Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Provas de Teoremas em Lógica Três-Valorada

Matheus B. de Castro

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof.a Dr.a Cláudia Nalon

Brasília
2017

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Provas de Teoremas em Lógica Três-Valorada

Matheus B. de Castro

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.a Dr.a Cláudia Nalon (Orientadora)
CIC/UnB

Prof. Dr. Flávio L. C. de Moura
CIC/UnB

Prof.a Dr.a Carla Denise Castanho
CIC/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
Coordenador do Bacharelado em Ciência da Computação

Brasília, 31 de outubro de 2017

Dedicatória

Dedico esse trabalho à minha família, por sua capacidade de acreditar e investir em mim. Mãe, seu cuidado e dedicação foi o que me deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

Agradecimentos

Gostaria de agradecer primeiramente à minha orientadora Prof.a Cláudia Nalon por ter sido um anjo que caiu do céu para me ajudar no momento que eu achei que já estava tudo perdido; aos meus amigos e colegas que conheci na UnB e que vou levar pra vida inteira e que, sem eles ao meu lado durante essa longa caminhada, eu não teria chegado tão longe; e, finalmente, aos excelentes professores com quem tive a oportunidade de aprender tanto durante esses sete anos.

Resumo

Existem poucas ferramentas disponíveis atualmente capazes de avaliar fórmulas em Lógica Três-Valorada. Este trabalho teve como objetivo a codificação de um provador de teoremas para Lógica Três-Valorada usando a Tradução Bivalente e Resolução Clausal formulados em [1]. O provador foi submetido a uma série de testes de desempenho e comparado com outro provador construído em linguagem Prolog, o MUltseq [2]. Os desempenhos de ambos foram comparados tanto quanto ao tempo consumido quando ao uso de memória RAM. São apresentados os fundamentos da Lógica infinitamente-valorada de Łukasiewicz, as definições para a formulação da Lógica Três-Valorada, a definição de Tradução Bivalente e o método de Resolução Clausal. Em seguida, explica-se como foi feita a implementação do programa tradutor proposto, as ferramentas e programas utilizados, a saída esperada, a realização dos testes de comparação e, por fim, a verificação de um melhor desempenho e de um melhor uso de memória do provador construído em relação ao MUltseq.

Palavras-chave: provas, teoremas, lógicas multivaloradas, três-valorada, lógica bivalorada, resolução clausal, formal normal, tradução bivalente, provador, forma normal conjuntiva, multseq, minisat

Abstract

To the best of our knowledge, there are very few tools currently available that can evaluate formulas in Three-Valued Logic. This work provides an implementation of a prover for Three-Valued Logic using the Bivalent Translation and the Clausal Resolution formulated in [1]. Our prover has undergone a series of performance tests, in comparison with another prover built in Prolog language, the MUltseq [2]. The performance of both provers took in consideration their efficiency in terms of runtime and RAM memory usage. We present the foundations of the infinitely-valued Logic of Łukasiewicz, the definitions for the formulation of Three-Valued Logic, the definition of Bivalent Translation and the method of Clausal Resolution for this logic. We then explain how the implementation of the proposed prover was done, the tools and programs used, the expected output, the realization of comparison tests and, finally, the verification of a better performance and better memory usage of the our prover in comparison to MUltseq.

Keywords: proofs, theorems, many-valued logics, three-valued, two-valued logic, clausal resolution, bivalent translation, prover, conjunctive normal form, multseq, minisat

Sumário

1 Introdução **1**
1.1 Contextualização 1
1.2 Problema 2
1.3 Objetivo 3
1.4 Descrição dos Capítulos 3

2 Lógica Três-Valorada **4**
2.1 Semântica 4
2.2 Lógica de Łukasiewicz 5
2.3 Lógica Três-Valorada de Łukasiewicz 6
2.4 Tabelas Verdade 6

3 Tradução Bivalente **9**
3.1 Declarações Bivalentes 9

4 Cálculo **12**
4.1 Forma Normal 12
4.2 Regras de Inferência 14

5 Implementação **16**
5.1 Ferramentas Utilizadas 16
5.2 Tokenizer 16
5.3 Gerador de Cláusulas 18
5.4 Provador 18
5.5 MUltseq 19

6 Avaliação Experimental **20**
6.1 Gerador de Fórmulas 20
6.2 Ambiente de Execução 21
6.3 Comparação de Resultados 21

6.4 Script de Execução 21
6.5 Testes 22
6.6 Resultados Obtidos 22

7 Conclusões **25**

Referências **27**

Lista de Figuras

4.1 Regras de Resolução para $\mathsf{RES}_{\mathcal{B}(\mathcal{L},\overrightarrow{\theta})}$ 15

6.1 Fluxograma de Execução 20
6.2 Gráfico: Tempo Médio de Execução 23
6.3 Gráfico: Fórmulas Executadas sem Timeout 23
6.4 Gráfico: Tempo Total de Execução 23

Lista de Tabelas

2.1 $\neg\,(\varphi)$ 7
2.2 $\wedge\,(\varphi, \psi)$ 7
2.3 $\vee\,(\varphi, \psi)$ 7
2.4 $\supset\,(\varphi, \psi)$ 7

3.1 Conjunto de Declarações Bivalentes para Ł$_3$ 11

Capítulo 1

Introdução

1.1 Contextualização

Lógica é a parte da filosofia que trata das formas do pensamento em geral (dedução, indução, hipótese e inferência) e das operações intelectuais que visam a determinação do que é verdadeiro ou não. Existem inúmeras classes diferentes de Lógica, mas a mais estudada e mais utilizada é a Lógica Clássica [12]. A Lógica Clássica, ou Lógica Bivalente, possui esse nome porque sua semântica admite apenas dois valores possíveis para a avaliação de uma fórmula: ou uma sentença é *verdadeira* ou ela é *falsa*. Por exemplo, considerado o atual estado de fatos, a sentença "Os Beatles foram uma banda de rock" é verdadeira, mas a sentença "O Brasil é uma monarquia" é falsa.

A Lógica Clássica é caracterizada por várias propriedades básicas. Uma delas é a *Lei do Terceiro Excluído* [12] que afirma que, para qualquer proposição, assume-se que seu valor é verdadeiro ou que sua negação é verdadeira. Considerando a proposição "O céu é azul", podemos afirmar que a disjunção Lógica "O céu é azul ou o céu não é azul" é verdadeira pois uma das afirmações têm que ser necessariamente verdadeira.

Temos também o *Princípio da Não Contradição* [12] que afirma que duas sentenças contraditórias não podem ser verdadeiras ao mesmo tempo. Usando o exemplo anterior, podemos afirmar que a conjunção Lógica "O céu é azul e o céu não é azul" é falsa pois apenas uma das afirmações pode ser verdadeira ao mesmo tempo.

