UNIVERSIDADE FEDERAL DE OURO PRETO
Instituto de Ciências Exatas e Biológicas
Departamento de Computação

Lógica Dinâmica Proposicional com Armazenamento, Recuperação e Composição Paralela

Uma formalização em assistente de provas

Ouro Preto, MG

2025

Felipe Péret Moraes Sasdelli

Lógica Dinâmica Proposicional com Armazenamento, Recuperação e Composição Paralela

Uma formalização em assistente de provas

Monografia apresentada ao Curso de Ciência da Computação como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Rodrigo Geraldo Ribeiro

Ouro Preto, MG

2025

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Felipe Péret Moraes Sasdelli

Lógica dinâmica proposicional com armazenamento, recuperação e composição paralela: uma formalização em assistente de provas

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 29 de Agosto de 2025.

Membros da banca

Rodrigo Geraldo Ribeiro (Orientador) - Doutor - Universidade Federal de Ouro Preto
Elon Máximo Cardoso (Examinador) - Doutor - Universidade Federal de Ouro Preto
Guilherme Augusto Anício Drummond do Nascimento (Examinador) - Bacharel - Universidade Federal de Ouro Preto

Rodrigo Geraldo Ribeiro, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 29/08/2025.

Documento assinado eletronicamente por **Rodrigo Geraldo Ribeiro**, **PROFESSOR 3 GRAU**, em 03/09/2025, às 15:59, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0965591** e o código CRC **0D113728**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Resumo

Esta dissertação desenvolve fundamentos teóricos e computacionais para a Lógica Dinâmica Proposicional com Armazenamento e Recuperação ($RSPDL_0$), uma extensão da PDL clássica que incorpora operadores para manipulação de estado. A motivação surge da necessidade de formalismos adequados para verificação de sistemas que implementam operações de salvamento e restauração de estado, fundamentais em linguagens imperativas modernas. A $RSPDL_0$ introduz quatro operadores primitivos: s_1 e s_2 para armazenamento, r_1 e r_2 para recuperação. Estes operadores modelam estados como pares ordenados, permitindo armazenar componentes e recuperá-las posteriormente. A semântica utiliza conjuntos estruturados (S, E, $\star$), uma generalização da semântica de Kripke com um conjunto S de estados, relação de equivalência E, e operação injetiva $\star$ para construir estados compostos. O fragmento estudado corresponde à $RSPDL_0$, que omite os operadores de iteração, teste e composição paralela da PRSPDL completa, focando nos operadores fundamentais de manipulação de estado. Desenvolvemos um sistema axiomático completo que captura as propriedades essenciais destes operadores através de esquemas finitos. Demonstramos corretude através de lemas que estabelecem validade semântica dos axiomas em frames próprios. A completude segue a construção canônica de Henkin-Lindenbaum, relacionando satisfação semântica com derivabilidade sintática via Lema da Verdade. A formalização em Lean 4 demonstra como lógicas modais complexas podem ser implementadas em sistemas de tipos dependentes. Utilizamos tipos indutivos mutuamente recursivos para sintaxe, classes de tipos para estruturas algébricas, e táticas para construção de provas. Esta correspondência entre teoria matemática e implementação verificada valida os resultados obtidos. A implementação atual estabelece fundações sintáticas e semânticas, propriedades do sistema dedutivo, e formaliza parcialmente corretude e completude. O Lema da Verdade para casos modais complexos e a verificação de propriedades do modelo canônico permanecem como trabalho futuro, juntamente com extensões para o fragmento completo da PRSPDL e aplicações em verificação de software.

Abstract

This dissertation develops theoretical and computational foundations for Propositional Dynamic Logic with Storing and Retrieving (RSPDL_0), extending classical PDL with operators for state manipulation. The motivation arises from the need for adequate formalisms to verify systems implementing state saving and restoration operations, fundamental in modern imperative languages. RSPDL_0 introduces four primitive operators: s_1 and s_2 for storing, r_1 and r_2 for retrieving. These operators model states as ordered pairs, allowing components to be stored and later retrieved. The semantics employs structured sets (S, E, $\star$), a generalization of Kripke semantics with a set S of states, equivalence relation E, and injective operation $\star$ for constructing composite states. The studied fragment corresponds to RSPDL_0, which omits iteration, test, and parallel composition operators from the complete PRSPDL, focusing on fundamental state manipulation operators. We develop a complete axiomatic system that captures the essential properties of these operators through finite schemas. We demonstrate soundness through lemmas establishing semantic validity of axioms in proper frames. Completeness follows the canonical Henkin-Lindenbaum construction, relating semantic satisfaction with syntactic derivability via the Truth Lemma. The Lean 4 formalization demonstrates how complex modal logics can be implemented in dependent type systems. We employ mutually recursive inductive types for syntax, type classes for algebraic structures, and tactics for proof construction. This correspondence between mathematical theory and verified implementation validates our results. The current implementation establishes syntactic and semantic foundations, deductive system properties, and partially formalizes soundness and completeness. The Truth Lemma for complex modal cases and verification of canonical model properties remain as future work, alongside extensions to the complete PRSPDL fragment and applications in software verification.

Sumário

1 Introdução

A verificação formal de sistemas computacionais representa um dos desafios mais fundamentais da ciência da computação contemporânea. As linguagens imperativas utilizadas na indústria caracterizam-se fundamentalmente pela manipulação explícita de estado e memória, aspectos que devem ser adequadamente considerados em qualquer abordagem de verificação formal. Este cenário demanda o desenvolvimento de formalismos matemáticos que possam capturar e raciocinar sobre as características específicas dos paradigmas imperativos modernos.

A Lógica Dinâmica Proposicional (PDL), estabelecida por Fischer e Ladner (1979) na década de 1970, constituiu um marco na formalização do raciocínio sobre programas. Sendo uma extensão da lógica modal, a PDL permite expressar propriedades de programas através de fórmulas que combinam operadores modais com construções de programas como composição sequencial, escolha não-determinística e iteração. Sua expressividade permite especificar tanto propriedades de segurança (algo ruim nunca acontece) quanto de vivacidade (algo bom eventualmente acontece), tornando-a uma ferramenta poderosa para verificação formal (HAREL; KOZEN; TIURYN, 2000).

Contudo, as limitações da PDL clássica tornam-se evidentes quando confrontada com características fundamentais das linguagens imperativas modernas. Especificamente, a PDL não oferece mecanismos adequados para modelar operações de armazenamento e recuperação de estado, operações que são centrais em linguagens que manipulam memória explicitamente. Ademais, a ausência de operadores para composição paralela limita sua aplicabilidade em sistemas concorrentes, cada vez mais comuns na computação contemporânea.

A necessidade de estender este formalismo para acomodar tais características motivou o desenvolvimento da Lógica Dinâmica Proposicional com Armazenamento, Recuperação e Composição Paralela (PRSPDL). Esta extensão preserva as propriedades desejáveis da PDL enquanto introduz novos operadores: s_1 e s_2 para armazenamento de estado, r_1 e r_2 para recuperação de estado, e $\|$ para composição paralela. Estes operadores permitem modelar uma classe muito mais ampla de programas, incluindo aqueles que implementam checkpointing, transações, backtracking e execução paralela.

A PRSPDL foi originalmente proposta por Benevides, Freitas e Viana (2011), que introduziram tanto a extensão sintática quanto uma semântica baseada em conjuntos estruturados: uma generalização da semântica de Kripke tradicional. Os autores demonstraram que esta nova semântica permite representar estruturas de dados e manipulá-las através dos novos operadores, expandindo significativamente o poder expressivo da PDL clássica. Importante destacar que Benevides, Freitas e Viana (2011) apresentaram uma axiomatização completa especificamente para o fragmento RSPDL_0, que exclui iteração, teste e composição paralela. Posteriormente, Balbi-

ani e Boudou (2018) estenderam este trabalho, oferecendo uma axiomatização completa para PRSPDL_0, que inclui composição paralela mas mantém a exclusão de iteração. O trabalho de Benevides et al. serve como principal referência teórica para nossa formalização, fornecendo os fundamentos conceituais sobre os quais construímos nossa investigação.

A abordagem de conjuntos estruturados substitui o conjunto simples de estados possíveis da semântica de Kripke por uma estrutura mais rica, equipada com uma relação de equivalência e uma operação de emparelhamento. Esta estruturação permite que estados possam carregar informação sobre dados armazenados, viabilizando a interpretação natural dos operadores de armazenamento e recuperação. A composição paralela, por sua vez, opera sobre pares de estados, permitindo modelar execução simultânea de programas.

Esta dissertação aborda a PRSPDL sob duas perspectivas complementares. Na vertente teórica, desenvolvemos uma apresentação sistemática da lógica, estabelecendo sua sintaxe, semântica e demonstrando suas propriedades fundamentais de correção e completude. Na vertente computacional, exploramos a formalização desta teoria no assistente de prova Lean 4, investigando como conceitos matemáticos abstratos podem ser expressos e verificados computacionalmente.

1.1 Justificativa

A escolha da PRSPDL como objeto de estudo fundamenta-se em sua relevância tanto teórica quanto prática. Teoricamente, a lógica representa uma extensão não-trivial da PDL que preserva suas propriedades desejáveis enquanto incorpora novos operadores para manipulação de estado e paralelismo. Praticamente, oferece um formalismo adequado para especificação e verificação de uma classe importante de sistemas computacionais.

A formalização em Lean 4 constitui um aspecto central deste trabalho. O Lean 4 (MOURA; ULLRICH, 2021), desenvolvido por Moura e Ullrich (2021), oferece características técnicas específicas que são essenciais para nossa formalização: tipos indutivos mutuamente recursivos para representar a sintaxe entrelaçada de fórmulas e programas, classes de tipos para capturar estruturas algébricas como conjuntos estruturados, e um sistema de táticas que facilita a construção de provas complexas sobre propriedades semânticas. A capacidade do Lean 4 de expressar tanto definições matemáticas abstratas quanto algoritmos concretos em uma linguagem unificada permite uma correspondência direta entre a teoria matemática da PRSPDL e sua implementação verificável.

O fragmento RSPDL_0 estudado neste trabalho omite alguns operadores da PRSPDL completa, especificamente iteração, teste e composição paralela. Esta restrição foi adotada por razões pragmáticas: a inclusão destes operadores introduziria complexidades técnicas significativas na formalização, particularmente na construção do modelo canônico e na prova do truth lemma. Nossa abordagem prioriza estabelecer fundamentos sólidos para os operadores de armazenamento e recuperação, com o objetivo de estender a formalização para incluir os operadores restantes em trabalhos futuros.

As principais contribuições deste trabalho são: (1) uma apresentação sistemática da sintaxe e semântica de PRSPDL, com ênfase na fundamentação matemática dos operadores de armazenamento e recuperação; (2) um sistema axiomático correto e completo para o fragmento RSPDL_0; (3) uma formalização computacional verificada em Lean 4, incluindo definições, propriedades fundamentais e parte da prova de completude; e (4) uma demonstração concreta de como lógicas modais complexas podem ser efetivamente formalizadas em assistentes de prova modernos, contribuindo para a crescente literatura sobre verificação formal mecanizada.

2 Sintaxe da PRSPDL

A sintaxe constitui o aspecto estrutural de qualquer linguagem formal, definindo as regras que determinam quais sequências de símbolos constituem expressões bem-formadas. Em lógica matemática, a sintaxe especifica como construir fórmulas válidas a partir de um alfabeto de símbolos primitivos, estabelecendo uma gramática formal que governa a formação de expressões. Esta distinção entre sintaxe e semântica é fundamental: enquanto a sintaxe trata da forma das expressões, a semântica concerne ao seu significado.

Na tradição da lógica formal, a sintaxe é tipicamente especificada através de regras de formação indutivas. Estas regras partem de elementos atômicos (símbolos proposicionais, constantes, variáveis) e definem como construir expressões mais complexas através da aplicação de operadores. Para lógicas modais como a PDL e suas extensões, a sintaxe deve acomodar tanto a estrutura proposicional quanto os operadores modais que expressam propriedades dinâmicas.

O desenvolvimento rigoroso da sintaxe serve múltiplos propósitos fundamentais. Primeiro, estabelece uma base precisa para a definição semântica, garantindo que cada expressão sintática tenha uma interpretação bem-definida. Segundo, fornece o fundamento para sistemas de prova formal, onde a manipulação sintática de fórmulas deve preservar propriedades semânticas. Terceiro, possibilita a implementação computacional da lógica, onde algoritmos devem processar representações sintáticas de fórmulas.

A PRSPDL estende a PDL clássica com novos operadores para manipulação de estado e execução paralela. A linguagem de programas é construída indutivamente a partir de um conjunto de programas atômicos, aplicando operadores de construção que preservam certas propriedades estruturais.

