

*Abd1: um protótipo de linguagem específica para
programação de raciocínios abdutivos.*

Carlos Eduardo Andrade Oliveira

Março / 2016

Dissertação de Mestrado em Ciência da
Computação

Abd1: um protótipo de linguagem específica para programação de raciocínios abduativos.

Esse documento corresponde à dissertação apresentada à Banca Examinadora no curso de Mestrado em Ciência da Computação da Faculdade Campo Limpo Paulista.

Campo Limpo Paulista, 03 de março de 2016.

Carlos Eduardo Andrade Oliveira

Prof. Dr. Osvaldo Luiz de Oliveira (Orientador)

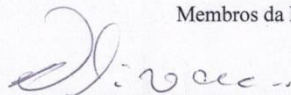
Faculdade Campo Limpo Paulista
Programa de Mestrado em Ciência da Computação

“Abd1: um protótipo de linguagem específica para programação de raciocínios abduativos”

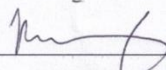
Carlos Eduardo Andrade Oliveira

Dissertação de Mestrado apresentado ao Programa de Mestrado em Ciência da Computação da Faculdade Campo Limpo Paulista, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

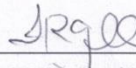
Membros da Banca:



Prof. Dr. Osvaldo Luiz de Oliveira
(Orientador –FACCAMP)



Prof. Dr. Rodrigo Bonacin
(FACCAMP)



Prof. Dr. Ivan Rizzo Guilherme
(UNESP)

Campo Limpo Paulista, 03 de março de 2016.

FICHA CATALOGRÁFICA

Dados Internacionais de Catalogação na Publicação (CIP)

Câmara Brasileira do Livro, São Paulo, Brasil.

Oliveira, Carlos Eduardo Andrade

Abd1: um protótipo de linguagem específica para programação de raciocínios abduativos / Carlos Eduardo Andrade Oliveira. Campo Limpo Paulista, SP: FACCAMP, 2016.

Orientador: Profº. Dr. Osvaldo Luiz de Oliveira

Dissertação (mestrado) – Faculdade Campo Limpo Paulista – FACCAMP.

1. Raciocínio abduativo. 2. Raciocínio automatizado. 3. Lógica. 4. Fatores humanos. 5. Inteligência artificial. 6. Linguagem de programação. I. Oliveira, Osvaldo Luiz de. II. Campo Limpo Paulista. III. Título.

CDD-006.3

AGRADECIMENTOS

A Deus por ter me dado saúde e força para superar as dificuldades.

Ao meu orientador prof. Dr. Osvaldo Luiz de Oliveira, pelos ensinamentos, suporte, paciência, correções e incentivos.

Aos professores e funcionários do programa de Mestrado em Ciência da Computação da FACCAMP.

A todos os colegas da nossa turma e em especial aos que participaram dos grupos de estudos.

A todos os participantes que doaram uma parte de seu tempo e participaram do experimento.

A minha esposa, pelo amor, compreensão, incentivo e apoio incondicional.

Ao Programa PROSUP-CAPES da CAPES pelo apoio recebido na forma de taxas entre 07/2014 e 06/2015.

A InformAction e a todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Resumo: *Abdução em programação lógica é uma técnica poderosa para resolver muitos problemas em Inteligência Artificial incluindo, entre outros, problemas de diagnóstico, entendimento de linguagem natural, planejamento e formulação de argumentos. As abordagens para abdução existentes se enquadram em duas categorias gerais: (1) são sistemas projetados para servirem a diferentes propósitos e que também podem ser utilizados para programação de raciocínios abdutivos; e (2) são sistemas que se declaram específicos para abdução, mas estão desenvolvidos como extensão de algum sistema de programação lógica tradicional (e.g., sistemas que implementam Prolog). Em ambos os casos as abordagens existentes requerem do programador vasto conhecimento de lógica e de programação lógica. Este trabalho identifica e relata diversas dificuldades de programação apresentadas pelas abordagens existentes e propõe um protótipo de linguagem, denominada Abd1, específica para a programação de raciocínios abdutivos, com o objetivo de superar as dificuldades identificadas. Um ambiente de desenvolvimento e execução de programas é desenvolvido para reunir, de forma integrada, um editor, o compilador e o autômato que executa programas escritos na Linguagem Abd1. Por fim, uma pesquisa experimental, em laboratório, é realizada para avaliar a usabilidade da Linguagem Abd1.*

Palavras-chave: raciocínio abductivo, raciocínio automatizado, lógica, fatores humanos, inteligência artificial, linguagem de programação.

Abstract: *Abduction in logic programming is a powerful technique to solve a range of problems in AI including, diagnosis problems, natural language understanding, arguments planning and formulation. The existing approaches for abduction are classified into two general categories: (1) systems designed to serve to different purposes and which can also be used for abductive reasoning programming; and (2) systems that self-declared specific to abduction, but are an extension of a traditional programming logic system (e.g., systems that implement Prolog). In both cases, the existing approaches require of the programmer a extensive knowledge of logic and logic programming. This work identifies and reports several programming difficulties encountered in existing approaches and proposes a language prototype, named Abd1, specific to program abductive reasoning, in order to overcome the identified difficulties. An environment for develop and run programs is designed to assemble, in an integrated manner, the programs editor, the compiler and the automaton that executes Abd1 programs. Lastly, an experimental survey, conducted in a laboratory, is performed to evaluate the usability of the Abd1 Language.*

Keywords: abductive reasoning, automated reasoning, logic, human factors, artificial intelligence, programming language.

Glossário

ACLP	Abductive Constraint Logic Programming
AIDEP	Ambiente Integrado de Desenvolvimento e Execução de Programas
ALP	Abductive Logic Programming
CHR	Constraint Handling Rules
CPB	Código Penal Brasileiro
FNC	Forma Normal Conjuntiva
FND	Forma Normal Disjuntiva
HF	Horn Form
IA	Inteligência Artificial
IC	Integrity Constraints
LP	Lógica Proposicional
MSA	Most Specific Abduction
SQL	Structured Query Language
TCHF	Teoria, Condição, Hipótese e Fatos

Sumário

Capítulo 1 Introdução.....	1
1.1 – O que é abdução?.....	2
1.2 – Abdução e outros tipos de raciocínio	4
1.3 – Escolha de boas hipóteses.....	6
1.4 – Lógica Proposicional como formalismo para representação de problemas de abdução ..	7
1.5 – Sobre este trabalho	10
1.6 – Organização deste texto	11
Capítulo 2 Contexto de Estudo, Objetivo e Metodologia	13
2.1 – Visão geral das abordagens lógicas para abdução.....	14
Linguagens de programação de propósito geral	14
Abordagens que estendem a programação lógica tradicional	15
2.2 – Visão detalhada de uma abordagem para abdução: a Linguagem CHR.....	17
Uso da Linguagem CHR para abdução	17
Sintaxe e semântica operacional de um programa CHR.....	18
2.3 – Dificuldades de uso das abordagens lógicas para programação de raciocínio abdutivo	28
2.4 – Objetivo	31
2.5 – Metodologia	32
Capítulo 3 Projeto da Linguagem Abd1	36
3.1 Fundamentos lógicos do <i>design</i> da linguagem Abd1.....	37
Forma HF	37
Estrutura de raciocínio TCHF	38
Formalização de problemas de abdução com a estrutura de raciocínios TCHF.....	39
Algoritmo Peirce.....	42
3.2 – Visão Informal da Linguagem Abd1	44
3.3 – Gramática da Linguagem Abd1.....	48
Capitulo 4 Implementação da Linguagem e do Ambiente de Desenvolvimento	51
4.1 – Compilador de programas Abd1.....	52
4.2 – Autômato com Pilha Abd1.....	56
4.3 – Ambiente Integrado de Desenvolvimento e Execução de Programas escritos em Abd1 (AIDEP-Abd1).....	60
Capitulo 5 Usabilidade da Linguagem Abd1	63
5.1 – Pesquisa experimental de usabilidade	64
5.2 – Materiais e métodos.....	65

Sujeitos.....	65
Formulário.....	66
Procedimento experimental.....	68
5.3 – Resultados	70
5.4 – Discussão dos resultados.....	72
Facilidade de aprendizagem e de escrita de programas.....	72
Satisfação subjetiva dos participantes.....	75
5.5 – Considerações finais	77
Capítulo 6 Discussão e Conclusão	78
6.1 – Contribuições deste trabalho	79
Contribuições da Linguagem Abd1 para a superação das dificuldades de programação identificadas	80
6.2 – Usabilidade da Linguagem Abd1.....	83
6.3 – Possíveis usos da Linguagem Abd1	85
6.4 – Limitações deste trabalho	86
6.5 – Trabalhos futuros	88
Apêndice I Projeto Gramatical da Linguagem Abd1	90
I.1 – Formalismo de expressão gramatical	91
I.2 – Ambiente de edição de programas.....	91
I.2 – Ambiente de execução de raciocínios abduativos.....	94
Apêndice II Primeiro artigo publicado	96
Apêndice III Segundo artigo publicado.....	112
Apêndice IV Formulário utilizado na pesquisa experimental de usabilidade da Linguagem Abd1	118
Referências.....	126

Capítulo 1

Introdução

Este capítulo inicia com a seguinte questão, tratada na Seção 1.1: “O que é abdução (ou raciocínio abdutivo)?”. A resposta a esta questão é dada pela discussão de três conceituações comumente empregadas em textos de Lógica, Inteligência Artificial, Filosofia e Psicologia. Tendo por objetivo situar os raciocínios abduativos em relação a outros tipos de raciocínio, a Seção 1.2 discute a diferença entre este tipo de raciocínio e outros raciocínios dedutivos e não dedutivos. Raciocínios abduativos formulam hipóteses para explicar fatos observados e ocorre que algumas hipóteses podem explicar melhor os fatos do que outras. A Seção 1.3 discute esta importante característica dos raciocínios abduativos. O assunto da escolha de boas hipóteses é controverso e ainda requer estudos, no entanto a Seção 1.3 apresenta tal assunto de um ponto de vista prático, afim de que possa ser empregado por este trabalho. Raciocínios abduativos podem ser formulados de muitas maneiras diferentes, entre elas como um problema de busca em grafo ou a partir de redes neurais. Este trabalho, por outro lado, fundamenta raciocínio abdutivo por meio de lógica, especificamente da Lógica Proposicional. A Seção 1.4 exemplifica como isto pode ser feito. As soluções existentes para programação de raciocínio abdutivo baseadas em lógica incluem o emprego de linguagens lógicas de propósito geral (e.g., Prolog, Linguagem CHR¹) e sistemas que estendem sistemas lógicos de propósito geral (e.g., ALP, ACLP²) para que executem questões (queries) que avaliam, não a satisfação de objetivos (goals), mas calculam predicados como hipóteses que respondem a um raciocínio abdutivo. As soluções existentes são difíceis de serem programadas porque demandam grande conhecimento de lógica e de programação lógica e, em contraposição a este problema, a Seção 1.5 introduz como este trabalho se

¹ CHR é uma sigla em Inglês para *Constraint Handling Rules*. A Linguagem CHR será discutida em detalhes no Capítulo 2.

² ALP e ACLP são *frameworks* para programação de raciocínio abdutivo e serão discutidos no Capítulo 2. ALP e ACLP são siglas do Inglês para, respectivamente, *Abductive Logic Programming* e *Abductive Constraint Logic Programming*.

situa em relação a isto. Por fim, a Seção 1.6 descreve a organização deste texto em capítulos.

1.1 – O que é abdução?

O termo abdução foi proposto pelo filósofo, matemático e semioticista Charles Sanders Peirce (1867) como uma forma de raciocínio que se diferencia dos raciocínios dedutivos e dos indutivos. Abdução refere-se a um tipo de raciocínio que objetiva formular hipóteses para explicar fatos observados. Este tipo de raciocínio é tão comum quanto o dedutivo, estando presente, por exemplo, quando alguém formula explicações sobre o atraso de uma pessoa a uma reunião, o defeito de um automóvel, as razões que levaram à independência do Brasil, a doença que aflige uma pessoa, quem assassinou uma personagem em um filme de investigação policial, o que irá acontecer com a economia no próximo ano, o que certa pintura abstrata está retratando e o que Drummond (Andrade, 1984) quis dizer quando escreveu “O Rio? É doce. A Vale. Amarga. ...”. Indo além do raciocínio comum, abdução também tem sido associada a um certo tipo de raciocínio (ou método) científico. Como exemplos, para Harman (1965) é raciocínio abduutivo a explicação da seleção natural proposta por Darwin para a origem dos espécimes e para Thagard (1989) é abdução a proposição da existência de um elemento no ar (o oxigênio), por Lavoisier, para explicar o aumento de massa dos corpos submetidos à combustão.

Peirce concebe a abdução, algumas vezes também chamada por ele de retrodução ou presunção, mas seus trabalhos limitam-se à tentativa de conceituá-la e de exemplificar o seu significado dentro de um corpo maior de conhecimento que é a sua Teoria Semiótica. Passados aproximadamente 150 anos da criação do conceito, três noções básicas de abdução se destacam segundo a adequação da noção a estudos em áreas como a Lógica, a Inteligência Artificial, a Psicologia e a Filosofia.

Noção 1: abdução é um tipo de raciocínio que formula hipóteses para explicar fatos observados. Neste sentido abdução é uma forma de criação de teoria para explicar fatos. Esta noção é muito geral e algumas vezes é apropriada para estudos de natureza psicológica ou filosófica.

Noção 2: abdução é um tipo de raciocínio que tem como objetivo formular possíveis hipóteses para explicar fatos observados, considerando uma teoria como fundamento. Segundo esta noção, a abdução compreende:

- T, uma teoria que define o conhecimento sobre algo.
- F, um conjunto de fatos (observações, sintomas, sinais etc.).
- H, um conjunto de hipóteses.

O objetivo do raciocínio abdutivo é encontrar o conjunto de hipóteses H que explique o conjunto de fatos F, sob a perspectiva da teoria T.

A Noção 2 é diferente da Noção 1 porque o elemento teoria é conhecido e utilizado no processo de buscar explicações (hipóteses) para os fatos. Na Noção 2 não há criação de teoria, mas há possivelmente criação de hipóteses H consistentes com uma teoria T. Duas situações podem surgir quando se considera a quantidade de hipóteses que se pode formular (conjunto H). Quando H tem pelo menos uma hipótese, então existe pelo menos uma explicação para o conjunto de fatos F e, se for conveniente, a teoria T pode crescer assimilando as hipóteses. Quando H é vazio então a teoria T não é suficiente para permitir a formulação de hipóteses para explicar o conjunto de fatos. Isto é um sinal de que a teoria precisa ser alterada ou substituída por outra. Em ciência, é comum a alteração ou substituição de uma teoria quando novos fatos são descobertos e a teoria não dá conta de explicá-los (e.g., a Mecânica Relativista é uma evolução da Mecânica Newtoniana).

É comum encontrar, mesmo que de forma oculta, a Noção 2 em textos de Lógica pura ou aplicada. Como exemplos: (1) o silogismo aristotélico é composto por premissa maior (teoria), premissa menor (hipótese) e conclusão (fato) e (2) a execução de um programa Prolog pode ser vista como a busca de respostas (hipóteses) para uma questão (fatos) na base de conhecimento definido pelo programa (teoria).

Noção 3: esta concepção de abdução é tal e qual a da Noção 2, exceto pelo fato de acrescentar à Noção 2 a ideia de que o conjunto de hipóteses deve ser formado não por todas as hipóteses que possam ser formuladas, mas apenas por boas hipóteses, as mais plausíveis. Por exemplo, em diagnóstico médico, é comum encontrar um amplo conjunto de doenças (hipóteses) que explicam os sintomas (fatos) apresentados por um paciente. No entanto, entre as várias doenças possíveis, um médico pode inferir que, apenas duas

ou três doenças, são boas explicações para os sintomas. Esta noção de abdução prescinde da definição de critérios para escolha de boas hipóteses. Critérios para escolha de boas hipóteses têm sido estudados e propostos (e.g., Caroprese & Trubitsyna, 2014; Harman, 1965; Thagard, 1978; Thagard, 1989), mas este assunto permanece controverso e aberto.

Esta noção de abdução tem sido encontrada tanto em trabalhos cujo foco está no campo da Inteligência Artificial (e.g., Dillig & Dillig, 2013; Josephson & Josephson, 1994), quanto em Psicologia (e.g., Magnani, 2009) e em Filosofia (e.g., Thagard, 1989).

Este trabalho adota a Noção 2 de abdução. A escolha da Noção 2 e não da Noção 1 se deve ao fato de que a Noção 1 é ampla e envolve raciocínios que extrapolam o campo da abdução para englobar algo que poderia ser classificado como raciocínio indutivo. A Noção 3 envolve a escolha de boas hipóteses, e este assunto ainda não possui uma boa teoria para ser formalizada por sistemas computacionais. Assim, considerando que este trabalho é sobre abdução a ser realizada por sistemas computacionais, prefere-se a Noção 2 do que a Noção 3. Mesmo assim, critérios pragmáticos de escolha de boas hipóteses serão tratados neste trabalho, o que conduz também a produzir algum resultado aderente à Noção 3.

1.2 – Abdução e outros tipos de raciocínio

Para Peirce (1867) existem três tipos de raciocínio: dedutivo, abdução e indutivo. Considerando um sistema de raciocínio constituído por uma teoria T, um conjunto de hipóteses H e um conjunto de fatos F:

- Um raciocínio dedutivo usa a teoria T e o conjunto de hipóteses H para concluir os fatos F. Este tipo de raciocínio conserva a verdade, no sentido que F é verdade se T é verdade, H é verdade e T e H juntos não formam uma contradição.
- Um raciocínio indutivo formula a teoria T a partir de amostras de fatos F e hipóteses H sobre os fatos F. Neste tipo de raciocínio não há garantia de que T seja verdade, porque T é concebido a partir de amostras de F e de hipóteses H sobre F. Quanto melhor e mais representativas de um fenômeno as amostras e as hipóteses forem, melhor a teoria T explicará o fenômeno.

- Um raciocínio abduutivo formula hipóteses H que explicam os fatos F , tendo como fundamento a teoria T . Neste tipo de raciocínio também não há garantia que uma hipótese $h \in H$ seja verdade, mesmo que T e F sejam considerados verdade. O que se pode afirmar sobre a hipótese h é que ela é uma possível hipótese. Entre todas as hipóteses presentes em H , pode ocorrer que algumas sejam mais plausíveis, críveis, acreditáveis, prováveis (Noção 3 de abdução). O que faz uma hipótese mais plausível do que outra, entretanto, é uma questão que vem sendo estudada, como comentamos na seção anterior. Nos dias atuais, sistemas computacionais para abdução adotam, na prática, critérios para escolha de boas hipóteses para eleger um conjunto de hipóteses que se acredita serem as mais plausíveis.

Exemplo 1.1 (adaptado de Peirce (1867)). O cenário deste exemplo inclui um saco contendo feijões e uma ou mais porções de feijões.

Raciocínio dedutivo: conhecendo-se que a teoria $T = \{ \text{todos os feijões do saco } S \text{ são brancos} \}$ e o conjunto unitário de hipóteses $H = \{ \text{uma porção } P \text{ de feijões foi retirada do saco } S \}$ pode-se concluir o conjunto unitário de fatos $F = \{ \text{os feijões da porção } P \text{ são brancos} \}$. O conjunto de fatos F é indubitavelmente verdade, considerando que as premissas T e H são verdade e que T e H juntos não são uma contradição.

Raciocínio indutivo: considere três amostras (P_1 , P_2 e P_3) de feijões retiradas aleatoriamente do saco S e o conjunto de fatos $F = \{ \text{os feijões da porção } P_1 \text{ são brancos, os feijões da porção } P_2 \text{ são brancos, os feijões da porção } P_3 \text{ são brancos} \}$. Admitindo $H = \{ P_1, P_2 \text{ e } P_3 \text{ representam toda população de feijões do saco } S \}$ pode-se formular a teoria $T = \{ \text{todos os feijões do saco } S \text{ são brancos} \}$. Não há garantia de que T é verdade, mas há motivo para acreditar que T é provavelmente verdade.

Raciocínio abduutivo: conhecendo-se que a teoria $T = \{ \text{todos os feijões do saco } S \text{ são brancos} \}$ e que o conjunto de fatos $F = \{ \text{os feijões da porção } P \text{ são brancos} \}$ pode-se inferir o conjunto de hipóteses $H = \{ \text{a porção de feijões } P \text{ foi retirada do saco } S \}$. Entretanto, o conjunto de hipóteses formulado é apenas uma possibilidade. Não há garantia de que H seja verdade, mesmo que se admita que T e F sejam verdade.

■

Lógica e raciocínios em particular, vem sendo estudados por filósofos e matemáticos desde a antiguidade e hoje existe uma multiplicidade de visões sobre estes assuntos. Assim, uma taxonomia dos tipos de raciocínio que compartimente e seja adequada a toda diversidade de estudos é praticamente impossível. Cabe observar que há muitas propostas taxonômicas diferentes. Como exemplos, uma delas propõe a classificação dos raciocínios em dedutivos e não dedutivos, o que insere o que estamos chamando de abdução e indução nesta última categoria. Outra taxonomia, diferentemente, trata a abdução como um tipo de indução e há, ainda, uma taxonomia que é o oposto disto.

Independentemente da existência destas diferentes taxonomias, este trabalho emprega a que declara raciocínios como sendo dedutivos, indutivos e abdutivos segundo conceituação descrita no início desta seção e está exemplificada no Exemplo 1.1

1.3 – Escolha de boas hipóteses

Algumas hipóteses parecem explicar melhor os fatos do que outras. Por exemplo, considere um cenário constituído por um gramado, irrigadores de jardim e uma pequena teoria restrita apenas a $T = \{ \text{se chover então o gramado fica molhado, se os irrigadores forem ligados então o gramado fica molhado} \}$. Se ocorre o fato $F = \{ \text{o gramado está molhado} \}$, pode-se abduzir as três hipóteses $H = \{ \text{choveu, os irrigadores foram ligados, choveu e os irrigadores foram ligados} \}$. Considerando a teoria T , cada uma das três hipóteses explica o fato em F . No entanto a terceira hipótese, “choveu e os irrigadores foram ligados”, parece menos plausível do que as duas primeiras. O que faz a terceira hipótese menos plausível do que as duas primeiras?

Thagard (1978) comenta que o termo “inferência da melhor explicação” é relativamente novo, mas que a ideia é antiga, tendo sido estudada por filósofos e matemáticos como Peirce, William Whewell, David Hartley, Leibniz e Descartes. Embora antiga, não existe ainda uma teoria geral para a escolha de boas hipóteses. Harman (1965) argumenta que a melhor hipótese é a mais simples, a mais plausível, a que explica mais fatos, e é menos *ad hoc*, i.e., criada especificamente para explicar o que se quer explicar. Thagard propõe que a melhor hipótese é a mais consiliente (explica mais fatos), a mais simples e a que tem maior analogia com hipóteses que explicam outros fatos. Caroprese & Trubitsyna (2014) propõem uma medida para o grau de arbitrariedade de uma hipótese. Quanto

mais arbitrária for uma hipótese, menos preferida ela deve ser. Enfim, o que se tem ainda é especulativo e é muitas vezes difícil formalizar estes conceitos tendo em vista aplicações computacionais.

Este trabalho é prático em relação à escolha de boas hipóteses. Dois critérios, que estão intensivamente presentes em diversos estudos, serão os únicos a serem utilizados:

- ***Simplicidade (ou, inversamente, complexidade)***. Seguindo o princípio de Occam (Russel & Norvig, 2014) é comum a muitas abordagens o critério da simplicidade de uma hipótese. Vários conceitos de simplicidade foram propostos, destacando-se o que associa a simplicidade a minimalidade de uma hipótese. Assim, por exemplo, é preferível a hipótese “choveu” do que a hipótese “choveu e os irrigadores foram ligados”.
- ***Força explanatória***. A ideia de força explanatória de uma hipótese também está presente de uma ou de outra maneira em muitas abordagens. Uma hipótese pode não explicar a totalidade do conjunto de fatos. Tem maior força explanatória e deve ser preferida a hipótese que explica a maior quantidade de fatos. Por exemplo, entre as várias doenças que explicam parcialmente um conjunto de sintomas, é preferível estabelecer como diagnóstico aquela que explica a maior quantidade de sintomas.

1.4 – Lógica Proposicional como formalismo para representação de problemas de abdução

Raciocínios abduativos podem ser formalizados de muitas maneiras, incluindo redes bayesianas (e.g., Fortier, Sheppard & Strasser, 2015), algoritmos genéticos (e.g., Romdhane & Ayeb, 2011), redes neurais (e.g., Thagard, 1989) e algum sistema lógico.

Este trabalho emprega a Lógica Proposicional (Howard, 1999), LP, como linguagem para expressão de raciocínios abduativos. Os conceitos de proposição, átomo, literal, conectivos lógicos ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$), semântica de conectivos lógicos, interpretação, tautologia, contradição, consequência lógica ($\models$), equivalência lógica ($\equiv$), cláusula, cláusula Horn, Forma Normal Conjuntiva (FNC), Forma Normal Disjuntiva (FND), resolução e prova de teoremas são definidos de maneira usual e não são rediscutidos neste texto. Este trabalho adota apenas uma extensão, nem sempre usual em textos de

Lógica de Predicados, que é a de permitir que um átomo seja denotado por mais do que uma letra (e.g., *abc*, *p2*, *chuva*, *bateria*).

Problemas de abdução podem ser formalizados por meio de sentenças da linguagem da LP³. Os exemplos a seguir mostram como isto pode ser feito.

Exemplo 1.1. O que é apresentado aqui é uma adaptação de um exemplo largamente utilizado dos estudos sobre raciocínio abduutivo. Considere:

$$\begin{aligned} T &= \{ \textit{chuva} \rightarrow \textit{gramadoMolhado}, \\ &\quad \textit{irrigadoresLigados} \rightarrow \textit{gramadoMolhado} \}, \\ F &= \{ \textit{gramadoMolhado} \}. \end{aligned}$$

A teoria T possui duas sentenças que objetivam capturar por meio das proposições *chuva*, *irrigadoresLigados*⁴ e *gramadoMolhado* as relações lógicas de (1) se houver chuva então o gramado fica molhado, e (2) se os irrigadores forem ligados então o gramado fica molhado. O conjunto de fatos F possui apenas um fato, definido pela proposição *gramadoMolhado*. Um raciocínio abduutivo nestas condições pode conduzir ao conjunto de hipóteses:

$$H = \{ \textit{chuva}, \textit{irrigadoresLigados}, \textit{chuva} \wedge \textit{irrigadoresLigados} \}.$$

■

Exemplo 1.2. Este exemplo é uma modificação do Exemplo 1.1 para incluir na teoria T uma sentença que descreve que se um tanque de água estiver vazio⁵ então os irrigadores não funcionam:

$$\begin{aligned} T &= \{ \textit{chuva} \rightarrow \textit{gramadoMolhado}, \\ &\quad \textit{irrigadoresLigados} \rightarrow \textit{gramadoMolhado}, \\ &\quad \textit{tanqueVazio} \rightarrow \neg \textit{irrigadoresLigados} \}, \end{aligned}$$

³ Por questões relacionadas a eficiência de algoritmos para a abdução, é proposto no Capítulo 3 uma restrição da linguagem da LP. No entanto, a restrição que é proposta não invalida o que está discutido nesta seção.

⁴ A semântica de uma proposição é, apenas, verdade ou falso. Este trabalho nomeia proposições por meio de “quase frases” (e.g., *irrigadoresLigados*) sugerindo a falsa impressão de que há uma outra semântica em uma proposição, mas não há. Isto é feito porque interessa a este trabalho a programação de modelos que representam o conhecimento sobre algo e, neste caso, é conveniente a nomeação de proposições como “quase frases”.

⁵ A linguagem da LP é atemporal, i.e., suas sentenças não descrevem relação com o tempo. Assim, não é possível pensar, por exemplo, que no passado poderia ter ocorrido do tanque não estar vazio.

$$F = \{ \textit{gramadoMolhado}, \textit{tanqueVazio} \}.$$

O conjunto F inclui uma nova proposição ao que já estava descrito no Exemplo 1.1: *tanqueVazio*. Nestas condições, um raciocínio abdutivo conduz ao conjunto de hipóteses:

$$H = \{ \textit{chuva} \}.$$

■

Exemplo 1.3. Uma interpretação de uma pequena parte do Código Penal Brasileiro (CPB) (Brasil, 2015) é apresentada neste exemplo. As sentenças da teoria T tem por objetivo modelar o seguinte. Ocorre um homicídio quando há um réu, uma vítima e acontece do réu ter matado a vítima. Um homicídio doloso é um tipo de homicídio em que houve por parte do réu a intenção de matar a vítima. Um homicídio culposo é um tipo de homicídio em que o réu agiu de forma imprudente⁶. Um homicídio qualificado é um homicídio praticado mediante pagamento do réu ou foi cometido por motivo torpe. Um homicídio privilegiado é um homicídio motivado por relevante valor moral ou foi precedido por injusta provocação da vítima ao réu. Homicídios podem ser classificados em mais de uma categoria (e.g., homicídio doloso e qualificado). As duas últimas sentenças da teoria T descrevem situações de impossibilidade de ocorrência simultânea: um homicídio doloso não pode ser ao mesmo tempo culposo e um homicídio culposo não pode ser ao mesmo tempo qualificado. O CPB prevê outros casos de homicídio doloso, culposo, qualificado e privilegiado, e outras relações entre estes conceitos mas, para manter o exemplo pequeno, apenas alguns casos estão sendo modelados pela teoria T .

$$\begin{aligned} T = \{ & \textit{homicídio} \rightarrow \textit{réu} \wedge \textit{vítima} \wedge \textit{matouRéuVítima}, \\ & \textit{homicídioDoloso} \rightarrow \textit{homicídio} \wedge \textit{intençãoMatar}, \\ & \textit{homicídioCulposo} \rightarrow \textit{homicídio} \wedge \textit{réuImprudente}, \\ & \textit{homicídioQualificado} \rightarrow \textit{homicídio} \wedge \textit{mediantePagamento}, \\ & \textit{homicídioQualificado} \rightarrow \textit{homicídio} \wedge \textit{porMotivoTorpe}, \end{aligned}$$

⁶ Esta e outras definições são, apenas, uma interpretação de parte do Código Penal Brasileiro. O Código prevê muito mais situações para classificação de um homicídio como culposo ou doloso e, além disso, o que aqui é descrito é uma interpretação, entre muitas possíveis.

$$\begin{aligned}
& homicídioPrivilegiado \rightarrow homicídio \wedge motivadoPorRelevanteValorMoral, \\
& homicídioPrivilegiado \rightarrow homicídio \wedge \\
& \quad precedidoPorInjustaProvacaoDaVítimaAoRéu, \\
& homicídioDoloso \rightarrow \neg homicídioCulposo, \\
& homicídioCulposo \rightarrow \neg homicídioQualificado \\
& \},
\end{aligned}$$

$F = \{ réu, vítima, matouRéuVítima, intençãoMatar, porMotivoTorpe \}.$

Um raciocínio abdutivo com os fatos descritos pelo conjunto F e a teoria T conduz ao conjunto unitário de hipóteses

$H = \{ homicidioDoloso \wedge homicídioQualificado \}.$

■

Uma vantagem da utilização da linguagem da LP para modelar problemas de abdução é poder utilizar todo o conhecimento existente sobre cálculo proposicional para formular algoritmos para abdução. De particular importância para este trabalho é o algoritmo para abdução chamado Peirce, proposto em Rodrigues, Oliveira e Oliveira (2014) e que opera com LP. Uma implementação do algoritmo Peirce, executando na Web, é a aplicação Peirce Online, disponível em pelo link “<http://peirceonline.gear.host/>”. O leitor pode utilizar a aplicação Peirce Online para executar os modelos estabelecidos nos exemplos desta seção⁷.

1.5 – Sobre este trabalho

A abdução fundamentada em lógica tem se revelado uma poderosa técnica para resolver uma variedade de problemas em IA incluindo, entre outros, problemas de diagnóstico, entendimento de linguagem natural, planejamento e formulação de argumentos (Denecker & Kakas, 2000).

⁷ A aplicação Peirce Online utiliza um subconjunto da linguagem da LP, mas os exemplos desta seção se enquadram neste subconjunto de linguagem e podem ser normalmente executados. Adicionalmente ao que foi exemplificado nesta seção, Peirce Online possui também parâmetros para controlar quais hipóteses são consideradas boas e, portanto, devem participar ou não da resposta a um raciocínio.

Se por um lado é inquestionável a utilidade da abdução na modelagem da solução de problemas, por outro lado a programação destas soluções é difícil. O problema fundamental com muitas das abordagens existentes (e.g., *Constraint Handling Rules Language* (CHR Bibliography, 2015)) é que elas estão formuladas sobre sistemas de programação lógica que servem a diferentes propósitos e não só para programar raciocínio abdutivo. Por outro lado, quando foram propostas especificamente para abdução (e.g., *Abductive Logic Programming Framework* (Kakas, Kowalsky & Toni, 1992)) elas não possuem em sua linguagem estruturas específicas para facilitar a modelagem de problemas de raciocínio abdutivo. Em suma, as soluções existentes demandam do programador conhecimentos profundos de lógica, linguagens de programação lógica e de mecanismos de resolução de sistemas lógicos.

Este trabalho de dissertação propõe o desenvolvimento de uma linguagem para programação de raciocínios abduativos, fundamentada em Lógica Proposicional, chamada de Abd1. Esta linguagem é concebida para tentar diminuir a demanda do programador por conhecimentos de lógica, de mecanismos de resolução de sistemas lógicos e de linguagens de programação de paradigma lógico, conhecimento estes necessários para se programar raciocínios abduativos utilizando as abordagens existentes.

1.6 – Organização deste texto

Além deste capítulo introdutório, esta dissertação de mestrado está organizada da seguinte maneira.

O Capítulo 2 descreve o contexto deste estudo, descrevendo as principais soluções para programação de raciocínio abdutivo existentes. O capítulo analisa as dificuldades de programação apresentadas pelas soluções existentes e declara o objetivo e metodologia empregada neste trabalho.

Tendo por objetivo o desenvolvimento de uma linguagem específica para programação de raciocínio abdutivo, Abd1, o Capítulo 3 apresenta o projeto desta linguagem. A Linguagem Abd1 é concebida tendo em vista a tentativa de diminuir as dificuldades de programação de raciocínios abduativos identificadas nas soluções existentes.

O Capítulo 4 descreve a implementação da Linguagem Abd1, e também a implementação de um ambiente integrado para desenvolvimento e execução de programas escritos em Abd1.

O Capítulo 5 relata uma pesquisa experimental em laboratório, conduzida com o objetivo de avaliar a usabilidade da Linguagem Abd1. Dois aspectos de usabilidade são avaliados: (1) a facilidade para aprendizagem e para o desenvolvimento de programas escritos em Linguagem Abd1 e (2) a satisfação subjetiva do programador em relação à experiência de usar a Linguagem Abd1 para desenvolver programas para realizar raciocínio abdutivo.

Finalmente, o Capítulo 6 discute as limitações, as potencialidades deste trabalho e apresenta conclusões.

Quatro apêndices suplementam esta dissertação. O Apêndice I descreve completamente o projeto gramatical da linguagem Abd1. Os Apêndices II e III contêm integralmente dois artigos publicados com resultados parciais deste trabalho. O Apêndice IV contém uma cópia do formulário usado na pesquisa experimental de usabilidade da Linguagem Abd1.

Capítulo 2

Contexto de Estudo, Objetivo e Metodologia

Este Capítulo descreve o contexto deste estudo, os objetivos e a metodologia propostos neste trabalho. Para uma visão global do contexto de estudo, a Seção 2.1 apresenta abordagens existentes para a programação de raciocínios abduativos descrevendo-as de maneira sintética. As abordagens existentes possuem muitos elementos em comum, assim uma delas, especificamente a Linguagem CHR (Constraint Handling Rules), é descrita em detalhes na Seção 2.2. A Seção 2.3 descreve dificuldades de programação presentes nas abordagens existentes. A Seção 2.4 declara o objetivo deste trabalho que é a proposta de uma linguagem de programação específica para raciocínios abduativos como tentativa de minimizar as dificuldades identificadas na Seção 2.3. Por fim, a Seção 2.5 apresenta a metodologia empregada na realização do trabalho.

