}', f(w) \Vdash \phi$.*

Corollary A.20. *Let $\mathcal{F} = (W, R)$ and $\mathcal{F}' = (W', R')$ be two frames such that $\mathcal{F} \Rightarrow \mathcal{F}'$. If $\mathcal{F} \Vdash \phi$, then $\mathcal{F}' \Vdash \phi$.*

Theorem A.21. *Let $\mathcal{M}_i = (W_i, R_i, \mathbf{V}_i)$ be a collection (finite or not) of disjoint models and $\biguplus \mathcal{M}_i = (W, R, \mathbf{V})$ their disjoint union. Then, $\mathcal{M}_i, w \Vdash \phi$ if and only if $\biguplus \mathcal{M}_i, w \Vdash \phi$.*

Corollary A.22. *Let $\mathcal{F}_i = (W_i, R_i)$ be a collection (finite or not) of disjoint frames and $\biguplus \mathcal{F}_i = (W, R)$ their disjoint union. If $\mathcal{F}_i \Vdash \phi$ for every i, then $\biguplus \mathcal{F}_i \Vdash \phi$.*

Theorem A.23. *The class of finite frames (which is equivalent to our definition of a graph) is not definable in the basic graph logic.*

Proof. The disjoint union of an infinite collection of finite disjoint frames is not finite. By corollary A.22, since this property is not preserved under taking disjoint unions, it is not definable in the basic graph logic. □

A.3.1 Connectivity

Definition A.24. *We can define two levels of connectivity for a graph. Firstly, a graph G is said to be* (weakly) connected *if and only if, for any two vertices v and w in G, there is a path from v to w in the underlying undirected graph of G. Secondly, a graph G is said to be* strongly connected *if and only if, for any two vertices v and w in G, there is a path from v to w in G itself.*

Theorem A.25. *Weak and strong connectivity are not definable in the basic graph logic.*

Proof. The disjoint union of connected graphs is not a connected graph. By corollary A.22, since connectivity is not preserved under taking disjoint unions, it is not definable in the basic graph logic. □

A.3.2 Acyclicity

Definition A.26. *A graph G is said to be acyclic if and only if there is no path in G that is a cycle, as defined in the first section.*

Theorem A.27. *Acyclicity is not definable in the basic graph logic.*

Proof. We can take a frame $\mathcal{F} = (W, R)$ where $W = \mathbb{N}$ and $R = \{\langle i, i+1\rangle, i \in \mathbb{N}\}$ and a frame $\mathcal{F}' = (W', R')$ where $W' = \{O, E\}$ and $R' = \{\langle O, E\rangle, \langle E, O\rangle\}$. If we define f as $f(i) = E$ if i is even and $f(i) = O$ otherwise, we have that f is a surjective bounded morphism between $\mathcal{F}$ and $\mathcal{F}'$. But $\mathcal{F}$ is acyclic while $\mathcal{F}'$ is not. Hence, by corollary A.20, since acyclicity is not preserved under bounded morphic images, it is not definable in the basic graph logic. □

At first, it may seem that the above theorem is too generic for our needs, since we only need to be able to define acyclicity on finite frames and we use an infinite frame as part of our proof. However, as theorem A.23 shows, we cannot write a formula that describes "acyclicity on finite frames" or any other property "on finite frames".

A.3.3 Hamiltonian Graphs

Definition A.28. *A connected graph G is said to be Hamiltonian if and only if there is a cycle in G that goes through every vertex of it.*

Theorem A.29. *The class of Hamiltonian graphs is not definable in the basic graph logic.*

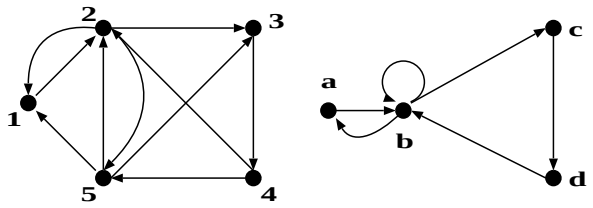

Figure A.2: Graph 1,2,3,4,5 is Hamiltonian and graph a,b,c,d is not.

Proof. From figure A.2, let $f = \{(1, a), (2, b), (3, c), (4, d), (5, b)\}$. It is straightforward to prove that f is a surjective bounded morphism. By corollary A.20, since

the Hamiltonian property is not preserved under bounded morphic images, it is not definable in the basic graph logic. □

A.3.4 Eulerian Graphs

Definition A.30. *A connected graph G is said to be Eulerian if and only if there is a closed path in G in which every edge of it appears exactly once.*

Theorem A.31 ([1])**.** *A connected graph G is Eulerian if and only if the out-degree of every vertex of G is equal to its in-degree.*

Theorem A.32. *The class of Eulerian graphs is not definable in the basic graph logic.*

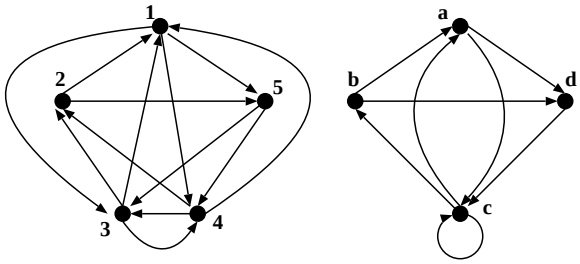

Figure A.3: Graph 1,2,3,4,5 is Eulerian and graph a,b,c,d is not.

Proof. From figure A.3, let $f = \{(1, a), (2, b), (3, c), (4, c), (5, d)\}$. It is straightforward to prove that f is a surjective bounded morphism. By corollary A.20, since the Eulerian property is not preserved under bounded morphic images, it is not definable in the basic graph logic. □

A.3.5 The Modal μ-Calculus

Looking at the results of the previous subsections, we see that, unfortunately, the language of the *basic graph logic* does not have enough expressive power to define the properties that we want. We need a stronger language. One idea could be to use the modal μ-calculus [57, 58]. Its language incorporates fixpoint operators and is very expressive. In fact, not only the *basic graph logic* can be embedded into the μ-calculus, but so can be the temporal logics LTL, CTL and CTL* [36].

Unfortunately, even with all this expressive power, the language of the μ-calculus fails to express these properties because of the same reasons exposed in the previous subsections. This happens because μ-calculus formulas, as the basic graph formulas, are invariant under bisimulations (disjoint unions and bounded morphisms are special cases of bisimulation). In fact, the μ-calculus is the bisimulation-invariant fragment of Monadic Second-Order Logic (MSOL) [57].

To bypass this problem, we introduce a different kind of language in the next section. This language has a mechanism to name vertices of the model and allows us to express the graph properties that we want.

A.4 Hybrid Graph Logic

As was shown in the previous section, the language of the basic graph logic does not have enough expressive power to describe the properties that we want. In order to achieve our goal, we need a logic that has a language with more expressive power but, if possible, is still decidable with respect to the problems stated in the definitions A.6 until A.9.

One interesting class of logics to take into consideration is the class of *hybrid logics* [14, 10]. In these logics, there is a new kind of atomic symbol: *nominals*. Nominals behave similarly to proposition symbols. The key difference between them is related to their valuation in a model. While the set $\mathbf{V}(p)$ for a proposition symbol p can be any element of $\mathcal{P}(W)$, the set $\mathbf{V}(i)$ for a nominal i has to be a singleton set. This way, each nominal is satisfied at exactly one vertex, and thus, can be used to reference a unique vertex of the model.

A hybrid extension of our previous logic is an interesting choice because of a combination of factors. Its language has an improved expressive power, since hybrid formulas are no longer invariant under neither disjoint unions nor bounded morphic images [14], but it is still a decidable logic, as discussed in the following subsection.

In this section, we define an extension of the basic graph logic that includes nominals. We call it *hybrid graph logic*. After that, we try to express, in this new logic, the graph properties that we are discussing.

A.4.1 Language

Definition A.33. *The language of the* hybrid graph logic *is a hybrid language consisting of a set* Φ *of countably many proposition symbols (the elements of* Φ *are denoted by* $p_1, p_2, \ldots$*), a set* $\mathcal{L}$ *of countably many nominals (the elements of* $\mathcal{L}$ *are denoted by* $i_1, i_2, \ldots$*) such that* $\Phi \cap \mathcal{L} = \emptyset$ *(the elements of* $\Phi \cup \mathcal{L}$ *are called* atoms*), the boolean connectives* $\neg$ *and* $\wedge$ *and the modal operators* $@_i$ *(called* satisfaction operators*), for each nominal* i*,* $\Diamond$ *and* $\Diamond^+$*. The formulas are defined as follows:*

$$\varphi ::= p \mid i \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond\varphi \mid \Diamond^+\varphi \mid @_i\varphi$$

Again, we freely use the standard abbreviations $\vee$, $\rightarrow$, $\leftrightarrow$, $\bot$, $\Box\varphi$, $\Box^+\varphi$, $\Diamond^*\varphi$ and $\Box^*\varphi$.

The definition of a *frame* is the same as the one from section A.2. The definition of a *model* is slightly different.

Definition A.34. *A* model *for the hybrid graph logic is a pair* $\mathcal{M} = (\mathcal{F}, \mathbf{V})$*, where* $\mathcal{F}$ *is a frame and* $\mathbf{V}$ *is a valuation function mapping proposition symbols into subsets of* W*, i.e.,* $\mathbf{V} : \Phi \mapsto \mathcal{P}(W)$*, and mapping nominals into singleton subsets of* W*, i.e, if* i *is a nominal then* $\mathbf{V}(i) = \{v\}$ *for some* $v \in W$*. We call this unique vertex that belongs to* $\mathbf{V}(i)$ *the* denotation *of* i *under* $\mathbf{V}$*. We can also say that* i denotes *or* names *the single vertex belonging to* $\mathbf{V}(i)$*.*

Definition A.35. *The notion of satisfaction is defined adding two extra clauses to definition A.4:*

1. $\mathcal{M}, v \Vdash i$ *iff* $v \in \mathbf{V}(i)$*;*

2. $\mathcal{M}, v \Vdash @_i\varphi$ *iff* $\mathcal{M}, d_i \Vdash \varphi$*, where* d_i *is the denotation of* i *under* $\mathbf{V}$*.*

For each nominal i, the formula $@_i\varphi$ means that if $\mathbf{V}(i) = \{v\}$ then φ is satisfied at v. As in section A.2, if $\mathcal{M}, v \Vdash \varphi$ for every vertex v, we say that φ is *globally satisfied* in the model $\mathcal{M}$ ($\mathcal{M} \Vdash \varphi$) and if φ is globally satisfied in all models $\mathcal{M}$ of a frame $\mathcal{F}$, we say that φ is *valid* in $\mathcal{F}$ ($\mathcal{F} \Vdash \varphi$).

Theorem A.36. 1. $\mathcal{M}, v \Vdash \neg@_i\phi$ *iff* $\mathcal{M}, v \Vdash @_i\neg\phi$*;*

2. $\mathcal{M}, v \Vdash @_i(\phi_1 \wedge \phi_2)$ *iff* $\mathcal{M}, v \Vdash @_i\phi_1 \wedge @_i\phi_2$*;*

3. $\mathcal{M}, v \Vdash @_i(\phi_1 \vee \phi_2)$ *iff* $\mathcal{M}, v \Vdash @_i\phi_1 \vee @_i\phi_2$;

4. $\mathcal{M}, v \Vdash @_i(\phi_1 \rightarrow \phi_2)$ *iff* $\mathcal{M}, v \Vdash @_i\phi_1 \rightarrow @_i\phi_2$.

Proof. 1. $\mathcal{M}, v \Vdash \neg@_i\phi$ iff $\mathcal{M}, v \nVdash @_i\phi$ iff $\mathcal{M}, d_i \nVdash \phi$, where d_i is the denotation of i under the valuation of $\mathcal{M}$, iff $\mathcal{M}, d_i \Vdash \neg\phi$ iff $\mathcal{M}, v \Vdash @_i\neg\phi$;

2. $\mathcal{M}, v \Vdash @_i(\phi_1 \wedge \phi_2)$ iff $\mathcal{M}, d_i \Vdash \phi_1 \wedge \phi_2$, where d_i is the denotation of i under the valuation of $\mathcal{M}$, iff $\mathcal{M}, d_i \Vdash \phi_1$ and $\mathcal{M}, d_i \Vdash \phi_2$ iff $\mathcal{M}, v \Vdash @_i\phi_1$ and $\mathcal{M}, v \Vdash @_i\phi_2$ iff $\mathcal{M}, v \Vdash @_i\phi_1 \wedge @_i\phi_2$;

3. It follows directly from the previous items;

4. It follows directly from the previous items.

□

Theorem A.37 ([31])**.** *The satisfiability problem and the validity problem for the hybrid graph logic are EXPTIME-Complete in the length of the formula.*

Theorem A.38 ([17])**.** *The model-checking problem for the hybrid graph logic is PTIME (linear) in the product of the size of the model and the length of the formula.*

The upper bound for the complexity of the frame-checking problem is a little different in the case of a hybrid logic, because of the special restriction on the valuation of nominals. For hybrid logics, the upper bound has the form

$$FC = O(2^{|p|\times n} \times n^{|i|} \times MC), \tag{A.1}$$

where $|p|$ is the number of distinct proposition symbols that occur in the given formula ϕ, $|i|$ is the number of distinct nominals that occur in ϕ and n is the number of vertices in $\mathcal{F}$. We need to apply the model-checking algorithm to every model $\mathcal{M}$ of the given frame $\mathcal{F}$. Every proposition symbol p that appears in ϕ may receive 2^n possible valuations $\mathbf{V}(p)$, while every nominal i may only receive n possible valuations $\mathbf{V}(i)$.

Theorem A.39. *The frame-checking problem for the hybrid graph logic is PTIME (linear) in the length of the formula and EXPTIME in the size of the frame, in the number of distinct proposition symbols that occur in the formula and in the number of distinct nominals that occur in the formula.*

Proof. This result follows directly from the discussion above. □

We can see then that the hybrid graph logic is indeed a very interesting choice, since we get a greater expressive power without any relevant increase in computational complexity.

A.4.2 Hybrid Graph Logic Definability

In the *hybrid graph logic* we can now express at least two of the properties that we want.

Theorem A.40. *Let G be a graph, G' be its underlying undirected graph, $\mathcal{F}$ be the frame that represents G and $\mathcal{F}'$ be the frame that represents G'. G is* strongly connected *if and only if $\mathcal{F} \Vdash \phi$ and* (weakly) connected *if and only if $\mathcal{F}' \Vdash \phi$, where ϕ is the formula*

$$\phi = @_i(\neg j \rightarrow \Diamond^+ j).$$

Proof. We prove the theorem only for strong connectivity. The other case is completely analogous.

($\Leftarrow$) Suppose that $\mathcal{F} \Vdash \phi$ but G is not strongly connected. Then, there are at least two distinct vertices v, w in G such that w is not reachable from v. We will evaluate ϕ in a model with a valuation $\mathbf{V}$ such that $\mathbf{V}(i) = \{v\}$ and $\mathbf{V}(j) = \{w\}$. Then, for any vertex u in G, $(\mathcal{F}, \mathbf{V}), u \nVdash \phi$, contradicting the fact that ϕ is valid in $\mathcal{F}$.

($\Rightarrow$) Suppose that G is strongly connected but $\mathcal{F} \nVdash \phi$. Then, there is a valuation $\mathbf{V}$ and a vertex u such that $(\mathcal{F}, \mathbf{V}), u \nVdash \phi$, which is equivalent, by theorem A.36, to $(\mathcal{F}, \mathbf{V}), u \Vdash \theta$, where $\theta = @_i\neg j \wedge @_i\neg\Diamond^+ j$. Let $\mathbf{V}(i) = \{v\}$ and $\mathbf{V}(j) = \{w\}$. If $v = w$, then θ is not satisfied, so we may assume that $v \neq w$. Then, for θ to be satisfied, we need $(\mathcal{F}, \mathbf{V}), u \Vdash @_i\neg\Diamond^+ j$. This, on the other hand, is equivalent to vR^+w being false in G, which means that w is not reachable from v. This contradicts the fact that G is strongly connected. □

We now want to determine how complex it is to test whether a graph is connected using the above formula ϕ. This consist of, given a frame $\mathcal{F}$ that represents the graph, frame-check whether $\mathcal{F} \Vdash \phi$. We should notice first that there are no

proposition symbols in ϕ and only two distinct nominals. Finally, the length of ϕ is constant and does not depend on the size of the graph. Let CON be the complexity of testing whether a graph is connected through a frame-checking of ϕ. Then, taking into account these observations and the formula in equation (A.1), we have that

$$CON = O(n^2 \times MC),$$

where, for the formula ϕ, MC is PTIME (in fact linear) in the size of the graph.

Theorem A.41. *The complexity to check whether a graph is connected using the above formula ϕ is PTIME (cubic) in the size of the graph.*

Proof. This result follows directly from the discussion above. □

Theorem A.42. *A graph G contains a closed path if and only if it contains a cycle.*

Proof. The right to left direction is immediate, as cycles are a particular case of closed paths. For the left to right direction, we proceed by induction on the length $n \geq 2$ of the closed path. If G contains a closed path of length $n = 2$, then this closed path is also a cycle. Now suppose that, for any $n < k$, if G contains a closed path of length n, then it contains a cycle. If G contains a closed path $\langle v_1, \ldots, v_k \rangle$ of length k that is not a cycle, then there are i and j such that $1 \leq i < j < k$ and $v_i = v_j$. But then, $\langle v_i, \ldots, v_j \rangle$ is a closed path of length smaller than k. Then, by the induction hypothesis, G contains a cycle. □

Theorem A.43. *A graph G with frame $\mathcal{F}$ is* acyclic *if and only if $\mathcal{F} \Vdash \phi$, where ϕ is the formula*

$$\phi = @_i \neg \Diamond^+ i.$$

Proof. ($\Leftarrow$) Suppose that $\mathcal{F} \Vdash \phi$ but G is not acyclic. Then, there is at least one vertex v in G such that there is a path in G from v to itself. We will evaluate ϕ in a model with a valuation $\mathbf{V}$ such that $\mathbf{V}(i) = \{v\}$. Then, for any vertex u in G, $(\mathcal{F}, \mathbf{V}), u \nVdash \phi$, contradicting the fact that ϕ is valid in $\mathcal{F}$.

($\Rightarrow$) Suppose that G is acyclic but $\mathcal{F} \nVdash \phi$. Then, there is a valuation $\mathbf{V}$ and a vertex u such that $(\mathcal{F}, \mathbf{V}), u \nVdash \phi$, which is equivalent, by theorem A.36, to $(\mathcal{F}, \mathbf{V}), u \Vdash \theta$, where $\theta = @_i \Diamond^+ i$. Let $\mathbf{V}(i) = \{v\}$. Then this, on the other hand, is equivalent to vR^+v being true in G, which means that v is reachable from itself. This implies

that G contains a closed path, which, by theorem A.42, contradicts the fact that G is acyclic. □

Let us now determine how complex it is to test whether a graph is acyclic using the above formula ϕ. Again, there are no proposition symbols in ϕ and, this time, ϕ has only one nominal. Also, the length of ϕ is, again, constant and does not depend on the size of the graph. Let ACY be the complexity of testing whether a graph is acyclic through a frame-checking of ϕ. Then, taking into account these observations and the formula in equation (A.1), we have that

$$ACY = O(n \times MC),$$

where, for the formula ϕ, MC is PTIME (in fact linear) in the size of the graph.

Theorem A.44. *The complexity to check whether a graph is acyclic using the above formula ϕ is PTIME (quadratic) in the size of the graph.*

Proof. This result follows directly from the discussion above. □

We consider the complexities in theorems A.41 and A.44 to be satisfactory. It is not necessary to search for alternative forms to express these properties in this logic or in another logic.

Before trying to find a formula to describe the Hamiltonian graphs, we need to consider some graph-theoretical issues. In graph theory [1], there is no known result that states a necessary and sufficient condition for a graph to be Hamiltonian. If we could find a formula that describes the Hamiltonian graphs without having to describe the Hamiltonian cycle itself, we would be finding such necessary and sufficient condition. Thus, what our formula does is to inspect all of the paths in the graph, searching for a Hamiltonian cycle. Not surprisingly then, the only formula we could find in this simple language to describe the Hamiltonian property has length proportional to $n!$, where n is the number of vertices in the graph.

Let $\mathcal{L}_n = \{i_1, \ldots, i_n\}$ be a set containing n nominals. Before defining a formula for the Hamiltonian property, we will define a formula that is globally satisfied in a model under a valuation $\mathbf{V}$ if and only if $\mathbf{V}(i_k) \neq \mathbf{V}(i_l)$, for all $i_k, i_l \in \mathcal{L}_n$ such that $k \neq l$.

Lemma A.45. *A valuation satisfies* $\mathbf{V}(i_k) \neq \mathbf{V}(i_l)$*, for all* $i_k, i_l \in \mathcal{L}_n$ *such that* $k \neq l$*, if and only if* $(\mathcal{F}, \mathbf{V}) \Vdash \psi_n$*, where* ψ_n *is the formula*

$$\psi_n = \bigwedge_{1 \leq k < n} \left(@_{i_k} \bigwedge_{k < l \leq n} \neg i_l \right).$$

Proof. It follows directly from the definitions of a valuation for a nominal and of satisfaction for a nominal and for a formula $@_i\varphi$. □

Definition A.46. *Let* $\mathcal{L}_n = \{i_1, \ldots, i_n\}$ *be a set of* n *nominals. Then, we define the following formulas with these nominals:*

$$\Delta_n = i_n \wedge \Diamond i_1;$$

$$\Delta_k = i_k \wedge \Diamond \Delta_{k+1}, \text{ for } 1 \leq k \leq n-1.$$

We now define a set F of permutations of the nominals in $\mathcal{L}_n$. This set has $n!$ elements. We represent a permutation as a bijective function $\sigma : \{1, \ldots, n\} \mapsto \mathcal{L}_n$.

Theorem A.47. *A connected graph* G *(with* n *vertices) with frame* $\mathcal{F}$ *is* Hamiltonian *if and only if* $\mathcal{F} \Vdash \phi$*, where* ϕ *is the formula*

$$\phi = \psi_n \rightarrow \delta_n,$$

where ψ_n *is the formula from lemma A.45 and*

$$\delta_n = \bigvee_{\sigma \in F} (\sigma(1) \wedge \Diamond(\sigma(2) \wedge \Diamond(\sigma(3) \ldots (\sigma(n-1) \wedge \Diamond(\sigma(n) \wedge \Diamond\sigma(1)) \ldots).$$

Proof. ($\Leftarrow$) Suppose that the formula ϕ is valid in $\mathcal{F}$. We will evaluate ϕ in an arbitrary vertex v of a model with a valuation $\mathbf{V}$ such that $\mathbf{V}$ satisfies ψ_n and $\mathbf{V}(i_1) = \{v\}$. First, this means that each nominal is denoting a different vertex. Second, $\mathbf{V}$ must also satisfy δ_n. If δ_n is satisfied, at least one of the members in its disjunction is satisfied. Let σ' be the permutation correspondent to this member. To simplify the notation and without loss of generality, we consider that $\sigma'(k) = i_k$. Thus, $(\mathcal{F}, \mathbf{V}), v \Vdash \Delta_1$. From this and from the construction rule of the formulas Δ_k, we have that there are vertices w_k in G such that $(\mathcal{F}, \mathbf{V}), w_k \Vdash \Delta_k$, $w_k R w_{k+1}$, for $2 \leq k \leq n-1$, vRw_2 and $w_n Rv$. We then have that $\langle v, w_2, \ldots, w_n, v\rangle$ is a Hamiltonian cycle in G.

($\Rightarrow$) Suppose that there is a Hamiltonian cycle $\langle v_1, \dots v_n, v_1 \rangle$ in G. We denote the vertices with nominals in such a way that $\mathcal{L}_n = \{i_1, \dots, i_n\}$ and i_k denotes v_k. This valuation satisfies ψ_n. We have that $v_n R v_1$, so Δ_n is satisfied at v_n. Similarly, Δ_k is satisfied at v_k. Since Δ_1 is a member of the disjunction in δ_n, δ_n is satisfied at v_1. Repeating the previous line of thought, but starting the cycle at v_2, v_3 and so on, we can see that δ_n is also satisfied at all the vertices in the cycle. Since the cycle is Hamiltonian, this means that δ_n is satisfied in all the vertices of G. Since ϕ is trivially satisfied in all the valuations that do not satisfy ψ_n, we only need to think about the ones that do. If we change the valuation of the nominals in $\mathcal{L}_n$ to another one that satisfies ψ_n, this is equivalent to applying a permutation to the nominals. As δ_n contains a member in its disjunction for each permutation, we conclude that in fact ϕ is valid in $\mathcal{F}$. $\square$

In this case, the number of distinct nominals in the formula and the length of the formula are both linked to the size of the graph. Even though the frame-checking complexity is PTIME in the length of the formula, since the formula has a factorial length in the size of the graph, it is infeasible to frame-check this formula. In the case of the Hamiltonian property, we must search for alternative forms to express it using other logics. Two attempts to do this are presented in the next two sections.

The difficulty in finding a formula to describe Eulerian graphs is of a completely different nature. Here, the limitation is on the language, not on the theoretical definition of the property. There is a known result that states a necessary and sufficient condition for a graph to be Eulerian (theorem A.31), so the argument used above for Hamiltonian graphs is not valid here. However, the hybrid graph language does not have the expressive power, at least not without performing a brute-force search through every possible path in the graph, in a similar way as the formula in theorem A.47 does, to state cardinality conditions on edges incident from and to a vertex, as is needed in theorem A.31.

The other way to describe the Eulerian property would be to find a formula that explicitly describes an Eulerian path in the graph. However, it is very hard to find such a formula, since the hybrid graph logic and many other modal logics are not good to talk about edges. One of the reasons for that is the fact that the modal operator $\Diamond$ does not differentiate between edges incident from a vertex. We now,

using nominals, have names for vertices, but we still cannot keep track of which edges we are using when we walk in a graph. This suggests that a possible solution would be to find a way to name the edges in some similar way to the use of nominals to name vertices. We do this in section A.7, where we describe a method to name edges within the framework of a hybrid logic and use it to find a formula for the Eulerian property.

A.5 The Temporal Logic Hybrid CTL*

The fact that the formula previously introduced to describe Hamiltonian graphs has factorial size makes its verification infeasible. Obviously, we can never expect to verify the Hamiltonian property in polynomial time, since determining whether a graph is Hamiltonian is an NP-Complete problem [33], but we certainly wish to verify it with a lower complexity than factorial time. This is our goal in this section and in the next one.

In our first attempt to write a short (with polynomial length) formula to describe the class of Hamiltonian graphs, we use the temporal branching-time logic CTL* [34] with nominals (*hybrid CTL**). We could use the full hybrid μ-calculus presented in [35], since it contains the hybrid CTL* [36, 35]. But we will not do this, since the hybrid CTL* is strong enough for what we want to do and its formulas are easier to read and to understand than hybrid μ-calculus formulas.

A.5.1 Language

Definition A.48. *The hybrid CTL* language is a temporal language consisting of a set* Φ *of countably many proposition symbols (the elements of* Φ *are denoted by* $p_1, p_2, \ldots$*), a set* $\mathcal{L}$ *of countably many nominals (the elements of* $\mathcal{L}$ *are denoted by* $i_1, i_2, \ldots$*) such that* $\Phi \cap \mathcal{L} = \emptyset$*, the boolean connectives* $\neg$ *and* $\wedge$ *and the operators* $@_i$*, for each nominal* i*, and* $\mathbf{A}$*,* $\mathbf{E}$*,* $\mathbf{X}$*,* $\mathbf{F}$*,* $\mathbf{G}$ *and* $\mathbf{U}$*. Formulas are divided into* vertex formulas $\mathbf{S}$ *and* path formulas $\mathbf{P}$ *defined by the following mutual induction:*

$$\mathbf{S} ::= p \mid i \mid \top \mid \neg\mathbf{S} \mid \mathbf{S_1} \wedge \mathbf{S_2} \mid \mathbf{AP} \mid \mathbf{EP} \mid @_i\mathbf{S}$$

$$\mathbf{P} ::= \mathbf{S} \mid \neg\mathbf{P} \mid \mathbf{P_1} \wedge \mathbf{P_2} \mid \mathbf{XP} \mid \mathbf{FP} \mid \mathbf{GP} \mid \mathbf{P_1 U P_2}$$

The language of hybrid CTL* is then the set of all vertex formulas *generated by the above rules.*

The definition of a *frame* and of a *model* are the same as the ones from the previous section. The notion of satisfaction in hybrid CTL* is defined as follows:

Definition A.49. *Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. We also need the auxiliary notation that if $\pi = \langle s_0, s_1, \ldots \rangle$ is a path, we denote by π^i the suffix of π starting at s_i. The notion of* satisfaction *of a vertex formula S in a model $\mathcal{M}$ at a vertex v or of a path formula P in a model $\mathcal{M}$ at a path π, notation $\mathcal{M}, v \Vdash S$ and $\mathcal{M}, \pi \Vdash P$, can be inductively defined as follows:*

1. *$\mathcal{M}, v \Vdash p$ iff $v \in \mathbf{V}(p)$;*
2. *$\mathcal{M}, v \Vdash i$ iff $v \in \mathbf{V}(i)$;*
3. *$\mathcal{M}, v \Vdash \top$ always;*
4. *$\mathcal{M}, v \Vdash \neg S$ iff $\mathcal{M}, v \nVdash S$;*
5. *$\mathcal{M}, v \Vdash S_1 \wedge S_2$ iff $\mathcal{M}, v \Vdash S_1$ and $\mathcal{M}, v \Vdash S_2$;*
6. *$\mathcal{M}, v \Vdash \mathbf{A}P$ iff for every path π starting in v, $\mathcal{M}, \pi \Vdash P$;*
7. *$\mathcal{M}, v \Vdash \mathbf{E}P$ iff there is a path π starting in v such that $\mathcal{M}, \pi \Vdash P$;*
8. *$\mathcal{M}, v \Vdash @_i S$ iff $\mathcal{M}, d_i \Vdash S$, where d_i is the denotation of i under $\mathbf{V}$;*
9. *$\mathcal{M}, \pi \Vdash S$ iff v is the first vertex of π and $\mathcal{M}, v \Vdash S$;*
10. *$\mathcal{M}, \pi \Vdash \neg P$ iff $\mathcal{M}, \pi \nVdash P$;*
11. *$\mathcal{M}, \pi \Vdash P_1 \wedge P_2$ iff $\mathcal{M}, \pi \Vdash P_1$ and $\mathcal{M}, \pi \Vdash P_2$;*
12. *$\mathcal{M}, \pi \Vdash \mathbf{X}P$ iff $\mathcal{M}, \pi^1 \Vdash P$;*
13. *$\mathcal{M}, \pi \Vdash \mathbf{F}P$ iff there is a $k \geq 0$ such that $\mathcal{M}, \pi^k \Vdash P$;*
14. *$\mathcal{M}, \pi \Vdash \mathbf{G}P$ iff for all $k \geq 0$, $\mathcal{M}, \pi^k \Vdash P$;*
15. *$\mathcal{M}, \pi \Vdash P_1 \mathbf{U} P_2$ iff there is a $k \geq 0$ such that $\mathcal{M}, \pi^k \Vdash P_2$ and for all $0 \leq j < k$, $\mathcal{M}, \pi^j \Vdash P_1$.*

We should think of **A** as "for all paths starting in the current vertex", **E** as "there is a path starting in the current vertex such that...", **X** as "in the next vertex of the current path", **F** as "in the current vertex or at some future vertex in the current path", **G** as "in the current vertex and for all future vertices in the current path" and **U** as "the first formula is satisfied in a path until the second formula is satisfied in this path, and the second formula will be satisfied eventually".

Lemma A.50 ([20])**.** *The satisfiability problem and the validity problem for CTL* are 2-EXPTIME-Complete in the length of the formula.*

Lemma A.51 ([36])**.** *Every CTL* formula can be translated into a μ-calculus formula. The complexity of this translation is 2-EXPTIME in the length of the original formula. The translated formula can be written with exponential length with respect to the length of the original formula.*

Theorem A.52. *Every hybrid CTL* formula can be translated into a hybrid μ-calculus formula.*

Proof. The only formulas that are in hybrid CTL*, but are not in CTL* are the formulas that contain nominals or satisfaction operators. We can easily extend the translation of lemma A.51 to also cover these cases, since nominals and satisfaction operators are present in the hybrid μ-calculus [35]. First, nominals are translated in the same way as proposition symbols: $tr(i) = i$. Second, we translate formulas of the form $@_i\psi$ in the obvious way: $tr(@_i\psi) = @_i tr(\psi)$. □

Theorem A.53 ([35])**.** *The satisfiability problem and the validity problem for the hybrid μ-calculus are EXPTIME-Complete in the length of the formula.*

Theorem A.54. *The satisfiability problem and the validity problem for hybrid CTL* are 2-EXPTIME-Hard in the length of the formula and are decidable.*

Proof. The 2-EXPTIME-hardness follows from lemma A.50 and the decidability follows from theorems A.52 and A.53. □

Theorem A.55. *The model-checking problem for the hybrid CTL* is PTIME (linear) in the size of the model and EXPTIME in the length of the formula.*

Proof. This follows from an elementary combination of the CTL* model-checking algorithm presented in [11], which is PTIME in the size of the model and EXPTIME in the length of the formula, with the basic hybrid logic model-checking algorithm presented in [17], which is PTIME both in the size of the model and in the length of the formula. □

The upper bound for the complexity of the frame-checking problem is the same as in the previous section:

$$FC = O(2^{|p|\times n} \times n^{|i|} \times MC), \tag{A.2}$$

where $|p|$ is the number of distinct proposition symbols that occur in the given formula ϕ, $|i|$ is the number of distinct nominals that occur in ϕ and n is the number of vertices in $\mathcal{F}$.

Theorem A.56. *The frame-checking problem for the hybrid CTL* is EXPTIME in the length of the formula, in the size of the frame, in the number of distinct proposition symbols that occur in the formula and in the number of distinct nominals that occur in the formula.*

Proof. This result follows directly from the discussion above. □

A.5.2 The Hamiltonian Property

Let G be a graph with n vertices and let $\mathcal{L}_n = \{i_1, \ldots, i_n\}$. Let us add a loop to all the vertices in G. We can then define the formula that is valid if and only if G is Hamiltonian.