A Lógica Clássica ou Booleana é equivalente à Álgebra Booleana onde os valores de verdadeiro ou falso assumem os valores numéricos 1 e 0 respectivamente, e as expressões podem ser submetidas a duas operações binárias (funções com duas variáveis) e uma operação unária (função com uma variável), cujas propriedades se comparam ao Princípio da Não Contradição e a Lei do Terceiro Excluído da Lógica Clássica.

A Lógica Booleana [12] é o fundamento da Lógica Computacional, que visa usar Lógica para raciocionar sobre problemas computacionais. Todos os eletrônicos que usamos hoje

em dia são, de uma maneira ou de outra, operações de mais baixo nível compostas de 0s e 1s. O simples ato de acender uma lâmpada é determinar dois estados possíveis para ela: ou ela está acesa ou ela está desligada, verdadeiro ou falso, 1 ou 0.

A Lógica Clássica, porém, tem suas limitações. Por possuir uma semântica bivalorada, ou seja, assumir apenas dois valores para uma sentença, pode não ser suficiente para expressar todos os problemas computacionais. Considere a seguinte frase: "No ano 3000 poderemos dirigir carros voadores". Como afirmar que essa sentença é verdadeira ou falsa se o fato por ela expresso ainda não aconteceu? A afirmação pode ser tanto verdadeira como sua negação também, o que fere a Lei do Terceiro Excluído. De que forma então poderíamos avaliar a sentença? A única afirmação correta para essa frase seria, simplesmente, *talvez*. Daí surgiu a necessidade de assumir um terceiro valor para a Lógica, um valor indeterminado (ou possível), capaz de validar proposições de cunho duvidoso ou de tempo futuro.

Desta necessidade que, em 1917, o lógico polonês Jan Łukasiewicz propôs a existência da primeira Lógica Multivalorada [7] onde, ao contrário da Lógica Clássica, possuia valores intermediários entre o que podia ser falso e o que podia ser verdadeiro. Não seria mais necessário determinar apenas dois valores para uma proposição, mas também qualquer valor entre 0 e 1.

As aplicações de uma Lógica Multivalorada abrangem, em grande parte, a Computação. Enquanto os circuitos atualmente são construídos de forma bivalorada, ou seja, cada transistor (componente eletrônico semicondutor) assume dois valores de tensão (alto e baixo) para caracterizar valores de verdadeiro ou falso da Lógica Clássica, um circuito multivalorado poderia, teoricamente, aumentar a densidade de informações em um mesmo espaço de memória ao permitir o uso de três ou mais níveis de tensão.

A existência da Computação Multivalorada, porém, ainda é um desafio. Até o momento não foram descobertos materiais que possam ter sua tensão controlada, mas cada vez mais estudos estão sendo feitos nessa área. Os pesquisadores Laurent Baudry, Igor Lukyanchuk e Valerii M. Vinokur, por exemplo, estudam o uso dos *ferroelétricos* [4], uma classe de materiais cuja polarização pode ser controlada com campos elétricos, para, teoricamente, controlar a tensão registrada em circuitos multivalorados.

1.2 Problema

Apesar da realização em *hardware* da Computação Multivalorada ainda ser um sonho distante, há muito que se estudar no que diz respeito à Lógicas Multivaloradas. A adição de novos valores lógicos desconstroem as propriedades básicas que conhecemos da Lógica Clássica e, portanto, implicam em novas maneiras de avaliar proposições. A Ló-

gica Multivalorada mais estudada é a Lógica Três-Valorada [5], objeto de estudo deste trabalho.

Ainda há poucas ferramentas automatizadas que permitam a avaliação de fórmulas (i.e. representações de proposições dentro de uma certa linguagem formal) em Lógica Três-Valorada. A única ferramenta encontrada na época da realização deste trabalho e que conseguia avaliar fórmulas em Lógica Três-Valorada foi o MUltseq [2], que foi utilizado para realização de testes de comparação. As ferramentas atualmente disponíveis possuem um foco mais abrangente e não apenas na resolução de fórmulas Três-Valoradas. Por esse motivo, muitas vezes o processo de avaliação das ferramentas pode não ser melhor adequado ou otimizado para atender essa Lógica.

1.3 Objetivo

O objetivo deste trabalho foi aplicar os teoremas propostos pela Prof.a Dr.a Cláudia Nalon e pelo Prof. Dr. João Marcos em [1] na construção de um programa provador de fórmulas Três-Valoradas. O provador consiste em traduzir uma fórmula na Lógica Três-Valorada em um conjunto de cláusulas que serão avaliadas para satisfatibilidade.

O segundo objetivo deste trabalho foi comparar o desempenho do nosso provador construído com o provador MUltseq e determinar quais das duas ferramentas é a mais adequada para avaliação de fórmulas na Lógica Três-Valorada no que diz respeito ao tempo de execução e uso de memória.

1.4 Descrição dos Capítulos

No Capítulo 2 são apresentados os fundamentos da Lógica Infinitamente-valorada de Łukasiewicz, da Lógica n-valorada de Łukasiewicz, da Lógica bivalente (ou clássica) e por fim da Lógica Três-Valorada de Łukasiewicz.

No Capítulo 3 é apresentada a definição de Tradução Bivalente e, no Capítulo 4, o método de resolução clausal baseado nesta tradução.

Já no Capítulo 5, são apresentados a implementação do programa tradutor proposto, as ferramentas e programas utilizados e a saída esperada.

No Capítulo 6, são relatados os testes de desempenho realizados e os resultados obtidos por ambos os provadores avaliados.

Por fim, são apresentadas as conclusões a respeito dos testes realizados e de que forma o trabalho contribuiu para a avaliação de fórmulas na Lógica Três-Valorada.

Capítulo 2

Lógica Três-Valorada

Apresentamos as definições da Lógica Infinitamente-Valorada de Łukasiewicz, da Lógica n-Valorada de Łukasiewicz, da Lógica Bivalente (ou clássica) e por fim da Lógica Três-Valorada de Łukasiewicz, comumente conhecida como Lógica Ł$_3$ [5].

2.1 Semântica

Precisamos definir primeiramente alguns conceitos que serão utilizados neste capítulo em diante. As seguintes definições são dadas em [1].