Definição 2.1 (Programas PRSPDL): Seja $Act = \{a, b, c, \ldots\}$ um conjunto enumerável de programas atômicos. O conjunto de programas PRSPDL, denotado por Π, é definido indutivamente pela gramática:

$$\pi ::= a \mid \pi_1; \pi_2 \mid \pi_1 \cup \pi_2 \mid \pi^* \mid \pi_1 \parallel \pi_2 \mid \phi? \mid s_1 \mid s_2 \mid r_1 \mid r_2$$

onde $a \in Act$ é um programa atômico, $\pi_1; \pi_2$ denota composição sequencial onde π_2 executa após a terminação bem-sucedida de π_1, $\pi_1 \cup \pi_2$ denota escolha não-determinística que seleciona entre executar π_1 ou π_2, π^* denota iteração (estrela de Kleene) que permite zero ou mais execuções consecutivas de π, $\pi_1 \parallel \pi_2$ denota composição paralela onde π_1 e π_2 executam simultaneamente, $\phi?$ denota teste que verifica se a fórmula ϕ é verdadeira no estado atual (terminando com sucesso se verdadeira, falhando caso contrário), e s_1, s_2, r_1, r_2 denotam respectivamente os operadores primitivos de armazenamento e recuperação.

A intuição por trás dos novos operadores fundamenta-se na manipulação de estados es-

truturados como pares ordenados. O operador s_1 (armazenar primeiro) armazena o estado atual como primeira componente do estado resultante, de modo que quando o programa s_1 executa no estado s, produz um novo estado (s, t) cuja primeira coordenada é s. Analogamente, s_2 (armazenar segundo) armazena o estado atual s como segunda coordenada do estado resultante (t, s). O operador r_1 (recuperar primeiro) recupera a primeira componente do estado atual, de forma que quando r_1 executa no estado (s, t), termina no estado recuperado s. Similarmente, r_2 (recuperar segundo) recupera a segunda componente do estado atual (s, t) e termina no estado t. Finalmente, quando o programa $\pi_1 \parallel \pi_2$ executa no estado (s_1, t_1), seu efeito é executar π_1 e π_2 em paralelo nos estados s_1 e t_1 respectivamente, produzindo um novo estado (s_2, t_2).

Definição 2.2 (Fórmulas PRSPDL): Seja $\Phi = \{p, q, r, \ldots\}$ um conjunto enumerável de símbolos proposicionais. O conjunto de fórmulas PRSPDL é definido indutivamente pela gramática:

$$\phi ::= \bot \mid p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \langle\pi\rangle\phi$$

onde $\bot$ denota a constante proposicional falsidade, $p \in \Phi$ é um símbolo proposicional atômico, $\neg\phi$ denota negação clássica de ϕ, $\phi_1 \wedge \phi_2$ denota conjunção binária de ϕ_1 e ϕ_2, e $\langle\pi\rangle\phi$ denota o operador modal de possibilidade, expressando que existe pelo menos uma execução do programa π que, quando iniciada no estado atual, termina com sucesso em um estado onde ϕ é verdadeira.

Convenções Sintáticas

Para evitar ambiguidades na interpretação de fórmulas e programas, estabelecemos uma ordem de precedência entre os operadores. Para programas, a precedência decresce na ordem: iteração ($*$), teste ($?$), composição sequencial ($;$), escolha ($\cup$), e composição paralela ($\|$). Para fórmulas, a precedência decresce na ordem: negação ($\neg$), operadores modais ($\langle\cdot\rangle, [\cdot]$), conjunção ($\wedge$), disjunção ($\vee$), e finalmente implicação e bicondicional ($\rightarrow, \leftrightarrow$).

Estas convenções permitem omitir parênteses em muitos casos, tornando as expressões mais legíveis. Por exemplo, a expressão $\langle a; b^*\rangle p \wedge q$ deve ser interpretada como $(\langle a; (b^*)\rangle p) \wedge q$, aplicando a precedência dos operadores da esquerda para a direita dentro de cada nível.

2.1 Operadores Derivados

A partir dos operadores primitivos, definimos operadores derivados que facilitam a expressão de propriedades e programas mais complexos. Estes operadores derivados não introduzem poder expressivo adicional, mas oferecem notações convenientes que tornam as especificações mais legíveis e próximas às construções familiares da matemática e programação.

Operadores Proposicionais Derivados

Os conectivos lógicos clássicos são definidos em termos dos operadores primitivos de negação e conjunção. A constante proposicional verdade é definida como $\top \equiv \neg\, \bot$, represen-

tando uma proposição que é sempre verdadeira. A disjunção $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$ segue as leis de De Morgan, sendo verdadeira quando pelo menos uma das proposições componentes é verdadeira. A implicação $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$ expressa dependência condicional, sendo falsa apenas quando ϕ_1 é verdadeira e ϕ_2 é falsa. O bicondicional $\phi_1 \leftrightarrow \phi_2 \equiv (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)$ captura equivalência lógica, sendo verdadeiro quando ambas as proposições têm o mesmo valor de verdade.

O operador modal de necessidade $[\pi]\phi \equiv \neg\langle\pi\rangle\neg\phi$ constitui o dual do operador de possibilidade. Enquanto $\langle\pi\rangle\phi$ expressa que existe pelo menos uma execução de π que resulta em um estado satisfazendo ϕ, o operador $[\pi]\phi$ expressa que todas as execuções bem-sucedidas de π (se houver alguma) terminam em estados onde ϕ é verdadeira. Esta dualidade reflete a relação fundamental entre possibilidade e necessidade na lógica modal, análoga à dualidade entre quantificadores existencial e universal na lógica de predicados.

Estruturas de Controle Derivadas

A expressividade dos operadores primitivos da PRSPDL permite codificar naturalmente as estruturas de controle fundamentais encontradas em linguagens de programação imperativas. Estas codificações demonstram como conceitos procedurais podem ser expressos declarativamente através de lógica modal.

Os programas fundamentais $skip \equiv \top?$ e $fail \equiv \bot?$ representam respectivamente o programa vazio que sempre termina com sucesso sem alterar o estado, e o programa que invariavelmente falha. Estes programas servem como elementos neutros e absorventes em muitas construções.

O condicional $if\ \phi\ then\ \pi_1\ else\ \pi_2 \equiv \phi?;\pi_1 \cup \neg\phi?;\pi_2$ implementa seleção baseada em condição. Se ϕ é verdadeira no estado atual, o teste $\phi?$ sucede e π_1 é executado; caso contrário, $\phi?$ falha, mas $\neg\phi?$ sucede (já que ϕ é falsa), levando à execução de π_2. A escolha não-determinística garante que exatamente um dos ramos seja tomado.

O laço $while\ \phi\ do\ \pi \equiv (\phi?;\pi)^*;\neg\phi?$ captura iteração condicional. A construção $(\phi?;\pi)^*$ repete zero ou mais vezes a sequência que verifica ϕ e, se verdadeira, executa π. O teste final $\neg\phi?$ garante que o laço termina apenas quando a condição ϕ torna-se falsa. Esta construção corresponde precisamente à semântica usual de laços while em linguagens imperativas.

O comando de repetição $repeat\ \pi\ until\ \phi \equiv \pi;(\neg\phi?;\pi)^*;\phi?$ implementa iteração com teste posterior. Primeiro π é executado incondicionalmente, depois a construção $(\neg\phi?;\pi)^*$ continua repetindo π enquanto ϕ permanece falsa. O teste final $\phi?$ confirma que a condição de saída foi satisfeita. Esta semântica garante que π seja executado pelo menos uma vez, diferentemente do laço while.

Construções mais gerais incluem o comando de seleção múltipla $if\ \phi_1 \rightarrow \pi_1 \mid \ldots \mid \phi_n \rightarrow \pi_n\ fi \equiv \phi_1?;\pi_1 \cup \ldots \cup \phi_n?;\pi_n$, onde múltiplas condições são avaliadas e o primeiro programa correspondente a uma condição verdadeira é executado.

O laço não-determinístico $do\ \phi_1 \rightarrow \pi_1 \mid \ldots \mid \phi_n \rightarrow \pi_n\ od \equiv (\phi_1?;\pi_1 \cup \ldots \cup \phi_n?;\pi_n)^*;(\neg\phi_1 \wedge \ldots \wedge \neg\phi_n)?$ generaliza o while permitindo múltiplas condições de continuação. O laço continua enquanto pelo menos uma das condições ϕ_i for verdadeira, terminando quando todas se tornam falsas simultaneamente.

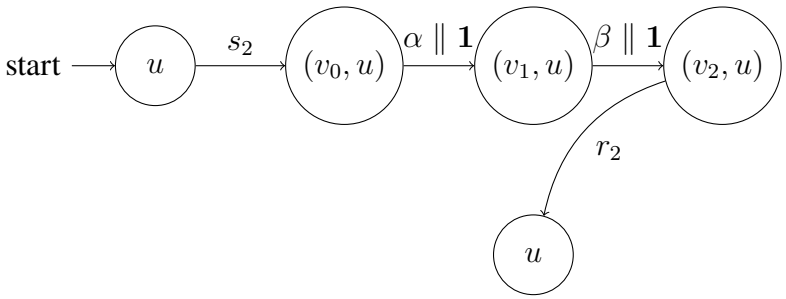
2.2 Exemplos de Programas PRSPDL

Para ilustrar a expressividade da sintaxe da PRSPDL, apresentamos exemplos que demonstram como os operadores de armazenamento, recuperação e composição paralela podem ser combinados para especificar comportamentos complexos de manipulação de estado.

Exemplo 2.1: Apresentamos um programa π_1 que, quando iniciado no estado u, armazena o estado inicial u na segunda coordenada de um par ordenado, depois executa as ações α e β sobre a primeira coordenada do par, sucessivamente e, após isso, retorna ao estado inicial restaurando a segunda coordenada:

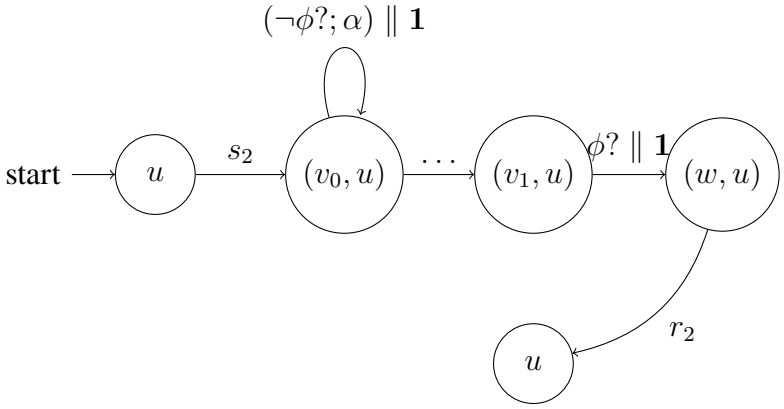
$\pi_1 \equiv s_2;(\alpha \parallel \mathbf{1});(\beta \parallel \mathbf{1});r_2$

onde $\mathbf{1} \equiv \top?$ denota o programa que sempre termina com sucesso.



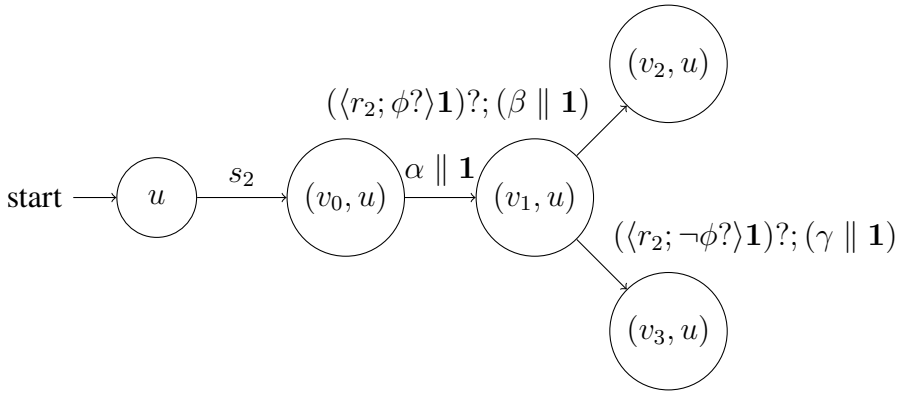
Exemplo 2.2: Apresentamos um programa π_2 que, quando iniciado no estado u, armazena o estado inicial u na segunda coordenada de um par ordenado, depois executa a ação α na primeira coordenada do par atual, até que a propriedade ϕ seja verdadeira e, após isso, retorna ao estado inicial restaurando a segunda coordenada do par atual:

$\pi_2 \equiv s_2; ((\neg\phi?; \alpha) \parallel \mathbf{1})^*; (\phi? \parallel \mathbf{1}); r_2$



Exemplo 2.3: Apresentamos um programa π_3 que, quando iniciado no estado u, armazena o estado inicial u na segunda coordenada de um par ordenado, depois executa a ação α sobre a primeira coordenada do par e, após isso, se a propriedade ϕ é verdadeira no estado inicial, então executa a ação β sobre a primeira coordenada do par atual, caso contrário executa a ação γ sobre ela. É importante notar que a fórmula ϕ é testada no estado inicial e não no estado atual:

$\pi_3 \equiv s_2; (\alpha \parallel \mathbf{1}); ((\langle r_2; \phi?\rangle\mathbf{1})?; (\beta \parallel \mathbf{1}) \cup (\langle r_2; \neg\phi?\rangle\mathbf{1})?; (\gamma \parallel \mathbf{1}))$



Estes exemplos ilustram padrões fundamentais de uso da PRSPDL: execução com possibilidade de retorno ao estado inicial, iteração condicional mantendo backup do estado, e decisão baseada em propriedades do estado inicial armazenado.