Theorem A.57. *A connected graph G (with n vertices) with frame $\mathcal{F}$ is* Hamiltonian *if and only if $\mathcal{F} \Vdash \phi$, where ϕ is the formula*

$$\phi = \psi_n \rightarrow \varepsilon_n,$$

where ψ_n is the formula from lemma A.45 and

$$\varepsilon_n = @_{i_1}\mathbf{E}[\mathbf{XF}i_1 \wedge \mathbf{F}i_2 \wedge \ldots \wedge \mathbf{F}i_n \wedge$$

$$\wedge \mathbf{XG}(i_1 \rightarrow \mathbf{G}i_1) \wedge \mathbf{G}(i_2 \rightarrow \mathbf{XG}\neg i_2) \wedge \ldots \wedge \mathbf{G}(i_n \rightarrow \mathbf{XG}\neg i_n)].$$

Proof. ($\Leftarrow$) Suppose that the formula ϕ is valid in $\mathcal{F}$. We will evaluate ϕ in an arbitrary vertex v of a model with a valuation $\mathbf{V}$ such that $\mathbf{V}$ satisfies ψ_n. First, this means that each nominal is denoting a different vertex. Second, $\mathbf{V}$ must also satisfy ε_n. If ε_n is satisfied, then there is a path π in G starting at the vertex denoted by i_1 such that the formula inside the brackets in ε_n is satisfied in this path for the valuation $\mathbf{V}$. This means that $\mathbf{F}i_k$ is satisfied for $2 \leq k \leq n$ and $\mathbf{XF}i_1$ is satisfied. Thus, every vertex in G appears at least once in π. Also, the formulas $\mathbf{G}(i_k \rightarrow \mathbf{XG}\neg i_k)$ are satisfied for $2 \leq k \leq n$. This means that the vertices denoted by i_k, for $2 \leq k \leq n$, appear exactly once in the path. Finally, $\mathbf{XG}(i_1 \rightarrow \mathbf{G}i_1)$ is satisfied, which means that after the second visit to the vertex denoted by i_1, no other vertex in G is visited anymore in the path. So, if we disregard the final looping in the vertex denoted by i_1, we have a path that starts and ends in this vertex and visit every other vertex of G exactly once. This is exactly a Hamiltonian cycle.

($\Rightarrow$) Suppose that there is a Hamiltonian cycle $\langle v_1, \ldots v_n, v_1\rangle$ in G. We denote the vertices with nominals in such a way that $\mathcal{L}_n = \{i_1, \ldots, i_n\}$ and i_k denotes v_k. This valuation satisfies ψ_n. Consider the extended path $\langle v_1, \ldots v_n, v_1,\ v_1, \ldots\rangle$. Clearly, the components of the conjunction inside brackets in ε_n are satisfied in this path. Then, since this path starts at v_1, $\mathbf{E}[\ldots]$ is satisfied in v_1 and ε_n is satisfied at all vertices, because v_1 is denoted by i_1. Since ϕ is trivially satisfied in all the valuations that do not satisfy ψ_n, we only need to think about the ones that do. Changing the valuation of the nominals in $\mathcal{L}_n$ to another one that satisfies ψ_n is harmless, since the @ operator in the beginning of the formula ε_n is marking a starting point in the cycle, which contains all the vertices. This means that no matter where $\mathbf{V}(i_1)$ send us, it will be a point inside the cycle. Thus, ϕ is valid in $\mathcal{F}$. □

Let us now determine how complex it is to test whether a graph is Hamiltonian using the above formula ϕ. First of all, we now have a formula with polynomial (quadratic) length in the size of the graph. Also, there are no proposition symbols. From the structure of the formula, it is not difficult to see that it would be a waste of time to check valuations that do not assign distinct vertices to each nominal, since ψ_n is a precondition of an implication. Besides that, the formula ε_n is completely symmetric with respect to the nominals $i_2, \ldots, i_n$. Thus, for each possible valuation of i_1, we only need to check one arbitrary valuation for $i_2, \ldots, i_n$ such that each

of these nominals names a distinct vertex. Let HAM be the complexity of testing whether a graph is Hamiltonian through a frame-checking of ϕ. Then, taking into account the above observations and the formula in equation (A.2), we have that

$$HAM = O(n \times MC),$$

where, for the formula ϕ, MC is EXPTIME in the size of the graph.

Theorem A.58. *The complexity to check whether a graph is Hamiltonian using the above formula ϕ is EXPTIME in the size of the graph.*

Proof. This result follows directly from the discussion above. □

A.6 Hybrid Graph Logic with the ↓ binder

In the previous section, we were able to reduce the upper-bound of the complexity of testing whether a graph is Hamiltonian using a frame-checking method from a factorial upper-bound ($O(k!)$, where k is the size of the frame) to an exponential upper-bound ($O(2^k)$). The key point involved in this reduction was the fact that in the second logic, we were able to describe the Hamiltonian property with a formula that has polynomial length in the size of the graph, instead of factorial length as in the first logic.

In this section, we describe a third logic, which is a slight extension of the hybrid graph logic. We then use it to build a formula that expresses the Hamiltonian property. With this formula, we are able to reduce even further the complexity of testing whether a graph is Hamiltonian using a frame-checking method. This happens because we are able, for this formula, to reduce the frame-checking problem to a particular case of the model-checking problem with lower complexity than the general case.

A.6.1 Language

Definition A.59. *The language of the* hybrid graph logic with the ↓ binder *is a hybrid language consisting of a set Φ of countably many proposition symbols (the elements of Φ are denoted by $p_1, p_2, \ldots$), a set $\mathcal{L}$ of countably many nominals (the*

elements of $\mathcal{L}$ are denoted by $i_1, i_2, \ldots$), a set $\mathcal{S}$ of countably many state-variables (the elements of $\mathcal{S}$ are denoted by $x_1, x_2, \ldots$), such that Φ, $\mathcal{L}$ and $\mathcal{S}$ are pairwise disjoint (the elements of $\Phi \cup \mathcal{L} \cup \mathcal{S}$ are called atoms*), the boolean connectives $\neg$ and $\wedge$ and the modal operators $@_i$, for each nominal i, $@_x$, for each state-variable x, $\Diamond$, $\Diamond^+$ and $\downarrow$. The formulas are defined as follows:*

$$\varphi ::= p \mid i \mid x \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond\varphi \mid \Diamond^+\varphi \mid @_i\varphi \mid @_x\varphi \mid \downarrow x.\varphi$$

Again, we freely use the standard abbreviations $\vee$, $\rightarrow$, $\leftrightarrow$, $\bot$, $\Box\varphi$, $\Box^+\varphi$, $\Diamond^*\varphi$ and $\Box^*\varphi$.

The definition of a *frame* and of a *model* are the same as the ones from section A.4

In order to deal with the state-variables, we need to introduce the notion of *assignments*.

Definition A.60. *An* assignment *is a function g that maps state-variables to vertices of the model, i.e., $g : \mathcal{S} \mapsto W$. We use the notation $g' = g[v_1/x_1, \ldots, v_n/\ x_n]$ to denote an assignment such that $g'(x) = g(x)$ if $x \notin \{x_1, \ldots, x_n\}$ and $g'(x_i) = v_i$, otherwise.*

The semantical notion of satisfaction is defined as follows:

Definition A.61. *The notion of satisfaction of a formula ϕ in a model $\mathcal{M}$ at a vertex v with an assignment g, notation $\mathcal{M}, g, v \Vdash \phi$, is inductively defined adding the assignment g to all the clauses in definitions A.4 and A.35 and the following three extra clauses:*

1. *$\mathcal{M}, g, v \Vdash x$ iff $g(x) = v$;*
2. *$\mathcal{M}, g, v \Vdash @_x\phi$ iff $\mathcal{M}, g, d \Vdash \phi$, where $d = g(x)$;*
3. *$\mathcal{M}, g, v \Vdash \downarrow x.\phi$ iff $\mathcal{M}, g[v/x], v \Vdash \phi$.*

The formula $\downarrow x.\phi$ means that, using x as a name for the present vertex (state-variables can be thought as "on-the-fly nominals"), ϕ is satisfied. The $\downarrow$ operator is the only operator that binds a variable. Free and bound variables are defined in the usual way. The only case worth mentioning is that in the formula $@_x\psi$, x is free.

A *sentence* is a formula with no free variables. We only consider formulas that are sentences, because we do not want to include the assignments in the model-checking and frame-checking problems.

The $\downarrow$ binder, just as the satisfaction operators, is dual to itself: $\mathcal{M}, g, v \Vdash \neg \downarrow x.\phi$ iff $\mathcal{M}, g, v \nVdash \downarrow x.\phi$ iff $\mathcal{M}, g[v/x], v \nVdash \phi$ iff $\mathcal{M}, g[v/x], v \Vdash \neg\phi$ iff $\mathcal{M}, g, v \Vdash \downarrow x.\neg\phi$.

Theorem A.62 ([16])**.** *The satisfiability problem and the validity problem for the hybrid graph logic with the $\downarrow$ binder are* undecidable*.*

This result shows that the inclusion of state-variables and the $\downarrow$ operator turns a logic that had the same computational complexity of the basic graph logic into an undecidable logic.

Theorem A.63 ([17])**.** *The model-checking problem for the hybrid graph logic with the $\downarrow$ binder is PSPACE-Complete both in the size of the model and in the length of the formula.*

In [19], it is shown that, for a family of hybrid logics with the $\downarrow$ binder, there are fragments of these logics in which the complexities of the satisfiability problem and the model-checking problem are lower than in the full logics. One of these fragments, which is defined using the notion of formulas in negation normal form, turns out to be also a fragment of the hybrid graph logic with the $\downarrow$ binder. According to [19], the complexity of the model-checking problem in this fragment is lower than the one stated above for the full hybrid graph logic with the $\downarrow$ binder. This result will be central to our discussion in this section.

Definition A.64. *A formula of the hybrid graph logic with the $\downarrow$ binder is in* negation normal form *(NNF) if the negation symbol ($\neg$) appears only in front of proposition symbols, nominals and state-variables.*

Lemma A.65. *If we consider the dual operators of $\top, \wedge, \Diamond$ and $\Diamond^+$, i.e., $\bot, \vee, \Box$ and $\Box^+$ as primitive operators of the language, then each formula of the hybrid graph logic with the $\downarrow$ binder is semantically equivalent to a formula in NNF.*

Proof. Let $(\triangle, \triangledown)$ be one of the following pairs of dual operators: $(\wedge, \vee)$, $(\Diamond, \Box)$, $(\Diamond^+, \Box^+)$, $(\downarrow, \downarrow)$ and $(@_z, @_z)$, where z is either a nominal or a state-variable. Using the semantic equivalences $\neg\top \equiv \bot$, $\neg\bot \equiv \top$, $\neg\neg\phi \equiv \phi$ and $\neg \triangle \phi \equiv \triangledown\neg\phi$, we

can push the negation symbols inward until they appear only in front of proposition symbols, nominals and state-variables. □

Theorem A.66 ([19])**.** *The model-checking problem for a formula in the hybrid graph logic with the ↓ binder that, when put in NNF, does not have any occurrence of □, $\Diamond^+$ or $\Box^+$ is NP-Complete both in the size of the model and in the length of the formula.*

The upper bound for the complexity of the frame-checking problem is again

$$FC = O(2^{|p|\times n} \times n^{|i|} \times MC), \tag{A.3}$$

where $|p|$ is the number of distinct proposition symbols that occur in the given formula ϕ, $|i|$ is the number of distinct nominals that occur in ϕ and n is the number of vertices in $\mathcal{F}$. It should be noticed that, if the model-checking problem can be solved in polynomial space, then the frame-checking problem can also be solved in polynomial space, since the frame-checking of a formula in a frame $\mathcal{F}$ is done through a finite series of completely independent model-checkings of that formula, one for each model $\mathcal{M}$ of $\mathcal{F}$.

Theorem A.67. *The frame-checking problem for the hybrid graph logic with the ↓ binder is PSPACE in the length of the formula and in the size of the frame.*

Proof. This result follows directly from theorem A.63 and the discussion above. □

A.6.2 The Hamiltonian Property

Let G be a graph with n vertices. We now use the language introduced in the previous subsection to define a formula that is valid if and only if G is Hamiltonian.

Theorem A.68. *A connected graph G (with n vertices) with frame $\mathcal{F}$ is* Hamiltonian *if and only if $\mathcal{F} \Vdash \phi$, where ϕ is the formula*

$$\phi = \downarrow x_1.\Diamond \downarrow x_2.(\neg x_1 \wedge \Diamond \downarrow x_3.(\bigwedge_{1\le k<3} \neg x_k \wedge \Diamond \downarrow x_4.(\ldots \wedge \downarrow x_{n-1}.(\bigwedge_{1\le k<n-1} \neg x_k \wedge \Diamond \downarrow x_n.$$

$$(\bigwedge_{1\le k<n} \neg x_k \wedge \Diamond x_1)\ldots).$$

Proof. ($\Leftarrow$) Suppose that the formula ϕ is valid in $\mathcal{F}$. We will evaluate ϕ in an arbitrary vertex v_1 of a model with an arbitrary valuation $\mathbf{V}$ and an arbitrary assignment g. If $\mathcal{M}, g, v_1 \Vdash \phi$, then $\mathcal{M}, g[v_1/x_1], v_1 \Vdash \Diamond \downarrow x_2.\neg x_1 \ldots$. This means that there is a vertex v_2 such that $v_1 R v_2$ and $\mathcal{M}, g[v_1/x_1], v_2 \Vdash \downarrow x_2.\neg x_1 \wedge \Diamond \downarrow x_3 \ldots$, which means that $\mathcal{M}, g[v_1/x_1, v_2/x_2], v_2 \Vdash \neg x_1 \wedge \Diamond \downarrow x_3 \ldots$. This implies that $v_2 \neq v_1$ and $\mathcal{M}, g[v_1/x_1, v_2/x_2], v_2 \Vdash \Diamond \downarrow x_3 \ldots$. If we keep repeating this, we conclude that there are n distinct vertices $v_1, \ldots, v_n$ such that $v_i R v_{i+1}$, $1 \leq i < n$ and $v_n R v_1$. We have a path that starts and ends in the vertex v_1 and visit every other vertex of G exactly once. This is exactly a Hamiltonian cycle.

($\Rightarrow$) Suppose that there is a Hamiltonian cycle $\langle v_1, \ldots v_n, v_1 \rangle$ in G. We have that $\mathcal{M}, g[v_1/x_1, \ldots, v_n/x_n], v_n \Vdash \bigwedge_{1 \leq k < n} \neg x_k \wedge \Diamond x_1$. This means that $\mathcal{M}, g[v_1/x_1, \ldots, v_{n-1}/x_{n-1}], v_n \Vdash \downarrow x_n. \bigwedge_{1 \leq k < n} \neg x_k \wedge \Diamond x_1$, which implies that $\mathcal{M}, g[v_1/x_1, \ldots, v_{n-1}/x_{n-1}], v_{n-1} \Vdash \bigwedge_{1 \leq k < n-1} \neg x_k \wedge \Diamond \downarrow x_n.(\bigwedge_{1 \leq k < n} \neg x_k \wedge \Diamond x_1)$. If we keep repeating this, we conclude that $\mathcal{M}, g, v_1 \Vdash \phi$, for an arbitrary assignment g. If we start the Hamiltonian cycle in another vertex, the same argument easily applies. Thus, ϕ is globally satisfied in $\mathcal{M}$. As the valuation in $\mathcal{M}$ is completely irrelevant, ϕ is valid in $\mathcal{F}$. □

Let us now determine how complex it is to test whether a graph is Hamiltonian using the above formula ϕ. First of all, we now have a formula that is a sentence with polynomial (quadratic) length in the size of the graph. Also, there are no proposition symbols and no nominals. This means that the valuation is completely irrelevant to the satisfaction of this sentence. Thus, the frame-checking problem is reduced to the model-checking problem for an arbitrary model of the frame. Let HAM be the complexity of testing whether a graph is Hamiltonian through a frame-checking of ϕ. Then, taking into account the above observations and the formula in equation (A.3), we have that

$$HAM = MC.$$

Now, we should notice that ϕ is already in NNF and it does not have any occurrence of $\Box$, $\Diamond^+$ or $\Box^+$. So, the reduction in the model-checking complexity stated in theorem A.66 applies, and the model-checking complexity for this formula is in NP.

Theorem A.69. *The complexity to check whether a graph is Hamiltonian using the above formula ϕ is NP in the size of the graph.*

Proof. This result follows directly from the discussion above. □

The above formula ϕ is an "optimal" formula to describe the Hamiltonian property, in the following sense: since the problem of testing whether a graph is Hamiltonian is NP-Complete [33] and the test that we developed using this formula is in NP, it is impossible to find any other formula, in this logic or in any other logic, that describes the Hamiltonian property and can be frame-checked with a lower complexity than ϕ (assuming that NP $\neq$ P).

A.7 Edge-Related Properties

As was mentioned in the end of section A.4, the best way to describe the Eulerian property would be to use theorem A.31. However, as was also mentioned in the end of that section, our standard $\Diamond$ operators do not have the expressive power, at least not without performing a brute-force search through every possible path in the graph, in a similar way as the formula in theorem A.47 does, to state cardinality conditions on edges incident from and to a vertex, as is needed in theorem A.31. This remains true for the temporal operators defined in section A.5. This happens because all of these operators are "existential" operators. All they can do is to differentiate between things like "there is some edge" and "there is no edge", or "there is some path" and "there is no path". We would need "counting" operators to be able to efficiently express theorem A.31. Modal logics with this sort of operator exist in the literature and are called *graded modal logics*, being first introduced in [37].

We present a graded modal logic in the end of this section and use it to express the Eulerian property with the help of theorem A.31. But first, we develop a method to express the Eulerian property in the hybrid logics already presented in the previous sections. This method could be useful to express other edge-related properties for which there is no theorem that states a necessary and sufficient condition for the property to hold, as does theorem A.31.

A.7.1 Graph Subdivisions

If we do not use theorem A.31, we need to define the Eulerian property with a formula that explicitly describes an Eulerian path in the graph. This is also a

difficult task, because of the reasons exposed in the end of section A.4. We need a way to identify particular edges, but hybrid logics only have names for vertices. So, we first develop a method to name edges *within* the formalism of a hybrid logic and later use it to define the Eulerian property.

Definition A.70. *Let $\langle v, w\rangle$ be an edge in a graph G. An* edge subdivision *consists of adding a new vertex u to G, deleting the edge $\langle v, w\rangle$ and adding the edges $\langle v, u\rangle$ and $\langle u, w\rangle$ to G. A* graph subdivision *of a graph G is a graph G' obtained from G by a finite number of edge subdivisions.*

Definition A.71. *Let G be a graph. We define $G' = \mathcal{E}(G)$ to be the graph obtained from G by subdividing every edge of G exactly once. We call G' an $\mathcal{E}$-graph.*

Thus, if G has n vertices and m edges, G' will have $m + n$ vertices. In fact, if we call W the set of vertices of G and W' the set of vertices of G', we have that $W' = W \cup W^*$ ($W \cap W^* = \emptyset$), where W^* is the set of new vertices added during the subdivision. We also have that every edge of G' has an extremity in W and the other in W^* and that there is a bijective map between elements of W^* and edges of G.

This bijective map between the set W^* and the edges of G is the key point in this construction. In the original graph G, we cannot identify particular edges using just an hybrid language. So, if we want to define a property in G, described using its edges, we build $G' = \mathcal{E}(G)$ and describe it in G', using the elements in W^*. These elements can be identified by standard nominals. This is what we do to express the Eulerian property.

For this method to work, we just have to pay attention to an important detail. In $\mathcal{E}$-graphs, it is fundamental to be able to distinguish whether a given vertex is in W or in W^*. Thus, instead of working with one set of nominals $\mathcal{L}$, we work with two such sets, $\mathcal{L}$ and $\mathcal{L}^*$ ($\mathcal{L} \cap \mathcal{L}^* = \emptyset$). Instead of writing $G' = (W', R')$, we write $G' = (W, W^*, R')$, to make clear the difference between the two sets of vertices, and define valuations $\mathbf{V}$ as $\mathbf{V}(p) \in \mathcal{P}(W \cup W^*)$, if p is a proposition symbol, $\mathbf{V}(i) = \{v\}$, such that $v \in W$, if $i \in \mathcal{L}$ and $\mathbf{V}(j) = \{w\}$, such that $w \in W^*$, if $j \in \mathcal{L}^*$. We will denote the nominals in $\mathcal{L}$ by $i_1, i_2, \ldots$ and the nominals in $\mathcal{L}^*$ by $j_1, j_2, \ldots$.

A.7.2 The Eulerian Property in the Hybrid Logics

Since we are going to define the Eulerian property with a formula that explicitly describes an Eulerian path in the graph, we can borrow ideas from three previously presented formulas: the formulas in theorems A.47, A.57 and A.68. But the formula in the first theorem has factorial length, so we will only adapt the formulas in the second and third theorems to the Eulerian case.

This is not a difficult task. The formulas in theorem A.57 and A.68 state that, for a graph $G = (W, R)$, there is a cycle that visits every vertex in W exactly once. To define the Eulerian property, we need formulas that check that, for a graph G, there is a closed path such that every edge in G appears exactly once in it. This is equivalent to check, in $G' = \mathcal{E}(G)$, whether there is a closed path that visits every vertex in W^* exactly once.

First, we adapt the formula in theorem A.57 to express the Eulerian property. Let $G' = \mathcal{E}(G)$ be a $\mathcal{E}$-graph with n vertices in W and m vertices in W^* and let $\mathcal{L}_m = \{j_1, \ldots, j_m\} \subset \mathcal{L}^*$. Let us add a loop to all its vertices in W^*. We can then define the formula that is valid if and only if G is Eulerian.

Theorem A.72. *A connected graph G (with m edges) is* Eulerian *if and only if $\mathcal{F} \Vdash \phi$, where $\mathcal{F}$ is the frame that represents $G' = \mathcal{E}(G)$ and ϕ is the formula*

$$\phi = \psi_m \rightarrow \varepsilon_m,$$

where ψ_m is the formula from lemma A.45 and

$$\varepsilon_m = @_{j_1}\mathbf{E}[\mathbf{XF}j_1 \wedge \mathbf{F}j_2 \wedge \ldots \wedge \mathbf{F}j_m \wedge$$

$$\wedge \mathbf{XG}(j_1 \rightarrow \mathbf{G}j_1) \wedge \mathbf{G}(j_2 \rightarrow \mathbf{XG}\neg j_2) \wedge \ldots \wedge \mathbf{G}(j_m \rightarrow \mathbf{XG}\neg j_m)].$$

Proof. The proof of the above theorem uses the same ideas that are presented in the proof of theorem A.57. □

Now, we adapt the formula in theorem A.68.

Theorem A.73. *A connected graph G (with m edges) is* Eulerian *if and only if $\mathcal{F} \Vdash \phi$, where $\mathcal{F}$ is the frame that represents $G' = \mathcal{E}(G)$ and ϕ is the formula*

$$\phi = @_{j_1} \downarrow x_1.\Diamond\Diamond \downarrow x_2.(\neg x_1 \wedge \Diamond\Diamond \downarrow x_3.(\bigwedge_{1 \leq k < 3} \neg x_k \wedge \Diamond\Diamond \downarrow x_4.(\ldots \wedge \downarrow x_{m-1}.$$

$$(\bigwedge_{1\leq k<m-1} \neg x_k \wedge \Diamond\Diamond \downarrow x_m.(\bigwedge_{1\leq k<m} \neg x_k \wedge \Diamond\Diamond x_1)\ldots).$$

Proof. Here, we use a satisfaction operator at the beginning of the formula because we are now dealing with two sets of vertices: W and W^*. We need the satisfaction operator to guarantee that we are starting the path in a vertex belonging to W^*. The double occurrences of the $\Diamond$ operator ($\Diamond\Diamond$) are designed to discard the vertices belonging to W and analyze only the behaviour of the vertices belonging to W^*. The rest of the proof of the above theorem uses the same ideas that are presented in the proof of theorem A.68. □

As the formulas in theorems A.72 and A.73 were adapted from the ones in theorems A.57 and A.68, the complexity results presented in theorems A.58 and A.69, respectively, also apply to them.

A.7.3 The Eulerian Property in a Graded Modal Logic

In this subsection, we define a graded modal logic that is appropriate for our needs and use it to express the Eulerian property. We call it *graded graph logic*.

Definition A.74. *The language of the* graded graph logic *is a modal language consisting of a set* Φ *of countably many proposition symbols (the elements of* Φ *are denoted by* $p_1, p_2, \ldots$*), the boolean connectives* $\neg$ *and* $\wedge$ *and the modal operators* $\Diamond_i$ *and* $\Diamond_i^{-1}$*, where* $i \in \mathbb{N}$*. The formulas are defined as follows:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_i\varphi \mid \Diamond_i^{-1}\varphi$$

Again, we freely use the standard boolean abbreviations $\vee$, $\rightarrow$, $\leftrightarrow$ and $\bot$ and also the following abbreviations for the duals: $\Box_i\varphi := \neg\Diamond_i\neg\varphi$ and $\Box_i^{-1}\varphi = \neg\Diamond_i^{-1}\neg\varphi$. Also, in order to make the language more elegant, we introduce the following abbreviations: $\blacklozenge_i\varphi = \Diamond_i\varphi \wedge \neg\Diamond_{i+1}\varphi$ and $\blacklozenge_i^{-1}\varphi = \Diamond_i^{-1}\varphi \wedge \neg\Diamond_{i+1}^{-1}\varphi$.

The definition of a *frame* and of a *model* are the same as the ones from section A.2. The semantical notion of satisfaction is defined as follows:

Definition A.75. *The notion of satisfaction of a formula* ϕ *in a model* $\mathcal{M}$ *at a vertex* v*, notation* $\mathcal{M}, v \Vdash \phi$*, is inductively defined as in definition A.4, with the following modifications:*

1. $\mathcal{M}, v \Vdash \Diamond_i \phi$ *iff* $\#\mathcal{R}(v, \phi) \geq i$;

2. $\mathcal{M}, v \Vdash \Diamond_i^{-1} \phi$ *iff* $\#\mathcal{R}^{-1}(v, \phi) \geq i$,

where $\mathcal{R}(v, \phi) = \{w : vRw \text{ and } \mathcal{M}, w \Vdash \phi\}$, $\mathcal{R}^{-1}(v, \phi) = \{w : wRv \text{ and } \mathcal{M}, w \Vdash \phi\}$ *and* $\#S$ *denotes the cardinality of the set* S.

A formula $\Diamond_i \varphi$ is satisfied at a vertex v if there are *at least* i distinct vertices v_k, $1 \leq k \leq i$, such that vRv_k and φ is satisfied at v_k. A formula $\Diamond_i^{-1} \varphi$ is satisfied at a vertex v if there are *at least* i distinct vertices v_k, $1 \leq k \leq i$, such that $v_k Rv$ and φ is satisfied at v_k. A formula $\blacklozenge_i \varphi$ is satisfied at a vertex v if there are *exactly* i distinct vertices v_k, $1 \leq k \leq i$, such that vRv_k and φ is satisfied at v_k. A formula $\blacklozenge_i^{-1} \varphi$ is satisfied at a vertex v if there are *exactly* i distinct vertices v_k, $1 \leq k \leq i$, such that $v_k Rv$ and φ is satisfied at v_k.

Definition A.76. *We may define the length of a formula* ϕ *in the graded graph logic in two different ways. We call them the* standard length*, denoted as* $|\phi|$*, and the* ungraded length*, denoted as* $||\phi||$*. We define both of these lengths in the same way as the length in definition A.10, except for formulas of the form* $\phi = \Diamond_i \psi$ *or* $\phi = \Diamond_i^{-1} \psi$*, where the standard length is defined as* $|\phi| = i + |\psi|$ *and the ungraded length is defined as* $||\phi|| = 1 + ||\psi||$.

Theorem A.77 ([12])**.** *The satisfiability problem and the validity problem for the graded graph logic are PSPACE-Complete in the standard length of the formula.*

Theorem A.78. *The model-checking problem for the graded graph logic is PTIME (linear) in the product of the size of the model and the ungraded length of the formula.*

Proof. The graded graph logic without the converse modalities ($\Diamond_k^{-1}$) is a fragment of Graded-CTL, a logic presented in [13]. Thus, the result follows from the complexity of the model-checking problem for Graded-CTL, which is also presented in [13], and the fact that the presence of converse modalities does not increase the model-checking complexity, as shown in [17]. □

The upper bound for the complexity of the frame-checking problem is the same as in section A.2:

$$FC = O(2^{|p| \times n} \times MC), \tag{A.4}$$

where $|p|$ is the number of distinct proposition symbols that occur in the given formula ϕ and n is the number of vertices in $\mathcal{F}$.

Theorem A.79. *The frame-checking problem for the graded graph logic is PTIME (linear) in the ungraded length of the formula and EXPTIME in the size of the frame and in the number of distinct proposition symbols that occur in the formula.*

Proof. This result follows directly from the discussion above. □

Let G be a graph with n vertices. We now use the language introduced above to define a formula that is valid if and only if G is Eulerian.

Theorem A.80. *A connected graph G (with n vertices) with frame $\mathcal{F}$ is* Eulerian *if and only if $\mathcal{F} \Vdash \phi$, where ϕ is the formula*

$$\phi = \bigvee_{0 \leq k \leq n} (\blacklozenge_k \top \wedge \blacklozenge_k^{-1} \top)$$

Proof. First, it is easy to see that $\blacklozenge_{i_v} \top$ is satisfied at a vertex v if and only if the out-degree of v is i_v and $\blacklozenge_{j_v}^{-1} \top$ is satisfied at a vertex v if and only if the in-degree of v is j_v. Then, by theorem A.31, G is Eulerian if and only if the out-degree of every vertex v in G is equal to its in-degree, which is true if and only if there is k_v for each vertex v, $0 \leq k_v \leq n$ such that both the out-degree and the in-degree of v are equal to k_v. This is true if and only if $\blacklozenge_{k_v} \top \wedge \blacklozenge_{k_v}^{-1} \top$ is satisfied in v, which is true if and only if ϕ is satisfied in every vertex v. □

Let us now determine how complex it is to test whether a graph is Eulerian using the above formula ϕ. First of all, we have a formula with linear ungraded length in the size of the graph. Also, there are no proposition symbols, which means that the valuation is completely irrelevant to the satisfaction of this formula. Let EUL be the complexity of testing whether a graph is Eulerian through a frame-checking of ϕ. Then, taking into account the above observation and the formula in equation (A.4), we have that

$$EUL = MC,$$

where, for the formula ϕ, MC is PTIME (in fact quadratic) in the size of the graph.

Theorem A.81. *The complexity to check whether a graph is Eulerian using the above formula ϕ is PTIME (quadratic) in the size of the graph.*

Proof. This result follows directly from the discussion above. □

A.8 Conclusions

Our goal in this paper is to express, using modal logics, some global graph properties that are central to many computer science applications. The work presented in [30] is closely related to this one. In that work, the interest was also in how to use modal logics to express global graph properties. However, in [30], only the basic graph logic was considered and only connectivity and acyclicity were analyzed. Moreover, the focus of that work was on how to build axiomatizations for classes of graphs with these global properties, while our focus is on how to find formulas expressing each of these properties that can be efficiently used to test whether a graph satisfies them.

The use of model-checking algorithms to verify properties encoded as logical formulas is each day getting more attention and more applications. This happens because many data structures in Computer Science can be encoded as graphs. Our use of model-checking to verify global graph properties is just one more application in a wide and diverse field from which we mention a few other interesting and recent examples. In [59], a fragment of first-order logic is used to express preconditions for graph transformations in a system of graph rewriting. In order to verify if a given graph satisfies the precondition, the system uses a model-checking algorithm. In [17], model-checking algorithms for hybrid logics are used to perform queries in XML files. In [60], the verification of business workflows is conducted using a model-checking algorithm for temporal logics. In [61], an automated model-checking technique is used in the verification of a security protocol. In [62] and [63], an extension of the temporal logic CTL, together with an appropriate model-checking algorithm, are used in the automated analysis of quantum information protocols.

In this work, we present various formalisms, from a very basic modal logic to a very powerful temporal logic and use them to express and test four graph properties: connectivity, acyclicity and the Hamiltonian and Eulerian properties. It would also be interesting to continue this line of work and try to express some other graph properties such as planarity and k-colorability of vertices and edges.

This work is an interesting way of exposing an important issue. Sometimes, standard modal logics, even the ones that are incredibly expressive, such as the μ-calculus, are not capable of expressing some important properties. This happens

because of some strong invariance conditions (such as the ones defined in section A.3) that these logics satisfy. In these cases, the use of a hybrid logic is a very simple way to bypass this problem. Hybrid logics have much weaker invariance conditions [14], which increases the number of definable properties.

In section A.6, we are able to find a formula (theorem A.68) in a hybrid logic with the $\downarrow$ operator that expresses the Hamiltonian property and can be checked in NP time. This is an optimal result, since the problem of deciding whether a graph is Hamiltonian is NP-Complete [33].

Theorem A.68 also implies that we can polynomially reduce the problem of deciding whether a graph is Hamiltonian, which is NP-Complete, to the problem of model-checking a formula of the hybrid graph logic with the $\downarrow$ operator that contain no $\Box$, $\Diamond^+$ or $\Box^+$. This is an alternative proof of the NP-hardness of this model-checking problem stated in theorem A.66. The original proof for the NP-hardness in theorem A.66, presented in [19], uses a different polynomial reduction, starting not from the problem of deciding whether a graph is Hamiltonian, but from the problem of deciding whether a propositional logic formula is satisfiable, which is also a NP-Complete problem [33].

In fact, the formula that expresses the Hamiltonian property and the formula presented in [19] to express the propositional satisfiability problem are remarkably similar, consisting of an alternating sequence of $\downarrow$'s and $\Diamond$'s. This suggests that the fragment presented in [19] could be a simple framework for the description of NP-Complete problems. Expressing these problems in a common language could highlight the underlying similarities between them. As graph theory has a rich collection of NP-Complete problems, it would also be interesting to analyze how we can express them in this fragment.

In section A.7, we use a graded modal logic to express the Eulerian property. The resulting formula can be checked with very good efficiency. In fact, graded modal logics seem to be a very good choice for the efficient verification of graph properties that involve numerical conditions.

Finally, also in section A.7, we describe a way to name edges in a hybrid logic using graph subdivisions. One open issue with this method is that some formulas satisfied at a vertex v in G are no longer satisfied in the same vertex in $\mathcal{E}(G)$. For

instance, a formula $\Diamond\varphi$ may be satisfied at v in G but not in G', because of the new vertices that are added between the old ones. If we just want to evaluate formulas in G' and forget about G, as we did in section A.7, then this is not a problem. But if we want to work with both graphs at the same time, then it would be very interesting to define a translation $\mathcal{T}$ between formulas, such that if φ is satisfied at v in G, then $\mathcal{T}(\varphi)$ is satisfied at v in G'. It would also be interesting to study what other properties could be expressed with hybrid logics using this construction that allows us to name not only vertices but also edges.

Apêndice B

A Study on Multi-Dimensional Products of Graphs and Hybrid Logics

B.1 Introduction[1]

The goal of this work is to address some issues related to products of graphs and products of modal logics. In particular, we want to define a necessary and sufficient condition for a graph to be a non-trivial product of other graphs and to use this characterization to provide sound and complete axiomatic systems for products of modal logics. We are especially interested in products of modal logics with dimension greater than 2, for which there are very few results in the literature [5].