Definição 1 [Assinatura Proposicional] Seja uma *assinatura (proposicional)* Σ a união de uma família $\{\Sigma_m\}_{m\in\mathbb{N}}$ de construtores, onde cada conjunto Σ_m contém apenas símbolos funcionais de aridade m, e onde Σ_i e Σ_j são ditos disjuntos sempre que $i \neq j$. Seja $\mathcal{A}$ a coleção enumerável de símbolos chamados *variáveis (atômicas)*, ditos serem disjuntos da assinatura Σ. Os símbolos nulários em Σ_0 são em geral chamados de *símbolos verdade*. Uma *linguagem proposicional* $\mathbb{L}$ é recursivamente gerada da forma usual, considerando que sua *fórmula composta* seja da forma $\odot(\varphi_0, \varphi_1, \ldots, \varphi_m)$, para algum $\odot \in \Sigma_{m+1}$. No último caso, o símbolo $\odot$ é dito ser o *conectivo principal* e a fórmulas φ_k, para $0 \leqslant k \leqslant m$, são chamadas de *subfórmulas imediatas* de $\odot(\varphi_0, \varphi_1, \ldots, \varphi_m)$. As fórmulas em $\mathcal{A} \cup \Sigma_0$ são chamadas de *não-compostas* e não possuem nenhuma subfórmula própria.

Definição 2 [Noção Canônica da Complexidade de uma Fórmula] A *noção canônica da complexidade de uma fórmula* $\mathsf{cp} : \mathbb{L} \longrightarrow \mathbb{N}$ pode ser definida ao configurar $\mathsf{cp}(\varphi) = 0$ se φ é não-composta, e $\mathsf{cp}(\varphi) = 1 + \mathsf{Max}_{0\leqslant k\leqslant m}\, \mathsf{cp}(\varphi_k)$ se φ é composta e φ_k, para $0 \leqslant k \leqslant m$, é sua subfórmula imediata.

Definição 3 [Substituição Uniforme] A *substituição uniforme* é um endorfismo no conjunto de fórmulas que mapeam cada construtor nele mesmo; é unicamente definido assim que as variáveis de uma linguagem são mapeadas em fórmulas. Por $\varphi[p \mapsto \psi]$ denotamos o resultado de uniformemente substituir ψ para cada ocorrência da variável p na fórmula φ.

Definição 4 [Valoração] Uma *valoração* para uma linguagem $\mathbb{L}$ é o mapeamento $w : \mathbb{L} \to \mathcal{V}_w$, onde $\mathcal{V}_w$ denota um conjunto não-vazio de *valores-verdade* contendo um subconjunto $\mathcal{D}_w$ de *valores designados*. Os valores em $\mathcal{V}_w \backslash \mathcal{D}_w$ são chamados de *não designados*. A valoração w *satisfaz* uma fórmula φ se a w atribuir um valor designado a φ; caso contrário, dizemos que w *falsifica* φ. Chamamos de $\mathcal{V}(2) = \{F, T\}$ o conjunto de *valores lógicos* para os quais fixamos $\mathcal{D}(2) = \{T\}$. Qualquer valoração $\mathcal{V}_2$ é referenciada como *bivaloração*.

Definição 5 [Semântica] Uma *semântica* para $\mathbb{L}$ é a família Sem de valorações. Uma fórmula é dita *válida* (ou é uma *tautologia*), em uma dada semântica, se não é negada por nenhuma valoração dessa semântica; uma fórmula é dita *insatisfatível* por uma dada semântica se todas as valorações correspondentes a negarem. Dada uma semântica Sem e um subconjunto $\triangle \subseteq \mathbb{L}$, denominamos $Mod(\triangle)$ como o conjunto de valorações de Sem que mapeam todas as fórmulas de $\triangle$ em valores designados.

2.2 Lógica de Łukasiewicz

Definição 6 [5] Lógica de Łukasiewicz Infinitamente-Valorada, indicada por $[0,1]_{\text{Ł}}$, é a Lógica Multivalorada $< \mathbb{L}_{\text{Ł}uk}, N, \varphi >$ equipada com um conjunto de valores designados $D = \{1\}$, onde N é o intervalo real $[0,1]$, e a interpretação dos símbolos proposicionais φ é dada por:

$$\varphi_{\perp} = 0 \qquad \varphi_{\wedge}(x,y) = min\{x,y\}$$
$$\varphi_{\top} = 1 \qquad \varphi_{\vee}(x,y) = max\{x,y\}$$
$$\varphi_{\neg}(x) = 1 - x \qquad \varphi_{\odot}(x,y) = max\{0, x+y-1\}$$
$$\varphi_{\supset}(x,y) = min\{1, 1-x+y\} \qquad \varphi_{\oplus}(x,y) = min\{1, x+y\}$$

Referenciamos $\perp$ como *falsum*, $\top$ como *verum*, $\neg$ como negação, $\supset$ como implicação, $\vee$ como disjunção fraca, $\wedge$ como conjunção fraca, $\odot$ como conjunção forte, e finalmente $\oplus$ como disjunção forte.

A Lógica de Łukasiewicz n-Valorada, indicada por $Ł_n$, é a Lógica definida a partir da Lógica de Łukasiewicz Infinitamente-Valorada, através da restrição do universo de avaliação para o conjunto $N_n = \{0, \frac{1}{n-1}, ..., \frac{n-1}{n-1}\}$. Desta definição, se restringirmos o universo para $n = 2$, obtemos definição de Lógica Booleana, também conhecida como Lógica Bivalente ou Lógica clássica.

2.3 Lógica Três-Valorada de Łukasiewicz

A Lógica Três-Valorada de Łukasiewicz possui três valores-verdade: *verdadeiro*, *falso* e um terceiro *indeterminado*. Sua forma conceitual foi formulada no século XX e é comumente conhecida como Lógica $Ł_3$ [5].

Utilizando o formulação da seção anterior e restringindo o universo para $n = 3$, obtemos o seguinte conjunto de valores verdade para a Lógica $Ł_3$:

$$N_n\{0, \tfrac{1}{3-1}, \tfrac{3-1}{3-1}\} = \{0, \tfrac{1}{2}, 1\}$$

Com a adição de um terceiro valor na Lógica ternária, temos também o aumento do número de operadores unários (operações com uma entrada). A Lógica clássica possui $2^2 = 4$ operadores unários, enquanto a Lógica ternária passa a ter $3^3 = 27$ operadores unários.

O mesmo ocorre com o número de operadores binários (operações com duas entradas). A Lógica clássica possui $2^{2^2} = 16$ operadores binários e a Lógica ternária possui $3^{3^2} = 19,683$ operadores binários. Apesar de podermos identificar uma pequena fração (e, ou, implicação, negação), seria difícil nomear todos os operadores binários da Lógica ternária e nem todos são comumente usados.

2.4 Tabelas Verdade

Como na Lógica Bivalente, podemos construir tabelas verdade ao realizar operações Lógicas entre os valores numéricos que representam os valores de verdade da Lógica $Ł_3$, utilizando-se do resultado como referência para outras futuras operações. As Tabelas 2.2 e 2.4 foram construídas a partir das regras definidas por Łukasiewicz na Definição 6.