2.1 – Visão geral das abordagens lógicas para abdução

O interesse pelo emprego da abdução em programação e especialmente em programação lógica teve início no final dos anos 1980 e início dos anos 1990 como uma tentativa de tratar de uma maneira uniforme uma variedade de problemas em Inteligência Artificial. Esta seção organiza a evolução das soluções empregadas desde então tendo como foco as estruturas linguísticas de alguns sistemas tomados como representativos da evolução. Uma descrição detalhada dos primeiros anos de desenvolvimento do campo é apresentada em Denecker e Kakas (2002) e uma descrição mais sucinta, porém mais atual, é dada em Caroprese & Trubitsyna (2014).

Linguagens de programação de propósito geral

Em princípio, qualquer linguagem de programação, de paradigma procedimental ou orientado a objetos (e.g., C, Java), pode ser utilizada para a programação de raciocínios abdutivos baseados em lógica. Entretanto, o esforço para programação de raciocínios abdutivos nestas linguagens é maior do que em linguagens do paradigma lógico, porque o programador tem que escrever código próprio para algoritmos que implementam a semântica e o cálculo de sistemas lógicos (e.g, algoritmo de resolução, algoritmo de unificação, algoritmo para converter sentenças para forma normal conjuntiva).

No campo dos sistemas de paradigma lógico, Prolog (Bramer, 2013) foi uma das primeiras soluções empregadas para programação de raciocínios abdutivos. Entretanto, mesmo Prolog não possui facilidades para programação destes raciocínios. Por exemplo, considere uma simples teoria $T = \{ chuva \rightarrow gramadoMolhado \}$, o fato $F = \{ gramadoMolhado \}$ e o seguinte programa Prolog para modelar a teoria T , por meio da regra:

gramadoMolhado :- chuva.

Em Prolog, executar um raciocínio abductivo sobre a teoria T , tendo como fato $F = \{ gramadoMolhado \}$ significa resolver a questão *gramadoMolhado*. Um sistema que executa Prolog dispara uma excessão porque o sistema falha ao resolver a questão *gramadoMolhado*, uma vez que não existe no programa um fato (sentido de fato em

programação lógica) *chuva*. Isto acontece porque o que deve ser abduzido é algo que falta ao programa e o que falta, neste simples raciocínio, é a hipótese *chuva*, pois

$$(chuva \rightarrow gramadoMolhado) \wedge chuva \models gramadoMolhado.$$

Por outro lado, se o programa Prolog fosse

chuva.

gramadoMolhado :- chuva.

o sistema responderia *true*, como indicação de que a questão *gramadoMolhado* foi satisfeita. Nenhum dos comportamentos discutidos é o que se espera como resposta do raciocínio abdutivo, i.e., a hipótese *chuva*.

Assim, não ajuda muito, em problemas de raciocínio abdutivo, apenas tirar proveito do mecanismo de resolução de teoremas existente em sistemas que implementam Prolog. No caso do raciocínio acima a solução poderia ser tão simples quanto incluir no predicado *chuva* um comando que tenha como efeito colateral a exibição de '*chuva*' como resposta ao raciocínio, assim:

chuva :- writeln('chuva').

gramadoMolhado :- chuva.

Neste caso simples, pode-se dizer que o sistema dá como resposta a hipótese '*chuva*' ao ser executada a questão *gramadoMolhado*, e este comportamento poderia ser considerado satisfatório para este simples problema de raciocínio abdutivo. No entanto, problemas de raciocínio abdutivo interessantes de serem resolvidos computacionalmente possuem muitas sentenças, uma relação lógica intrincada entre predicados/proposições e soluções para estes casos são complicadas de serem programadas, além de serem específicas para cada novo problema de abdução.

Abordagens que estendem a programação lógica tradicional

Abductive Logic Programming (ALP), segundo seus idealizadores Kakas, Kowalski, & Toni (1992), é um *framework* que estende programação lógica tradicional (e.g., Prolog). Neste caso, o termo *framework* refere-se à definição de um sistema $\langle P, A, IC \rangle$ onde:

- P: é um programa lógico, ou seja, contém regras e fatos no sentido usual empregado por estes termos em programação lógica;
- A: é um conjunto de predicados especiais, denominados de predicados abdutíveis.
- IC: é um conjunto de regras de restrição no seguinte formato

$$\textit{false} \text{ :- } R_1, R_2, R_3, \dots, R_n,$$

onde R_i ($1 \leq i \leq n$) são predicados.

A ideia da extensão da programação lógica tradicional com os predicados abdutíveis prende-se ao fato de que predicados que compõem uma hipótese abdutiva frequentemente não estão presentes no programa lógico P, conforme discutido na subseção anterior.

O conjunto de regras de restrição IC adicionam uma outra facilidade comumente necessária na modelagem de problemas de abdução: a definição de inconsistências.

Exemplo 2.1. Considere novamente o cenário definido no Exemplo 1.2 cujo conjunto teoria e conjunto de fatos está reproduzido a seguir:

$$\begin{aligned} T = \{ & \textit{chuva} \rightarrow \textit{gramadoMolhado}, \\ & \textit{irrigadoresLigados} \rightarrow \textit{gramadoMolhado}, \\ & \textit{tanqueVazio} \rightarrow \neg \textit{irrigadoresLigados} \}, \\ F = \{ & \textit{gramadoMolhado}, \textit{tanqueVazio} \}. \end{aligned}$$

Um programa em ALP para modelar esta teoria é sistema $\langle P, A, IC \rangle$ onde:

```
% A: predicados abdutíveis.

abducible_predicate (chuva, irrigadoresLigados).

% P: programa lógico.

gramadoMolhado :- chuva.

gramadoMolhado :- irrigadoresLigados.

% IC: restrições de integridade.

false :- tanqueVazio, irrigadoresLigados.
```

O sistema tem duas potenciais hipóteses definidas como predicados abdutíveis. Ao ser executado para a questão

gramadoMolhado, tanqueVazio.

o sistema usa P e IC para concluir que a hipótese *irrigadoresLigados* é inconsistente com o fato *tanqueVazio* e que, portanto, *chuva* é única hipótese possível.

A carência de variedade regras para compor o conjunto IC por ALP limita muito a modelagem de problemas de abdução. Isto conduziu ao estabelecimento de diversas alternativas com o objetivo de oferecer soluções. *Abductive Constraint Logic Programming* (ACLP) é a mais notável das soluções neste sentido. ACLP enriquece a ALP com um conjunto maior de regras de restrição usando solucionadores de restrição CLP (Constraint Logic Programming) (Kakas, Michael & Mourlas, 2000).

2.2 – Visão detalhada de uma abordagem para abdução: a Linguagem CHR

As abordagens existentes para abdução compartilham muitos conceitos e técnicas. Por exemplo, é comum entre muitas abordagens a declaração de predicados/proposições abdutíveis, ser definida como extensão funcional ou linguística de sistemas que implementam Prolog, o uso de regras de derivação para implementar a lógica da abdução, semântica operacional declarada por meio da metáfora de banco de restrições, estado de um banco de restrições etc.. A Linguagem CHR (*Constraint Handling Rules*) é um bom exemplar destes conceitos e técnicas utilizadas para estabelecer um sistema para abdução e, por este motivo, dedicaremos toda esta seção ao assunto. O objetivo é explorar por meio da Linguagem CHR os detalhes comuns a muitas abordagens existentes. Exemplos serão descritos para ilustrar como a Linguagem CHR pode ser utilizada para abdução.

Uso da Linguagem CHR para abdução

Entre as diversas propostas para programação de raciocínio abdutivo destaca-se nos dias de hoje a Linguagem CHR. A Linguagem CHR foi apresentada pela primeira vez em Frühwirth (1998) e a ideia de usá-la para implementar abdução foi proposta pela primeira vez em Abdennadher & Christiansen (2000). Hoje ela está presente como

extensão linguística dos principais sistemas que implementam Prolog (e.g., SWI-Prolog (2015), SICStus Prolog (2015)). Diferentemente das outras abordagens, pesquisas envolvendo a Linguagem CHR são atualmente intensas. O repositório CHR Bibliography (2015) é uma boa medida desta intensidade. Este repositório reúne referências para artigos em periódicos, artigos em conferências, livros, capítulos de livros, lista de autores organizados por ano e categoria.

A Linguagem CHR permite a modelagem de problemas utilizando a Lógica de Predicados. Assim, nas discussões desta Seção nós faremos referência a conceitos da Lógica de Predicados para explicar a Linguagem CHR, mas criaremos exemplos com predicados de aridade⁸ zero, ou seja, é como estivéssemos trabalhando com Lógica Proposicional, a lógica empregada neste trabalho.

A implementação da Linguagem CHR presente como extensão linguística do sistema SWI-Prolog será utilizada nas discussões desta seção. Embora nesta condição seja possível escrever programas que misturam sentenças escritas tanto em Linguagem CHR quanto em Linguagem Prolog, nossos exemplos se restringirão à Linguagem CHR.

De forma geral, um programa CHR para abdução possui a seguinte estrutura: (1) a teoria é descrita por meio de regras; (2) as hipóteses ($h_1, h_2, \dots, h_n$) que possivelmente possam ser respostas ao raciocínio são declaradas como predicados de restrição (e.g., “:- chr_constraint $h_1, h_2, \dots, h_n$ ”), neste contexto também chamados de predicados abduíves; (3) os fatos ($f_1, f_2, \dots, f_m$) são inseridos como uma questão (e.g., “?- $f_1, f_2, \dots, f_m$ ”).

Sintaxe e semântica operacional de um programa CHR

Um programa CHR é composto por três tipos regras de derivação cujos formatos são:

- Regras de simplificação: $h_1, h_2, \dots, h_n \Leftarrow\Rightarrow \text{Guarda} \mid b_1, b_2, \dots, b_m$.
- Regras de propagação: $h_1, h_2, \dots, h_n \Rightarrow \text{Guarda} \mid b_1, b_2, \dots, b_m$.
- Regras de “Simpagação”: $h_1, h_2, \dots, h_n \setminus h_{n+1}, h_{n+2}, \dots, h_p \Rightarrow \text{Guarda} \mid b_1, b_2, \dots, b_m$.

Cada h_i e cada b_i é um predicado especialmente denominado predicado de restrição ou, simplesmente, restrição ($n \geq 1, m \geq 1$ e $p \geq 2$). Os predicados de restrição h_i e b_i formam

⁸Aridade refere-se à quantidade de argumentos de um termo composto (Pospecel, 2002). Por exemplo, ‘pai(X,Y)’ é um predicado de aridade igual a 2 e ‘bateriaSemCarga’ é um predicado de aridade igual a zero.

o que, de maneira correlata às regras de Prolog, denomina-se respectivamente, de cabeça e corpo da regra. A “,” (vírgula) é o operador de sequenciamento e corresponde a conjunção. No corpo de uma regra é possível usar o operador “;” (ponto e vírgula) no lugar do operador “,” (vírgula) e significa a disjunção entre predicados. O *Guarda* é um conjunto, possivelmente vazio, de predicados separados pelo operador “,” (vírgula). Um *Guarda* vazio é interpretado como verdade (true) e pode deixar de ser descrito juntamente com a barra vertical.

As regras de um programa CHR podem ser entendidas como regras de reescrita sobre estados, sendo um estado definido pelo conjunto de restrições presentes em um certo instante em um banco restrições, a partir daqui chamado apenas de banco. Inicialmente o banco é preenchido com a questão a ser avaliada. Uma questão é um conjunto de restrições separadas pelo operador “,” (vírgula) ou pelo operador “;” (ponto e vírgula). Regras são tentadas ser aplicadas para a restrição ativa. Uma só restrição é ativa em cada momento. Inicialmente a restrição ativa é a primeira restrição da questão, i.e., aquela mais à esquerda. Aplicadas as possíveis regras a uma restrição ativa, ela deixa de ser ativa. Restrições são tornadas ativas da esquerda para a direita na ordem em que elas são colocadas nas questões. Uma regra é aplicada se (1) a cabeça dela coincide com a cabeça de restrições presentes no banco, sendo uma das restrições da cabeça a restrição ativa e (2) o guarda da regra é satisfeito. Regras de simplificação trocam um conjunto de restrições no banco por outro conjunto de restrições equivalente. Regras de propagação adicionam novas restrições sem remover seus “antecedentes”. Regras de “simpagação” são uma mistura das outras duas, sendo que as restrições na cabeça que aparecem antes da barra invertida permanecem no banco e aquelas que estão após a barra invertida são removidas. A unificação de átomos ocorre como em Prolog. Uma descrição formal da semântica operacional de CHR é apresentada em Duck *et al.* (2004).

Exemplo 2.1. O programa CHR ilustrado na Figura 2.1-a estabelece regras para uma interpretação de parte do Código Penal Brasileiro (Brasil, 2015) e está escrito de acordo com a sintaxe da biblioteca CHR disponível no sistema SWI-Prolog. O Exemplo 1.3 do Capítulo 1 contém detalhes sobre esta interpretação do Código Penal Brasileiro. As linhas de 1 até 5 declaram os predicados de restrição (restrições). Neste caso todas as restrições têm aridade 0 (zero), ou seja, são proposições. As regras estão descritas entre as linhas 7 e 22, sendo que todas as regras possuem cabeça, corpo e guarda vazio, o que

significa que o guarda é sempre verdade (*true*). A linha 8 contém um exemplo de uma regra de propagação e descreve que, na ocorrência das restrições *reu*, *vitima* e *matouReuVitima* elas devem ser mantidas e uma nova restrição, *homicidio*, deve ser escrita no banco. As linhas de 11 até 18 contêm exemplos de regras de simplificação. Especificamente, a regra da linha 11 descreve que na ocorrência das restrições *homicidio* e *intencaoMatar* a primeira deve ser mantida e a segunda deve ser removida do banco. As demais regras de simplificação podem ser entendidas de forma semelhante à regra da linha 11. As linhas 21 e 22 contêm exemplos de regras de simplificação. Já a regra da linha 21 descreve que a ocorrência das restrições *homicidioDoloso* e *homicidioCulposo* deve ser substituída por *false*, ou seja, sinaliza que o banco chegou a um estado de contradição e não há solução para a questão apresentada. A regra de simplificação da linha 22 pode ser entendida de forma semelhante à regra da linha 21.

```

1 :- chr_constraint homicidio/0, reu/0, vitima/0, matouReuVitima/0, homicidioDoloso/0,
2   homicidioCulposo/0, intencaoMatar/0, reuImprudente/0,
3   homicidioQualificado/0, homicidioPrivilegiado/0, mediantePagamento/0,
4   porMotivoTorpe/0, motivadoPorRelevanteValorMoral/0,
5   precedidoPorInjustaProvocacaoDaVitimaAoReu/0.
6
7 % Regra de propagação.
8 reu , vitima , matouReuVitima ==> homicidio.
9
10 % Regras de simplificação.
11 homicidio \ intencaoMatar <=> homicidioDoloso.
12 homicidio \ reuImprudente <=> homicidioCulposo.
13 homicidio \ mediantePagamento <=> homicidioQualificado.
14 homicidio \ porMotivoTorpe <=> homicidioQualificado.
15 homicidio \ motivadoPorRelevanteValorMoral <=> homicidioPrivilegiado.
16 homicidio \ precedidoPorInjustaProvocacaoDaVitimaAoReu <=> homicidioPrivilegiado.
17 homicidioQualificado \ homicidioQualificado <=> true.
18 homicidioPrivilegiado \ homicidioPrivilegiado <=> true.
19
20 % Regras de simplificação.
21 homicidioDoloso , homicidioCulposo <=> false.
22 homicidioCulposo , homicidioQualificado <=> false.

```

(a)

```

1 ?- reu , vitima , matouReuVitima , intencaoMatar , porMotivoTorpe , mediantePagamento.
2 reu
3 vitima
4 matouReuVitima
5 homicidio
6 homicidioDoloso
7 homicidioQualificado
8 true ;
9 false.

```

(b)

```

1 ?- reu , vitima , matouReuVitima , intencaoMatar , reuImprudente.
2 false.

```

(c)

Figura 2.1. Exemplo de um programa escrito na Linguagem CHR.

Operacionalmente um programa CHR usa regras para, passo a passo, modificar um banco B, inicialmente em um estado B^1 preenchido com uma questão Q. Em cada passo uma regra é escolhida para ser aplicada. A regra a ser escolhida é a primeira da ordem em que são escritas (de cima para baixo, da esquerda para a direita) a casar suas cabeças com restrições em B, sendo que uma destas restrições é a restrição ativa. A aplicação de uma regra é chamada de derivação. Quando a quantidade de derivações é finita e termina com B em um estado B^n ($n \geq 1$), se B^n for uma contradição então diz-se que as derivações falharam e não há solução para o conjunto de restrições apresentado como questão, caso contrário, as derivações foram bem-sucedidas e B^n é o conjunto de restrições que respondem à questão Q.

Exemplo 2.2. A Figura 2.1-b mostra uma execução do programa da Figura 2.1-a para uma questão, digamos Q1, onde $Q1 = \text{reu}, \text{vitima}, \text{matouReuVitima}, \text{intencaoMatar}, \text{porMotivoTorpe}, \text{mediantePagamento}$ (linha 1, Figura 2.1-b). O sistema executa a questão Q1 e dá como resultado o conjunto de restrições indicados nas linhas de 2 até 7. Ou seja, para a questão apresentada, a resposta é o conjunto $\{ \text{reu}, \text{vitima}, \text{matouReuVitima}, \text{homicídio}, \text{homicidioDoloso}, \text{homicidioQualificado} \}$. Em termos do jargão de CHR estas são as restrições que respondem à questão. Em termos do jargão do domínio de raciocínios abduativos, esta é a hipótese que responde ao raciocínio abduativo cujos fatos foram expressos na questão apresentada para execução.

Operacionalmente, o sistema faz o seguinte: inicia um banco, digamos B, que em seu estado B^1 é igual à questão a ser avaliada,

$B^1 = \{ \text{reu}, \text{vitima}, \text{matouReuVitima}, \text{intencaoMatar}, \text{porMotivoTorpe}, \text{mediantePagamento} \}$.

A primeira restrição é eleita restrição ativa, ou seja, a restrição *reu* é tornada ativa, o que aqui está representado em negrito:

$B^1 = \{ \textbf{reu}, \text{vitima}, \text{matouReuVitima}, \text{intencaoMatar}, \text{porMotivoTorpe}, \text{mediantePagamento} \}$.

Regras contendo na cabeça a restrição ativa e demais restrições no banco B são procuradas para serem aplicadas. A regra de propagação

$\text{reu}, \text{vitima}, \text{matouReuVitima} ==> \text{homicidio}$

é selecionada e aplicada, o que faz manter em B as restrições *reu*, *vitima* e *matouReuVitima*, e incluir a restrição *homicídio*, conduzindo o banco ao estado B^2 :

$B^2 = \{ \text{reu}, \text{vitima}, \text{matouReuVitima}, \text{intencaoMatar}, \text{porMotivoTorpe}, \text{mediantePagamento}, \text{homicidio} \}.$

Novas regras são procuradas para serem aplicadas para a restrição ativa. Não encontrando mais regras a serem aplicadas à restrição *reu*, ela deixa de ser ativa e a restrição *vitima* passa a ser a restrição ativa:

$B^2 = \{ \text{reu}, \textbf{vitima}, \text{matouReuVitima}, \text{intencaoMatar}, \text{porMotivoTorpe}, \text{mediantePagamento}, \text{homicidio} \}.$

Não havendo novas regras que tenham na cabeça a restrição ativa *vitima* (a regra que possui *vitima* na cabeça já foi aplicada a esta restrição anteriormente) ela deixa de ser ativa e a regra *matouReuVitima* passa a ser a restrição ativa:

$B^2 = \{ \text{reu}, \text{vitima}, \textbf{matouReuVitima}, \text{intencaoMatar}, \text{porMotivoTorpe}, \text{mediantePagamento}, \text{homicidio} \}.$

Analogamente ao que aconteceu à restrição ativa *vitima*, não há regras para serem aplicadas à restrição ativa *matouReuVitima*. Assim, ela deixa de ser a restrição ativa e a restrição *intencaoMatar* passa a ser a restrição ativa:

$B^2 = \{ \text{reu}, \text{vitima}, \text{matouReuVitima}, \textbf{intencaoMatar}, \text{porMotivoTorpe}, \text{mediantePagamento}, \text{homicidio} \}.$

A regra de simpagação

$$\text{homicidio} \setminus \text{intencaoMatar} \leq \text{homicidioDoloso}$$

é escolhida para ser aplicada, o que mantém em B a restrição *homicidio*, exclui a restrição *intencaoMatar* e inclui a restrição *homicidioDoloso*, alterando o estado do banco para B^3 :

$B^3 = \{ \text{reu}, \text{vitima}, \text{matouReuVitima}, \text{porMotivoTorpe}, \text{mediantePagamento}, \text{homicidio}, \text{homicidioDoloso} \}.$

A restrição *porMotivoTorpe* é eleita ativa:

$B^3 = \{ \text{reu}, \text{vitima}, \text{matouReuVitima}, \textbf{porMotivoTorpe}, \text{mediantePagamento}, \text{homicidio}, \text{homicidioDoloso} \}.$

A regra de simpagação

$$\text{homicidio} \setminus \text{porMotivoTorpe} \leq \text{homicidioQualificado}$$

é escolhida para ser aplicada, o que mantém em B a restrição *homicidio*, exclui a restrição *porMotivoTorpe* e inclui a restrição *homicidioQualificado*, conduzindo o banco ao estado B⁴:

$B^4 = \{ \text{reu, vitima, matouReuVitima, mediantePagamento, homicidio, homicidioDoloso, homicidioQualificado} \}.$

A restrição *mediantePagamento* é tornada ativa:

$B^5 = \{ \text{reu, vitima, matouReuVitima, **mediantePagamento**, homicidio, homicidioDoloso, homicidioQualificado} \}.$

A regra de simpagação

$$\text{homicidio} \setminus \text{mediantePagamento} \leq \text{homicidioQualificado}$$

é escolhida para ser aplicada, o que mantém em B a restrição *homicidio*, exclui a restrição *mediantePagamento* e inclui uma segunda restrição *homicidioQualificado*, conduzindo o banco ao estado B⁶:

$B^6 = \{ \text{reu, vitima, matouReuVitima, homicidio, homicidioDoloso, homicidioQualificado, homicidioQualificado} \}.$

A restrição *homicidioDoloso* é tornada ativa:

$B^6 = \{ \text{reu, vitima, matouReuVitima, homicidio, **homicidioDoloso**, homicidioQualificado, homicidioQualificado} \}.$

Como não há regras para serem aplicadas a esta restrição ativa, ela deixa de ser ativa e a restrição *homicidioQualificado* torna-se ativa:

$B^6 = \{ \text{reu, vitima, matouReuVitima, homicidio, homicidioDoloso, **homicidioQualificado**, homicidioQualificado} \}.$

A regra de simpagação

$$\text{homicidioQualificado} \setminus \text{homicidioQualificado} \leq \text{true}$$

é escolhida para ser aplicada, o que mantém no banco apenas uma restrição *homicidioQualificado*, conduzindo o banco ao estado B⁷:

$B^7 = \{ \text{reu, vitima, matouReuVitima, homicidio, homicidioDoloso, homicidioQualificado} \}.$

Como não há novas restrições para se tornarem ativas então B⁷ é o estado final, ou seja, o conjunto de restrições que responde à questão apresentada.

■

Exemplo 2.3. A Figura 2.1-c mostra uma execução do programa da Figura 2.1-a para uma questão, digamos $Q2$, onde $Q2 = \text{reu} , \text{vitima} , \text{matouReuVitima} , \text{intencaoMatar} , \text{reuImprudente}$ (linha 1, Figura 2.1-c). O sistema executa a questão $Q2$ e dá como resultado *false*, indicando que não há restrições que respondem à questão $Q2$, ou seja, o que quer dizer a mesma coisa, o conjunto de restrições que respondem à questão $Q2$ é vazio. Em termos de raciocínio abductivo pode-se dizer que não há hipótese que responda aos fatos apresentados.

Operacionalmente, o início da execução deste exemplo é similar à execução discutida no Exemplo 2.2, porque as quatro primeiras restrições e a ordem delas na questão $Q2$ é a mesma da questão $Q1$ discutida no Exemplo 2.2. Assim, partindo do estado B^1 ,

$$B^1 = \{ \text{reu}, \text{vitima}, \text{matouReuVitima}, \text{intencaoMatar}, \text{reuImprudente} \},$$

chega-se de forma similar ao estado B^3 , onde

$$B^3 = \{ \text{reu}, \text{vitima}, \text{matouReuVitima}, \text{reuImprudente}, \text{homicidio}, \text{homicidioDoloso} \}.$$

A restrição *reuImprudente* torna-se ativa,

$$B^3 = \{ \text{reu}, \text{vitima}, \text{matouReuVitima}, \textbf{reuImprudente}, \text{homicidio}, \text{homicidioDoloso} \},$$

e a regra

$$\text{homicidio} \setminus \text{reuImprudente} \leq \Rightarrow \text{homicidioCulposo}$$

é executada, o que leva o banco ao estado B^4 ,

$$B^4 = \{ \text{reu}, \text{vitima}, \text{matouReuVitima}, \textbf{reuImprudente}, \text{homicidio}, \text{homicidioDoloso}, \text{homicidioCulposo} \}.$$

Não havendo mais regras a serem aplicadas à restrição ativa *reuImprudente*, ela se torna inativa e a restrição *homicidio* torna-se ativa,

$$B^4 = \{ \text{reu}, \text{vitima}, \text{matouReuVitima}, \text{reuImprudente}, \textbf{homicidio}, \text{homicidioDoloso}, \text{homicidioCulposo} \}.$$

Como antes, não há novas regras para serem aplicadas à restrição ativa *homicidio*, assim ela se torna inativa e a restrição *homicidioDoloso* torna-se ativa,

$$B^4 = \{ \text{reu}, \text{vitima}, \text{matouReuVitima}, \text{reuImprudente}, \text{homicidio}, \textbf{homicidioDoloso}, \text{homicidioCulposo} \}.$$

A regra

$$\text{homicidioDoloso} \setminus \text{homicidioCulposo} \leq \Rightarrow \text{false}$$

é seleccionada e aplicada, o que simplifica o banco para o estado de contradição (*false*), levando-o ao estado B^5 ,

$B^5 = \{ \}$.

Este é o estado final, ou seja, o conjunto de restrições que respondem à questão Q2 é vazio.

■

Exemplo 2.4. Este exemplo difere dos exemplos anteriores porque o operador “;” (operador “ou”) está sendo usado no corpo da regra da linha 3 do programa CHR ilustrado na Figura 2.2-a. O programa, constituído por duas regras, é bastante simples. A regra da linha 3 estabelece que *gramadoMolhado* propaga para *chuva* ou *irrigadoresLigados* e a regra da linha 4 estabelece que a presença simultânea de *tanqueVazio* e *irrigadoresLigados* simplifica para *false*, i.e., determina uma situação de contradição. A execução deste programa para a questão, digamos Q, $Q = \text{gramadoMolhado}, \text{tanqueVazio}$ é apresentada na Figura 2.2-b. O sistema executa a questão Q e dá como resultado o conjunto de restrições indicado nas linhas de 2 até 4, ou seja, para a questão apresentada, a resposta é o conjunto de restrições { *gramadoMolhado*, *tanqueVazio*, *chuva* }. Em termos de raciocínio abdutivo, esta é a hipótese que responde aos fatos apresentados na questão inserida para execução.

```
1 :- chr_constraint chuva/0, gramadoMolhado/0, irrigadoresLigados/0, tanqueVazio/0.
2
3 gramadoMolhado ==> chuva ; irrigadoresLigados.
4 tanqueVazio , irrigadoresLigados <=> false.
```

(a)

```
1 ?- gramadoMolhado , tanqueVazio.
2 gramadoMolhado
3 tanqueVazio
4 chuva
5 true ;
6 false.
```

(b)

Figura 2.2. um programa CHR que faz uso do operador “;” no corpo de uma regra.

Operacionalmente, o programa tem o comportamento ilustrado na Figura 2.3. Inicialmente, o banco é preenchido com a questão Q, resultando no estado B^1

apresentado na Figura. A restrição mais à esquerda de B^1 , *gramadoMolhado* é tornada ativa, e a regra *gramadoMolhado* $\Rightarrow$ *chuva* ; *irrigadoresLigados* é selecionada para ser aplicada. O resultado da aplicação desta regra é a criação de dois estados distintos, um deles propagando para *chuva* (estado B^2) e o outro propagando para *irrigadoresLigados* (estado B^3) a partir do estado B^1 . O estado B^2 e o estado B^3 seguem derivações independentes a partir disto. Seguindo as derivações do lado de B^2 , a restrição *tanqueVazio* é tornada ativa e, não tendo regra para ser aplicada, ela torna-se inativa. A restrição *chuva* é tornada ativa e, como também não existem regras para serem aplicadas, ela se torna inativa. Não tendo nenhuma outra nova restrição para se tornar ativa, as derivações do lado B^2 da figura têm fim. As derivações do lado B^3 da figura iniciam com a restrição *tanqueVazio* se tornando ativa. A regra *tanqueVazio* , *irrigadoresLigados* $\Leftrightarrow$ *false* é selecionada e aplicada. Esta regra descreve uma simplificação para uma contradição, o que determina que o banco mude do estado B^3 para o estado B^4 , vazio, sem restrições que respondam a questão *Q*. Assim, após o término das derivações do lado B^2 e do lado B^3 da figura, o conjunto de restrições { *gramadoMolhado*, *tanqueVazio*, *chuva* } responde de forma geral à questão *Q* formulada inicialmente.

■

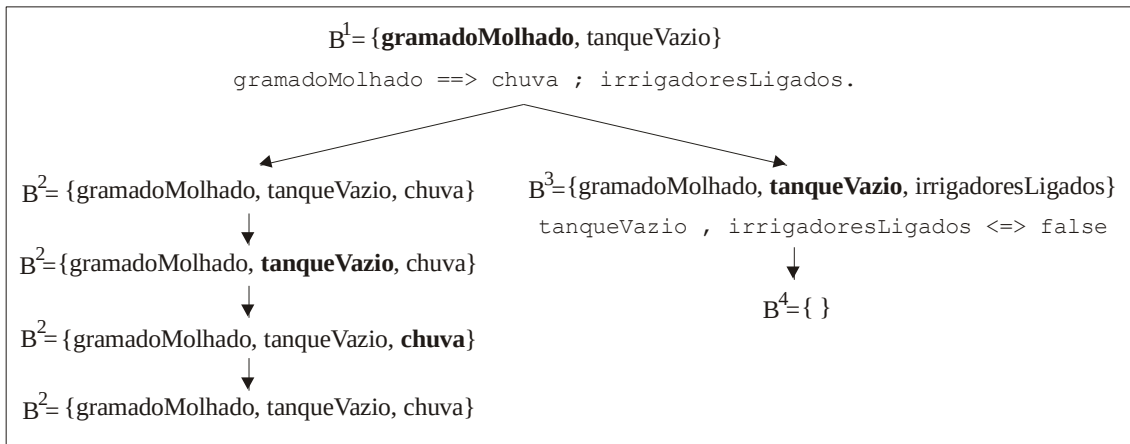


Figura 2.3. execução da questão *gramadoMolhado* , *tanqueVazio*: os diferentes estados do banco de restrições são apresentados por B^i ($i = 1, 2$ e 3); as restrições ativas estão destacadas em negrito .

Exemplo 2.5. A Figura 2.4 apresenta o diagrama elétrico de um circuito constituído por duas lâmpadas, dois interruptores, uma bateria e fios. Neste circuito a lâmpada 2 acende se a bateria possuir carga, os fios estiverem conectados e o interruptor 1 estiver ligado.

Para que a lâmpada 1 acenda é necessário, adicionalmente, que o interruptor 2 esteja ligado. Um programa CHR modelando possíveis falhas e funcionalidades que levam as lâmpadas do circuito estarem apagadas ou acesas é apresentado na Figura 2.5-a. Na linha 1 estão declarados os possíveis predicados que podem ocorrer como resposta de um raciocínio abdutivo. As linhas de 2 a 5 contêm regras para modelar falhas e funcionalidades do circuito. As linhas de 6 a 8 definem regras para manipular inconsistências como, por exemplo, não é possível ao mesmo tempo o interruptor 1 estar ligado e desligado (linha 6). As linhas de 9 a 12 definem regras para eliminar duplicatas (efeito colateral de outras regras). A Figura 2.5-b ilustra a execução do programa para o fato *lâmpada_1_apagada*. A série de pontos e vírgula que separam o resultado indica que o raciocínio realizado conduziu a três hipóteses que são *interruptor_1_desligado* ou *bateria_sem_carga* ou *interruptor_2_desligado*. A Figura 2.5-c ilustra a execução do programa para os fatos *lâmpada_1_apagada* , *lâmpada_2_acesa*. Neste caso, o resultado conduziu a uma hipótese que deve ser lida como *interruptor_2_desligado* e *bateria_com_carga* e *interruptor_1_ligado*.

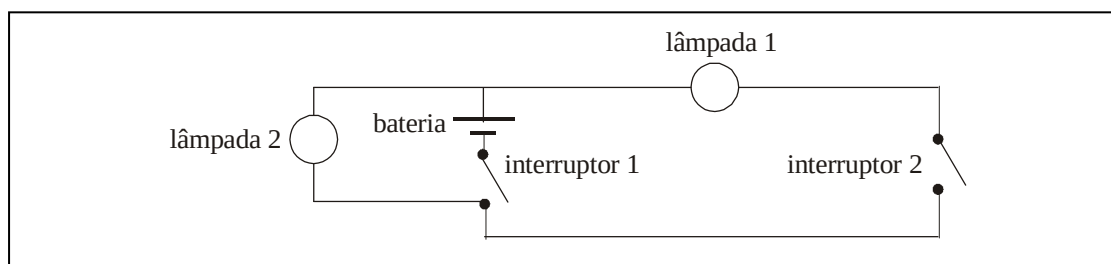


Figura 2.4. Diagrama de um circuito elétrico simples.

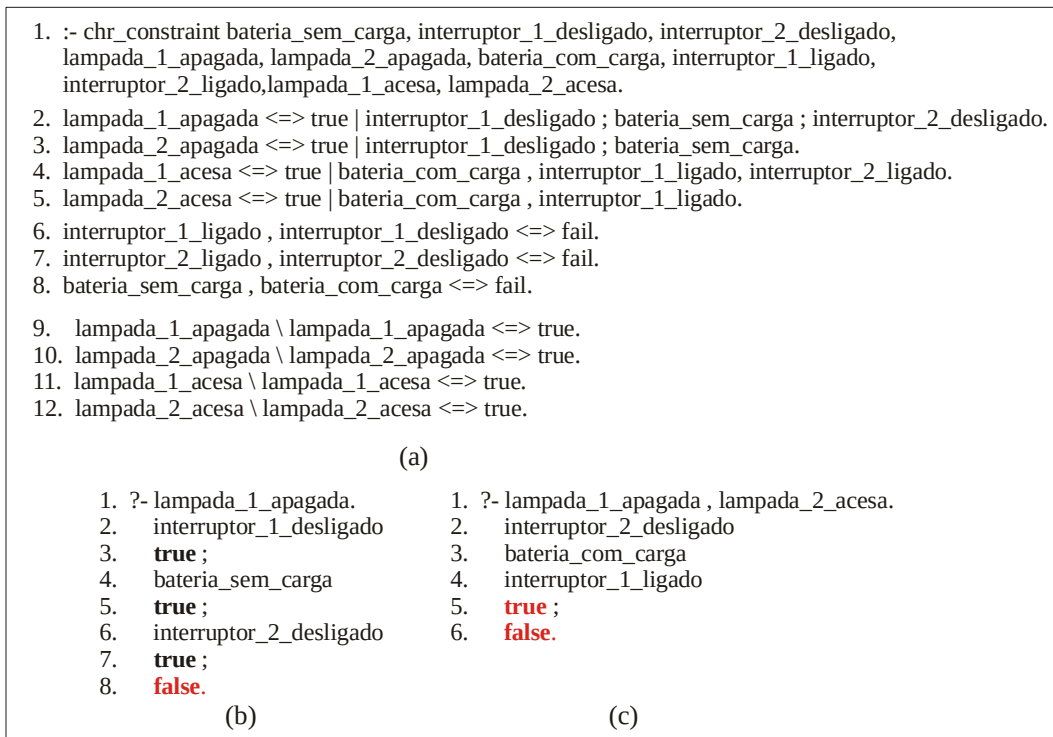


Figura 2.5. Programa CHR que modela o comportamento do circuito da Figura 2.4: (a) Texto do programa, (b) e (c) execuções do programa.