3 Semântica da PRSPDL

A semântica formal especifica o significado das construções sintáticas através de interpretações matemáticas precisas. Enquanto a sintaxe define a estrutura das expressões válidas, a semântica estabelece como estas expressões se relacionam com conceitos matemáticos que capturam seu significado pretendido. Na tradição da lógica modal, a semântica é tipicamente fornecida através de estruturas relacionais, onde mundos ou estados são conectados por relações de acessibilidade que interpretam os operadores modais.

Para a PRSPDL, a semântica deve acomodar tanto os aspectos tradicionais da PDL quanto as extensões específicas para manipulação de estado e execução paralela. Os operadores de armazenamento e recuperação requerem uma estrutura de estados mais rica que a tradicionalmente empregada na PDL, enquanto a composição paralela necessita de uma interpretação que capture adequadamente a simultaneidade de execução.

O desenvolvimento da semântica procede em etapas, partindo das estruturas semânticas básicas da PDL e estendendo-as sistematicamente para acomodar as extensões da PRSPDL. Esta abordagem incremental garante compatibilidade com a semântica estabelecida da PDL clássica, ao mesmo tempo que fornece interpretações precisas para os novos operadores.

3.1 Fundamentos Semânticos da PDL

A semântica da PDL clássica baseia-se em estruturas relacionais que interpretam programas como relações entre estados e fórmulas como propriedades de estados. Esta abordagem, conhecida como semântica de mundos possíveis, fornece uma interpretação natural para os operadores modais de possibilidade e necessidade.

Definição 3.1 (Frame PDL): Um frame PDL é um par $\mathcal{F} = (W, \{R_\pi : \pi \text{ é um programa}\})$ onde:

- W é um conjunto não-vazio de estados
- $R_\pi \subseteq W \times W$ é uma relação binária em W para cada programa π

O conjunto W representa o universo de estados possíveis do sistema, enquanto cada relação R_π captura as transições que o programa π pode efetuar entre estados. A intuição é que $(s, t) \in R_\pi$ se e somente se existe uma execução do programa π que, iniciada no estado s, termina com sucesso no estado t.

Definição 3.2 (Modelo PDL): Um modelo PDL é um par $\mathcal{M} = (\mathcal{F}, V)$ onde:

- $\mathcal{F}$ é um frame PDL
- $V : \Phi \to 2^W$ é uma função de valoração que associa a cada símbolo proposicional um subconjunto de W

A função de valoração especifica em quais estados cada proposição atômica é verdadeira, fornecendo a base para a interpretação de fórmulas mais complexas.

Definição 3.3 (Relação de Satisfação): Dado um modelo $\mathcal{M} = (\mathcal{F}, V)$ e um estado $w \in W$, a relação de satisfação $\mathcal{M}, w \models \phi$ é definida indutivamente por:

- $\mathcal{M}, w \models \bot$ nunca
- $\mathcal{M}, w \models p$ se e somente se $w \in V(p)$
- $\mathcal{M}, w \models \neg\phi$ se e somente se $\mathcal{M}, w \not\models \phi$
- $\mathcal{M}, w \models \phi_1 \wedge \phi_2$ se e somente se $\mathcal{M}, w \models \phi_1$ e $\mathcal{M}, w \models \phi_2$
- $\mathcal{M}, w \models \langle\pi\rangle\phi$ se e somente se existe $w' \in W$ tal que $(w, w') \in R_\pi$ e $\mathcal{M}, w' \models \phi$

Esta definição captura a intuição dos operadores modais: $\langle\pi\rangle\phi$ é verdadeira em um estado w quando existe pelo menos uma execução de π partindo de w que alcança um estado onde ϕ é verdadeira.

Para que a semântica seja adequada, as relações R_π devem satisfazer certas condições que refletem a estrutura dos programas construídos indutivamente.

Definição 3.4 (Modelo Padrão): Um modelo PDL $\mathcal{M}$ é padrão quando satisfaz as seguintes condições para todos os programas π, π_1, π_2 e fórmula ϕ:

- *Composição:* $R_{\pi_1;\pi_2} = R_{\pi_1} \circ R_{\pi_2}$
- *Escolha:* $R_{\pi_1 \cup \pi_2} = R_{\pi_1} \cup R_{\pi_2}$
- *Iteração:* $R_{\pi^*} = (R_\pi)^*$
- *Teste:* $R_{\phi?} = \{(w, w) : \mathcal{M}, w \models \phi\}$

onde $R_1 \circ R_2 = \{(w, u) : \exists v.(w, v) \in R_1$ e $(v, u) \in R_2\}$ denota composição relacional e R^* denota o fecho reflexivo-transitivo de R.

Estas condições garantem que a interpretação dos programas compostos corresponde precisamente à composição das interpretações dos programas componentes, preservando a estrutura indutiva da sintaxe na semântica.

3.2 Estados Estruturados

A extensão da PDL para acomodar os operadores de armazenamento e recuperação da PRSPDL requer uma generalização fundamental da noção de estado. Enquanto a PDL tradicional trata estados como entidades atômicas, a PRSPDL necessita de estados que possam ser estruturados como pares ordenados, permitindo armazenar e recuperar componentes específicas.

Definição 3.5 (Conjunto de Estados Estruturados): Um conjunto de estados estruturados é uma tripla $(S, E, \star)$ onde:

- S é um conjunto não-vazio
- $E \subseteq S \times S$ é uma relação de equivalência em S
- $\star : S \times S \to S$ é uma operação binária injetiva

A operação $\star$ permite construir estados compostos a partir de estados componentes, com a injetividade garantindo que a estrutura dos estados compostos seja preservada: se $s_1 \star t_1 = s_2 \star t_2$, então $s_1 = s_2$ e $t_1 = t_2$. A relação de equivalência E captura a noção de que certos estados podem ser considerados "equivalentes" para propósitos específicos da lógica.

A intuição por trás desta estrutura é que estados podem ser tanto simples (elementos primitivos de S) quanto compostos (da forma $s \star t$ para $s, t \in S$). Os operadores de armazenamento transformam estados simples em estados compostos, enquanto os operadores de recuperação extraem componentes específicas de estados compostos.

Definição 3.6 (Frame Estruturado): Um frame estruturado é um par $\mathcal{F} = ((S, E, \star), \{R_\pi : \pi \text{ é um programa}\})$ onde:

- $(S, E, \star)$ é um conjunto de estados estruturados
- $R_\pi \subseteq E$ para cada programa π
- $\mathcal{F}$ constitui um frame PDL com conjunto de estados S

A condição $R_\pi \subseteq E$ expressa que todas as transições respeitam a relação de equivalência, garantindo uma forma de coerência na estrutura semântica.

3.3 Frames Próprios

Para fornecer interpretações específicas aos operadores de armazenamento, recuperação e composição paralela, introduzimos a noção de frame próprio, que estende frames estruturados com condições semânticas precisas para os novos operadores.

Definição 3.7 (Frame Próprio): Um frame estruturado $\mathcal{F}$ é próprio quando satisfaz as seguintes condições para todos os estados $s, t \in S$:

- *Armazenamento Primeiro:* $(s,t) \in R_{s_1}$ se e somente se existe $u \in S$ tal que $s = u$ e $t = u \star s$
- *Armazenamento Segundo:* $(s,t) \in R_{s_2}$ se e somente se existe $u \in S$ tal que $s = u$ e $t = s \star u$
- *Recuperação Primeiro:* $(s,t) \in R_{r_1}$ se e somente se existem $u, v \in S$ tais que $s = u \star v$ e $t = u$
- *Recuperação Segundo:* $(s,t) \in R_{r_2}$ se e somente se existem $u, v \in S$ tais que $s = u \star v$ e $t = v$
- *Composição Paralela:* $(s,t) \in R_{\pi_1 \| \pi_2}$ se e somente se existem $s_1, t_1, s_2, t_2 \in S$ tais que $s = s_1 \star t_1$, $t = s_2 \star t_2$, $(s_1, s_2) \in R_{\pi_1}$ e $(t_1, t_2) \in R_{\pi_2}$

Estas condições capturam precisamente a intuição operacional dos novos operadores. O armazenamento primeiro transforma qualquer estado u no estado composto $u \star u$, efetivamente "duplicando" o estado atual. A recuperação primeiro extrai a primeira componente de um estado composto. A composição paralela opera simultaneamente sobre as duas componentes de um estado composto, aplicando programas distintos a cada componente.

Definição 3.8 (Modelo PRSPDL): Um modelo PRSPDL é um modelo padrão cujo frame subjacente é próprio.

Esta definição integra as condições semânticas da PDL clássica com as extensões específicas da PRSPDL, garantindo uma interpretação coerente para toda a linguagem.

3.4 Propriedades Fundamentais dos Frames Próprios

Os frames próprios satisfazem várias propriedades algébricas importantes que refletem as relações entre os operadores de armazenamento e recuperação. Estas propriedades são fundamentais para o desenvolvimento de um sistema de axiomas completo para a PRSPDL.

Lema 3.1 (Propriedades de Composição): Em qualquer frame próprio $\mathcal{F}$, as seguintes identidades relacionais são válidas:

- *(I):* $R_{s_1} \circ R_{r_1} = \mathrm{Id}_S$ e $R_{s_2} \circ R_{r_2} = \mathrm{Id}_S$
- *(II):* $R_{s_1} \circ R_{r_2} = E$
- *(III):* $(R_{r_1} \circ R_{s_1}) \cap (R_{r_2} \circ R_{s_2}) \subseteq \mathrm{Id}_S$
- *(IV):* $R_{r_1} \circ E = R_{r_2} \circ E$

A propriedade *(I)* estabelece que armazenar e depois recuperar a mesma coordenada não altera o estado. A propriedade *(II)* mostra que armazenar na primeira coordenada e recuperar da segunda produz estados equivalentes. A propriedade *(III)* expressa que estados relacionados simultaneamente por ambos os ciclos recuperação-armazenamento devem ser idênticos. A propri-

edade *(IV)* demonstra que ambos os operadores de recuperação interagem de forma equivalente com a relação de equivalência.

Prova:

(I) Para mostrar $R_{s_1} \circ R_{r_1} = \mathrm{Id}_S$, consideremos $(s, u) \in R_{s_1} \circ R_{r_1}$. Então existe t tal que $(s, t) \in R_{s_1}$ e $(t, u) \in R_{r_1}$. Pela definição de frame próprio, temos $s = s_1, t = s_1 \star t_1$ para alguns s_1, t_1, e também $t = s_2 \star t_2$, $u = s_2$ para alguns s_2, t_2. Logo $s_1 \star t_1 = s_2 \star t_2$, e pela injetividade de $\star$, obtemos $s_1 = s_2$ e $t_1 = t_2$. Portanto $u = s_2 = s_1 = s$. Reciprocamente, se $s = u$, construímos explicitamente $(s, s \star s) \in R_{s_1}$ e $(s \star s, s) \in R_{r_1}$. A prova para $R_{s_2} \circ R_{r_2} = \mathrm{Id}_S$ é análoga, utilizando a segunda coordenada.

(II) Para $(s, t) \in R_{s_1} \circ R_{r_2}$, existe w tal que $(s, w) \in R_{s_1}$ e $(w, t) \in R_{r_2}$. Pela definição de frame próprio, temos $s = s_1, w = s_1 \star t_1$, e também $w = s_2 \star t_2$, $t = t_2$. Logo $s_1 \star t_1 = s_2 \star t_2$, implicando $s_1 = s_2$ e $t_1 = t_2$. Portanto $s = s_1$ e $t = t_2 = t_1$. Para estabelecer $(s, t) \in E$, observamos que s está relacionado ao estado composto $s \star t$ através de R_{s_1}, e t também está relacionado ao mesmo estado composto através de R_{s_2}. Como frames estruturados satisfazem $R_\pi \subseteq E$, ambos s e t estão em E com $s \star t$, e pela transitividade de E, temos $(s, t) \in E$. A recíproca constrói explicitamente a sequência de transições através de $s \star t$.