φ	$\neg\varphi$
0	1
0.5	0.5
1	0

Tabela 2.1: $\neg\ (\varphi)$

$\varphi \wedge \psi$		ψ		
		0	**0.5**	**1**
φ	**0**	0	0	0
	0.5	0	0.5	0.5
	1	0	0.5	1

Tabela 2.2: $\wedge\ (\varphi, \psi)$

$\varphi \vee \psi$		ψ		
		0	**0.5**	**1**
φ	**0**	0	0.5	1
	0.5	0.5	0.5	1
	1	1	1	1

Tabela 2.3: $\vee\ (\varphi, \psi)$

$\varphi \supset \psi$		ψ		
		0	**0.5**	**1**
φ	**0**	1	1	1
	0.5	0.5	1	1
	1	0	0.5	1

Tabela 2.4: $\supset\ (\varphi, \psi)$

A Tabela 2.1 representa a negação dos valores de φ; a Tabela 2.2 a conjunção entre dois valores φ e ψ; a Tabela 2.3 representa a disjunção entre dois valores φ e ψ; e a Tabela 2.4 a implicação entre dois valores φ e ψ.

Utilizando a negação e a implicação de Łukasiewicz, podemos apresentar as seguintes equivalências:

- $\neg\varphi = \varphi \supset \bot$
- $\varphi \vee \psi = (\varphi \supset \psi) \supset \psi$
- $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$
- $\varphi \leftrightarrow \psi = (\varphi \supset \psi) \wedge (\psi \supset \varphi)$

Para o próximo capítulo é preciso que uma fórmula dada seja transformada em uma série de implicações. As equivalências acima nos permitem realizar tais transformações como mostrado no Exemplo 1.

Exemplo 1 *Considere a fórmula* $(A \wedge B)$. *Podemos mostrar as seguintes equivalências semânticas:*

1. $A \wedge B =$
2. $\neg(\neg A \vee \neg B) =$
3. $\neg((\neg A \supset \neg B) \supset \neg B) =$
4. $(((A \supset \bot) \supset (B \supset \bot)) \supset (B \supset \bot)) \supset \bot$

Como observado no Exemplo 1, uma fórmula formada de apenas uma conjunção se transforma em uma série de implicações que atendem as equivalências mostradas anteriormente. Uma vez realizado o conjunto de transformações, podemos passar para o próximo capítulo onde definiremos Tradução Bivalente.

Capítulo 3

Tradução Bivalente

O procedimento da tradução bivalente consiste em realizar a tradução entre o conjunto de valores designados $\{1\}$ e o conjunto de valores não designados $\{0, 0.5\}$ da Lógica Ł$_3$ para valores de verdade T e F da Lógica Clássica a partir da definição do que seja uma impressão binária, que será apresentada logo a seguir. As definições deste capítulo foram retiradas de [1].

3.1 Declarações Bivalentes

Definição 7 [Impressão Binária] Uma *sequência separadora* $\overrightarrow{\theta} = \langle \theta_0, \theta_1, \ldots, \theta_s \rangle$ para $\mathcal{L}$ é uma sequência de conectores unários (a serem referenciados como *separadores*) com a propriedade de que para cada par de avaliações distintas $w_1, w_2 \in \mathsf{Sem}$, ou seja, cada par de avaliações para as quais $w_1(p) \neq w_2(p)$ para alguma variável p, existe um θ_r, para $0 \leqslant r \leqslant s$, tal que $b_{w_1}(\theta_r(p)) \neq b_{w_2}(\theta_r(p))$. Dada uma valoração $x \in \mathcal{V}$ atribuída por uma função de avaliação w para uma determinada fórmula φ, a *impressão binária* é $\overrightarrow{\theta}(x)$ a sequência única $\overrightarrow{X} = \langle b_w(\theta_0(\varphi)), b_w(\theta_1(\varphi)), \ldots, b_w(\theta_s(\varphi)) \rangle \in \{F, T\}^{s+1}$. Se uma sequência de F's e T's de tamanho $(s+1)$ não corresponde a nenhuma impressão binária de um valor-verdade em $\mathcal{V}$, dizemos que essa sequência *representa um absurdo*.

Definição 8 [Medida de Complexidade Generalizada] Fórmulas na forma $\theta_r(\varphi)$, onde θ_r é um separador e φ não é composto, são chamados de *básicos*. Uma *noção generalizada da complexidade de uma fórmula* $\mathsf{gcp} : \mathcal{S} \longrightarrow \mathbb{N}$ é definida como:

- $\mathsf{gcp}(\varphi) = 0$ se φ é básico; e

- $\mathsf{gcp}(\varphi) = 1 + \mathsf{Max}_{0 \leqslant k \leqslant m}\, \mathsf{gcp}(\varphi_k)$, se φ tem a forma $\theta_r(\odot(\varphi_0, \varphi_1, \ldots, \varphi_m))$, para $\odot \in \Sigma_{m+1}$ e $0 \leqslant r \leqslant s$.

Fórmulas com complexidades positivas gcp são ditas *analisáveis*.

Seja $\overrightarrow{\theta}(X) = \langle X_0, X_1, \ldots, X_s \rangle$ a impressão binária de algum valor-verdade $X \in \mathcal{V}$. Dados a fórmula φ da forma $\theta_r(\odot(\varphi_0, \varphi_1, \ldots, \varphi_m))$ e um valor lógico X, chamaremos $R_X^{\theta_r \odot}$ o conjunto de tuplas de valores de $\mathcal{V}$ para as subfórmulas $\varphi_0, \varphi_1, \ldots, \varphi_m$ associadas à avaliação $w \in \mathsf{Sem}$ de modo a garantir que $b_w(\varphi) = X$.