2.3 – Dificuldades de uso das abordagens lógicas para programação de raciocínio abdutivo

Pode-se destacar as seguintes vantagens gerais das abordagens para programação de raciocínio abdutivo existentes: (1) elas são abrangentes, no sentido de poderem ser aplicadas para programação não só de raciocínios abduativos; (2) muitas delas se valem da riqueza de um sistema hospedeiro bem implementado e estável, como sistemas que implementam Prolog, uma vez que estão inseridas como extensão funcional e linguística destes sistemas (e.g., Linguagem CHR); (3) estão fundamentadas em modelos teóricos bem construídos e estudados; e (4) executam de forma eficiente, uma vez que seus sistemas de execução foram aprimorados ao longo de muitos anos.

Entretanto, o uso destas abordagens impõe uma série de dificuldades identificadas neste trabalho por meio de investigação empírica. Esta seção relata dificuldades comuns à maioria das abordagens estudadas e, para exemplificar, a Linguagem CHR e os exemplos de programas CHR discutidos na Seção 2.2 são utilizados.

Dificuldade 1: *linguagem com grande diferença sintática com a linguagem natural.* Há uma grande diferença sintática entre sentenças descrevendo regras em língua natural e as respectivas sentenças expressas nos sistemas para abdução estudados. Por exemplo, é natural a expressão da sentença “se a bateria está sem carga ou o interruptor 1 está desligado então a lâmpada 2 está apagada”. No entanto, a estrutura sintática da correspondente sentença nos sistemas estudados é muito diferente. Por exemplo, a escrita deste mesmo conteúdo em Linguagem CHR é muito diferente (Figura 2.5-a, linha 3), o que impõe ao programador esforço cognitivo de tradução. De forma similar, a leitura e interpretação dos resultados nos sistemas para abdução existentes também tem seus desafios. Por exemplo, no sistema SWI Prolog, executando a Linguagem CHR, o resultado apresentado na Figura 2.5-b é “interruptor 1 desligado ou bateria sem carga ou interruptor 2 desligado” e o resultado da Figura 2.5-c é “interruptor 2 desligado e bateria com carga e interruptor 1 ligado”. No entanto, o que é apresentado nestas figuras é sintaticamente muito diferente disto.

Dificuldade 2: *não uso de terminologia do domínio dos raciocínios abdutivos.* Com algumas exceções como ALP e ACLP, de forma geral os sistemas estudados prestam-se a muitos propósitos sendo que podem também ser utilizados para programação de raciocínios abdutivos. Não tendo sido propostos originalmente para a realização de raciocínios abdutivos, mas para resolver diferentes tipos de problemas: (1) os sistemas estudados não incorporam, diretamente nas suas terminologias, estruturas linguísticas do domínio dos raciocínios abdutivos (e.g., teoria, hipóteses, fatos etc.) e (2) obrigam determinar por antecipação os predicados abdutíveis e a descrevê-los em alguma seção especial do programa. Por exemplo, na Linguagem CHR os predicados abdutíveis devem ser descritos na seção `chr_constraint` (e.g., Figura 2.5-a, linha 1).

Dificuldade 3: *mistura de paradigmas de programação.* De forma geral, os sistemas estudados, assim como Prolog, são declarativos, mas para programar nestes sistemas é preciso lidar com outros paradigmas de programação não declarativos. Para programar em Prolog, e isto também vale para vários sistemas que se prestam à abdução, a exemplo de CHR, há a necessidade do programador conhecer profundamente a semântica operacional destas linguagens (e.g., *backtracking*, negação como falha, derivação de regra CHR, *looping*). Muitas vezes regras devem ser incluídas para “contornar” efeitos não desejáveis, apesar de nada a terem a ver com o que fundamentalmente quer-se

modelar. O Exemplo 2.4 ilustra isto, regras para lidar com eliminação de duplicatas tiveram que ser incorporadas ao programa da Figura 2.5-a (linhas 9 até 12), apesar de fundamentalmente querer-se declarar apenas regras relacionadas ao funcionamento de um circuito.

Dificuldade 4: *ausência de suporte específico para expressar raciocínios complexos a partir de raciocínios mais simples.* Os sistemas estudados não oferecem facilidades especiais para o desenvolvimento de raciocínios mais complexos a partir da associação dos resultados de dois ou mais raciocínios, seja pela união, intersecção ou diferença que existe entre eles. Um raciocínio tem como resposta um conjunto de hipóteses. É natural, por exemplo, pensar na intersecção de raciocínios (hipóteses) para compor raciocínios mais elaborados. Por exemplo, considerando o Exemplo 2.4, as hipóteses que explicam simultaneamente as lâmpadas 1 e 2 apagadas podem ser obtidas a partir da intersecção dos resultados de dois raciocínios independentes que explicam, um deles, por que a lâmpada 1 está apagada e, o outro, porque a lâmpada 2 está apagada.

Dificuldade 5: *segmentação de programas grandes alheia aos conceitos do domínio dos raciocínios abduativos.* Programas constituídos por muitas regras em um único texto são confusos. De forma geral, os sistemas estudados oferecem facilidades para segmentação de programas grandes em vários arquivos ou módulos. Por exemplo, um programa CHR em SWI-Prolog pode ser dividido em vários módulos, cada um deles contendo um certo conjunto de regras. Embora seja funcional, esta forma de segmentação não emprega os conceitos do domínio dos raciocínios abduativos (teorias, hipóteses e fatos), o que traz dificuldades para administrar grandes programas. Ao contrário, a possível segmentação, por exemplo, por diferentes teorias, analogamente ao que acontece com a segmentação por classes em sistemas orientados a objetos, traz facilidades de escrita e manutenção de programas grandes.

Dificuldade 6: *fraco suporte a necessidades expressivas recorrentes na modelagem de teorias para raciocínio abduativo.* Quando se está modelando teorias para serem empregadas em raciocínio abduativo é muito comum encontrar-se situações em que há a necessidade de descrever sentenças que criam uma “circularidade lógica” entre proposições. Por exemplo, modelar a equivalência lógica entre as proposições *lindo* e *bonito*, digamos por meio da seguinte sentença,

lindo $\leftrightarrow$ *bonito*.

Entretanto, isto não tem uma solução simples, por exemplo, em linguagem CHR ou nos *frameworks* ALP e ACLP. Sem escrever uma certa quantidade de código adicional, por exemplo em Prolog, não é fácil evitar que simples regras escritas nestes sistemas provoquem uma execução infinita de um raciocínio. Outra necessidade recorrente na modelagem de raciocínios abduativos é a escrita de sentenças que empregam negação, como em

bateria_sem_carga $\rightarrow \neg$ *bateria_com_carga*,

sendo que os sistemas existentes não oferecem ou oferecem fraco suporte linguístico para a descrição direta de sentenças que empregam negação.

Resumindo, é possível supor que as dificuldades apresentadas pelas soluções existentes, mesmo por aquelas que se declaram motivadas a estender a programação lógica tradicional para realizar raciocínios abduativos, estão ligadas ao fato de que não foram concebidas para realmente representar o domínio dos raciocínios abduativos, seus conceitos, termos, relações e necessidades de modelagem. Muitas das abordagens existentes são extensões de sistemas de programação lógica existentes (e.g., Prolog) e, por isto, guardam uma relação forte com a tradição conceitual e linguística destes sistemas. Em contrapartida, para representar realmente o domínio dos raciocínios abduativos é preciso lidar diretamente com conceitos como os de teoria, fatos e hipóteses e com a relação lógica entre eles. É preciso também, que haja mecanismo simples para expressar algum conjunto de critérios para segregar hipóteses que se considerem boas. Além disso, por se tratar da representação de um domínio de raciocínios, é necessário que haja mecanismo descomplicado para expressar raciocínios complexos como composição de raciocínios elementares.

2.4 – Objetivo

Hoje a programação de raciocínios abduativos é difícil e está repleta de desafios para o programador. Um conjunto de dificuldades do uso das abordagens existentes para programação de raciocínios abduativos está descrito na Seção 2.3.

Duas são as principais matrizes das dificuldades identificadas: (1) muitos das abordagens estudadas (e.g., Linguagem CHR) servem a vários propósitos e não foram projetados especialmente para a programação de raciocínios abduativos; (2) sistemas produzidos com o objetivo primário para realizar raciocínio abduativo (e.g., ALP e ACPL) foram projetados como extensão de programação lógica tradicional e, assim, não incorporam em sua linguagem estruturas específicas para facilitar a modelagem de problemas de raciocínio abduativo.

Este trabalho é motivado pela tentativa de facilitar a escrita de programas para realizar raciocínios abduativos pela tentativa de minimizar as dificuldades de uso das abordagens existentes (Seção 2.3), dificuldades estas assim sintetizadas: (1) possuem linguagem com grande diferença sintática da linguagem natural, (2) não usam terminologia do domínio dos raciocínios abduativos, (3) misturam paradigmas de programação, (4) não possuem suporte específico para expressar raciocínios complexos a partir de raciocínios mais simples, (5) promovem a segmentação de programas grandes de forma alheia aos conceitos do domínio dos raciocínios abduativos e (6) oferecem fraco suporte a necessidades expressivas recorrentes na modelagem de teorias para raciocínio abduativo.

Tendo em vista a motivação descrita, este trabalho objetiva o desenvolvimento de uma linguagem de programação específica para a modelagem de problemas de raciocínio abduativo que minimize as dificuldades identificadas e descritas em detalhes na Seção 2.3.

2.5 – Metodologia

Esta seção dá uma visão geral da metodologia empregada neste trabalho. Em outros capítulos contêm o detalhamento metodológico do que está discutido nesta seção: o Capítulo 3 detalha a fundamentação lógica do *design* da Linguagem Abd1, o Capítulo 4 detalha o modelo do autômato que executa programas Abd1 e o Capítulo 5 detalha o método empregado na pesquisa experimental de usabilidade da Linguagem Abd1.

O elemento metodológico central deste trabalho é a pesquisa empírica conduzida com o objetivo de levantar as principais dificuldades de programação de raciocínios abduativos apresentadas pelas abordagens existentes. Os resultados desta pesquisa empírica foram discutidos nas Seções 2.1, 2.2 e 2.3. Estes resultados são obtidos principalmente a partir

de vasta experimentação envolvendo o desenvolvimento de programas para raciocínio abduutivo nas soluções estudadas e a vivência dos pesquisadores nos campos da Programação, das Linguagens de Programação, dos Fatores Humanos e da Cognição. O referencial bibliográfico levantado é utilizado, de forma geral, para identificação das soluções mais significantes em cada categoria estudada.

O desenvolvimento do protótipo de linguagem para raciocínio abduutivo, Abd1, segue métodos teóricos, técnicas e artefatos tecnológicos conhecidos e retratados pela literatura de Ciência da Computação, como será descrito, para a consecução das seguintes tarefas:

- Tarefa 1: Estabelecimento de fundamentação lógica para o *design* da linguagem Abd1.
- Tarefa 2: Elaboração do projeto formal da Abd1.
- Tarefa 3: Elaboração do modelo para o autômato que executa programas escritos em Abd1.
- Tarefa 4: Desenvolvimento do compilador da Abd1.
- Tarefa 5: Desenvolvimento do autômato da Abd1.
- Tarefa 6: Desenvolvimento do Ambiente Integrado de Desenvolvimento e Execução de Programas escritos na Linguagem Abd1 (AIDEP-Abd1).

Uma avaliação da Linguagem Abd1, define a:

- Tarefa 7: Pesquisa experimental em laboratório para avaliar a usabilidade da Linguagem Abd1.

Tarefa 1: *Estabelecimento de fundamentação lógica para o design da linguagem Abd1.*

A fundamentação lógica para o *design* da linguagem Abd1 segue as ideias de Rodrigues, Oliveira e Oliveira (2014) para uma estrutura de raciocínio proposta para programação de raciocínios abduativos.

Tarefa 2: *Elaboração do projeto formal da Abd1.* O projeto formal da Abd1 inicia com o esboço de protótipos de programas escritos em papel. O método é o da iteração progressiva dos protótipos de papel a partir das sessões de avaliação para fins de refinamento por meio da interação entre os pesquisadores. Os critérios de avaliação incluem (1) o atendimento aos fundamentos lógicos de *design* adotados e (2) o potencial de contribuição das versões de protótipo de papel para a superação das dificuldades de

programação. Estabelecido o protótipo de programa em papel, o projeto formal da Abd1 é materializado com o uso de Gramática Livre de Contexto por meio de um formalismo adaptado de textos clássicos de Linguagens Formais e Autômatos (e.g., Hopcroft, Motwani & Ullman, 2006).

Tarefa 3: *Elaboração do modelo para o autômato que executa programas escritos em Abd1.* O autômato da Abd1 é do tipo Autômato com Pilha (Hopcroft, Motwani & Ullman, 2006). O conjunto de instruções do autômato é definido para permitir avaliação de expressões envolvendo a intersecção, a união e a diferença entre raciocínios definidos de diferentes maneiras. O modelo de execução para o autômato é descrito de forma gráfica, representando diferentes configurações da pilha, e pseudocódigo do conjunto de instruções⁹.

Tarefa 4: *Desenvolvimento do compilador da Abd1.* O compilador da Abd1 é desenvolvido utilizando-se a técnica da análise sintática descendente recursiva (Aho, Lam, Sethi, & Ullman, 2006). A plataforma de desenvolvimento é o Microsoft .NET (Microsoft, 2015), empregando linguagem C#. A escolha recai sobre esta plataforma por questão de compatibilidade com uma existente implementação do algoritmo Peirce (Rodrigues, 2015) utilizado na implementação do autômato da Abd1.

Tarefa 5: *Desenvolvimento do autômato Abd1.* O autômato da Abd1 segue o modelo estabelecido na Tarefa 4 e é implementado usando a plataforma de desenvolvimento Microsoft .NET. Este autômato faz uso de uma implementação existente do algoritmo Peirce (Rodrigues, 2015) para a plataforma Microsoft .NET, empregando linguagem C#.

Tarefa 6: *Desenvolvimento do Ambiente Integrado de Desenvolvimento e Execução de Programas escritos na Linguagem Abd1 (AIDEP-Abd1).* Protótipos de interfaces da AIDEP-Abd1 é desenvolvido em papel e refinados por método iterativo e iterativo (entre os pesquisadores). A AIDEP-Abd1 é implementada também usando a Plataforma Microsoft .NET.

Tarefa 7: *Pesquisa experimental em laboratório para avaliar a usabilidade da Linguagem Abd1.* Uma avaliação da usabilidade da Linguagem Abd1, é realizada com o objetivo de verificar a facilidade de aprendizagem e de escrita de programas, bem como a satisfação subjetiva do programador em usar a linguagem para escrever programas que

⁹ Opção pela definição menos formal do que pela definição por meio de relações de transição.

fazem raciocínio abductivo. O Capítulo 5 discute em detalhes o método de pesquisa empregado.

Capítulo 3

Projeto da Linguagem Abd1

Este capítulo descreve a concepção e o projeto de uma linguagem para programação de raciocínios abduativos, nomeada Abd1, fundamentada em Lógica Proposicional. O nome Abd1 é devido às iniciais do termo “abdução” (e também “*abduction*”, do Inglês) e o numeral “1” refere-se ao primeiro de um possível conjunto de protótipos de linguagens para programação de raciocínios abduativos. A Seção 3.1 descreve a concepção lógica da linguagem Abd1, abordando os principais elementos que influenciam o *design* desta linguagem: (1) estrutura de raciocínio TCHF, forma HF de sentenças em Lógica Proposicional, (2) uso de terminologia do domínio da abdução (e.g. teoria, hipóteses, fatos), (3) uso de um único paradigma de programação, o declarativo lógico (puro), (4) critérios para escolha de boas hipóteses, (5) entendimento dos resultados de raciocínios como conjuntos, e (6) possibilidade de composição de raciocínios mais complexos por meio de operações entre conjuntos de hipóteses etc. A Seção 3.2 define informalmente os principais elementos léxicos, sintáticos e semânticos da Linguagem Abd1 e, por fim, a Seção 3.3 apresenta o projeto da Abd1 formalizado por meio de uma Gramática Livre de Contexto.

3.1 Fundamentos lógicos do *design* da linguagem Abd1

A Abd1 é fundamentada em Lógica Proposicional (LP). As várias subseções desta seção apresentam em detalhes os fundamentos lógicos do *design* da linguagem Abd1. O uso da palavra “*design*” e não da palavra “projeto” no título desta seção é para fazer referência a elementos ligados à concepção, conceito, paradigma, ideia e não propriamente ao “desenho”, “plano” da linguagem como algo descrito por uma gramática.

Forma HF

Por razões ligadas a eficiência de algoritmos para abdução, este trabalho emprega uma restrição à linguagem da LP. Esta restrição é conseguida por meio da imposição da escrita de sentenças da linguagem da LP à Forma HF (Definição 3.1), proposta originalmente em Rodrigues, Oliveira e Oliveira (2014). A vantagem desta abordagem é que sentenças na Forma HF podem ser eficientemente convertidas em sentenças na Forma Normal Conjuntiva com cláusulas Horn que, por sua vez, permite a implementação de algoritmo de resolução eficiente que, por fim, possibilita o desenvolvimento de algoritmos de abdução eficientes. O algoritmo Peirce que é utilizado pelo autômato com pilha que executa programas escritos em Abd1 tira partido disto para executar raciocínio eficientemente.

Definição 3.1 (Forma HF) (cf. Rodrigues, Oliveira e Oliveira (2014)). Uma sentença da linguagem da LP na forma HF é uma sentença escrita em um dos seguintes formatos:

- **Forma 1:** $a_1 \wedge a_2 \wedge a_3 \wedge \dots \wedge a_n$, onde a_i ($1 \leq i \leq n$) são literais.
- **Forma 2:** $a_1 \vee a_2 \vee a_3 \vee \dots \vee a_n$, onde a_i ($1 \leq i \leq n$) são átomos negados.
- **Forma 3:** $a_1 \wedge a_2 \wedge a_3 \wedge \dots \wedge a_n \rightarrow b_1 \wedge b_2 \wedge b_3 \wedge \dots \wedge b_m$, onde a_i ($1 \leq i \leq n$) são átomos e b_j ($1 \leq j \leq m$) são literais.
- **Forma 4:** $a_1 \vee a_2 \vee a_3 \vee \dots \vee a_n \rightarrow b_1 \vee b_2 \vee b_3 \vee \dots \vee b_m$, onde a_i ($1 \leq i \leq n$) são literais e b_j ($1 \leq j \leq m$) são átomos negados.

■

Exemplo 3.1. As seguintes sentenças estão escritas na forma HF:

- $p, \neg chuva, p \wedge q, \neg p \wedge \neg q \wedge r$ (Forma 1).
- $\neg p, \neg chuva \vee \neg nuvem, \neg p \vee \neg q \vee \neg r$ (Forma 2).

- $chuva \rightarrow gramadoMolhado, p \wedge q \rightarrow r, p \rightarrow \neg q \wedge \neg r \wedge s$ (Forma 3).
- $p \rightarrow \neg q, p \vee \neg q \rightarrow \neg r, \neg p \vee q \rightarrow \neg r \vee \neg s$ (Forma 4).

■

Exemplo 3.2. As seguintes sentenças **não** estão escritas na forma HF:

- $p \vee \neg q$ (aparentemente seria Forma 2, mas p não é um átomo negado).
- $p \wedge q \vee r$ (não se enquadra em nenhuma forma).
- $\neg p \wedge q \rightarrow r$ (aparentemente seria Forma 3, mas p é um átomo negado).
- $p \vee \neg q \rightarrow r$ (aparentemente seria Forma 4, mas r não é um átomo negado).

■

Estrutura de raciocínio TCHF

Rodrigues, Oliveira e Oliveira (2014) propõem a estrutura de raciocínio denominada TCHF (Definição 3.2) que inclui um novo elemento à estrutura clássica de raciocínio constituída por conjuntos teoria (T), hipóteses (H) e fatos (F). O novo elemento é o conjunto de condições C. A motivação para adição do conjunto C à estrutura clássica é a conveniência de empregá-lo para modelar intenções, contextos, circunstâncias, etc. sem ter que alterar o conjunto teoria T para capturar tais situações. Assim, é possível, por exemplo, usar uma mesma teoria T e fatos observados F para realizar um raciocínio ora em um contexto ora em outro contexto, apenas variando o conjunto C. O conjunto C também é útil para formalizar as condições que permitem empregar duas ou mais teorias em um mesmo raciocínio.

Definição 3.2 (Estrutura de raciocínio TCHF) (cf. Rodrigues, Oliveira e Oliveira (2014)). A estrutura TCHF para raciocínio abdutivo é um sistema $\langle T, C, H, F \rangle$ que consiste em:

- Um conjunto finito e não vazio, $T = \{ t_1, t_2, t_3, \dots, t_m \}$, de sentenças pertencentes à linguagem da LP, na Forma HF, denotando $t_1 \wedge t_2 \wedge t_3 \wedge \dots \wedge t_m$. Este conjunto representa as asserções que devem ser assumidas como verdade no momento do raciocínio abdutivo.
- Um conjunto finito, $C = \{ c_1, c_2, c_3, \dots, c_p \}$, de sentenças pertencentes à linguagem da LP, na Forma HF, denotando $c_1 \wedge c_2 \wedge c_3 \wedge \dots \wedge c_p$. Este conjunto

representa condições que devem ser assumidas como verdade no momento do raciocínio abduutivo.

- Um conjunto finito, $H = \{ h_1, h_2, h_3, \dots, h_n \}$, de sentenças pertencentes à linguagem da LP denotando $h_1 \vee h_2 \vee h_3 \vee \dots \vee h_n$. Este conjunto representa as hipóteses que juntamente com a teoria (T) e com as condições (C) explicam os fatos observados (F).
- Um conjunto finito e não vazio, $F = \{ f_1, f_2, f_3, \dots, f_q \}$, de átomos pertencentes à linguagem LP, denotando $f_1 \wedge f_2 \wedge f_3 \wedge \dots \wedge f_q$. Este conjunto representa os fatos que devem ser explicados através do raciocínio abduutivo. Fatos representam evidências, sintomas, observações, marcas, sinais, etc..

■

Os principais elementos linguísticos da linguagem Abd1 são inspirados na estrutura TCHF. Assim, como será descrito, a linguagem Abd1 possui estruturas *theory*, *conditions*, *facts* que permitem definir uma ou mais teorias, condições e fatos. As execuções de sentenças da Abd1 permitem realizar raciocínios cujo resultado são conjunto de hipóteses.

Exemplos de emprego da estrutura de raciocínios TCHF para formalizar raciocínios abduativos são dados na próxima subseção.

Formalização de problemas de abdução com a estrutura de raciocínios TCHF

Problemas de abdução podem ser convenientemente formalizados por meio de sentenças da linguagem da LP na forma HF utilizando a estrutura de raciocínio TCHF (Definição 3.3).

Definição 3.3 (Problema de abdução descrito pela estrutura TCHF). Um problema de abdução descrito pela estrutura da raciocínio TCHF é um sistema $\mathcal{P} = \langle T, C, H, F \rangle$ constituído por um conjunto teoria T, um conjunto de condições C e um conjunto de fatos F, satisfazendo as restrições impostas pela estrutura TCHF (Definição 3.2). Resolver um problema $\mathcal{P}$ é encontrar um conjunto de hipóteses H satisfazendo:

$$T \cup C \models F, \quad (3.1)$$

$$T \cup C \cup \{ h \} \models_p F, \forall h \in H, \quad (3.2)$$

$$T \cup C \cup \{ h \} \not\models \perp, \forall h \in H, \quad (3.3)$$

■

Garantia de generalidade de $T \cup C$. A condição (3.1) garante que $T \cup C$ seja geral, i.e., não explica sozinho os fatos do conjunto F . Se $T \cup C \models F$, então não haveria necessidade de se formular um conjunto de hipóteses H , pois $T \cup C$ sozinhos explicariam F . Além disso, este fato garante que $T \cup C$ não é uma contradição, pois se fosse, teria como consequência lógica qualquer coisa e, em particular, teria como consequência lógica F .

Hipóteses candidatas. Hipóteses $h \in H$ satisfazendo à condição (3.2) são chamadas *hipóteses candidatas*, e explicam um ou mais fatos presentes no conjunto F .

Hipóteses consistentes. Hipóteses candidatas satisfazendo à condição (3.3) são *hipóteses consistentes*, i.e., hipóteses que quando empregadas em conjunção com T e C , não geram uma Contradição.

A solução do problema $\mathcal{P}$ admite hipóteses chamadas de explanatórias, mas também, hipóteses conhecidas como não explanatórias. Uma hipótese h é não explanatória quando $\{ h \} \models F$, ou seja, h sozinha (sem a necessidade de $T \cup C$) explica o conjunto de fatos F e é explanatória caso contrário. Hipóteses não explanatórias são também conhecidas como triviais ou ingênuas.

Exemplo 3.3. Considerando novamente uma adaptação do “clássico da chuva”, este exemplo propõe a seguinte instanciação:

$$\begin{aligned} T = \{ &chuva \rightarrow gramadoMolhado, \\ &irrigadoresLigados \rightarrow gramadoMolhado, \\ &tanqueVazio \rightarrow \neg irrigadoresLigados \\ &\}, \\ C = \{ &tanqueVazio \}, \\ F = \{ &gramadoMolhado \}. \end{aligned}$$

O conjunto T é um possível modelo para os fenômenos que fazem um gramado estar molhado, incluindo proposições e relações lógicas que modelam chuva, irrigadores ligados e tanque vazio. O conjunto C está estabelecendo um contexto, no qual o tanque de água está vazio. Uma vantagem de usar o conjunto C é poder empregar a teoria T , ora para um raciocínio no contexto definido por C , ora sem este contexto. A teoria T não precisa ser modificada para permitir diferentes raciocínios.

Este modelo induz um problema de raciocínio abdutivo $\mathcal{P} = \langle T, C, H, F \rangle$ que consiste em determinar um conjunto de hipóteses H satisfazendo as sentenças (3.1), (3.2) e (3.3) da Definição 3.3. Observe que os conjuntos T , C e F satisfazem a sentença (3.1). Em outra situação, por exemplo, em que $C = \{ chuva \}$, ocorre de $T \cup C$ ter como consequência lógica F e, neste caso, não há satisfação da sentença (3.1).

Consideremos agora que, por algum método do cálculo proposicional, o seguinte conjunto de hipóteses H foi calculado:

$$H = \{ chuva, irrigadoresLigados, chuva \wedge irrigadoresLigados, \\ gramadoMolhado \}.$$

Cada hipótese h em H satisfaz à condição (3.2), portanto são hipóteses candidatas.

Cada hipótese

$$h \in \{ chuva, gramadoMolhado \}$$

satisfaz à condição (3.3), portanto, são hipóteses consistentes.

Cada hipótese

$$h \in \{ irrigadoresLigados, chuva \wedge irrigadoresLigados \}$$

não satisfaz (3.3), portanto são inconsistentes, devendo ser descartadas.

Após esta análise pode-se dizer que o problema de abdução $\mathcal{P}$ tem como solução o conjunto de hipóteses

$$H = \{ chuva, gramadoMolhado \}.$$

A hipótese

$$chuva$$

é explanatória, pois depende de $T \cup C$ para explicar F .

No entanto, a hipótese

gramadoMolhado

é não explanatória, pois $\{ \textit{gramadoMolhado} \} \models F$. Ela é trivial ou ingênua já que sugere que a hipótese *gramadoMolhado* explica o fato *gramadoMolhado*.

■

Algoritmo Peirce

Rodrigues, Oliveira e Oliveira (2014) propõe um algoritmo, chamado Peirce, para resolver um problema de abdução $\mathcal{P} = \langle T, C, H, F \rangle$ satisfazendo às restrições de solução da Definição 3.3. Adicionalmente, o algoritmo admite dois parâmetros ajustáveis (λ_1 e λ_2) para controlar o critério de seleção de boas hipóteses estabelecido pela Definição 3.4.

Definição 3.4 (critério para a seleção de boas hipóteses) (cf., Rodrigues, Oliveira e Oliveira (2014)). Dado um conjunto H de hipóteses para explicar um conjunto F de fatos, $h \in H$ é considerada uma boa hipótese se satisfaz às seguintes condições:

- A força explanatória de h é maior ou igual a uma constante λ_1 . O algoritmo Peirce assume por padrão uma constante $\lambda_1 = 0.5$.
- A complexidade de h é menor ou igual a uma constante λ_2 . O algoritmo Peirce assume por padrão uma constante $\lambda_2 = 5$.¹⁰
- A hipótese h possui complexidade mínima entre todas as hipóteses que tem a força explanatória máxima em H .

A força explanatória de uma hipótese é igual à relação entre a quantidade de fatos que ela explica e a quantidade total de fatos em F . A complexidade de uma hipótese é igual à quantidade de proposições que ela possui.

■

¹⁰ Os valores padrão de λ_1 e λ_2 foram escolhidos para atender a fatores humanos. Considerando o experimento de Miller (Mackenzie, 2013), a memória humana e a capacidade de processamento humano estão limitadas a 7 ± 2 elementos simultâneos, assim $\lambda_2 = 5$. Boas hipóteses explicam pelo menos 50% dos fatos observados, assim $\lambda_1 = 0.5$ (Rodrigues, Oliveira e Oliveira 2014, p. 128).

Exemplo 3.4. (traduzido de Rodrigues, Oliveira e Oliveira, 2014). Doenças são manifestadas por sintomas. Considere:

- As proposições "c: Doença é resfriado", "p: Doença é pneumonia", "r: Doença é rinite", "f: Sintoma é febre", "h: Sintoma é dor de cabeça", "z: Sintoma é coriza".
- A teoria $T = \{ p \rightarrow f \wedge z \wedge h, c \rightarrow f \wedge z, r \rightarrow h \wedge z \}$, o conjunto vazio de condições $C = \{ \}$ e os fatos (sintomas) observados $F = \{ f, z, h \}$.
- Um conjunto de hipóteses $H = \{ p, c, r, p \wedge c, p \wedge r, c \wedge r, p \wedge c \wedge r \}$.

A Tabela 3.1 descreve os fatos explicados, a força explanatória e a complexidade de cada hipótese candidata $h \in H$.

Tabela 3.1. Fatos explicados, força explanatória e complexidade de hipóteses do Exemplo 3.4. O símbolo '√' sinaliza um fato explicado.

Hipóteses	Fatos explicados			Força explanatória	Complexidade
	<i>f</i>	<i>z</i>	<i>h</i>		
<i>p</i>	√	√	√	1	1
<i>c</i>	√	√		0.66	1
<i>r</i>		√	√	0.66	1
$p \wedge c$	√	√	√	1	2
$p \wedge r$	√	√	√	1	2
$c \wedge r$	√	√	√	1	2
$p \wedge c \wedge r$	√	√	√	1	3

Todas as hipóteses possuem força explanatória maior do que $\lambda_1 = 0,5$ e complexidade menor que $\lambda_2 = 5$. As hipóteses $p, p \wedge c, p \wedge r, c \wedge r, p \wedge c \wedge r$ possuem força explanatória igual a 1, o máximo dentre todas as hipóteses, no entanto p possui a menor complexidade entre elas, fazendo com que p seja selecionada como uma boa hipótese.

■

Como foi descrito no Capítulo 1, o algoritmo Peirce está disponível para uso por meio da aplicação Web Peirce Online, disponível em “<http://peirceonline.gear.host/>”. Rodrigues (2015) apresenta uma descrição detalhada do algoritmo Peirce e da aplicação Peirce Online.

O autômato com pilha da Abd1, usa o algoritmo Peirce para computar os raciocínios abdutivos.

3.2 – Visão Informal da Linguagem Abd1

O projeto da linguagem Abd1 é motivado pelos fundamentos lógicos de *design* discutidos na Seção 3.1 e pela intenção de estabelecer uma linguagem para minimizar as dificuldades de programação de raciocínios abduativos identificadas na Seção 2.3, Capítulo 2.

Tendo por objetivo uma introdução didática à linguagem Abd1, esta seção descreve informalmente as suas estruturas léxicas e sintáticas. Uma descrição formal da gramática da Abd1 será dada na Seção 3.3.

A estrutura geral de um programa na linguagem Abd1 é inspirada na estrutura de raciocínio TCHF (Definição 3.2), que faz com que um programa nesta linguagem seja composto por teorias, condições e fatos. Teorias descrevem conhecimentos sobre um assunto. Na Linguagem Abd1 teorias podem ser descritas por meio de expressões que iniciam com o termo `Theory` seguido por um nome atribuído à teoria e por um conjunto de sentenças em LP na forma HF (Definição 3.1). Conjuntos de condições são empregados para estabelecer condições específicas (contextos, circunstâncias, intenções entre outras) para a realização de um raciocínio abduativo sem que teorias tenham que ser alteradas para incorporar tais elementos especializados. Condições são declaradas pelo termo `Conditions` seguido por um nome atribuído às condições e, tal como em declarações de teorias, por um conjunto de sentenças da LP na forma HF. Conjuntos de fatos servem para descrever, como o próprio termo sugere, fatos, observações, sinais ou sintomas que devem ser explicados. Na Linguagem Abd1 fatos são descritos por meio de expressões que iniciam com o termo `Facts` seguido por um nome atribuído ao fato e por um conjunto de proposições.

Um raciocínio abduativo é descrito pelo termo `AbductiveReasoning` seguido por um nome atribuído ao raciocínio e pela declaração dos elementos (teorias, fatos e condições), incluídos entre parênteses angulares (“{” e “}”), que participarão do raciocínio. A linguagem admite, na descrição de um raciocínio abduativo, a escrita de expressões contendo a união, a intersecção ou a subtração de teorias, de condições e de fatos. É possível também especificar, opcionalmente, critérios para seleção de boas hipóteses abduativas, incluindo numa seção iniciada pelo termo `options`, os parâmetros de controle do critério de escolha de boas hipóteses (Definição 3.4).

O resultado da execução de um raciocínio abduutivo é um conjunto de hipóteses. Cada hipótese dentro do conjunto de hipóteses é uma possível explicação para o raciocínio.

Exemplo 3.5. Considere o circuito constituído por duas lâmpadas, dois interruptores, uma bateria e fios, cujo diagrama elétrico está ilustrado na Figura 3.1. Como já especificado no exemplo 2.5, no circuito a lâmpada 2 acende se a bateria possuir carga, os fios estiverem conectados e o interruptor 1 estiver ligado. Para que a lâmpada 1 acenda é necessário, adicionalmente, que o interruptor 2 esteja ligado.

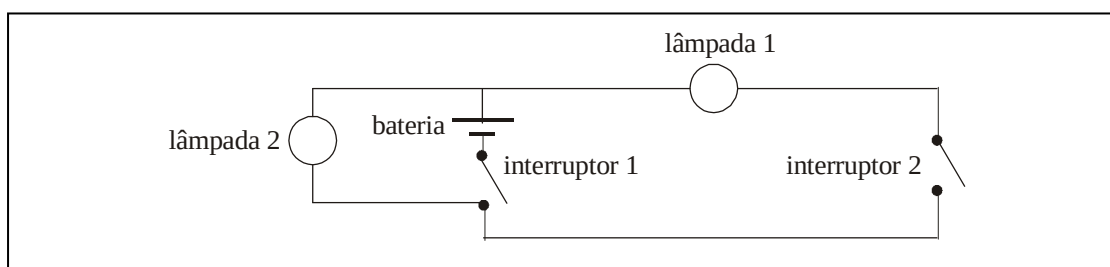


Figura 3.1. Diagrama elétrico de um circuito que é utilizado como cenário para exemplificar um programa na Linguagem Abd1 (reprodução da Figura 2.5).