(III) Se $(s, t) \in (R_{r_1} \circ R_{s_1}) \cap (R_{r_2} \circ R_{s_2})$, então existem estados intermediários i_1 e i_2 tais que $(s, i_1) \in R_{r_1}$, $(i_1, t) \in R_{s_1}$, $(s, i_2) \in R_{r_2}$, e $(i_2, t) \in R_{s_2}$. Pela definição de frame próprio, decompondo cada relação: $s = s_1 \star t_1$, $i_1 = s_1$, $i_1 = s_2$, $t = s_2 \star t_2$, e também $s = s_3 \star t_3$, $i_2 = t_3$, $i_2 = t_4$, $t = s_4 \star t_4$. A análise das igualdades resultantes, aplicando repetidamente a injetividade de $\star$, estabelece que $s_1 = s_2 = t_3 = t_4$ e $t_1 = t_2$, implicando $s = s_1 \star t_1 = s_2 \star t_2 = t$.

(IV) Para mostrar $R_{r_1} \circ E = R_{r_2} \circ E$, consideremos $(s, t) \in R_{r_1} \circ E$. Então existe i tal que $(s, i) \in R_{r_1}$ e $(i, t) \in E$. Pela definição de frame próprio, $s = s_1 \star t_1$ e $i = s_1$. Para estabelecer $(s, t) \in R_{r_2} \circ E$, usamos o estado intermediário t_1: temos $(s, t_1) \in R_{r_2}$ e, aplicando a propriedade (II), $(s_1, t_1) \in E$. Como $i = s_1$ e $(i, t) \in E$, a transitividade de E garante $(t_1, t) \in E$. A direção recíproca é simétrica.

3.5 Validade e Consequência Semântica

Com a semântica estabelecida, podemos definir precisamente as noções de validade e consequência semântica para a PRSPDL, generalizando os conceitos correspondentes da PDL clássica.

Definição 3.9 (Satisfação Global): Uma fórmula ϕ é globalmente satisfeita em um modelo PRSPDL $\mathcal{M}$, denotado $\mathcal{M} \models \phi$, quando $\mathcal{M}, w \models \phi$ para todo estado w no domínio de $\mathcal{M}$.

Definição 3.10 (Validade em Frame): Uma fórmula ϕ é válida em um frame próprio $\mathcal{F}$, denotado $\mathcal{F} \models \phi$, quando $\mathcal{M} \models \phi$ para todo modelo PRSPDL $\mathcal{M}$ baseado em $\mathcal{F}$.

Definição 3.11 (Validade Global): Uma fórmula ϕ é válida, denotado $\models \phi$, quando $\mathcal{F} \models \phi$ para todo frame próprio $\mathcal{F}$.

Definição 3.12 (Equivalência Semântica): Duas fórmulas ϕ_1 e ϕ_2 são semanticamente equivalentes, denotado $\phi_1 \equiv \phi_2$, quando $\models (\phi_1 \leftrightarrow \phi_2)$.

4 Sistema Dedutivo para RSPDL$_0$

Um sistema dedutivo fornece uma caracterização sintática das verdades lógicas de uma linguagem formal, complementando a caracterização semântica fornecida pela teoria de modelos. Enquanto a semântica define validade através de interpretações em estruturas matemáticas, um sistema dedutivo estabelece um conjunto finito de axiomas e regras de inferência que permitem derivar sintaticamente todas as fórmulas válidas da lógica. Para a PRSPDL, o desenvolvimento de um sistema dedutivo apresenta desafios específicos relacionados aos novos operadores de armazenamento e recuperação, que introduzem interdependências complexas que devem ser adequadamente capturadas através de axiomas que reflitam as propriedades algébricas demonstradas na análise semântica.

Por razões de tractabilidade, focamos no fragmento RSPDL$_0$, que omite os operadores de iteração ($\star$), teste (?) e composição paralela ($\|$). Esta restrição permite estabelecer fundamentos sólidos para os operadores centrais de armazenamento e recuperação, com a extensão para o fragmento completo constituindo trabalho futuro. O sistema dedutivo da RSPDL$_0$ baseia-se na lógica proposicional clássica, estendida com axiomas modais que capturam tanto as propriedades gerais dos operadores PDL quanto as características específicas dos novos operadores de manipulação de estado.

Definição 4.1 (Axiomas da RSPDL$_0$): O sistema axiomático da RSPDL$_0$ consiste nos seguintes axiomas, organizados por categoria:

Axiomas Proposicionais:

- *Identidade (I):* $\vdash \phi \to \phi$
- *Combinação (K):* $\vdash \phi \to (\psi \to \phi)$
- *Distributividade (S):* $\vdash (\phi \to (\psi \to \chi)) \to ((\phi \to \psi) \to (\phi \to \chi))$
- *Introdução da Conjunção:* $\vdash \phi \to (\psi \to (\phi \wedge \psi))$
- *Contradição:* $\vdash (\phi \wedge \neg\phi) \to \bot$

Axioma Clássico:

- *Redução ao Absurdo:* $\vdash (\neg\phi \to \bot) \to \phi$

Axiomas Modais Gerais:

- *Composição Modal:* $\vdash [\alpha;\beta]\phi \leftrightarrow [\alpha][\beta]\phi$
- *Escolha Modal:* $\vdash [\alpha \cup \beta]\phi \leftrightarrow ([\alpha]\phi \wedge [\beta]\phi)$
- *Distributividade Modal (K):* $\vdash [\alpha](\phi \to \psi) \to ([\alpha]\phi \to [\alpha]\psi)$

Axiomas de Funcionalidade:

- *Funcionalidade de r_1:* $\vdash \langle r_1 \rangle \phi \to [r_1]\phi$
- *Funcionalidade de r_2:* $\vdash \langle r_2 \rangle \phi \to [r_2]\phi$

Axiomas Temporais:

- *Temporal Direto:* $\vdash \phi \to [s_1]\langle r_1 \rangle \phi$
- *Temporal Inverso:* $\vdash \phi \to [r_1]\langle s_1 \rangle \phi$
- *Temporal Direto 2:* $\vdash \phi \to [s_2]\langle r_2 \rangle \phi$
- *Temporal Inverso 2:* $\vdash \phi \to [r_2]\langle s_2 \rangle \phi$

Axiomas de Domínio:

- *Mesmo Domínio:* $\vdash \langle r_1 \rangle \top \leftrightarrow \langle r_2 \rangle \top$
- *Mesmo Domínio 2:* $\vdash \langle s_1 \rangle \top \leftrightarrow \langle s_2 \rangle \top$

Axiomas de Unicidade:

- *Unicidade:* $\vdash \langle s_1; r_1 \rangle \phi \leftrightarrow [s_1; r_1]\phi$
- *Unicidade 2:* $\vdash \langle s_2; r_2 \rangle \phi \leftrightarrow [s_2; r_2]\phi$

Axiomas de Equivalência Estrutural:

- *Armazenamento e Restauração (Identidade):* $\vdash [s_1; r_2]\phi \to \phi$
- *Alcançabilidade de Equivalência:* $\vdash \phi \to [s_1; r_2]\langle s_1; r_2 \rangle \phi$
- *Iteração de Equivalência:* $\vdash [s_1; r_2]\phi \to [s_1; r_2][s_1; r_2]\phi$

Os axiomas de funcionalidade expressam que os operadores de recuperação são funcionais, significando que cada estado estruturado possui componentes únicos que podem ser recuperadas. Os axiomas temporais capturam a relação fundamental entre armazenamento e recuperação, refletindo a propriedade semântica $R_{s_i} \circ R_{r_i} = \mathrm{Id}_S$. Os axiomas de conversão expressam a interdependência entre operadores de armazenamento e recuperação, estabelecendo que os operadores correspondentes são interdefiníveis em termos de alcançabilidade modal.

Os axiomas de equivalência estrutural governam a relação entre diferentes coordenadas de armazenamento e recuperação, refletindo a propriedade semântica $R_{s_1} \circ R_{r_2} = E$. O axioma de equivalência de armazenamento estabelece que armazenar na primeira coordenada e recuperar da segunda preserva verdade. O axioma de alcançabilidade garante que esta operação é sempre possível, enquanto o axioma de iteração expressa que a operação é idempotente.

Definição 4.2 (Regras de Inferência): O sistema dedutivo emprega as seguintes regras de inferência:

Modus Ponens: Se $\Gamma \vdash \phi$ e $\Gamma \vdash \phi \to \psi$, então $\Gamma \vdash \psi$.

Necessitação: Se $\vdash \phi$, então $\Gamma \vdash [\alpha]\phi$ para qualquer programa α e contexto Γ.

A regra de necessitação requer que a premissa seja derivável sem hipóteses adicionais, refletindo o fato de que apenas verdades lógicas podem ser necessitadas universalmente.

Definição 4.3 (Relação de Dedução): A relação de dedução $\Gamma \vdash \phi$ é definida indutivamente como a menor relação que satisfaz:

- Se $\phi \in \Gamma$, então $\Gamma \vdash \phi$ (Premissa)
- Se ϕ é um axioma, então $\Gamma \vdash \phi$ (Axioma)
- Se $\Gamma \vdash \phi$ e $\Gamma \vdash \phi \to \psi$, então $\Gamma \vdash \psi$ (Modus Ponens)
- Se $\vdash \phi$, então $\Gamma \vdash [\alpha]\phi$ (Necessitação)

Denotamos $\vdash \phi$ quando $\emptyset \vdash \phi$, indicando que ϕ é um teorema derivável apenas dos axiomas.

Lema 4.1 (Weakening): Para todos os conjuntos de fórmulas Γ e Δ e fórmula ϕ, se $\Gamma \subseteq \Delta$ e $\Gamma \vdash \phi$, então $\Delta \vdash \phi$.

Prova: Por indução na estrutura da derivação de $\Gamma \vdash \phi$. Se ϕ é derivada como premissa de Γ, então $\phi \in \Gamma$, e como $\Gamma \subseteq \Delta$, temos $\phi \in \Delta$, permitindo derivar ϕ como premissa de Δ. Se ϕ é um axioma, então pode ser derivada diretamente em Δ. Para modus ponens, se $\Gamma \vdash \chi$ e $\Gamma \vdash \chi \to \phi$, então por hipótese de indução $\Delta \vdash \chi$ e $\Delta \vdash \chi \to \phi$, permitindo aplicar modus ponens para obter $\Delta \vdash \phi$. Para necessitação, se $\emptyset \vdash \chi$, então $\Delta \vdash [\alpha]\chi$ segue diretamente pela regra de necessitação.

Lema 4.2 (Monotonicity): Para todos os conjuntos de fórmulas Γ e Δ e fórmula ϕ, se $\Gamma \vdash \phi$, então $(\Gamma \cup \Delta) \vdash \phi$.

Prova: Segue diretamente da Proposição 4.1, observando que $\Gamma \subseteq (\Gamma \cup \Delta)$.

Lema 4.3 (Teorema da Dedução): Para todo conjunto de fórmulas Γ e fórmulas ϕ e ψ, se $\Gamma \cup \{\phi\} \vdash \psi$, então $\Gamma \vdash (\phi \to \psi)$.

Prova: Por indução na estrutura da derivação de $\Gamma \cup \{\phi\} \vdash \psi$. Se ψ é derivada como premissa, há dois casos: se $\psi \in \Gamma$, aplicamos o axioma $\psi \to (\phi \to \psi)$ seguido de modus ponens; se $\psi = \phi$, usamos o axioma de identidade $\phi \to \phi$. Se ψ é um axioma, aplicamos o axioma $\psi \to (\phi \to \psi)$ seguido de modus ponens. Para modus ponens, se $\Gamma \cup \{\phi\} \vdash \chi$ e $\Gamma \cup \{\phi\} \vdash \chi \to \psi$, então por hipótese de indução temos $\Gamma \vdash (\phi \to \chi)$ e $\Gamma \vdash (\phi \to (\chi \to \psi))$. Aplicamos o axioma de distributividade $(\phi \to (\chi \to \psi)) \to ((\phi \to \chi) \to (\phi \to \psi))$ seguido de

duas aplicações de modus ponens para obter $\Gamma \vdash (\phi \to \psi)$. Para necessitação, se $\emptyset \vdash \chi$, então $\Gamma \vdash [\alpha]\chi$ e aplicamos o axioma $[\alpha]\chi \to (\phi \to [\alpha]\chi)$.