Dessa forma, para cada rótulo $X \in \{F, T\}$, cada separador $\theta_r \in \Sigma^\theta$ e cada operador m-ário $\odot$, que não seja separador, da assinatura $\mathcal{L}$ associaremos a seguinte *declaração* B:

$$X{:}\theta_r(\odot(p_1, p_2, \ldots, p_m)) \quad \Longrightarrow \quad ||_{\langle x_1, x_2 \ldots, x_m \rangle \in R_X^{\theta_r \odot}} \, (\&_{k=1}^{m} V(p_k, \overrightarrow{\theta}(x_k))) \qquad (B_X^{\theta_r \odot})$$

Seja $\uparrow$ um valor lógico *indefinido*. Chamaremos a sequência $\overrightarrow{Y} \in \{F, T, \uparrow\}^{s+1}$ *inatingível* quando qualquer sequência de F's e T's de tamanho $(s+1)$ representa um absurdo. Dada a sequência $\overrightarrow{Y} \in \{F, T, \uparrow\}^{s+1}$, por $\mathsf{dom}(\overrightarrow{Y})$ denominamos o conjunto $\{0 \leqslant r \leqslant s : Y_r \neq \uparrow\}$. Para cada sequência inatingível $\overrightarrow{Y}$ associaremos a seguinte *declaração* U:

$$\&_{r \in \mathsf{dom}(\Upsilon)} \, Y_r{:}\theta_r(p_0) \quad \Longrightarrow \quad \ominus \qquad (U_Y)$$

Definição 9 [Declarações bivalentes induzidas por $\mathcal{L}$] Sejam $\mathcal{L}$ uma Lógica finitamente-valorada sobre a assinatura Σ e $\mathcal{V}$ um conjunto de valores-verdade, $\overrightarrow{\theta}$ a sequência separadora para $\mathcal{L}$, $\Sigma^\theta \subseteq \Sigma_1$ o conjunto de separadores $\overrightarrow{\theta}$ e $\{Y_j\}_{j \in \lambda}$, para algum $\lambda \in \mathbb{N}$, a família de sequências binárias inatingíveis mínima que cobrem todas as impressões binárias que não correspondem a valores-verdade em $\mathcal{V}$. As semânticas bivalentes que chamaremos $\mathcal{B}(\mathcal{L}, \overrightarrow{\theta})$ é descrita pela coleção de declarações B, $B_X^{\theta_r \odot}$, para cada rótulo $X \in \{F, T\}$, para cada $\theta_r \in \Sigma^\theta$ e cada $\odot \in \Sigma \backslash \Sigma^\theta$, e a coleção de todas as declarações U_{Y_j}, para $j \in \lambda$.

Com as declarações B e U podemos descrever uma lista de declarações bivalentes (Tabela 3.1) para caracterizar a Lógica finitamente-valorada Ł$_3$, onde o separador é a função identidade, isto é, $\theta(p_0) = p_0$ para qualquer símbolo proposicional p_0.

Em posse das declarações, podemos submeter uma fórmula composta de uma série de implicações à tradução bivalente. Apresentamos abaixo um exemplo de tradução.

$(U\langle T,F\rangle)$	$(T{:}p_0 \ \& \ F{:}\theta(p_0))$	$\Longrightarrow$	$\ominus$
$(B_T^{\perp})$	$T{:}\perp$	$\Longrightarrow$	$\ominus$
$(B_T^{\theta\perp})$	$T{:}\theta\perp$	$\Longrightarrow$	$\ominus$
$(B_F^{\theta\theta})$	$F{:}\theta(\theta(p_0))$	$\Longrightarrow$	$F{:}\theta(p_0)$
$(B_T^{\theta\theta})$	$T{:}\theta(\theta(p_0))$	$\Longrightarrow$	$T{:}\theta(p_0)$
$(B_F^{\supset})$	$F{:}p_0 \supset p_1$	$\Longrightarrow$	$(T{:}p_0 \ \& \ F{:}p_1) \ \| \ (T{:}\theta(p_0) \ \& \ F{:}\theta(p_1))$
$(B_T^{\supset})$	$T{:}p_0 \supset p_1$	$\Longrightarrow$	$(F{:}p_0 \ \& \ T{:}\theta(p_1)) \ \| \ F{:}\theta(p_0) \ \| \ T{:}p_1$
$(B_F^{\theta\supset})$	$F{:}\theta(p_0 \supset p_1)$	$\Longrightarrow$	$(T{:}p_0 \ \& \ F{:}\theta(p_1))$
$(B_T^{\theta\supset})$	$T{:}\theta(p_0 \supset p_1)$	$\Longrightarrow$	$F{:}p_0 \ \| \ T{:}\theta(p_1)$

Tabela 3.1: Conjunto de Declarações Bivalentes para Ł$_3$

Exemplo 2 *Considere a fórmula* $(p \supset p) \supset p$*. Aplicando a metanegação em* $B_{X^{\mathfrak{c}}}^{\theta_T\odot}$ *e utilizando o conjunto de declarações bivalentes contidas na Tabela 3.1, podemos realizar as seguintes transformações:*

1. $F{:}(p \supset p) \supset p =$
2. $(T{:}(p \supset p) \ \& \ F{:}p) \ \| \ (T{:}\theta(p \supset p) \ \& \ F{:}\theta(p)) =$
3. $(T{:}(p \supset p) \ \& \ F{:}p) \ \| \ ((F{:}p \ \| \ T{:}\theta(p)) \ \& \ F{:}\theta(p)) =$
4. $(((F{:}p \ \& \ T{:}\theta(p)) \ \| \ F{:}\theta(p) \ \| \ T{:}p) \ \& \ F{:}p) \ \| \ ((F{:}p \ \| \ T{:}\theta(p)) \ \& \ F{:}\theta(p))$

Como se pode observar no Exemplo 2, aplicamos as regras da Tabela 3.1 para traduzir de forma recursiva a fórmula fornecida em conjunto de declarações bivalentes. Essa nova estrutura é utilizada no próximo capítulo onde veremos a resolução clausal para a Lógica Ł$_3$.

Capítulo 4

Cálculo

Neste capítulo apresentaremos o método de Resolução Clausal, onde o método é aplicado a fórmulas em uma Forma Normal Conjuntiva específica, CNF_{BS}. As definições deste capítulo form retiradas de [1].

4.1 Forma Normal

Resolução Clausal normalmente é aplicada à forma normal da negação de uma fórmula e é um método para verificar a satisfação dessa fórmula. O método consiste em aplicar um conjunto de regras baseadas no Princípio de Resolução [9] até que a cláusula vazia seja encontrada ou até que nenhuma outra cláusula (i.e. uma metadisjunção de fórmulas básicas rotuladas) possa ser gerada. Se a cláusula vazia é encontrada, a fórmula original é insatisfatível; caso contrário a fórmula é satisfatível e podemos construir um modelo que é testemunho para sua satisfação.

Seja $\mathcal{L} = \langle \mathcal{S}, \succ_{\mathcal{L}} \rangle$, onde $\mathcal{S}$ é recursivamente definida sobre a assinatura finita $\Sigma = \bigcup_{k \in \mathbb{N}} \Sigma_k$, uma Lógica Finitamente Valorada fixa com semântica funcional Sem. Seja $\mathcal{B}(\mathcal{L}, \overrightarrow{\theta})$ a descrição de semânticas bivalentes $\mathsf{Sem}(\mathbf{2})$ correspondendo à redução bivalente Sem. Definiremos $\mathsf{RES}_{\mathcal{B}(\mathcal{L}, \overrightarrow{\theta})}$ como o cálculo de Resolução para as declarações bivalentes $\mathcal{B}(\mathcal{L}, \overrightarrow{\theta})$ que descrevem a Lógica Multivalorada $\mathcal{L}$ com a sequência separadora $\overrightarrow{\theta}$.