A Figura 3.2 apresenta um programa que estabelece um modelo sobre o funcionamento do circuito. Duas teorias estão sendo especificadas. A teoria denominada `Teoria_Não_Funcionamento` especifica, por meio de quatro sentenças (linhas 3 a 6), falhas e funcionalidades que fazem as lâmpadas do circuito estarem apagadas. Por exemplo, a sentença da linha 3 estabelece que “se `bateria_sem_carga` então `lâmpada_1_apagada` e `lâmpada_2_apagada`”. As demais sentenças desta teoria estabelecem relações lógicas entre as proposições `fios_rompidos`, `interruptor_1_desligado`, `interruptor_2_desligado`, `lâmpada_1_apagada` e `lâmpada_2_Apagada`. A teoria `Teoria_Funcionamento` (linhas 9 a 12) especifica, em duas sentenças, o que faz as lâmpadas do circuito estarem acesas. Por exemplo, a sentença da linha 11 estabelece que “se `bateria_com_carga` e `fios_conectados` e `interruptor_1_ligado` então `lâmpada_2_acesa`”. Há uma particularidade em relação ao que estabelecem estas duas teorias que é importante ressaltar: elas modelam, por meio das proposições `fios_rompidos` (linha 4) e `fios_conectados` (linhas 10 e 11), os fios do circuito como se fossem um bloco único, isto significa que os fios não estão sendo tratados como diversos segmentos independentes.

As duas teorias descritas explicam diferentes fenômenos que ocorrem no circuito. Estas teorias podem ser usadas isoladamente na realização de um raciocínio abduutivo, mas para que possam ser utilizadas conjuntamente a condição `Condição_Correlação_Teorias` está sendo especificada (linhas 14 a 17). Basicamente, esta condição estabelece relações lógicas entre as proposições utilizadas nas duas teorias. Assim, por exemplo, uma sentença na linha 15 estabelece que a proposição `bateria_com_carga` da teoria `Teoria_Funcionamento` possui a seguinte relação lógica com a proposição `bateria_sem_carga` da teoria `Teoria_Não_Funcionamento`: “se `bateria_com_carga` então não `bateria_sem_carga`”. A condição `Condição_Interruptores_Não_Desligados` (linha 19) estabelece um contexto especial para realização de raciocínios com a teoria `Teoria_Não_Funcionamento`. Esta condição estabelece que os interruptores 1 e 2 não estão desligados. Um fato é descrito na linha 21. O fato denominado `Fato_1` é um conjunto unitário que especifica `lâmpada_1_apagada`.

Raciocínios abdutivos estão sendo descritos nas linhas 23 a 28. `Explicações_Lâmpada_1_Apagada` especifica um raciocínio para encontrar hipóteses que explicam o `Fato_1` tendo como teoria a `Teoria_Não_Funcionamento`. `Explicações_Lâmpada_2_Apagada` especifica algo semelhante ao primeiro raciocínio. As diferenças desta especificação para a primeira são: (1) em vez de referenciar um fato já nomeado no programa, o conjunto unitário de fatos { `lâmpada_2_apagada` } está sendo diretamente descrito na especificação do raciocínio; (2) objetiva-se obter hipóteses que expliquem o fato `lâmpada_2_apagada`. `Explicações_Lâmpada_1_Apagada_Lâmpada_2_Acesa` especifica um raciocínio para encontrar hipóteses que explicam fatos presentes nas duas teorias definidas. Para isto, estão sendo descritas uma expressão para especificar a união das duas teorias e uma condição para correlação lógica entre elas. Outra particularidade da especificação deste raciocínio é o uso de uma expressão para descrever a união de dois conjuntos de fatos.

```

1 // Teoria sobre falhas e funcionalidades que fazem as lâmpadas do circuito estarem apagadas.
2 Theory Teoria_Não_Funcionamento {
3   bateria_sem_carga → lâmpada_1_apagada ∧ lâmpada_2_apagada,
4   fios_rompidos → lâmpada_1_apagada ∧ lâmpada_2_apagada,
5   interruptor_1_desligado → lâmpada_1_apagada ∧ lâmpada_2_apagada,
6   interruptor_2_desligado → lâmpada_1_apagada
7 }

8 // Teoria sobre o que faz as lâmpadas do circuito estarem acesas.
9 Theory Teoria_Funcionamento {
10  bateria_com_carga ∧ fios_conectados ∧ interruptor_1_ligado ∧ interruptor_2_ligado → lâmpada_1_acesa,
11  bateria_com_carga ∧ fios_conectados ∧ interruptor_1_ligado → lâmpada_2_acesa
12 }

13 // Condições para correlacionar as teorias Teoria_Não_Funcionamento e Teoria_Funcionamento.
14 Conditions Condição_Correlação_Teorias {
15  bateria_com_carga → ¬ bateria_sem_carga, interruptor_1_desligado → ¬ interruptor_1_ligado,
16  interruptor_2_desligado → ¬ interruptor_2_ligado, fios_rompidos → ¬ fios_conectados
17 }

18 // Condições que especificam que os interruptores estão ligados.
19 Conditions Condição_Interruptores_Não_Desligados { ¬ interruptor_1_desligado, ¬ interruptor_2_desligado }

20 // Alguns conjuntos de fatos.
21 Facts Fato_1 { lâmpada_1_apagada }

22 // Alguns raciocínios abduativos.
23 AbductiveReasoning Explicações_Lâmpada_1_Apagada < Teoria_Não_Funcionamento, Fato_1 >
24 AbductiveReasoning Explicações_Lâmpada_2_Apagada < Teoria_Não_Funcionamento, { lâmpada_2_apagada } >
25 AbductiveReasoning Explicações_Lâmpada_1_Apagada_Lâmpada_2_Acesa <
26   Teoria_Não_Funcionamento ∪ Teoria_Funcionamento, { lâmpada_1_apagada, lâmpada_2_acesa },
27   Condição_Correlação_Teorias
28 >

```

Figura 3.2. Programa estabelecendo um modelo sobre o funcionamento do circuito da Figura 3.1.

A Figura 3.3 apresenta quatro exemplos de execução do programa da Figura 3.2 por meio de um ambiente interativo, onde se pode executar o que está descrito no programa. O ambiente interativo sinaliza, com o sinal de pronto “>” na linha de comando, que aguarda a digitação de uma expressão descrevendo a união, a interseção e a subtração de raciocínios abduativos. Em resposta à expressão digitada, o interpretador executa o raciocínio e apresenta como resposta um conjunto de hipóteses para explicar os fatos definidos no raciocínio. A Figura 3.3-a ilustra a execução do raciocínio *Explicações_Lâmpada_1_Apagada*. A Figura 3.3-b exemplifica uma intersecção de dois raciocínios, cujo resultado descreve as hipóteses que explicam, concomitantemente, os fatos das lâmpadas 1 e 2 estarem apagadas. A Figura 3.3-c exemplifica o emprego de uma condição na realização de um raciocínio. Este exemplo, descreve um raciocínio para buscar hipóteses que explicam os fatos das lâmpadas 1 e 2 estarem apagadas, tendo como condição para realização do raciocínio os interruptores 1 e 2 não estarem desligados. Uma particularidade presente neste exemplo é a descrição direta dos

elementos que compõem um raciocínio (teoria, fatos e condição) na linha de comando do ambiente interativo. O exemplo da Figura 3.3-d mostra a execução de um raciocínio envolvendo conjuntamente as duas teorias definidas no programa. O raciocínio objetiva encontrar hipóteses que explicam os fatos `lâmpada_1_apagada` e `lâmpada_2_acesa`.

<pre>> Explicações_Lâmpada_1_Apagada. > { bateria_sem_carga, fios_rompidos, interruptor_1_desligado, interruptor_2_desligado }</pre> (a)	<pre>> Explicações_Lâmpada_1_Apagada ∩ Explicações_Lâmpada_2_Apagada. > { bateria_sem_carga, fios_rompidos, interruptor_1_desligado }</pre> (b)
<pre>> (Teoria_Não_Funcionamento, { lâmpada_1_apagada, lâmpada_2_apagada }, Condição_Interruptores_Não_Desligados). > { bateria_sem_carga, fios_rompidos }</pre> (c)	<pre>> Explicações_Lâmpada_1_Apagada_Lâmpada_2_Acesa. > { bateria_com_carga ∧ fios_conectados ∧ interruptor_1_ligado ∧ interruptor_2_desligado }</pre> (d)

Figura 3.3. Exemplos de execução do programa da Figura 3.2.

3.3 – Gramática da Linguagem Abd1

O projeto gramatical da linguagem Abd1 está completamente descrito no Apêndice I. Esta seção contém um resumo dos principais elementos da gramática da linguagem Abd1 com o objetivo de oferecer uma visão geral dela. Aspectos específicos podem ser consultados no Apêndice I.

O formalismo de expressão gramatical empregado nesta seção e no Apêndice I é uma adaptação do formalismo usualmente empregado por textos tradicionais de Linguagens Formais e Autômatos (e.g., Hopcroft, Motwani & Ullman, 2006) para descrever Gramáticas Livres de Contexto. A adaptação¹¹ é a seguinte:

- Variáveis: são denotadas por palavras escritas entre os símbolos “<” e “>”, em vez das tradicionais letras maiúsculas.
- Símbolos terminais: são denotados por palavras e símbolos especiais de comprimento igual a um (‘{’, ‘}’, ‘(’, ‘)’, ‘:’, ‘;’, ‘→’ etc.) destacados em negrito, em vez das tradicionais letras minúsculas.
- Produções: são denotadas no formato $X \rightarrow \alpha$ onde X é uma variável e α pertence ao fecho reflexivo e transitivo da união do conjunto de variáveis com o conjunto de símbolos terminais. A diferença em relação aos textos tradicionais é

¹¹ Esta adaptação tem o propósito de facilitar a leitura de gramática longas, como é o caso da gramática da linguagem Abd1, e não é o caso das gramáticas usadas tradicionalmente em livros de Linguagens Formais.

que “->” está sendo usado no lugar de “→”, uma vez que este último símbolo é amplamente utilizado nas produções da gramática de Abd1.

A gramática da linguagem Abd1 é dividida em dois ambientes:

- Ambiente de edição de programas
- Ambiente de execução de raciocínios abdutivos.

O conjunto de sentenças que podem ser escritas em cada um destes ambientes é diferente um do outro. Embora existam dois ambientes, a gramática da linguagem Abd1 não é uma gramática sensível ao contexto. O que se tem são duas gramáticas do tipo livre de contexto, uma para cada um dos ambientes.

No ambiente de edição de programas (Seção I.2, Apêndice I) o símbolo inicial da gramática é <programa> sendo que um programa definido como uma sequência de sentenças declarativas:

<programa> -> <sequenciaSentencaDeclarativa>

Uma sequência de sentenças declarativas é uma repetição de uma ou mais sentenças declarativas:

<sequenciaSentencaDeclarativa> -> <sentencaDeclarativa> |
<sentencaDeclarativa> <sequenciaSentencaDeclarativa>

Uma sentença declarativa, por sua vez, é a declaração de uma teoria, ou uma declaração de condições, ou uma declaração de fatos ou uma declaração de raciocínio abductivo:

<sentencaDeclarativa>-> <declTeoria> |
<declCondicoes> |
<declFatos> |
<declRaciocinioAbductivo>

No ambiente de execução de raciocínios abdutivos (Seção I.3, Apêndice I) o símbolo inicial da gramática é <questao> sendo uma questão definida como uma sequência de sentenças de execução:

<questao> -> <sequenciaSentencaExecucao>

Uma sequência de sentenças de execução é uma sequência de uma ou mais expressões de raciocínio separadas por um ponto e vírgula. A última expressão de raciocínio da sequência tem que terminar por um ponto:

```
<sequenciaSentencaExecucao> -> <expressaoRaciocinio> . |
<expressaoRaciocinio> ; <sequenciaSentencaExecucao>
```

Uma expressão descrevendo um raciocínio é uma expressão que envolve identificadores de raciocínio e operadores de intersecção, união e diferença entre conjuntos (nesta ordem de prioridade). Parêntesis podem ser utilizados para modificar a ordem de execução dos raciocínios.

Do ponto de vista léxico, a Abd1 oferece facilidades de não ser sensível a maiúsculas e minúsculas, permitir o uso de caracteres do Português (cedilha, acentuação etc.) e usar sinônimos para muitos dos seus símbolos terminais, o que, por um lado, permite facilidade de edição e, por outro lado, elegância das sentenças. Alguns destes sinônimos são¹²:

- ^, ∧, and, AND.
- v, ∨, or, OR.
- !, ¬, not, NOT.
- ->, →.
- ∩, intersection, INTERSECTION.
- U, union, UNION.
- -, minus, MINUS.
- <, ⟨.
- >, ⟩.
- λ1, lambda1, Lambda1, MinExplanatoryPower.
- λ2, lambda2, Lambda2, MaxComplexity.

¹² Cada tópico (linha) descreve um conjunto de símbolos que são sinônimos.

Capítulo 4

Implementação da Linguagem e do Ambiente de Desenvolvimento

Este capítulo descreve a implementação da Linguagem Abd1 (compilador e autômato), e também a implementação do Ambiente Integrado de Desenvolvimento e Execução de Programas escritos em Abd1 (AIDEP-Abd1). O capítulo inicia com a descrição do compilador de programas escritos em Abd1 (Seção 4.1). Códigos objetos gerados pelo compilador são executados pelo Autômato com Pilha Abd1, cujo funcionamento é discutido na Seção 4.2. Por fim, a Seção 4.3 retrata o AIDEP-Abd1.

4.1 – Compilador de programas Abd1

O compilador é formado por componentes pertencentes a dois ambientes: o ambiente do programa e o ambiente da execução de raciocínios abduativos.

A Figura 4.1 ilustra, por meio de um diagrama de blocos, os principais componentes do compilador de programas Abd1.

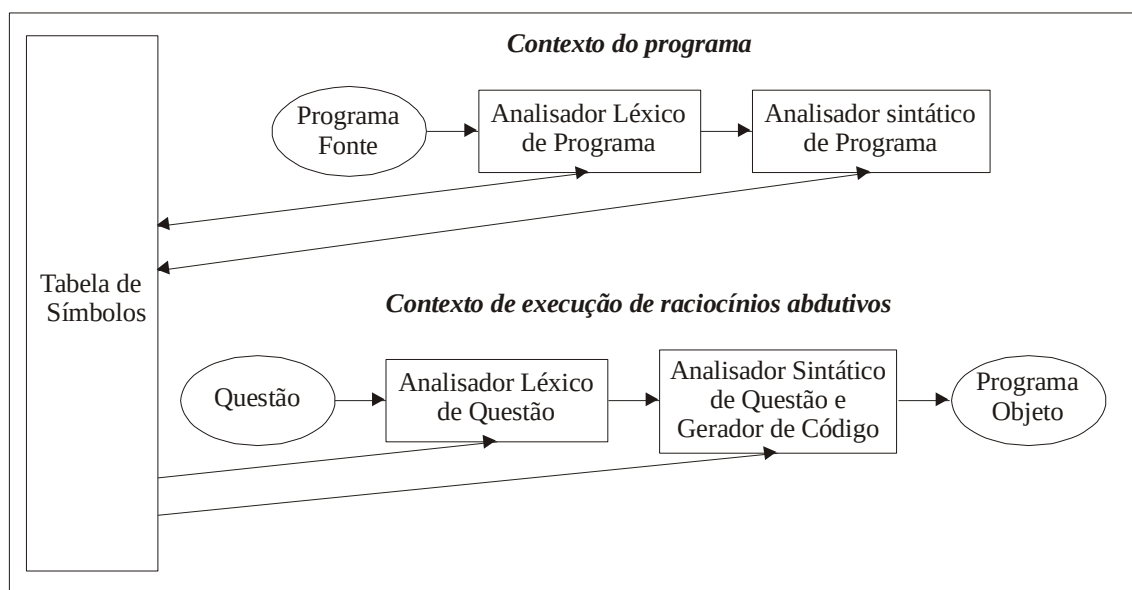


Figura 4.1. Diagrama de blocos do compilador de programas Abd1.

No ambiente do programa, um programa fonte é analisado lexicamente e sintaticamente de acordo com a gramática do ambiente de edição de programas (Apêndice I, Seção I.2). Erros léxicos e sintáticos são reportados, caso existam. Os resultados da análise léxica e sintática de um programa correto léxico e gramaticalmente são armazenados na Tabela de Símbolos para uso por outros componentes do compilador. A Tabela de Símbolos armazena informações sobre todos os elementos do programa: teorias, condições, fatos, raciocínios abduativos, proposições e sentenças declarativas.

Diversas questões podem ser formuladas tendo como base um programa. No ambiente de execução de raciocínios abduativos, uma questão é analisada lexicalmente e sintaticamente de acordo com a gramática do ambiente de execução de raciocínios abduativos (Apêndice I, Seção I.3) e erros são reportados caso existam. O analisador léxico e sintático de questão utiliza os elementos da Tabela de Símbolos para analisar e gerar o código do Programa Objeto correspondente a uma questão formulada sobre um programa.

Tecnicamente, o Analisador Léxico de Programa e o Analisador Léxico de Questão são autômatos finitos determinísticos (Aho, Lam, Sethi & Ullman, 2006) e o código destes autômatos foram codificados sem o uso de ferramentas automáticas. O Analisador Sintático de Programa e o Analisador Sintático de Questão foram escritos usando a técnica da análise sintática descendente recursiva (Aho, Lam, Sethi & Ullman, 2006). Ao todo 5005 linhas de código em linguagem C# em ambiente de desenvolvimento .NET foram codificadas.

O Programa Objeto gerado pelo compilador é para ser executado por um autômato com pilha que chamamos de Autômato com Pilha Abd1. Por ora, é suficiente mencionar que o autômato usa um pequeno conjunto de instruções para executar um raciocínio (*exec*) e fazer a intersecção (*inter*), a união (*uni*) e a diferença entre raciocínios (*dif*). O funcionamento do Autômato com Pilha Abd1 será discutido na próxima seção.

O Exemplo 4.1 ilustra o funcionamento do compilador de programas Abd1.

Exemplo 4.1. A Figura 4.2 contém um programa fonte escrito na Linguagem Abd1. O programa estabelece um modelo simplificado do circuito discutido na Seção 3.2 e ilustrado na Figura 3.1. O programa contém a definição de uma teoria (*Teoria_Não_Funcionamento*), dois raciocínios abdutivos (*Explicações_Lâmpada_1_Apagada* e *Explicações_Lâmpada_2_Apagada*) e seis proposições (*bateria_sem_carga*, *lâmpada_1_apagada*, *lâmpada_2_apagada*, *fios_rompidos*, *interruptor_1_desligado* e *interruptor_2_desligado*). O raciocínio abductivo *Explicações_Lâmpada_1_Apagada* possui um campo de opções que declara um valor mínimo para a força explanatória e um valor máximo para a complexidade. O raciocínio abductivo *Explicações_Lâmpada_2_Apagada* não possui o campo de opções, o que significa que os valores para força explanatória mínima e complexidade máxima são os valores padrão, ou seja, 0.5 e 5, respectivamente.

```

1  Theory Teoria_Não_Funcionamento {
2    bateria_sem_carga  $\rightarrow$  lâmpada_1_apagada  $\wedge$  lâmpada_2_apagada,
3    fios_rompidos  $\rightarrow$  lâmpada_1_apagada  $\wedge$  lâmpada_2_apagada,
4    interruptor_1_desligado  $\rightarrow$  lâmpada_1_apagada  $\wedge$  lâmpada_2_apagada,
5    interruptor_2_desligado  $\rightarrow$  lâmpada_1_apagada
6  }

7  AbductiveReasoning Explicações_Lâmpada_1_Apagada
8  <
9    Teoria_Não_Funcionamento,
10   { lâmpada_1_apagada }
11  > Options { MinExplanatoryPower = 0.5, MaxComplexity = 5 }

12 AbductiveReasoning Explicações_Lâmpada_2_Apagada
13 <
14   Teoria_Não_Funcionamento,
15   { lâmpada_2_apagada }
16 >

```

Figura 4.2. Um código fonte para exemplificar o funcionamento do compilador Abd1.

A análise léxica e sintática do programa fonte da Figura 4.2 produz a Tabela de Símbolos apresentada na Figura 4.3. Uma Tabela de Símbolos contém um símbolo para cada teoria, condição, fato, raciocínio abdutivo e proposição; informações sobre cada símbolo são justapostas a ele. Em particular, nesta figura podem ser vistos na Tabela de Símbolos, os símbolos para a teoria, para os dois raciocínios e para uma proposição do programa fonte da Figura 4.2. As outras cinco proposições do programa fonte não estão representadas na figura, embora deve-se supor que existem os correspondentes símbolos na tabela.

O símbolo `Teoria_Não_Funcionamento` está associado a informações sobre o tipo do símbolo, neste caso “T”, representando que o símbolo se refere à declaração de uma teoria, e a lista de sentenças que define a teoria.

O símbolo `Explicações_Lâmpada_1_Apagada` está associado a informações sobre o tipo do símbolo, neste caso “R”, representando que o símbolo se refere à declaração de um raciocínio abdutivo, uma lista de teorias, neste caso uma lista com um único nó, uma lista de fatos, neste caso também uma lista com um único nó, uma lista de condições, neste caso uma lista vazia, um valor para a força explanatória mínima e um valor para a complexidade máxima. A descrição do símbolo `Explicações_Lâmpada_2_Apagada` é semelhante à descrição do símbolo `Explicações_Lâmpada_1_Apagada`. Cabe observar que, mesmo não tendo sido declarado no programa fonte valores para a força

explanatória mínima e para a complexidade máxima, os valores padrão destes parâmetros são preenchidos na tabela de símbolos.

Proposições são associadas na Tabela de Símbolos ao tipo a que se refere, sendo utilizado a letra “P” para este propósito. A figura apresenta o símbolo `bateria_sem_carga` associado ao tipo “P”.

Símbolo	Tipo de símbolo (T - teoria)	Lista de sentenças
Teoria_Não_Funcionamento	T	$bateria_sem_carga \rightarrow lâmpada_1_apagada \wedge lâmpada_2_apagada$
		$fios_rompidos \rightarrow lâmpada_1_apagada \wedge lâmpada_2_apagada$
		$interruptor_1_desligado \rightarrow lâmpada_1_apagada \wedge lâmpada_2_apagada$
		$interruptor_2_desligado \rightarrow lâmpada_1_apagada$
Explicações_Lâmpada_1_Apagada	R	Tipo de símbolo (R - raciocínio)
		Teoria_Não_Funcionamento $\rightarrow$ Lista de teorias
		lâmpada_1_apagada $\rightarrow$ Lista de fatos
		Lista de condições
Explicações_Lâmpada_2_Apagada	R	Tipo de símbolo (R - raciocínio)
		Teoria_Não_Funcionamento $\rightarrow$ Lista de teorias
		lâmpada_2_apagada $\rightarrow$ Lista de fatos
		Lista de condições
bateria_sem_carga	P	Força explanatória mínima
		Complexidade máxima
...		Tipo de símbolo (P - proposição)

Figura 4.3. Tabela de Símbolos resultante da análise léxica e sintática do programa fonte da Figura 4.2.

A Figura 4.4 apresenta duas questões e os respectivos programas objeto gerados pelo compilador da linguagem Abd1. As duas questões apresentadas (Figuras 4.4-a e 4.4-c) são parecidas, mas diferem na ordem em que os raciocínios devem ser executados. Como a operação de intersecção tem prioridade sobre a operação de diferença de conjuntos, a expressão da Figura 4.4-a indica que o resultado da execução do raciocínio `Explicações_Lâmpada_1_Apagada` deve ser subtraído da intersecção dos resultados dos raciocínios `Explicações_Lâmpada_2_Apagada` e `Explicações_Lâmpada_2_Apagada`. Diferentemente, a expressão da Figura 4.4-c indica que primeiro deve ser realizado o

cálculo da diferença entre o resultado do raciocínio `Explicações_Lâmpada_1_Apagada` e o raciocínio `Explicações_Lâmpada_2_Apagada` e, posteriormente, deve-se realizar a intersecção entre o resultado deste calculo e o resultado do raciocínio `Explicações_Lâmpada_2_Apagada`. O programa objeto da Figura 4.4-b é a tradução da expressão da Figura 4.4-a nas instruções do Autômato Abd1. Similarmente, o programa objeto da Figura 4.4-d é a tradução da expressão da Figura 4.4-c nas instruções do Autômato Abd1. A próxima Seção descreve o funcionamento do Autômato Abd1 e exemplifica como este autômato executa estes programas objeto.

<i>Questão</i>	<i>Programa Objeto</i>
$\text{Explicações_Lâmpada_1_Apagada} - \text{Explicações_Lâmpada_2_Apagada}$ $\cap \text{Explicações_Lâmpada_2_Apagada.}$ (a)	$\text{exec (Explicações_Lâmpada_1_Apagada)}$ $\text{exec (Explicações_Lâmpada_2_Apagada)}$ $\text{exec (Explicações_Lâmpada_2_Apagada)}$ inter dif (b)
<i>Questão</i>	<i>Programa Objeto</i>
$(\text{Explicações_Lâmpada_1_Apagada} - \text{Explicações_Lâmpada_2_Apagada})$ $\cap \text{Explicações_Lâmpada_2_Apagada.}$ (c)	$\text{exec (Explicações_Lâmpada_1_Apagada)}$ $\text{exec (Explicações_Lâmpada_2_Apagada)}$ dif $\text{exec (Explicações_Lâmpada_2_Apagada)}$ inter (d)

Figura 4.4. Questões e programas objetos gerados pelo compilador: (a) e (c) são questões, (b) e (d) são os respectivos programas objeto.

■

4.2 – Autômato com Pilha Abd1

O Autômato com Pilha Abd1 é um autômato com pilha (Hopcroft, Motwani & Ullman, 2006) que utiliza o seguinte conjunto de instruções:

- `exec(r)`: instrução para executar o raciocínio abdutivo de nome *r* e empilhar o conjunto de hipóteses resultante do raciocínio. Esta instrução utiliza o algoritmo Peirce (Seção 3.1) para executar o raciocínio abdutivo.
- `inter`: instrução para desempilhar, em sequência, dois conjuntos de hipóteses, digamos h_1 e h_2 , computar o conjunto $h = h_1 \cap h_2$ e empilhar *h*.
- `uni`: instrução para desempilhar, em sequência, dois conjuntos de hipóteses, digamos h_1 e h_2 , computar o conjunto $h = h_1 \cup h_2$ e empilhar *h*.

- `dif`: instrução para desempilhar, em sequência, dois conjuntos de hipóteses, digamos h_1 e h_2 , computar o conjunto $h = h_2 - h_1$ e empilhar h .

Para execução das instruções o Autômato com Pilha Abd1 faz uso da Tabela de Símbolos gerada pelo compilador da Linguagem Abd1. O Exemplo 4.2 e o Exemplo 4.3 ilustram o funcionamento do Autômato com Pilha Abd1.

Exemplo 4.2. Considere o programa objeto da Figura 4.4-b. A Figura 4.5¹³ ilustra a execução deste programa objeto pelo Autômato com Pilha Abd1. Inicialmente a pilha está vazia (Figura 4.5-a). A instrução `exec(Explicações_Lâmpada_1_Apagada)` consulta a Tabela de Símbolos (Figura 4.3) para montar as informações necessárias para executar o algoritmo Peirce e realizar o raciocínio abdutivo. O conjunto de hipóteses resultante é empilhado (Figura 4.5-b). De forma semelhante, em sequência, a instrução `exec(Explicações_Lâmpada_2_Apagada)` é executada, o resultado é empilhado (Figura 4.5-c), a instrução `exec(Explicações_Lâmpada_2_Apagada)` é executada e o resultado também é empilhado (Figura 4.5-d). A instrução `inter` desempilha duas vezes e associa o que foi desempilhado às variáveis, digamos h_1 e h_2 , sendo

$h_1 = \{ \text{bateria_sem_carga}, \text{ fios_rompidos}, \text{ interruptor_1_desligado} \}$ e

$h_2 = \{ \text{bateria_sem_carga}, \text{ fios_rompidos}, \text{ interruptor_1_desligado} \}.$

O resultado da intersecção entre h_1 e h_2 é empilhado (Figura 4.5-e). Por fim, a instrução `dif` desempilha duas vezes e associa o que foi desempilhado às variáveis, digamos h_1 e h_2 , sendo

$h_1 = \{ \text{bateria_sem_carga}, \text{ fios_rompidos}, \text{ interruptor_1_desligado} \}$ e

$h_2 = \{ \text{bateria_sem_carga}, \text{ fios_rompidos}, \text{ interruptor_1_desligado}, \text{ interruptor_2_desligado} \}.$

O resultado da subtração de h_2 por h_1 ($h_2 - h_1$) é empilhado (Figura 4.5-f). O que está no topo da pilha da Figura 4.5-f é o resultado da execução do programa objeto.

¹³ Para facilitar a leitura optou-se por representar os elementos da pilha como conjuntos, em vez da real estrutura de dados utilizada na implementação.

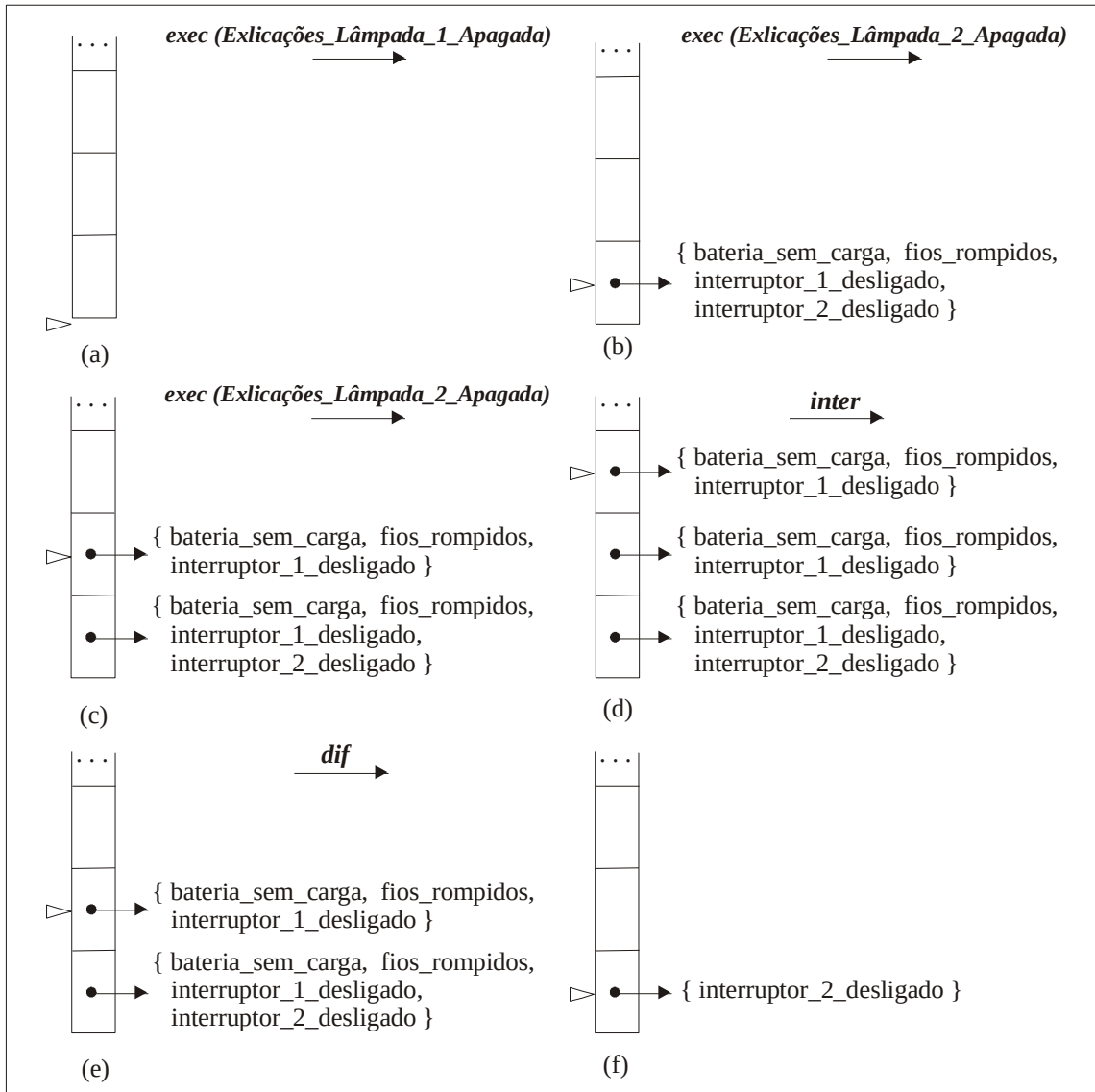


Figura 4.5. Execução do programa objeto da Figura 4.4-b pelo Autômato com Pilha Abd1.

Exemplo 4.3. Considere agora o programa objeto apresentado na Figura 4.4-d. A Figura 4.6 ilustra a execução deste programa objeto pelo Autômato com Pilha Abd1. As duas primeiras instruções deste programa objeto são iguais às duas primeiras instruções do programa objeto discutido no exemplo anterior. Assim, partindo da pilha vazia (Figura 4.6-a), as execuções das instruções `exec(Explicações_Lâmpada_1_Apagada)` e `exec(Explicações_Lâmpada_2_Apagada)` produzem os efeitos ilustrados, em sequência, pela Figura 4.6-b e Figura 4.6-c. A instrução `dif` desempilha duas vezes e associa o que foi desempilhado às variáveis, digamos h_1 e h_2 , sendo

$h_1 = \{ \text{bateria_sem_carga}, \text{ fios_rompidos}, \text{ interruptor_1_desligado} \}$ e

$h_2 = \{ \text{bateria_sem_carga}, \text{ fios_rompidos}, \text{ interruptor_1_desligado}, \text{ interruptor_2_desligado} \}$.

O resultado da subtração de h_2 por h_1 ($h_2 - h_1$) é empilhado (Figura 4.5-d). A instrução `exec(Explicações_Lâmpada_2_Apagada)` é executada e o resultado é empilhado (Figura 4.5-e). Por fim, a instrução `inter` desempilha duas vezes e associa o que foi desempilhado às variáveis, digamos h_1 e h_2 , sendo

$h_1 = \{ \text{bateria_sem_carga}, \text{ fios_rompidos}, \text{ interruptor_1_desligado} \}$ e

$h_2 = \{ \text{interruptor_2_desligado} \}$.

O resultado da intersecção entre h_1 e h_2 é empilhado (Figura 4.5-f). O que está no topo da pilha da Figura 4.5-f, i.e, o conjunto vazio, é o resultado da execução do programa objeto.