Lema 4.4 (Admissibilidade do Corte): Para todo conjunto de fórmulas Γ e fórmulas ϕ e ψ, se $\Gamma \vdash \phi$ e $\Gamma \cup \{\phi\} \vdash \psi$, então $\Gamma \vdash \psi$.

Prova: Por indução na estrutura da derivação de $\Gamma \cup \{\phi\} \vdash \psi$. Se ψ é derivada como premissa, há dois casos: se $\psi \in \Gamma$, então $\Gamma \vdash \psi$ diretamente; se $\psi = \phi$, então $\Gamma \vdash \psi$ pela hipótese $\Gamma \vdash \phi$. Se ψ é um axioma, então $\Gamma \vdash \psi$ diretamente. Para modus ponens, se $\Gamma \cup \{\phi\} \vdash \chi$ e $\Gamma \cup \{\phi\} \vdash \chi \to \psi$, então por hipótese de indução $\Gamma \vdash \chi$ e $\Gamma \vdash \chi \to \psi$, permitindo aplicar modus ponens para obter $\Gamma \vdash \psi$. Para necessitação, se $\emptyset \vdash \chi$, então $\Gamma \vdash [\alpha]\chi$ diretamente pela regra de necessitação.

5 Corretude para RSPDL$_0$

A corretude (soundness) de um sistema dedutivo estabelece que toda fórmula derivável sintaticamente é semanticamente válida. Este resultado fundamental garante que o sistema axiomático não permite a derivação de fórmulas falsas, constituindo uma propriedade essencial para a confiabilidade do formalismo. Para a RSPDL$_0$, a demonstração de corretude requer verificar que cada axioma é semanticamente válido e que as regras de inferência preservam validade semântica.

A prova de corretude procede em duas etapas principais. Primeiro, estabelecemos a validade semântica de cada axioma individual através de lemas específicos. Para os axiomas proposicionais e modais gerais da PDL, a validade segue de resultados estabelecidos na literatura. Focamos aqui nos axiomas específicos da RSPDL$_0$ que governam os operadores de armazenamento e recuperação. Segundo, provamos que as regras de inferência preservam validade, garantindo que a aplicação das regras a fórmulas válidas resulta em fórmulas válidas. A combinação destes resultados, através de indução na estrutura das derivações, estabelece a corretude do sistema completo.

Para os axiomas específicos da RSPDL$_0$, apresentamos provas detalhadas apenas para representantes de cada categoria, já que axiomas análogos (como os pares envolvendo s_1/r_1 e s_2/r_2) seguem a mesma estrutura lógica com adaptações diretas.

Lema 5.1 (Validade da Composição Modal): Para todos os programas α e β e fórmula ϕ, temos $\models ([\alpha;\beta]\phi) \leftrightarrow ([\alpha][\beta]\phi)$.

Prova: Para a direção da esquerda para a direita, assumimos $\neg([\alpha][\beta]\phi)$ e mostramos $\neg([\alpha;\beta]\phi)$. Por definição de satisfação, existe um estado s tal que $(w,s) \in R_\alpha$ e $\neg([\beta]\phi)$ em s. Isto implica que existe t tal que $(s,t) \in R_\beta$ e $\neg\phi$ em t. Pela propriedade de composição de modelos padrão, $R_{\alpha;\beta} = R_\alpha \circ R_\beta$, portanto $(w,t) \in R_{\alpha;\beta}$, estabelecendo $\neg([\alpha;\beta]\phi)$. A direção inversa procede analogamente, decompondo uma transição via $\alpha;\beta$ em transições separadas via α e β.

Lema 5.2 (Validade da Escolha Modal): Para todos os programas α e β e fórmula ϕ, temos $\models ([\alpha \cup \beta]\phi) \leftrightarrow (([\alpha]\phi) \wedge ([\beta]\phi))$.

Prova: Para a direção da esquerda para a direita, assumimos $\neg(([\alpha]\phi) \wedge ([\beta]\phi))$ e mostramos $\neg([\alpha \cup \beta]\phi)$. Pela lei de De Morgan ($\neg(A \wedge B) \equiv (\neg A \vee \neg B)$), temos $\neg([\alpha]\phi) \vee \neg([\beta]\phi)$. Considerando o primeiro caso, existe s tal que $(w,s) \in R_\alpha$ e $\neg\phi$ em s. Pela propriedade de escolha de modelos padrão, $R_{\alpha\cup\beta} = R_\alpha \cup R_\beta$, portanto $(w,s) \in R_{\alpha\cup\beta}$, estabelecendo $\neg([\alpha \cup \beta]\phi)$. O caso para β é simétrico. Para a direção inversa, se $(w,s) \in R_{\alpha\cup\beta}$ e $\neg\phi$ em s, então por definição da união, $(w,s) \in R_\alpha$ ou $(w,s) \in R_\beta$, contradizendo a hipótese de que ambos $[\alpha]\phi$ e $[\beta]\phi$ são

verdadeiros.

Lema 5.3 (Validade do Axioma K Modal): Para todo programa α e fórmulas ϕ_1 e ϕ_2, temos $\models ([\alpha](\phi_1 \to \phi_2)) \to (([\alpha]\phi_1) \to ([\alpha]\phi_2))$.

Prova: Assumimos $[\alpha](\phi_1 \to \phi_2)$ e $[\alpha]\phi_1$ e mostramos $[\alpha]\phi_2$. Suponha por contradição que $\neg([\alpha]\phi_2)$, então existe s tal que $(w, s) \in R_\alpha$ e $\neg\phi_2$ em s. Por hipótese, $\phi_1 \to \phi_2$ é verdadeira em s e ϕ_1 é verdadeira em s. Por modus ponens, ϕ_2 é verdadeira em s, contradição.

Lema 5.4 (Validade da Funcionalidade de r_1): Para toda fórmula ϕ, temos $\models (\langle r_1\rangle\phi) \to ([r_1]\phi)$.

Prova: Assumimos $\langle r_1\rangle\phi$ e mostramos $[r_1]\phi$. Por definição, existe s tal que $(w, s) \in R_{r_1}$ e ϕ é verdadeira em s. Pela definição de frame próprio, $(w, s) \in R_{r_1}$ implica que existem s_1, s_2 tais que $w = s_1 \star s_2$ e $s = s_1$. Suponha que existe s' tal que $(w, s') \in R_{r_1}$ e $\neg\phi$ em s'. Então existem s'_1, s'_2 tais que $w = s'_1 \star s'_2$ e $s' = s'_1$. Pela injetividade da operação $\star$, temos $s_1 = s'_1$ e $s_2 = s'_2$, portanto $s = s'$, contradizendo $\neg\phi$ em s'.

Lema 5.5 (Validade Temporal Direta): Para toda fórmula ϕ, temos $\models \phi \to ([s_1]\langle r_1\rangle\phi)$.

Prova: Assumimos ϕ em w e mostramos $[s_1]\langle r_1\rangle\phi$ em w. Suponha por contradição que existe s tal que $(w, s) \in R_{s_1}$ e $\neg(\langle r_1\rangle\phi)$ em s. Pela definição de frame próprio, $(w, s) \in R_{s_1}$ implica que existem u, t tais que $w = u$ e $s = u \star t$. Para mostrar $\langle r_1\rangle\phi$ em s, construímos $(s, u) \in R_{r_1}$: pela definição de frame próprio, isto requer existirem s', t' tais que $s = s' \star t'$ e $u = s'$. Como $s = u \star t$ e $u = w$, tomamos $s' = u = w$ e $t' = t$, obtendo a relação desejada $(s, w) \in R_{r_1}$. Como ϕ é verdadeira em w, temos $\langle r_1\rangle\phi$ em s, contradição.

Os axiomas análogos para a segunda coordenada (Temporal Direto 2, Funcionalidade de r_2, e os axiomas de conversão s_2r_2 e r_2s_2) seguem a mesma estrutura lógica, substituindo a primeira coordenada pela segunda nas definições de frame próprio e aplicando as propriedades correspondentes.

Lema 5.6 (Validade do Mesmo Domínio): Temos $\models (\langle r_1\rangle\top) \leftrightarrow (\langle r_2\rangle\top)$.

Prova: Para a direção da esquerda para a direita, assumimos $\langle r_1\rangle\top$ em w, então existe s tal que $(w, s) \in R_{r_1}$. Pela definição de frame próprio, existem s', t tais que $w = s' \star t$ e $s = s'$. Para mostrar $\langle r_2\rangle\top$, construímos $(w, t) \in R_{r_2}$: pela definição de frame próprio, isto é válido pois $w = s' \star t$ e $t = t$. Como $\top$ é sempre verdadeira, temos $\langle r_2\rangle\top$. A direção inversa é simétrica.

Lema 5.7 (Validade da Unicidade): Para toda fórmula ϕ, temos $\models (\langle s_1; r_1\rangle\phi) \leftrightarrow ([s_1; r_1]\phi)$.

Prova: Para a direção da esquerda para a direita, assumimos $\langle s_1; r_1\rangle\phi$ e $\neg([s_1; r_1]\phi)$. Por definição, existem estados s e x tais que $(w, s) \in R_{s_1;r_1}$, ϕ é verdadeira em s, $(w, x) \in R_{s_1;r_1}$, e $\neg\phi$ em x. Pela propriedade semântica $R_{s_1} \circ R_{r_1} = \mathrm{Id}_S$, temos $s = w$ e $x = w$, portanto ϕ e $\neg\phi$ são ambas verdadeiras em w, contradição. Para a direção inversa, assumimos $[s_1; r_1]\phi$ e mostramos $\langle s_1; r_1\rangle\phi$. Pela mesma propriedade semântica, $(w, w) \in R_{s_1;r_1}$, e como $[s_1; r_1]\phi$

implica que ϕ é verdadeira em w, temos $\langle s_1; r_1\rangle\phi$.

Lema 5.8 (Validade da Equivalência de Armazenamento): Para toda fórmula ϕ, temos $\models ([s_1; r_2]\phi) \to \phi$.

Prova: Assumimos $[s_1; r_2]\phi$ em w e mostramos ϕ em w. Pela propriedade semântica $R_{s_1} \circ R_{r_2} = E$, temos $(w, w) \in R_{s_1;r_2}$. Como $[s_1; r_2]\phi$ garante que ϕ é verdadeira em todos os estados alcançáveis via $s_1; r_2$, e w é alcançável de si mesmo, ϕ é verdadeira em w.

Lema 5.9 (Validade da Alcançabilidade de Equivalência): Para toda fórmula ϕ, temos $\models \phi \to ([s_1; r_2]\langle s_1; r_2\rangle\phi)$.

Prova: Assumimos ϕ em w e mostramos $[s_1; r_2]\langle s_1; r_2\rangle\phi$ em w. Para qualquer estado s tal que $(w, s) \in R_{s_1;r_2}$, devemos mostrar $\langle s_1; r_2\rangle\phi$ em s. Pela propriedade semântica, $(s, w) \in R_{s_1;r_2}$ (pela simetria da relação de equivalência), e como ϕ é verdadeira em w, temos $\langle s_1; r_2\rangle\phi$ em s.

Lema 5.10 (Validade da Iteração de Equivalência): Para toda fórmula ϕ, temos $\models ([s_1; r_2]\phi) \to ([s_1; r_2][s_1; r_2]\phi)$.

Prova: Assumimos $[s_1; r_2]\phi$ em w e mostramos $[s_1; r_2][s_1; r_2]\phi$ em w. Para qualquer estado s tal que $(w, s) \in R_{s_1;r_2}$, devemos mostrar $[s_1; r_2]\phi$ em s. Para qualquer estado t tal que $(s, t) \in R_{s_1;r_2}$, pela transitividade da relação de equivalência, $(w, t) \in R_{s_1;r_2}$. Como $[s_1; r_2]\phi$ em w, temos ϕ em t, estabelecendo $[s_1; r_2]\phi$ em s.

Lema 5.11 (Preservação por Modus Ponens): Se $\models \phi_1$ e $\models \phi_1 \to \phi_2$, então $\models \phi_2$.

Prova: Para qualquer modelo M e estado w, temos $(M, w) \models \phi_1$ e $(M, w) \models \phi_1 \to \phi_2$. Por definição de implicação, $(M, w) \models \phi_2$.

Lema 5.12 (Preservação por Necessitação): Se $\models \phi$, então $\models [\alpha]\phi$ para qualquer programa α.

Prova: Para qualquer modelo M, estado w, assumimos $\neg([\alpha]\phi)$ em w. Então existe s tal que $(w, s) \in R_\alpha$ e $\neg\phi$ em s. Mas por hipótese $\models \phi$, portanto $(M, s) \models \phi$, contradição.