A Forma Normal Conjuntiva de uma fórmula, que chamaremos de CNF_{BS}, consiste em declarações metalinguísticas da forma $\&_{i=0}^{a}(||_{j=0}^{b_i} X_{ij}{:}\varphi_{ij})$ onde $a, b_i \in \mathbb{N}$, $X_{ij} \in \{T, F\}$ e cada φ_{ij} é uma fórmula básica. Como a metaconjunção é comutativa, associativa e idempotente, trataremos qualquer fórmula em CNF_{BS} como sendo simplesmente um conjunto de cláusulas.

Se uma fórmula φ é subfórmula de ψ, ou seja, se ψ tem a forma $\psi[p \mapsto \varphi]$, então a fórmula $X{:}\psi'$ & $X{:}(t_\varphi \Longrightarrow \varphi)$ & $X{:}(\varphi \Longrightarrow t_\varphi)$, onde $\psi' = \psi[p \mapsto t_\varphi]$ para uma nova variável atômica t_φ, pode ser usada para substituir a fórmula $X{:}\psi$. A metabi-implicação

$X{:}(t_\varphi \Longrightarrow \varphi)$ & $X{:}(\varphi \Longrightarrow t_\varphi)$ é comumente referenciada como a definição de φ. Apenas um lado dessa definição é necessária porém, precisamente a metaimplicação $X{:}(t_\varphi \Longrightarrow \varphi)$.

A função de transformação τ recebe uma fórmula $X{:}\varphi$ como entrada e a converte na forma normal por aplicações recursivas de reescrita e renomeamentos. Para cada declaração B da forma: $X_0{:}\varphi_0 \Longrightarrow (||_{i=1}^{n}\ X_i{:}\varphi_i)$ associamos a seguinte regra de reescrita: $\tau(X_0{:}\varphi_0) \longmapsto \tau(||_{i=1}^{n}\ X_i{:}\varphi_i)$. Se $\odot$ é o operador principal de φ_0, indicamos a correspondente regra de reescrita como $\tau_{X_0}^{\odot}$. A transformação distribui sobre metaconjunções e metadisjunções, como a seguir:

$$\tau(\&_{i=1}^{n}\ X_i{:}\varphi_i) \longmapsto \&_{i=1}^{n}\ \tau(X_i{:}\varphi_i). \qquad (\tau^{\&})$$

$$\tau(||_{i=1}^{n}\ X_i{:}\varphi_i) \longmapsto ||_{i=1}^{n}\ \tau(X_i{:}\varphi_i). \qquad (\tau^{||})$$

A próxima regra renomeia metaconjunções que aparecem entre metadisjunções:

$$\tau(\psi\ ||\ X{:}\varphi) \longmapsto \tau(\psi\ ||\ T{:}t_\varphi)\ \&\ \tau(T{:}t_\varphi \Longrightarrow X{:}\varphi) \qquad (\tau^{ren})$$

onde ψ é uma fórmula, $X{:}\varphi$ é a metaconjunção e t_φ é uma nova variável atômica. Metaconjunções do lado direito de metaimplicações são reescritas da forma usual:

$$\tau(T{:}t \Longrightarrow \&_{i=1}^{n}\ X_i{:}\varphi_i) \longmapsto \&_{i=1}^{n}\ \tau(T{:}t \Longrightarrow \tau(X_i{:}\varphi_i)) \qquad (\tau^{\Longrightarrow \&})$$

A última regra de reescrita transforma metaimplicações em metadisjunções:

$$\tau(T{:}t \Longrightarrow D) \longmapsto \begin{cases} F{:}t\ ||\ D, & \text{se } D \text{ é uma cláusula} \\ \tau(T{:}p \Longrightarrow \tau(D)), & \text{caso contrário} \end{cases} \qquad (\tau^{\Longrightarrow})$$

Como caso base da função de transformação, definimos:

$$\tau(X{:}\varphi) \longmapsto X{:}\varphi, \text{ se } \varphi \text{ é uma fórmula básica.} \qquad (\tau^{b})$$

Cláusulas são mantidas na forma simplificada, i.e. as seguintes regras de simplificação são aplicadas em todas as etapas de transformação (onde D é uma cláusula e φ é uma fórmula básica):

$$\begin{aligned} \sigma(D\ ||\ X{:}\varphi\ ||\ X{:}\varphi) &\longmapsto \sigma(D\ ||\ X{:}\varphi) \\ \sigma(D\ ||\ \oplus) &\longmapsto \oplus \\ \sigma(D\ ||\ X{:}\varphi\ ||\ X^{\mathbf{c}}{:}\varphi) &\longmapsto \oplus \\ \sigma(D\ ||\ \ominus) &\longmapsto \sigma(D) \end{aligned}$$

Exemplo 3 *Considere o resultado das transformações aplicadas sobre a fórmula* $(p \supset p) \supset p$ *no Exemplo 2 do Capítulo 3. Aplicaremos a função* τ *e as regras de reescrita e renomeamento vistas acima sobre o resultado para obtermos o seguinte conjunto de cláusula na forma normal descrita acima:*