So, our first task in this work is to find a necessary and sufficient condition for a graph to be isomorphic to a (cartesian) product[2] of non-trivial graphs and to verify whether this condition can be expressed in a modal language or in a hybrid language.

[1]Este capítulo expõe o conteúdo do artigo [32], submetido para o periódico Theoretical Computer Science. Uma versão preliminar deste artigo ([38]) foi publicada nos anais do congresso LSFA 2009, Electronic Notes in Theoretical Computer Science, v. 256, p. 103-118, 2009.

[2]In graph theoretical terminology, a product of graphs would be called a *multi-graph*, since it has many distinct sets of edges. In the context of modal logics and Kripke semantics, this notational difference is often lost and all these structures are called simply *labeled graphs*.

In [4] and [5], three properties that are satisfied in graphs that are products are presented: *left commutativity, right commutativity and the Church-Rosser property.* However, although these properties, together with the *reverse Church-Rosser property*, are necessary for a graph to be a product, they are not sufficient (as illustrated by an example in [4]). There are graphs that satisfy left and right commutativity and the Church-Rosser and reverse Church-Rosser properties, but cannot be decomposed as a product of other graphs.

In this work, we introduce a new property called *intransitivity* that, together with the previous ones, form a necessary and sufficient condition for a countable and (weakly) connected graph to be a product. The proof of the necessity of these properties is fairly simple and is done directly, without the need to assume that the graph is countable or connected. On the other hand, the proof of the sufficiency is done in two steps. First, we prove that if a countable and connected graph satisfies the five properties stated above, then its components must satisfy a particular isomorphism. Then, we show that if a countable and connected graph satisfies intransitivity and its components satisfy this particular isomorphism, then the graph is a product.

The limits to the expressive power of basic modal languages are fairly well known. There are a series of standard results that state that frames that are "similar" in a number of ways must agree on the validity of formulas [10]. Using these techniques, we show that the property of intransitivity is not definable in a basic modal language. In fact, we also show that no condition that is necessary and sufficient for a graph to be a product can be definable in a basic modal language.

Hybrid logics are extensions of modal logics that allow explicit references to individual states of a model. Their goal is to extend the expressive power of ordinary modal logics. Besides proposition symbols, they have a second set of atomic formulas, called *nominals*, which have the property of being satisfied at exactly one state [14, 15]. Using a hybrid language, we are able to build a formula that describes intransitivity.

We then proceed to determine the computational complexity of testing, for a finite connected graph, whether it is a product. For this test, we use a model-checking algorithm to verify the formulas that describe each of the five properties

that characterize a product: left and right commutativity, the Church-Rosser and reverse Church-Rosser properties and intransitivity.

Finally, we use this characterization of countable connected products to provide sound and complete axiomatic systems for a large class of products of modal logics.

Products of graphs come up naturally as a possible extension of ordinary Kripke semantics to multi-dimensional modal logics. [4] presents a good textbook discussion of multi-dimensional modal logics and provides many examples of products of modal logics, where the semantics is built using products of graphs. Most of the sound and complete axiomatic systems for products of modal logics presented in the literature are for products of a pair of modal logics, while we are able, using hybrid logics, to provide sound and complete axiomatizations for many products of arbitrary dimensions.

The paper is organized as follows. In section B.2, we introduce the definition of a product of graphs and present four properties related to this definition: left and right commutativity and the Church-Rosser and reverse Church-Rosser properties. We also introduce a new property called intransitivity. In section B.3, we present the concept of graph decomposition and use it to prove that the five properties presented in the previous section form a necessary and sufficient condition for a countable and connected graph to be a product. Section B.4 shows that the property of intransitivity is not definable in a basic modal language and that no necessary and sufficient condition for a graph to be a product can be definable in a basic modal language. In section B.5, we extend the modal language of the previous section to a hybrid language and show that intransitivity can be expressed by a hybrid formula. In section B.6, we determine the computational complexity of testing, through a modal checking algorithm, whether a finite connected graph is a product. In section B.7, we present the notion of a product of modal logic and, using a hybrid language, provide sound and complete axiomatic systems to a large class of products of modal logics. We summarize our results and present potential future works in section B.8.

B.2 Product of Graphs

In this section, we define the product of graphs, following [4] and [5].

Definition B.1. *Given $n \geq 2$ directed graphs $G_i = \langle V_i, E_i \rangle$, $1 \leq i \leq n$, we define their product G, notation $G = G_1 \times G_2 \times \ldots \times G_n$, as the graph $G = \langle V_1 \times V_2 \times \ldots \times V_n, \overline{E}_1, \overline{E}_2, \ldots, \overline{E}_n \rangle$, where for each i, $1 \leq i \leq n$, $\overline{E}_i$ is a binary relation on $V_1 \times V_2 \times \ldots \times V_n$ such that*

$$\langle u_1, \ldots, u_n \rangle \overline{E}_i \langle v_1, \ldots, v_n \rangle \text{ iff } u_i E_i v_i \text{ and } u_k = v_k, \text{ for } k \neq i.$$

An important particular case of the above definition concerns the product of two graphs. In this case, instead of the subscripts 1 and 2, it is common to use the subscripts h and v. They refer to the geometrical intuition of horizontal and vertical accessibility relations.

Definition B.2. *Given two directed graphs $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$, we define their product G, notation $G = G_1 \times G_2$, as the graph $G = \langle V_1 \times V_2, E_h, E_v \rangle$, where for all $x, u \in V_1$ and $y, v \in V_2$*

1. *$\langle x, y \rangle E_h \langle u, v \rangle$ iff $x E_1 u$ and $y = v$ and*
2. *$\langle x, y \rangle E_v \langle u, v \rangle$ iff $y E_2 v$ and $x = u$.*

An example of a product graph is shown in figure B.1.

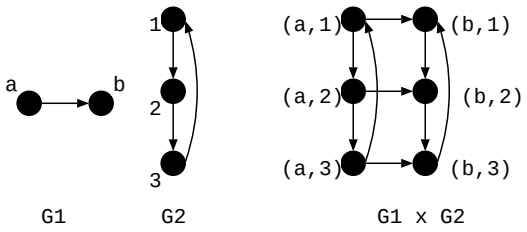

Figure B.1: Product of Graphs

In this work, we would like to identify a necessary and sufficient condition for a graph to be a product of non-trivial graphs. A graph is said to be *trivial* if it has only one vertex and no edges. Every graph can be described as a product of itself with a trivial graph.

In [4] and [5], three properties that are satisfied in graphs that are products are presented. These properties, together with the reverse Church-Rosser property, are necessary for a graph to be a product (figure B.2). Let $G = \langle V, \overline{E}_1, \ldots, \overline{E}_n \rangle$ and $1 \leq i, j \leq n$, $i \neq j$. These properties are defined as follows:

1. Left Commutativity: $\forall x, y, z \in V(x\overline{E}_j y \wedge y\overline{E}_i z \rightarrow \exists u \in V(x\overline{E}_i u \wedge u\overline{E}_j z))$

2. Right Commutativity: $\forall x, y, z \in V(x\overline{E}_i y \wedge y\overline{E}_j z \rightarrow \exists u \in V(x\overline{E}_j u \wedge u\overline{E}_i z))$

3. Church-Rosser Property: $\forall x, y, z \in V(x\overline{E}_j y \wedge x\overline{E}_i z \rightarrow \exists u \in V(y\overline{E}_i u \wedge z\overline{E}_j u))$

4. Reverse Church-Rosser Property: $\forall x, y, z \in V(y\overline{E}_j x \wedge z\overline{E}_i x \rightarrow \exists u \in V(u\overline{E}_i y \wedge u\overline{E}_j z))$

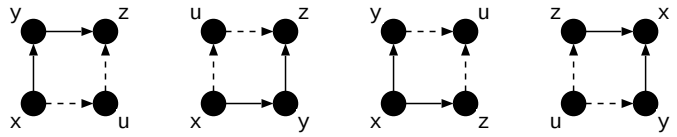

Figure B.2: Left and Right Commutativity and Church-Rosser and Reverse Church-Rosser Properties

However, although these properties are necessary for a graph to be a product, they are not sufficient: there are graphs that satisfy left and right commutativity and the Church-Rosser and reverse Church-Rosser properties, but cannot be decomposed as a product of non-trivial graphs, as shows an example from [4] in figure B.3.

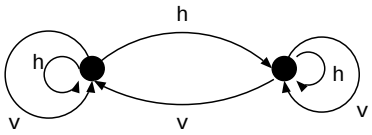

Figure B.3: Counterexample to the sufficiency of the basic properties

In order to obtain a necessary and sufficient condition we need to add a fifth property to the four stated before. We call it *intransitivity*.

Definition B.3. *Let* $G = \langle V, \overline{E}_1, \ldots, \overline{E}_n \rangle$ *and* $1 \leq i, j \leq n$, $i \neq j$. *We say that* G *satisfies* intransitivity *if and only if every triple* $\langle u, v, w \rangle$ *of vertices of* G *that satisfies the conditions*

1. $u \neq v$;

2. $v \neq w$;

3. *there is an undirected path through edges of* $\overline{E}_j$ *from* u *to* v *and*

4. there is an undirected path through edges of $\overline{E}_i$ from v to w

also satisfies the following three conditions:

5. $u \neq w$;

6. $\langle u, w \rangle \notin \overline{E}_i$;

7. $\langle u, w \rangle \notin \overline{E}_j$.

Let xU_iy and xU_jy denote that there is an undirected path through edges of $\overline{E}_i$ ($\overline{E}_j$, respectively) from x to y. Intransitivity is illustrated in figure B.4.

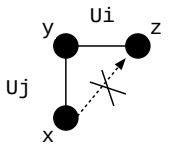

Figure B.4: Intransitivity

Definition B.3 lists the three conditions (items 5, 6 and 7) that we need for intransitivity. However, it turns out that they can be simplified, as, under the hypotheses in definition B.3 (items 1 through 4), the first condition (item 5) implies the other two (items 6 and 7). Suppose that all triples $\langle u, v, w \rangle$ that satisfy the hypotheses in definition B.3 also satisfy the first condition ($u \neq w$). Now, suppose that there is one such triple $\langle a, b, c \rangle$ such that $\langle a, c \rangle \in \overline{E}_i$ (does not satisfy the second condition). Then, aU_jb, bU_ic and $a\overline{E}_ic$. This implies that bU_ia. But then, $\langle a, b, a \rangle$ is a triple that satisfies the hypotheses in definition B.3 but does not satisfy the first condition ($u \neq w$), which is a contradiction to our initial assumption. An analogous argument can be made for the third condition as well.

Thus, when we need to test whether a graph satisfies intransitivity, we just need to verify the first condition in definition B.3. On the other hand, when we know that a graph satisfies intransitivity, we may use any one of the three conditions according to our needs.

Following the above simplification, intransitivity can be described in the following way:

$$\forall x, y, z \in V((xU_jy \wedge yU_iz \wedge x \neq y \wedge y \neq z) \rightarrow (x \neq z))^3.$$

[3]It is important to notice that intransitivity cannot be expressed by a first order formula, since

B.3 Graph Decomposition

The problem of graph decomposition consists of, given a graph, to determine whether this graph can be decomposed in a product of non-trivial graphs. In this work, we consider a restricted version of this problem.

Problem B.4. *Given a countable*[4]*, directed and weakly connected*[5] *(called just* connected *from now on) graph* $G = \langle V, \overline{E}_1, \ldots, \overline{E}_n \rangle$*, where* $\overline{E}_i \neq \emptyset$ *for all* $1 \leq i \leq n$*, determine whether* G *is isomorphic to a product* $G' = G_1 \times G_2 \times \ldots \times G_n$*, where* G_i *is non-trivial for all* $1 \leq i \leq n$*.*

In the general problem, the graph would not have to be necessarily countable or connected.

Hypothesis B.5. *From now on, all the graphs* G *are considered to be directed, countable and connected and to be given in the form* $G = \langle V, \overline{E}_1, \ldots, \overline{E}_n \rangle$*.*

Remark B.6. *We denote by* $\mathcal{V}(G)$ *the set of vertices of a graph* G*.*

In this section, we want to prove that a countable and connected graph G is a product if and only if it satisfies left and right commutativity, the Church-Rosser and reverse Church-Rosser properties and intransitivity. We start with the simpler direction.

Theorem B.7. *If* G *is a product, then* G *satisfies left and right commutativity, the Church-Rosser and reverse Church-Rosser properties and intransitivity.*

Proof. We start with left commutativity. Let us take three vertices u, v and w of G such that $u\overline{E}_j v$ and $v\overline{E}_i w$. As G is a product $G_1 \times \ldots \times G_n$, $u = (u_1, \ldots, u_n)$, $v = (v_1, \ldots, v_n)$ and $w = (w_1, \ldots, w_n)$. Then, as $u\overline{E}_j v$, $u_k = v_k$, for all $k \neq j$, and $u_j E_j v_j$ and, as $v\overline{E}_i w$, $v_k = w_k$, for all $k \neq i$, and $v_i E_i w_i$. In particular, this means that, for $k \neq i$ and $k \neq j$, $u_k = v_k = w_k$. Now, take the vertex x such that $x_k = u_k$, for $k \neq i$ and $k \neq j$, $x_i = w_i$ and $x_j = u_j$ (this vertex exists, since

the definitions of U_i and U_j depend on transitive closures. Nevertheless, this property is still elementary, as it can be defined by a set of first order formulas.

[4] A graph is countable if its set of vertices is countable.

[5] A graph G is weakly connected if, for any pair of vertices u and v of G, there is an undirected path from u to v in G.

$\mathcal{V}(G) = \mathcal{V}(G_1) \times \ldots \times \mathcal{V}(G_n)$). Then, as $u_k = v_k$, for $k \neq j$, $x_i = w_i$ and $v_i E_i w_i$, then $u_i E_i x_i$. This, together with $u_k = x_k$, for $k \neq i$, implies that $u\overline{E}_i x$. Now, as $u_j = x_j$, $v_k = w_k$, for $k \neq i$, and $u_j E_j v_j$, then $x_j E_j w_j$. This, together with $x_k = w_k$, for $k \neq j$, implies that $x\overline{E}_j w$. Right commutativity and the Church-Rosser and reverse Church-Rosser properties follow by analogous arguments.

Now, suppose that G does not satisfy intransitivity. Then, we have vertices x, y and z, such that $x \neq y$, $y \neq z$, there is an undirected $\overline{E}_j$-path from x to y and an undirected $\overline{E}_i$-path from y to z and $x = z$. As G is a product, $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$ and $z = (z_1, \ldots, z_n)$. Then, as there is an undirected $\overline{E}_j$-path from x to y, then $x_k = y_k$, for $k \neq j$ (*). Also, as there is an undirected $\overline{E}_i$-path from y to z, $y_k = z_k$, for $k \neq i$. As $x = z$, then $x_k = z_k$, for all k, which means that $x_k = y_k$, for $k \neq i$ (**). (*) and (**) imply that the three vertices are the same, contradicting the fact that $x \neq y$ and $y \neq z$. □

Notice that, in this direction of the proof, we make no use of hypothesis B.5. This means that theorem B.7 holds for any graph G. Now, we proceed to prove the other direction.

Proposition B.8. *If a graph satisfies left and right commutativity and the Church-Rosser and reverse Church-Rosser properties, it also satisfies the following properties:*

1. *Extended Left Commutativity:* $\forall x, y, z \in V(xU_j^l y \wedge yU_i^k z \rightarrow \exists u \in V(xU_i^k u \wedge uU_j^l z))$;
2. *Extended Right Commutativity:* $\forall x, y, z \in V(xU_i^k y \wedge yU_j^l z \rightarrow \exists u \in V(xU_j^l u \wedge uU_i^k z))$.

where $uU_i^k v$ and $uU_j^l v$ denote that there is an undirected path through edges of $\overline{E}_i$ ($\overline{E}_j$, respectively) of length k (l) from u to v.

Proof. These properties follow by a straightforward induction on the length of the paths, using one of the four hypotheses for each of the four possible cases of edge incidences in the "corner" vertices: horizontal and vertical inward (reverse Church-Rosser), horizontal inward and vertical outward (right commutativity), horizontal outward and vertical inward (left commutativity) and horizontal and vertical outward (Church-Rosser). □

Definition B.9. *Let* $G = \langle V, \overline{E}_1, \ldots, \overline{E}_n \rangle$ *and let* $G_k = \langle V, \overline{E}_k \rangle$, $1 \leq k \leq n$, *be subgraphs of* G. *The* k-components *are the maximal connected subgraphs of* G_k, $\{G_k^1, G_k^2, \ldots\}$ *Notice that the set of* k*-components is countable, since* G *is countable.*

Just as we presented the components of dimension one of the graph in the above definition, we also need to define the components of *co-dimension* one. Notice that in the particular case of bidimensional products, these definitions are equivalent.

Definition B.10. *Let* $G = \langle V, \overline{E}_1, \ldots, \overline{E}_n \rangle$ *and let* $\widetilde{G}_k = \langle V, \overline{E}_1, \ldots, \overline{E}_{k-1}, \overline{E}_{k+1}, \ldots, \overline{E}_n \rangle$, $1 \leq k \leq n$, *be subgraphs of* G. *The* $\widetilde{k}$-components *are the maximal connected subgraphs of* $\widetilde{G}_k$, $\{\widetilde{G}_k^1, \widetilde{G}_k^2, \ldots\}$ *Notice that the set of* $\widetilde{k}$*-components is also countable, since* G *is countable.*

Definition B.11. *We denote by* $\overline{E}_{\widetilde{k}}$ *the set of edges* $\overline{E}_1 \cup \ldots \cup \overline{E}_{k-1} \cup \overline{E}_{k+1} \cup \ldots \cup \overline{E}_n$.

A basic aspect about k-components and $\widetilde{k}$-components that needs to be noticed is that if you are on a vertex u in some k-component and you go through a path of edges in $\overline{E}_k$, you remain in the same k-component and if you are on a vertex u in some $\widetilde{k}$-component and you go through a path of edges in $\overline{E}_{\widetilde{k}}$, you remain in the same $\widetilde{k}$-component.

Proposition B.12. *Two distinct* k*-components (or two distinct* $\widetilde{k}$*-components) have no vertices in common.*

Proof. We show the proof for $\widetilde{k}$-components. The proof for k-components is entirely analogous. Suppose that there are a vertex u and a pair of distinct $\widetilde{k}$-components $\widetilde{G}_k^i$ and $\widetilde{G}_k^j$ such that $u \in \widetilde{G}_k^i$ and $u \in \widetilde{G}_k^j$. Then, as the $\widetilde{k}$-components are maximal connected with respect to all the edges $\overline{E}_l$ such that $l \neq k$, every vertex $x \neq u \in \widetilde{G}_k^i$ has an undirected path from it to u through edges of $\overline{E}_{\widetilde{k}}$. Similarly, every vertex $y \neq u \in \widetilde{G}_k^j$ has an undirected path from it to u through edges of $\overline{E}_{\widetilde{k}}$. But this means that there is an undirected path from x to y through edges of $\overline{E}_{\widetilde{k}}$, which implies that x and y are in the same $\widetilde{k}$-component, contradicting the hypothesis that $\widetilde{G}_k^i$ and $\widetilde{G}_k^j$ are distinct. □

Remark B.13. *From now on, every time that we need to consider a pair of* k*-components* G_k^i *and* G_k^j *or a pair of* $\widetilde{k}$*-components* $\widetilde{G}_k^i$ *and* $\widetilde{G}_k^j$, *the two components in the pair* do not *need to be distinct, unless explicitly mentioned.*

Definition B.14. *For $1 \leq k, l \leq n$, $k \neq l$, we say that a k-component G_k^i is a* l-neighbor to *the k-component G_k^j if there is an edge $\langle u, w\rangle \in \overline{E}_l$ such that $u \in G_k^i$ and $w \in G_k^j$. Notice that it is possible for a k-component to be a l-neighbor to itself.*

Definition B.15. *For $1 \leq k, l \leq n$, $k \neq l$, let f_{kl}^{ij} be the (possibly partial and multi-valued) map that associates to each vertex $u \in G_k^i$ the set of vertices w such that $\langle u, w\rangle \in \overline{E}_l$ and $w \in G_k^j$. We say that f_{kl}^{ij} is the* l-induced map *from G_k^i to G_k^j.*

Proposition B.16. *Let G be a graph that satisfies left and right commutativity, the Church-Rosser and reverse Church-Rosser properties and intransitivity and G_k^i be a l-neighbor to G_k^j. Then, the l-induced map f_{kl}^{ij} from G_k^i to G_k^j is an isomorphism.*

Proof.

1. *f_{kl}^{ij} is a function*: Suppose that there are vertices u, v and w, such that $v \neq w$, $u \in G_k^i$, $v, w \in G_k^j$ and $f_{kl}^{ij}(u) = \{v, w\}$ ($\langle u, v\rangle, \langle u, w\rangle \in \overline{E}_l$). If $u \neq v$, then we have an undirected $\overline{E}_l$-path from u to v, an undirected $\overline{E}_k$-path from v to w (since they are in the same k-component) and an edge from u to w, contradicting intransitivity. If $u = v$, then we have an undirected $\overline{E}_l$-path from v to w and an undirected $\overline{E}_k$-path from w to v, also contradicting intransitivity.

2. *f_{kl}^{ij} is injective*: Analogous to the previous item.

3. $\mathrm{Im}(f_{kl}^{ij}) = \mathcal{V}(G_k^j)$ (f_{kl}^{ij} is surjective): Let v in G_k^j. We need to find a vertex u in G_k^i such that $\langle u, v\rangle \in \overline{E}_l$. As G_k^i is a l-neighbor to G_k^j, there are vertices x in G_k^i and y in G_k^j such that $\langle x, y\rangle \in \overline{E}_l$. We may assume that $y \neq v$, otherwise the proof is over. Now, v and y are in G_k^j, so yU_kv. Then, by extended right commutativity, there is u such that xU_ku (which means that $u \in G_k^i$) and $u\overline{E}_lv$.

4. $\mathrm{Dom}(f_{kl}^{ij}) = \mathcal{V}(G_k^i)$ (f_{kl}^{ij} is total): Analogous to the previous item, using extended left commutativity instead.

5. $u\overline{E}_kw$ if and only if $f_{kl}^{ij}(u)\overline{E}_kf_{kl}^{ij}(w)$: First of all, $u\overline{E}_lf_{kl}^{ij}(u)$ and $w\overline{E}_l\ f_{kl}^{ij}(w)$. If $u\overline{E}_kw$, we can use left commutativity to conclude that $f_{kl}^{ij}(u)\overline{E}_k\ f_{kl}^{ij}(w)$. On the other hand, if $f_{kl}^{ij}(u)\overline{E}_kf_{kl}^{ij}(w)$, we can use right commutativity to conclude that $u\overline{E}_kw$.

□

Definition B.17. *If G_k^i is a l-neighbor to G_k^j and the l-induced map f_{kl}^{ij} is an isomorphism between G_k^i and G_k^j, we call f_{kl}^{ij} a* l-primitive isomorphism.

Now, in a case such as the one in the above proposition, where all of the induced maps between neighbor k-components are isomorphisms, we can easily extend the isomorphisms beyond immediate neighbor k-components.

Let $L = [l_1, \ldots, l_n]$ denote a finite list. In order to define the composition of isomorphisms between k components, we extend the notation and write the maps as f_{kL}^{ij}. This notation can easily be used to denote the original l-induced maps and l-primitive isomorphisms, since in this case we just have to take $L = [l]$.

Remark B.18. *If all the elements in the set $\{f_{kL_1}^{i,i+1}, f_{kL_2}^{i+1,i+2}, \ldots, f_{kL_{j-i}}^{j-1,j}\}$ are isomorphisms, then*

$$f_{kL}^{ij} = f_{kL_{j-i}}^{j-1,j} \circ \cdots \circ f_{kL_2}^{i+1,i+2} \circ f_{kL_1}^{i,i+1},$$

where $L = L_1 \circ L_2 \circ \cdots \circ L_{j-i}$, is also an isomorphism.

Remark B.19. *If f_{kL}^{ij} is an isomorphism, then its inverse is also an isomorphism and is denoted by $f_{kL'}^{ji}$, where L' is the list symmetric to L.*

Definition B.20. *If f_{kL}^{ij} is a primitive isomorphism or is obtained from primitive isomorphisms using composition and inverse, we call f_{kL}^{ij} an* orthogonal isomorphism *or* O-isomorphism.

Lemma B.21. *Let G be a graph that satisfies left and right Commutativity, the Church-Rosser and reverse Church-Rosser properties and intransitivity. Then, for all pairs G_k^i and G_k^j of k-components, there is an O-isomorphism f_{kL}^{ij} between them. This means that all k-components are isomorphic between themselves and that the set of undirected paths through $\overline{E}_{\widetilde{k}}$-edges between G_k^i and G_k^j induce an isomorphism between the two k-components.*

Proof. First, as G is connected, for every k-component G_k^i there must be a k-component G_k^j such that either G_k^i is a neighbor to G_k^j or G_k^j is a neighbor to G_k^i. Then, as G is countable (in particular, as the number of k-components in G is countable), for any pair of k-components, we can build an O-isomorphism between them from the primitive isomorphisms given by proposition B.16 using a *finite* number of inverse and composition operations on them, as described in remarks B.19 and B.18. □

Definition B.22. *Let G be a graph. If G satisfies intransitivity and, for all pairs G_k^i and G_k^j of k-components, there is an O-isomorphism f_{kL}^{ij} between them, we say that G is* well-behaved.

Lemma B.23. *Every $\widetilde{k}$-component contains as a subgraph at least one complete l-component, for all $l \neq k$.*

Proof. Let $\widetilde{G}_k^{i_k}$ be a $\widetilde{k}$-component and u be an arbitrary vertex in $\widetilde{G}_k^{i_k}$. It belongs to some l-component G_l^i. But all the vertices reachable from u through an undirected path of $\overline{E}_l$-edges, which are the vertices in the same l-component as u, are also in $\widetilde{G}_k^{i_k}$, if $l \neq k$. Hence, $\widetilde{G}_k^{i_k}$ contains as a subgraph at least one complete l-component. □

Lemma B.24. *Let G be a well-behaved graph, G_k^i be a k-component and $\widetilde{G}_k^j$ be a $\widetilde{k}$-component. Then, $\mathcal{V}(G_k^i) \cap \mathcal{V}(\widetilde{G}_k^j)$ is a singleton set.*

Proof. First, we show that $\mathcal{V}(G_k^i) \cap \mathcal{V}(\widetilde{G}_k^j) \neq \emptyset$. Let u be a vertex in $\mathcal{V}(\widetilde{G}_k^j)$. It belongs to some k-component G_k^l. Using the O-isomorphism between G_k^i and G_k^l, there is a vertex w in $\mathcal{V}(G_k^i)$ such that there is an undirected path through $\overline{E}_{\widetilde{k}}$-edges from u to w in G. But this means that u and w belong to the same $\widetilde{k}$-component. Thus, w belongs to $\mathcal{V}(G_k^i) \cap \mathcal{V}(\widetilde{G}_k^j)$.

Now, if there were more then one vertex in the intersection $\mathcal{V}(G_k^i) \cap \mathcal{V}(\widetilde{G}_k^j)$, then there are at least two vertices x and y in G_k^i such that there are undirected paths through $\overline{E}_{\widetilde{k}}$-edges from u to x and from u to y in G (as $u, x, y \in \widetilde{G}_k^j$). But this contradicts the fact that undirected paths through $\overline{E}_{\widetilde{k}}$-edges induce an O-isomorphism between k-components. □

Corollary B.25. *Let G be a well-behaved graph, H be a subgraph of G that contains as a subgraph at least one complete k-component and $\widetilde{G}_k^i$ be a $\widetilde{k}$-component. Then, $\mathcal{V}(H) \cap \mathcal{V}(\widetilde{G}_k^i) \neq \emptyset$.*

Proof. Let G_k^l be a k-component such that H contains G_k^l. From lemma B.24, $\mathcal{V}(G_k^l) \cap \mathcal{V}(\widetilde{G}_k^i)$ is a singleton set. Then, as $\mathcal{V}(G_k^l) \subseteq \mathcal{V}(H)$, we have that $\mathcal{V}(H) \cap \mathcal{V}(\widetilde{G}_k^i) \neq \emptyset$. □

Corollary B.26. *Let G be a well-behaved graph, $S = \{\widetilde{G}_k^{i_k} : 1 \leq k \leq n\}$ be a set that contains one $\widetilde{k}$-component for each $1 \leq k \leq n$. Then, $\bigcap\{\mathcal{V}(H) : H \in S\} \neq \emptyset$.*

Proof. Let $V_j = \bigcap\{\mathcal{V}(\widetilde{G}_k^{i_k}) : 1 \leq k \leq j+1\}$. We start with $V_1 = \mathcal{V}(\widetilde{G}_1^{i_1}) \cap \mathcal{V}(\widetilde{G}_2^{i_2})$. $\widetilde{G}_1^{i_1}$ is a maximal connected subgraph of G and, by lemma B.23, contains as subgraph at least one complete l-component, for every $l \neq 1$. In particular, it contains at least one complete 2-component. Then, by corollary B.25, $V_1 \neq \emptyset$.

Now, for $1 < k < n$, $V_k = V_{k-1} \cap \mathcal{V}(\widetilde{G}_{k+1}^{i_{k+1}})$. Let u be an arbitrary vertex in V_{k-1}. It belongs to some $(k+1)$-component G_{k+1}^i. But all the vertices reachable from u through an undirected path of $\overline{E}_{k+1}$-edges, which are the vertices in the same $(k+1)$-component as u, are also in the same $\widetilde{l}$-component that u is, if $l \neq k+1$. Then, the complete $(k+1)$-component that u belongs to is in every $\widetilde{G}_j^{i_j}$, for $1 \leq j \leq k$, which means that the subgraph of G generated by the vertices in V_{k-1} contains as a subgraph at least one complete $(k+1)$-component. Then, by corollary B.25, $V_k \neq \emptyset$. □

Lemma B.27. *Let G be a graph. If G satisfies intransitivity and, for all pairs G_k^i and G_k^j of k-components, there is an O-isomorphism f_{kL}^{ij} between them, then G is (isomorphic to) a product.*

Proof. For $1 \leq k \leq n$, let $G_k^* = \langle V_k, E_k \rangle$ be an arbitrary k-component with $V_k = \{v_k^1, k_k^2, \ldots\}$ (E_k is the restriction of the set $\overline{E}_k$ of G to V_k). Lemma B.24 implies that each $\widetilde{k}$-component contains exactly one of the vertices in V_k. Besides that, by proposition B.12, a vertex in V_k cannot be in more than one $\widetilde{k}$-component. Thus, without loss of generality, we can enumerate the vertices in V_k, such that $v_k^i \in \widetilde{G}_k^i$.

Let $P = \prod G_k^* = \langle \prod V_k, E_1^P, \ldots, E_n^P \rangle$. We want to prove that there is an isomorphism between G and P. Let us consider the map $\mathcal{L}$ that associates to each vertex $u \in \mathcal{V}(G)$ the vertex $(u_1, \ldots, u_n) \in \mathcal{V}(P)$ where $u_k = v_k^{i_k}$ if and only if $u \in \widetilde{G}_k^{i_k}$.

1. $\mathcal{L}$ is a well-defined total function: Clearly $\mathcal{L}$ is total, since every vertex u of G belongs to some $\widetilde{k}$-component $\widetilde{G}_k^{i_k}$, for each $1 \leq k \leq n$. Besides that, the map $\mathcal{L}$ associates to each vertex u of G, a unique vertex $(u_1, \ldots, u_n)$ of P. Otherwise, there would be, for at least one vertex u and at least one k, $1 \leq k \leq n$, a pair of distinct $\widetilde{k}$-components $\widetilde{G}_k^i$ and $\widetilde{G}_k^j$ such that $u \in \widetilde{G}_k^i$ and $u \in \widetilde{G}_k^j$. But this contradicts proposition B.12.

2. If $u\overline{E}_k w$ in G, then $\mathcal{L}(u) E_k^P \mathcal{L}(w)$ in P: If $u\overline{E}_k w$, then, for all $l \neq k$, $u \in \widetilde{G}_l^{i_l}$ if and only if $w \in \widetilde{G}_l^{i_l}$. This means that if $\mathcal{L}(u) = (u_1, \ldots, u_n)$ and $\mathcal{L}(w) =$

$(w_1, \ldots, w_n)$, then $u_l = w_l$, for all $l \neq k$. Now, we have that the vertex v_k^i of G belongs to $\widetilde{G}_k^i$, so if $u_k = v_k^j$ and $w_k = v_k^l$, this means that $u, v_k^j \in \widetilde{G}_k^j$ and $w, v_k^l \in \widetilde{G}_k^l$. Hence, there is an undirected path through $\overline{E}_{\widetilde{k}}$-edges in G from u to v_k^j and another undirected path through $\overline{E}_{\widetilde{k}}$-edges in G from w to v_k^l. Let G_k^i be the k-component that contains u and w. Using the O-isomorphism between G_k^i and G_k^*, as $u\overline{E}_k w$ in G_k^i, then $u_k = v_k^j \overline{E}_k v_k^l = w_k$ in G_k^*. This, together with $u_l = w_l$, for all $l \neq k$, implies that $\mathcal{L}(u) E_k^P \mathcal{L}(w)$.

3. If $\mathcal{L}(u) E_k^P w'$ in P, then there is a *unique* vertex w in G such that $w' = \mathcal{L}(w)$ and $u\overline{E}_k w$: Let $L(u) = (u_1, \ldots, u_n)$ and $w' = (w_1, \ldots, w_n)$. Then, as $\mathcal{L}(u) E_k^P w'$ in P, we have that $u_l = w_l$, for all $l \neq k$ and $u_k E_k w_k$. If $u_k = v_k^i$ and $w_k = v_k^j$, then, $v_k^i E_k v_k^j$ in G_k^*. Besides that, this means that u and v_k^i are in the same $\widetilde{k}$-component $\widetilde{G}_k^i$, which implies that there is an undirected path through $\overline{E}_{\widetilde{k}}$-edges in G from u to v_k^i. Let G_k^l be the k-component that contains u. The O-isomorphism between G_k^l and G_k^* relates u to v_k^i, so, as $v_k^i E_k v_k^j$ in G_k^*, then there is a vertex $x \in G_k^i$ such that $u\overline{E}_k x$ and the O-isomorphism between G_k^l and G_k^* relates x to v_k^j, which is equivalent to saying that there is an undirected path through $\overline{E}_{\widetilde{k}}$-edges in G from x to v_k^j. Now, the vertex x satisfies the properties of the vertex that we were looking for. First, $u\overline{E}_k x$. Second, this implies that $u \in \widetilde{G}_l^{i_l}$ if and only if $w \in \widetilde{G}_l^{i_l}$, for all $l \neq k$. Thus, if $\mathcal{L}(x) = (x_1, \ldots, x_n)$, then $u_l = x_l$, for all $l \neq k$. Third, as $x \in \widetilde{G}_k^j$, then $x_k = v_k^j$. Hence, $\mathcal{L}(x) = w'$.