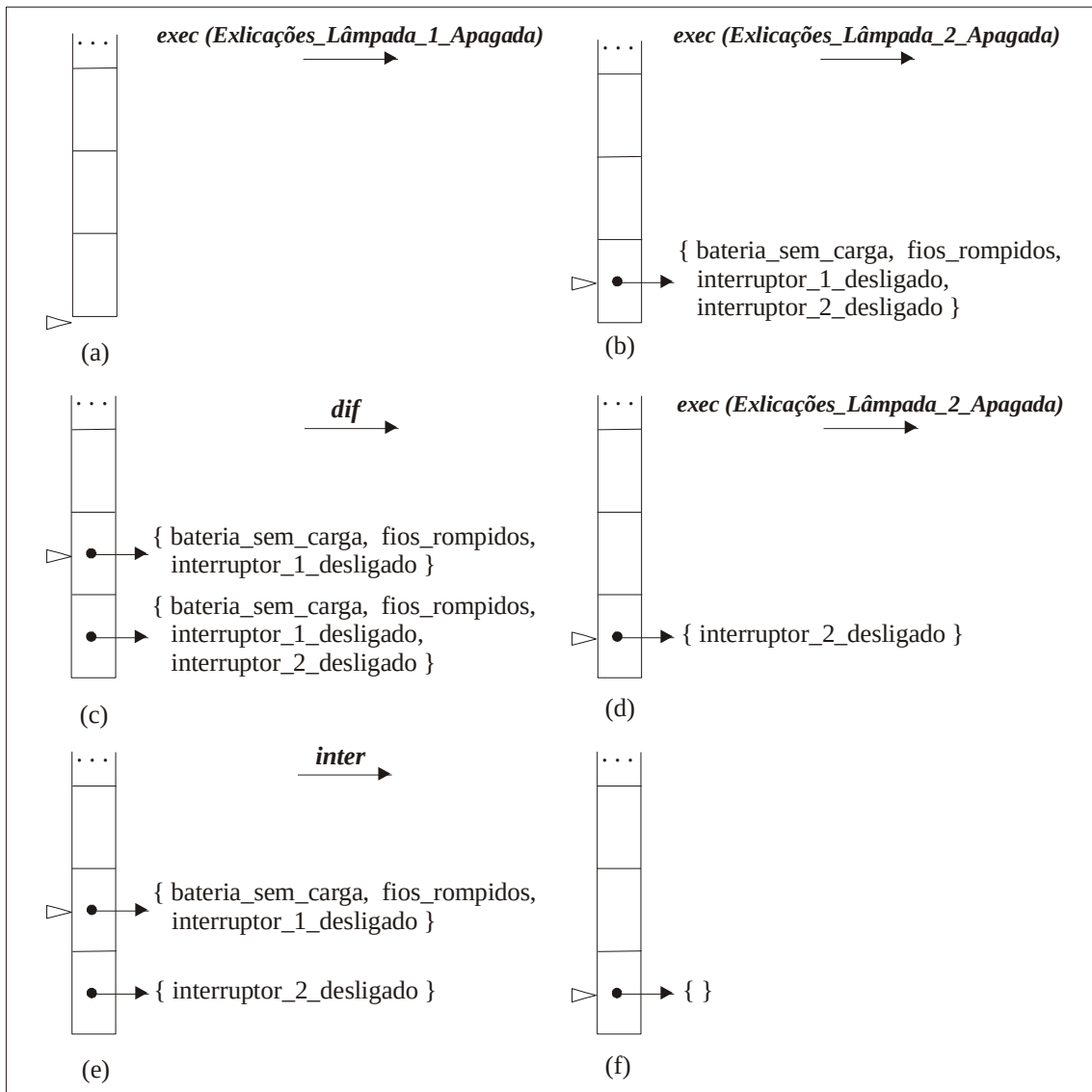


Figura 4.6. Execução do programa objeto da Figura 4.4-D pelo Autômato com Pilha Abd1.

4.3 – Ambiente Integrado de Desenvolvimento e Execução de Programas escritos em Abd1 (AIDEP-Abd1)

O Ambiente Integrado de Desenvolvimento e Execução de Programas escritos em Abd1 (AIDEP-Abd1) está disponível para *download* no site “<http://abd1.gear.host/>”. Implementado para plataforma *Windows* em ambiente *.NET, framework 4.5*, foi desenvolvido na linguagem de programação *C#*.

AIDEP-Abd1 é um facilitador para a escrita e execução de programas em Abd1 e está dividido em dois ambientes: edição e execução.

No ambiente de edição, Figura 4.7, é possível escrever programas na linguagem Abd1 por meio de seu editor de texto. Os botões localizados na parte superior da tela são atalhos que inserem *templates* dos comandos *Theory*, *Conditions*, *Facts*, *AbductiveReasoning* e *Options* além dos símbolos lógicos “ $\neg$ ”, “ $\wedge$ ”, “ $\vee$ ”, “ $\rightarrow$ ”, “ $<$ ”, “ $>$ ”, “ $\cup$ ”, “ $\cap$ ” e “ $-$ ” diretamente no editor de textos na posição do cursor. Estes botões de atalhos também funcionam no ambiente de execução da mesma forma.

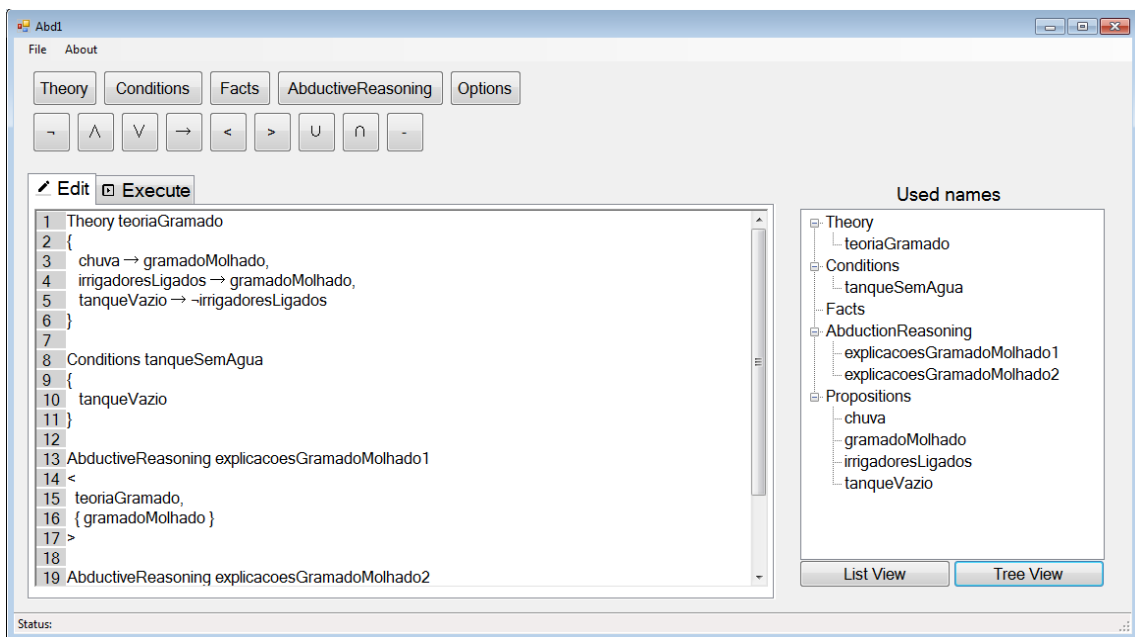


Figura 4.7. Interface do AIDEP-Abd1 no ambiente de edição.

No ambiente de execução, Figura 4.8, executamos os raciocínios abduativos (*AbductiveReasoning*) que foram escritos no editor ou criamos uma questão diretamente na caixa de texto *Question*.

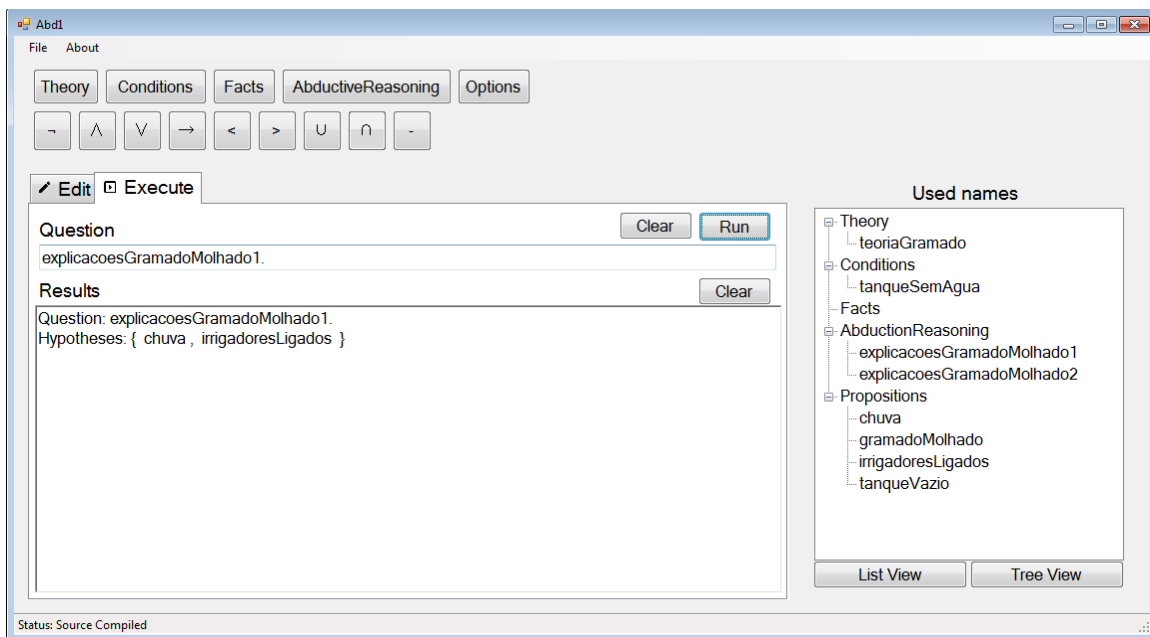


Figura 4.8. Interface do AIDEP-Abd1 no ambiente de execução.

Também existe um recurso compartilhado chamado *Used names* com dois modos de visualização: *List View* e *Tree View*. Este recurso é uma caixa de visualização que é atualizada ao clicar no botão *List View*, *Tree View* ou ao executar uma questão. O objetivo é permitir visualizar todas as proposições, teorias, condições, fatos e raciocínios abdutivos já escritos no editor e está disponível nos ambientes de edição e no ambiente de execução. Um duplo clique em qualquer um dos nomes disponíveis nesta caixa, o mesmo é copiado para a posição onde está o cursor no ambiente em que está ativo.

Além dos ambientes descritos, na parte superior há uma barra de menu. No menu *File*, existe o submenu *New*, *Open*, *Save*, *Save As* e *Exit*. É possível iniciar programas novos, abrir e salvar programas já salvos, salvar programas já salvos com outro nome e sair do AIDEP-Abd1. No menu *About*, há informações sobre as pessoas que participaram da implementação da ferramenta e e-mail para contato. Também há uma barra de *status* localizada na parte inferior onde é exibido mensagens de erros, compilação e execução.

Capítulo 5

Usabilidade da Linguagem Abd1

Este capítulo descreve uma pesquisa experimental de usabilidade da Linguagem Abd1, conduzida em laboratório, com 16 participantes. O experimento é sobre a escrita de programas em Abd1 utilizando o Ambiente Integrado de Desenvolvimento e Execução de Programas (AIDEP-Abd1) descrito no Capítulo 4. O foco desta pesquisa de usabilidade é a Linguagem Abd1 e não o ambiente AIDEP-Abd1, embora a usabilidade deste último afeta a usabilidade da primeira. A Seção 5.1 esclarece que tipo de avaliação de usabilidade é conduzida neste trabalho e como ela se diferencia de outros métodos clássicos de inspeção de usabilidade. A Seção 5.2 declara os materiais e os métodos empregados. A Seção 5.3 apresenta os resultados da pesquisa de usabilidade da Linguagem Abd1 e a Seção 5.4 discute estes resultados. Por fim, a Seção 5.5 apresenta as considerações finais deste capítulo.

5.1 – Pesquisa experimental de usabilidade

Existem vários tipos de avaliação de usabilidade, assim é preciso descrever sobre estes diferentes tipos para que se possa tornar claro o que é proposto neste trabalho como método de avaliação da usabilidade da Linguagem Abd1. Sobre estes diferentes tipos, Lazar, Feng & Hochheiser (2010) distinguem três categorias de avaliação de usabilidade: avaliação baseada em conhecimento de especialistas, avaliação automatizada e avaliação baseada em usuários.

Avaliações baseadas em conhecimento de especialistas são essencialmente inspeções feitas por especialistas e normalmente envolvem técnicas amplamente difundidas pela área de Interação Humano-Computador conhecidas como revisão heurística, inspeção de consistência ou percurso cognitivo.

Avaliações automatizadas usam software para inspecionar o uso de um sistema e, de forma geral, geram um relatório que, posteriormente, pode ser analisado por um especialista em usabilidade ou um pesquisador.

Avaliações baseadas em usuários envolve a observação de usuários realizando um conjunto de tarefas. Os métodos utilizados para realizar uma avaliação baseada em usuários são semelhantes aos métodos da área de pesquisa experimental (Lazar, Feng & Hochheiser, 2010). Este trabalho faz uso de pesquisa experimental em laboratório (Mackenzie, 2013) para avaliar a usabilidade da Linguagem Abd1.

A pesquisa experimental de usabilidade da Linguagem Abd1 aqui descrita foi realizada no mês de janeiro de 2016 em laboratório da Faculdade Campo Limpo Paulista. Os participantes utilizam o Ambiente Integrado de Desenvolvimento e Execução de Programas (AIDEP-Abd1, Capítulo 4) em tarefas de programação empregando a Linguagem Abd1. A pesquisa objetiva avaliar somente a Linguagem Abd1 no que se refere a dois aspectos de usabilidade:

- Facilidade de aprendizagem e de escrita de programas; e
- Satisfação subjetiva dos participantes quanto ao uso da linguagem para escrever programas que fazem raciocínio abdutivo.

A métrica para determinação da facilidade/dificuldade de aprendizagem e escrita de programas é dada pela quantidade de exercícios (programas escritos em Linguagem

Abd1) concluídos e corretos. A satisfação subjetiva é medida por respostas dadas pelos participantes em formulário.

Não é objetivo desta pesquisa avaliar a usabilidade da interface do ambiente AIDEP-Abd1, embora se saiba que a interface AIDEP-Abd1 comprometa de forma positiva e negativa a experiência de desenvolver programas usando a Linguagem Abd1. Em um trabalho futuro, pretende-se avaliar a usabilidade da interface do ambiente AIDEP-Abd1.

5.2 – Materiais e métodos

Esta pesquisa envolve o desenvolvimento, pelos participantes, de programas escritos em Linguagem Abd1 e é constituída por quatro fases. Sinteticamente, estas fases objetivam:

- Fase 1: delineamento do perfil dos participantes;
- Fase 2: rápido treinamento dos participantes no desenvolvimento de programas em Linguagem Abd1, com a assistência do pesquisador;
- Fase 3: escrita de programas em Linguagem Abd1 sem a ajuda do pesquisador;
- Fase 4: verificação da satisfação subjetiva dos participantes em relação à experiência de usar a Linguagem Abd1 para desenvolver programas para fazer raciocínio abdutivo.

Um formulário (Apêndice IV), constituído por quatro partes, foi desenvolvido para assistir a cada uma das fases da pesquisa experimental de usabilidade, sendo que a Parte 1 corresponde à Fase 1, a Parte 2 corresponde à Fase 2 e, assim, sucessivamente. Esta seção descreve os materiais e métodos empregados na pesquisa experimental de usabilidade da Linguagem Abd1. Ela inicia descrevendo os sujeitos da pesquisa para, em seguida, descrever o Formulário projetado e o procedimento experimental planejado.

Sujeitos

Um total de 16 participantes (Tabela 5.1) com curso superior completo, predominantemente da área relacionada a Computação (76%) (Tabela 5.2), atuam no experimento. Os participantes são estudantes do primeiro trimestre do curso de um mestrado em Ciência da Computação, portanto, todos possuem curso de graduação

completo, a maioria é sexo masculino (94%) e têm idade variando entre 22 e 62 anos, sendo 38 anos a média e também a mediana das idades.

Tabela 5.1. Gênero e idade dos participantes.

Gênero, <i>n</i> (%)		Idade, em anos		
Feminino	Masculino	Intervalo	Média	Mediana
1 (6)	15 (94)	22 - 62	38	38

Tabela 5.2. Curso de graduação concluído pelo grupo de participantes.

Curso de graduação	<i>n</i> (%)
Sistemas de Informação	5 (31)
Ciência da Computação	2 (13)
Tecnologia em Informática	2 (13)
Processamento de Dados	2 (13)
Redes de Computadores	1 (6)
Física	1 (6)
Matemática	1 (6)
Engenharia Elétrica	1 (6)
Gestão de Tecnologia da Informação	1 (6)

Os participantes declaram, por meio de formulário, os seus conhecimentos sobre Lógica e sobre Programação. A Tabela 5.3 sintetiza as respostas apresentadas. A maior parte dos participantes informa que já escreveu programas computacionais (94%) e sente-se capaz de programar usando alguma linguagem de programação (88%). Por outro lado, a maioria declara que nunca estudou Prolog (88%) e não se sente capaz de programar em Prolog (94%). Um pouco mais da metade dos participantes declara que nunca estudou Lógica Proposicional (56%).

Formulário

Desenvolveu-se um formulário para assistir a todas as fases da sessão experimental da pesquisa. O formulário foi validado previamente em um experimento piloto que contou com a participação de duas pessoas. Pequenos problemas foram removidos e o

formulário foi melhorado. O Formulário está integralmente descrito no Apêndice IV e esta seção discute o seu conteúdo.

Tabela 5.3. Conhecimento sobre Lógica e sobre Programação declarado pelos participantes.

Conhecimento	Sim, <i>n</i> (%)	Não, <i>n</i> (%)
Escreveu algum programa de computador	15 (94)	1 (6)
Sente-se capaz de programar em alguma linguagem de programação	14 (88)	2 (12)
Estudou Lógica Proposicional alguma vez	7 (44)	9 (56)
Estudou Prolog alguma vez	2 (12)	14 (88)
Sente-se capaz de programar em Prolog	1 (6)	15 (94)
Escreve programas em Prolog atualmente	1 (6)	15 (94)

O Formulário é constituído por quatro partes, cada uma delas projetada para assistir a uma fase específica do procedimento experimental que será descrito na próxima seção. O formulário não solicita a identificação pessoal do participante; apenas um código numérico é atribuído às partes do formulário para que se possa agrupar correspondentemente as respostas dadas pelos participantes.

A Parte 1 explica ao participante o propósito do experimento e permite que ele declare o seu perfil. O perfil está sendo caracterizado pelo sexo, idade, curso de graduação realizado pelo participante e por uma série de perguntas que têm por objetivo permitir que ele declare o quanto acha que possui conhecimentos sobre Lógica e sobre Programação.

A Parte 2 apresenta a Linguagem Abd1 por meio de exemplos de programas escritos nesta linguagem¹⁴. Pretende-se com a Parte 2 um treinamento na Linguagem Abd1 bem como que o material desta parte possa servir de referência para o participante durante os exercícios propostos na Parte 3. Três exercícios com nível crescente de complexidade são dados como exemplos (Apêndice IV). O Exemplo I aborda a declaração de teoria e de raciocínio abduativo. O Exemplo II adiciona ao Exemplo I a declaração de condições e a possibilidade de se declarar vários raciocínios abduativos. O Exemplo III aborda, de forma integrada, a declaração de mais do que uma teoria, condições, fatos e raciocínios

¹⁴ Optou-se por uma introdução à Linguagem Abd1 por meio de exemplos em vez de uma exposição teórica-conceitual.

abdutivos. Em especial, o Exemplo III mostra como programar raciocínios utilizando duas ou mais teorias e condições, e como raciocínios mais complexos podem ser programados a partir de operações envolvendo raciocínios mais simples.

A Parte 3 contém exercícios (propostas de programas) para serem desenvolvidos pelos participantes. É principalmente por meio da análise dos resultados destes exercícios que os pesquisadores poderão inferir problemas de usabilidade e o quanto a Linguagem Abd1 oferece facilidades ou dificuldades para programar raciocínios abdutivos. Dois exercícios com nível crescente de complexidade são propostos (Apêndice IV). O Exercício I propõe uma modificação do programa apresentado no Exemplo III da Parte 2. O exercício propõe três tarefas para serem realizadas em sequência. O Exercício II solicita a escrita, desde o “zero”, de um programa para modelar uma pequena interpretação da teoria econômica da oferta e da demanda. Os exercícios propõem cinco tarefas para serem realizadas em sequência. Para responder às tarefas propostas o participante deve programar, pelo menos, uma teoria, uma condição, dois raciocínios abdutivos e um fato. Opcionalmente, o participante pode fazer uso do recurso de compor um raciocínio a partir de raciocínios mais simples.

A Parte 4 possibilita que o participante declare a sua satisfação subjetiva em relação à experiência de usar a Linguagem Abd1 para desenvolver programas para raciocínio abductivo (Apêndice IV). Adicionalmente, esta parte possibilita que o participante atribua notas para o ambiente AIDEP-Abd1 como forma de sinalizar aos pesquisadores se falhas neste software afetam demasiadamente a pesquisa de usabilidade da Linguagem Abd1, o que, considerando uma situação hipotética de muitos efeitos negativos, pode invalidar a análise dos resultados da usabilidade da Linguagem Abd1.

Procedimento experimental

Um laboratório com 20 estações de trabalho, etiquetadas com um código numérico de 01 a 20, tendo cada uma um computador equipado com processador Intel i3, 3.3 GHz, 4 GB de memória RAM, executando Windows 7 é preparado para atender ao experimento. O ambiente AIDEP-Abd1 é instalado previamente em cada computador e cada um dos 16 participantes toma lugar em uma estação de trabalho e são identificados pelo código da etiqueta. A sessão experimental é composta por quatro fases, detalhadas a seguir.

A Fase 1, programada para ser realizada em 5 minutos, inicia com o acolhimento, cumprimento e agradecimento aos participantes pela contribuição voluntária e abnegada de recompensas financeiras. Com o auxílio de um projetor multimídia, o pesquisador expõe o propósito do experimento, distribui a Parte 1 do Formulário (Apêndice IV) e solicita que os participantes respondam às questões apresentadas sobre o perfil dos participantes. O processo de distribuição da Parte 1 do formulário leva em conta o código da estação de trabalho, fazendo com que o código atribuído à Parte 1 do formulário coincida com o código da estação de trabalho. Após os participantes responderem às questões da Parte 1 do Formulário, o pesquisador recolhe os formulários respondidos.

A Fase 2, programada para ser realizada em 30 minutos, objetiva o treinamento dos participantes na programação de raciocínios abduativos com a Linguagem Abd1. A estratégia pedagógica é a da aprendizagem através de exemplos. Assim, para cada um dos três exemplos previstos na Parte 2 do Formulário (Apêndice IV), o pesquisador faz o seguinte: (1) expõe oralmente o exemplo usando um projetor multimídia e responde eventuais questões dos participantes; (2) solicita que cada participante utilize o ambiente AIDEP-Abd1 e obtenha o texto do programa do exemplo, gravado previamente no diretório “Exemplos”, para carregar programas e executar raciocínios abduativos. Nesta fase, o pesquisador interage com os participantes para responder eventuais dúvidas relativas à Linguagem Abd1 e também relativas ao uso do ambiente AIDEP-Abd1. Após a exposição e o estudo dos três exemplos, o pesquisador distribui a Parte 2 do Formulário. O pesquisador explica que esta parte do formulário contém todos exemplos expostos oralmente e que pode ser utilizado como referência para realização das atividades da Fase 3.

A Fase 3, programada para ser realizada em 30 minutos, objetiva a escrita de programas usando a Linguagem Abd1. O pesquisador distribui a Parte 3 do Formulário, observando a correspondência com o código da estação de trabalho. O pesquisador comunica aos participantes que eles terão 30 minutos para responder às questões propostas e que deverão fazer isto sem comunicação entre si e sem comunicação com o pesquisador. Após 30 minutos, o pesquisador recolhe as respostas dadas pelos participantes aos exercícios da Parte 3 do Formulário.

A Fase 4, programada para ser realizada em 5 minutos, objetiva medir a satisfação subjetiva dos participantes em utilizar a Linguagem Abd1 para programar raciocínios abduativos. O pesquisador distribui a Parte 4 do Formulário (Apêndice IV), atentando para a devida correspondência com os códigos das estações de trabalho, e solicita que os participantes respondam às questões propostas. Ao final dos 5 minutos, o pesquisador recolhe as respostas.

A sessão experimental termina com um *debriefing*, com duração de 5 minutos, no qual o pesquisador apresenta a solução dos exercícios propostos (programas) na Fase 3. Os participantes são convidados a, opcionalmente, comentarem suas soluções.

Finalmente, os programas gravados pelos participantes no diretório “Exercícios” são recolhidos pelo pesquisador, identificando-os pelo código da estação trabalho.

As respostas dos participantes às questões apresentadas na Parte 3 do Formulário são analisadas independentemente por dois pesquisadores, que atribuem parecer sobre os exercícios (programas) I e II estarem completos e totalmente corretos. As divergências de parecer dadas pelos pesquisadores são discutidas até o estabelecimento de um parecer consensual.

5.3 – Resultados

As respostas dos participantes para a Parte 1 do Formulário são digitalizadas e a síntese destes resultados é apresentada nas tabelas 5.1, 5.2 e 5.3 que descrevem o perfil do participante. Estas tabelas estão apresentadas na Seção 5.2.

A Tabela 5.4 descreve analiticamente o desempenho dos participantes nas respostas aos exercícios propostos na Parte 3 do Formulário e a Tabela 5.5 sintetiza isto.

Tabela 5.4. Desempenho dos participantes nas respostas aos exercícios propostos na Parte 3 do Formulário.

Participante	Exercício I	Exercício II
1	Correto	Correto
2	Incorreto	Incorreto
5	Correto	Correto
6	Correto	Correto
7	Correto	Correto

8	Correto	Sem resposta
9	Correto	Incorreto
10	Correto	Correto
11	Incorreto	Incorreto
12	Incorreto	Incorreto
13	Correto	Correto
15	Correto	Sem resposta
17	Correto	Correto
18	Correto	Sem resposta
19	Correto	Correto
20	Incorreto	Sem resposta

Tabela 5.5. Síntese do desempenho dos participantes nas respostas aos exercícios propostos na Parte 3 do Formulário.

Atividade	Respostas corretas	Respostas incorretas	Sem resposta
Exercício I	12	4	0
Exercício II	8	4	4
Total	20	8	4

As respostas dos participantes às questões apresentadas na Parte 4 do Formulário estão apresentadas na Tabela 5.6. Estas respostas correspondem à média das notas atribuídas pelo grupo de participantes às questões apresentadas.

Tabela 5.6. Média e desvio padrão (DP) das notas atribuídas pelos participantes às questões propostas na Parte 4 do Formulário (satisfação subjetiva). A escala tem o seguinte significado: 1 (muito difícil), 2 (difícil), 3 (regular), 4 (fácil) e 5 (muito fácil).

Questão	Média (1..5)	DP
Nível de facilidade em compreender a exposição da Linguagem Abd1.	3.6	0.8
Nível de facilidade dos exercícios feitos na Parte 3	3.4	1.0
Percepção sobre a facilidade de usar a Linguagem Abd1 para modelar fenômenos	3.5	0.7
Quanto o software (não a Linguagem Abd1) é fácil para editar, compilar, exibir mensagens de erro e executar raciocínios	3.2	1.1
Média das médias	3.4	-

5.4 – Discussão dos resultados

Dois aspectos de usabilidade são avaliados nesta pesquisa de usabilidade: (1) a facilidade de aprendizagem e de escrita de programas e (2) a satisfação subjetiva dos participantes quanto ao uso da linguagem Abd1.

Facilidade de aprendizagem e de escrita de programas

Os exercícios propostos foram focados no uso, pelos participantes, dos conceitos presentes na linguagem. Os exercícios não tiveram foco nas habilidades necessárias para análise de fenômenos reais e posterior modelagem destes fenômenos. Os exercícios propostos refletem o que foi planejado, requerendo o uso e a compreensão de quase a totalidade dos conceitos envolvidos na linguagem (descrição de teoria, condições, fatos, raciocínios abduativos e sentenças). Dependendo da solução pensada, o participante tem a opção de usar ou não o recurso de composição de raciocínios mais complexos a partir de raciocínios mais simples. Os exercícios não exigem do participante o controle da utilização dos conceitos de força explanatória mínima e complexidade máxima de uma hipótese; valores padrão são suficientes.

Especificamente, o Exercício I requer do participante (1) saber interpretar as teorias, condições, fatos e raciocínios abduativos previamente definidos no texto de um programa e (2) a escrita de sentenças para modificar uma teoria. Especificamente, o Exercício II exige do participante saber escrever teorias, fatos, condições e sentenças.

Para avaliar quantitativamente a facilidade de aprendizagem e escrita de programas em Abd1, propõe-se:

- Como métrica, a quantidade de acertos dos participantes aos exercícios propostos.
- Uma escala para medir diferentes níveis de facilidade de aprendizagem e escrita de programas. A escala compreende os níveis “Muito fraca”, “Fraca”, “Regular”, “Boa” e “Muito boa” para quantidade de acertos variando em torno de, respectivamente, 5%, 25%, 50%, 75% e 95%.

Considerando a quantidade de acertos aos exercícios propostos na Fase 3 do experimento (20 acertos em 32 possíveis, ocorreram 12 não acertos, Tabela 5.5),

realizou-se um Teste-G (Sokal & Rohlf, 1994), com significância α igual a 1%, para cada nível de facilidade de aprendizagem e escrita de programas na escala estabelecida.

A Tabela 5.7 descreve os Valores-G e os respectivos Valores-p calculados. Observa-se na tabela que, para os níveis “Muito fraca”, “Frac” e “Muito boa” da escala, o Valor-p é menor do que a significância α estabelecida (células da tabela destacadas em fundo de cor cinza), o que conduz a rejeitar a hipótese de que a quantidade de acertos do grupo estudado adere a estes níveis da escala. Ou seja, os Testes G realizados sugerem que a quantidade de acertos do grupo estudado possui diferença estatisticamente significativa com os níveis “Muito fraca”, “Frac” e “Muito boa” da escala.

Tabela 5.7. Valor-G e respectivo Valor-p para diferentes níveis de facilidade de aprendizagem e escrita de programas, calculados tendo como métrica a quantidade de acertos (20 em 32 possíveis) nos exercícios propostos na Fase 3 do experimento.

	Escala de níveis facilidade de aprendizagem e escrita de programas				
	Muito fraca (5% acertos)	Frac (25% acertos)	Regular (50% acertos)	Boa (75% acertos)	Muito boa (95% acertos)
Valor-G	78.72	20.02	2.02	2.44	21.61
Valor-p	< 0.0001	< 0.0001	0.21	0.17	< 0.0001

Pelos Testes G realizados **não** é possível concluir que a quantidade de acertos do grupo estudado **não** adere ao nível “Regular” e **não** adere ao nível “Boa” (Valores-p maiores do que a significância α , células da tabela com fundo de cor branca). Ou seja, os testes estatísticos realizados sugerem que a quantidade de acertos do grupo estudado adere ao nível “Regular” ou ao nível “Boa”, não sendo possível concluir em qual destes dois níveis da escala a quantidade de acertos se situa.

Este resultado obtido tem influências negativas e positivas.

Entre as influências positivas destacam-se os *templates* da linguagem disponíveis no ambiente AIDEP-Abd1 por meio do acionamento de botões identificados com o nome da estrutura linguística. Relatos dos participantes, na forma escrita e verbal, descrevem que, na edição de programas, o recurso de poder obter *templates* sobre como escrever teorias, fatos, condições etc. ajudam a lembrar a sintaxe da linguagem e isto facilita o seu uso.

As influências negativas revelam (1) problemas de usabilidade da Linguagem Abd1, (2) problemas de usabilidade do ambiente AIDEP-Abd1 (não era o foco do experimento) e (3) efeitos colaterais (*bias*) não previstos e, portanto, não controlados pela pesquisa experimental realizada.

Um total de 12 respostas não podem ser consideradas corretas, mas entre elas 8 apresentam texto incompleto que é útil na avaliação de problemas.

Problemas de usabilidade identificado na Linguagem Abd1 são descritos a seguir. A análise de parte das respostas incorretas, porém com texto incompleto, sugere que:

- A escrita de sentenças em Lógica Proposicional com os conectivos convencionais (e.g., $\neg$, $\wedge$, $\vee$, $\rightarrow$) criou dificuldades para alguns participantes. Embora a Linguagem Abd1 admita o uso de conectivos escritos em inglês (e.g., *not*, *and*, *or*), o treinamento se limitou aos conectivos convencionais e, embora o grupo de estudo seja predominantemente da área de Computação (76%), a maioria dos participantes (56%) declara que nunca estudou Lógica Proposicional.
- Algumas sentenças não estavam escritas na Forma HF (Seção 3.1). A Forma HF é difícil de ser memorizada e treinada. O problema de usabilidade neste caso nem se deve ao fato de que a Forma HF seja difícil de ser memorizada, mas ao fato de que ela nem foi mencionada na fase de treinamento.

Em relação ao ambiente AIDEP-Abd1, a tentativa de compilação dos textos incompletos das respostas considerados errados sugere, em pelo menos quatro casos, que o compilador não ajudou ao participante a descobrir erros léxicos e sintáticos, mas ao contrário, os impedia de continuar o trabalho. Esta hipótese é reforçada pelas seguintes declarações feitas por participantes:

- Participante 9: *Sugiro melhorar as mensagens de erro.*
- Relato verbal de participante: *Vi que o colega do lado não conseguia identificar que o erro era a falta de uma vírgula e o compilador não pegou isto para ele.*
- Relato verbal de participante: *O sistema não emitia mensagens de erro.*
- Participante 13: *Senti falta de debug.* Embora o participante refira-se a *debug* é possível supor que um bom funcionamento das mensagens de erro emitidas pelo compilador pudesse mudar a sua percepção.

- Participante 18: *Quando ocorreu erro de vetor fora dos limites, após algumas tentativas o abd1 fechou sozinho.* Estes erros indicam falha da aplicação, mas independentemente da falha, não deveriam ser emitidos para o usuário e sim controlados pelo mecanismo de controle de exceções implementado.

O tempo destinado na fase de treinamento na Linguagem Abd1 e na fase de desenvolvimento dos exercícios, 30 minutos para cada fase, é o efeito colateral de método da pesquisa experimental mais notavelmente identificado. Isto foi relatado verbalmente e também na forma escrita por muitos participantes:

- Participante 11: *Esses exercícios foram legais mas precisava de mais tempo para a realização dos mesmos.*
- Participante 12: *O tempo de treinamento foi insuficiente para desenvolver os exercícios, que também tiveram tempo curto.*
- Participante 17: *... Um pouco mais de tempo para os exercícios.*
- Participante 20: *... meu nível de conhecimento da linguagem Abd1 é nenhum. Precisaria de (mais) explicações sobre a linguagem abd1.*

Satisfação subjetiva dos participantes

A média das notas atribuídas pelos participantes a cada uma das quatro questões de satisfação subjetiva, apresentadas na Fase 4 do experimento e relatadas na Tabela 5.6, está entre o “Regular” (3) e o “Fácil” (4).

Uma pergunta de avaliação que se coloca é: uma ou mais destas médias desvia-se significativamente das outras? Um desvio significativo de uma média é um indicativo, neste caso, de que o grupo de estudo deu respostas a uma questão de maneira significativamente diferente das demais e, assim, tal questão merece uma análise especial. Realizou-se um Teste de Análise de Variância (ANOVA (Boslaugh, 2012)), ao nível de significância α igual a 1%, sobre as respostas dadas pelos participantes a cada uma das questões apresentadas. Para o grupo estudado, o Teste ANOVA sugere que não existe diferença significativa nas respostas dadas às questões apresentadas ($F(3,60) = 0.49$, Valor-p = 0.69). Ou seja, não há diferença significativa entre as médias de notas das questões, embora as médias sejam diferentes e, assim, nenhuma questão se destaca das demais em relação à satisfação subjetiva do grupo de estudo.

De forma específica:

- As médias atribuídas às questões “Nível de facilidade em compreender a exposição da Linguagem Abd1” e “Percepção sobre a facilidade de usar a Linguagem Abd1 para modelar fenômenos” são, respectivamente, iguais a 3.6 e 3.5, ou seja, indicam algo entre o nível “Regular” e o nível “Fácil” da escala de satisfação subjetiva utilizada. Isto é coerente com o desempenho “Regular” ou “Bom” (cf. subseção anterior) medido pela quantidade de respostas corretas apresentadas pelo grupo em estudo aos exercícios realizados.
- A média 3.4 atribuída à questão “Nível de facilidade dos exercícios feitos na Parte 3” indica que o grupo considerou que os exercícios foram equilibrados no sentido de não serem muito fáceis e nem muito difíceis, ambas as situações que sugeririam prejuízo para a pesquisa.
- A média 3.2 atribuída à questão “Quanto o software (não a Linguagem Abd1) é fácil para editar, compilar, exibir mensagens de erro e executar raciocínios” é um indicativo de que o grupo em estudo percebe que o ambiente AIDEP-Abd1 afeta positivamente, mas também negativamente a experiência de escrever programas na Linguagem Abd1. Este resultado reflete as observações discutidas na subseção anterior relativo aos impactos positivos e negativos do ambiente AIDEP-Abd1 na escrita de programas Abd1.