Teorema 5.1 (Corretude Geral): Para todo conjunto de fórmulas Γ e fórmula ϕ, se $\Gamma \vdash \phi$ e para toda $\psi \in \Gamma$ temos $\models \psi$, então $\models \phi$.

Prova: Por indução na estrutura da derivação de $\Gamma \vdash \phi$. Se ϕ é uma premissa de Γ, então $\phi \in \Gamma$ e por hipótese $\models \phi$. Se ϕ é um axioma, então $\models \phi$ pelos Lemas 5.1-5.10. Para modus ponens, se $\Gamma \vdash \chi$ e $\Gamma \vdash \chi \to \phi$, então por hipótese de indução $\models \chi$ e $\models \chi \to \phi$, e pelo Lema 5.11, $\models \phi$. Para necessitação, se $\emptyset \vdash \chi$ implica $\Gamma \vdash [\alpha]\chi$, então por hipótese de indução $\models \chi$, e pelo Lema 5.12, $\models [\alpha]\chi$.

Teorema 5.2 (Corretude): Para toda fórmula ϕ, se $\vdash \phi$, então $\models \phi$.

Prova: Aplicação direta do Teorema 5.1 com $\Gamma = \emptyset$.

6 Completude para RSPDL$_0$

A completude de um sistema dedutivo estabelece que toda fórmula semanticamente válida é sintaticamente derivável. Este resultado fundamental garante que o sistema axiomático é suficientemente expressivo para capturar todas as verdades semânticas do formalismo. Para a RSPDL$_0$, a demonstração de completude segue a abordagem clássica de construção de modelos canônicos através do Lema de Lindenbaum.

A prova de completude procede em três etapas principais. Primeiro, estabelecemos propriedades de consistência e definimos conjuntos maximais consistentes. Segundo, utilizamos o Lema de Lindenbaum para estender qualquer conjunto consistente a um conjunto maximal consistente. Terceiro, construímos um modelo canônico a partir destes conjuntos e demonstramos que este modelo satisfaz todas as propriedades necessárias da RSPDL$_0$. A combinação destes resultados estabelece que toda fórmula válida é derivável no sistema.

Estado da Formalização: Apresentamos uma prova matemática completa da completude, acompanhada de formalização parcial em Lean 4. A implementação estabelece rigorosamente as fundações da construção de Lindenbaum, incluindo definições de consistência, construções iterativas e preservação de propriedades estruturais. Dois componentes críticos permanecem não formalizados: o Lema da Verdade e as propriedades específicas do modelo canônico para RSPDL$_0$. Estes são marcados explicitamente no texto e representam direções para desenvolvimento futuro.

6.1 Consistência e Conjuntos Maximais Consistentes

Definição 6.1 (Consistência): Um conjunto de fórmulas Γ é *consistente* se e somente se $\Gamma \nvdash \bot$.

Definição 6.2 (Conjunto Maximal Consistente): Um conjunto de fórmulas Γ é *maximal consistente* se e somente se Γ é consistente e para toda fórmula $\phi \notin \Gamma$, o conjunto $\Gamma \cup \{\phi\}$ é inconsistente.

Lema 6.1 (Equivalência por Dedução): Para qualquer conjunto Γ e fórmula ϕ, temos $\Gamma \vdash \phi$ se e somente se $\Gamma \cup \{\neg\phi\}$ é inconsistente.

Prova: Para a direção da esquerda para a direita, assumimos $\Gamma \vdash \phi$ e mostramos que $\Gamma \cup \{\neg\phi\}$ é inconsistente. Por monotonicidade, $\Gamma \cup \{\neg\phi\} \vdash \phi$. Adicionalmente, $\Gamma \cup \{\neg\phi\} \vdash \neg\phi$ por premissa. Aplicando o axioma de introdução da conjunção, obtemos $\Gamma \cup \{\neg\phi\} \vdash \phi \wedge \neg\phi$. Pelo axioma de contradição, $\Gamma \cup \{\neg\phi\} \vdash (\phi \wedge \neg\phi) \to \bot$. Por modus ponens, $\Gamma \cup \{\neg\phi\} \vdash \bot$.

Para a direção inversa, assumimos $\Gamma \cup \{\neg\phi\}$ é inconsistente. Então $\Gamma \cup \{\neg\phi\} \vdash \bot$. Pelo

teorema da dedução, $\Gamma \vdash \neg\phi \to \bot$. Pelo axioma de redução clássica, $\Gamma \vdash ((\neg\phi) \to \bot) \to \phi$. Por modus ponens, $\Gamma \vdash \phi$.

Lema 6.2 (Propriedade de Não-Contradição): Se Γ é um conjunto maximal consistente e $\phi \in \Gamma$, então $\neg\phi \notin \Gamma$.

Prova: Suponha por contradição que $\phi \in \Gamma$ e $\neg\phi \in \Gamma$. Então $\Gamma \vdash \phi$ e $\Gamma \vdash \neg\phi$ por premissa. Aplicando introdução da conjunção e o axioma de contradição como no lema anterior, obtemos $\Gamma \vdash \bot$, contradizendo a consistência de Γ.

6.2 O Lema de Lindenbaum

A construção de Lindenbaum procede através de uma enumeração de todas as fórmulas e uma extensão iterativa do conjunto inicial.

Definição 6.3 (Função de Inserção): Para um conjunto Γ e fórmula ϕ, definimos:

$$\text{insert}(\phi, \Gamma) = \begin{cases} \Gamma \cup \{\phi\} & \text{se } \Gamma \cup \{\phi\} \text{ é consistente} \\ \Gamma \cup \{\neg\phi\} & \text{caso contrário} \end{cases}$$

Definição 6.4 (Construção Delta): Para um conjunto inicial Γ e enumeração $\phi_0, \phi_1, \phi_2, \ldots$ de todas as fórmulas, definimos: - $\Delta_0(\Gamma) = \Gamma$ - $\Delta_{n+1}(\Gamma) = \text{insert}(\phi_n, \Delta_n(\Gamma))$

Definição 6.5 (Conjunto Limite): $\max(\Gamma) = \bigcup_{n=0}^{\infty} \Delta_n(\Gamma)$

Lema 6.3 (Disjunção de Consistência): Se Γ é consistente, então $\Gamma \cup \{\phi\}$ é consistente ou $\Gamma \cup \{\neg\phi\}$ é consistente.

Prova: Suponha por contradição que ambos $\Gamma \cup \{\phi\}$ e $\Gamma \cup \{\neg\phi\}$ são inconsistentes. Pelo Lema 6.1, temos $\Gamma \vdash \neg\phi$ e $\Gamma \vdash \neg\neg\phi$. Por redução clássica, $\Gamma \vdash \phi$. Então $\Gamma \vdash \phi \wedge \neg\phi$, e pelo axioma de contradição, $\Gamma \vdash \bot$, contradizendo a consistência de Γ.

Lema 6.4 (Preservação de Consistência): Se Γ é consistente, então $\text{insert}(\phi, \Gamma)$ é consistente para qualquer fórmula ϕ.

Prova: Por definição da função insert e pelo Lema 6.3, a função sempre escolhe uma extensão consistente.

Lema 6.5 (Indução de Consistência): Se Γ é consistente, então $\Delta_n(\Gamma)$ é consistente para todo $n \geq 0$.

Prova: Por indução em n. O caso base $n = 0$ segue da hipótese. O passo indutivo segue do Lema 6.4.

Lema 6.6 (Suporte Finito de Derivações): Se $\Gamma \vdash \phi$, então existe um subconjunto finito $\Delta \subseteq \Gamma$ tal que $\Delta \vdash \phi$.

Prova: Por indução na estrutura da derivação. Se ϕ é uma premissa, então $\{\phi\} \vdash \phi$. Se ϕ é um axioma, então $\emptyset \vdash \phi$. Para modus ponens, se $\Gamma \vdash \psi$ e $\Gamma \vdash \psi \to \phi$ com suportes finitos Δ_1 e Δ_2 respectivamente, então $\Delta_1 \cup \Delta_2 \vdash \phi$. Para necessitação, se $\emptyset \vdash \psi$, então $\emptyset \vdash [\alpha]\psi$.

Lema 6.7 (Subconjuntos Finitos em Deltas): Se Δ é finito e $\Delta \subseteq \max(\Gamma)$, então existe n tal que $\Delta \subseteq \Delta_n(\Gamma)$.

Prova: Se $\Delta = \emptyset$, então $\Delta \subseteq \Delta_0(\Gamma) = \Gamma$ trivialmente. Caso contrário, representamos Δ como um conjunto finito e procedemos por indução na sua estrutura. Para cada $\phi \in \Delta$, como $\phi \in \max(\Gamma) = \bigcup_{n=0}^{\infty} \Delta_n(\Gamma)$, existe algum n_ϕ tal que $\phi \in \Delta_{n_\phi}(\Gamma)$. Construímos indutivamente um limitante superior: para $\Delta = \{\phi\} \cup \Delta'$ onde o resultado vale para Δ' com limitante $N_{\Delta'}$, tomamos $N = \max(n_\phi, N_{\Delta'})$. Por monotonicidade da construção delta, temos $\phi \in \Delta_N(\Gamma)$ e $\Delta' \subseteq \Delta_N(\Gamma)$, logo $\Delta \subseteq \Delta_N(\Gamma)$.

Teorema 6.1 (Lema de Lindenbaum): [Parcialmente formalizado em Lean] Todo conjunto consistente Γ pode ser estendido a um conjunto maximal consistente.

Prova: Mostramos que $\max(\Gamma)$ é maximal consistente.

Consistência: [Formalizado] Suponha por contradição que $\max(\Gamma) \vdash \bot$. Pelo Lema 6.6, existe subconjunto finito $\Delta \subseteq \max(\Gamma)$ tal que $\Delta \vdash \bot$. Pelo Lema 6.7, existe n tal que $\Delta \subseteq \Delta_n(\Gamma)$. Então $\Delta_n(\Gamma) \vdash \bot$, contradizendo o Lema 6.5.

Maximalidade: [Não formalizado] Para qualquer $\phi \notin \max(\Gamma)$, a fórmula ϕ deve ter sido rejeitada em algum estágio n da construção, implicando que $\Delta_n(\Gamma) \cup \{\phi\}$ era inconsistente. Como $\Delta_n(\Gamma) \subseteq \max(\Gamma)$, temos que $\max(\Gamma) \cup \{\phi\}$ é inconsistente.

6.3 Modelo Canônico

Definição 6.6 (Frame Canônico): O frame canônico tem como mundos os conjuntos maximais consistentes e relações definidas por:

$$\Gamma R_\alpha \Delta \text{ se e somente se } \forall \phi([\alpha]\phi \in \Gamma \Rightarrow \phi \in \Delta)$$

Definição 6.7 (Valoração Canônica): Para literal p e conjunto maximal consistente Γ:

$$V_{\text{can}}(p)(\Gamma) \text{ se e somente se } p \in \Gamma$$

Definição 6.8 (Modelo Canônico): $M_{\text{can}} = (F_{\text{can}}, V_{\text{can}})$

Lema 6.8 (Lema da Verdade): [Não formalizado] Para toda fórmula ϕ e conjunto maximal consistente Γ:

$$(M_{\text{can}}, \Gamma) \models \phi \text{ se e somente se } \phi \in \Gamma$$

Lema 6.9 (Propriedades do Frame Canônico): [Não formalizado] O frame canônico satisfaz propriedades relacionais necessárias para modelar adequadamente os operadores da RSPDL$_0$.

6.4 Teorema Principal de Completude

Teorema 6.2 (Completude Contrapositiva): Se $\nvdash \phi$, então existe um modelo próprio padrão M tal que $M \not\models \phi$.

Prova: Se $\nvdash \phi$, então pelo Lema 6.1, $\{\neg\phi\}$ é consistente. Pelo Lema de Lindenbaum, existe conjunto maximal consistente Γ tal que $\{\neg\phi\} \subseteq \Gamma$, logo $\neg\phi \in \Gamma$. Pelo Lema 6.2, $\phi \notin \Gamma$. Pelo Lema da Verdade, $(M_{\text{can}}, \Gamma) \not\models \phi$. Usando o modelo canônico (que satisfaz as propriedades próprias e padrão), temos um contraexemplo à validade global de ϕ.

Teorema 6.3 (Completude): Para toda fórmula ϕ, se $\models \phi$, então $\vdash \phi$.

Prova: Contraposição direta do Teorema 6.2.

Corolário 6.1 (Corretude e Completude): Para toda fórmula ϕ:

$$\vdash \phi \text{ se e somente se } \models \phi$$

Prova: Combinação dos Teoremas de Corretude (5.2) e Completude (6.3).