   Now, suppose that there is another vertex $y \neq x$ that also satisfies these properties, i.e., $u\overline{E}_k y$ and $\mathcal{L}(y) = \mathcal{L}(x)$. But if $u\overline{E}_k y$, then u, x and y belong to to the same k-component G_k^l. However, by lemma B.24, as there is only one vertex in the intersection of the set of vertices of a k-component and the set of vertices of a $\widetilde{k}$-component, x and y are in distinct $\widetilde{k}$-components, so $\mathcal{L}(y) \neq \mathcal{L}(x)$.

4. If $u' E_k^P \mathcal{L}(w)$ in P, then there is a *unique* vertex u in G such that $u' = \mathcal{L}(u)$ and $u\overline{E}_k w$: Analogous to the previous item.

5. $\mathcal{L}$ is surjective: Suppose that there is no u in G such that $L(u) = (x_1, \ldots, x_n)$. Let $x_k = v_k^{i_k}$. This is equivalent to saying that the intersection $\bigcap\{\mathcal{V}(\widetilde{G}_k^{i_k}) :$

$1 \leq k \leq n\}$ is empty, contradicting corollary B.26.

6. $\mathcal{L}$ is injective: By item 5, as $\mathcal{L}$ is surjective, every vertex in P is of the form $\mathcal{L}(x)$ for some vertex x in G. For all $1 \leq k \leq n$, a vertex $\mathcal{L}(x)$ has at least one k-neighbor $\mathcal{L}(y)$ in P ($\mathcal{L}(x)E_k^P\mathcal{L}(y)$ or $\mathcal{L}(y)E_k^P\mathcal{L}(x)$). In order to see this, suppose that there is a vertex $\mathcal{L}(x)$ and a given k such that $\mathcal{L}(x)$ does not have any k-neighbors. Then, this vertex alone forms a k-component. However, as all k-components are isomorphic between themselves, then all k-components are isolated vertices, which means that there are no E_k^P-edges in P. Using item 2, this also means that there are no $\overline{E}_k$-edges in G. This contradicts the definition of problem B.4. Now, suppose that there are two distinct vertices u and w in G such that $\mathcal{L}(u) = \mathcal{L}(w)$. Let $\mathcal{L}(x)$ be a k-neighbor of $\mathcal{L}(u)$ and $\mathcal{L}(y)$ be a l-neighbor of $\mathcal{L}(u)$. Then, using items 3 and 4, we have that x is a k-neighbor to both u and w and that y is a l-neighbor to both u and w. But this means that there are both undirected paths through $\overline{E}_k$-edges and through $\overline{E}_l$-edges from u to w, which contradicts intransitivity.

□

Theorem B.28. *If G satisfies left and right commutativity, the Church-Rosser and reverse Church-Rosser properties and intransitivity, then G is a product.*

Proof. Straightforward from lemmas B.21 and B.27. □

Theorem B.29. *Let G be a graph. G is a product if and only if G satisfies left and right commutativity, the Church-Rosser and reverse Church-Rosser properties and intransitivity.*

Proof. Straightforward from theorems B.7 and B.28. □

B.4 Modal Definability

In this section, we show that the property of intransitivity is not definable in a basic modal language. In fact, we also show that no condition that is necessary and sufficient for a graph to be a product can be definable in a basic modal language. Even tough we restricted ourselves to countable and connected graphs in the previous

section, this restriction is not necessary for the undefinability results presented in this section.

B.4.1 A Basic Modal Language

In this section, we define a modal language with a family of modal operators: $\Diamond_i$, $\Diamond_i^{-1}$ and $\blacklozenge_i$, for $1 \leq i \leq n$.

Definition B.30. *Let us consider a modal language consisting of a set Φ of countably many proposition symbols, the boolean connectives $\neg$ and $\wedge$ and the modal operators $\Diamond_i$, $\Diamond_i^{-1}$ and $\blacklozenge_i$, for $1 \leq i \leq n$. The formulas are defined as follows:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_i\varphi \mid \Diamond_i^{-1}\varphi \mid \blacklozenge_i\varphi,$$

where $p \in \Phi$.

We freely use the standard boolean abbreviations $\vee$, $\rightarrow$, $\leftrightarrow$ and $\bot$ and also the abbreviation $\widetilde{\Box}_i\varphi = \neg\widetilde{\Diamond}_i\neg\varphi$, where $\widetilde{\Diamond}_i \in \{\Diamond_i, \Diamond_i^{-1}, \blacklozenge_i\}$ and $\widetilde{\Box}_i$ is the correspondent $\Box$.

We now define the structures in which we evaluate our formulas: *frames* and *models*.

Definition B.31. *A* frame *is a tuple $\mathcal{F} = (V, \{R_i\}_{1\leq i\leq n})$, where V is a set (finite or not) of vertices and R_i, $1 \leq i \leq n$, are binary relations over V, i.e., $R_i \subseteq V \times V$. We also define the auxiliary relations U_i, $1 \leq i \leq n$, as the transitive closures of the relations $R_i \cup R_i^{-1}$.*

As we can see, a frame is a graph with the distinct sets of edges R_i, $1 \leq i \leq n$. Also, a graph with an appropriate number of distinct sets of edges (one for each modality $\Diamond_i$ of the logic) can be used as a frame for the logic. So, in the rest of this work, the terms *graph* and *frame* are considered equivalent.

Definition B.32. *A* model *is a pair $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, where $\mathcal{F}$ is a frame and $\mathbf{V}$ is a valuation function mapping proposition symbols into subsets of V, i.e., $\mathbf{V} : \Phi \mapsto \mathcal{P}(V)$.*

The notion of satisfaction is defined as follows:

Definition B.33. *Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. The notion of* satisfaction *of a formula φ in a model $\mathcal{M}$ at a vertex v, notation $\mathcal{M}, v \Vdash \varphi$, can be inductively defined as follows:*

1. *$\mathcal{M}, v \Vdash p$ iff $v \in \mathbf{V}(p)$;*

2. *$\mathcal{M}, v \Vdash \top$ always;*

3. *$\mathcal{M}, v \Vdash \neg\varphi$ iff $\mathcal{M}, v \nVdash \varphi$;*

4. *$\mathcal{M}, v \Vdash \varphi_1 \wedge \varphi_2$ iff $\mathcal{M}, v \Vdash \varphi_1$ and $\mathcal{M}, v \Vdash \varphi_2$;*

5. *$\mathcal{M}, v \Vdash \Diamond_i\varphi$ iff there is a $w \in V$ such that vR_iw and $\mathcal{M}, w \Vdash \varphi$;*

6. *$\mathcal{M}, v \Vdash \Diamond_i^{-1}\varphi$ iff there is a $w \in V$ such that wR_iv and $\mathcal{M}, w \Vdash \varphi$;*

7. *$\mathcal{M}, v \Vdash \blacklozenge_i\varphi$ iff there is a $w \in V$ such that vU_iw and $\mathcal{M}, w \Vdash \varphi$.*

If $\mathcal{M}, v \Vdash \varphi$ for every vertex v in a model $\mathcal{M}$, we say that φ is *globally satisfied* in $\mathcal{M}$, notation $\mathcal{M} \Vdash \varphi$. And if φ is globally satisfied in all models $\mathcal{M}$ of a frame $\mathcal{F}$, we say that φ is *valid* in $\mathcal{F}$, notation $\mathcal{F} \Vdash \varphi$. We also write $G \Vdash \phi$ to denote that the formula ϕ is valid in G when G is used as a frame for the logic. Finally, if φ is valid in every frame $\mathcal{F}$, we say that φ is valid, notation $\Vdash \varphi$.

When we say that a formula ϕ defines or describes some graph property, this means that a graph G has the desired property if and only if $G \Vdash \phi$.

As shown in [4] and [5], a graph satisfies left commutativity, right commutativity and the Church-Rosser property if the following formulas are valid in it for all pairs $1 \leq i, j \leq n$, $i \neq j$:

1. $com_{ij} = \Diamond_j\Diamond_i p \leftrightarrow \Diamond_i\Diamond_j p$ (left and right commutativity);

2. $chr_{ij} = \Diamond_i\Box_j p \rightarrow \Box_j\Diamond_i p$ (Church-Rosser property).

Then, a graph satisfies the reverse Church-Rosser property if the following formula, analogous to chr_{ij}, is valid in it for all pairs $1 \leq i, j \leq n$, $i \neq j$:

3. $rchr_{ij} = \Diamond_i^{-1}\Box_j^{-1}\varphi \rightarrow \Box_j^{-1}\Diamond_i^{-1}\varphi$.

B.4.2 A Limitative Result

The limits to the expressive power of basic modal languages are fairly well known. There are a series of standard results that state that frames that are "similar" in a number of ways must agree on the validity of formulas. We can then use these results to prove that a certain property *cannot* be expressed by any modal formula. To do this, we take two frames that are "similar" and show that in one the desired property holds, while in the other it does not. We present one of these "similarity" results (more details about it and other related results may be found in [10]), and then we prove two results for graph products using it.

Definition B.34. *Let* $\mathcal{M} = (W, \{R_i\}_{1 \leq i \leq n}, \mathbf{V})$ *and* $\mathcal{M}' = (W', \{R'_i\}_{1 \leq i \leq n}, \mathbf{V}')$ *be two models. A function* $f : W \to W'$ *is a* bounded morphism *from* $\mathcal{M}$ *to* $\mathcal{M}'$ *if it satisfies the following conditions:*

1. *w and $f(w)$ satisfy the same proposition symbols;*
2. *f is a homomorphism with respect to R_i (if wR_iv, then $f(w)R'_i\ f(v)$);*
3. *if $f(w)R'_iv'$, then there is a v such that wR_iv and $f(v) = v'$;*
4. *if $w'R'_if(v)$, then there is a w such that wR_iv and $f(w) = w'$.*

A similar definition can be given for a bounded morphism of frames, just removing the part of the above definition that deals with valuations (item (i)). If there is a bounded morphism from a model (frame) $\mathcal{M}$ ($\mathcal{F}$) to a model (frame) $\mathcal{M}'$ ($\mathcal{F}'$), we use the notation $\mathcal{M} \to \mathcal{M}'$ ($\mathcal{F} \to \mathcal{F}'$). If there is a surjective bounded morphism, then we say that $\mathcal{M}'$ ($\mathcal{F}'$) is a bounded morphic image *of $\mathcal{M}$ ($\mathcal{F}$) and use the notation $\mathcal{M} \Rightarrow \mathcal{M}'$ ($\mathcal{F} \Rightarrow \mathcal{F}'$).*

The last item of the previous definition is usually not necessary. However, as the modalities $\Diamond_i^{-1}$ and $\blacklozenge_i$ deal with the inverses of the relations R_i, we have to enforce it to get the preservation result that we want. It may seem like conditions such as "if wU_iv, then $f(w)U'_i\ f(v)$", which is analogous to condition (ii), and others analogous to conditions (iii) and (iv) should also be added. However, this is not necessary, as the definition of U_x, with its use of transitive closure, and conditions (ii), (iii) and (iv) already imply such conditions.

Below is a basic theorem about modal definability that is going to be used to prove our results. Its proof for a language that contains only one modality can be found at [10]. It is not difficult to extend that proof to a language that contains a family of modalities, each with its accessibility relation.

Theorem B.35. *Let $\mathcal{M}$ and $\mathcal{M}'$ be two models such that $\mathcal{M} \to \mathcal{M}'$. Then, $\mathcal{M}, w \Vdash \phi$ if and only if $\mathcal{M}', f(w) \Vdash \phi$.*

Corollary B.36. *Let $\mathcal{F}$ and $\mathcal{F}'$ be two frames such that $\mathcal{F} \Rightarrow \mathcal{F}'$. If $\mathcal{F} \Vdash \phi$, then $\mathcal{F}' \Vdash \phi$.*

Theorem B.37. *Neither intransitivity nor its negation are modally definable.*

Proof. In figure B.5, let $f = \{(1, a), (2, b), (3, a), (4, b)\}$ and $g = \{(a, A), (b, A)\}$. It is straightforward to prove that f and g are surjective bounded morphisms. It is also not difficult to see that the first and third graphs respect intransitivity, while the second does not. By corollary B.36, since neither intransitivity nor its negation are preserved under bounded morphic images, they are not modally definable. □

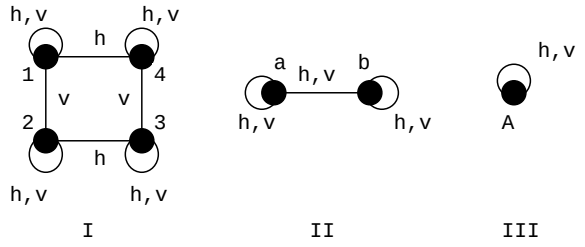

Figure B.5: Graph III is a bounded morphic image of graph II, which is a bounded morphic image of graph I (each undirected edge represents a pair of symmetric edges)

Theorem B.38. *No necessary and sufficient condition for a graph* to be *a product or for a graph* not to be *a product can be modally definable.*

Proof. We take again the same bounded morphisms between the graphs in figure B.5. It is not difficult to see that the first and third graphs are products while the second is not. By corollary B.36, since neither the property of being a product nor the property of not being a product are preserved under bounded morphic images, they are not modally definable. □

This is not the only possible proof of theorem B.38. However, as the counter-example used in theorem B.37 could also be used in theorem B.38 without any change, it was our choice to prove both theorems through the use of bounded morphic images.

B.5 A Hybrid Extension

As was shown in the previous section, a basic modal language does not have enough expressive power to describe the properties that we want. In order to achieve our goal, we need a language that is more expressive. In this section we describe a simple hybrid language and then use it to define intransitivity.

B.5.1 Language

A good way to improve the expressive power of a modal logic is to consider hybrid extensions of it. The fundamental resource that allows a logic to be called "hybrid" is a set of *nominals*. Nominals are a new kind of atomic symbol and they behave similarly to proposition symbols. The key difference between a nominal and a proposition symbol is related to their valuation in a model. While the set $\mathbf{V}(p)$ for a proposition symbol p can be any element of $\mathcal{P}(V)$, the set $\mathbf{V}(a)$ for a nominal a has to be a singleton set. This way, each nominal is true at exactly one state of the model, and thus, can be used to refer to this unique state. This is why these logics are called "hybrid": they are still modal logics, but they have the capacity to refer to specific states of the model, like in first-order logic.

The expressive power and computational complexity of a hybrid extension of a given modal logic usually lie between the ones of the original modal logic and the ones of first-order logic. This, however, depends on which operators, besides the nominals, are added to build the hybrid language. With the addition of state-variables and quantifiers, it is possible to achieve full first-order expressivity and complexity (undecidability). For a general introduction to hybrid logics, [14] and [15] can be consulted.

Here, we consider a simple hybrid extension of the modal logic presented in the previous section. We add nominals and the so-called satisfaction operators to the

language.

Definition B.39. *Let us consider a hybrid language consisting of a set Φ of countably many proposition symbols and a set Ω of countably many nominals such that $\Phi \cap \Omega = \emptyset$, the boolean connectives $\neg$ and $\wedge$, the modal operators $\Diamond_i$, $\Diamond_i^{-1}$ and $\blacklozenge_i$, $1 \leq i \leq n$ and the satisfaction operators $@_a$, for each nominal a. The formulas are defined as follows:*

$$\varphi ::= p \mid a \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_i\varphi \mid \Diamond_i^{-1}\varphi \mid \blacklozenge_i\varphi \mid @_a\varphi,$$

where $p \in \Phi$ and $a \in \Omega$.

It is common in the literature to use the letters i, j and k for nominals. However, as we already use these letters for the multiple edges of the graphs and frames, we choose to denote the nominals by the first letters a, b and c.

The definition of a frame for this language is the same as definition B.31, but the definition of a model is slightly different from definition B.32.

Definition B.40. *A* hybrid model *is a pair $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, where $\mathcal{F}$ is a frame and $\mathbf{V}$ is a valuation function mapping proposition symbols into subsets of V, i.e., $\mathbf{V} : \Phi \mapsto \mathcal{P}(V)$ and mapping nominals into singleton subsets of V, i.e, if a is a nominal then $\mathbf{V}(a) = \{v\}$ for some $v \in V$. We call this unique state that belongs to $\mathbf{V}(a)$ the* denotation *of a under $\mathbf{V}$. We can also say that a denotes the single state belonging to $\mathbf{V}(a)$.*

The notion of satisfaction is defined as follows:

Definition B.41. *The notion of satisfaction is defined adding the following extra clauses to definition B.33:*

1. *$\mathcal{M}, v \Vdash a$ iff $v \in \mathbf{V}(a)$;*
2. *$\mathcal{M}, v \Vdash @_a\varphi$ iff $\mathcal{M}, d_a \Vdash \varphi$, where d_a is the denotation of a under $\mathbf{V}$.*

B.5.2 Hybrid Definability

Using this hybrid language, we can now express intransitivity.

Theorem B.42. *A graph G respects intransitivity if and only if $G \Vdash int_{ij}$ for all pairs $1 \leq i, j \leq n$, $i \neq j$, where int_{ij} is the formula*

$$int_{ij} = (a \wedge \neg b \wedge \blacklozenge_j(b \wedge \neg c \wedge \blacklozenge_i c)) \rightarrow \neg c.$$

Proof. ($\Leftarrow$) Suppose that $G \Vdash int_{ij}$ for all pairs $1 \leq i, j \leq n$, $i \neq j$, but G does not respect intransitivity. Then, there are at least three vertices x, y, z, $x \neq y$ and $y \neq z$, in G such that xU_jy, yU_iz and $x = z$. We evaluate int_{ij} in a model with a valuation $\mathbf{V}$ such that $\mathbf{V}(a) = \{x\}$, $\mathbf{V}(b) = \{y\}$ and $\mathbf{V}(c) = \{z\}$. Then, it is straightforward to see that $(G, \mathbf{V}), x \nVdash int_{ij}$, which contradicts the fact that int_{ij} is valid in G.

($\Rightarrow$) Suppose that G respects intransitivity but $G \nVdash int_{ij}$ for some pair $1 \leq i, j \leq n$, $i \neq j$. Then, there is a valuation $\mathbf{V}$ and a vertex u such that $(G, \mathbf{V}), u \nVdash int_{ij}$. Let $\mathbf{V}(a) = \{x\}$, $\mathbf{V}(b) = \{y\}$ and $\mathbf{V}(c) = \{z\}$. Then, we must have that $u = x$, $x \neq y$, $y \neq z$, xU_jy, yU_iz and $(G, \mathbf{V}), u \Vdash c$, which means that $u = x = z$. This contradicts the fact that G respects intransitivity. □

It should be noticed that we do not need to use the satisfaction operators to describe intransitivity. However, we have the satisfaction operators in the language because they are very useful in completeness proofs of axiomatizations of hybrid logics, as is shown in section B.7.

B.6 Verification of the Product Property

In this section, we analyze the computational complexity to verify whether a finite connected graph is a product. This issue involves two basic decision problems.

Definition B.43. *The* model-checking problem *consists of, given a formula ϕ and a finite model $\mathcal{M} = (W, R, \mathbf{V})$, determining the set $S_{\mathcal{M}}(\phi) = \{v \in W : \mathcal{M}, v \Vdash \phi\}$.*

Definition B.44. *The* frame-checking problem *consists of, given a formula ϕ and a finite frame $\mathcal{F}$, determining whether $\mathcal{F} \Vdash \phi$.*

Definition B.45. *We define the length of a formula φ, denoted by $|\varphi|$, inductively in the following way: $|p| = |a| = |\top| = 1$, $|\neg\phi| = |\Diamond_i\phi| = |\Diamond_i^{-1}\phi| = |\blacklozenge_i\phi| = |@_a\phi| = 1 + |\phi|$ and $|\phi_1 \wedge \phi_2| = 1 + |\phi_1| + |\phi_2|$.*

Definition B.46. *Let $\mathcal{M} = (W, R, \mathbf{V})$ be a model. Let $|W|$ be the number of vertices in W and $|R|$ the number of pairs in R. We define the size of the model (or the frame, or the graph) as $|W| + |R|$.*

Theorem B.47 ([17])**.** *The model-checking problem for the logic presented in definition B.39 is PTIME (linear) in the product of the size of the model and the length of the formula.*

We can provide a simple upper bound for the complexity of the frame-checking problem based on the complexity of the correspondent model-checking problem. We have that $\mathcal{F} \Vdash \phi$ if and only if $S_{\mathcal{M}}(\neg\phi) = \emptyset$ for every model $\mathcal{M}$ of $\mathcal{F}$. So, an algorithmic way to check whether $\mathcal{F} \Vdash \phi$ is to apply the model-checking algorithm to all the pairs $(\neg\phi, \mathcal{M})$, where $\mathcal{M}$ is a model of $\mathcal{F}$.

Thus, let FC be the complexity of the frame-checking problem and MC be the complexity of the model-checking problem. Then,

$$FC = O(2^{|p|\times m} \times m^{|a|} \times MC), \tag{B.1}$$

where $|p|$ is the number of distinct proposition symbols that occur in the given formula ϕ, $|a|$ is the number of distinct nominals that occur in ϕ and m is the number of vertices in $\mathcal{F}$. The distinction between proposition symbols and nominals in the above equation comes from the special restriction on the valuation of nominals. We need to apply the model-checking algorithm to every model $\mathcal{M}$ of the given frame $\mathcal{F}$. Every proposition symbol p that appears in ϕ may receive 2^m possible valuations $\mathbf{V}(p)$, while every nominal a may only receive m possible valuations $\mathbf{V}(a)$.

Theorem B.48. *The frame-checking problem for the logic presented in definition B.39 is PTIME (linear) in the length of the formula and EXPTIME in the size of the frame, in the number of distinct proposition symbols that occur in the formula and in the number of distinct nominals that occur in the formula.*

Proof. This result follows directly from the discussion above. □

It should be noticed that this calculation of the complexity of the frame-checking problem is just a general upper-bound and it can be reduced in some concrete situations.

From equation (B.1), we can see that, from the point of view of computational complexity, it is interesting to try to express the properties with formulas that use only nominals and no proposition symbols, since the presence of proposition symbols makes the verification of the property be exponential on the size of the frame.

Definition B.49. *A* pure formula *is a formula with no occurrences of proposition symbols.*

So, pure formulas are interesting from the point of view of the complexity of the frame-checking problem. Besides that, as is shown in section B.7, pure formulas also have advantages when used as axioms in an axiomatic system.

The formula in theorem B.42, that describes intransitivity, is already pure. Now, we need to find pure formulas that describe left and right commutativity and the Church-Rosser and reverse Church-Rosser properties.

Theorem B.50. *A graph G respects left and right commutativity if and only if $G \Vdash com^*_{ij}$ for all pairs $1 \leq i, j \leq n$, $i \neq j$, where com^*_{ij} is the formula*

$$com^*_{ij} = \Diamond_j \Diamond_i a \leftrightarrow \Diamond_i \Diamond_j a.$$

Proof. ($\Leftarrow$) Suppose that $G \Vdash com^*_{ij}$ for all pairs $1 \leq i, j \leq n$, $i \neq j$, but G does not respect at least one of left and right commutativity. We suppose that it does not respect left commutativity, as the symmetric case is entirely analogous. Then, there are at least a pair i and j and at least three vertices x, y, z in G such that xR_jy, yR_iz but there is no u such that xR_iu and uR_jz. We evaluate com^*_{ij} in a model with a valuation $\mathbf{V}$ such that $\mathbf{V}(a) = \{z\}$. Then, it is straightforward to see that $(G, \mathbf{V}), x \Vdash \Diamond_j \Diamond_i a$ but $(G, \mathbf{V}), x \nVdash \Diamond_i \Diamond_j a$, which contradicts the fact that com^*_{ij} is valid in G.

($\Rightarrow$) Suppose that G respects left and right commutativity but $G \nVdash com^*_{ij}$ for some pair $1 \leq i, j \leq n$, $i \neq j$. Then, there is a valuation $\mathbf{V}$ and a vertex u such that $(G, \mathbf{V}), u \nVdash com^*_{ij}$. So, either the left side of com^*_{ij} is satisfied at u but the right side is not or the other way around. We suppose that it is the first case, as the symmetric case is entirely analogous. Let $\mathbf{V}(a) = \{x\}$. Then, there is a vertex y such that uR_jy and yR_ix, but there is no vertex z such that uR_iz and zR_jx. This contradicts the fact that G respects left commutativity. □

We can see that for left and right commutativity, the task of finding a pure formula that describes these properties was fairly easy, as we just have to substitute de propositional symbol p in the original formula com_{ij} by the nominal a. For the Church-Rosser property, this task is more difficult. The simple substitution of p by a in the original formula chr_{ij} does not work. However, it is possible to describe the Church-Rosser property with a pure formula. In [39], a pure formula for the Church-Rosser property is presented: $chr^*_{ij} = \Diamond_j a \rightarrow \Box_i \Diamond_j \Diamond_i^{-1} a$. It is also shown in [39] that it is not possible to describe the Church-Rosser property with a pure formula without the use of a converse modality $\Diamond_i^{-1}$.

Theorem B.51. *A graph G respects the Church-Rosser property if and only if $G \Vdash chr^*_{ij}$ for all pairs $1 \leq i, j \leq n$, $i \neq j$, where chr^*_{ij} is the formula*

$$chr^*_{ij} = \Diamond_j a \rightarrow \Box_i \Diamond_j \Diamond_i^{-1} a.$$

Proof. ($\Leftarrow$) Suppose that $G \Vdash chr^*_{ij}$ for all pairs $1 \leq i, j \leq n$, $i \neq j$, but G does not respect the Church-Rosser property. Then, there are at least a pair i and j and at least three vertices x, y, z in G such that xR_iy, xR_jz but there is no u such that yR_ju and zR_iu. We evaluate com^*_{ij} in a model with a valuation $\mathbf{V}$ such that $\mathbf{V}(a) = \{z\}$. Then, it is straightforward to see that $(G, \mathbf{V}), x \Vdash \Diamond_j a$. Besides that, as there is no u such that such that yR_ju and zR_iu, we have that $(G, \mathbf{V}), x \Vdash \Diamond_i \Box_j \Box_i^{-1} \neg a$, which is equivalent to $(G, \mathbf{V}), x \nVdash \Box_i \Diamond_j \Diamond_i^{-1} a$. This contradicts the fact that chr^*_{ij} is valid in G.

($\Rightarrow$) Suppose that G respects the Church-Rosser property but $G \nVdash chr^*_{ij}$ for some pair $1 \leq i, j \leq n$, $i \neq j$. Then, there is a valuation $\mathbf{V}$ and a vertex u such that $(G, \mathbf{V}), u \nVdash chr^*_{ij}$. So, the left side of chr^*_{ij} is satisfied at u but the right side is not. Let $\mathbf{V}(a) = \{x\}$. Then, uR_jx. Besides that, there is a vertex y such that uR_iy and for all vertices z such that yR_jz it is not the case that xR_iz. This contradicts the fact that G respects the Church-Rosser property. □

Theorem B.52. *A graph G respects the reverse Church-Rosser property if and only if $G \Vdash rchr^*_{ij}$ for all pairs $1 \leq i, j \leq n$, $i \neq j$, where $rchr^*_{ij}$ is the formula*

$$rchr^*_{ij} = \Diamond_j^{-1} a \rightarrow \Box_i^{-1} \Diamond_j^{-1} \Diamond_i a.$$

Proof. The proof is analogous to the one from the previous theorem. □

We now can determine how complex it is to test whether a finite connected graph is a product. By theorems B.29, B.42, B.50, B.51 and B.52, a finite connected graph G is a product if and only if $G \Vdash pro$, where pro is the formula

$$pro = \bigwedge_{1 \leq i,j \leq n, i \neq j} (com^*_{ij} \wedge chr^*_{ij} \wedge rchr^*_{ij} \wedge int_{ij})$$

As the test consists of, given a graph G, frame-check whether $G \Vdash pro$, we can calculate the complexity of this test in the following way. Let $FC(\phi)$ be the complexity to frame-check whether $G \Vdash \phi$. Then, we want to calculate $FC(pro)$. But, as there are $O(n^2)$ pairs $1 \leq i, j \leq n$, $i \neq j$ we have that

$$FC(pro) = O(n^2) \times FC_{ij},$$

where

$$FC_{ij} = FC(com^*_{ij}) + FC(chr^*_{ij}) + FC(rchr^*_{ij}) + FC(int_{ij}).$$

We should notice first that there are no proposition symbols in pro and that the length of pro is constant and does not depend on the size of the graph. Besides that, com^*_{ij}, chr^*_{ij} and $rchr_{ij}$ have only one nominal and int_{ij} has three distinct nominals. However, as the frame-checking of a formula ϕ is done through a series of model-checkings of the formula $\neg\phi$, we can decrease the number of nominals involved in the test using the fact that the valuations of nominals are always singleton sets.

$$\neg int_{ij} \equiv \neg(\neg(a \wedge \neg b \wedge \blacklozenge_j(b \wedge \neg c \wedge \blacklozenge_i c)) \vee \neg c) \equiv a \wedge \neg b \wedge \blacklozenge_j(b \wedge \neg c \wedge \blacklozenge_i c) \wedge c$$

But a formula $a \wedge c$ in hybrid logic means that the valuations of a and c are the same, so we can use only one of these nominals, substituting all of the occurrences of the second by the first. Besides that, a formula $a \wedge \neg b$ means that a and b denote distinct vertices, so it is redundant to write at another point of the formula $b \wedge \neg a$. Hence,

$$\neg int_{ij} \equiv a \wedge \neg b \wedge \blacklozenge_j(b \wedge \blacklozenge_i a)$$

So, we can perform the test with only two nominals. Then, taking into account these observations and the formula in equation (B.1), we have that

$$FC_{ij} = O(m^2 \times MC),$$

where, in this case, MC is PTIME (in fact linear) in the size of the graph. This implies that

$$FC(pro) = O(n^2 \times m^2 \times MC).$$

Theorem B.53. *The complexity to check whether a finite connected graph is a product using the above formula pro is PTIME (cubic) in the size of the graph and PTIME (quadratic) in the number of distinct sets of edges (number of dimensions).*

Proof. This result follows directly from the discussion above. □

B.7 Hybrid Axiomatizations of Products of Modal Logics

Products of graphs come up naturally as a possible extension of ordinary Kripke semantics to multi-dimensional modal logics. In this section, we present the concept of a *product of modal logics*, where the semantics is built using products of Kripke frames, and then use hybrid logic to build complete axiomatizations for a large class of products of modal logics.

Let Φ be a countable set of proposition symbols and Ω a countable set of nominals such that $\Phi \cap \Omega = \emptyset$. For a finite set of modal operators $\mathcal{O}$, we define the set of hybrid formulas $HFor(\mathcal{O})$ through the rule

$$\varphi ::= p \mid a \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid O\varphi \mid @_a\varphi,$$

where $p \in \Phi$, $a \in \Omega$ and $O \in \mathcal{O}$. We also define the set of basic modal formulas $MFor(\mathcal{O})$ as the subset of $HFor(\mathcal{O})$ that contain only the formulas without nominals and satisfaction operators. When it is not relevant whether we are working with a hybrid language or with a basic modal language, we write $For(\mathcal{O})$ for the set of formulas over the language.

There are two common ways of defining a modal logic: the "syntactical" way and the "semantical" way. Let $F = For(\mathcal{O})$ be the set of all formulas in a given language (be it a hybrid language or a basic modal language). The syntactical way consists of taking a set $\mathcal{A} \subseteq F$ of axioms and a set $\mathcal{R}$ of rules and defining a modal logic L as the smallest set of formulas such that $\mathcal{A} \subseteq L$ and L is closed under the rules in $\mathcal{R}$. The semantical way consists of taking a class $\mathcal{C}$ of frames and defining a modal logic $L = \mathrm{Log}(F, \mathcal{C})$ as the set

$$\mathrm{Log}(F, \mathcal{C}) = \{\phi \in F : \mathcal{F} \Vdash \phi, \text{ for all } \mathcal{F} \in \mathcal{C}\}.$$

We can also define the duals of the sets $\mathrm{Log}(F, \mathcal{C})$. Let Fr_F be the class of frames in which formulas of F are evaluated and $\Sigma \subseteq F$ be a set of formulas. We define the class of frames $\mathrm{Fr}(F, \Sigma)$ as

$$\mathrm{Fr}(F, \Sigma) = \{\mathcal{F} \in Fr_F : \mathcal{F} \Vdash \phi, \text{ for all } \phi \in \Sigma\}.$$

If $L = \mathrm{Log}(F, \mathcal{C})$ is a semantically defined modal logic, we write $\Vdash_{\mathcal{C}} \phi$ to denote that $\phi \in L$.

Suppose that $\nVdash_{\mathcal{C}} \neg\phi$. This means that there is a frame $\mathcal{F} \in \mathcal{C}$ such that $\mathcal{F} \nVdash \neg\phi$. This, on the other hand, means that there is a model $\mathcal{M}$ of the frame $\mathcal{F}$ and a vertex v such that $\mathcal{M}, v \Vdash \phi$. So, ϕ is satisfied at a vertex of a model of a frame in $\mathcal{C}$.

Definition B.54. *We say that a formula ϕ is* $\mathcal{C}$-satisfiable *if it is satisfied at a vertex of a model of a frame in $\mathcal{C}$ (equivalently, if $\nVdash_{\mathcal{C}} \neg\phi$). We say that a set Ψ of formulas is $\mathcal{C}$-satisfiable if there is a vertex of a model of a frame in $\mathcal{C}$ that satisfies all formulas $\phi \in \Psi$.*

Definition B.55. *A modal logic L is called* Kripke-complete *if $L = Log(F, \mathcal{C})$ for some class $\mathcal{C}$ of frames. In this case, we say that L is characterized (or determined) by $\mathcal{C}$.*

Let $MFor_1 = MFor(\{\Diamond\})$ be the set of formulas over the basic mono-modal language. As a trivial example of a modal logic that is a subset of $MFor_1$ and can be defined in this semantical way, the set $\mathbf{K} = \{\phi \in MFor_1 : \Vdash \phi\}$ of all the valid formulas in $MFor_1$ is a modal logic, as $\mathbf{K} = \mathrm{Log}(MFor_1, \mathcal{C})$, where $\mathcal{C}$ in this case is the class of *all* frames. The set $MFor_1$ itself is also a modal logic, as $MFor_1 = \mathrm{Log}(MFor_1, \emptyset)$.

The notation **K** is usual in the modal logic literature for the logic over the basic mono-modal language determined by the class of all frames. Common notations for other modal logics that we use in this section are **K4**, **K5**, **T**, **B**, **D**, **Alt**, **B4** and **S5** for, respectively, the logics over the basic mono-modal language determined by the classes of transitive frames, euclidean frames, reflexive frames, symmetric frames, serial frames, functional frames, transitive and symmetric frames and transitive, symmetric and reflexive frames[6].