Por fim, manifestações verbais e escritas sugerem que o grupo em estudo reconhece valor na Linguagem Abd1, embora identifique problemas com o ambiente AIDEP-Abd1:

- Participante 5: *Muito bom o trabalho e pesquisa. Parabéns. O SW precisa de algum refinamento, mas está muito bom. ...* . Suspeita-se que aqui o participante quis se referir à Linguagem Abd1 quando usa o termo “trabalho” e que quis se referir ao ambiente AIDEP-Abd1 quando usa o termo “SW”.
- Participante 6: *A linguagem é interessante, mas a IDE ainda precisa melhorar.*
- Participante 10: *Parabéns pelo desenvolvimento da ferramenta. Eu nunca tinha tido contato com essa linguagem e facilmente consegui implementar os exercícios propostos. Parabéns novamente.*
- Participante 13: *Estou disposto a colaborar com o projeto.*

5.5 – Considerações finais

Preferiu-se que as conclusões desta pesquisa experimental de usabilidade da Linguagem Abd1 fossem descritas juntamente com outras discussões e conclusões no capítulo final desta dissertação. A Seção 6.2 do Capítulo 6 é dedicada a isto.

Capítulo 6

Discussão e Conclusão

Este capítulo discute e apresenta as conclusões deste trabalho. A Seção 6.1 inicia com uma síntese sobre as principais contribuições deste trabalho e, posteriormente, discute detalhadamente como a Linguagem Abd1 contribui para superar as dificuldades de programação identificadas e descritas na Seção 2.3. Uma discussão mais geral sobre a usabilidade da Linguagem Abd1 do que a apresentada no Capítulo 5 é dada na Seção 6.2. A seção também apresenta conclusões sobre a usabilidade da linguagem. Cenários de possíveis usos da Linguagem Abd1 são abordados na Seção 6.3. Por fim, as seções 6.4 e 6.5 discutem, respectivamente, as limitações deste trabalho e propostas de trabalhos futuros para tentar superar as limitações identificadas.

6.1 – Contribuições deste trabalho

Este trabalho tem como foco a proposta e a implementação de uma linguagem para descrição de raciocínios abduktivo e é inovador em vários aspectos, mas especialmente porque:

- Propõe uma linguagem baseada em uma estrutura de raciocínio, a TCHF, que permite organizar o texto de programas com conceitos nativos do domínio da abdução (teoria, hipóteses, fatos e condições). Além de serem conceitos nativos ao domínio de interesse, a estrutura proposta facilita: (1) a análise de sistemas e a modelagem de raciocínios nestes sistemas, porque atua como um modelo conceitual para entendê-los; (2) a segmentação de programas grandes em unidades compostas por diversas teorias, condições, fatos e raciocínios. Isto, por um lado, estrutura (no sentido de organizar) o programa e, por outro lado, permite a reutilização de código.
- Utiliza apenas o conceito de conjunto para modelar raciocínios abduktivos. Assim, teorias, fatos e condições são todos eles tratados como conjuntos e isto traz simplicidade conceitual para formulação e escrita de raciocínios. O resultado de um raciocínio (um conjunto de hipóteses) também é visto como um conjunto. Isto tem valor porque mantém a simplicidade conceitual, mas sobretudo porque possibilita a descrição de raciocínios mais complexos a partir de operações (operações sobre conjuntos) envolvendo raciocínios mais simples. Este aspecto é singular na pesquisa sobre raciocínios automatizados por computador e é uma das mais importantes inovações propostas neste trabalho.

Decorrente dos esforços de desenvolvimento da linguagem para descrição de raciocínios abduktivos, este trabalho tem como contribuições artefatos tecnológicos que, agora, estão disponíveis para uso:

- O compilador da Linguagem Abd1.
- O Autômato com Pilha Abd1, que executa programas Abd1 compilados.
- O software AIDEP-Abd1, Ambiente Integrado de Desenvolvimento e Execução de Programas escritos em Abd1, que reúne em uma só peça, o compilador, o autômato e facilidades de edição de programas escritos em Abd1.

Os elementos de *design* da Linguagem Abd1 foram motivados pelo objetivo de minimizar as dificuldades de programação de raciocínios abduativos identificadas nas abordagens existentes (Seção 2.3). A subseção seguinte descreve como a Linguagem Abd1 auxilia na superação destas dificuldades, revisitando-as, uma a uma e na mesma sequência em que foram apresentadas na Seção 2.3, só que agora com os argumentos sobre como a Linguagem Abd1 auxilia na superação delas.

Contribuições da Linguagem Abd1 para a superação das dificuldades de programação identificadas

A título de comparação, as abordagens existentes para abdução, tais como Prolog, CHR, ALP e ACLP (Seção 2.1 e Seção 2.2), são mencionadas nas argumentações apresentadas.

Auxílio para superar a dificuldade 1 (*linguagem com grande diferença sintática com a linguagem natural*¹⁵). As seguintes sentenças podem ser escritas na Linguagem Abd1:

$interruptor_1_desligado \rightarrow lâmpada_1_apagada \wedge lâmpada_2_apagada,$

$interruptor_1_desligado \rightarrow lâmpada_1_apagada \text{ and } lâmpada_2_apagada.$

Estas sentenças ainda estão longe da correspondente sentença em linguagem natural, “se interruptor 1 está desligado então a lâmpada 1 está apagada e a lâmpada 2 está apagada”. No entanto estas sentenças estão mais próximas da linguagem natural do que, por exemplo, a correspondente sentença em linguagem CHR

$lâmpada_1_apagada, lâmpada_2_apagada ==> true \mid interruptor_1_desligado.$

O resultado de um raciocínio abduativo em Abd1 é um conjunto de hipóteses, por exemplo,

$\{ interruptor_2_desligado \wedge bateria_sem_carga \wedge interruptor_1_ligado \},$

que significa “interruptor 2 desligado e bateria sem carga e interruptor 1 ligado”. Isto também é bem mais próximo da linguagem natural do que a correspondente resposta obtida em CHR executando no sistema SWI-Prolog:

¹⁵ Não é o objetivo deste trabalho que a descrição de raciocínios abduativos sejam feitas em linguagem natural. No entanto, observa-se que a proximidade linguística com a linguagem natural facilita a descrição de raciocínios abduativos.

interruptor_2_desligado

bateria_sem_carga

interruptor_1_ligado

true ;

false.

Auxílio para superar a dificuldade 2 (*não uso de terminologia do domínio dos raciocínios abdutivos*). O *design* da linguagem Abd1 é motivado pela estrutura de raciocínio TCHF (Teoria, Condições, Hipóteses e Fatos). Esta estrutura captura a essência linguística e conceitual daquilo que se entende ser o domínio dos raciocínios abdutivos. Programas em Abd1 são, portanto, organizados por meio de conceitos genuinamente pertencentes ao domínio dos raciocínios abdutivos. Isto não ocorre com outras abordagens (e.g., CHR, ALP e ACLP) que, como extensões linguísticas de sistemas que implementam por exemplo Prolog, têm que agregar seus conceitos ao já vasto conjunto de conceitos presentes nos tradicionais de sistemas de programação lógica: base de conhecimento, questão, objetivo, predicado abduível, restrição, banco de restrições, termo, átomo, *backtracking*, etc..

Auxílio para superar a dificuldade 3 (*mistura de paradigmas de programação*). Programar em Abd1 consiste apenas em declarar sentenças que expressam o conhecimento sobre um assunto. Tudo o que um programador deve dominar é o conhecimento e a técnica para declarar coisas sobre um assunto. O paradigma declarativo lógico puro é possível porque Abd1 é uma linguagem específica¹⁶ para descrição de raciocínios abdutivos e o problema que deve ser resolvido está implícito nas declarações e é um só: calcular hipóteses que respondam ao que está declarado. Isto, obviamente, não acontece com sistemas que servem a vários propósitos. Por exemplo, em CHR há a necessidade do programador lidar com declarações Prolog, mecanismo procedimental de derivação de regras CHR, execução sequencial de sentenças Prolog, execução com *backtracking*, recursão, *looping* etc.. Ou seja, tem que lidar com uma variedade de paradigmas de programação e isto torna a programação mais difícil.

¹⁶ Justifica-se o desenvolvimento de linguagens específicas quando as atividades que elas tratam ocorrem muito frequentemente. Este é o caso de raciocínio abduativo, porque há uma quantidade e variedade grande de fenômenos que podem ser formulados como problemas de abdução. Sem dúvida, este é o caso também da Linguagem SQL para definição e manipulação de dados compartimentados em tabelas relacionais.

Auxílio para superar a dificuldade 4 (*ausência de suporte específico para expressar raciocínios complexos a partir de raciocínios mais simples*). Um importante diferencial da linguagem Abd1 é a concepção de que o resultado de um raciocínio é um conjunto. Esta concepção permite que raciocínios complexos sejam formulados, de forma simples e elegante, por meio de operações sobre conjuntos (união, intersecção e diferença). Considerando o Exemplo 3.5, um raciocínio que busca as causas comuns aos fatos das lâmpadas 1 e 2 estarem apagadas pode ser formulado por meio da intersecção de dois raciocínios mais simples que objetivam explicar, um deles, por que a lâmpada 1 está apagada e, o outro, por que a lâmpada 2 está apagada:

$$\langle \text{Teoria_Não_Funcionamento}, \{ \text{lâmpada_1_apagada} \} \rangle \cap \langle \text{Teoria_Não_Funcionamento}, \{ \text{lâmpada_2_apagada} \} \rangle$$

As abordagens existentes não oferecem facilidades específicas para o desenvolvimento de raciocínios complexos a partir de raciocínios mais simples. De modo geral, para realizar tal coisa, o programador tem que criar estruturas de dados e desenvolver código específico para tratar os dados.

Auxílio para superar a dificuldade 5 (*segmentação de programas grandes alheia aos conceitos do domínio dos raciocínios abduativos*). Programas grandes precisam ser segmentados de alguma maneira para que não sejam difíceis de serem escritos, entendidos por terceiros e mantidos. Em Abd1 esta segmentação pode ocorrer de maneira descomplicada utilizando-se os próprios elementos nativos da linguagem¹⁷, i.e, teorias, condições, fatos e raciocínios. Por exemplo, em vez de desenvolver uma única e grande teoria sobre um circuito, várias teorias menores podem ser desenvolvidas para tratar diferentes aspectos de um circuito: teoria do funcionamento, teoria dos defeitos, teoria do aquecimento, teoria da vida útil, teoria do custo dos componentes etc.. Isto não ocorre tão facilmente e nativamente com muitas linguagens. Por exemplo, em CHR ou Prolog frequentemente programas formam um bloco “monolítico” constituído de dezenas ou centenas de regras e fatos. As facilidades de segmentação, quando existentes, são oferecidas pelo sistema que implementa estas linguagens (e.g, SWI-Prolog) por meio, por exemplo, de divisão dos textos dos programas em diversos arquivos denominados módulos.

¹⁷ Vale a seguinte analogia: os conhecimentos podem ser segmentados em teorias, condições, fatos e raciocínios, assim como um sistema pode ser segmentado em classes e objetos.

Auxílio para superar a dificuldade 6 (fraco suporte a necessidades expressivas recorrentes na modelagem de teorias para raciocínio abduutivo). Algumas coisas são frequentes na modelagem de raciocínios abduativos. Uma delas é a “circularidade lógica” entre proposições e a outra é o uso de negação. A linguagem Abd1 permite a escrita de sentenças que descrevem circularidade, como por exemplo:

$$\text{homicídio} \rightarrow \text{réu} \wedge \text{vítima} \wedge \text{réuMatouVítima},$$
$$\text{réu} \wedge \text{vítima} \wedge \text{réuMatouVítima} \rightarrow \text{homicídio}.$$

A inclusão de circularidade em um programa não se dá apenas em razão da circularidade ser algo inerente a um fenômeno que está sendo modelado. Permitir escrever com facilidade sentenças que contenham circularidade é importante também por razões ligadas à imprevisibilidade do uso do programa em um ambiente interativo no qual diversos raciocínios podem ser formulados. Por exemplo, no programa acima alguém pode querer fazer um raciocínio usando como fato { *homicídio* }, mas pode querer também fazer outro raciocínio usando como fatos { *réu*, *vítima*, *réuMatouVítima* }. Isto é relativamente simples no exemplo acima, mas pode acontecer também em um programa com centenas de sentenças com diversos ciclos entre as proposições. É possível administrar a circularidade entre sentenças escritas em CHR, Prolog ou ALP, mas isto não pode ser feito sem escrever uma boa quantidade de código adicional para prevenir que o programa não execute infinitamente.

6.2 – Usabilidade da Linguagem Abd1

Uma pesquisa experimental sobre a usabilidade da Linguagem Abd1 foi conduzida e os detalhes desta pesquisa estão descritos no Capítulo 5. Pesquisas experimentais são custosas e de modo geral, um projeto de pesquisa de mestrado não conta com orçamento para realizá-las. A pesquisa conduzida neste trabalho contou com a participação voluntária e sem remuneração dos participantes, de forma alguma pode ser considerado sem custo porque o tempo dos participantes, nestas condições, tem valor inestimável. Tendo por objetivo uma pesquisa experimental de baixo custo, optou-se por projetar um experimento de curta duração com o tempo de aproximadamente 1 hora.

Alunos de uma população de estudantes recém-ingressados em um programa de mestrado em Ciência da Computação foram os sujeitos da pesquisa experimental. Esta é

uma população de interesse deste trabalho, mas muitas outras populações de variados perfis também são interessantes de serem pesquisadas, considerando os potenciais utilizadores da linguagem investigada. Entre as populações de interesse, o fator que definiu a escolha da população estudada foi a conveniência da disponibilidade do grupo de estudantes que participou do experimento. Não foi utilizado um procedimento especial para amostrar os participantes, porque a quantidade de sujeitos do grupo que participou do experimento corresponde a quase a totalidade (80%) de indivíduos da população estudada.

Os resultados da pesquisa indicam que a usabilidade da Linguagem Abd1 é regular ou boa¹⁸ no que se refere à facilidade de aprendizagem e escrita de programas e que a satisfação subjetiva do grupo estudado, em relação à facilidade de utilizar a Linguagem, é algo que se situa entre o regular e o fácil¹⁹. Os resultados se referem, evidentemente, à população estudada e para as condições experimentais utilizadas (30 minutos de treinamento, 30 minutos de exercícios, todos os sujeitos participando simultaneamente do experimento em um laboratório de computação com várias estações de trabalho etc.). Estes resultados podem ser diferentes se as condições experimentais forem modificadas, por exemplo, pelo aumento do tempo de treinamento. Os resultados podem, também, serem diferentes para outras populações, por exemplo, uma população de estudantes de educação média interessados em utilizar a Linguagem Abd1 para modelar fenômenos que estão sendo discutidos na escola. A melhoria da usabilidade do ambiente AIDEP-Abd1 também poderia modificar os resultados, por exemplo, pela melhoria das mensagens de erro emitidas pelo Compilador, considerando os problemas de usabilidade identificados no experimento e que foram descritos no Capítulo 5.

A pesquisa realizada é a primeira pesquisa de usabilidade da Linguagem Abd1 e, por isto, não há como comparar os resultados obtidos com outros resultados. Como também não foi pesquisada a usabilidade de outras soluções para programação de raciocínio abduutivo (e.g., CHR e Prolog), nas mesmas condições experimentais da pesquisa de usabilidade conduzida, no momento não é possível estabelecer discussões comparativas. Pensa-se em fazer isto em um trabalho futuro.

¹⁸ Escala de cinco níveis compreendendo muito fraca, fraca, regular, boa, muito boa.

¹⁹ Escala de cinco níveis compreendendo muito difícil, difícil, regular, fácil e muito fácil.

6.3 – Possíveis usos da Linguagem Abd1

A Linguagem Abd1 é, antes de tudo, uma linguagem concebida com o propósito de pesquisa no campo dos raciocínios abduativos. Supõe-se que o seu principal uso é para pesquisa e, neste sentido, ela pode ser útil para uma variedade de estudos relacionados, por exemplo: (1) a uma melhor compreensão das necessidades expressivas de linguagens que se propõem a programar raciocínios abduativos; (2) à melhoria da eficiência da computação de raciocínios abduativos; (3) ao entendimento de fatores que influenciam a escolha de boas hipóteses.

No campo educacional o desenvolvimento de programas computacionais para criar modelos de fenômenos possui raízes nos trabalhos de Papert (1993) e, hoje em dia, tem se mostrado ferramenta didática bastante efetiva; Scratch (Marji, 2014) tem sido intensamente utilizado para modelagem de assuntos da agenda da educação fundamental e média enquanto Matlab (Gilat, 2014) é utilizado para modelagem em engenharia. Linguagens para modelagem com propósitos educacionais precisam ser fáceis de serem aprendidas e utilizadas, uma vez que um aprendiz deve concentrar seu esforço cognitivo no fenômeno que está estudado e não na escrita de programas em si. Considerando que o *design* da Linguagem Abd1 prioriza a facilidade de utilização, então um cenário possível de futura utilização dela pode ser a modelagem de fenômenos com propósitos educacionais. A Linguagem Abd1 e o ambiente AIDEP-Abd1 podem ser facilmente inseridos em um ambiente educacional construcionista (Papert, 1993) onde o aprendiz é motivado a se colocar em um ciclo no qual expressa a sua compreensão sobre um fenômeno, executa o modelo desenvolvido, interpreta os resultados da execução e, dependendo dos resultados, reformula o modelo desenvolvido, à medida em que também reconstrói e cresce no entendimento do fenômeno.

Embora não tenha sido desenvolvida para uso em um cenário industrial, comercial ou de serviços de manutenção, não se pode descartar o uso da Linguagem Abd1 também para modelagem e simulação de fenômenos de interesse de especialistas. Por exemplo, ela poderia ser utilizada em engenharia para criar modelos de máquinas reais, para posteriormente ajudar técnicos, interessados na manutenção destas máquinas, a formularem hipóteses sobre possíveis defeitos delas.

Outro possível cenário de aplicação da Linguagem Abd1 tem relações com a área de Arquitetura de Sistemas de Desenvolvimento de Software e, portanto, tem natureza puramente computacional, ao contrário das potenciais aplicações descritas anteriormente. Linguagens de propósito específico, como SQL, são utilizadas em conjunto com outras linguagens, e a arquitetura geral do sistema de desenvolvimento de software fica mais rica devido à especialidade da linguagem SQL em permitir a descrição de consultas de dados armazenados em tabelas relacionais. A Linguagem Abd1 permite operar raciocínios sobre um modelo, o TCHF, capaz de armazenar uma base de conhecimento. Raciocínios podem ser expressos utilizando Abd1 e o resultado destes raciocínios pode ser tornado disponível, para utilização por outra linguagem, na forma de um conjunto, uma vez que o resultado de um raciocínio em Abd1 é um conjunto. O uso concomitante da Linguagem Abd1 e alguma linguagem de programação de propósito geral (e.g., Prolog, Java, C#) tem o potencial de criar uma arquitetura de desenvolvimento de sistemas inteligentes que se enriquece pela especialidade da linguagem Abd1 em permitir a descrição de problemas de raciocínio abduativo.

6.4 – Limitações deste trabalho

Modelos em Abd1 são descritos por meio de sentenças escritas na linguagem da Lógica Proposicional. Como qualquer lógica, a Lógica Proposicional é limitada na sua capacidade expressiva e esta limitação constitui a mais notável limitação da Linguagem Abd1. O compromisso ontológico da Lógica Proposicional com o mundo, i.e., o que a Lógica Proposicional pressupõe sobre a natureza da realidade são os fatos (Russel & Norvig, 2014). Ou seja, modelar o mundo em Lógica Proposicional significa modelar o mundo por meio de fatos capturados por proposições e sentenças sobre estas proposições. A Lógica Proposicional se ressentida de falta de capacidade expressiva para modelar de forma concisa, entre outras coisas, ambientes com muitos objetos, relações temporais e de ordem entre objetos, relações gerais (não necessariamente temporais ou de ordem) entre objetos e fatos com certo grau de verdade. A Lógica de Primeira Ordem pressupõe que o mundo é constituído por fatos, objetos e relações entre estes objetos permitindo expressar, de maneira mais concisa, fenômenos sobre quantidade grande de objetos, o que em Lógica Proposicional seria enfadonho ou, até mesmo, impossível diante de uma situação onde são necessárias infinitas sentenças proposicionais. Outras

lógicas dão atenção especial a uma ou mais elementos dentro da lógica. Por exemplo, a Lógica Temporal pressupõe que fatos estão ordenados e são válidos em tempos ou intervalos de tempo particulares e a Lógica Difusa pressupõe que fatos possuem um grau de verdade indicado por um valor real pertencente ao intervalo $[0..1]$, onde 0 significa totalmente falso e 1 significa totalmente verdade. A modelagem de fenômenos temporais ou com certo grau de validade de forma geral conduz a uma enorme quantidade de sentenças em Lógica Proposicional e, em muitas situações é impossível. Uma grande quantidade de sentenças para descrever um modelo, não só torna o modelo difícil de ser criado, interpretado e mantido, mas também tem execução computacional custosa.

Particularmente, a expressividade da Linguagem Abd1 é ainda mais limitada do que a Linguagem da Lógica Proposicional. Sentenças em Abd1 devem satisfazer à Forma HF (Seção 3.1) e esta forma impõem à Linguagem Abd1 uma limitação ainda maior do que as limitações da Linguagem da Lógica Proposicional.

A Forma HF é útil porque permite o desenvolvimento de algoritmos de resolução lógica mais eficientes do que o que se teria com a utilização da linguagem da Lógica Proposicional. No entanto, somente após ter sido implementada e ter sido utilizada para compor sentenças é que se constatou que a Forma HF é difícil de ser memorizada, se considerarmos que o principal fundamento do *design* da Abd1 é a facilidade de uso por um programador. Na prática, nas experimentações de modelagem que realizamos, o programador parece escrever sentenças sem se importar conscientemente se elas estão ou não na Forma HF, mas recorrentemente recebe mensagens de que a sentença criada não pode ser aceita e, assim, é obrigado a reescrever.

O Algoritmo Peirce, em sua formulação original (Rodrigues, Oliveira e Oliveira, 2014), incorpora dois critérios para escolha de boas hipóteses: a força explanatória mínima (λ_1) e a complexidade máxima de hipóteses (λ_2). O algoritmo permite o controle destes critérios por meio da variação dos valores destes parâmetros, $0 \leq \lambda_1 \leq 1$ e $\lambda_2 \geq 1$, mas ele sempre computa como boas hipóteses aquela com menor complexidade entre as que possuem máxima força explanatória. Isto é suficiente para muitos propósitos, mas experimentos de modelagem realizados mostraram que há situações em que seria necessário desativar a computação de hipóteses baseadas nestes critérios e isto não é possível na concepção e implementação original do algoritmo Peirce.

Sentenças em Lógica Proposicional (em outras lógicas também) nem sempre representam o “senso comum” e isto pode afetar a correta descrição de modelos abdutivos, principalmente por não especialistas em Lógica. Por exemplo, considere que a seguinte sentença em Língua Portuguesa representa o entendimento de uma pessoa sobre um homicídio

Se ocorreu um homicídio então há um réu e há uma vítima,

e considere também que a sentença

$$\text{homicídio} \rightarrow \text{réu} \wedge \text{vítima}$$

é um possível correspondente modelo deste entendimento em Lógica Proposicional.

O “senso comum” conduz em geral a se pensar que é necessário haverem um *réu* e uma *vítima* para que se possa estabelecer a hipótese abdutiva de que houve um *homicídio*.

No entanto, do ponto de vista lógico, a ocorrência isolada do fato *réu* é suficiente para o cálculo da hipótese abdutiva *homicídio*, pois

$$(\text{homicídio} \rightarrow \text{réu} \wedge \text{vítima}) \wedge \text{homicídio} \models \text{réu}.$$

As hipóteses calculadas pelo Algoritmo Peirce são sempre MSA (*most specific abduction*) e isto constitui uma outra limitação da Linguagem Abd1. Uma hipótese MSA é uma hipótese que representa uma causa final, ao contrário de hipóteses que são causas intermediárias ou indiretas. Por exemplo, considerando a teoria

Theory T { chuva $\rightarrow$ gramadoMolhado, nuvemNimbus $\rightarrow$ chuva, irrigadoresLigados $\rightarrow$ gramadoMolhado },

{ *nuvemNimbus, irrigadoresLigados* } é o conjunto de hipóteses calculado pelo algoritmo Peirce para a ocorrência do fato *gramadoMolhado* (hipóteses do tipo MSA). No entanto, hipóteses intermediárias, como *chuva*, podem ser desejadas em algumas situações e não há como obtê-las usando o algoritmo Peirce.

6.5 – Trabalhos futuros

As limitações da Linguagem Abd1 explicitadas na seção anterior constituem uma fonte de inspiração para trabalhos futuros. Sendo a Lógica Proposicional a mais destacada limitação deste trabalho é natural propor, como trabalho futuro, a extensão da

Linguagem Abd1 para que ela permita a descrição de sentenças em Lógica de Primeira Ordem, o que representaria um avanço em termos de capacidade expressiva.

O estudo de formas de sentenças menos restritivas e mais fáceis de serem memorizadas do que a Forma HF, mas que também permitam a escrita de algoritmos de resolução eficientes, é um importante desafio futuro também. Em particular, tem importância um estudo teórico sobre o poder expressivo de sentenças na Forma HF e de novas possíveis formas que venham a ser propostas.

O desafio de eliminar limitações do algoritmo Peirce também fazem parte de uma possível agenda de trabalhos futuros. Em especial, é útil o desenvolvimento de um algoritmo para raciocínio abdutivo que (1) não vincule o cálculo das hipóteses ao cálculo de boas hipóteses e (2) forneça como resposta não só as hipóteses MSA (Seção 6.4), mas também possíveis hipóteses intermediárias. Mais adequadamente ainda, tal algoritmo poderia parametrizar o controle do tipo de hipótese (MSA somente, MSA e intermediárias) a ser calculada.

A realização de experimentos para comparar a usabilidade da Linguagem Abd1 com outras soluções para raciocínio abdutivo será útil na investigação do real valor da Linguagem Abd1 como proposta alternativa.

A Seção 6.3 descreve possíveis usos da Linguagem Abd1. Em especial, um trabalho de interesse no futuro é a investigação do uso desta linguagem em um contexto educacional de modelagem de fenômenos que fazem parte do programa de estudos de um curso ou escola.

A Linguagem Abd1 foi desenvolvida dentro de um contexto de pesquisa sobre raciocínios abduativos. Finalmente, existe um ambiente que a implementa e agora, ela pode ser útil como um “motor” para fazer avançar em uma velocidade maior estudos teóricos e experimentais que (1) busquem compreender melhor as necessidades linguísticas de sistemas para modelagem de raciocínios abduativos, (2) que investiguem fatores que influenciam a escolha de boas hipóteses abduativas e (3) que avaliem a efetividade da Abd1 na produção de sistemas que possam vir a ser úteis para alguém, algum grupo, algum domínio ou algum segmento social.

Apêndice I

Projeto Gramatical da Linguagem

Abd1

Este apêndice descreve completamente a gramática da linguagem Abd1. A gramática da linguagem Abd1 é dividida em dois ambientes: o ambiente de edição de programas e o ambiente de execução de raciocínios abdutivos. O conjunto de sentenças que podem ser escritas em cada um destes ambientes é diferente um do outro. Embora existam dois ambientes, a gramática da linguagem Abd1 não é uma gramática sensível a contexto. O que se tem são duas gramáticas do tipo livre de contexto uma para cada um dos ambientes. Assim, a Seção I.2 deste apêndice trata da gramática do ambiente de edição de programas e a Seção I.3 trata da gramática do ambiente de execução de raciocínios abdutivos. A Seção I.1 declara o formalismo de expressão gramatical empregado.

I.1 – Formalismo de expressão gramatical²⁰

O formalismo de expressão gramatical empregado neste apêndice é uma adaptação do formalismo usualmente empregado por textos tradicionais de Linguagens Formais e Autômatos (e.g., Hopcroft & Ullman, 1979) para expressar gramáticas livres de contexto. A adaptação é a seguinte:

- Variáveis: são denotadas por palavras escritas entre os símbolos “<” e “>” em vez das tradicionais letras maiúsculas.
- Símbolos terminais: são denotados por palavras e símbolos especiais de comprimento igual a um (‘{’, ‘}’, ‘(’, ‘)’, ‘.’, ‘;’, ‘→’ etc.) destacados em negrito em vez das tradicionais letras minúsculas.
- Produções: são denotadas no formato $X \rightarrow \alpha$ onde X é uma variável e α pertence ao fecho reflexivo e transitivo da união do conjunto de variáveis com o conjunto de símbolos terminais. A diferença em relação aos textos tradicionais é que “ $\rightarrow$ ” está sendo usado no lugar de “ $\Rightarrow$ ”, uma vez que este último símbolo é amplamente utilizado nas produções da gramática de Abdl.

I.2 – Ambiente de edição de programas

```
<programa> -> <sequenciaSentencaDeclarativa>
```

```
<sequenciaSentencaDeclarativa> -> <sentencaDeclarativa> |  
    <sentencaDeclarativa> <sequenciaSentencaDeclarativa>
```

```
<sentencaDeclarativa> -> <declTeoria> |  
    <declCondicoes> |  
    <declFatos> |  
    <declRaciocinioAbdutivo>
```

```
<declTeoria> -> Theory <identificador> { <sequenciaSentencaHF> }
```

²⁰ Esta seção repete o texto da Seção 3.3 do Capítulo 3, com objetivo de facilitar a leitura daquele que se dirige diretamente a este apêndice.

$\langle \text{sequenciaSentencaHF} \rangle \rightarrow \langle \text{sentencaHF} \rangle \mid$
 $\langle \text{sentencaHF} \rangle , \langle \text{sequenciaSentencaHF} \rangle$

$\langle \text{sentencaHF} \rangle \rightarrow \langle \text{forma1} \rangle \mid \langle \text{forma2} \rangle \mid \langle \text{forma3} \rangle \mid \langle \text{forma4} \rangle$

$\langle \text{forma1} \rangle \rightarrow \langle \text{parte1SentencaConjuntiva} \rangle$

$\langle \text{forma2} \rangle \rightarrow \langle \text{parte1SentencaDisjuntiva} \rangle$

$\langle \text{forma3} \rangle \rightarrow \langle \text{parte1SentencaConjuntiva} \rangle \rightarrow \langle \text{parte2SentencaConjuntiva} \rangle$

$\langle \text{forma4} \rangle \rightarrow \langle \text{parte1SentencaDisjuntiva} \rangle \rightarrow \langle \text{parte2SentencaDisjuntiva} \rangle$

$\langle \text{parte1SentencaConjuntiva} \rangle \rightarrow \langle \text{literal} \rangle \mid$
 $\langle \text{literal} \rangle \wedge \langle \text{parte1SentencaConjuntiva} \rangle$

$\langle \text{parte1SentencaDisjuntiva} \rangle \rightarrow \langle \text{literal} \rangle \mid$
 $\langle \text{literal} \rangle \vee \langle \text{parte1SentencaDisjuntiva} \rangle$

$\langle \text{parte2SentencaConjuntiva} \rangle \rightarrow \langle \text{literal} \rangle \mid$
 $\langle \text{literal} \rangle \wedge \langle \text{parte2SentencaConjuntiva} \rangle$

$\langle \text{parte2SentencaDisjuntiva} \rangle \rightarrow \langle \text{literal} \rangle \mid$
 $\langle \text{literal} \rangle \vee \langle \text{parte2SentencaDisjuntiva} \rangle$

$\langle \text{literal} \rangle \rightarrow \langle \text{proposi\c{c}\~ao} \rangle \mid \neg \langle \text{proposicao} \rangle$

$\langle \text{declFatos} \rangle \rightarrow \textbf{Facts} \langle \text{identificador} \rangle \{ \langle \text{sequenciaProposicao} \rangle \}$

$\langle \text{sequenciaProposicao} \rangle \rightarrow \langle \text{proposi\c{c}\~ao} \rangle \mid$
 $\langle \text{proposi\c{c}\~ao} \rangle , \langle \text{sequenciaProposicao} \rangle$

```

<declCondicoes> -> Conditions <identificador>

    { <sequenciaSentencaHF> }

<declRaciocinioAbdutivo> ->

    AbductiveReasoning <identificador> { <dclComposicaoRaciocinio> } |

    AbductiveReasoning <identificador> { <dclComposicaoRaciocinio> }

        Options { <sequenciaOpcoesRaciocinio> }

<dclComposicaoRaciocinio> ->

    <sequenciaTeoria> , <sequenciaFato> |

    <sequenciaTeoria> , <sequenciaFato> , <sequenciaCondicao>

<sequenciaTeoria> -> <especificacaoTeoria> |

    <especificacaoTeoria> U <sequenciaTeoria>

<especificacaoTeoria> -> <identificadorParaTeoria> |

    { <sequenciaSentencaHF> }

<sequenciaFato> -> <especificacaoFato> |

    <especificacaoFato> U <sequenciaFato>

<especificacaoFato> -> <identificadorParaFato> |

    { <sequenciaProposicao> }

<sequenciaCondicao> -> <especificacaoCondicao> |

    <especificacaoCondicao> U <sequenciaCondicao>

<especificacaoCondicao> -> <identificadorParaCondicao> |

    { <sequenciaSentencaHF> }

```

<sequenciaOpcoesRaciocinio> ->
 <especificacaoMaxComplexidade>, <especificacaoMinForcaExplanatoria> |
 <especificacaoMinForcaExplanatoria>, <especificacaoMaxComplexidade>
 <especificacaoMaxComplexidade> -> $\lambda 2$ = <inteiroMaiorQueUm>
 <especificacaoMinForcaExplanatoria> -> $\lambda 1$ = <decimalEntreZeroUm>

I.2 – Ambiente de execução de raciocínios abduativos

<questao> -> <sequenciaSentencaExecucao>
 <sequenciaSentencaExecucao> -> <expressaoRaciocinio> . |
 <expressaoRaciocinio> ; <sequenciaSentencaExecucao>
 <expressaoRaciocinio> -> <TermoPri1> | <TermoPri1> -
 <RepeticaoTermoPri1>
 <RepeticaoTermoPri1> -> <TermoPri1>
 | <TermoPri1> - <RepeticaoTermoPri1>
 <TermoPri1> -> <TermoPri2> | <TermoPri2> **U** <RepeticaoTermoPri2>
 <RepeticaoTermoPri2> -> <TermoPri2>
 | <TermoPri2> **U** <RepeticaoTermoPri2>
 <TermoPri2> -> <TermoPri3> | <TermoPri3> **N** <RepeticaoTermoPri3>
 <RepeticaoTermoPri3> -> <TermoPri3>
 | <TermoPri3> **N** <RepeticaoTermoPri3>
 <TermoPri3> -> <identificadorRaciocinio> |
 (<dclComposicaoRaciocinio>)
 (<expressaoRaciocinio>)

```

<dclComposicaoRaciocinio> ->
    <sequenciaTeoria> , <sequenciaFato> |
    <sequenciaTeoria> , <sequenciaFato> , <sequenciaCondicao>

<sequenciaTeoria> -> <especificacaoTeoria> |
    <especificacaoTeoria> U <sequenciaTeoria>

<especificacaoTeoria> -> <identificadorParaTeoria> |
    { <sequenciaSentencaHF> }

<sequenciaFato> -> <especificacaoFato> |
    <especificacaoFato> U <sequenciaFato>

<especificacaoFato> -> <identificadorParaFato> |
    { <sequenciaProposicao> }

<sequenciaCondicao> -> <especificacaoCondicao> |
    <especificacaoCondicao> U <sequenciaCondicao>

<especificacaoCondicao> -> <identificadorParaCondicao> |
    { <sequenciaSentencaHF> }

```

Apêndice II

Primeiro artigo publicado

O artigo *Peirce - an Algorithm for Abductive Reasoning Operating with a Quaternary Reasoning Framework*, foi submetido e aceito na 13th Mexican International Conference on Artificial Intelligence (MICAI), Qualis-CAPES B2 em Ciência da Computação. O artigo foi publicado nos anais da conferência e também no *Research in Computing Science Journal*, ISSN 1870-4069.