7 Formalização em Lean 4

A formalização matemática em assistentes de prova representa um avanço significativo na verificação rigorosa de resultados teóricos. Para a RSPDL_0, desenvolvemos uma implementação em Lean 4 que estabelece as fundações sintáticas e semânticas do formalismo, demonstra propriedades estruturais fundamentais, e formaliza parcialmente os teoremas de corretude e completude. Esta implementação serve tanto como validação da correção dos resultados quanto como base para extensões futuras do trabalho.

A formalização segue uma arquitetura modular que separa claramente sintaxe, semântica, sistema axiomático e propriedades dedutivas. Esta estrutura facilita a manutenção do código e permite verificações incrementais de correção. As definições principais capturam precisamente os conceitos matemáticos apresentados nos capítulos anteriores, enquanto os teoremas formalizados estabelecem rigorosamente as propriedades fundamentais do sistema.

A formalização encontra-se disponível no repositório: https://github.com/felipeperet/pdl-parallel-storing/

7.1 Introdução ao Lean 4

Lean 4 é um assistente de prova moderno que combina um sistema de tipos dependentes expressivo com táticas automatizadas para construção de provas. O sistema permite expressar proposições matemáticas como tipos e provas como habitantes destes tipos, seguindo a correspondência de Curry-Howard. Esta abordagem garante que toda prova aceita pelo sistema é logicamente válida por construção.

Consideremos exemplos básicos que ilustram a metodologia de prova em Lean. Uma prova simples por contradição:

```
lemma contradict_example (P : Prop) : ¬¬P → P := by
  intro h_double_neg
  by_contra h_not_P
  exact h_double_neg h_not_P
```

Aqui, a tática `intro` introduz a hipótese, `by_contra` inicia uma prova por contradição, e `exact` aplica diretamente uma função às suas premissas. Para provas por indução estrutural, Lean oferece sintaxe especializada:

```
inductive List (α : Type) where
  | nil : List α
  | cons : α → List α → List α

lemma list_length_nonneg (l : List Nat) : length l ≥ 0 := by
  induction l with
  | nil => simp [length]
  | cons h t ih => simp [length]; exact Nat.succ_pos _
```

A construção `induction ... with` decompõe automaticamente a prova em casos base e indutivo, enquanto `simp` simplifica expressões usando lemas conhecidos. Para propriedades relacionais, Lean permite manipulação direta de existenciais e universais:

```
lemma relation_composition (R S : α → α → Prop) (x z : α) :
    (∃ y, R x y ∧ S y z) → Relation.Comp R S x z := by
  intro ⟨y, hR, hS⟩
  exact ⟨y, hR, hS⟩
```

A notação $\langle$`y, hR, hS`$\rangle$ simultaneamente destrói e constrói tuplas existenciais, proporcionando sintaxe concisa para manipulação de estruturas de dados complexas.

7.2 Definições Principais

7.2.1 Sintaxe

A sintaxe da RSPDL$_0$ é capturada através de tipos indutivos mutuamente recursivos que refletem precisamente a gramática formal. A definição de fórmulas inclui casos para falsidade, átomos proposicionais, negação, conjunção e modalidades diamante:

```
inductive Formula where
  | false : Formula
  | atomic : Literal → Formula
  | neg : Formula → Formula
  | conj : Formula → Formula → Formula
  | diamond : Program → Formula → Formula
```

Programas são definidos simetricamente, incluindo operadores de composição, escolha, e os operadores específicos de armazenamento e recuperação:

```
inductive Program where
  | atomic : Literal → Program
  | comp : Program → Program → Program
  | choice : Program → Program → Program
  | s₁ : Program
  | s₂ : Program
  | r₁ : Program
  | r₂ : Program
```

Operadores derivados são implementados como abreviações que preservam transparência definicional. A modalidade box é definida como:

```
abbrev box (α : Program) (φ : Formula) : Formula :=
  ¬ (⟨α⟩ (¬ φ))
```

enquanto conectivos proposicionais seguem definições clássicas:

```
abbrev impl (φ₁ φ₂ : Formula) : Formula :=
  (¬ φ₁) ∨ φ₂

abbrev bicond (φ₁ φ₂ : Formula) : Formula :=
  (φ₁ → φ₂) ∧ (φ₂ → φ₁)
```

A enumerabilidade de fórmulas é axiomatizada através de funções com propriedade de inversão:

```
axiom encode : Formula → Nat
axiom decode : Nat → Option Formula
axiom countable : ∀ {φ}, decode (encode φ) = some φ
```

Esta axiomatização é suficiente para a construção de Lindenbaum sem requerer implementação concreta de algoritmos de enumeração.

7.2.2 Semântica

A semântica é estruturada em camadas hierárquicas, começando com frames básicos e progredindo para modelos próprios padrão. Um frame consiste de um conjunto não-vazio de mundos e relações de acessibilidade para cada programa:

```
structure Frame where
  W : Type
  [nonempty : Nonempty W]
  R : Program → W → W → Prop
```

Modelos estendem frames com valorações que mapeiam literais proposicionais a subconjuntos de mundos:

```
structure Model where
  F : Frame
  V : Literal → F.W → Prop
```

A relação de satisfação é definida indutivamente sobre a estrutura de fórmulas:

```
def satisfies (M : Model) (w : M.F.W) : Formula → Prop
  | Formula.false => False
  | Formula.atomic ψ => M.V ψ w
  | Formula.neg φ => ¬ satisfies M w φ
  | Formula.conj φ₁ φ₂ => satisfies M w φ₁ ∧ satisfies M w φ₂
  | Formula.diamond α φ => ∃ w', M.F.R α w w' ∧ satisfies M w' φ
```

O caso modal é particularmente importante: satisfação de modalidade diamante requer existência de estado sucessor acessível onde a subfórmula é verdadeira.

Estruturas de estados são modeladas através de classes de tipos que capturam as propriedades algébricas necessárias:

```
class State (S : Type) where
  [nonempty : Nonempty S]
  E : S → S → Prop
  [equiv : Equivalence E]
  star : S → S → S
  [inject : ∀ {s₁ t₁ s₂ t₂}, (star s₁ t₁ = star s₂ t₂) ↔ (s₁ = s₂) ∧ t₁ = t₂]
```

Frames estruturados e próprios são definidos como extensões hierárquicas que adicionam progressivamente restrições sobre as relações de acessibilidade:

```
class Proper (F : Frame) extends Structured F where
  s₁ : ∀ {s' t'}, F.R s₁ s' t' ↔ ∃ s t, (s' = s) ∧ t' = s ⋆ t
  r₁ : ∀ {s' t'}, F.R r₁ s' t' ↔ ∃ s t, (s' = s ⋆ t) ∧ t' = s
  s₂ : ∀ {s' t'}, F.R s₂ s' t' ↔ ∃ s t, (s' = t) ∧ t' = s ⋆ t
  r₂ : ∀ {s' t'}, F.R r₂ s' t' ↔ ∃ s t, (s' = s ⋆ t) ∧ t' = t
```

7.2.3 Sistema Axiomático

O sistema de prova é implementado como sistema de Hilbert com contexto, permitindo derivações a partir de conjuntos arbitrários de premissas. Esta abordagem aproveita naturalmente o sistema de tipos dependentes do Lean, onde proposições são tipos e provas são habitantes destes tipos. A correspondência de Curry-Howard garante que cada prova sintática corresponde a um programa bem-tipado, estabelecendo correção por construção.

Axiomas são enumerados através de um tipo indutivo que especifica precisamente quais fórmulas são axiomaticamente válidas:

```
inductive Axiom : Formula → Prop where
  | functionalR₁ φ : Axiom ((⟨r₁⟩ φ) → ([r₁] φ))
  | functionalR₂ φ : Axiom ((⟨r₂⟩ φ) → ([r₂] φ))
  | temporalForward φ : Axiom (φ → ([s₁] ⟨r₁⟩ φ))
  | temporalBackward φ : Axiom (⟨s₁⟩ ⟨r₁⟩ φ → φ)
  | sameDomain : Axiom ((⟨r₁⟩ ⊤') ↔ (⟨r₂⟩ ⊤'))
  | unicity φ : Axiom ((⟨s₁ ; r₁⟩ φ) ↔ ([s₁ ; r₁] φ))
  | storeRestoreId φ : Axiom (([s₁ ; r₂] φ) → φ)
  | storeRestoreDiamond φ : Axiom (φ → ([s₁ ; r₂] ⟨s₁ ; r₂⟩ φ))
  | storeRestoreIterate φ : Axiom (([s₁ ; r₂] φ) → ([s₁ ; r₂] [s₁ ; r₂] φ))
```

Esta definição estabelece um tipo dependente onde cada construtor especifica um esquema axiomático infinito parametrizado sobre fórmulas arbitrárias.

O sistema dedutivo incorpora regras de inferência através de construtores que preservam tipagem:

```
inductive Deduction : Set Formula → Formula → Prop where
  | premise Γ φ : (φ ∈ Γ) → Deduction Γ φ
  | axiom' Γ φ : Axiom φ → Deduction Γ φ
  | modusPonens Γ φ ψ : Deduction Γ φ → Deduction Γ (φ → ψ) → Deduction Γ ψ
  | necessitation Γ α φ : Deduction ∅ φ → Deduction Γ ([α] φ)
```

A tipagem dependente permite verificação automática de correção sintática. Uma derivação só pode ser construída se existir uma sequência válida de aplicações de regras.

7.3 Lemas e Teoremas Principais

A formalização em Lean estabelece um conjunto robusto de resultados teóricos que podem ser organizados em três categorias principais: propriedades estruturais do sistema dedutivo, conexões entre sintaxe e semântica, e os teoremas fundamentais de corretude e completude. Cada categoria contribui para diferentes aspectos da confiabilidade e expressividade do formalismo.

7.3.1 Propriedades do Sistema Dedutivo

Todo sistema formal requer propriedades estruturais básicas para facilitar a construção e manipulação de provas complexas. Nossa formalização estabelece três propriedades fundamentais que governam o comportamento do sistema dedutivo:

Teorema (Weakening):

```
∀ {Γ Δ : Set Formula} {φ : Formula},
  (Γ ⊆ Δ) → (Γ ⊢ φ) → Δ ⊢ φ
```

Esta propriedade garante que derivações permanecem válidas quando premissas adicionais são introduzidas. A monotonicidade é essencial para composição modular de provas, permitindo que resultados estabelecidos em contextos menores sejam reutilizados em contextos maiores sem necessidade de re-derivação.

Teorema (Teorema da Dedução):

```
∀ {Γ : Set Formula} {φ ψ : Formula},
  (Γ ∪ {φ} ⊢ ψ) → (Γ ⊢ (φ → ψ))
```

O teorema da dedução estabelece equivalência entre derivações hipotéticas e implicações. Esta propriedade é fundamental para raciocínio condicional e permite transformar provas que assumem premissas temporárias em teoremas sobre implicações.

Teorema (Admissibilidade do Corte):

```
∀ {Γ : Set Formula} {φ ψ : Formula},
  (Γ ⊢ φ) → (Γ ∪ {φ} ⊢ ψ) → Γ ⊢ ψ
```

A regra de corte permite eliminar lemas intermediários, combinando uma prova de φ com uma prova que usa φ para estabelecer ψ. Embora não seja primitiva no sistema, sua admissibilidade garante que o poder dedutivo não é limitado pela ausência desta regra.

7.3.2 Propriedades Semânticas

A conexão entre axiomas sintáticos e propriedades semânticas é estabelecida através de lemas que demonstram como frames próprios realizam precisamente as relações especificadas pelos axiomas da RSPDL_0:

Lema (Identidade $\mathbf{Rs}_1$;$\mathbf{Rr}_1$):

```
∀ {F : Frame} [Proper F] {s u : F.W},
  Relation.Comp (F.R s₁) (F.R r₁) s u ↔ s = u
```

Este resultado fundamental estabelece que a composição dos operadores de armazenamento e recuperação da primeira coordenada implementa a relação identidade. A prova utiliza a injetividade da operação $\star$ para mostrar que armazenar um estado e depois recuperar a primeira coordenada retorna exatamente o estado original.

Lema (Equivalência $\mathbf{Rs}_1$;$\mathbf{Rr}_2$):

```
∀ {F : Frame} [Proper F] {s t : F.W},
  Relation.Comp (F.R s₁) (F.R r₂) s t ↔ s ≈ t
```

A composição de armazenamento da primeira coordenada com recuperação da segunda coordenada caracteriza a relação de equivalência entre estados. Esta propriedade captura a intuição de que estados são equivalentes quando podem ser relacionados através de operações de armazenamento/recuperação que preservam informação essencial.