[6]More details on these classes of frames can be found in [4] and [40].

Let $MFor_n = MFor(\{\Diamond_1, \ldots, \Diamond_n\})$, $HFor_1 = HFor(\{\Diamond\})$ and $HFor_n = HFor(\{\Diamond_1, \ldots, \Diamond_n\})$ be the sets of formulas over the basic multi-modal language, the hybrid mono-modal language and the hybrid multi-modal language, respectively. If $L = \mathrm{Log}(MFor_1, \mathcal{C})$, then we denote by $L(n), L(@)$ and $L(n, @)$ the sets $\mathrm{Log}(MFor_n, \mathcal{C})$, $\mathrm{Log}(HFor_1, \mathcal{C})$ and $\mathrm{Log}(HFor_n, \mathcal{C})$, respectively. So, we have $\mathbf{K4}(@)$, $\mathbf{S5}(n)$ and so on.

A *product of modal logics* is a multi-modal logic that is defined semantically in the following way.

Definition B.56. *Let $\{L_i : 1 \leq i \leq n\}$ be a finite set of Kripke-complete modal logics defined over the sets $For(\mathcal{O}_i)$, respectively, such that the sets of modalities $\mathcal{O}_i$, $1 \leq i \leq n$, are pairwise disjoint (in the present work, we consider that each L_i is mono-modal, having only one modality $\Diamond_i$). Let $F = For(\cup_i \mathcal{O}_i)$. We consider that $For(\mathcal{O}_i)$, $1 \leq i \leq n$, are either all sets of basic modal formulas or all sets of hybrid formulas. Then, the product of $L_1, \ldots, L_n$ is the multi-modal logic defined as*

$$L_1 \times \cdots \times L_n = Log(F, \mathcal{C}),$$

where

$$\mathcal{C} = \{\mathcal{F}_1 \times \cdots \times \mathcal{F}_n : \mathcal{F}_i \in Fr(L_i), \text{ for all } 1 \leq i \leq n\}.$$

Definition B.57. *We denote by L^n the product $L \times \cdots \times L$, where L occurs n times.*

For example, $\mathbf{K} \times \mathbf{K}$ is the modal logic determined by the class of all product frames $\mathcal{F}_1 \times \mathcal{F}_2$, $\mathbf{K4} \times \mathbf{S5}$ is the modal logic determined by all product frames $\mathcal{F}_1 \times \mathcal{F}_2$ such that $\mathcal{F}_1$ is transitive and $\mathcal{F}_2$ is transitive, symmetric and reflexive and $\mathbf{S5}^n$ is the modal logic determined by all product frames $\mathcal{F}_1 \times \cdots \times \mathcal{F}_n$ such that each $\mathcal{F}_i$, $1 \leq i \leq n$ is transitive, symmetric and reflexive.

It should be noticed that the product of modal logics, being defined on Kripke-complete modal logics, is also Kripke-complete by definition.

Now, one relevant issue about products of modal logics is the so-called *axiomatization problem*. Given the fact that a product $L = L_1 \times \cdots \times L_n$ is semantically defined, is it also possible to find a correspondent syntactical definition for L?

In order to better define the axiomatization problem, we need the notions of soundness and completeness of an axiomatization. Let $\mathbb{A} = (\mathcal{A}, \mathcal{R})$ be an axiomatic

system with the set of axioms $\mathcal{A}$ and the set of rules $\mathcal{R}$. Let $L_{\mathbb{A}}$ be the smallest set of formulas in the language under consideration such that $\mathcal{A} \subseteq L_{\mathbb{A}}$ and $L_{\mathbb{A}}$ is closed under the rules in $\mathcal{R}$. If $\phi \in L_{\mathbb{A}}$, we say that ϕ is a *theorem* of $\mathbb{A}$ and use the notation $\vdash_{\mathbb{A}} \phi$.

Definition B.58. *We say that a formula ϕ is* $\mathbb{A}$-consistent *if $\nvdash_{\mathbb{A}} \neg\phi$. We say that a finite set Ψ_0 of formulas is* $\mathbb{A}$-consistent *if $\nvdash_{\mathbb{A}} \neg \bigwedge\{\phi : \phi \in \Psi_0\}$. Finally, we say that a set Ψ of formulas is* $\mathbb{A}$-consistent *if every finite subset $\Psi_0 \subseteq \Psi$ is $\mathbb{A}$-consistent.*

Definition B.59. *If $\mathbb{A} = (\mathcal{A}, \mathcal{R})$ is an axiomatic system and Σ is a set of formulas, we denote by $\mathbb{A} + \Sigma$ the axiomatic system $(\mathcal{A} \cup \Sigma, \mathcal{R})$.*

Definition B.60. *Let $L = Log(F, \mathcal{C})$ be a semantically defined modal logic and $\mathbb{A}$ an axiomatic system. We say that $\mathbb{A}$ is* sound *for L if and only if $\vdash_{\mathbb{A}} \phi$ implies $\Vdash_{\mathcal{C}} \phi$ for each formula ϕ in the language under consideration. Equivalently, as can be seen directly from the definitions of $\mathbb{A}$-consistency (definition B.58) and $\mathcal{C}$-satisfiability (definition B.54), $\mathbb{A}$ is* sound *for L if and only if every $\mathcal{C}$-satisfiable formula in the language under consideration is $\mathbb{A}$-consistent.*

Theorem B.61. *$\mathbb{A}$ is* sound *for L if and only if every $\mathcal{C}$-satisfiable set of formulas in the language under consideration is $\mathbb{A}$-consistent.*

Proof. ($\Rightarrow$) Let Ψ be a $\mathcal{C}$-satisfiable set of formulas. Suppose that Ψ is not $\mathbb{A}$-consistent. Then, there are formulas $\phi_1, \ldots, \phi_n \in \Psi$ such that $\vdash_{\mathbb{A}} \neg(\phi_1 \wedge \cdots \wedge \phi_n)$. By soundness, $\Vdash_{\mathcal{C}} \neg(\phi_1 \wedge \cdots \wedge \phi_n)$, which means that $\phi_1 \wedge \cdots \wedge \phi_n$ is not $\mathcal{C}$-satisfiable. This implies that there is no vertex in no model of no frame in $\mathcal{C}$ that satisfies all of the formulas $\phi_1, \ldots, \phi_n$. As all of these formulas are in Ψ, this contradicts the fact that Ψ is $\mathcal{C}$-satisfiable.

($\Leftarrow$) Let ϕ be a $\mathcal{C}$-satisfiable formula. Then, $\{\phi\}$ is a $\mathcal{C}$-satisfiable set. Now, by the hypothesis, $\{\phi\}$ is $\mathbb{A}$-consistent, which means that ϕ is $\mathbb{A}$-consistent, proving soundness. □

Definition B.62. *Let $L = Log(F, \mathcal{C})$ be a semantically defined modal logic and $\mathbb{A}$ an axiomatic system. We say that $\mathbb{A}$ is* complete *for L if and only if $\Vdash_{\mathcal{C}} \phi$ implies $\vdash_{\mathbb{A}} \phi$ for each formula ϕ in the language under consideration. Equivalently, $\mathbb{A}$ is* complete *for L if and only if every $\mathbb{A}$-consistent formula in the language under consideration is $\mathcal{C}$-satisfiable.*

In general, completeness does not automatically imply that every $\mathbb{A}$-consistent set of formulas in the language under consideration is $\mathcal{C}$-satisfiable. This is different from soundness, where we could jump from formulas to set of formulas (theorem B.61).

Definition B.63. *Let $L = Log(F, \mathcal{C})$ be a semantically defined modal logic and $\mathbb{A}$ an axiomatic system. We say that $\mathbb{A}$ is* strongly complete *for L if and only if every $\mathbb{A}$-consistent set of formula in the language under consideration is $\mathcal{C}$-satisfiable. Strong completeness implies completeness (the argument is analogous to the one in the second part of the proof of theorem B.61), but, in general, completeness does not imply strong completeness.*

Then, for a product $L = L_1 \times \cdots \times L_n$, if we have a sound and complete axiomatic system $\mathbb{A}_i$ for each logic L_i, $1 \leq i \leq n$, is there a way to combine them in order to get a sound and complete axiomatic system for L?

In the basic modal language, this issue is actually more complex than it may seem at first sight and there is no general method in the literature to take sound and complete axiomatizations of n arbitrary Kripke-complete logics and generate a sound and complete axiomatization for their product.

The problem seems to come from the fact that, as theorem B.38 shows, no necessary and sufficient condition for a graph to be a product can be expressed in a basic modal language. If L_i is syntactically defined by the axiomatic system $\mathbb{A}_i = (\mathcal{A}_i, \mathcal{R}_i)$, for $1 \leq i \leq n$ and $C = \{com_{ij} \wedge chr_{ij} : 1 \leq i, j \leq n, i \neq j\}$, then $[L_1, \ldots, L_n]$ denotes the modal logic syntactically defined by $\mathbb{C} = (\cup_i \mathcal{A}_i, \cup_i \mathcal{R}_i) + C$. $[L_1, \ldots, L_n]$ is called the *commutator* of $L_1, \ldots, L_n$. As left and right commutativity and the Church-Rosser property are necessary conditions for a graph to be a product, we have that $[L_1, \ldots, L_n] \subseteq L_1 \times \cdots \times L_n$. However, as they are not sufficient, we do not have in general that $L_1 \times \cdots \times L_n \subseteq [L_1, \ldots, L_n]$. Logics for which this second inclusion is true are called *product-matching*.

The proof, for given logics L_i, $1 \leq i \leq n$ that $\phi \in L_1 \times \cdots \times L_n$ (which is a semantically defined logic) implies $\phi \in [L_1, \ldots, L_n]$ (which is a syntactically defined logic) is nothing more that a completeness proof for $\mathbb{C}$ with respect to $L_1 \times \cdots \times L_n$. The standard method for completeness proofs for modal logics is the construction of so-called *canonical models*. The vertices of the canonical model are all

the possible maximal consistent sets of formulas of the language under consideration. As there are countably many proposition symbols and countably many nominals in the language, then there are also countably many formulas, which means that there are countably many maximal consistent sets. Hence, the canonical model has a countable frame. The idea of the completeness proof using canonical models is that the canonical model should satisfy the semantical properties of the class $\mathcal{C}$ of frames that characterizes the logic. If this happens, then we can proceed to show that every $\mathbb{C}$-consistent formula is $\mathcal{C}$-satisfiable. For details on completeness proofs using canonical models, [10] should be consulted.

What happens in the specific case of commutators is that, as left and right commutativity and the Church-Rosser property are not sufficient for a graph to be a product, it is not possible to guarantee that the canonical model obtained from $\mathbb{C}$ is a product. So, canonical models are insufficient by themselves to derive completeness. Either another approach must be taken or, at least, some complementary steps must be done in order to reach completeness. Hence, there is no general method for completeness proofs of axiomatizations for products of logics, but there are particular logics that have been shown to be product-matching. As a start, $\mathbf{K} \times \mathbf{K}$ is product-matching, i.e., the axiomatic system of $[\mathbf{K}, \mathbf{K}]$ is complete for $\mathbf{K} \times \mathbf{K}$ [40].

An interesting result, shown in [40], states that for an arbitrary $n \in \mathbb{N}$, $n > 1$, the logic $\mathbf{Alt}^n$ is product-matching, i.e., the axiomatic system of the commutator of n copies of $\mathbf{Alt}$ is complete for $\mathbf{Alt}^n$.

Another result shown in [40] states that, for every pair of so-called PTC-logics L_1 and L_2, the logic $L_1 \times L_2$ is product-matching, i.e., the axiomatic system of $[L_1, L_2]$ is complete for $L_1 \times L_2$. It is also shown in [40] that some non-PTC-logics fail to be product-matching. There are also logics for which it is unknown whether they are product-matching or not. Several of the most common logics are, in fact, PTC-logics, as $\mathbf{K4}$, $\mathbf{T}$, $\mathbf{B}$, $\mathbf{D}$ and $\mathbf{S5}$. For the formal definition of a PTC-logic and the detailed results, [40] should be consulted.

The completeness results for $\mathbf{K} \times \mathbf{K}$ and for a product of a pair of PTC-logics do not generalize to products of higher dimensions, as is shown by the following theorem from [41].

Theorem B.64 ([41])**.** *Let $n \geq 3$ and let $L \subseteq MFor_n$ be any modal logic such that*

$\mathbf{K}^n \subseteq L \subseteq \mathbf{S5}^n$. *Then, L is not finitely axiomatizable.*

Summing up, while working in the basic modal language, we have no general method to build complete axiomatizations for products of logics and a significant group of products of logics of dimension greater than 2 cannot be axiomatized. As we already showed, a hybrid language offers the possibility to describe products of graphs, so it might also be well suited for an improvement in these results regarding axiomatizations for products of logics.

So, let us now consider logics $L_1 \times \cdots \times L_n$ over the hybrid language. First of all, $\mathbf{K}(n, @) \subseteq L_1 \times \cdots \times L_n$, so a good starting point to axiomatize $L_1 \times \cdots \times L_n$ is to get a complete axiomatization for $\mathbf{K}(n, @)$. In fact, this starting point will prove to be even better than expected, as completeness proofs for hybrid logics can be automatically extended to other hybrid logics under certain conditions, as shown in theorem B.66.

We consider the set of axioms and rules shown in figure B.6, where p and q are proposition symbols, a and b are nominals and φ and ψ are formulas.

(CT) All classical tautologies

(K$_i$) $\vdash \Box_i(p \to q) \to (\Box_i p \to \Box_i q)$

(Du$_i$) $\vdash \Box_i p \leftrightarrow \neg\Diamond_i\neg p$

(K$_@$) $\vdash @_a(p \to q) \to (@_a p \to @_a q)$

(SD) $\vdash @_a p \leftrightarrow \neg@_a\neg p$

(Ref) $\vdash @_a a$

(Agr) $\vdash @_a @_b p \leftrightarrow @_b p$

(Int) $\vdash a \to (p \leftrightarrow @_a p)$

(Back$_i$) $\vdash \Diamond_i @_a p \to @_a p$

(MP) If $\vdash \varphi$ and $\vdash \varphi \to \psi$, then $\vdash \psi$

(Gen$_i$) If $\vdash \varphi$, then $\vdash \Box_i\varphi$

(Gen$_@$) If $\vdash \varphi$, then $\vdash @_a\varphi$

(Nam) If $\vdash @_a\varphi$ and a does not occur in φ, then $\vdash \varphi$

(BG$_i$) If $\vdash @_a\Diamond_i b \to @_b\varphi$ and $b \neq a$ does not occur in φ, then $\vdash @_a\Box_i\varphi$

(Sub) If $\vdash \varphi$, then $\vdash \varphi^\sigma$, where σ is a substitution that uniformly replaces nominals by nominals and proposition symbols by arbitrary formulas

Figure B.6: An axiomatic system $\mathbb{A}_{\mathbf{K}(n,@)}$ for the logic $\mathbf{K}(n, @)$

Theorem B.65. *The axiomatic system $\mathbb{A}_{\mathbf{K}(n,@)}$, shown in figure B.6, is sound and strongly complete for the logic $\mathbf{K}(n,@)$.*

Proof. The proof of soundness and strong completeness for the mono-modal logic $\mathbf{K}(@)$ is well established in the literature. It is presented, in different levels of detail, in [10], [64] and [18]. The proof for the multi-modal case follows directly along the same lines. □

Theorem B.66. *Let $\mathcal{C}$ be a class of frames defined as $\mathcal{C} = \{\mathcal{F} \in Fr_{HFor_n} : \mathcal{F} \Vdash \phi \text{ for all } \phi \in P\}$, where P is a $\mathbb{A}_{\mathbf{K}(n,@)}$-consistent set of pure formulas. Then, the axiomatic system $\mathbb{P} = \mathbb{A}_{\mathbf{K}(n,@)} + P$ is sound and strongly complete for the logic $Log(HFor_n, \mathcal{C})$.*

Proof. Again, as in the previous theorem, the proof for the mono-modal logic $\mathbf{K}(@)$ is well established in the literature. It is also presented in [10], [64] and [18] and the proof for the multi-modal case follows directly along the same lines. The key point in the proof is that the canonical model for $\mathbb{P}$ is necessarily in $\mathcal{C}$. □

The theorem above provides automatic soundness and strong completeness for a large class of multi-modal hybrid logics. So, if we have a finite set P of pure formulas that defines the class of product frames, then all we need to do is add this set as axioms and we get completeness for $\mathbf{K}(@)^n$. Then, completeness for more restricted classes of product frames can also be obtained, as long as these classes can be described by pure formulas.

Theorem B.29 describes a necessary and sufficient condition for a countable and connected graph to be a product and theorems B.42, B.50, B.51 and B.52 show that this condition can be expressed by a pure formula. However, we cannot just plug this formula into theorem B.66 and get the completeness that we want, as there are two issues that need to be addressed first.

We start by the most obvious one. There are two hypotheses that need to be satisfied before we can apply theorem B.29: the graph needs to be connected and the graph needs to be countable. Canonical models are countable, as previously discussed, so this hypothesis is not a problem. However, when working with a hybrid language, the canonical models are not guaranteed to be connected. Thus,

we need to explicitly restrict ourselves to the class of connected frames and the best way to do this is to find a pure formula that describes connectivity.

Theorem B.67. *A graph $G = G_1 \times \cdots \times G_n$ is connected if and only if G_i is connected for every $1 \leq i \leq n$.*

Proof. ($\Rightarrow$) Suppose that there is at least one G_i, $1 \leq i \leq n$, that is not connected. So, there is a pair of vertices $x, y \in G_i$ such that there is no undirected path between them in G_i. Now, take any two vertices $u = (u_1, \ldots, u_n)$ and $w = (w_1, \ldots, w_n)$ in G such that $u_k = w_k$, for all $k \neq i$, $u_i = x$ and $w_i = y$. Then, u and w belong to the same i-component and this i-component is not connected, because there would only be an undirected path through i-edges from u to w if there were an undirected path from x to y in G_i. As u_k can be chosen arbitrarily for $k \neq i$, all of the i-components of G are not connected. It is straightforward to see that this means that G itself is not connected.

($\Leftarrow$) Suppose that G_i is connected for every $1 \leq i \leq n$. Take two vertices $u = (u_1, \ldots, u_n)$ and $w = (w_1, \ldots, w_n)$ in G. We show how to find an undirected path between them in G. Let k_1 be the first coordinate such that $u_{k_1} \neq w_{k_1}$ (that is, $u_k = w_k$ for every $k \leq k_1$). Let $u_{k_1} = x$ and $w_{k_1} = y$. Now, $x, y \in G_{k_1}$ and, as G_{k_1} is connected, there is an undirected path in G_{k_1} between x and y. This implies that there is an undirected path through k_1-edges from u to $u' = (u'_1, \ldots, u'_n)$, where $u'_k = u_k$, for all $k \neq k_1$ and $u'_{k_1} = w_{k_1} = y$. Now, we repeat this process with the pair of vertices u' and w. Notice that if k_2 is the first coordinate such that $u'_{k_2} \neq w_{k_2}$, then $k_2 > k_1$. As n is finite, this process eventually stops in a vertex $u^* = w$. Then, if we attach together the undirected paths from u to u', from u' to u'' and so on, we get an undirected path from u to w. $\square$

Theorem B.68. *A graph $G = (V, E)$ is connected if and only if $G \Vdash a \vee \blacklozenge a$.*

Proof. ($\Rightarrow$) Suppose that $G \not\Vdash a \vee \blacklozenge a$. Then, there is a valuation $\mathbf{V}$ and a vertex x such that $(G, \mathbf{V}), x \Vdash \neg a \wedge \blacksquare \neg a$. Let $\mathbf{V}(a) = y$. Then, we have that $x \neq y$ and that if there is an undirected path from x to a vertex z, then $z \neq y$. This means that G is not connected.

($\Leftarrow$) If $G \Vdash a \vee \blacklozenge a$, then $(G, \mathbf{V}), x \Vdash a \vee \blacklozenge a$ for every valuation $\mathbf{V}$ and every vertex x. Let $\mathbf{V}$ be such that $\mathbf{V}(a) = y \neq x$. Then, there is an undirected path between

x and y in G. So, as the vertex x and the valuation $\mathbf{V}$ are arbitrary, this means that we have an undirected path between every pair of distinct vertices. Hence, G is connected. □

Now, we address the second issue that prevents us from applying theorem B.66. For this, we need to state the notion of *compactness*.

Definition B.69 ([42])**.** *Let $L = Log(F, \mathcal{C})$ be a semantically defined modal logic. We say that L is* compact *if a set Σ of formulas is $\mathcal{C}$-satisfiable if and only if every finite subset $\Sigma_0 \subseteq \Sigma$ is $\mathcal{C}$-satisfiable.*

We need to add the modalities $\blacklozenge_i$ and $\Diamond_i^{-1}$ (as it is used in the definition of $\blacklozenge_i$) to the language, since the formulas that describe intransitivity and connectivity make use of them. However, when we add the modalities $\blacklozenge_i$ to the language, the modal logics lose compactness. For example, consider the set

$$\Sigma = \{\blacklozenge p, \neg\Diamond p, \neg\Diamond^{-1} p, \neg\Diamond\Diamond p, \neg\Diamond\Diamond^{-1} p, \neg\Diamond^{-1}\Diamond p, \neg\Diamond^{-1}\Diamond^{-1} p, \ldots\}.$$

Every finite set $\Sigma_0 \subseteq \Sigma$ is $\mathcal{C}$-satisfiable, where $\mathcal{C}$ in this case is the class of all frames. But Σ is not satisfiable in any vertex of any model of any frame.

The proof of completeness using canonical models gives strong completeness as its result (see theorems B.65 and B.66, for instance). The following theorem shows that if we have a sound and strongly complete axiomatic system for a logic $L = \mathrm{Log}(F, \mathcal{C})$, then L is compact.

Theorem B.70. *Let $L = Log(F, \mathcal{C})$ be a semantically defined modal logic and $\mathbb{A}$ an axiomatic system. If $\mathbb{A}$ is sound and strongly complete for L, then L is compact.*

Proof. First, we should notice that if a set Σ is $\mathcal{C}$-satisfiable, then trivially every finite subset $\Sigma_0 \subseteq \Sigma$ is also $\mathcal{C}$-satisfiable. So, in order to show compactness we only need to prove the other direction.

If every finite subset $\Sigma_0 \subseteq \Sigma$ is $\mathcal{C}$-satisfiable, then, by soundness, every finite subset $\Sigma_0 \subseteq \Sigma$ is $\mathbb{A}$-consistent. Then, by definition B.58, this means that Σ is $\mathbb{A}$-consistent. Then, by strong completeness Σ is $\mathcal{C}$-satisfiable. □

So, if a logic L is not compact, as the final result of completeness proofs using canonical models is strong completeness, such a proof will fail for L. When the

logic is not compact, sometimes it is still possible to build a completeness (but not strong completeness) proof using *finite canonical models* that are built from maximal consistent subsets of a finite set. However, for a completeness proof using finite canonical models to be successful, the logic must have the *finite model property*, i.e., every satisfiable formula must be satisfied in a vertex of a model of a *finite* frame. In our case, as we also need the canonical model to be a product, we would actually need the logic to have the so-called *product finite model property.*

Definition B.71. *A product of modal logics has the* product finite model property *if every satisfiable formula is satisfied in a vertex of a model of a* finite *product frame.*

As the following theorem from [41] shows, many products of modal logics of dimension greater than 2 do not have the product finite model property.

Theorem B.72 ([41])**.** *Let $n \geq 3$ and let $L \subseteq MFor_n$ be any modal logic such that $\mathbf{K}^n \subseteq L \subseteq \mathbf{S5}^n$. Then, L lacks the product finite model property.*

As $MFor_n \subset HFor_n$, the corollary below is immediate.

Corollary B.73. *Let $n \geq 3$ and let $L \subseteq HFor_n$ be any modal logic such that $\mathbf{K}(@)^n \subseteq L \subseteq \mathbf{S5}(@)^n$. Then, L lacks the product finite model property.*

Then, we need to find a way to express intransitivity and connectivity without the modalities $\blacklozenge$, so we can have compactness back and use theorem B.66 to derive the completeness proofs. Let us restrict our attention to the class of transitive and symmetric frames. Over this class of frames, using symmetry, we have that $\Diamond_i^{-1}\phi \equiv \Diamond_i\phi$, which also implies (using transitivity) that $\blacklozenge_i\phi \equiv \Diamond_i\phi$. Besides that, the Church-Rosser property and the reverse Church-Rosser property are equivalent. So, over the class of transitive and symmetric frames, we can characterize the class of connected products with the pure formulas in figure B.7.

Now, to finally get the result we are looking for, all that is left to do is to characterize the class of transitive and symmetric frames using pure formulas. In [10], the necessary formulas are presented. We show them in figure B.8.

Theorem B.74. *Let P_{B4} be the set of pure formulas in figure B.8 and let $\mathbb{A}_{\mathbf{B4}(n,@)}$ be the axiomatic system $\mathbb{A}_{\mathbf{B4}(n,@)} = \mathbb{A}_{\mathbf{K}(n,@)} + P_{B4}$. The axiomatic system $\mathbb{A}_{\mathbf{B4}(n,@)}$ is sound and complete for the logic $\mathbf{B4}(n, @)$.*

(Con$_i$) $a \vee \Diamond_i a$

(Com$_{ij}$) $\Diamond_j \Diamond_i a \leftrightarrow \Diamond_i \Diamond_j a$

(Chr$_{ij}$) $\Diamond_j a \rightarrow \Box_i \Diamond_j \Diamond_i a$

(Int$_{ij}$) $(a \wedge \neg b \wedge \Diamond_j (b \wedge \neg c \wedge \Diamond_i c)) \rightarrow \neg c$

Figure B.7: Pure formulas that characterize the class of connected products

(4$_i$) $\Diamond_i \Diamond_i a \rightarrow \Diamond_i a$ (transitivity)

(B$_i$) $a \rightarrow \Box_i \Diamond_i a$ (symmetry)

Figure B.8: Pure formulas that characterize the class of transitive and symmetric frames

Proof. This follows directly from theorem B.66. □

Definition B.75. *Let $L = Log(F, \mathcal{C})$ be a semantically defined logic. Then, we denote by $\mathbf{C}L$ the logic defined by the class of connected frames in $\mathcal{C}$.*

Theorem B.76. *Let P_{Prod} be the set of pure formulas in figure B.7 and let $\mathbb{P}$ be the axiomatic system $\mathbb{P} = \mathbb{A}_{\mathbf{B4}(n,@)} + P_{Prod}$. The axiomatic system $\mathbb{P}$ is sound and complete for the logic $\mathbf{CB4}(@)^n$.*

Proof. This follows directly from theorem B.66. □

Theorem B.77. *Let $L = L_1 \times \cdots \times L_n$ be a product of modal logics. If $\mathbf{CB4}(@) \subseteq L_i$, for all $1 \leq i \leq n$, and $\mathbb{A}_i = \mathbb{A}_{\mathbf{B4}(n,@)} + P_i$, where P_i is a finite set of pure formulas, is a sound and complete axiomatic system for L_i, then $\mathbb{A} = \mathbb{A}_{\mathbf{B4}(n,@)} + \sum_i P_i + P_{Prod}$ is a sound and complete axiomatic system for L.*

Proof. This follows directly from theorem B.66. □

As examples of properties that can be defined by pure formulas, we can mention reflexivity, irreflexivity, density, determinism, universality and trichotomy. Many of these properties cannot be defined by formulas in the basic modal language. So, theorem B.77 provides sound and complete axiomatic systems for a large family of

products of modal logics of arbitrary dimensions, while most of the results presented in the literature so far deal with bidimensional products.

As a last comment, what can we say about products of modal logics over the basic modal language, given the axiomatic systems that we built for products of modal logics over the hybrid language? First, as connectivity is not definable in the basic modal language [29], it is straightforward to see that $L = \mathbf{C}L$, if L is a modal logic over the basic modal language. Also, as $MFor_n \subset HFor_n$, $L \subset L(@)$. Then, $L_1 \times \cdots \times L_n \subset \mathbf{C}L_1(@) \times \cdots \times \mathbf{C}L_n(@)$, which means that if we have a sound and complete axiomatic system $\mathbb{A}$ for $\mathbf{C}L_1(@) \times \cdots \times \mathbf{C}L_n(@)$, then all formulas $\phi \in L_1 \times \cdots \times L_n$ are also theorems of $\mathbb{A}$ ($\vdash_{\mathbb{A}} \phi$). As an example, we can get a sound and complete axiomatic system for $\mathbf{CS5}(@)^n$ using theorem B.77. So, even tough $\mathbf{S5}^n$ cannot be finitely axiomatized over the basic modal language, all of its formulas are deductible from the axiomatization of $\mathbf{CS5}(@)^n$. In fact, these formulas are exactly the subset of $\mathbf{CS5}(@)^n$ that contains only the formulas without nominals and satisfaction operators.

B.8 Conclusion

It is known that left and right commutativity and the Church-Rosser and reverse Church-Rosser properties are necessary conditions for a graph to be a non-trivial cartesian product of two other graphs, but their conjunction is not a sufficient condition. We introduce a new property called intransitivity, that, together with the former ones, form a necessary and sufficient condition for a countable and connected graph to be a product. We show that the property of intransitivity is not definable in a basic modal language. We also show that no condition that is necessary and sufficient for a graph to be a product can be definable in a basic modal language. We then extend our language to a hybrid language and show that in such a language we are able to express intransitivity.

We also determine the computational complexity of testing, for a finite connected graph, whether it is a product. For this test, we use a model-checking algorithm to verify the formulas that describe each of the five properties that characterize a product. We show that this test can be performed in polynomial time both in the

size of the graph (cubic time) and in the number of dimensions (quadratic time).

Finally, we use the above characterization of countable connected products to provide sound and complete axiomatic systems for a large class of products of modal logics. While most of the sound and complete axiomatic systems for products of modal logics presented in the literature are for products of a pair of modal logics, we are able, using hybrid logics, to provide sound and complete axiomatizations for many products of arbitrary dimensions.

The logic $\mathbf{S5}^n$ is related to the field of cylindric algebras [53, 54, 55]: the modal algebras corresponding to $\mathbf{S5}^n$ are the representable diagonal-free cylindric algebras of dimension n. As a future work, it would be interesting to analyze whether the axiomatization that we can provide for $\mathbf{S5}^n$ using the results in section B.7 could be useful from the algebraic point of view.

In [56], products of logics are studied from the point of view of spatial and topological logics. In the last chapter of [56], the author develops a preliminary work on the possible use of hybrid logic for the study of spatial and topological logics and their products. As another future work, it would be interesting to analyze whether our results are useful or meaningful for the case of spatial and topological logics.

Apêndice C

A Propositional Dynamic Logic for CCS Programs

C.1 Introduction[1]

Propositional Dynamic Logic (PDL) plays an important role in formal specification and reasoning about programs and actions. PDL is a multi-modal logic with one modality $\langle\pi\rangle$ for each program π. The logic has a finite set of basic programs and a set of operators (sequential composition, iteration and nondeterministic choice) that can be used to build more complex programs from simpler ones. PDL has been used in formal specification to reason about properties of programs and their behaviour. Correctness, termination, fairness, liveness and equivalence of programs are among the properties that one usually wants to verify. A Kripke semantics can be provided, with a frame $\mathcal{F} = \langle W, R_\pi \rangle$, where W is a set of possible program states and, for each program π, R_π is a binary relation on W such that $(s,t) \in R_\pi$ if and only if there is a computation of π starting in s and terminating in t.

The Calculus for Communicating Systems (CCS) is a well known process algebra, proposed by Robin Milner [21], for the specification of concurrent systems. It models the concurrency and interaction between processes through individual acts of communication. A pair of processes can communicate through a common channel and each act of communication consists simply of a signal being sent at one end of

[1]Este capítulo expõe o conteúdo do artigo [43], publicado nos anais do congresso WoLLIC 2008, Lecture Notes in Artificial Intelligence, v. 5110, p. 83-97, 2008.

the channel and (immediately) being received at the other. A CCS specification is a description (in the form of algebraic equations) of the behaviour expected from a system, based on the communication events that may occur. As in PDL, CCS has a set of operators (action prefix, parallel composition, nondeterministic choice and restriction on acts of communication) to build more complex specifications from simpler ones. Iteration can also be described through the use of recursive equations.

This work presents a Propositional Dynamic Logic in which the programs are CCS terms (CCS-PDL). Its goal is to reason about properties of concurrent systems specified in CCS. The idea is to bring together the logical and the algebraic formalisms. Quoting from Milner [21]:

> "The calculus of this book is indeed largely algebraic, but I make no claim that everything can be done by algebra. (...) It is perhaps equally true that not everything can be done by logic; thus one of the outstanding challenges in concurrency is to find the right marriage between logical and behavioural approaches."

CCS-PDL is related to Concurrent PDL (with channels) [7, 6] and the logic developed in [8]. Both of these logics are expressive enough to represent interesting properties of concurrent systems. However, neither of them has a simple Kripke semantics. The first has a semantics based on *super-states and super-processes* and its satisfiability problem can be proved undecidable (in fact, it is Π^1_1-hard) [7]. Also, it does not have a complete axiomatization [7]. The second makes a semantic distinction between *final* and *non-final* states, which makes its semantics and its axiomatization rather complex. On the other hand, due to the full use of the CCS mechanism of communication, CCS-PDL has a simple Kripke semantics and the finite model property.

The rest of this paper is organized as follows. In section C.2, we introduce the necessary background concepts: Propositional Dynamic Logic and the Calculus for Communicating Systems. A first version of our logic, together with an axiomatic system, is presented in section C.3. In this preliminary version, we do not use constants in the CCS processes. In section C.4, we present the full logic, in which we allow the presence of constants in the CCS processes. We also give an axiomatization for this second logic and prove its completeness using a Fischer-Ladner construction.

Finally, in section C.5, we state our final remarks.

C.2 Background

This section presents two important subjects. First, we make a brief review of the syntax and semantics of PDL. Second, we present the process algebra CCS and the Expansion Law, which is closely related to the way we deal with the parallel composition operator.

C.2.1 Propositional Dynamic Logic

In this section, we present the syntax and semantics of PDL.

Definition C.1. *The PDL language consists of a set* Φ *of countably many propositional symbols, a finite set of basic programs* Π*, the boolean connectives* $\neg$ *and* $\wedge$*, the program constructors* $;$*,* $\cup$ *and* $*$ *and a modality* $\langle\pi\rangle$ *for every program* π*. The formulas are defined as follows:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\pi\rangle\varphi, \text{ with } \pi ::= a \mid \pi_1;\pi_2 \mid \pi_1 \cup \pi_2 \mid \pi^* \ ,$$

where $p \in \Phi$ *and* $a \in \Pi$*.*

In this logic and in every other logic defined in this paper, we use the standard abbreviations $\bot \equiv \neg\top$, $\varphi\vee\phi \equiv \neg(\neg\varphi\wedge\neg\phi)$, $\varphi \rightarrow \phi \equiv \neg(\varphi\wedge\neg\phi)$ and $[\pi]\varphi \equiv \neg\langle\pi\rangle\neg\varphi$.