Peirce: an Algorithm for Abductive Reasoning Operating with a Quaternary Reasoning Framework

Felipe Rodrigues, Carlos Eduardo A. Oliveira, and Osvaldo Luiz de Oliveira

FACCAMP, Campo Limpo Paulista (SP),
Brazil

rodrigues_felipe7@hotmail.com, carlos.br@gmail.com, osvaldo@faccamp.br

Abstract. Abductive reasoning algorithms formulate possible hypotheses to explain observed facts using a theory as the basis. These algorithms have been applied to various domains such as diagnosis, planning and interpretation. In general, algorithms for abductive reasoning based on logic present the following disadvantages: (1) they do not allow the explicit declaration of conditions that may affect the reasoning, such as intention, context and belief; (2) they allow little or no consideration for criteria required to select good hypotheses. Using Propositional Logic as its foundation, this study proposes the algorithm Peirce, which operates with a framework that allows one to explicitly include conditions to conduct abductive reasoning and uses a criterion to select good hypotheses that employs metrics to define the explanatory power and complexity of the hypotheses. Experimental results suggest that abductive reasoning performed by humans has the tendency to coincide with the solutions computed by the algorithm Peirce.

Keywords: Abductive reasoning, automated Reasoning, logic, human factors.

1 Introduction

Abductive reasoning formulates hypotheses to explain observed facts using a theory as the basis. Numerous intellectual tasks make use of abductive reasoning, including medical diagnostics, fault diagnostics, scientific discovery, legal argumentation and interpretation.

Abductive reasoning algorithms based on logic frequently operate with the Theory, Hypotheses and Facts (THF) ternary reasoning framework (as shown in [2], [4], [5] and [11]). When these algorithms are formalized using Propositional Logic (PL) [9], the THF framework is frequently instantiated according to Definition 1.

Definition 1 (THF reasoning framework). The THF reasoning framework for abductive reasoning is a system $\langle T, H, F \rangle$ that consists of the following:

- A finite and non-empty theory set, $T = \{ t_1, t_2, t_3, \dots, t_m \}$, of PL sentences denoting $t_1 \wedge t_2 \wedge t_3 \wedge \dots \wedge t_m$. This set represents the hypotheses that must be assumed as truth during the reasoning process.

- A finite hypotheses set, $H = \{ h_1, h_2, h_3, \dots, h_n \}$, of PL sentences denoting $h_1 \vee h_2 \vee h_3 \vee \dots \vee h_n$. This set represents the hypotheses that along with the set T explain the facts represented by the set F.
- A set with a single fact, $F = \{ f \}$, where f is a PL literal (atom or negated atom). This set represents an occurrence of an evidence, a manifestation, a symptom, an observation, a mark or a sign to be explained through abductive reasoning.

Having the T and F sets as input, an abductive reasoning algorithm should find a set of hypotheses H that satisfies the following conditions:

$$T \not\models F, \quad (1)$$

$$T \cup \{ h \} \models F, \forall h \in H, \quad (2)$$

$$T \cup \{ h \} \not\models \perp, \forall h \in H, \quad (3)$$

$$\{ h \} \not\models F, \forall h \in H. \quad (4)$$

The statements above refer to the concept of logical consequence described in Definition 2.

Definition 2 ($A \models B$, i.e., B is a logical consequence of A). Let $A = \{ a_1, a_2, a_3, \dots, a_n \}$ and $B = \{ b_1, b_2, b_3, \dots, b_m \}$ be two finite and non-empty sets of PL sentences. Then, $A \models B$ if and only if the interpretations in which $a_1 \wedge a_2 \wedge a_3 \wedge \dots \wedge a_n$ is true, $b_1 \wedge b_2 \wedge b_3 \wedge \dots \wedge b_m$ is also true.

The condition (1) prevents that the theory set T alone has as logical consequence the facts set F. Hypotheses satisfying condition (2) are called **candidate hypotheses**, and they can explain the single fact denoted by F. Candidate hypotheses satisfying condition (3) are called **consistent hypotheses**. Conversely, candidate hypotheses that do not satisfy condition (3) are called inconsistent hypotheses and should be discarded. Candidate hypotheses that satisfy condition (4) are called **explanatory hypotheses**. Conversely, candidate hypotheses that do not satisfy condition (4) are called non-explanatory hypotheses and should be discarded.

Example 1. Joseph has a large lawn in front of his house. One day, Joseph arrives at home and observes that the lawn is wet. Considering only that (1) rain can make the lawn wet and that (2) sprinklers installed across the lawn can make it wet, which hypotheses can explain the fact that the lawn is wet?

One possible formalization using the THF framework consists in defining:

- Propositions ‘ r : **Rain** occurred’, ‘ s : **Sprinklers** were activated’ and ‘ w : Lawn is **wet**’.
- A theory set $T = \{ r \rightarrow w, s \rightarrow w \}$.
- A fact set $F = \{ w \}$.

The theory set T and the fact set F satisfy condition (1), whereas a theory $T_1 = \{ r \rightarrow w, s \rightarrow w, w \}$ has F as its logical consequence; therefore T_1 and F do not satisfy condition (1). Let $H = \{ r, s, r \wedge s, r \wedge \neg w, w \}$ be a set of candidate hypotheses. Each hypothesis $h \in H$ satisfies condition (2), and each hypothesis $h \in \{ r, s, r \wedge s, w \}$ satisfies condition (3), i.e., they are consistent. However, the hypothesis $r \wedge \neg w$ is inconsistent because $T \cup \{ r \wedge \neg w \} \models \perp$; therefore, it must be discarded. Each

hypothesis $h \in \{ r, s, r \wedge s \}$ satisfies condition (4); however, the hypothesis w is not explanatory because $\{ w \} \models F$; therefore, it must also be discarded. Thus, removing the inconsistent and non-explanatory hypotheses from H , we obtain $H = \{ r, s, r \wedge s \}$.

In general, abductive algorithms work as follows: having the theory set T and the fact set F as the input, the algorithm verifies whether or not the condition (1) has been satisfied; if the condition (1) is not satisfied, then there are no hypotheses to be formulated because F is a logical consequence of T ; however, if the condition (1) is satisfied, then the algorithm formulates a finite set of possible hypotheses H that satisfies the condition (2). Next, the algorithm removes from H the hypotheses that do not satisfy the conditions (3) and (4), and thus returning the resulting set H as an answer by the algorithm.

Some algorithms, however, include an additional step with the goal of letting in H only hypotheses considered good, according to extra-logical criteria. A criterion commonly used is “simplicity”, which considers, for example, an atomic hypothesis better than a composite hypothesis, e.g., r is better than $r \wedge s$.

The need to represent conditions such as context, circumstance and intention is common and important when conducting abductive reasoning. For example, reasoning to make a medical diagnosis considering the context of diseases of a region. Operating with a THF reasoning framework, the existing algorithms to perform abductive reasoning have the disadvantage of forcing the representation of these conditions in the theory set T . This solution is not appropriate because representing conditions in the theory set T mischaracterizes the theory, making it less general and more *ad hoc* (specific to explain what one wants to explain).

Abductive reasoning formulates hypotheses, and some of these hypotheses may be better at explaining the facts than others. Today we do not know, exactly, which criteria determine what makes a hypothesis better than another, authors from several fields [3] [8] [10] [13] [16] [17] have suggested that abductive reasoning involves the selection of good hypotheses. However, the existing abductive reasoning algorithms have the disadvantage of dedicating little or no consideration for criteria required to select good hypotheses.

Many practical applications of reasoning require the definition of a set of $n \geq 2$ facts. However, the many existing algorithms have the disadvantage of operating with only a single fact.

This work proposes an algorithm, called Peirce, that performs abductive reasoning, and this algorithm differs from the existing solutions mainly because (1) it works with a reasoning framework called TCHF (Theory, accepted Conditions, Hypotheses and Facts), thus allowing conditions to be explicitly represented; (2) it allows $n \geq 2$ facts to be represented; and (3) it introduces a criterion to select good hypotheses that employ metrics to define the explanatory power and the complexity of the hypotheses.

Section 2 describes the algorithm Peirce, dedicating particular attention to the design and operation of the TCHF reasoning framework (Subsection 2.1) and the definition of a criterion to select good abductive hypotheses (Subsection 2.2). The pseudocode for the algorithm Peirce is presented and discussed in Subsection 2.3. Section 3 details an experimental study conducted to verify whether the solutions computed by the algorithm Peirce tend to coincide with the abductive reasoning

performed by humans. Section 4 describes related works, highlighting the differences with this work. Section 5 presents the conclusions.

2 The Algorithm Peirce

The abductive reasoning algorithm proposed in this study has been named Peirce in honor of the American philosopher Charles Sanders Peirce, who created the concept of abductive reasoning [14]. The following subsections detail the reasoning framework used by the algorithm Peirce, a criteria to select good hypotheses and the pseudocode of the algorithm.

2.1 TCHF Reasoning Framework

The TCHF reasoning framework proposed in this study differs from the classic THF reasoning framework (Definition 1) by including the accepted conditions set C and by redefining the facts set F to allow the declaration of not just one single fact, but rather a finite number of one or more facts. The TCHF framework is formalized in Definition 3 and uses the PL sentences in HF form as specified in Definition 4.

Definition 3 (TCHF reasoning framework). The TCHF framework for abductive reasoning is a system $\langle T, C, H, F \rangle$ consisting of the following:

- A finite and non-empty theory set, $T = \{ t_1, t_2, t_3, \dots, t_m \}$, of PL sentences in HF form denoting $t_1 \wedge t_2 \wedge t_3 \wedge \dots \wedge t_m$. This set represents the hypotheses that must be assumed as truth during the reasoning process.
- A finite hypotheses set, $H = \{ h_1, h_2, h_3, \dots, h_n \}$, of PL sentences in HF form denoting $h_1 \vee h_2 \vee h_3 \vee \dots \vee h_n$. This set represents the hypotheses that along with the sets T and C explain the facts represented by the set F.
- A finite accepted conditions set, $C = \{ c_1, c_2, c_3, \dots, c_p \}$, of PL sentences in HF form denoting $c_1 \wedge c_2 \wedge c_3 \wedge \dots \wedge c_p$. This set represents the conditions that must be assumed as truth during the reasoning process.
- A finite and non-empty facts set $F = \{ f_1, f_2, f_3, \dots, f_q \}$ of PL positive literals, denoting $f_1 \wedge f_2 \wedge f_3 \wedge \dots \wedge f_q$. The role of this set is to represent evidences, manifestations, symptoms, observations, marks or signs to be explained by the abductive reasoning.

Definition 4 (HF form). A sentence of PL in the HF form is an acyclic sentence written in one of the following formats:

- $a_1 \wedge a_2 \wedge a_3 \wedge \dots \wedge a_n$, where a_i ($1 \leq i \leq n$) are literals.
- $a_1 \vee a_2 \vee a_3 \vee \dots \vee a_n$, where a_i ($1 \leq i \leq n$) are negative literals.
- $a_1 \wedge a_2 \wedge a_3 \wedge \dots \wedge a_n \rightarrow b_1 \wedge b_2 \wedge b_3 \wedge \dots \wedge b_m$, where a_i ($1 \leq i \leq n$) are positive literals and b_j ($1 \leq j \leq m$) are literals.
- $a_1 \vee a_2 \vee a_3 \vee \dots \vee a_n \rightarrow b_1 \vee b_2 \vee b_3 \vee \dots \vee b_m$, where a_i ($1 \leq i \leq n$) are literals and b_j ($1 \leq j \leq m$) are negative literals.

The restriction of TCHF framework to sentences in the HF form aims to make the algorithm Peirce run the conversion of sentences in polynomial time, because sentences in HF form can be easily converted into Horn Clauses [9] (disjunction of literals with at most one positive literal). As will be described in Subsection 2.3, algorithm Peirce uses Resolution as the inference mechanism and this mechanism can be efficiently implemented on Conjunctive Normal Form (CNF) sentences with Horn Clauses.

The set C gives the TCHF reasoning framework the advantage of allowing the explicit definition of conditions that in the classical THF framework, would normally be declared within the theory set T. Thus, the set C avoids “contaminating” the set T with sentences that fundamentally do not belong to the theory. Moreover, this makes it easier to represent two or more instances of abductive reasoning that share the declarations of T and F but differ in the set of accepted conditions. Example 2, which is described next, illustrates the use of the TCHF reasoning framework.

Example 2. Consider once more the scenario described in Example 1 in which Joseph arrives at home and observes his lawn wet. However, let us say that Joseph knows that the water tank supplying the sprinklers has been empty for a month; therefore, under this condition, the sprinklers could not have been activated.

One possible formalization using the TCHF framework is to define the following:

- Propositions ‘*r*: **Rain** occurred’, ‘*s*: **Sprinklers** were activated’, ‘*w*: Lawn is **wet**’ and ‘*t*: Water **tank** that supplies sprinklers is empty’.
- A theory $T = \{ r \rightarrow w, s \rightarrow w \}$, which is the same as in Example 1.
- A set of accepted conditions $C = \{ t, t \rightarrow \neg s \}$.
- A set of facts $F = \{ w \}$, which is the same as in Example 1.

The conditions in abductive reasoning are motivated by several factors which are linked to context (information associated to space), circumstances (information associated with time), intention (manifestation of the will to reach some wanted conclusions), belief or faith (information that is accepted on principle) etc. Examples of specific conditions used in abductive reasoning are as follows: (1) In abductive reasoning used for medical diagnoses, regional context may allow one to specify a set of diseases that are common or uncommon for a given region; (2) In abductive reasoning used for anthropological studies, the specification of possible agents that might have been responsible for the death of a hominid based on the knowledge that the hominid lived 4 million years ago (circumstance); (3) In abductive reasoning for judicial decisions, possible conditions may be specified with the intent of acquitting (or condemning) a defendant; and (4) In abductive reasoning for religious or metaphysical beliefs, the faith or belief that there is life after death can be declared as a condition upon which reasoning are made.

Taking the sets T and F as inputs, an abductive reasoning algorithm operating with the TCHF framework should find a set of hypotheses H that satisfies the following conditions:

$$T \cup C \neq F, \quad (5)$$

$$T \cup C \cup \{ h \} \models_p F, \forall h \in H, \quad (6)$$

$$T \cup C \cup \{ h \} \not\models \perp, \forall h \in H, \quad (7)$$

The condition (6) uses the partial logical consequence as defined in Definition 5.

Definition 5 ($A \models_p B$, i.e., B is a partial logical consequence of A). Let $A = \{ a_1, a_2, a_3, \dots, a_n \}$, $B = \{ b_1, b_2, b_3, \dots, b_m \}$ and $C = \{ c_1, c_2, c_3, \dots, c_q \}$, $C \subseteq B$, be three finite and non-empty sets of PL sentences. Then, $A \models_p B$ if only if the interpretations in which $a_1 \wedge a_2 \wedge a_3 \wedge \dots \wedge a_n$ is true, $c_1 \wedge c_2 \wedge c_3 \wedge \dots \wedge c_q$ is also true.

2.2 Selection of Good Abductive Hypotheses

In general, several hypotheses may be able to explain observed facts. However, certain hypotheses may explain facts better than others. Therefore, abductive reasoning can be observed as a process that formulates $m \geq 1$ general hypotheses followed by the selection of $n \leq m$ good hypotheses. Naturally, selection criteria must be established, but it is still difficult to define the conditions that make a hypothesis good.

Contemporary philosophers have analyzed the issue of selecting good hypotheses. Harman [8] considers abduction to be an inference of the best explanation and argues that the best hypothesis is the simplest, most plausible and is the least *ad hoc*. By comparing theories (e.g., Darwin's Theory of Evolution vs. Creationist Theory or Lavoisier's Theory of Combustion vs. Phlogiston Theory), Thagard [16, 17] establish criteria that explain the preference for one hypothesis over another and considers the best hypothesis to be the most consilient (explains more facts), the most simple, and it would provide the best analogy with hypotheses that explain facts in other domains.

Criteria to select good hypotheses have been extensively studied in the fields of philosophy (e.g., [2] [8] [16]), psychology (e.g., [13]) and artificial intelligence (e.g., [3] [10] [15]). However, the precise formulation of these criteria remains controversial. In general, factors such as the "explanatory power" and the "complexity" of a hypothesis are recurrent and have similar connotations across several studies. Therefore, this study has proposed using these two factors to develop a selection criterion. Aiming at the development of algorithms to perform abduction that need dealing with quantitative measures for the explanatory power and the complexity of a hypothesis, this study proposes an understanding of these factors as follows:

- **Explanatory power** (or comprehensiveness): the explanatory power of a hypothesis quantifies the degree to which it is capable of explaining the facts involved in the reasoning. A metric for a hypothesis' explanatory power is given by the ratio between the number of facts it can explain and the total number of facts to be explained by the abductive reasoning process. For example, a hypothesis that explains 4 out of 5 facts has an explanatory power of 4/5, and a hypothesis that explains all of the facts has an explanatory power of 1.
- **Complexity**: the complexity factor refers to how many different elements and relationships are present in a hypothesis. A metric for hypothesis complexity is the number of atomic propositions that it contains. For example, hypothesis r has a complexity of 1, and hypothesis $r \wedge s \wedge w$ has a complexity of 3.

Based on the metrics for explanatory power and complexity, this study proposes a criterion to select good hypotheses, which is declared in Definition 6.

Definition 6 (A criterion to select good hypotheses). Given a set H of candidate hypotheses to explain a set F of facts, $h \in H$ is considered a good hypothesis if it satisfies all the following conditions:

- The explanatory power of h is equal to or greater than a constant λ_1 . The constant $\lambda_1 = 0.5$ has been used in the experiments described in this article.
- The complexity of h is equal to or less than a constant λ_2 . The constant¹ $\lambda_2 = 5$ has been used in the experiments described in this article.
- The hypothesis h has the minimum complexity among all of the hypotheses that have the maximum explanatory power in H .

Examples 3 and 4 illustrate the application of Definition 6.

Example 3. Diseases manifest themselves through symptoms. Consider the following:

- Propositions ‘ c : Disease is **cold**’, ‘ p : Disease is **pneumonia**’, ‘ r : Disease is **rhinitis**’, ‘ f : Symptom is **fever**’, ‘ h : Symptom is **headache**’, and ‘ z : Symptom is **coryza**’;
- Theory $T = \{ p \rightarrow f \wedge z \wedge h, c \rightarrow f \wedge z, r \rightarrow h \wedge z \}$, the empty set $C = \{ \}$ of accepted conditions and observed facts set $F = \{ f, z, h \}$ (symptoms);
- A set of candidate hypotheses $H = \{ p, c, r, p \wedge c, p \wedge r, c \wedge r, p \wedge c \wedge r \}$.

Table 1 describes the explained facts, explanatory power and complexity of each candidate hypothesis $h \in H$.

Table 1. Explained facts, explanatory power and complexity of candidate hypothesis of the Example 3. The ‘ $\checkmark$ ’ signals an explained fact.

Hypothesis	Explained facts			Explanatory power	Complexity
	f	z	h		
p	$\checkmark$	$\checkmark$	$\checkmark$	1	1
c	$\checkmark$	$\checkmark$		0.66	1
r		$\checkmark$	$\checkmark$	0.66	1
$p \wedge c$	$\checkmark$	$\checkmark$	$\checkmark$	1	2
$p \wedge r$	$\checkmark$	$\checkmark$	$\checkmark$	1	2
$c \wedge r$	$\checkmark$	$\checkmark$	$\checkmark$	1	2
$p \wedge c \wedge r$	$\checkmark$	$\checkmark$	$\checkmark$	1	3

All of the hypotheses have an explanatory power equal to or greater than $\lambda_1 = 0.5$ and complexity equal to or less than $\lambda_2 = 5$. The hypotheses $p, p \wedge c, p \wedge r, c \wedge r, p \wedge c \wedge r$ have explanatory power equal to 1, which is the maximum among all candidate

¹ The λ_1 and λ_2 values were chosen to coincide with human factors. Considering Miller experiment [12], human memory and human processing capacity is limited to 7 ± 2 simultaneous elements, hence $\lambda_2 = 5$. Good hypotheses explain at least 50% of the facts, hence $\lambda_1 = 0.5$.

hypotheses. Among these hypotheses with maximum explanatory power, hypothesis p has the complexity equals to 1, which is the minimum among the hypotheses. Therefore, p is a good hypothesis according to Definition 6.

Example 4. Another example involving diseases and symptoms. Consider the following:

- Propositions ‘ d : Disease is **dengue**’, ‘ u : Disease is **flu**’, ‘ b : Symptom is **breathlessness**’, ‘ f : Symptom is **fever**’, ‘ h : Symptom is **headache**’, ‘ m : Symptom is **muscle pain**’, ‘ r : Symptom is **red spots**’ and ‘ s : Symptom is **sneezing**’;
- Theory $T = \{ u \rightarrow f \wedge h \wedge m \wedge s, d \rightarrow f \wedge h \wedge m \wedge r \}$, the empty set $C = \{ \}$ of accepted conditions and observed facts set $F = \{ f, h, m, b \}$ (symptoms);
- A set $H = \{ u, d, u \wedge d \}$ of candidate hypotheses.

Table 2 describes the explained facts, explanatory power and complexity of each candidate hypothesis $h \in H$.

Table 2. Explained facts, explanatory power and complexity of candidate hypothesis of the Example 4. The ‘ $\checkmark$ ’ signals an explained fact.

Hypothesis	Explained facts				Explanatory power	Complexity
	f	h	m	b		
U	$\checkmark$	$\checkmark$	$\checkmark$		0.75	1
D	$\checkmark$	$\checkmark$	$\checkmark$		0.75	1
$u \wedge d$	$\checkmark$	$\checkmark$	$\checkmark$		0.75	2

All of the hypotheses have an explanatory power equals to 0.75 (i.e., explanatory power equal to or greater than $\lambda_1 = 0.5$) and complexity equal to or less than $\lambda_2 = 5$. Hypotheses u and d have a complexity of 1, which is the minimum across all of the candidates. Therefore, u and d are good hypotheses according to Definition 6.

2.3 Pseudocode for the Algorithm Peirce

Figure 1 presents the pseudocode for the algorithm Peirce. The algorithm Peirce formulates hypotheses that comply with equations (5), (6), (7) and the criterion to select good hypotheses of the Definition 6. Synthetically, the algorithm Peirce formulates candidate hypotheses and stores them in set H (line 11). Next, the algorithm removes inconsistent hypotheses from H (line 12) and then selects and leaves only the good hypotheses in H (line 13). The details of the algorithm are described below.

The algorithm uses the Resolution rule of inference for PL sentences in CNF expressed with Horn Clauses [9]. Candidate hypotheses are hypotheses h that satisfy equation (6). To compute these hypotheses, the algorithm translates the set of sentences $T \cup C \cup \neg F$ to CNF expressed with Horn Clauses (line 4) and applies the mechanism of resolution (line 5). The result of the resolution is stored in the data structure R (set of clauses). If R contains at least one empty clause, then $T \cup C \models F$ and no hypotheses are formulated (lines 6 and 7). If R does not contain an empty

clause, then $T \cup C \models F$, equation (5) is met and candidate hypotheses can be formulated.

At line 11 each clause in R presents the possibility of formulating a hypothesis. Because R is in CNF, negating each clause results in a candidate hypothesis. The algorithm `Formulate_Candidate_Hypotheses` (line 11), operates as follows: (1) the algorithm negates each of the $m \geq 1$ clauses in R to obtain m first-candidates hypotheses $h_1, h_2, h_3, \dots, h_m$, (2) combines these m first-candidates hypotheses in pairs to obtain conjunctive hypotheses of the type $h_i \wedge h_j$ ($i \neq j$); and (3) combines the m first-candidates hypotheses three by three to obtain conjunctive hypotheses of the type $h_i \wedge h_j \wedge h_k$ ($i \neq j \neq k$); ... and then combines the m first-candidates hypotheses q by q to obtain conjunctive hypotheses of the type $h_i \wedge h_j \wedge \dots \wedge h_q$ ($i \neq j \neq \dots \neq q$), where $q = \min(m, \lambda_2)$ and is λ_2 the constant that defines the maximum complexity of the hypotheses (λ_2 is defined in Definition 6).

```

Algorithm Peirce( $T, C, F$ )
Input
Theory set  $T$ , accepted condition set  $C$  and facts set  $F$ 
(specification is given in Definition 3).
Output
Hypotheses set  $H$  (specification is given in Definition 3).
1  {
2    if Consistent( $T, C$ ) then
3    {
4       $R :=$  Conjunctive_Normal_Form_Horn_Clauses( $T, C, \neg F$ );
5       $R :=$  Resolution( $R$ );
6      if  $R$  contains an empty clause then
7        write ("No hypotheses to formulate:  $T \cup C \models F$ ");
9      else
10     {
11        $H :=$  Formulate_Candidate_Hypotheses( $R$ );
12        $H :=$  Remove_Inconsistent_Hypotheses( $T, C, H$ );
13        $H :=$  Select_Good_Hypotheses( $T, C, H, F$ );
14     }
15   }
16 else
17   write("Unable to formulate hypotheses:  $T \cup C \models \perp$ .");
18 }

```

Fig. 1. Algorithm Peirce.

At line 12, the algorithm `Remove_Inconsistent_Hypotheses` receives a set H of candidate hypotheses and removes from H hypotheses that do not satisfy $T \cup C \cup \{h\} \models \perp$ (conformity to equation (7)). The algorithm works as follows: For each $h \in H$: (1) the algorithm translates the sentences in the set $T \cup C \cup \{h\}$ to CNF expressed with Horn Clauses, (2) applies the Resolution mechanism to this system of

sentences and (3) removes hypothesis h from H if the Resolution mechanism derives an empty clause.

The algorithm `Select_Good_Hypotheses` receives a set H of candidate hypotheses that are all consistent and then operates as follows: (1) it computes the explanatory power and complexity of each hypothesis $h \in H$, (2) removes all of the hypotheses h with explanatory power below some constant λ_1 (0.5 in our experiments) or complexity above some constant λ_2 (5 in our experiments), (3) computes set E with the hypotheses that have the maximum explanatory power in H , (4) computes set X with the hypotheses that have the minimum complexity in E and (5) returns set X as answer.

It can be proved that the Peirce algorithm computes three different types of solutions: (1) $H = F$ when the theory and the accepted conditions does not allow Peirce algorithm to formulate explanatory hypotheses; (2) $H = \{ \}$ when Peirce algorithm does not consider any hypotheses to be good, among the candidate hypotheses; (3) H contains at least one explanatory hypothesis; In this last type, H does not contain non-explanatory hypothesis.

Example 5 illustrates a run of the algorithm Peirce.

Example 5. This example illustrates the execution of the algorithm Peirce using the scenario and formalization from Example 2. Therefore, the algorithm Peirce receives as input the theory $T = \{ r \rightarrow w, s \rightarrow w \}$, the set of accepted conditions $C = \{ t, t \rightarrow \neg s \}$ and the set of facts $F = \{ w \}$. Because $T \cup C \not\models \perp$, algorithm `Consistent`(T, C) returns the value true (line 2), and the data structure R is filled in with $T \cup C \cup \neg F$ in CNF expressed with Horn Clauses. The following is then established:

- At line 4: $R = \{ \{ \neg r, w \}, \{ \neg s, w \}, \{ t \}, \{ \neg t, \neg s \}, \{ \neg w \} \}$;
- At line 5 after Resolution: $R = \{ \{ \neg r \}, \{ \neg s \} \}$.

Because there are no empty clauses in R (test at line 6), the candidate hypotheses are formulated at line 11. Thus, $H = \{ r, s, r \wedge s \}$ at line 11 after executing `Formulate_Candidate_Hypotheses`. Because $T \cup C \cup \{ s \} \models \perp$ and $T \cup C \cup \{ r \wedge s \} \models \perp$, hypotheses s and $r \wedge s$ are removed from H by the algorithm `Remove_Inconsistent_Hypotheses` (line 12), leaving $H = \{ r \}$. The hypothesis r has an explanatory power of 1, a complexity of 1 and the minimum complexity of all hypotheses with maximum explanatory power in H (r is the only hypothesis in H), therefore the algorithm `Select_Good_Hypotheses` (line 13) selects r as a good hypothesis. The algorithm Peirce thus returns as answer $H = \{ r \}$.

In general, the complexity of logic-based abduction is NP-complete [6]. However, the algorithm Peirce has a running time $O(n^{2+\lambda_2})$. As λ_2 is a constant, typically equals to 5, Peirce algorithm runs in polynomial time. This occurs by the following facts. The algorithm `Conjunctive_Normal_Form_Horn_Clauses` has running time $O(n)$ because since every sentence of T , C and F is restricted to HF form (Definition 4) they can be transformed directly into Horn Clauses in $O(1)$. The execution of the Resolution mechanism of the PL sentences in CNF with Horn Clauses can be done in $O(n^2)$. Thus, `Consistent` and `Resolution` algorithms have running time $O(n^2)$. The algorithm `Formulate_Candidate_Hypotheses` has a running time of $O(n^{\lambda_2})$ because produces at most hypotheses combinations $O(n^2) + O(n^3) + \dots + O(n^{\lambda_2})$. The

algorithm `Remove_Inconsistent_Hypotheses` has a running time of $O(n^{2+\lambda_2})$ because executes at most a constant amount of $O(n^{\lambda_2})$ resolutions each of them in $O(n^2)$. The algorithm `Select_Good_Hypotheses` has a running time of $O(n \log n)$, to sort and select the set of hypotheses with minimal complexity among the hypotheses with maximum explanatory power.

3 Tendency of Solutions Computed by the Algorithm Peirce to Coincide with Abductive Reasoning Done by Humans

A study was realized to verify whether the abductive reasoning performed by humans tends to coincide with the solutions computed by the algorithm Peirce. The study was conducted using a questionnaire containing ten questions, with each question presenting an implicit description of a theory, observed facts and accepted conditions. The alternatives for each question present possible abductive hypotheses. Table 3 illustrates in the left column one question in the questionnaire.

Table 3. Example of a question used in the questionnaire. The left column describes the question itself, and the right column presents the corresponding formalization to the question and solution as computed by the algorithm Peirce.

Question	Formalization and solution computed by the algorithm Peirce
Joshua is in the desert and sees something green in the distance. What would best explain what Joshua sees? a) I am convinced that it is a lawn. b) I am convinced that it is a cactus. c) I am convinced that it is a green flag. d) It could be either a cactus or a green flag.	Propositions: ‘c: It is a cactus’, ‘d: It is a desert’, ‘f: It is a green flag’, ‘l: It is a lawn’, ‘s: Joshua sees something green’. $T = \{ l \rightarrow s, c \rightarrow s, f \rightarrow s \}, C = \{ d, d \rightarrow \neg l \}, F = \{ s \}.$ Solution - After formulating candidate hypotheses (line 11): $H = \{ l, c, f, l \wedge c, l \wedge f, c \wedge f, l \wedge c \wedge f \}.$ - After removing inconsistent hypotheses (line 12): $H = \{ c, f, c \wedge f \}.$ - After selecting good hypotheses (line 13): $H = \{ c, f \},$ i.e., the alternative ‘d’ coincides with the solution of algorithm Peirce.

The questionnaire, validated by a pilot-test with 25 individuals, was designed to be answered in 15 minutes. A total of 133 undergraduate and graduate students participated in the study. The profile of the participants showed a slight predominance of female individuals (53%) and ages ranging from 18 to 60 years, with an average and median close to 25 years.

Each participant’s answers to the questionnaire were computed, and one point was attributed to each answer on the questionnaire that coincided with a solution produced by the algorithm Peirce. The results showed an average of 86 answers coinciding with the algorithm Peirce and 47 that did not coincide.

The Chi-square (χ^2) test at 1% significance was used as a statistical measure of the significance with which the participants' answers coincided with solutions produced by the algorithm Peirce. For the studied population, the χ^2 test suggested that the coincidence between the participants' answers and the solutions computed by the algorithm Peirce was significant: ($\chi^2(1) = 11.44, p\text{-value} = 0.001 < 0.01$).

4 Related Works

Different approaches have been used to develop algorithms for abductive reasoning. Among the many contributions, there are proposals that use search techniques [15] and probabilistic reasoning over Bayesian Networks [7]. Logic approaches are based on two types of contributions: (1) proposal of new algorithms and (2) extension of traditional logical programming to process abductive reasoning problems.

Examples of type 1 contributions include [2] and [5]. Both of the proposals refer to abductive reasoning algorithms that operate with a THF reasoning framework (Definition 1). The main differences between these proposals and those of the present study are as follows: (1) They allow only one fact to be declared; (2) They do not allow define explicitly a set of accepted conditions; and (3) Semantic Tableaux is used in the proposal described in [2] instead of Resolution as the mechanism of inference.

Contributions of type 2 include Abductive Logic Programming (ALP) [11] and use the languages Prolog with Constraint Handling Rules (CHR) [1] [4]. The main differences between these proposals and those of the present study are as follows: (1) They operate with Predicate Logic; (2) They require special "abducible" predicates (possible hypotheses) to be declared; and (3) They dedicate little attention to criteria to select good abductive hypotheses.

Studies related to the one presented here, that address the selection of good hypotheses, include [8], [16] and, recently, [3]. This work differs from proposals [8] and [16] mainly by the proposed metrics for complexity and explanatory power of hypotheses.

5 Conclusions

The abductive reasoning algorithm Peirce is distinct from other solutions mainly because it employs the TCHF reasoning framework and a simple criterion for selecting good hypotheses that consider quantitative metrics to define the explanatory power and complexity of the formulated hypotheses.

The TCHF reasoning framework has shown itself to be useful in organizing the elements that participate in abductive reasoning because it does not "contaminate" the theory with sentences that fundamentally do not belong to the theory. This framework provides an additional advantage because it explicitly exposes the conditions (contexts, circumstances, intentions etc.) under which the reasoning process is conducted, which is fundamental and frequent in the formulation of abductive reasoning.

The criteria for selecting good hypotheses are subjects of ongoing research. There is no consensus as to which criteria should be used and under which circumstances or in which domains they work. The criterion used by the algorithm Peirce, which is described in Definition 6, attempt to produce a simple algorithm that works in practice. Alternatives to Definition 6 exist and can be proposed.

The study that depicted the coincidence of the solutions computed by the algorithm Peirce to those derived through abductive reasoning performed by humans was not exhaustive because there is such a high number of domains, and it did not include the diversity and quantity of individuals. However, these results provided value suggesting that the abductive reasoning conducted by humans tends to coincide with the solutions computed by the algorithm Peirce.