Lema (Unicidade Modal):

```
∀ {F : Frame} [Proper F] {s t : F.W},
  (Relation.Comp (F.R r₁) (F.R s₁) s t ∧
   Relation.Comp (F.R r₂) (F.R s₂) s t) → s = t
```

Este lema estabelece que se dois estados são relacionados simultaneamente pelas composições inversas de ambas as coordenadas, então são idênticos. A propriedade garante unicidade de representação, evitando ambiguidades na interpretação de operações de memória.

7.3.3 Corretude

A demonstração de corretude procede através de lemas individuais que estabelecem a validade semântica de cada categoria de axioma:

Lema (Validade da Funcionalidade):

```
∀ {φ : Formula}, ⊨ (⟨r₁⟩ φ) → ([r₁] φ)
```

A funcionalidade dos operadores de recuperação garante que se é possível alcançar um estado onde φ é verdadeira via r_1, então φ é verdadeira em todos os estados alcançáveis via r_1. Esta propriedade reflete o comportamento determinístico dos operadores de memória.

Lema (Validade Temporal):

```
∀ {φ : Formula}, ⊨ φ → ([s₁] ⟨r₁⟩ φ)
```

A propriedade temporal estabelece que fórmulas verdadeiras permanecem acessíveis após operações de armazenamento. Se φ é verdadeira no estado atual, então após armazenar este estado é possível recuperar um estado onde φ é verdadeira.

Lema (Validade da Unicidade):

```
∀ {φ : Formula}, ⊨ (⟨s₁ ; r₁⟩ φ) ↔ ([s₁ ; r₁] φ)
```

A equivalência entre modalidades diamante e box para a composição $s_1;r_1$ reflete o fato de que esta operação é funcionalmente equivalente à identidade, tornando existência e universalidade coincidentes.

Teorema (Corretude):

```
∀ {φ : Formula}, (⊢ φ) → (⊨ φ)
```

O teorema principal é demonstrado por indução na estrutura de derivações, aplicando os lemas de validade axiomática para casos base e propriedades de preservação semântica para regras de inferência.

7.3.4 Completude

A construção de completude adapta a metodologia canônica padrão às especificidades dos operadores de armazenamento e recuperação:

Definição (Consistência):

```
IsConsistent (Γ : Set Formula) : Prop := ¬ (Γ ⊢ ⊥')
```

A consistência é definida de forma padrão como ausência de derivação de contradição. Esta definição sintática permite caracterizar precisamente quais conjuntos de fórmulas são semanticamente satisfazíveis.

Definição (Consistência Maximal):

```
def IsMaximalConsistent (Γ : Set Formula) : Prop :=
  IsConsistent Γ ∧
  ∀ {φ}, (φ  Γ) → ¬ IsConsistent (Γ ∪ {φ})
```

Um conjunto é maximal consistente quando é consistente e não pode ser estendido consistentemente com nenhuma fórmula adicional. Esta propriedade garante que conjuntos maximais consistentes são "completos" no sentido de que para toda fórmula φ, ou φ pertence ao conjunto ou sua negação pertence.

Construção de Extensão Maximal:

```
def insert : Option Formula → Set Formula → Set Formula
  | none, Γ => Γ
  | some φ, Γ =>
      if IsConsistent (Γ ∪ {φ})
      then Γ ∪ {φ}
      else Γ ∪ {¬ φ}

def delta (Γ : Set Formula) : Nat → Set Formula
  | 0 => Γ
  | n + 1 => insert (decode n) (delta Γ n)

def max (Γ : Set Formula) : Set Formula :=
   n, delta Γ n
```

Estas definições formalizam o procedimento construtivo de extensão maximal consistente. A função `insert` realiza a operação fundamental de decisão: dado um conjunto consistente Γ e uma fórmula φ, determina se φ pode ser consistentemente adicionada a Γ; caso afirmativo, produz $\Gamma \cup \{\varphi\}$, caso contrário, adiciona a negação $\neg\varphi$ para preservar maximalidade. Esta abordagem garante que o conjunto resultante permanece consistente enquanto toma uma decisão definida sobre cada fórmula.

A função `delta` implementa a iteração sistemática sobre a enumeração completa de fórmulas, aplicando `insert` transfinitamente para construir uma sequência crescente de conjuntos $\{\Gamma_n\}$ onde cada estágio incorpora decisões sobre fórmulas adicionais. A função `max` completa a construção através da união de todos os estágios, produzindo um conjunto que é simultaneamente consistente (por preservação em cada estágio) e maximal (por exaustividade da enumera-

ção). Esta formalização é essencial para estabelecer construtivamente a existência de extensões maximais consistentes, fornecendo os mundos necessários para a construção do modelo canônico na demonstração de completude.

Lema (Equivalência Dedutiva):

```
∀ {Γ : Set Formula} {φ : Formula},
  (Γ ⊢ φ) ↔ ¬ IsConsistent (Γ ∪ {¬ φ})
```

Este resultado fundamental conecta derivabilidade com inconsistência, estabelecendo que uma fórmula é derivável se e somente se sua negação torna o conjunto inconsistente.

Lema (Suporte Finito):

```
∀ {Γ : Set Formula} {φ : Formula}, (Γ ⊢ φ) →
  ∃ (Δ : Set Formula), Finite Δ ∧ (Δ ⊆ Γ) ∧ (Δ ⊢ φ)
```

A propriedade de suporte finito garante que toda derivação depende apenas de um subconjunto finito de premissas. Esta propriedade é crucial para a construção de Lindenbaum.

Teorema (Lindenbaum):

```
∀ {Γ : Set Formula}, IsConsistent Γ →
  ∃ (Δ : MaximalConsistentSet), Γ ⊆ Δ.val
```

O Lema de Lindenbaum estabelece que todo conjunto consistente pode ser estendido a um conjunto maximal consistente. A construção utiliza enumeração de fórmulas e extensão iterativa.

Definição (Relação Canônica):

```
canonicalRelation (α : Program) (Γ Δ : MaximalConsistentSet) : Prop :=
  ∀ {φ}, (([α] φ) ∈ Γ.val) → φ ∈ Δ.val
```

O modelo canônico é construído usando conjuntos maximais consistentes como mundos e relações definidas através da contenção de fórmulas modais.

Lema (Lema da Verdade):

```
∀ {φ : Formula} {Γ : canonicalModel.F.W},
  ((canonicalModel, Γ) ⊨ φ) ↔ φ ∈ Γ.val
```

O Lema da Verdade estabelece correspondência bidirecional entre satisfação semântica e pertinência sintática em conjuntos maximais consistentes.

Teorema (Completude):

```
∀ {φ : Formula}, (⊨ φ) → (⊢ φ)
```

O teorema de completude garante que toda fórmula semanticamente válida é sintaticamente derivável. A prova utiliza contraposição através do modelo canônico.

Status da Formalização: A implementação atual estabelece a infraestrutura para completude, incluindo a construção de Lindenbaum e definições do modelo canônico. Componentes que permanecem não formalizados incluem a verificação completa do Lema da Verdade para casos modais complexos e a demonstração de que o modelo canônico satisfaz propriedades de frames próprios. Estas limitações refletem a complexidade técnica inerente à verificação formal de construções canônicas para lógicas modais com operadores de estado não-standard.

8 Conclusão

A formalização em Lean 4 representa uma contribuição metodológica significativa para a área de verificação formal mecanizada. Nossa implementação demonstra como lógicas modais complexas com operadores não-standard podem ser efetivamente formalizadas usando sistemas de tipos dependentes modernos. As definições sintáticas utilizando tipos indutivos mutuamente recursivos capturam precisamente a estrutura entrelaçada de fórmulas e programas, enquanto as classes de tipos permitem expressar estruturas algébricas abstratas como conjuntos estruturados de forma modular e reutilizável. O sistema de táticas do Lean facilita a construção de provas complexas sobre propriedades semânticas, permitindo que o trabalho se concentre na estrutura lógica dos argumentos em vez de detalhes de verificação.

A formalização estabelece um padrão de qualidade para trabalhos futuros em verificação formal de lógicas modais. A correspondência direta entre a teoria matemática apresentada nos capítulos teóricos e sua implementação verificada em Lean fornece uma validação robusta da correção dos resultados. Além disso, a modularidade da implementação facilita extensões futuras, tanto para fragmentos mais expressivos da PRSPDL quanto para lógicas modais relacionadas com características similares.

As limitações atuais do trabalho são bem definidas e oferecem direções claras para desenvolvimento futuro. A restrição ao fragmento RSPDL_0, omitindo iteração, teste e composição paralela, foi uma escolha metodológica que permitiu estabelecer fundamentos sólidos para os operadores centrais de armazenamento e recuperação. A extensão para o fragmento completo da PRSPDL constitui um próximo passo natural que se beneficiará das técnicas e infraestrutura desenvolvidas neste trabalho.

Do ponto de vista computacional, a formalização parcial em Lean identifica precisamente quais componentes da teoria permanecem não mecanizados. O Lema da Verdade para casos modais complexos e a verificação de que o modelo canônico satisfaz propriedades de frames próprios representam desafios técnicos específicos que podem ser abordados através de técnicas mais avançadas de automação de provas e táticas especializadas para lógica modal.

8.1 Trabalhos Futuros

A continuação natural deste trabalho envolve tanto a finalização dos componentes teóricos e computacionais iniciados quanto a exploração de extensões que ampliem o escopo e aplicabilidade da RSPDL_0. As direções identificadas abordam limitações específicas da implementação atual e oportunidades para desenvolvimento de aplicações práticas.

A finalização da formalização da completude em Lean 4 constitui a prioridade imediata. Os componentes principais que permanecem não mecanizados incluem a verificação completa do Lema da Verdade e a demonstração de que o modelo canônico satisfaz as propriedades de frames próprios.

A prova construtiva da enumerabilidade das fórmulas representa outro aspecto técnico fundamental que atualmente é axiomatizado na implementação. O desenvolvimento de funções concretas de codificação e decodificação que satisfaçam as propriedades requeridas eliminaria esta dependência axiomática e fortaleceria os fundamentos computacionais do sistema.

A investigação da decidibilidade da RSPDL_0 também constitui uma questão teórica fundamental com implicações práticas significativas. A adaptação das técnicas estabelecidas para decidibilidade da PDL clássica, particularmente o método de tableaux analíticos e a construção de modelos finitos, deve acomodar as complexidades introduzidas pelos operadores de estado.

Referências

BALBIANI, P.; BOUDOU, J. Iteration-free pdl with storing, recovering and parallel composition: a complete axiomatization. *Journal of Logic and Computation*, v. 28, n. 4, p. 705–731, 2018.

BENEVIDES, M.; GOMES, L.; LOPES, B. Towards determinism in pdl: relations and proof theory. *Journal of Logic and Computation*, 2024. Hal-05023810.

BENEVIDES, M. R. F.; FREITAS, R. de; VIANA, P. Propositional dynamic logic with storing, recovering and parallel composition. *Electronic Notes in Theoretical Computer Science*, Elsevier, v. 269, p. 95–107, 2011.

FISCHER, M. J.; LADNER, R. E. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, v. 18, n. 2, p. 194–211, 1979.

HAREL, D.; KOZEN, D.; TIURYN, J. *Dynamic Logic (Foundations of Computing)*. [S.l.]: MIT Press, 2000. ISBN 978-0262082891.

HOARE, C. A. R. An axiomatic basis for computer programming. *Communications of the ACM*, v. 12, n. 10, p. 576–580, 1969.

MOURA, L. de; ULLRICH, S. The lean 4 theorem prover and programming language. In: *Automated Deduction – CADE 28*. [S.l.]: Springer, 2021. (Lecture Notes in Computer Science), p. 625–635.

NUNES, M. A.; ROGGIA, K. G.; TORRENS, P. H. Soundness-preserving fusion of modal logics in coq. In: NOGUEIRA, C. S.; TEODOROV, C. (Ed.). *Formal Methods: Foundations and Applications*. Cham: Springer, 2025. (Lecture Notes in Computer Science, v. 15403). SBMF 2024.

PRATT, V. R. Semantical considerations on floyd-hoare logic. In: *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*. [S.l.]: IEEE, 1976. p. 109–121.

SILVEIRA, A. A. D.; RIBEIRO, R.; NUNES, M. A.; TORRENS, P.; ROGGIA, K. A sound deep embedding of arbitrary normal modal logics in coq. In: *Proceedings of the XXVI Brazilian Symposium on Programming Languages*. New York, NY, USA: Association for Computing Machinery, 2022. (SBLP '22), p. 17. ISBN 9781450397445. Disponível em: <https://doi.org/10.1145/3561320.3561329>.