Definition C.2. *A* frame *for PDL is a tuple* $\mathcal{F} = (W, R_a, R_\pi)$ *where*

- W *is a non-empty set of states;*
- R_a *is a binary relation for each basic program* a*;*
- R_π *is a binary relation for each non-basic program* π*, inductively built using the rules* $R_{\pi_1;\pi_2} = R_{\pi_1} \circ R_{\pi_2}$*,* $R_{\pi_1\cup\pi_2} = R_{\pi_1} \cup R_{\pi_2}$ *and* $R_{\pi^*} = R^*_\pi$*, where* R^*_π *denotes the reflexive transitive closure of* R_π*.*

Definition C.3. *A* model *for PDL is a pair* $\mathcal{M} = (\mathcal{F}, \mathbf{V})$*, where* $\mathcal{F}$ *is a PDL frame and* $\mathbf{V}$ *is a valuation function* $\mathbf{V} : \Phi \mapsto 2^W$*.*

The semantical notion of satisfaction for PDL is defined as follows:

Definition C.4. *Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. The notion of* satisfaction *of a formula φ in a model $\mathcal{M}$ at a state w, notation $\mathcal{M}, w \Vdash \varphi$, can be inductively defined as follows:*

- $\mathcal{M}, w \Vdash p$ *iff* $w \in \mathbf{V}(p)$;
- $\mathcal{M}, w \Vdash \top$ *always;*
- $\mathcal{M}, w \Vdash \neg\varphi$ *iff* $\mathcal{M}, w \nVdash \varphi$;
- $\mathcal{M}, w \Vdash \varphi_1 \wedge \varphi_2$ *iff* $\mathcal{M}, w \Vdash \varphi_1$ *and* $\mathcal{M}, w \Vdash \varphi_2$;
- $\mathcal{M}, w \Vdash \langle\pi\rangle\varphi$ *iff there is* $w' \in W$ *such that* $wR_\pi w'$ *and* $\mathcal{M}, w' \Vdash \varphi$.

C.2.2 Calculus for Communicating Systems

The Calculus for Communicating Systems (CCS) is a well known process algebra, proposed by Robin Milner [21], for the specification of concurrent systems. It models the concurrency and interaction between processes through individual actions of communication. A CCS specification is a description (in the form of algebraic equations) of the behaviour expected from a system, based on the communication events that may occur.

In CCS, a pair of processes can communicate through a common channel and each act of communication consists simply of a signal being sent at one end of the channel and (immediately) being received at the other. Each process connects itself to a channel through a port.

Let $\mathcal{N} = \{a, b, c, \ldots\}$ be a set of names and $\overline{\mathcal{N}} = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}$ be the correspondent set of co-names. Each port in a CCS specification is labelled by either a name or a co-name. Using the convention that $\overline{\overline{a}} = a$, if a port connected to an end of a channel is labelled with $\alpha \in \mathcal{N} \cup \overline{\mathcal{N}}$, then a port connected to the other end of this same channel must be labelled with $\overline{\alpha}$. Moreover, two ports of a process cannot have the same label. Channels in CCS can only transmit signals in one direction. By convention, signals are sent through the ports labelled by co-names and received through ports labelled by names.

The labels of the ports are also used to describe the communication actions performed by the processes. For example, if a process is connected to a channel

through port $\overline{\alpha}$, the action of sending a signal into this channel is also denoted by $\overline{\alpha}$. So, in terms of actions, α means that the process receives a signal through the port labelled with α and $\overline{\alpha}$ means that the process sends a signal through the port labelled with $\overline{\alpha}$. A process can never perform a communication action α without its complementary action $\overline{\alpha}$ also being performed at the same time by some other process.

Besides the actions denoted by the elements of $\mathcal{N} \cup \overline{\mathcal{N}}$, CCS admits only one other action: the silent action, denoted by τ. The silent action is used to represent any internal action performed by any of the processes that does not involve any act of communication (e.g.: a memory update). Thus, we have that the set of CCS actions is $\mathcal{A} = \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$.

There are two different ways in CCS to consider the τ action: it can be regarded as being observable, in the same way as the communication actions, or it can be regarded as being invisible. We will adopt the first point of view, since it simplifies the semantics considerably.

In CCS, process specifications can be built using the following operations ($\alpha \in \mathcal{A}$ is an action, P, P_1 and P_2 are process specifications, A is a constant and $L \subseteq \mathcal{N}$):

$$P ::= \alpha \mid \alpha.P \mid \alpha.A \mid P_1 + P_2 \mid P_1|P_2 \mid P\backslash L \ ,^2$$

where every constant A has a (unique) *defining equation* $A \stackrel{def}{=} P_A$, where P_A is a process specification. In this work, every time that a process is linked to a constant A through a defining equation, it will be denoted by P_A.

The *prefix* operator (.) denotes that the process will first perform the action α and then behave as P or A. The *summation* (or *nondeterministic choice*) operator (+) denotes that the process will make a nondeterministic choice to behave as either P_1 or P_2. The *parallel composition* operator (|) denotes that the processes P_1 and P_2 may proceed independently or may communicate through a pair of complementary ports (one performing an action α and the other $\overline{\alpha}$). Finally, the *restriction* operator ($\backslash$) denotes that for all ports α such that $\alpha \in L$ or $\overline{\alpha} \in L$, α is unreachable outside P. Iteration in CCS is modeled through recursive defining equations, i.e., equations $A \stackrel{def}{=} P_A$ where A occurs in P_A.

[2]Originally, CCS has also a relabelling operator.

Table C.1: Transition Relations of CCS

Action	Prefix	Constant	Summation (1)
$\frac{}{\alpha \xrightarrow{\alpha} \mathbf{0}}$	$\frac{}{\alpha.P \xrightarrow{\alpha} P}$	$\frac{A \stackrel{def}{=} P_A}{\alpha.A \xrightarrow{\alpha} P_A}$	$\frac{E \xrightarrow{\alpha} E'}{E+F \xrightarrow{\alpha} E'}$
Summation (2)	Restriction	Par. Comp. (1)	Par. Comp. (2).
$\frac{F \xrightarrow{\beta} F'}{E+F \xrightarrow{\beta} F'}$	$\frac{E \xrightarrow{\alpha} E', \alpha, \overline{\alpha} \notin L}{E \backslash L \xrightarrow{\alpha} E' \backslash L}$	$\frac{E \xrightarrow{\alpha} E'}{E \mid F \xrightarrow{\alpha} E' \mid F}$	$\frac{F \xrightarrow{\beta} F'}{E \mid F \xrightarrow{\beta} E \mid F'}$
Par. Comp. (3)			
$\frac{E \xrightarrow{\lambda} E', F \xrightarrow{\overline{\lambda}} F'}{E \mid F \xrightarrow{\tau} E' \mid F'}$			

We write $P \xrightarrow{\alpha} P'$ to express that the process P can perform the action α and after that behave as P'. We write $P \xrightarrow{\alpha} \mathbf{0}$ to express that the process P finishes after performing the action α. A process only finishes when there is not any possible action left for it to perform. For example, $\beta \xrightarrow{\beta} \mathbf{0}$. When a process finishes inside a parallel composition, we write P instead of $P|\mathbf{0}$. In table C.1, we present the semantics for the operators based on this notation.

The notion of equality between process specifications is defined through the concept of *strong bisimulation*.

Definition C.5. *Let $\mathcal{P}$ be the set of all possible process specifications. A set $Z \subseteq \mathcal{P} \times \mathcal{P}$ is a* strong bisimulation *if $(P, Q) \in Z$ implies, for all $\alpha \in \mathcal{A}$,*

- *Whenever $P \xrightarrow{\alpha} P'$ then, for some Q', $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in Z$*
- *Whenever $Q \xrightarrow{\alpha} Q'$ then, for some P', $P \xrightarrow{\alpha} P'$ and $(P', Q') \in Z$*

Definition C.6. *Two process specifications P and Q are* strongly bisimilar *(or simply* bisimilar*), denoted by $P \sim Q$, if there is a strong bisimulation Z such that $(P, Q) \in Z$. Two process specifications are considered "equal" if they are bisimilar.*

Bisimilarity is preserved by all of CCS operators:

Theorem C.7 ([21])**.** *Let $P_1 \sim P_2$. Then*

1. *$\alpha.P_1 \sim \alpha.P_2$;*
2. *$P_1 + Q \sim P_2 + Q$;*
3. *$P_1|Q \sim P_2|Q$;*

4. $P_1 \backslash L \sim P_2 \backslash L$.

A very useful property of CCS processes is that they can be rewritten as a summation of all their possible actions. This is what states the Expansion Law below. This theorem will be very important in the definition of the semantics of our logic in the next section.

Theorem C.8 (Expansion Law [21])**.** *Let* $P = (P_1 \mid P_2)$*. Then*

$$P \sim \sum\{\alpha.(P_1' \mid P_2) : P_1 \xrightarrow{\alpha} P_1'\} + \sum\{\alpha.(P_1 \mid P_2') : P_2 \xrightarrow{\alpha} P_2'\} +$$

$$+ \sum\{\tau.(P_1' \mid P_2') : P_1 \xrightarrow{\lambda} P_1', P_2 \xrightarrow{\bar{\lambda}} P_2'\} \ .$$

In the rest of this paper, we consider that the restriction operator does not occur in the processes. In order to motivate the use of CCS, we present a simple example below:

Example C.9 (Vending Machine [21, 23])**.** *Consider a vending machine where one can put coins of one or two euro and buy a little or a big chocolate bar. After inserting the coins, one must press the little button for a little chocolate or the big button for a big chocolate. The machine is also programmed to shutdown on its own following some internal logic (represented by a* τ *action). A CCS term describing the behaviour of this machine is the following:*

$$V = 1e.little.\overline{collect}.A + 1e.1e.big.\overline{collect}.A + 2e.big.\overline{collect}.A$$

$$A \stackrel{def}{=} 1e.little.\overline{collect}.A + 1e.1e.big.\overline{collect}.A + 2e.big.\overline{collect}.A + \tau$$

Let us now suppose that Chuck wants to use this vending machine. We could describe Chuck as

$$C = \overline{1e}.\overline{little}.collect + \overline{1e}.\overline{1e}.\overline{big}.collect + \overline{2e}.\overline{big}.collect \ .$$

Notice that Chuck does not have an iterative behaviour. Once he collects the chocolate, he is done. Now, if we want to model the process of Chuck buying a chocolate from the vending machine, we could write $(V|C)$*.*

C.3 PDL for CCS Programs without Constants

This section presents a suitable fragment of CCS-PDL. In this fragment, all CCS processes do not use constants. We call this fragment Small CCS-PDL or SCCS-PDL. Our goal is to introduce a smaller version of our logic and discuss some of the issues concerning the axioms and the relational interpretation of formulas.

C.3.1 Language and Semantics

Definition C.10. *The SCCS-PDL language consists of a set Φ of countably many propositional symbols, a finite set of actions $\mathcal{A}$ that includes the silent action τ, the boolean connectives $\neg$ and $\wedge$, the CCS operators ., + and | and a modality $\langle P \rangle$ for every process P. The formulas are defined as follows:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle P \rangle \varphi, \text{ with } P ::= \alpha \mid \alpha.P \mid P_1 + P_2 \mid P_1 | P_2 \ ,$$

where $p \in \Phi$ and $\alpha \in \mathcal{A}$.

Going back to the example of the vending machine, we could use this language to express that after the insertion of two one euro coins, the machine does not accept coins anymore. This can be expressed with the formula $[1e.1e][1e + 2e]\bot$.

Definition C.11. *We define the length of a process P, denoted by $||P||$, as the number of symbols in P. We also define the length of a finished process as 0.*

Theorem C.12. *If P is a process without any constants and $P \xrightarrow{\alpha} P'$, then $||P'|| < ||P||$.*

Definition C.13. *A* frame *for SCCS-PDL is a tuple $\mathcal{F} = (W, R_\alpha, R_P)$ where*

- *W is a non-empty set of states;*
- *R_α is a binary relation for each basic action α, including τ;*
- *R_P is a binary relation for each non-basic process P, inductively built as:*
  - *$R_{\alpha.P} = R_\alpha \circ R_P$;*
  - *$R_{(P_1+P_2)} = R_{P_1} \cup R_{P_2}$;*

$$- \; R_{(P_1|P_2)} = \bigcup_{P_1 \xrightarrow{\alpha} P_1'} R_\alpha \circ R_{(P_1'|P_2)} \cup \bigcup_{P_2 \xrightarrow{\alpha} P_2'} R_\alpha \circ R_{(P_1|P_2')} \cup \bigcup_{\substack{P_1 \xrightarrow{\lambda} P_1' \\ P_2 \xrightarrow{\overline{\lambda}} P_2'}} R_\tau \circ R_{(P_1'|P_2')}$$

It should be noticed that the Expansion Law stated in theorem C.8 is what allows us to define a simple relational semantic to the parallel composition operator. Besides that, theorem C.12 guarantees that we can fully define the relation R_P in terms of the relations R_α, for any process P, applying the above rules a finite number of times.

Definition C.14. *A* model *for SCCS-PDL is a pair $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, where $\mathcal{F}$ is a SCCS-PDL frame and $\mathbf{V}$ is a valuation function $\mathbf{V} : \Phi \mapsto 2^W$.*

The semantical notion of satisfaction for SCCS-PDL is defined as follows:

Definition C.15. *Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. The notion of* satisfaction *of a formula φ in a model $\mathcal{M}$ at a state w, notation $\mathcal{M}, w \Vdash \varphi$, can be inductively defined as follows:*

- $\mathcal{M}, w \Vdash p$ *iff* $w \in \mathbf{V}(p)$;
- $\mathcal{M}, w \Vdash \top$ *always;*
- $\mathcal{M}, w \Vdash \neg\varphi$ *iff* $\mathcal{M}, w \nVdash \varphi$;
- $\mathcal{M}, w \Vdash \varphi_1 \wedge \varphi_2$ *iff* $\mathcal{M}, w \Vdash \varphi_1$ *and* $\mathcal{M}, w \Vdash \varphi_2$;
- $\mathcal{M}, w \Vdash \langle P \rangle \varphi$ *iff there is* $w' \in W$ *such that* $w R_P w'$ *and* $\mathcal{M}, w' \Vdash \varphi$.

If $\mathcal{M}, w \Vdash \varphi$ for every state w, we say that φ is *globally satisfied* in the model $\mathcal{M}$, notation $\mathcal{M} \Vdash \varphi$. If φ is globally satisfied in all models $\mathcal{M}$ of a frame $\mathcal{F}$, we say that φ is *valid* in $\mathcal{F}$, notation $\mathcal{F} \Vdash \varphi$. Finally, if φ is valid in all frames, we say that φ is valid, notation $\Vdash \varphi$.

C.3.2 Proof Theory

We consider the following set of axioms and rules, where p and q are proposition symbols and φ and ψ are formulas.

(PL) Enough propositional logic tautologies

(K) $[P](p \to q) \to ([P]p \to [P]q)$

(Pr) $\langle \alpha.P \rangle p \leftrightarrow \langle \alpha \rangle \langle P \rangle p$,

(NC) $\langle P_1 + P_2 \rangle p \leftrightarrow \langle P_1 \rangle p \vee \langle P_2 \rangle p$

(PC) $\langle P_1 \mid P_2 \rangle p \leftrightarrow \bigvee_{P_1 \xrightarrow{\alpha} P_1'} \langle \alpha \rangle \langle P_1' \mid P_2 \rangle p \vee \bigvee_{P_2 \xrightarrow{\alpha} P_2'} \langle \alpha \rangle \langle P_1 \mid P_2' \rangle p \vee \vee \bigvee_{\substack{P_1 \xrightarrow{\lambda} P_1' \\ P_2 \xrightarrow{\overline{\lambda}} P_2'}} \langle \tau \rangle \langle P_1' \mid P_2' \rangle p$

(Sub) If $\vdash \varphi$, then $\vdash \varphi^\sigma$, where σ uniformly substitutes proposition symbols by arbitrary formulas.

(MP) If $\vdash \varphi$ and $\vdash \varphi \to \psi$, then $\vdash \psi$.

(Gen) If $\vdash \varphi$, then $\vdash [P]\varphi$.

These axioms are closely related to the conditions imposed in Definition C.13. The second axiom is the K axiom for modal logics. The third and fourth axioms are well-known from PDL literature and define sequential composition and choice operators respectively. The fifth axiom corresponds to the Expansion Law for the parallel composition operator.

The logic presented in this section is sound and complete w.r.t. the class of frames described in Definition C.13. It also has the finite model property. We omit the proofs here, because they are analogous to the proofs presented in section C.4, where constants are added to the language.

C.4 PDL for CCS Programs

The logic presented in this section uses the same CCS operators as in the previous section plus constants. This is the full CCS-PDL logic. Our goal in this section is to build an axiomatic system to CCS-PDL and prove its completeness.

C.4.1 Language and Semantics

Definition C.16. *The CCS-PDL language consists of a set Φ of countably many propositional symbols, a finite set of actions $\mathcal{A}$ that includes the silent action τ,*

the boolean connectives $\neg$ *and* $\wedge$*, the CCS operators* $.$*,*$+$ *and* $|$*, constants, with its correspondent defining equations, and a modality* $\langle P \rangle$ *for every process* P*. The formulas are defined as follows:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle P \rangle \varphi, \text{ with } P ::= \alpha \mid \alpha.P \mid \alpha.A \mid P_1 + P_2 \mid P_1 | P_2 \ ,$$

where $p \in \Phi$ *and* $\alpha \in \mathcal{A}$*.*

Let P be a process and $\{A_1, \ldots, A_n\}$ be the constants that occur in P. We define $Cons(P)$ as the smallest set of constants such that $Cons(P) \supseteq \{A_1, \ldots, A_n\}$ and, for every constant $A_i \in Cons(P)$, if A_k occurs in P_{A_i}, then $A_k \in Cons(P)$. We make the restriction that $Cons(P)$ must be a finite set for every process P.

Another restriction concerns the construction of defining equations. We only allow defining equations that fit into one of the following models: $A \stackrel{def}{=} P_A$, where $A \notin Cons(P_A)$, called *non-recursive equations*, or $A \stackrel{def}{=} \overrightarrow{\alpha}_1.A + \ldots + \overrightarrow{\alpha}_n.A + T_A$, called *recursive equations*, where $\overrightarrow{\alpha}_i$ denotes a sequence of actions $\alpha_1^i.\alpha_2^i \ldots \alpha_{m_i}^i$ and $A \notin Cons(T_A)$.

Definition C.17. *We say that a process* P *is a* looping process *if* $P = P_A$ *for some constant* A *with a recursive defining equation or if* $P = P_1 \mid P_2$ *where* P_1 *or* P_2 *is a looping process. Otherwise, we say that* P *is a* non-looping process*.*

Definition C.18. *We write* $P \stackrel{\vec{\alpha}}{\Rightarrow} P'$ *to express that the process* P *can perform the sequence of actions* $\overrightarrow{\alpha}$ *and after that behave as* P'*. We write* $P \stackrel{\vec{\alpha}}{\Rightarrow} \mathbf{0}$ *to express that the process* P *finishes after performing the sequence of actions* $\overrightarrow{\alpha}$*.*

Definition C.19. *We call a sequence* $\overrightarrow{\alpha}$ *a* breaker *for a looping process* P *if* $P = P_A$*,* $P \stackrel{\vec{\alpha}}{\Rightarrow} P'$ *and* $A \notin Cons(P')$ *(or* $P \stackrel{\vec{\alpha}}{\Rightarrow} \mathbf{0}$*) or if* $P = P_1 \mid P_2$*,* P_i *(*$i \in \{1, 2\}$*) is a looping process and* $\overrightarrow{\alpha}$ *is a breaker for* P_i*. These sequences are called breakers because after performing* $\overrightarrow{\alpha}$*, there is no sequence of actions* $\overrightarrow{\alpha}'$ *such that* $P' \stackrel{\vec{\alpha}'}{\Rightarrow} P$*, i. e., the loop around* P *is broken by* $\overrightarrow{\alpha}$*.*

Using the concept of a breaker, we can split a looping process P into two parts: the *looping part*, denoted by L_P, and the *tail part*, denoted by T_P. Informally, L_P describes one loop around P and T_P describes the behaviour of P when it stops looping. Let $Br(P)$ be the set of all breakers for P. Then,

$$T_P = \sum \{ \overrightarrow{\alpha}.P' : \overrightarrow{\alpha} \in Br(P), P \stackrel{\vec{\alpha}}{\Rightarrow} P' \text{ and } \forall \overrightarrow{\alpha}' \subsetneq \overrightarrow{\alpha}, P \stackrel{\vec{\alpha}'}{\not\Rightarrow} P \} \text{ and}$$

$$L_P = \sum\{\overrightarrow{\alpha} : P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} P \text{ and } \forall \overrightarrow{\alpha}' \subsetneq \overrightarrow{\alpha}, P \stackrel{\overrightarrow{\alpha}'}{\not\Rightarrow} P\} \ .$$

If $P = P_A = \overrightarrow{\alpha}_1.A + \ldots + \overrightarrow{\alpha}_n.A + T_A$, it is not difficult to see that $L_P = \overrightarrow{\alpha}_1 + \ldots + \overrightarrow{\alpha}_n$ and $T_P = T_A$. We also define the process L'_P, that describes one or more loops around P, as $L'_P = \sum\{\overrightarrow{\alpha}.Z_P : P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} P \text{ and } \forall \overrightarrow{\alpha}' \subsetneq \overrightarrow{\alpha}, P \stackrel{\overrightarrow{\alpha}'}{\not\Rightarrow} P\} + L_P$, where Z_P is a new constant with defining equation $Z_P \stackrel{def}{=} L'_P$. It is important to notice that $Cons(L_P) \subsetneq Cons(P)$ and $Cons(T_P) \subsetneq Cons(P)$.

Definition C.20. *We say that a process* can unfold a constant *if it has the form $\alpha.A$ or if it has the form $P_1 + P_2$ or $P_1 \mid P_2$, where P_1 or P_2 can unfold a constant.*

Theorem C.21. *If P is a process that cannot unfold a constant and $P \stackrel{\alpha}{\rightarrow} P'$, then $||P'|| < ||P||$.*

Definition C.22. *A* **frame** *for CCS-PDL is a tuple $\mathcal{F} = (W, R_\alpha, R_P)$ where:*

- *W is a non-empty set of states;*
- *R_α is a binary relation for each basic action α, including τ;*
- *R_P is a binary relation for each non-basic process P, inductively built as:*
  - *For* **looping processes***: $R_P = R^*_{L_P} \circ R_{T_P}$*
  - *For* **non-looping processes***:*
    - *$R_{\alpha.P}$, $R_{P_1+P_2}$ and $R_{P_1|P_2}$ as in definition C.13;*
    - *$R_{\alpha.A} = R_\alpha \circ R_{P_A}$*

It should be noticed that the restriction on the set $Cons(P)$ and the restriction on the formation of the defining equations, both presented in the beginning of the section, together with the definition of the relation R_P for looping processes P based on R_{L_P} and R_{T_P} make it impossible for the relations to be able to unfold constants forever. This, together with theorem C.21, guarantee that we can fully define the relation R_P in terms of the relations R_α, for any process P, applying the above rules a finite number of times and that we can build well-founded proofs by induction on the structure of a process P.

The notions of models and satisfaction are defined analogously to Definitions C.14 and C.15.

C.4.2 Proof Theory

The proof theory is similar to the one presented in section C.3.2. We consider the following set of axioms and rules, where p, q and r are proposition symbols and φ and ψ are formulas.

- The axioms **(PL)** and **(K)** and the rules **(Sub)**, **(MP)** and **(Gen)**.
- Axioms for looping processes:

  (Rec) $\langle P\rangle p \leftrightarrow \langle T_P\rangle p \vee \langle L_P\rangle\langle P\rangle p$

  (FP) $(r \rightarrow ([T_P]\neg p \wedge [L_P]r)) \wedge [L'_P](r \rightarrow ([T_P]\neg p \wedge [L_P]r)) \rightarrow (r \rightarrow [P]\neg p)$
- Axioms for non-looping processes:

  (SCCS) The axioms **(Pr)**, **(NC)** and **(PC)**.

  (Cons) $\langle \alpha.A\rangle p \leftrightarrow \langle \alpha\rangle\langle P_A\rangle p$

The proof of soundness is analogous to the proof of soundness for PDL and CTL, as all of our axioms are very closely related to axioms for these logics.

Theorem C.23 (Completeness)**.** *Every consistent formula is satisfiable in a finite model that respects definition C.22.*

Proof. See section C.6. □

C.5 Final Remarks and Future Work

In this work, we present a PDL-like logic in which the programs are CCS terms (CCS-PDL). We provide a simple Kripke semantics for it and also give an axiomatization for this logic. We prove the completeness of the axiomatic system and the finite model property for the logic using a Fischer-Ladner construction.

As a continuation of this work, it would be interesting to study the complexity of the satisfiability problem for this logic, possibly relating it to the satisfiability problem for standard PDL.

We would also like to investigate some extension of CCS-PDL to deal with the restriction operator and a PDL for π-Calculus programs [24]. It would be interesting

to develop an automatic theorem prover for CCS-PDL. This would involve, among other things, efficient algorithmic methods to determine the processes L_P and T_P related to a looping process P and to deal with the expansion of parallel processes.

C.6 Completeness Proof

Definition C.24. *Let ϕ be a formula. We define the formula $\overline{\phi}$ as $\overline{\phi} = \psi$, if $\phi = \neg\psi$, or $\overline{\phi} = \neg\phi$, otherwise.*

Definition C.25 (Fischer-Ladner Closure)**.** *Let Γ be a set of formulas. The* Fischer-Ladner Closure *of Γ, notation $C(\Gamma)$, is the smallest set of formulas that contains Γ and satisfies the following conditions:*

- *$C(\Gamma)$ is closed under sub-formulas;*
- *if $\phi \in C(\Gamma)$, then $\overline{\phi} \in C(\Gamma)$;*
- *For looping processes:*
  - *If $\langle P\rangle\varphi \in C(\Gamma)$, then $\langle T_P\rangle\varphi \vee \langle L_P\rangle\langle P\rangle\varphi \in C(\Gamma)$.*
- *For non-looping processes:*
  - *If $\langle \alpha.P\rangle\varphi \in C(\Gamma)$, then $\langle\alpha\rangle\langle P\rangle\varphi \in C(\Gamma)$;*
  - *If $\langle \alpha.A\rangle\varphi \in C(\Gamma)$, then $\langle\alpha\rangle\langle P_A\rangle\varphi \in C(\Gamma)$;*
  - *If $\langle P_1 + P_2\rangle\varphi \in C(\Gamma)$, then $\langle P_1\rangle\varphi \vee \langle P_2\rangle\varphi \in C(\Gamma)$;*
  - *If $\langle P_1 \mid P_2\rangle\varphi \in C(\Gamma)$, then $\bigvee_{P_1 \xrightarrow{\alpha} P_1'}\langle\alpha\rangle\langle P_1' \mid P_2\rangle\varphi \vee \bigvee_{P_2 \xrightarrow{\alpha} P_2'}\langle\alpha\rangle\langle P_1 \mid P_2'\rangle\varphi \vee \bigvee_{\substack{P_1 \xrightarrow{\lambda} P_1' \\ P_2 \xrightarrow{\overline{\lambda}} P_2'}}\langle\tau\rangle\langle P_1' \mid P_2'\rangle\varphi \in C(\Gamma)$.*

It is not difficult to prove that if Γ is finite, then the closure $C(\Gamma)$ is also finite. We assume Γ to be finite from now on.

Definition C.26. *Every formula ϕ that is derivable from the set of axioms and rules in section C.4.2 is called a* theorem *and denoted as $\vdash \phi$. We say that a formula ϕ is* consistent *iff $\neg\phi$ is not a theorem, i.e., iff $\nvdash \neg\phi$ and* inconsistent *otherwise. A set of formulas $\Delta = \{\phi_1, \ldots, \phi_n\}$ is said to be consistent iff $\psi = \phi_1 \wedge \ldots \wedge \phi_n$ is consistent.*

Definition C.27. *A set of formulas $\mathcal{A}$ is said to be an* atom *over Γ if it is a maximal consistent subset of $C(\Gamma)$. The set of all atoms over Γ is denoted by $At(\Gamma)$. We denote the conjunction of all the formulas in an atom $\mathcal{A}$ as $\bigwedge \mathcal{A}$.*

Lemma C.28. *Every atom $\mathcal{A} \in At(\Gamma)$ has the following properties:*

1. *For every $\phi \in C(\Gamma)$, exactly one of ϕ and $\neg\phi$ belongs to $\mathcal{A}$.*
2. *For every $\phi \wedge \psi \in C(\Gamma)$, $\phi \wedge \psi \in \mathcal{A}$ iff $\phi \in \mathcal{A}$ and $\psi \in \mathcal{A}$.*

Proof. This follows immediately from the definition of atoms as maximal consistent subsets of $C(\Gamma)$. □

Lemma C.29. *If $\Delta \subseteq C(\Gamma)$ and Δ is consistent then there exists an atom $\mathcal{A} \in At(\Gamma)$ such that $\Delta \subseteq \mathcal{A}$.*

Proof. We can construct the atom $\mathcal{A}$ as follows. First, we enumerate the elements of $C(\Gamma)$ as $\phi_1, \ldots, \phi_n$. We start the construction making $\mathcal{A}_0 = \Delta$. Then, for $0 \leq i < n$, we know that $\bigwedge \mathcal{A}_i \leftrightarrow (\bigwedge \mathcal{A}_i \wedge \phi_{i+1}) \vee (\bigwedge \mathcal{A}_i \wedge \overline{\phi_{i+1}})$ is a tautology and therefore either $\mathcal{A}_i \cup \{\phi_{i+1}\}$ or $\mathcal{A}_i \cup \{\overline{\phi_{i+1}}\}$ is consistent. We take $\mathcal{A}_{i+1}$ as the consistent extension. At the end, we make $\mathcal{A} = \mathcal{A}_n$. □

Corollary C.30. *If $\varphi \in C(\Gamma)$ is a consistent formula, then there is an atom $\mathcal{A} \in At(\Gamma)$ such that $\varphi \in \mathcal{A}$.*

Definition C.31 (Canonical model over Γ)**.** *Let Γ be a finite set of formulas. The* canonical model *over Γ is the tuple $\mathcal{M}^\Gamma = (At(\Gamma), \{S_P\}, \mathbf{V})$ where, for all elements $p \in \Phi$, we have $\mathbf{V}(p) = \{\mathcal{A} \in At(\Gamma) \mid p \in \mathcal{A}\}$ and for all atoms $\mathcal{A}, \mathcal{B} \in At(\Gamma)$,*

$$\mathcal{A} S_P \mathcal{B} \quad \textit{iff} \quad \bigwedge \mathcal{A} \wedge \langle P \rangle \bigwedge \mathcal{B} \ \textit{is consistent} \ .$$

$\mathbf{V}$ is called the canonical valuation *and S_P the* canonical relations*, where P is a CCS process.*

Definition C.32 (CCS-PDL model over Γ)**.** *The* CCS-PDL model *over Γ is the tuple $\mathcal{N}^\Gamma = (At(\Gamma), \{R_P\}, \mathbf{V})$, where R_P is defined as $R_\alpha = S_\alpha$ for all basic processes α and according to definition C.22 for all complex processes. $\mathbf{V}$ is the canonical valuation.*

If $\Gamma = \{\varphi\}$, we write $C(\varphi)$, $At(\varphi)$, $\mathcal{M}^{\varphi}$ and $\mathcal{N}^{\varphi}$ instead of $C(\{\varphi\})$, $At(\{\varphi\})$, $\mathcal{M}^{\{\varphi\}}$ and $\mathcal{N}^{\{\varphi\}}$.

Lemma C.33 (Existence Lemma for Basic Processes)**.** *Let $\mathcal{A}$ be an atom, and let α be a basic process. Then, for all formulas $\langle\alpha\rangle\phi \in C(\Gamma)$, $\langle\alpha\rangle\phi \in \mathcal{A}$ iff there is a $\mathcal{B} \in At(\Gamma)$ such that $\mathcal{A}R_\alpha\mathcal{B}$ and $\phi \in \mathcal{B}$.*

Proof. ($\Rightarrow$) Suppose $\langle\alpha\rangle\phi \in \mathcal{A}$. We can build an appropriate atom $\mathcal{B}$ by forcing choices. Enumerate the formulas in $C(\Gamma)$ as $\phi_1, \ldots, \phi_n$. Define $\mathcal{B}_0 = \{\phi\}$. Suppose, as an inductive hypothesis that $\mathcal{B}_m$ is defined such that $\bigwedge \mathcal{A} \wedge \langle\alpha\rangle \bigwedge \mathcal{B}_m$ is consistent, for $0 \leq m < n$. We have that

$$\vdash \langle\alpha\rangle \bigwedge \mathcal{B}_m \leftrightarrow \langle\alpha\rangle((\bigwedge \mathcal{B}_m \wedge \phi_{m+1}) \vee (\bigwedge \mathcal{B}_m \wedge \overline{\phi_{m+1}})) \ ,$$

thus

$$\vdash \langle\alpha\rangle \bigwedge \mathcal{B}_m \leftrightarrow (\langle\alpha\rangle(\bigwedge \mathcal{B}_m \wedge \phi_{m+1}) \vee \langle\alpha\rangle(\bigwedge \mathcal{B}_m \wedge \overline{\phi_{m+1}})) \ .$$

Therefore, either for $\mathcal{B}' = \mathcal{B}_m \cup \{\phi_{m+1}\}$ or for $\mathcal{B}' = \mathcal{B}_m \cup \{\overline{\phi_{m+1}}\}$, we have that $\bigwedge \mathcal{A} \wedge \langle\alpha\rangle \bigwedge \mathcal{B}'$ is consistent. We take $\mathcal{B}_{m+1}$ as the consistent extension. At the end, we make $\mathcal{B} = \mathcal{B}_n$. We have that $\phi \in \mathcal{B}$ and, as $\bigwedge \mathcal{A} \wedge \langle\alpha\rangle \bigwedge \mathcal{B}$ is consistent, $\mathcal{A}S_\alpha\mathcal{B}$, by definition C.31, which implies that $\mathcal{A}R_\alpha\mathcal{B}$.