1. $\tau((((F{:}p \ \& \ T{:}\theta(p)) \,||\, F{:}\theta(p) \,||\, T{:}p) \ \& \ F{:}p) \,||\, ((F{:}p \,||\, T{:}\theta(p)) \ \& \ F{:}\theta(p))) =$
2. $(T{:}t_0 \,||\, T{:}t_1) \ \& \ \tau(T{:}t_0 \Longrightarrow ((F{:}p \ \& \ T{:}\theta(p)) \,||\, F{:}\theta(p) \,||\, T{:}p) \ \& \ F{:}p)$
   $\& \ \tau(T{:}t_1 \Longrightarrow ((F{:}p \,||\, T{:}\theta(p)) \ \& \ F{:}\theta(p)) =$
3. $(T{:}t_0 \,||\, T{:}t_1) \ \& \ \tau(T{:}t_0 \Longrightarrow \tau(F{:}p \ \& \ T{:}\theta(p)) \,||\, F{:}\theta(p) \,||\, T{:}p) \ \& \ \tau(T{:}t_0 \Longrightarrow F{:}p)$
   $\& \ \tau(T{:}t_1 \Longrightarrow F{:}p \,||\, T{:}\theta(p)) \ \& \ \tau(T{:}t_1 \Longrightarrow F{:}\theta(p)) =$
4. $(T{:}t_0 \,||\, T{:}t_1) \ \& \ (F{:}t_0 \,||\, \tau(F{:}p \ \& \ T{:}\theta(p)) \,||\, F{:}\theta(p) \,||\, T{:}p) \ \& \ (F{:}t_0 \,||\, F{:}p)$
   $\& \ (F{:}t_1 \,||\, F{:}p \,||\, T{:}\theta(p)) \ \& \ (F{:}t_1 \,||\, F{:}\theta(p)) =$
5. $(T{:}t_0 \,||\, T{:}t_1) \ \& \ (F{:}t_0 \,||\, T{:}t_2 \,||\, F{:}\theta(p) \,||\, T{:}p) \ \& \ (F{:}t_0 \,||\, F{:}p)$
   $\& \ (F{:}t_1 \,||\, F{:}p \,||\, T{:}\theta(p)) \ \& \ (F{:}t_1 \,||\, F{:}\theta(p))$
   $\& \ \tau(T{:}t_2 \Longrightarrow F{:}p \ \& \ T{:}\theta(p)) =$
6. $(T{:}t_0 \,||\, T{:}t_1) \ \& \ (F{:}t_0 \,||\, T{:}t_2 \,||\, F{:}\theta(p) \,||\, T{:}p) \ \& \ (F{:}t_0 \,||\, F{:}p)$
   $\& \ (F{:}t_1 \,||\, F{:}p \,||\, T{:}\theta(p)) \ \& \ (F{:}t_1 \,||\, F{:}\theta(p))$
   $\& \ \tau(T{:}t_2 \Longrightarrow F{:}p) \ \& \ \tau(T{:}t_2 \Longrightarrow T{:}\theta(p)) =$
7. $(T{:}t_0 \,||\, T{:}t_1) \ \& \ (F{:}t_0 \,||\, T{:}t_2 \,||\, F{:}\theta(p) \,||\, T{:}p) \ \& \ (F{:}t_0 \,||\, F{:}p)$
   $\& \ (F{:}t_1 \,||\, F{:}p \,||\, T{:}\theta(p)) \ \& \ (F{:}t_1 \,||\, F{:}\theta(p))$
   $\& \ (F{:}t_2 \,||\, F{:}p) \ \& \ (F{:}t_2 \,||\, T{:}\theta(p)) =$
8. $(T{:}t_0 \,||\, T{:}t_1)$
   $(F{:}t_0 \,||\, T{:}t_2 \,||\, F{:}\theta(p) \,||\, T{:}p)$
   $(F{:}t_0 \,||\, F{:}p)$
   $(F{:}t_1 \,||\, F{:}p \,||\, T{:}\theta(p))$
   $(F{:}t_1 \,||\, F{:}\theta(p))$
   $(F{:}t_2 \,||\, F{:}p)$
   $(F{:}t_2 \,||\, T{:}\theta(p))$
   $(F{:}p \,||\, T{:}\theta(p))$

Novas Variáveis	*Renomeamentos*
t_0	$(T{:}t_2 \,\|\|\, F{:}\theta(p) \,\|\|\, T{:}p) \ \& \ F{:}p$
t_1	$(F{:}p \,\|\|\, T{:}\theta(p)) \ \& \ F{:}\theta(p)$
t_2	$F{:}p \ \& \ T{:}\theta(p)$

4.2 Regras de Inferência

Seja $\mathcal{B}(\mathcal{L}, \overrightarrow{\theta})$ a descrição bivalente de uma Lógica Finitamente-valorada $\mathcal{L}$ em uma linguagem $\mathcal{S}$, e seja $\{U_j\}_{j\in\lambda}$, para uma $\lambda \in \mathbb{N}$, a família de declarações-U nesta descrição. Seja $X{:}\varphi$ a fórmula rotulada como $\varphi \in \mathcal{S}$. Seja Φ o conjunto de cláusulas obtidas ao transformar $X{:}\varphi$ no correspondente CNF_{BS} baseado no $\mathcal{B}(\mathcal{L}, \overrightarrow{\theta})$. O cálculo baseado em Resolução para $\mathcal{L}$ vai conter a regra de Resolução binária (RES), que é a variação sintá-

tica de uma regra de Resolução clássica (binária) usual [9], e um conjunto de regras de inferência baseadas em hiperResolução chamadas de (RES_{U_j}), para $j \in \lambda$, para lidar com as restrições de validação relacionadas à declarações-U_j correspondentes.

HiperResolução [10] é um refinamento do método de Resolução que combina vários passos de Resolução binária, assim evitando a geração de cláusulas intermediárias (e a interação entre elas) quando em busca de uma prova. Declarações-U expressam as sequências binárias que não podem ser obtidas como impressões binárias dos valores-verdade originais de $\mathcal{L}$. Dessa forma, a metaconjunção de uma fórmula rotulada do lado esquerdo da metaimplicação em uma declaração-U corresponde a um absurdo lógico na semântica da metalinguagem. As regras de Resolução são dadas na Figura 4.1, onde D_i é uma cláusula, $X_i \in \{T, F\}$, e φ, φ_i são fórmulas básicas (onde $1 \leqslant i \leqslant n_j$, $j \in \lambda$, e n_j é o número de metaconjunções em U_j). Premissas das regras de inferência são chamadas de *cláusulas parentes*. Conclusões nessas regras são chamadas de *resolventes*. Também, referenciamos como $\{T{:}\varphi, F{:}\varphi\}$ (resp. $\{X_1{:}\varphi_1, \ldots, X_{n_j}{:}\varphi_{n_j}\}$) ocorrendo em cláusulas parentes de (RES) (resp. (RES_{U_j})) como *conjunto contraditório de fórmulas*. Fórmulas rotuladas em uma conjunto contraditório de fórmulas são ditas *resolvidas* pela regra de inferência respectiva.