References

1. Alberti, M., Gavanelli, M., Lamma, E.: The CHR-based Implementation of the SCIFF Abductive System. *Fundamenta Informaticae* 124 (4), pp. 365–381 (2013)
2. Aliseda, A.: *Abductive Reasoning: Logical Investigations into Discovery and Explanation*. Springer, Netherlands (2006)
3. Caroprese, L., Trubitsyna, I., Truszczyński, M., Zumpano, E.: A Measure of Arbitrariness in Abductive Explanations. To appear in *Theory and Practice of Logic Programming*, 25 p (2014)
4. Christiansen, H.: Executable specifications for hypothesis-based reasoning with Prolog and Constraint Handling Rules. *Journal of Applied Logic* 7 (3), pp. 341–362 (2008)
5. Dillig, I., Dillig, T.: Explain: A Tool for Performing Abductive Inference. In Sharygina, N., Veith, H. (eds) *CAV 2013*. LNCS, vol. 8044, pp. 684–689. Springer, Heidelberg (2013)
6. Eiter, T., Gottlob, G.: The Complexity of Logic-Based Abduction. *Journal of the Association for Computing Machinery*, 42 (1), pp. 3–42 (1995)
7. Fortier, N., Sheppard, J., Strasser, S.: Abductive inference in Bayesian networks using distributed overlapping swarm intelligence. *Soft Computing Journal*, May 2014, pp. 1–21 (2014)
8. Harman, G. H.: The inference to the best explanation. *The Philosophical Review* 74 (1), pp. 88–95 (1965)
9. Howard, P.: *Introduction to Logic: Propositional Logic*. Prentice-Hall, New Jersey (1999)
10. Josephson, J. R., Josephson, S. G.: *Abductive Inference: Computation, Philosophy, Technology*. Cambridge University Press, Cambridge (1994)
11. Kakas, A. C., Kowalski, R. A., Toni, F.: Abductive Logic Programming. *Journal of Logic and Computation* 2 (6), pp. 719–770 (1995)
12. Mackenzie, I. S.: *Human-Computer Interaction: An Empirical Research Perspective*. Morgan Kaufmann, New York (2013)
13. Magnani, L.: *Abductive Cognition: The Epistemological and Eco-cognitive Dimensions of Hypothetical Reasoning*. Springer, Berlin (2009)
14. Peirce, C. S.: *Collected Papers of Charles Sanders Peirce*. Oxford University Press, London (1958)
15. Romdhane, L. B., Ayeb, B.: An Evolutionary Algorithm for Abductive Reasoning. *Journal of Experimental & Theoretical Artificial Intelligence* 23, pp. 529–544 (2011)

Felipe Rodrigues, Carlos Eduardo A. Oliveira, and Osvaldo Luiz de Oliveira

16. Thagard, P. R.: The Best Explanation: Criteria for Theory Choice. *The journal of philosophy* 75 (2), pp. 76–92 (1978)
17. Thagard, P. R.: Explanatory Coherence. *Behavioral and Brain Sciences* 12, pp. 435–502 (1989)

Apêndice III

Segundo artigo publicado

O artigo *Abd1: uma linguagem de paradigma lógico projetada especificamente para a programação de raciocínios abdutivos*, foi submetido e aceito no *XI Workshop de Computação na Faccamp* (WCF 2015), ISSN 2446-6808.

Abd1: uma linguagem de paradigma lógico projetada especificamente para a programação de raciocínios abduativos

Carlos Eduardo A. Oliveira, Osvaldo Luiz de Oliveira

Faculdade Campo Limpo Paulista (FACCAMP)
Rua Guatemala, 167, Jardim América – Campo Limpo Paulista – SP – Brasil
carlos.br@gmail.com, osvaldo@faccamp.br

Abstract. *Abduction in logic programming is a powerful technique to solve a range of problems in AI (Artificial Intelligence). However, existing approaches make use of general purpose logic programming systems and, as a result, require extensive knowledge of these systems by a programmer. This article proposes a language addressed to non-specialists in logic programming, designed for abductive reasoning programming.*

Resumo. *Abdução em programação lógica é uma técnica poderosa para resolver muitos problemas em IA (Inteligência Artificial). Entretanto, as abordagens existentes estão formuladas sobre sistemas de programação lógica que servem a diferentes propósitos e, com isto, requerem do programador vasto conhecimento destes sistemas. Este trabalho propõe uma linguagem projetada para a programação de raciocínios abduativos, dirigida a não especialistas em programação lógica.*

1. Introdução

Racocínios abduativos formulam possíveis hipóteses para explicar fatos observados tendo como fundamento uma teoria (Aliseda, 2006). Em Computação, a abdução em programação lógica tem se revelado uma poderosa técnica para resolver uma variedade de problemas em IA incluindo, entre outros, problemas de diagnóstico, entendimento de linguagem natural, planejamento e formulação de argumentos (Denecker & Kakas, 2000). Se por um lado é inquestionável a utilidade da abdução na modelagem da solução de problemas, por outro lado a programação destas soluções é difícil e demanda do programador conhecimentos profundos de lógica, linguagens de programação lógica e de mecanismos de resolução sistemas lógicos. O problema fundamental com as abordagens existentes é que elas estão formuladas sobre sistemas de programação lógica que servem a diferentes propósitos, não só para programar raciocínio abduativo (e.g., Christiansen, 2009 e Kakas, Michael & Mourlas, 2000). Este trabalho de dissertação de mestrado, em elaboração, propõe o desenvolvimento de uma linguagem de programação específica para a programação de raciocínios abduativos, que não demanda do programador conhecimentos de mecanismos de resolução de sistemas lógicos e conhecimentos linguísticos além dos associados à descrição de raciocínios abduativos. Esta Linguagem, Abd1, está sendo desenvolvida originalmente baseada em Lógica Proposicional e é descrita neste artigo.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta uma visão geral da Linguagem Abd1. A Seção 3 discute, a título de exemplo, um

programa em Abd1, escrito para modelar raciocínios abduativos sobre um circuito elétrico. A Seção 4 finaliza o artigo com considerações sobre o possível uso da Abd1.

2. Visão Geral da Linguagem Abd1

O projeto da Linguagem Abd1 é motivado pela estrutura de raciocínio TCHF, proposta em Rodrigues, Oliveira & Oliveira (2014). Esta estrutura define um sistema de raciocínio em que sentenças lógicas são descritas por meio de conjuntos Teoria (T), Condições (C), Hipóteses (H) e Fatos (F). Realizar um raciocínio abduativo no sistema TCHF é calcular um conjunto de hipóteses H que, juntamente com os conjuntos teoria T e condições C explicam um conjunto de Fatos F. A definição formal do sistema de raciocínio TCHF pode ser encontrada em Rodrigues, Oliveira & Oliveira (2014).

Um programa descrito na Linguagem Abd1 é composto por teorias, condições e fatos. Teorias descrevem conhecimentos sobre um assunto. Na Linguagem Abd1 teorias podem ser descritas por meio de expressões que iniciam com o termo *Theory* seguido por um nome atribuído à teoria e por um conjunto de sentenças em Lógica Proposicional escritas em uma forma denominada HF, definida em Rodrigues (2015). Embora a forma HF imponha restrições à escrita de sentenças da Lógica Proposicional, ela é bastante abrangente nas construções sintáticas. Conjuntos de condições são empregados para estabelecer condições específicas (contextos, circunstâncias, intenções entre outras) para a realização de um raciocínio abduativo sem que teorias tenham que ser alteradas para incorporar tais elementos especializados. Uma condição é declarada pelo termo *Conditions* seguido por um nome atribuído à condição e, tal como em declarações de teorias, por um conjunto de sentenças da Lógica Proposicional na forma HF. Conjuntos de fatos servem para descrever, como o próprio termo sugere, fatos, observações, sinais ou sintomas que devem ser explicados. Na Linguagem Abd1 fatos são descritos por meio de expressões que iniciam com o termo *Facts* seguido por um nome atribuído ao fato e por um conjunto de proposições.

Um raciocínio abduativo é descrito pelo termo *AbductiveReasoning* seguido por um nome atribuído ao raciocínio e pela declaração dos elementos (teorias, fatos e condições), incluídos entre parênteses angulares (“(” e “)”), que participarão do raciocínio. A Linguagem admite, na descrição de um raciocínio abduativo, a escrita de expressões contendo a união, a intersecção ou a subtração de teorias, de condições e de fatos. É possível também especificar, opcionalmente, critérios para seleção de boas hipóteses abduativas. Por carência de espaço, este artigo não aborda este aspecto. O resultado da execução de um raciocínio abduativo é um conjunto de hipóteses. Cada hipótese dentro do conjunto de hipóteses é uma possível explicação para o raciocínio.

3. Exemplo de um Programa Escrito na Linguagem Abd1

Considere o circuito constituído por duas lâmpadas, dois interruptores, uma bateria e fios, cujo diagrama elétrico está ilustrado na Figura 1. Neste circuito a lâmpada 2 acende se a bateria possuir carga, os fios estiverem conectados e o interruptor 1 estiver ligado. Para que a lâmpada 1 acenda é necessário, adicionalmente que o interruptor 2 esteja ligado.

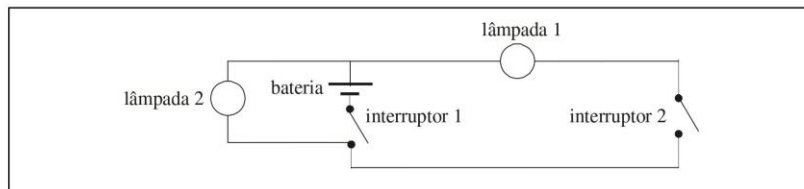


Figura 1. Diagrama elétrico de um circuito que é utilizado como cenário para exemplificar o uso da Linguagem Abd1.

A Figura 2 apresenta um programa que estabelece um modelo sobre o funcionamento do circuito. Duas teorias estão sendo especificadas. A teoria denominada *Teoria_Não_Funcionamento* especifica, por meio de quatro sentenças (linhas 3 a 6), falhas e funcionalidades que fazem as lâmpadas do circuito estarem apagadas. Por exemplo, a sentença da linha 3 estabelece que “se *bateria_sem_carga* então *lâmpada_1_apagada* e *lâmpada_2_apagada*”. As demais sentenças desta teoria estabelecem relações lógicas entre as proposições *fios_rompidos*, *interruptor_1_desligado*, *interruptor_2_desligado*, *lâmpada_1_apagada* e *lâmpada_2_apagada*. A teoria *Teoria_Funcionamento* (linhas 9 a 12) especifica, em duas sentenças, o que faz as lâmpadas do circuito estarem acesas. Por exemplo, a sentença da linha 11 estabelece que “se *bateria_com_carga* e *fios_conectados* e *interruptor_1_ligado* então *lâmpada_2_acesa*”. Há uma particularidade em relação ao que estabelecem estas duas teorias que é importante ressaltar: elas modelam, por meio das proposições *fios_rompidos* (linha 4) e *fios_conectados* (linhas 10 e 11), os fios do circuito como se fossem um bloco único, isto significa que os fios não estão sendo tratados como diversos segmentos independentes.

As duas teorias descritas explicam diferentes fenômenos que ocorrem no circuito. Estas teorias podem ser usadas isoladamente na realização de um raciocínio abduutivo, mas para que possam ser utilizadas conjuntamente a condição *Condição_Correlação_Teorias* está sendo especificada (linhas 14 a 17). Basicamente, esta condição estabelece relações lógicas entre as proposições utilizadas nas duas teorias. Assim, por exemplo, uma sentença na linha 15 estabelece que a proposição *bateria_com_carga* da teoria *Teoria_Funcionamento* possui a seguinte relação lógica com a proposição *bateria_sem_carga* da teoria *Teoria_Não_Funcionamento*: “se *bateria_com_carga* então não *bateria_sem_carga*”. A condição *Condição_Interruptores_Não_Desligados* (linha 19) estabelece um contexto especial para realização de raciocínios com a teoria *Teoria_Não_Funcionamento*. Esta condição estabelece que os interruptores 1 e 2 não estão desligados. Um fato é descrito na linha 21. O fato denominado *Fato_1* é um conjunto unitário que especifica *lâmpada_1_apagada*.

Raciocínios abdutivos estão sendo descritos nas linhas 23 a 28. *Explicações_Lâmpada_1_Apagada* especifica um raciocínio para encontrar hipóteses que explicam o *Fato_1* tendo como teoria a *Teoria_Não_Funcionamento*. *Explicações_Lâmpada_2_Apagada* especifica algo semelhante ao primeiro raciocínio. As diferenças desta especificação para a primeira são: (1) em vez de referenciar um fato já nomeado no programa, o conjunto unitário de fatos { *lâmpada_2_apagada* } está

sendo diretamente descrito na especificação do raciocínio; (2) objetiva-se obter hipóteses que expliquem o fato lâmpada_2_apagada. Explicações_Lâmpada_1_Apagada_Lâmpada_2_Acesa especifica um raciocínio para encontrar hipóteses que explicam fatos presentes nas duas teorias definidas. Para isto, estão sendo descritas uma expressão para especificar a união das duas teorias e uma condição para correlação lógica entre elas. Outra particularidade da especificação deste raciocínio é o uso de uma expressão para descrever a união de dois conjuntos de fatos.

```

1. // Teoria sobre falhas e funcionalidades que fazem as lâmpadas do circuito estarem apagadas.
2. Theory Teoria_Não_Funcionamento {
3.   bateria_sem_carga → lâmpada_1_apagada ∧ lâmpada_2_apagada,
4.   fios_rompidos → lâmpada_1_apagada ∧ lâmpada_2_apagada,
5.   interruptor_1_desligado → lâmpada_1_apagada ∧ lâmpada_2_apagada,
6.   interruptor_2_desligado → lâmpada_1_apagada
7. }
8. // Teoria sobre o que faz as lâmpadas do circuito estarem acesas.
9. Theory Teoria_Funcionamento {
10.  bateria_com_carga ∧ fios_conectados ∧ interruptor_1_ligado ∧ interruptor_2_ligado → lâmpada_1_acesa,
11.  bateria_com_carga ∧ fios_conectados ∧ interruptor_1_ligado → lâmpada_2_acesa
12. }

13. // Condições para correlacionar as teorias Teoria_Não_Funcionamento e Teoria_Funcionamento.
14. Conditions Condição_Correlação_Teorias {
15.  bateria_com_carga → ¬ bateria_sem_carga, interruptor_1_desligado → ¬ interruptor_1_ligado,
16.  interruptor_2_desligado → ¬ interruptor_2_ligado, fios_rompidos → ¬ fios_conectados
17. }
18. // Condições que especificam que os interruptores não estão desligados.
19. Conditions Condição_Interruptores_Não_Desligados { ¬ interruptor_1_desligado, ¬ interruptor_2_desligado }

20. // Um conjunto de fatos.
21. Facts Fato_1 { lâmpada_1_apagada }

22. // Alguns raciocínios abduativos.
23. AbductiveReasoning Explicações_Lâmpada_1_Apagada ( Teoria_Não_Funcionamento, Fato_1 )
24. AbductiveReasoning Explicações_Lâmpada_2_Apagada ( Teoria_Não_Funcionamento, { lâmpada_2_apagada } )
25. AbductiveReasoning Explicações_Lâmpada_1_Apagada_Lâmpada_2_Acesa {
26.  Teoria_Não_Funcionamento ∪ Teoria_Funcionamento, { lâmpada_1_apagada } ∪ { lâmpada_2_acesa },
27.  Condição_Correlação_Teorias
28. }

```

Figura 2. Programa estabelecendo um modelo sobre o funcionamento do circuito da Figura 1.

A Figura 3 apresenta quatro exemplos de execução do programa da Figura 2 por meio de um interpretador da Linguagem Abdl, em fase de desenvolvimento. O interpretador sinaliza, com o sinal de pronto “>” na linha de comando, que aguarda a digitação de uma expressão descrevendo a união, a interseção e a subtração de raciocínios abduativos. Em resposta à expressão digitada, o interpretador executa o raciocínio e apresenta como resposta um conjunto de hipóteses para explicar os fatos definidos no raciocínio. A Figura 3(a) ilustra a execução do raciocínio Explicações_Lâmpada_1_Apagada. A Figura 3(b) exemplifica uma interseção de dois raciocínios, cujo resultado descreve as hipóteses que explicam, concomitantemente, os fatos das lâmpadas 1 e 2 estarem apagadas. A Figura 3(c) exemplifica o emprego de uma condição na realização de um raciocínio. Este exemplo, descreve um raciocínio para buscar hipóteses que explicam os fatos das lâmpadas 1 e 2 estarem apagadas, tendo

como condição para realização do raciocínio os interruptores 1 e 2 não estarem desligados. Uma particularidade presente neste exemplo é a descrição direta dos elementos que compõem um raciocínio (teoria, fatos e condição) na linha de comando do interpretador. O exemplo da Figura 3(d) mostra a execução de um raciocínio envolvendo conjuntamente as duas teorias definidas no programa. O raciocínio objetiva encontrar hipóteses que explicam os fatos lâmpada_1_apagada e lâmpada_2_acesa.

<pre>> Explicações Lâmpada_1_Apagada. > { bateria_sem_carga, fios_rompidos, interruptor_1_desligado, interruptor_2_desligado }</pre> (a)	<pre>> Explicações Lâmpada_1_Apagada $\cap$ Explicações Lâmpada_2_Apagada. > { bateria_sem_carga, fios_rompidos, interruptor_1_desligado }</pre> (b)
<pre>> < Teoria_Não_Funcionamento, { lâmpada_1_apagada, lâmpada_2_apagada }, Condição_Interruptores_Não_Desligados >. > { bateria_sem_carga, fios_rompidos }</pre> (c)	<pre>> Explicações Lâmpada_1_Apagada Lâmpada_2_Acesa. > { bateria_com_carga $\wedge$ fios_conectados $\wedge$ interruptor_1_ligado $\wedge$ interruptor_2_desligado }</pre> (d)

Figura 3. Exemplos de execução do programa da Figura 2.

4. Considerações Finais

A Linguagem Abd1 é uma possível resposta ao problema do alto nível de conhecimento exigido de um programador para modelar raciocínios abduativos utilizando os sistemas de programação lógica existentes. Sendo projetada especificamente para a programação de raciocínios abduativos, espera-se que Abd1 possa ser utilizada por não especialistas em programação lógica como, por exemplo, engenheiros interessados em desenvolver modelos para diagnóstico de falhas em máquinas ou estudantes interessados em aprender sobre um assunto por meio da atividade de construir modelos sobre ele. A segmentação do modelo em teoria, condições, hipóteses e fatos proposta na Linguagem Abd1 permite uma “modularização” da base de conhecimento (programa lógico), o que contribui para diminuir o esforço cognitivo do programador na programação, à medida em que organiza o programa e facilita a reutilização e modificação do “código fonte”.

Referências

- Aliseda, A. (2006) “Abductive Reasoning: logical investigations into discovery and explanation”, Springer, The Netherlands.
- Christiansen, H. (2009) “Executable specifications for hypothesis-based reasoning with Prolog and Constraint Handling Rules”, *Journal of Applied Logic*, v. 7, pp. 341-366.
- Denecker, M. & Kakas, A. C. (2000) (eds.), “Abductive logic programming” Special issue of the *Journal of Logic Programming*, v. 44, pp. 1-3.
- Kakas, A. C., Michael, A. & Mourlas, C. (2000) “ACLP: Abductive Constraint Logic Programming” *The Journal of Logic Programming*, v. 44, pp. 129-177.
- Rodrigues, F., Oliveira, C. E. A. & Oliveira, O. L. (2014) “Peirce: an algorithm for abductive reasoning operating with a quaternary reasoning framework”, *Research in Computer Science*, v. 82, pp. 53-66.
- Rodrigues, F. (2015) “Um algoritmo para abdução peirceana baseado em uma estrutura de raciocínio quaternária em Lógica”, *Dissertação de Mestrado, Faccamp*, [on-line], Disponível em http://www.cc.faccamp.br/Dissertacoes/Felipe_2015.pdf.

Apêndice IV

Formulário utilizado na pesquisa experimental de usabilidade da Linguagem Abd1

Este apêndice apresenta integralmente o formulário utilizado na pesquisa experimental de usabilidade da Linguagem Abd1. O formulário está dividido em quatro partes. A Parte 1 declara para o participante o propósito do experimento e permite que ele declare o seu perfil, i.e., sexo, idade, curso de graduação feito e conhecimentos gerais sobre Lógica e sobre Programação. A Parte 2 é uma apresentação da Linguagem Abd1 por meio de exemplos de programas escritos nesta linguagem. Pretende-se com a Parte 2 um treinamento na Linguagem Abd1 bem como que o material da Parte 2 possa servir de referência para o participante durante os exercícios propostos na Parte 3. A Parte 3, portanto, contém exercícios para serem realizados pelos participantes. Os exercícios são propostas de programas para serem escritos usando Abd1. A Parte 4 permite que o participante declare a sua satisfação subjetiva em usar a Linguagem Abd1 para escrever programas para realizar raciocínio abduutivo. Adicionalmente, a Parte 4 possibilita que o participante atribua notas para o ambiente AIDEP-Abd1 e para a organização da sessão experimental como forma de possibilitar aos pesquisadores inferir o quanto falhas de software e de organização possam afetar a pesquisa de usabilidade da Linguagem Abd1.

Faculdade Campo Limpo Paulista
Programa de Mestrado em Ciência da Computação
Experimento de Usabilidade da Linguagem Abd1

Parte 1

Propósito do experimento e perfil do participante

Código: _____

Propósito do experimento

Abdução é um tipo de raciocínio que tem como objetivo formular possíveis hipóteses para explicar fatos observados, considerando uma teoria como fundamento. A Linguagem Abd1 permite programar raciocínios abduativos e este experimento objetiva medir o quanto ela é fácil ou difícil de usar.

Perfil do participante

Sexo: ☐ Masculino ☐ Feminino

Idade: _____ anos.

Qual curso de graduação você fez?

Você já escreveu algum programa de computador?

☐ Sim ☐ Não

Você se sente capaz de programar em alguma linguagem de programação?

☐ Sim ☐ Não

Estudou Lógica Proposicional alguma vez?

☐ Sim ☐ Não

Estudou Prolog alguma vez?

☐ Sim ☐ Não

Você se sente capaz de programar em Prolog?

☐ Sim ☐ Não

Escreve programas em Prolog atualmente?

☐ Sim ☐ Não

Faculdade Campo Limpo Paulista
Programa de Mestrado em Ciência da Computação
Experimento de Usabilidade da Linguagem Abd1

Parte 2
Apresentação da Linguagem Abd1 por meio de exemplos e interação dos pesquisadores para treinamento na Linguagem Abd1

Exemplo I:

Texto do programa (Arquivo Gramado1.ab1 do diretório Exemplos)

```
Theory teoriaGramado
{
  chuva → gramadoMolhado
}

AbductiveReasoning explicacoesGramadoMolhado
<
  teoriaGramado,
  { gramadoMolhado }
>
```

Execução (Caixa de texto <i>Question</i>)	Resultado (Caixa de texto <i>Results</i>)
explicacoesGramadoMolhado.	{ chuva }

Exemplo II:

Texto do programa (Arquivo Gramado2.ab1 do diretório Exemplos)

```
Theory teoriaGramado
{
  chuva → gramadoMolhado,
  irrigadoresLigados → gramadoMolhado,
  tanqueVazio → ¬irrigadoresLigados
}

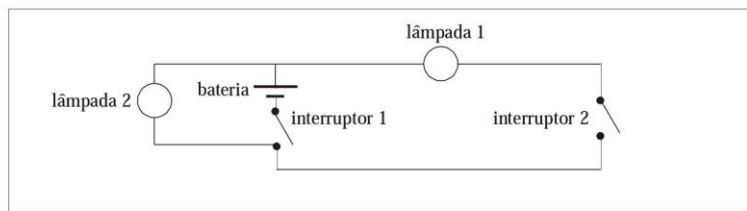
Conditions tanqueSemAgua
{
  tanqueVazio
}

AbductiveReasoning explicacoesGramadoMolhado1
<
  teoriaGramado,
  { gramadoMolhado }
>

AbductiveReasoning explicacoesGramadoMolhado2
<
  teoriaGramado,
  { gramadoMolhado },
  tanqueSemAgua
>
```

Execução (Caixa de texto <i>Question</i>)	Resultado (Caixa de texto <i>Results</i>)
explicacoesGramadoMolhado1.	{ chuva, irrigadoresLigados }
explicacoesGramadoMolhado2.	{ chuva }

Exemplo III:



Texto do programa (Arquivo `Circuito.ab1` do diretório `Exemplos`)

```
// Teoria sobre falhas e funcionalidades que fazem as lâmpadas do circuito estarem apagadas.
Theory teoriaNaoFuncionamento
{
  bateriaSemCarga → lâmpada1Apagada ∧ lâmpada2Apagada,
  fiosRompidos → lâmpada1Apagada ∧ lâmpada2Apagada,
  interruptor1Desligado → lâmpada1Apagada ∧ lâmpada2Apagada,
  interruptor2Desligado → lâmpada1Apagada
}

// Teoria sobre o que faz as lâmpadas do circuito estarem acesas.
Theory teoriaFuncionamento
{
  bateriaComCarga ∧ fiosConectados ∧ interruptor1Ligado ∧ interruptor2Ligado →
  lâmpada1Acesa,
  bateriaComCarga ∧ fiosConectados ∧ interruptor1Ligado → lâmpada2Acesa
}

// Condições para correlacionar as teorias teoriaNaoFuncionamento e teoriaFuncionamento.
Conditions condicaoCorrelacaoTeorias
{
  bateriaComCarga → ¬ bateriaSemCarga, interruptor1Desligado → ¬ interruptor1Ligado,
  interruptor2Desligado → ¬ interruptor2Ligado, fiosRompidos → ¬ fiosConectados
}

// Alguns conjuntos de fatos.
Facts fato1 { lâmpada1Apagada }
Facts lâmpada1AcesaLâmpada2Apagada { lâmpada1Acesa, lâmpada2Apagada }

// Alguns raciocínios abduativos.
AbductiveReasoning explicacoesLampada1Apagada
<
  teoriaNaoFuncionamento,
  fato1
>

AbductiveReasoning explicacoesLampada2Apagada
```

```

<
  teoriaNaoFuncionamento,
  { lampada2Apagada }
>

AbductiveReasoning explicacoesLampada1ApagadaLampada2Acesa
<
  teoriaNaoFuncionamento U teoriaFuncionamento,
  { lampada1Apagada, lampada2Acesa },
  condicaoCorrelacaoTeorias
>

AbductiveReasoning explicacoesLampada1AcesaLampada2Apagada
<
  teoriaNaoFuncionamento U teoriaFuncionamento,
  lampada1AcesaLampada2Apagada,
  condicaoCorrelacaoTeorias
>

```

Execução (Caixa de texto <i>Question</i>)	Resultado (Caixa de texto <i>Results</i>)
explicacoesLampada1Apagada.	{ bateriaSemCarga, fiosRompidos, interruptor1Desligado, interruptor2Desligado }
explicacoesLampada2Apagada.	{ bateriaSemCarga, fiosRompidos, interruptor1Desligado }
explicacoesLampada1Apagada $\cap$ explicacoesLampada2Apagada.	{ bateriaSemCarga, fiosRompidos, interruptor1Desligado }
explicaçõesLampada1ApagadaLampada2Acesa.	{ bateriaComCarga $\wedge$ fiosConectados $\wedge$ interruptor1Lgado $\wedge$ interruptor2Desligado }
explicacoesLampada1AcesaLampada2Apagada.	{ }

■

Faculdade Campo Limpo Paulista
Programa de Mestrado em Ciência da Computação
Experimento de Usabilidade da Linguagem Abd1

Parte 3

Exercícios sobre a Linguagem Abd1 sem a interação dos pesquisadores

Código: _____

Exercício I: Considere novamente o circuito e o programa do Exemplo 3. O texto original do programa está no arquivo `Circuito.ab1` que está no diretório `Exercicios`. O problema consiste em acrescentar à teoria `teoriaNaoFuncionamento`:

- A proposição `lampada1Queimada` para que lâmpada 1 “queimada” também seja uma explicação para a lâmpada 1 estar apagada.
- A proposição `lampada2Queimada` para que lâmpada 2 “queimada” também seja uma explicação para a lâmpada 2 estar apagada.

1. Salve o seu programa no diretório `Exercicios` com o nome `Exercicio_I`.
2. Execute o raciocínio `explicacoesLampada2Apagada`. Qual foi o resultado da execução?

3. Execute o raciocínio `explicacoesLampada1AcesaLampada2Apagada`. Qual foi o resultado da execução?

Exercício II: Este exercício propõe modelar na Linguagem Abdl uma interpretação da conhecida teoria econômica da oferta e demanda.

1. Crie uma teoria chamada `ofertaEDemanda` com as seguintes sentenças:
 - Se `ofertaAlta` então `precoBaixo`.
 - Se `ofertaBaixa` então `precoAlto`.
 - Se `demandaAlta` então `precoAlto`.
 - Se `demandaBaixa` então `precoBaixo`.
 - `ofertaBaixa` é o contrário de `ofertaAlta`.
 - `demandaBaixa` é o contrário de `demandaAlta`.
2. Crie um raciocínio abduativo chamado `porQuePrecoBaixo1` que usa a teoria `ofertaEDemanda` para achar explicações para `precoBaixo`. Execute este raciocínio abduativo. Qual foi o resultado que você encontrou?

3. Crie uma condição denominada `escassez` caracterizada pela presença de `ofertaBaixa`.
4. Crie um raciocínio abduativo chamado `porQuePrecoBaixo2` que usa a teoria `ofertaEDemanda` para achar explicações para `precoBaixo` na condição de `escassez`. Execute este raciocínio abduativo. Qual foi o resultado que você encontrou?

5. Salve o seu programa no diretório `Exercicios` com o nome `Exercicio_II`.

Faculdade Campo Limpo Paulista
Programa de Mestrado em Ciência da Computação
Experimento de Usabilidade da Linguagem Abd1

Parte 4
Avaliação da satisfação subjetiva e Debriefing

Código: _____

Em uma escala de 1 a 5 onde:

- 1 – Muito difícil
- 2 – Difícil
- 3 – Regular
- 4 – Fácil
- 5 – Muito Fácil

1. Dê uma nota para o nível de facilidade em compreender a exposição da Linguagem Abd1.
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
2. Dê uma nota para o nível de facilidade dos exercícios feitos.
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
3. Dê uma nota para a sua percepção sobre a facilidade de usar a Linguagem Abd1 para modelar fenômenos.
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
4. Dê uma nota para o quanto o software (não a Linguagem Abd1) é fácil para editar, compilar, exibir mensagens de erro e executar raciocínios.
☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5
5. (Opcional): Você gostaria de escrever algo para os pesquisadores?

Referências

- Abdennadher, S., Christiansen, H. (2000). An experimental CLP platform for integrity constraints and abduction. In *Proceedings of the Flexible Query Answering Systems: Advances in Soft Computing Series*, FQAS 2000, pp. 141–152.
- Andrade, C. D (1984). Lira Itabirana. *Jornal O Cometa Itabirano*.
- Aho, A. V., Lam, M. S., Sethi, R. & Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools*, 2nd Edition. Boston: Addison-Wesley.
- Brasil (2015). *Código Penal Brasileiro*. Consultado em dezembro de 2015. [on-line] Disponível em “http://www.planalto.gov.br/ccivil_03/decreto-lei/Del2848compilado.htm”.
- Bramer, M (2013). *Logic Programming with Prolog*, 2nd Edition. London: Springer.
- Boslaugh, S. (2012). *Statistics in a Nutshell*, 2nd Edition. O’Reilly Media.
- Caroprese, L., Trubitsyna, I. (2014). A Measure of Arbitrariness in Abductive Explanations. *Theory and Practice of Logic Programming*, v. 14, n. 4-5.
- CHR Bibliography (2015). *CHR Bibliography Repository*. Consultado em dezembro de 2015. Disponível em <https://dtai.cs.kuleuven.be/CHR/biblio.shtml>.
- Dillig, I. & Dillig, T. (2013). Explain: a tool for performing abductive inference. In: Sharygina, N., Veith, H. (2013), eds. CAV 2013. *Lecture Notes in Computer Science*, Heidelberg: Springer, v. 8044, pp. 684-689.
- Denecker, M. & Kakas, A. (2002). Abduction in Logic Programming. In: Kakas, A. C. & Sadri, F. (2002), eds. *Computational Logic: logic programming and beyond: essays*. In honour of Robert A. Kowalski Part I. Berlin: Springer-Verlag, v. 2407, n. 1, pp. 402-436
- Duck, G. J., Stuckey, P. J., Banda, M. J. G., Holzbaur, C. (2004). The refined operational semantics of Constraint Handling Rules. In *Proceedings of the 20th International Conference on Logic Programming*, ICLP 2004, pp. 90–104.

- Fortier, Sheppard & Strasser (2015). Abductive inference in Bayesian networks using distributed overlapping swarm intelligence. *Soft Computing*, v. 19, n. 4, pp. 981-1001.
- Frühwirth, T. (1998). Theory and practice of constraint handling rules. *The Journal of Logic Programming*, v. 37, n. 3, pp. 95–138.
- Gilat, A. (2014). *MATLAB: An Introduction with Applications*. 5nd ed.. New Jersey: Wiley.
- Harman, G. H. (1965). The inference to the best explanation. *The Philosophical Review*, v. 74, n. 1, pp. 88-95.
- Hopcroft, J. E., Motwani & Ullman, J. (2006). *Introduction to Automata Theory, Languages and Computation*. 3rd ed. New York: Addison-Wesley.
- Howard, P. (1999). *Introduction to Logic: propositional logic*. New Jersey: Prentice-Hall.
- Josephson, J. R. & Josephson, S. G. (1994). *Abductive Inference: computation, philosophy, technology*. Cambridge: Cambridge University Press.
- Kakas, A. C., Kowalski, R. A. & Toni, F. (1992). Abductive logic programming. *Journal of Logic and Computation*, v. 2, n. 6, pp. 719-770.
- Kakas, Michael & Mourlas (2000). ACLP: Abductive Constraint Logic Programming. *Journal of Logic Programming*, v. 44, pp. 129-177.
- Lazar, J., Feng, J. H. & Hochheiser, H. (2010). *Research Methods in Human-Computer Interaction*. London: John Wiley & Sons.
- Mackenzie, I. S. (2013). *Human-computer Interaction: an empirical research perspective*. New York: Morgan Kaufmann.
- Magnani, L. (2009) Abductive Cognition: the epistemological and eco-cognitive dimensions of hypothetical reasoning. *Cognitive systems monographs*, v. 3, Berlin: Springer-Verlag.
- Marji, M. (2014). *Learn to Program with Scratch: A Visual Introduction to Programming with Games, Art, Science, and Math*. New York: William Pollock Publisher.

- Microsoft (2015). *Microsoft .NET framework*. Consultado em dezembro de 2015. [on-line] Disponível em <https://www.microsoft.com/net>.
- Papert, S. (1993). *Mindstorms: Children, Computers, And Powerful Ideas*. 2nd ed.. New York: Basic Books.
- Peirce, C. S. (1867) On The Natural Classification of Arguments. *Proceedings of the American Academy of Arts and Sciences*, v. 7, pp. 261–287. Reprinted at *Collected Papers*, v. 2, paragraphs 461–516 and *Writings* v. 2, pp. 23–49.
- Pospel, H. (2002). *Introduction to Logic: predicate logic*, 2nd Edition, New York: Pearson.
- Rodrigues, F., Oliveira, C. E. A. & Oliveira, O. L. (2014). Peirce: an algorithm for abductive reasoning operating with a quaternary reasoning framework, *Research in Computing Science*, v. 82, pp. 53-66.
- Rodrigues, F. (2015). *Um algoritmo para abdução peirceana baseado em uma estrutura de raciocínio quaternária em Lógica Proposicional*, Dissertação de Mestrado, Faccamp, [on-line], Disponível em http://www.cc.faccamp.br/Dissertacoes/Felipe_2015.pdf.
- Romdhane, L. B. & Ayeb, B. (2011). An Evolutionary Algorithm For Abductive Reasoning. *Journal of Experimental & Theoretical Artificial Intelligence*, v. 23, n. 4, pp. 529-544.
- Russell, S. & Norvig, P. (2014). *Artificial Intelligence. A Modern Approach*, 3st Edition, Prentice Hall, Englewood Cliffs, New Jersey, pp 471 – 497.
- SICStus Prolog (2015). *SICStus Prolog Homepage*. Consultado em dezembro de 2015. [on-line]. Disponível em <https://sicstus.sics.se/>.
- Sokal, R. R. & Rohlf, F. J. (1994). *Biometry: The Principles and Practices of Statistics in Biological Research*, 3st Edition, W. H. Freeman.
- SWI-Prolog (2015). *SWI-Prolog Organization*. Consultado em Dezembro de 2015. [on-line]. Disponível em <http://www.swi-prolog.org/>.
- Thagard, P. R. (1978). The Best Explanation: criteria for theory choice. *The Journal of Philosophy*, v. 75, n. 2, pp. 76-92.

Thagard, P. R. (1989). Explanatory Coherence. *Behavioral and Brain Sciences*, v. 12, n. 3, pp. 435-502.