($\Leftarrow$): Suppose that there is an atom $\mathcal{B}$ such that $\phi \in \mathcal{B}$ and $\mathcal{A}R_\alpha\mathcal{B}$. Then $\mathcal{A}S_\alpha\mathcal{B}$ and $\bigwedge \mathcal{A} \wedge \langle\alpha\rangle \bigwedge \mathcal{B}$ is consistent by definition C.31. As ϕ is one of the conjuncts of $\bigwedge \mathcal{B}$, $\bigwedge \mathcal{A} \wedge \langle\alpha\rangle\phi$ is also consistent. As $\langle\alpha\rangle\phi$ is in $C(\Gamma)$, it must also be in $\mathcal{A}$, since $\mathcal{A}$ is a maximal consistent subset of $C(\Gamma)$. □

Lemma C.34. *For all looping processes P, $S_P \subseteq S'_P$, where $S'_P = S^*_{L_P} \circ S_{T_P}$.*

Proof. For an atom $\mathcal{B} \in At(\Gamma)$ and a relation S, we denote the set of atoms $\{\mathcal{A} \mid \mathcal{A}S\mathcal{B}\}$ as $\langle S\rangle\mathcal{B}$. Suppose there are two atoms $\mathcal{A}, \mathcal{B} \in At(\Gamma)$ such that $\mathcal{A} \in \langle S_P\rangle\mathcal{B}$, but $\mathcal{A} \notin \langle S'_P\rangle\mathcal{B}$. Let $V = \{\mathcal{C} \in At(\Gamma) \mid \mathcal{C} \in \langle S_P\rangle\mathcal{B} \text{ but } \mathcal{C} \notin \langle S'_P\rangle\mathcal{B}\} \cup \{\mathcal{C} \in At(\Gamma) \mid \mathcal{C} \notin \langle S_P\rangle\mathcal{B}\}$ and $\overline{V} = At(\Gamma) \setminus V = \{\mathcal{C} \in At(\Gamma) \mid \mathcal{C} \in \langle S_P\rangle\mathcal{B} \text{ and } \mathcal{C} \in \langle S'_P\rangle\mathcal{B}\}$. Thus, $\mathcal{A} \in V$. Let $r = \bigvee\{\bigwedge \mathcal{C} \mid \mathcal{C} \in V\}$. It is not difficult to see that $\neg r = \bigvee\{\bigwedge \mathcal{C} \mid \mathcal{C} \in \overline{V}\}$.

First, we have that $\vdash r \rightarrow [T_P]\neg \bigwedge \mathcal{B}$. Otherwise, $\neg(r \rightarrow [T_P]\neg \bigwedge \mathcal{B}) \equiv r \wedge \langle T_P\rangle \bigwedge \mathcal{B}$ is consistent. This means that there is $\mathcal{A}' \in V$ such that $\bigwedge \mathcal{A}' \wedge \langle T_P\rangle \bigwedge \mathcal{B}$ is consistent. On one hand, this implies, by **(Rec)**, that $\bigwedge \mathcal{A}' \wedge \langle P\rangle \bigwedge \mathcal{B}$ is consistent,

which means that $\mathcal{A}' \in \langle S_P \rangle \mathcal{B}$. On the other hand, it implies that $\mathcal{A}' S_{T_P} \mathcal{B}$, which means that $\mathcal{A}' \in \langle S'_P \rangle \mathcal{B}$. These two conclusions contradict the fact that $\mathcal{A}' \in V$.

Second, we also have that $\vdash r \rightarrow [L_P]r$. Otherwise, $\neg(r \rightarrow [L_P]r) \equiv r \wedge \langle L_P \rangle \neg r$ is consistent. This means that there are $\mathcal{A}' \in V$ and $\mathcal{B}' \in \overline{V}$ such that $\bigwedge \mathcal{A}' \wedge \langle L_P \rangle \bigwedge \mathcal{B}'$ is consistent, which implies that $\mathcal{A}' S_{L_P} \mathcal{B}'$. Since $\mathcal{B}' \in \overline{V}$, $\mathcal{B}' S_P \mathcal{B}$ and $\mathcal{B}' S'_P \mathcal{B}$. On one hand, $\mathcal{A}' S_{L_P} \mathcal{B}'$ and $\mathcal{B}' S'_P \mathcal{B}$ imply that $\mathcal{A}' S'_P \mathcal{B}$ (*). On the other hand, $\mathcal{A}' S_{L_P} \mathcal{B}'$ and $\mathcal{B}' S_P \mathcal{B}$ imply that $\bigwedge \mathcal{A}' \wedge \langle L_P \rangle \langle P \rangle \bigwedge \mathcal{B}$ is consistent, which, by **(Rec)**, implies that $\bigwedge \mathcal{A}' \wedge \langle P \rangle \bigwedge \mathcal{B}$ is consistent, which means that $\mathcal{A}' S_P \mathcal{B}$ (**). The conclusions in (*) and (**) contradict the fact that $\mathcal{A}' \in V$.

Taking these two results together, we conclude that $\vdash r \rightarrow ([T_P] \neg \bigwedge \mathcal{B} \wedge [L_P]r)$. By **(Gen)**, **(PL)**, **(FP)** and **(MP)**, $\vdash r \rightarrow [P] \neg \bigwedge \mathcal{B}$. But, as $\mathcal{A} \in V$, $\vdash \bigwedge \mathcal{A} \rightarrow r$, which means that $\vdash \bigwedge \mathcal{A} \rightarrow [P] \neg \bigwedge \mathcal{B}$. This implies that $\bigwedge \mathcal{A} \wedge \langle P \rangle \bigwedge \mathcal{B}$ is inconsistent, contradicting the fact that $\mathcal{A} S_P \mathcal{B}$. Thus, there cannot be a pair of atoms $\mathcal{A}, \mathcal{B} \in At(\Gamma)$ such that $\mathcal{A} \in \langle S_P \rangle \mathcal{B}$, but $\mathcal{A} \notin \langle S'_P \rangle \mathcal{B}$. □

Lemma C.35. *For all processes P, $S_P \subseteq R_P$.*

Proof. The proof is by induction on the structure of the process P.

- The base case is immediate, for we defined $R_\alpha = S_\alpha$ for all basic processes α.
- P is a non-looping process:
  - Suppose $\mathcal{A} S_{\alpha.P} \mathcal{B}$, that is, $\bigwedge \mathcal{A} \wedge \langle \alpha.P \rangle \bigwedge \mathcal{B}$ is consistent. By **(Pr)**, $\bigwedge \mathcal{A} \wedge \langle \alpha \rangle \langle P \rangle \bigwedge \mathcal{B}$ is consistent as well. Using a "forcing choices" argument (as exemplified in lemma C.33), we can construct an atom $\mathcal{C}$ such that $\bigwedge \mathcal{A} \wedge \langle \alpha \rangle \bigwedge \mathcal{C}$ and $\bigwedge \mathcal{C} \wedge \langle P \rangle \bigwedge \mathcal{B}$ are both consistent. But then, by the inductive hypothesis, $\mathcal{A} R_\alpha \mathcal{C}$ and $\mathcal{C} R_P \mathcal{B}$. It follows that $\mathcal{A} R_{\alpha.P} \mathcal{B}$ as required.
  - Suppose $\mathcal{A} S_{\alpha.A} \mathcal{B}$, that is, $\bigwedge \mathcal{A} \wedge \langle \alpha.A \rangle \bigwedge \mathcal{B}$ is consistent. By **(Cons)**, $\bigwedge \mathcal{A} \wedge \langle \alpha \rangle \langle P_A \rangle \bigwedge \mathcal{B}$ is consistent as well. Using a "forcing choices" argument, we can construct an atom $\mathcal{C}$ such that $\bigwedge \mathcal{A} \wedge \langle \alpha \rangle \bigwedge \mathcal{C}$ and $\bigwedge \mathcal{C} \wedge \langle P_A \rangle \bigwedge \mathcal{B}$ are both consistent. But then, by the inductive hypothesis, $\mathcal{A} R_\alpha \mathcal{C}$ and $\mathcal{C} R_{P_A} \mathcal{B}$. It follows that $\mathcal{A} R_{\alpha.A} \mathcal{B}$ as required.

– Suppose $\mathcal{A}S_{P_1+P_2}\mathcal{B}$, that is, $\bigwedge\mathcal{A}\wedge\langle P_1+P_2\rangle\bigwedge\mathcal{B}$ is consistent. By **(NC)**, $\bigwedge\mathcal{A}\wedge\langle P_1\rangle\bigwedge\mathcal{B}$ is consistent or $\bigwedge\mathcal{A}\wedge\langle P_2\rangle\bigwedge\mathcal{B}$ is consistent. But then, by the inductive hypothesis, $\mathcal{A}R_{P_1}\mathcal{B}$ or $\mathcal{A}R_{P_2}\mathcal{B}$. It follows that $\mathcal{A}R_{P_1+P_2}\mathcal{B}$ as required.

– Suppose $\mathcal{A}S_{P_1|P_2}\mathcal{B}$, that is, $\bigwedge\mathcal{A}\wedge\langle P_1\mid P_2\rangle\bigwedge\mathcal{B}$ is consistent. By **(PC)**, $\bigwedge\mathcal{A}\wedge\langle\alpha\rangle\langle P'\rangle\bigwedge\mathcal{B}$ is consistent for some basic process α and some process P'. Using a "forcing choices" argument, we can construct an atom $\mathcal{C}$ such that $\bigwedge\mathcal{A}\wedge\langle\alpha\rangle\bigwedge\mathcal{C}$ and $\bigwedge\mathcal{C}\wedge\langle P'\rangle\bigwedge\mathcal{B}$ are both consistent. But then, by the inductive hypothesis, $\mathcal{A}R_\alpha\mathcal{C}$ and $\mathcal{C}R_{P'}\mathcal{B}$. It follows that $\mathcal{A}R_{\alpha.P'}\mathcal{B}$, which means that $\mathcal{A}R_{P_1|P_2}\mathcal{B}$ as required.

- Suppose $\mathcal{A}S_P\mathcal{B}$, where P is a looping process. By lemma C.34, $S_P\subseteq S'_P$, where $S'_P=S^*_{L_P}\circ S_{T_P}$. By the induction hypothesis, $S_{L_P}\subseteq R_{L_P}$ and $S_{T_P}\subseteq R_{T_P}$. This implies that $S'_P\subseteq R_P$, which proves the result.

□

Lemma C.36 (Existence Lemma)**.** *For all atoms $\mathcal{A}\in At(\Gamma)$ and all formulas $\langle P\rangle\phi\in C(\Gamma)$, $\langle P\rangle\phi\in\mathcal{A}$ iff there is $\mathcal{B}\in At(\Gamma)$ such that $\mathcal{A}R_P\mathcal{B}$ and $\phi\in\mathcal{B}$.*

Proof. ($\Rightarrow$) Suppose $\langle P\rangle\phi\in\mathcal{A}$. We can build an atom $\mathcal{B}$ such that $\phi\in\mathcal{B}$ and $\mathcal{A}S_P\mathcal{B}$ by "forcing choices". But, by lemma C.35, $S_P\subseteq R_P$, thus $\mathcal{A}R_P\mathcal{B}$ as well.

($\Leftarrow$) We proceed by induction on the structure of P.

- The base case is just the Existence Lemma for basic processes.

- P is a non-looping process:

  – Suppose P has the form $\alpha.P'$, $\mathcal{A}R_{\alpha.P'}\mathcal{B}$ and $\phi\in\mathcal{B}$. Thus, there is an atom $\mathcal{C}$ such that $\mathcal{A}R_\alpha\mathcal{C}$ and $\mathcal{C}R_{P'}\mathcal{B}$. By the Fischer-Ladner closure conditions, $\langle P'\rangle\phi\in C(\Gamma)$, hence by the induction hypothesis, $\langle P\rangle\phi\in C$. Similarly, as $\langle\alpha\rangle\langle P'\rangle\phi\in C(\Gamma)$, $\langle\alpha\rangle\langle P'\rangle\phi\in\mathcal{A}$. Hence, by **(Pr)**, $\langle\alpha.P\rangle\phi\in\mathcal{A}$.

  – Suppose P has the form $\alpha.A$, $\mathcal{A}R_{\alpha.A}\mathcal{B}$ and $\phi\in\mathcal{B}$. Thus, there is an atom $\mathcal{C}$ such that $\mathcal{A}R_\alpha\mathcal{C}$, $\mathcal{C}R_{P_A}\mathcal{B}$ and $\phi\in\mathcal{B}$. By the Fischer-Ladner closure conditions, $\langle P_A\rangle\phi\in C(\Gamma)$, hence by the induction hypothesis, $\langle P_A\rangle\phi\in C$. Similarly, as $\langle\alpha\rangle\langle P_A\rangle\phi\in C(\Gamma)$, $\langle\alpha\rangle\langle P_A\rangle\phi\in\mathcal{A}$. Hence, by **(Cons)**, $\langle\alpha.A\rangle\phi\in\mathcal{A}$.

- Suppose P has the form $P_1 + P_2$, $\mathcal{A}R_{P_1+P_2}\mathcal{B}$ and $\phi \in \mathcal{B}$. Thus, $\mathcal{A}R_{P_1}\mathcal{B}$ or $\mathcal{A}R_{P_2}\mathcal{B}$. By the Fischer-Ladner closure conditions, $\langle P_1\rangle\phi, \langle P_2\rangle\phi \in C(\Gamma)$, hence by the inductive hypothesis, $\langle P_1\rangle\phi \in \mathcal{A}$ or $\langle P_2\rangle\phi \in \mathcal{A}$. Hence, by **(NC)**, $\langle P_1 + P_2\rangle\phi \in \mathcal{A}$.
- Suppose P has the form $P_1 \mid P_2$, $\mathcal{A}R_{P_1|P_2}\mathcal{B}$ and $\phi \in \mathcal{B}$. Thus, $\mathcal{A}R_{\alpha.P'}\mathcal{B}$ for some process α and some process P'. Then, there is an atom $\mathcal{C}$ such that $\mathcal{A}R_\alpha\mathcal{C}$ and $\mathcal{C}R_{P'}\mathcal{B}$. By the Fischer-Ladner closure conditions, $\langle\alpha.P'\rangle\phi, \langle\alpha\rangle\langle P'\rangle\phi, \langle P'\rangle\phi \in C(\Gamma)$, hence by the inductive hypothesis, $\langle P\rangle\phi \in C$ and $\langle\alpha\rangle\langle P'\rangle\phi \in \mathcal{A}$. Hence, by **(Pr)**, $\langle\alpha.P\rangle\phi \in \mathcal{A}$ and, by **(PC)**, $\langle P_1 \mid P_2\rangle\phi \in \mathcal{A}$.

- Suppose P is a looping process, $\mathcal{A}R_P\mathcal{B}$ and $\phi \in \mathcal{B}$. Then, there is a finite sequence of atoms $\mathcal{C}_0 \ldots \mathcal{C}_n$ such that $\mathcal{A} = \mathcal{C}_0 R_{L_P} \mathcal{C}_1 \ldots \mathcal{C}_{n-1} R_{L_P} \mathcal{C}_n R_{T_P} \mathcal{B}$. We prove by a sub-induction on n that $\langle P\rangle\phi \in \mathcal{C}_i$, for all i. The desired result for $\mathcal{A} = \mathcal{C}_0$ follows immediately.
  - Base case: $n = 0$. This means $\mathcal{A}R_{T_P}\mathcal{B}$. By the Fischer-Ladner closure conditions, $\langle T_P\rangle\phi \in C(\Gamma)$, hence by the inductive hypothesis, $\langle T_P\rangle\phi \in \mathcal{A}$. Hence, by **(Rec)**, $\langle P\rangle\phi \in \mathcal{A}$.
  - Inductive step: Suppose the result holds for $k < n$, and that $\mathcal{A} = \mathcal{C}_0 R_{L_P} \mathcal{C}_1 \ldots R_{L_P} \mathcal{C}_n R_{T_P} \mathcal{B}$. By the inductive hypothesis, $\langle P\rangle\phi \in \mathcal{C}_1$. Hence $\langle L_P\rangle\langle P\rangle\phi \in \mathcal{A}$, as $\langle L_P\rangle\langle P\rangle\phi \in C(\Gamma)$. By **(Rec)**, $\langle P\rangle\phi \in \mathcal{A}$.

□

Lemma C.37 (Truth Lemma)**.** *Let $\mathcal{N}^\Gamma = (At(\Gamma), \{R_P\}, \mathbf{V})$ be the CCS-PDL model over Γ. For all atoms $\mathcal{A} \in At(\Gamma)$ and all formulas $\varphi \in C(\Gamma)$, $\mathcal{N}^\Gamma, \mathcal{A} \Vdash \varphi$ iff $\varphi \in \mathcal{A}$.*

Proof. The proof is by induction on the structure of the formula φ.

- ϕ is a proposition symbol: The proof follows directly from the definition of $\mathbf{V}$.
- $\phi = \neg\psi$ or $\phi = \psi_1 \wedge \psi_2$: The proof follows directly from lemma C.28.
- $\phi = \langle P\rangle\psi$:

($\Rightarrow$) Suppose that $\mathcal{N}^{\Gamma}, \mathcal{A} \Vdash \langle P \rangle \psi$. Then, there exists $\mathcal{A}' \in \mathcal{N}^{\Gamma}$ such that $\mathcal{A} R_P \mathcal{A}'$ and $\mathcal{N}^{\Gamma}, \mathcal{A}' \Vdash \psi$. By the induction hypothesis, we know that $\psi \in \mathcal{A}'$ and, by the Existence Lemma, we have that $\langle P \rangle \psi \in \mathcal{A}$.

($\Leftarrow$) Suppose that $\langle P \rangle \psi \in \mathcal{A}$. Then, by the Existence Lemma, there is $\mathcal{A}' \in \mathcal{N}^{\Gamma}$ such that $\mathcal{A} R_P \mathcal{A}'$ and $\psi \in \mathcal{A}'$. By the induction hypothesis, $\mathcal{N}^{\Gamma}, \mathcal{A}' \Vdash \psi$, which implies $\mathcal{N}^{\Gamma}, \mathcal{A} \Vdash \langle P \rangle \psi$.

□

Theorem C.38 (Completeness)**.** *Every consistent formula is satisfiable in a finite model that respects definition C.22.*

Proof. Let φ be a consistent formula. Let $C(\varphi)$ be its closure under the conditions of definition C.25. As φ is consistent, by corollary C.30, there is an atom $\mathcal{A} \in At(\varphi)$ such that $\varphi \in \mathcal{A}$. Let $\mathcal{N}^{\varphi}$ be the CCS-PDL model over φ. Then, by the Truth Lemma (lemma C.37), as $\varphi \in \mathcal{A}$, we conclude that $\mathcal{N}^{\varphi}, \mathcal{A} \Vdash \varphi$, which proves the theorem. □

Apêndice D

A Propositional Dynamic Logic for Concurrent Programs Based on the π-Calculus

D.1 Introduction[1]

Propositional Dynamic Logic (PDL) [46] plays an important role in formal specification and reasoning about sequential programs and systems. PDL is a multimodal logic with one modality $\langle P \rangle$ for each program P. The logic has a set of basic programs and a set of operators (sequential composition, iteration and nondeterministic choice) that can be used to build more complex programs from simpler ones. A Kripke semantics can be provided, with a frame $\mathcal{F} = (W, R_P)$, where W is a non-empty set of possible program states and, for each program P, R_P is a binary relation on W such that $(s, t) \in R_P$ if and only if there is a computation of P starting in s and terminating in t.

The π-Calculus is a well known process algebra, proposed by Milner, Parrow and Walker [25], for the specification of communicating concurrent systems. It is an extension of Milner's CCS [21] that is able to describe not only non-determinism and concurrency, but also *mobility* of processes. It models the concurrency and

[1]Este capítulo expõe o conteúdo do artigo [45], publicado nos anais do congresso M4M 2009, Computer Science Research Report, Roskilde University, v. 128, p. 42-56, 2009. A ser publicado no Electronic Notes in Theoretical Computer Science.

interaction between processes through individual acts of communication. A pair of processes can communicate through a common channel and each act of communication consists of a message (which, in the π-Calculus, is also a channel name) being sent at one end of the channel and immediately being received at the other. A π-Calculus specification is a description of the behaviour expected from a system, based on the communication events that may occur. As in PDL, the π-Calculus has a set of operators (action prefix, parallel composition, nondeterministic choice and restriction on acts of communication) that are used to inductively build process specifications from a set of basic actions.

This work presents a Propositional Dynamic Logic (πDL[2]) in which the programs are described in a language based on the π-Calculus without replication. There are, in the literature, some other logics that make use of CCS or the π-Calculus. However, they use these process algebras as a language for the description of frames and models, while using standard modal logics for the description of properties (see, for example, [47] and [48]). The logic that we develop in the present work uses the π-Calculus in a distinct way. Its operators and constructions are *added* to a basic modal logic in order to create a dynamic logic where it is simple to describe and verify properties of communicating, concurrent, non-deterministic and mobile programs and systems, in a similar way as PDL is used for the sequential case. As such, this logic is an extension of the logics from our previous works [43] and [44], which develop propositional dynamic logics based on CCS.

It should be emphasized that the contribution of this work is on the field of dynamic logics and not on the field of process algebras. From process algebras, we just borrow a set of operators that are suitable for the description of communication and concurrency. We use these operators because they have a well-established theory behind them and we can use many of its concepts and results to help us build our logic.

Our logic is related to Concurrent PDL (CPDL) [6] and Channel-CPDL [7], but has advantages over both. The former can only describe properties of concurrent systems with no communication between the components and while the latter is

[2]The pun here comes from the fact that the name of the letter π in Greek and the name of the letter P in English are pronounced exactly the same way.

able to describe interesting properties of communicating concurrent systems, it does not have a simple Kripke semantics (in fact, "a formal definition of the semantics of channel-CPDL is rather complicated" [7]) and its satisfiability problem can be proved undecidable (Π^1_1-hard), which also implies that it does not have a complete axiomatization. On the other hand, due to the use of the π-Calculus mechanisms of communication and concurrency, our logic has a simple Kripke semantics, the finite model property, a straightforward axiomatization and can also deal with *mobility*. Our logic can also be seen as an extension of PDL with Interleaving (iPDL) [9]. In iPDL, the parallel operator that is present in the logic is similar to the parallel operator of the π-Calculus, but it only allows interleaving of the actions in parallel programs, while the parallel operator of the π-Calculus (in conjunction with the restriction operator) also allows communication, synchronization and mobility.

The rest of this paper is organized as follows. In Section D.2, we introduce the necessary background concepts: Propositional Dynamic Logic and the π-Calculus. Our logic (πDL), together with a couple of simple examples of its application and a complete axiomatic system, is presented in Section D.3. Finally, in Section D.4, we state our final remarks. We omit some of the proofs in the text, when they follow directly from previously stated results.

D.2 Background

This section presents two important subjects. First, we make a brief review of the syntax and semantics of PDL. Second, we present the π-Calculus together with some useful concepts, properties and results from its theory. We do not assume a familiarity with the π-Calculus, since process algebras are by no means a universally studied topic among (modal) logicians. We introduce here all that is necessary for our presentation in the next sections, trying to make this work as self-contained as possible.

D.2.1 Propositional Dynamic Logic

In this section, we present the syntax and semantics of PDL. For a more detailed account, [10] can be consulted.

Definition D.1. *The PDL language consists of a set Φ of countably many proposition symbols, a set $\mathbb{P}$ of countably many basic programs, the boolean operators $\neg$ and $\wedge$, the program constructors ;, $\cup$ and * and a modality $\langle P\rangle$ for every program P. The formulas are defined as follows:*

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle P\rangle\varphi, \text{ with } P ::= a \mid P_1; P_2 \mid P_1 \cup P_2 \mid P^*,$$

where $p \in \Phi$ and $a \in \mathbb{P}$.

In all the logics that appear in this paper, we use the standard abbreviations $\bot \equiv \neg\top$, $\varphi \vee \phi \equiv \neg(\neg\varphi \wedge \neg\phi)$, $\varphi \to \phi \equiv \neg(\varphi \wedge \neg\phi)$ and $[P]\varphi \equiv \neg\langle P\rangle\neg\varphi$.

Definition D.2. *A* frame *for PDL is a tuple $\mathcal{F} = (W, \{R_a\}_{a\in\mathbb{P}})$ where W is a non-empty set of states and R_a is a binary relation for each basic program a. Besides that, we inductively build binary relations R_P, for each non-basic program P, using the rules $R_{P_1;P_2} = R_{P_1} \circ R_{P_2}$, $R_{P_1\cup P_2} = R_{P_1} \cup R_{P_2}$ and $R_{P^*} = R_P^*$, where R_P^* denotes the reflexive transitive closure of R_P.*

Definition D.3. *A* model *for PDL is a pair $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, where $\mathcal{F}$ is a PDL frame and $\mathbf{V}$ is a valuation function $\mathbf{V} : \Phi \mapsto 2^W$.*

Definition D.4. *Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. The semantical notion of* satisfaction *of a formula φ in a model $\mathcal{M}$ at a state w, notation $\mathcal{M}, w \Vdash \varphi$, is defined in PDL in the standard way for modal logics [10] for the atomic formulas and the boolean operators. The following rule takes care of the modalities: $\mathcal{M}, w \Vdash \langle P\rangle\varphi$ iff there is $w' \in W$ such that wR_Pw' and $\mathcal{M}, w' \Vdash \varphi$.*

D.2.2 The π-Calculus

The π-Calculus is a well known process algebra, proposed by Milner, Parrow and Walker [25], for the specification of communicating concurrent systems. It is an extension of Milner's CCS [21] that is able to describe not only non-determinism and concurrency, but also *mobility* of processes. A π-Calculus specification is a description of the behaviour expected from a system, based on the communication events that may occur. For a broad introduction to the π-Calculus, [26] can be consulted.

In the π-Calculus, a pair of processes can communicate through a common channel and each act of communication consists of a message (which, in the π-Calculus, is also a channel name) being sent at one end of the channel and immediately being received at the other.

Let $\mathcal{N} = \{a, b, c, \ldots\}$ be a set of names. Each channel in a π-Calculus specification is labelled by a name. The labels of the channels are also used to describe the communication actions (sending and receiving messages) performed by the processes, as is shown below. Besides these communication actions, the π-Calculus has only one other action: the silent action, denoted by τ, used to represent any internal action performed by any of the processes that does not involve an act of communication (e.g.: a memory update).

Definition D.5. *In our presentation of the π-Calculus, process specifications can be built using the following operations:*

$$P ::= \mathbf{0} \mid \mathit{END} \mid \alpha.P \mid P_1; P_2 \mid P_1 + P_2 \mid P_1 | P_2 \mid P^* \mid (\nu a)P,$$

with

$$\alpha ::= a(x) \mid \overline{a}\langle x \rangle \mid \overline{a}\langle \nu x \rangle \mid \tau,$$

where $a, x \in \mathcal{N}$.

Usually, the π-Calculus is presented with a replication operator (!), that denotes the ability of a process to generate multiple copies of itself, or with constants, that may be used to describe recursion. In [44], in the context of CCS, we present a dynamic logic that uses processes with constants. However, in order to keep the finite model property and a complete axiomatization, we had to restrict the interaction between constants and the | operator in order to prevent potentially self-replicating processes. Besides that, with constants, the axiomatization and the theory behind its completeness proof became considerably more complex. On the other hand, the issue of whether it is possible to keep the finite model property and a complete axiomatization in the presence of replication remains open and we defer it to a future work, as explained in Section D.4.

Thus, at the present time, we restrict ourselves to the language without constants and the replication operator. However, as is also shown in [44], it is much simpler

to deal with iteration (which is a restricted form of recursion) in the logic than with recursion in its more general form and the resulting axiomatization is more elegant. So, in order to express iterative behaviours, we add to our presentation of the π-Calculus the PDL-inspired operators * and ;.

As it is explained in details in [44], the somewhat loose definition of the null process $\mathbf{0}$ in the π-Calculus, which fails to differentiate between a deadlock and a successful termination (unlike other process algebras, as ACP [22] for instance, in which the deadlocked process and the terminated process are different), can get in the way of a fully compositional semantics for a dynamic logic based on the π-Calculus. To solve this, we introduce an extra action, with a special meaning: the *ending action*, denoted by *END*. All other actions are called *running actions*. A process can only successfully finish after performing the action *END* and it always successfully finishes after performing such action. If a process cannot perform any running action and cannot successfully finish, it is called a *deadlocked* process.

$\mathbf{0}$ is the *null process*. It is a deadlocked process, since it is incapable of performing any running action and of successfully finishing. *END* is a process that is incapable of performing any running action, but it is capable of successfully finishing. The *prefix* operator (.) denotes that the process will first perform the running action α and then behave as P. The *sequential composition* operator (;) denotes that the process will first behave as P_1 and if and when P_1 successfully terminates, it will proceed behaving as P_2. The *nondeterministic choice* operator (+) denotes that the process will make a nondeterministic choice to behave as either P_1 or P_2. The *parallel composition* operator (|) denotes that the processes P_1 and P_2 may proceed independently or may communicate through a common channel. The *iteration* operator (*) denotes that the process P is capable of being iterated zero or more times. Finally, the *restriction* operator (νa) denotes that the channel a is only accessible inside P (the scope of a is P).

The action $a(x)$, called *input action*, denotes that the process receives a name through the channel labelled by a and the name x marks, in P, the places where the received name should be put. The actions $\overline{a}\langle x\rangle$, called *free output action*, and $\overline{a}\langle \nu x\rangle$, called *bound output action*, both denote that the process sends the name x through the channel labelled by a. The difference between the two is that, in $\overline{a}\langle \nu x\rangle$, x is a

restricted name, with this action working as an abbreviation for $(\nu x)\overline{a}\langle x\rangle$. Finally, τ denotes the silent action. We define the bound output action as a primitive action because, as is shown below, under certain circumstances, the only form of restriction that is needed is the one provided by bounded outputs.

We say that the actions $a(x)$ and $\overline{a}\langle\nu x\rangle$ and the restriction (νx) *bind* the name x, calling them *binders*. We say that a name is *bound* in P if it occurs inside the scope of an action or a restriction that binds it. Otherwise, we say that a name is *free* in P. We denote by $f(P)$ the set of free names in P and by $b(P)$ the set of bound names in P. Similarly, we denote by $f(\alpha)$ and $b(\alpha)$ the free and bound names in an action α. In the actions $a(x)$, $\overline{a}\langle x\rangle$ and $\overline{a}\langle\nu x\rangle$, a is called the *subject*, denoted by $s(\alpha)$, where α is the action, and x is called the *object*, denoted by $o(\alpha)$. The τ action has neither subject nor object.

Definition D.6. *We say that a relation $\cong$ between processes is a* congruence *if it is an equivalence relation and it is preserved by all π-Calculus operators, that is, if $P \cong Q$, then $\alpha.P \cong \alpha.Q$, $P + R \cong Q + R$ and so on.*

Definition D.7. *A syntactic substitution of a* bound name *by a* fresh name *(a name that does not occur in the process specification) in its binder and in every occurrence of the name in the scope of this binder is called an* alpha-conversion.

Definition D.8. Structural congruence, *denoted by $\equiv$, is a relation between processes defined by the following set of axioms and rules:*

1. *It is a congruence;*
2. *It is closed under alpha-conversion;*
3. *Commutativity of $+$ and $|$;*
4. *If $a \neq x$, $(\nu x)\overline{a}\langle x\rangle.P \equiv \overline{a}\langle\nu x\rangle.P$;*
5. *$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$;*
6. *If $x \notin f(P)$, $(\nu x)P \equiv P$.*

Definition D.9. *We say that a process P is* clean *if no name appears both free and bound in P and no name is bound by more than one binder. We say that a process is* unrestricted *if it has no occurrences of the ν operator. We say that a clean process is in ν-*prefix form *if it has the form $(\nu x_1)\ldots(\nu x_n)P$, $n \geq 0$, where P is unrestricted. Finally, we say that a clean process is in ν-*standard *form if the only occurrences of the ν operator are inside bound output prefixes.*

We write $P \xrightarrow{\alpha} P'$ to express that the process P can perform the action α and after that behave as P'. We write $P \xrightarrow{END} \surd$ to express that the process P can perform the action *END* and successfully finish. In Table D.1, we present the semantics for the operators of the π-Calculus based on this notation. This semantics is called *late semantics*. For more details on this and other semantics, [26] can be consulted.

$\frac{P' \equiv P, P \xrightarrow{\alpha} Q, Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$	$\alpha.P \xrightarrow{\alpha} P$	$END \xrightarrow{END} \surd$	$P^* \xrightarrow{END} \surd$
$\frac{P \xrightarrow{\alpha} P'}{P;Q \xrightarrow{\alpha} P';Q}$	$\frac{P \xrightarrow{END} \surd, Q \xrightarrow{\alpha} Q'}{P;Q \xrightarrow{\alpha} Q'}$	$\frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$	$\frac{P \xrightarrow{\alpha} P', b(\alpha) \cap f(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$
$\frac{P \xrightarrow{a(x)} P', Q \xrightarrow{\overline{a}\langle u \rangle} Q'}{P \mid Q \xrightarrow{\tau} P'\{u/x\} \mid Q'}$	$\frac{P \xrightarrow{a(x)} P', Q \xrightarrow{\overline{a}\langle \nu u \rangle} Q'}{P \mid Q \xrightarrow{\tau} (\nu u)(P'\{u/x\} \mid Q')}$	$\frac{P \xrightarrow{\alpha} P'}{P^* \xrightarrow{\alpha} P';P^*}$	$\frac{P \xrightarrow{\alpha} P', x \notin \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}$
$\frac{P \xrightarrow{END} \surd, Q \xrightarrow{END} \surd}{P;Q \xrightarrow{END} \surd}$	$\frac{P \xrightarrow{END} \surd}{P+Q \xrightarrow{END} \surd}$	$\frac{Q \xrightarrow{END} \surd}{P+Q \xrightarrow{END} \surd}$	$\frac{P \xrightarrow{END} \surd, Q \xrightarrow{END} \surd}{P \mid Q \xrightarrow{END} \surd}$
$\frac{P \xrightarrow{END} \surd}{(\nu x)P \xrightarrow{END} \surd}$			

Table D.1: Transition Relations of the π-Calculus

From Table D.1, we can see that we have a clear distinction between deadlock and termination. A specification of the form $\alpha.\mathbf{0}$ denotes that a process performs the action α and then deadlocks, while a specification of the form $\alpha.END$ denotes that a process performs the action α and then successfully terminates.

Definition D.10. *Let $\mathcal{P}$ be the set of all possible process specifications. A* late bisimulation *is a* symmetric *binary relation $Z \subseteq \mathcal{P} \times \mathcal{P}$ such that*

1. *If $(P, Q) \in Z$ and $P \xrightarrow{\alpha} P'$, where $b(\alpha)$ is fresh in P and Q, then*
   (a) *If $\alpha = a(x)$, then there is $Q' \in \mathcal{P}$ such that $Q \xrightarrow{\alpha} Q'$ and for all $u \in \mathcal{N}$, $(P'\{u/x\}, Q'\{u/x\}) \in Z$;*
   (b) *If α is not an input action, then there is $Q' \in \mathcal{P}$ such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in Z$;*
2. *If $(P, Q) \in Z$ and $P \xrightarrow{END} \surd$, then $Q \xrightarrow{END} \surd$.*

The reason why the only running actions that need to be considered are the ones that satisfy the freshness condition above is explained in details in [26].

Definition D.11. *Two process specifications P and Q are* late bisimilar *(or simply* bisimilar*), denoted by $P \sim Q$, if there is a late bisimulation Z such that $(P, Q) \in Z$. In the π-Calculus, bisimilarity is an equivalence relation but is not a congruence.*

Theorem D.12. *If $P \equiv Q$, then $P \dot{\sim} Q$.*

It should be noticed that while $\mathbf{0}$ is the neutral element for the $+$ operator, that is, $P + \mathbf{0} \dot{\sim} P$, *END* is the neutral element for the $|$ operator, as $P|END \dot{\sim} P$[3].

We now present a few particular bisimilarities that are going to be useful in the axiomatization of our logic. We start with the Expansion Law.

Theorem D.13 (Expansion Law [26])**.** *Let $P = P_1 \mid P_2$, where P is clean and unrestricted and $|$ does not occur in P_1 and P_2. Then*

$$P \dot{\sim} \sum_{P_1 \xrightarrow{\alpha} P_1'} \alpha.(P_1' \mid P_2) + \sum_{P_2 \xrightarrow{\beta} P_2'} \beta.(P_1 \mid P_2') + \sum_{R \in A_\tau} \tau.R + E_P,$$

where $A_\tau = \{(P_1'\{u/x\} \mid P_2') : P_1 \stackrel{a(x)}{\rightarrow} P_1'$ and $P_2 \stackrel{\overline{a}\langle u\rangle}{\rightarrow} P_2'$, for some $a \in \mathcal{N}\} \cup \{(P_1' \mid P_2'\{u/x\}) : P_1 \stackrel{\overline{a}\langle u\rangle}{\rightarrow} P_1'$ and $P_2 \stackrel{a(x)}{\rightarrow} P_2'$, for some $a \in \mathcal{N}\}$ and $E_P = END$, if $P_1 \stackrel{END}{\rightarrow} \surd$ and $P_2 \stackrel{END}{\rightarrow} \surd$ or $E_P = \mathbf{0}$, otherwise. We denote the right side of this bisimilarity by $Exp(P)$.

Definition D.14. ν-bisimilarity*, denoted by $P \stackrel{\nu}{\sim} Q$, is a relation between processes defined by the following set of axioms and rules, where $x \notin \alpha$ denotes that x is neither the subject nor the object of the action α:*

1. *If $P \equiv Q$, then $P \stackrel{\nu}{\sim} Q$;*
2. *$(\nu x)\mathbf{0} \stackrel{\nu}{\sim} \mathbf{0}$;*
3. *$(\nu x)END \stackrel{\nu}{\sim} END$;*
4. *If $x \notin \alpha$, $(\nu x)\alpha.P \stackrel{\nu}{\sim} \alpha.(\nu x)P$;*
5. *If $x = s(\alpha)$, $(\nu x)\alpha.P \stackrel{\nu}{\sim} \mathbf{0}$;*
6. *$(\nu x)(P;Q) \stackrel{\nu}{\sim} (\nu x)P;(\nu x)Q$;*
7. *$(\nu x)(P + Q) \stackrel{\nu}{\sim} (\nu x)P + (\nu x)Q$;*
8. *If $x \notin f(P)$, $P|(\nu x)Q \stackrel{\nu}{\sim} (\nu x)(P|Q)$;*
9. *$(\nu x)P^* \stackrel{\nu}{\sim} ((\nu x)P)^*$.*

It follows from Table D.1 and Definition D.11 that items (ii)-(ix) are indeed bisimilarities. Hence, the relation of ν-bisimilarity is a subset of the relation of bisimilarity. ν-bisimilarity is a convenient relation because it has a simple axiomatization and it is sufficient for all our needs in this work.

[3]Notice that, according to table D.1, $P|\mathbf{0}$ can never successfully finish, so $P|\mathbf{0} \not\dot{\sim} P$ in general.

Theorem D.15. *Every process is structurally congruent to a clean process. Every clean process is ν-bisimilar to a process in ν-prefix form. Finally, every clean process with no occurrences of the $|$ operator is ν-bisimilar to a process in ν-standard form.*

D.3 πDL

In this section, we define a Propositional Dynamic Logic (πDL) in which the programs are built in a language based on the π-Calculus without replication (Definition D.5). First, we introduce the key concept of *finite possible runs* of a process. We then proceed to describe, using this concept, the syntax and semantics of πDL and provide a couple of simple examples of its application. Finally, we present an axiomatization for πDL and prove its soundness and completeness.

D.3.1 Action Sequences and Possible Runs

Here, we introduce the concept of *finite possible runs* of a process.

Definition D.16. *We use the notation $\overrightarrow{\alpha}$ to denote a potentially infinite sequence of actions $\alpha_1.\alpha_2.\ldots.\alpha_n(.\cdots)$ (the empty sequence is denoted by $\overrightarrow{\varepsilon}$). The empty sequence follows the rule $\overrightarrow{\alpha}.\overrightarrow{\varepsilon} = \overrightarrow{\varepsilon}.\overrightarrow{\alpha} = \overrightarrow{\alpha}$, for all $\overrightarrow{\alpha}$. We denote the i-th term of the sequence $\overrightarrow{\alpha}$ by $(\overrightarrow{\alpha})_i$.*

Definition D.17. *For a finite sequence of actions $\overrightarrow{\alpha}$, we write $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} P'$ to express that the process P may perform the sequence $\overrightarrow{\alpha}$ and after that behave as P'. Besides that, we write $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$ to express that the process P may successfully finish after performing the sequence $\overrightarrow{\alpha}$.*

We can define notions of alpha-conversion of bound names in a sequence of actions and of a *clean* sequence of actions, in analogy with Definitions D.7 and D.9. We can also extend our notation and write $b(\overrightarrow{\gamma})$ and $f(\overrightarrow{\gamma})$ for the sets of bound and free names in the sequence $\overrightarrow{\gamma}$. If a sequence of actions $\overrightarrow{\gamma}$ can be alpha-converted to a sequence $\overrightarrow{\sigma}$, we write $\overrightarrow{\gamma} \equiv_\alpha \overrightarrow{\sigma}$. It is not difficult to see that, if $P \stackrel{\overrightarrow{\gamma}}{\Rightarrow} \surd$, then $P \stackrel{\overrightarrow{\sigma}}{\Rightarrow} \surd$, where $\overrightarrow{\gamma} \equiv_\alpha \overrightarrow{\sigma}$ and $\overrightarrow{\sigma}$ is clean. Let $S(P)$ be the set of all such $\overrightarrow{\sigma}$. Then, it is also not difficult to see that, if we establish the convention that $\overrightarrow{\sigma} \equiv_\alpha \overrightarrow{\sigma}$, $\equiv_\alpha$ is an equivalence relation for the elements of the set $S(P)$.

Definition D.18. *We define the set of finite possible runs of a process P, denoted by $\overrightarrow{\mathcal{R}_f}(P)$, as the quotient set $\overrightarrow{\mathcal{R}_f}(P) = S(P)/\equiv_\alpha$. If $\overrightarrow{\gamma} \in S(P)$, then $[\overrightarrow{\gamma}] \in \overrightarrow{\mathcal{R}_f}(P)$ denotes its equivalence class.*

We want to define a semantics for our logic that only takes into account the *finite* possible runs of the processes, i.e., situations in which the processes successfully finish. So, we present some useful results about finite possible runs.

Definition D.19. *Let $\overrightarrow{\alpha}$ and $\overrightarrow{\sigma}$ be two sequences of actions and let P and Q be two process specifications. If $b(\overrightarrow{\alpha})$ is fresh in Q and $b(\overrightarrow{\sigma})$ is fresh in P, we write $(\overrightarrow{\alpha}, \overrightarrow{\sigma}) \bowtie (P, Q)$.*

Definition D.20. *Let $T = \overrightarrow{\mathcal{R}_f}(P)$ and $U = \overrightarrow{\mathcal{R}_f}(Q)$ and let $\natural(\overrightarrow{\alpha}) = \overrightarrow{\lambda}$, where $\overrightarrow{\alpha} = \overrightarrow{\lambda}.END$. We can define the following operations on the sets T and U:*

- $T \circ U = \{[\natural(\overrightarrow{\alpha}).\overrightarrow{\beta}] : [\overrightarrow{\alpha}] \in T, [\overrightarrow{\beta}] \in U \text{ and } (\overrightarrow{\alpha}, \overrightarrow{\beta}) \bowtie (P, Q)\}$;
- $T \cup U = \{[\overrightarrow{\alpha}] : [\overrightarrow{\alpha}] \in T \text{ or } [\overrightarrow{\alpha}] \in U\}$;
- $R^0 = \{[\overrightarrow{\varepsilon}]\}, R^n = R \circ R^{n-1} (n \geq 1)$ and $R^* = \bigcup_{n \in \mathbb{N}} R^n$.

Lemma D.21. *If $P \dot{\sim} Q$ and $b(\overrightarrow{\alpha})$ is fresh in P and Q, then $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$ iff $Q \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$.*

Proof. We prove this by induction on the length n of $\overrightarrow{\alpha}$. If $n = 0$, then $\overrightarrow{\alpha} = \overrightarrow{\varepsilon}$ and neither P nor Q may successfully finish without executing any action. If $n = 1$, then $\overrightarrow{\alpha} = END$. Then, $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd \Leftrightarrow P \stackrel{END}{\rightarrow} \surd$. By the hypothesis that $P \dot{\sim} Q$, $P \stackrel{END}{\rightarrow} \surd \Leftrightarrow Q \stackrel{END}{\rightarrow} \surd$. Finally, $Q \stackrel{END}{\rightarrow} \surd \Leftrightarrow Q \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$.

Suppose that the theorem is true for all $n < k$. Let $\overrightarrow{\alpha}$ be a sequence of length k. Let α be the first action of the sequence and let $\overrightarrow{\beta}$ be a sequence of length $k-1$ such that $\overrightarrow{\alpha} = \alpha.\overrightarrow{\beta}$. Then, $P \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$ if and only if there is a process P' such that $P \stackrel{\alpha}{\rightarrow} P'$ and $P' \stackrel{\overrightarrow{\beta}}{\Rightarrow} \surd$. But if $P \stackrel{\alpha}{\rightarrow} P'$ and $P \dot{\sim} Q$, then there is a process Q' such that $Q \stackrel{\alpha}{\rightarrow} Q'$ and $P' \dot{\sim} Q'$. Now, $\overrightarrow{\beta}$ is a sequence of length shorter than k, so by the induction hypothesis, as $P' \dot{\sim} Q'$ and $P' \stackrel{\overrightarrow{\beta}}{\Rightarrow} \surd$, then $Q' \stackrel{\overrightarrow{\beta}}{\Rightarrow} \surd$. This means that $Q \stackrel{\overrightarrow{\alpha}}{\Rightarrow} \surd$, proving the theorem. □

Theorem D.22. *If $P \dot{\sim} Q$, then $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$.*

Proof. Suppose that $[\overrightarrow{\alpha}] \in \overrightarrow{\mathcal{R}_f}(P)$. Then, there is a clean sequence $\overrightarrow{\sigma}$ such that $[\overrightarrow{\sigma}] = [\overrightarrow{\alpha}]$ (*) and $b(\overrightarrow{\sigma})$ is fresh in P and Q (**). By (*), $P \stackrel{\overrightarrow{\sigma}}{\Rightarrow} \surd$. As $P \dot{\sim} Q$, this, together with (**) and Lemma D.21, implies that $Q \stackrel{\overrightarrow{\sigma}}{\Rightarrow} \surd$, which means that $[\overrightarrow{\alpha}] = [\overrightarrow{\sigma}] \in \overrightarrow{\mathcal{R}_f}(Q)$. Thus, $\overrightarrow{\mathcal{R}_f}(P) \subseteq \overrightarrow{\mathcal{R}_f}(Q)$. The proof that $\overrightarrow{\mathcal{R}_f}(Q) \subseteq \overrightarrow{\mathcal{R}_f}(P)$ is entirely analogous. □

We present some equalities between sets of finite possible runs that are useful to the development of our axiomatization.

Theorem D.23. *The following set equalities are true:*

1. $\overrightarrow{\mathcal{R}_f}(\mathbf{0}) = \emptyset$;
2. $\overrightarrow{\mathcal{R}_f}(END) = \{[END]\}$;
3. $\overrightarrow{\mathcal{R}_f}(\alpha.P) = \{[\alpha.END]\} \circ \overrightarrow{\mathcal{R}_f}(P)$;
4. $\overrightarrow{\mathcal{R}_f}(P_1; P_2) = \overrightarrow{\mathcal{R}_f}(P_1) \circ \overrightarrow{\mathcal{R}_f}(P_2)$;
5. $\overrightarrow{\mathcal{R}_f}(P_1 + P_2) = \overrightarrow{\mathcal{R}_f}(P_1) \cup \overrightarrow{\mathcal{R}_f}(P_2)$;
6. $\overrightarrow{\mathcal{R}_f}(P^*) = (\overrightarrow{\mathcal{R}_f}(P))^*$;
7. $\overrightarrow{\mathcal{R}_f}(P_1|P_2) = \bigcup\{\overrightarrow{\mathcal{R}_f}(\overrightarrow{\alpha} \mid \overrightarrow{\beta}) : [\overrightarrow{\alpha}] \in \overrightarrow{\mathcal{R}_f}(P_1) \text{ and } [\overrightarrow{\beta}] \in \overrightarrow{\mathcal{R}_f}(P_2)\}$;
8. *If* $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$, *then* $\overrightarrow{\mathcal{R}_f}((\nu x)P) = \overrightarrow{\mathcal{R}_f}((\nu x)Q)$;
9. *If* $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(A) \circ \overrightarrow{\mathcal{R}_f}(P) \cup \overrightarrow{\mathcal{R}_f}(B)$ *and* $[END] \notin \overrightarrow{\mathcal{R}_f}(A)$, *then* $\overrightarrow{\mathcal{R}_f}(P) = (\overrightarrow{\mathcal{R}_f}(A))^* \circ \overrightarrow{\mathcal{R}_f}(B)$.

Proof. The proof of the first eight items is straightforward from Table D.1 and Theorem D.22. The ninth item is simply Arden's Rule [49] applied in our context. This application is sound, since the elements of $\overrightarrow{\mathcal{R}_f}(P)$, for any P, are classes of finite strings. □

D.3.2 Language and Semantics

In this section, we present the syntax and semantics of πDL.

Definition D.24. *The πDL language consists of a set Φ of countably many proposition symbols, a set $\mathcal{N}$ of countably many names, the silent action τ, the ending action END, the boolean connectives $\neg$ and $\wedge$, the π-Calculus operators ., ;, +, |, * and ν, a pair of modalities $\langle a \rangle$ and $\langle \overline{a} \rangle$, for each $a \in \mathcal{N}$, and a modality $\langle P \rangle$ for every*

process P, including the atomic processes $\mathbf{0}$ and END. The formulas are defined as follows:

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle a\rangle\varphi \mid \langle\overline{a}\rangle\varphi \mid \langle P\rangle\varphi,$$

where $p \in \Phi$ and P is built as in Definition D.5.

Definition D.25. *A* frame *for πDL is a tuple $\mathcal{F} = (W, \{R_a, R_{\overline{a}}\}_{a\in\mathcal{N}}, R_{END}, R_\tau)$ where*

- *W is a non-empty set of states;*
- *R_a, $R_{\overline{a}}$, for each $a \in \mathcal{N}$, R_{END} and R_τ are the basic binary relations, where $R_{END} = \{(w, w) : w \in W\}$.*

Definition D.26. *A* model *for πDL is a pair $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, where $\mathcal{F}$ is a πDL frame and $\mathbf{V}$ is a valuation function $\mathbf{V} : \Phi \mapsto 2^W$.*

Definition D.27. *We define the* core *of an action α, denoted by $c(\alpha)$, in the following way: $c(a(x)) = a$, $c(\overline{a}\langle x\rangle) = c(\overline{a}\langle\nu x\rangle) = \overline{a}$ and $c(\alpha) = \alpha$, if $\alpha = \tau$ or $\alpha = END$.*

We now define the semantical notion of satisfaction for πDL as follows:

Definition D.28. *Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. The notion of* satisfaction *of a formula φ in a model $\mathcal{M}$ at a state w, notation $\mathcal{M}, w \Vdash \varphi$, can be inductively defined as follows:*

- *$\mathcal{M}, w \Vdash p$ iff $w \in \mathbf{V}(p)$;*
- *$\mathcal{M}, w \Vdash \top$ always;*
- *$\mathcal{M}, w \Vdash \neg\varphi$ iff $\mathcal{M}, w \nVdash \varphi$;*
- *$\mathcal{M}, w \Vdash \varphi_1 \wedge \varphi_2$ iff $\mathcal{M}, w \Vdash \varphi_1$ and $\mathcal{M}, w \Vdash \varphi_2$;*
- *$\mathcal{M}, w \Vdash \langle\kappa\rangle\varphi$ iff there is $w' \in W$ such that $wR_\kappa w'$ and $\mathcal{M}, w' \Vdash \varphi$, where $\kappa = a$ or $\kappa = \overline{a}$, for some $a \in \mathcal{N}$, or $\kappa = \tau$;*
- *$\mathcal{M}, w \Vdash \langle P\rangle\varphi$ iff there is a finite path $(v_0, v_1, \ldots, v_n)$, $n \geq 1$, such that $v_0 = w$, $\mathcal{M}, v_n \Vdash \varphi$ and there is $\overrightarrow{\alpha}$ such that $[\overrightarrow{\alpha}] \in \overrightarrow{\mathcal{R}_f}(P)$, the length of $\overrightarrow{\alpha}$ is n and $(v_{i-1}, v_i) \in R_\kappa$ if and only if $c((\overrightarrow{\alpha})_i) = \kappa$, for $1 \leq i \leq n$. We say that such $\overrightarrow{\alpha}$* matches *the path $(v_0, \ldots, v_n)$.*

If $\mathcal{M}, w \Vdash \varphi$ for every state w, we say that φ is *globally satisfied* in the model $\mathcal{M}$, notation $\mathcal{M} \Vdash \varphi$. If φ is globally satisfied in all models $\mathcal{M}$ of a frame $\mathcal{F}$, we say that φ is *valid* in $\mathcal{F}$, notation $\mathcal{F} \Vdash \varphi$. Finally, if φ is valid in all frames, we say that φ is valid, notation $\Vdash \varphi$. Two formulas φ and ψ are *semantically equivalent* if $\Vdash \varphi \leftrightarrow \psi$.

Theorem D.29. *$\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$ if and only if $\Vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$.*

Proof. ($\Rightarrow$) Suppose that $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$, but $\nVdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$. Then, we may assume, without loss of generality, that there is a model $\mathcal{M}$ and a state v_0 in this model such that $\mathcal{M}, v_0 \Vdash \langle P\rangle p$ (*), but $\mathcal{M}, v_0 \nVdash \langle Q\rangle p$ (**). By Definition D.28, (*) implies that there is a path $(v_0, v_1, \ldots, v_n)$, $n \geq 1$, in $\mathcal{M}$ such that $\mathcal{M}, v_n \Vdash p$ (***) and there is a sequence $\overrightarrow{\alpha}$, such that $[\overrightarrow{\alpha}] \in \overrightarrow{\mathcal{R}_f}(P)$, that matches this path. But as $\overrightarrow{\mathcal{R}_f}(P) = \overrightarrow{\mathcal{R}_f}(Q)$, then $[\overrightarrow{\alpha}] \in \overrightarrow{\mathcal{R}_f}(Q)$. This and (***) imply, by Definition D.28, that $\mathcal{M}, v_0 \Vdash \langle Q\rangle p$, contradicting (**).

($\Leftarrow$) Suppose that $\Vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$ (*), but $\overrightarrow{\mathcal{R}_f}(P) \neq \overrightarrow{\mathcal{R}_f}(Q)$. Then, we may assume, without loss of generality, that there is a clean sequence $\overrightarrow{\alpha}$ such that $[\overrightarrow{\alpha}] \in \overrightarrow{\mathcal{R}_f}(P)$, but $[\overrightarrow{\alpha}] \notin \overrightarrow{\mathcal{R}_f}(Q)$. Let us build a frame $\mathcal{F}$ that consists solely of a path $(v_0, \ldots, v_n)$, $n \geq 1$, such that $R_a = \{(v_{i-1}, v_i) : 1 \leq i \leq n \text{ and } c((\overrightarrow{\alpha})_i) = a\}$. Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$, such that $v_n \in \mathbf{V}(p)$ and $v_i \notin \mathbf{V}(p)$, $0 \leq i < n$. Then, we have a path $(v_0, \ldots, v_n)$ such that $\mathcal{M}, v_n \Vdash p$ and $\overrightarrow{\alpha}$ matches this path. By Definition D.28, $\mathcal{M}, v_0 \Vdash \langle P\rangle p$. However, $[\overrightarrow{\alpha}] \notin \overrightarrow{\mathcal{R}_f}(Q)$, so $(v_0, \ldots, v_n)$ is not matched by any sequence in $\overrightarrow{\mathcal{R}_f}(Q)$. Besides that, there is no other path $(v_0, \ldots, v_m)$, $m \geq 1$, in $\mathcal{M}$ such that $\mathcal{M}, v_m \Vdash p$. Thus, by Definition D.28, $\mathcal{M}, v_0 \nVdash \langle Q\rangle p$, which contradicts (*). □

Corollary D.30. *If $P \dot{\sim} Q$, then $\Vdash \langle P\rangle p \leftrightarrow \langle Q\rangle p$.*

D.3.3 Examples

In this section, we present two simple examples of applications of πDL.

Example D.31. *Let $\mathcal{M}$ be a Kripke model representing the behaviour of a local network in a business office. Suppose that, in this network, there are three computers and two printers, managed by a common server. For a restricted analysis of $\mathcal{M}$ with*

respect only to the printing protocols, we may assume that all that the employees of the office do at the computers is to print documents.

Let s_1 be the communication channel between the computers and the server and s_2 be the communication channel between the server and the printers. Then, $C_i = (\nu c_i)\overline{s_1}\langle c_i\rangle.c_i(p).\overline{p}\langle d\rangle.END$ describes a program that sends a document that can be retrieved through channel d to be printed, $S = (s_1(n).s_2(p).\overline{n}\langle p\rangle\ .END)^*$ describes a program that controls the server and distributes the printers following requests from the computers and $P_j = (\nu p_j)\overline{s_2}\langle p_j\rangle.p_j(d).\tau.END$ describes a program that controls the printer and prints the document (the act of printing is represented by τ) that can be retrieved through channel d.

We may want to verify if $\mathcal{M}$ allows for any computer to print a sequence of any number of documents at any time, no matter what state $\mathcal{M}$ is presently in. If we assume that each computer cannot simultaneously request the printing of more than one document, this is equivalent to checking whether the formula $\langle C_1^* \mid C_2^* \mid C_3^* \mid S \mid P_1^* \mid P_2^*\rangle\top$ is globally satisfied in $\mathcal{M}$. We may also want to verify that the actions of one computer do not affect the actions of the others. For this, we could check, for instance, if $\langle C_i\rangle\top \leftrightarrow [(C_j \mid C_k)^*]\langle C_i\rangle\top$, $i \neq j \neq k$, is globally satisfied in $\mathcal{M}$.

Example D.32. *The parallel composition operator ($\mid$) has a dual role in the π-Calculus. It represents both interleaving and synchronization of processes. In van Benthem's paradigm of* games-as-processes *[51], this could be used to represent simultaneous games, where each player chooses his actions unaware of what are the actions taken by the other player. We consider a concrete example. Let $\mathcal{M}$ be a game board and the proposition symbols w_i, $i = 1, 2$, denote that player i wins if the game reaches a state where w_i is satisfied. Let $\{a, b, c\}$ be the possible actions for player 1 and $\{d, e, f\}$ the possible actions for player 2. Each player has to perform a sequence of three actions, completely unaware of which actions the other player performed or even how many of the three actions the other player performed so far. This means that the two sequences are interleaved in an arbitrary order. Then, $\mathcal{M}, w \Vdash \langle a.b.b.END + a.b.c.END \mid d.d.d.END\rangle w_1$ means that if the game starts in w, there is an interleaved sequence of a, b, b and d, d, d or a, b, c and d, d, d that leads to a victory of player 1. On the other hand, $\mathcal{M}, w \Vdash [a.b.b.END \mid d.d.d.END]w_1$ means that, if the game starts in w, no matter in what order the six actions take*

place, if player 1 plays a, b, b and player 2 plays d, d, d, player 1 is guaranteed to win. This can also be generalized to games with more than two players.

The $|$ operator can also represent synchronization of processes. This allows us to model richer games, where we can get "rounds" of blind, simultaneous games separated by some exchange of information between the players. For instance, in the game described above, suppose now that the two players may select their actions from the same set $\{a, b, c\}$. To differentiate between the sequences of actions of each player, each sequence starts with p_i, $i = 1, 2$. Besides that, each player will now perform the three actions in the following way: a player performs two actions, then informs the other player of one of them and performs the final action. Then we can express properties of this game using formulas of our logic, in an analogous way to the paragraph above. For instance, $\mathcal{M}, w \Vdash [(\nu x)(\nu y)(p_1.b.b.\overline{x}\langle b\rangle.y(w).w.END \mid p_2.a.b.x(z).\overline{y}\langle a\rangle.b.END + p_2.a.b.x(z).\overline{y}\langle b\rangle.b.END)]w_1$ means that if the game starts in w, no matter in what order the four initial and the two final actions take place, if player 2 starts with a, b and finishes with b, then player 1 can always win by playing b, b and then finishing with the action informed by player 2.

This interplay between interleaving and synchronization can then be used to describe a fairly large group of games. A recent paper [52] also works with this idea that concurrency operators can be used to model simultaneous games. The authors use CPDL [6] as a stepping stone to build a concurrent dynamic game logic. However, since CPDL does not admit communication, their logic also has that limitation. As the generalization from CPDL to channel-CPDL has as drawbacks the loss of decidability and the loss of a complete axiomatization, our logic may be better suited for the generalization of the logic presented in [52] to deal with simultaneous games with communication, as we briefly illustrated.

D.3.4 Axiomatic System

We consider the following set of axioms and rules, where p and q are proposition symbols and φ and ψ are formulas.

(PL) Enough propositional logic tautologies

(K) $\vdash [P](p \to q) \to ([P]p \to [P]q)$

(Du) $\vdash [P]p \leftrightarrow \neg\langle P\rangle\neg p$

(0) $\vdash \neg\langle \mathbf{0} \rangle p$

(Rec) $\vdash \langle P^* \rangle p \leftrightarrow p \vee \langle P \rangle \langle P^* \rangle p$

(END) $\vdash \langle END \rangle p \leftrightarrow p$

(FP) $\vdash p \wedge [P^*](p \rightarrow [P]p) \rightarrow [P^*]p$

(Pr) $\vdash \langle \alpha.P \rangle p \leftrightarrow \langle c(\alpha) \rangle \langle P \rangle p$

(MP) If $\vdash \varphi$ and $\vdash \varphi \rightarrow \psi$, then $\vdash \psi$

(SC) $\vdash \langle P_1; P_2 \rangle p \leftrightarrow \langle P_1 \rangle \langle P_2 \rangle p$

(Gen) If $\vdash \varphi$, then $\vdash [P]\varphi$

(NC) $\vdash \langle P_1 + P_2 \rangle p \leftrightarrow \langle P_1 \rangle p \vee \langle P_2 \rangle p$

(νBi) If $P \overset{\nu}{\sim} Q$, then $\vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$

(Sub) If $\vdash \varphi$, then $\vdash \varphi^\sigma$, where σ uniformly substitutes proposition symbols by arbitrary formulas

(PCSub) If $\vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$, then $\vdash \langle P|R \rangle p \leftrightarrow \langle Q|R \rangle p$

(RSub) If $\vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$, then $\vdash \langle (\nu x)P \rangle p \leftrightarrow \langle (\nu x)Q \rangle p$

(Exp) If the Expansion Law can be applied to P, then $\vdash \langle P \rangle p \leftrightarrow \langle Exp(P) \rangle p$

(Ard) If $\vdash \langle P \rangle p \leftrightarrow \langle A; P + B \rangle p$ and $A \overset{END}{\not\rightarrow} \surd$, then $\vdash \langle P \rangle p \leftrightarrow \langle A^*; B \rangle p$

The axioms **(PL)**, **(K)** and **(Du)** and the rules **(MP)**, **(Gen)** and **(Sub)** are standard in the modal logic literature. The soundness of the other axioms and rules follows directly from the set equalities in Theorem D.23 (with the help of Theorem D.29), Theorem D.13, Corollary D.30 and Definition D.25.

Definition D.33. *We define the following relation between processes:* $P \leftrightarrow Q$ *iff* $\vdash \langle P \rangle p \leftrightarrow \langle Q \rangle p$.

Theorem D.34. $\leftrightarrow$ *is a* congruence.

Definition D.35. *Let* $\Omega_k = \{P_1, \ldots, P_k\}$ *be a set of processes such that* $P_i \not\equiv P_j$, *if* $i \neq j$. *Let* $E(\Omega_k) = \{E_1, \ldots, E_k\}$ *such that* $E_i = (P_i, T_i)$, $P_i \leftrightarrow T_i$, $T_i = \sum_j A^i_j; Q^i_j$ *and, for all* (i, j), A^i_j *has no occurrence of* $|$. *We say that* $E(\Omega_k)$ *is* closed *if, for all* (i, j), $Q^i_j \in \Omega_k$.

Theorem D.36. *Let* $P = P_1 \mid P_2$, *where* P *is clean and unrestricted. Then there is* $\overline{P}$ *such that* $P \leftrightarrow \overline{P}$ *and* $\overline{P}$ *has no occurrence of the* $|$ *operator.*

Proof. The proof is by induction on the number n of occurrences of the $|$ operator in P. If $n = 0$, then $\overline{P} = P$ and there is nothing to be done.

If $n = 1$ then the Expansion Law can be applied to P, so we can use **(Exp)** to build pairs (P_i, T_i) that satisfy Definition D.35. Let $P_1 = P$ and Ω_k be the smallest set such that $P_1 \in \Omega_k$ and $E(\Omega_k)$ is closed. Such a set always exists, as otherwise there would be an infinite set of processes $\{A_i : i \in \mathbb{N}\}$ such that $A_i \not\equiv A_j$, if $i \neq j$, and $P \xrightarrow{\alpha_0} A_0 \xrightarrow{\alpha_1} A_1 \ldots$, which, by a careful inspection of Table D.1, cannot happen.

Take the pair E_k. If there is no $Q_j^k = P_k$ (*), then we can substitute in the processes T_i, $1 \leq i < k$, all the occurrences of P_k by T_k. Otherwise, we can use **(Ard)** to substitute the pair (P_k, T_k) by a pair (P_k, T'_k) where (*) holds and then proceed as in the previous case. We then continue this process with the pair E_{k-1} and so on, until we finally get a pair (P_1, T'_1) such that no process in Ω_k occurs in T'_1. By the use of **(Exp)** to build the initial pairs and the fact that neither **(Ard)** nor the substitution process introduce new $|$ operators, we have $\overline{P} = T'_1$. This method, based on the solution of a "system of equations", was inspired by Brzozowski's algebraic method to obtain the regular expression that describes the language accepted by a finite automaton [50].

Suppose that the theorem is true for all $n < k$. Let P have k occurrences of $|$. As $P = P_1|P_2$, we can obtain $\overline{P}$ as $\overline{\overline{P_1}|\overline{P_2}}$. □

Two formulas ϕ and ψ are equi-consistent if $\vdash \phi \leftrightarrow \psi$. By soundness, if ϕ and ψ are equi-consistent, then they are also semantically equivalent.

Theorem D.37 (Completeness)**.** *Every consistent formula is satisfiable in a finite πDL model.*

Proof. Let φ be a consistent formula and let $\mathbf{P}(\varphi)$ be the set of processes that appear in φ. For all $P \in \mathbf{P}(\varphi)$, we can use **($\nu$Bi)**, **(RSub)**, **(MP)** and Theorems D.15, D.34 and D.36 to get a sequence $P \leftrightarrow P' \leftrightarrow P'' \leftrightarrow P'''$, where P' is clean and in ν-prefix form, P'' is also without any occurrence of the $|$ operator and P''' is like P'' but is instead in ν-standard form. We can then obtain an equi-consistent formula $\varphi' = \varphi[P'''/P, P \in \mathbf{P}(\varphi)]$ in which the only π-Calculus operators that appear are ., ;, + and *. All of these operators and its correspondent axioms are analogous to the operators and axioms in standard PDL. Thus, we can follow the completeness proof

of standard PDL (the PDL axioms and its completeness proof are presented in details in [10]), treating the actions as basic PDL programs, to show that φ' is satisfiable in a finite model. As φ and φ' are equi-consistent, they are also semantically equivalent, which means that φ is also satisfied in that same finite model. □

D.4 Final Remarks and Future Work

In this work, we present a Propositional Dynamic Logic for communicating concurrent systems (πDL) in which the programs are described in a language based on the π-Calculus without replication. From the point of view of dynamic logics, this logic represents an improvement on the current scenario, as previous dynamic logics could not effectively deal with both concurrency and communication. CPDL [6] dealt with concurrency, but there was no possibility of communication between the components of a concurrent system. Channel-CPDL [7] models concurrency and communication but it has a "rather complicated" [7] semantics, is undecidable and lacks a complete axiomatization. On the other hand, we are able to provide a simple Kripke semantics for our logic, based on the idea of finite possible runs of processes, build a complete axiomatization for it and show that it has the finite model property.

We also provide a method, in a language with iteration (*) and sequential composition (;) operators, to rewrite any process specification to a form without the parallel composition operator (|) while preserving the set of finite possible runs of the process. This method is based on Brzozowski's algorithm to find the regular expression that corresponds to a finite automaton [50]. We feel that this is an interesting and original application of Brzozowski's idea and that it provides an elegant proof to a key result to the completeness of our axiomatization.

It should also be noticed that, while the | operator can be written out of the specifications, in practice it can be very hard to describe a complex concurrent behaviour without it from the start. Besides that, even though both specifications, with and without |, may be equivalent, the one with | will be more succinct.

It would be interesting to study the complexities of the satisfiability and model-checking problems for this logic and the ones in [43] and [44]. It would also be

interesting to develop an automatic model-checker for these logics, which would involve an efficient algorithmic method to deal with the expansion of parallel processes. We would also like to analyze the issue of self-replicating processes, which was left out of the present work. We would like to study what would change in the logic with the addition of the π-Calculus replication operator (!).

Finally, we would also like to study in more detail the possible connections between our logic and the ideas presented in [52] and analyze how to use our logic as a tool for the description of simultaneous games with communication.