Definição 10 [Derivações & Refutações] Seja Φ um conjunto de cláusulas em CNF_{BS}. Uma *derivação para* Φ é a sequência $\Phi_0, \Phi_1, \ldots$ de um conjunto de cláusulas em CNF_{BS}, com $\Phi_0 = \Phi$ e para todo $i > 0$, $\Phi_{i+1} = \Phi_i \cup \{D\}$, onde D é uma cláusula (na forma simplificada) obtida pela aplicação de (RES) ou (RES_{U_j}) em $\mathsf{RES}_{\mathcal{B}(\mathcal{L}, \overrightarrow{\theta})}$ para cláusulas em Φ_i. Fórmulas repetidas não são adicionadas ao próximo passo de derivação, i.e. temos que $(\Phi_{i+1} \backslash \Phi) \neq \varnothing$, e que D não seja uma tautologia. Uma *refutação* para um conjunto de cláusulas Φ é uma derivação finita de Φ, $\Phi_0, \ldots, \Phi_m$, com $m \in \mathbb{N}$, tal que $\ominus \in \Phi_m$.

$$(\mathsf{RES}) \quad \frac{D_1 \;||\; T{:}\varphi \qquad D_2 \;||\; F{:}\varphi}{D_1 \;||\; D_2}$$

$$(\mathsf{RES}_{U_j}) \quad \frac{D_1 \;||\; X_1{:}\varphi_1 \qquad \vdots \qquad D_{n_j} \;||\; X_{n_j}{:}\varphi_{n_j}}{D_1 \;||\; \ldots \;||\; D_{n_j}}$$

para cada $U_j = X_1{:}\varphi_1 \& \ldots \& X_{n_j}{:}\varphi_{n_j} \Longrightarrow \ominus$

Figura 4.1: Regras de Resolução para $\mathsf{RES}_{\mathcal{B}(\mathcal{L}, \overrightarrow{\theta})}$.

Capítulo 5

Implementação

Todas as definições e transformações para a Lógica Três-Valorada de Łukasiewicz definidas nos capítulos anteriores foram, então, aplicadas na criação de um programa gerador de cláusulas que recebe uma fórmula da Lógica Três-Valorada de entrada e fornece um conjunto de cláusulas na metalinguagem bivalente derivadas. Esse conjunto de cláusulas é então exportado para um provador de satisfatibilidade proposicional chamado MiniSAT [15].

Todos os programas descritos neste capítulo, bem como conjuntos de testes utilizados e resultados obtidos na avaliação experimental que são descritos no próximo capítulo, estão disponíveis em `www.cic.unb.br/~nalon/#software`.

5.1 Ferramentas Utilizadas

Para a codificação do programa gerador de cláusulas, foram necessários o uso de duas ferramentas:

- Bison [13];
- Flex [14].

Flex é um gerador de analisadores léxicos e o Bison, em conjunto com o Flex, gera um analisador sintático a partir de uma gramática fornecida. O Flex é responsável por separar a entrada em *tokens*, definindo um tipo para cada conjunto específico de caracteres lidos. Os tokens são então passados para o Bison, que por sua vez determinará conexões sintáticas entre eles seguindo um conjunto de regras de uma gramática.

5.2 Tokenizer

Os tokens definidos no Flex foram os seguintes:

```
TTRUE       "true"
TFALSE      "false"
TDOT        "."
TAND        "&"
TOR         "|"
TNOT        "-"
TIFF        "<->"
TONLYIF     "<-"
TIMPLY      "->"
NAME        [[:alpha:]][[:alnum:]_]*
LPAREN      "("
RPAREN      ")"
WS          [ \t\v\f]
EOL         [\n\r]
```

Cada elemento dessa lista é um token. `TTRUE` e `TFALSE` representam os valores lógicos `true` e `false`; `TDOT` é o token que representa ponto final; `TAND` representa o conectivo de conjunção; `TOR` representa o conectivo de disjunção; `TNOT` representa o conectivo de negação; `TIFF` representa o conectivo de dupla implicação; `TONLYIF` representa o conectivo de condição necessária; `TIMPLY` representa o conectivo de implicação; `NAME` representa as variáveis proposicionais; `LPAREN` e `RPAREN` representam os parênteses; por fim, `WS` e `EOL` representam espaço em branco e fim de linha respectivamente.

Uma vez definidos os tokens, é necessário definir uma gramática para o Bison. A gramática utilizada foi a seguinte:

```
begin:          | formula TDOT

formula:        formula TIFF formula
                | formula TAND formula
                | formula TOR formula
                | formula TIMPLY formula
                | formula TONLYIF formula
                | TNOT formula
                | TLPAREN formula TRPAREN
                | proposition

proposition:    NAME
                | TFALSE
```

| TTRUE

Como entrada para o tradutor, uma fórmula precisa estar no formato especificado na gramática acima. Uma fórmula precisa ser seguida de um ponto no final e pode ser composta por um conjunto de conjunções, disjunções, implicações e negações que conectam símbolos proposicionais que podem ser nomes diversos ou os valores lógicos *true* ou *false*.

Com essa gramática definida e com o auxílio de algumas funções adicionais para construção de árvores em memória (retiradas do provador KSP [3], implementado pela Prof.a Cláudia Nalon), o Bison é capaz de gerar uma árvore sintática anotada contendo as informações da estrutura da fórmula passada. Essa árvore é então usada como base para todas as transformações necessárias durante a geração das cláusulas na Lógica Ł$_3$.

5.3 Gerador de Cláusulas

Uma vez gerada a árvore sintática para uma fórmula, realizamos uma série de transformações e renomeamentos, seguindo as regras definidas nos Capítulos 2, 3 e 4. Essas transformações são gerenciadas por um programa chamado *prover*, codificado na linguagem C.

Dentro do *prover*, três funções básicas são chamadas: *getnnf* transformará uma fórmula qualquer em uma série de implicações; *getfnb*, então, transformará essa nova árvore para uma árvore na forma bivalente; por fim *getfnf* irá realizar uma série de renomeamentos à essa árvore e em seguida gerar o conjunto de cláusulas na metalinguagem bivalente.

5.4 Provador

As cláusulas são então exportadas para um arquivo texto em um formato que possa ser lido pelo programa MiniSAT, responsável por analisar o conjunto de cláusulas passado e dizer se esse conjunto é satisfatível ou insatisfatível. Usamos esse programa para obter os resultados necessários para este trabalho. O formato de saída do tradutor segue a seguinte construção:

```
p cnf nbvar nbclauses
*clausula 1*
*clausula 2*
...
```

A primeira linha indica que a instância está no formato CNF; `nvbar` indica o número de variáveis presentes; e `nbclauses` indica o número de cláusulas no conjunto. Todas

as linhas escritas abaixo serão, cada uma, uma cláusula seguida de 0 no final. Uma vez exportadas, as cláusulas são então submetidas ao MiniSAT cujo resultado dirá se o conjunto é `SATISFIABLE` ou `UNSATISFIABLE`.

5.5 MUltseq

MUltseq [2] é um provador de Lógicas Finitamente Valoradas implementado em Prolog e desenvolvido por Angel Gil e Gernot Salzer [11]. O provador é capaz de checar a validade de uma fórmula fornecida além de gerar um documento em PDF detalhado com mais informações de execução. Para o objetivo deste trabalho a única informação que nos interessa é se a fórmula é `VALID` ou `INVALID`.

Para tanto, foi necessário modificar o código fonte do programa original para que fornecesse apenas a informação desejada ao término da execução. O MUltseq lê de um arquivo chamado *luka* o qual possui as regras necessárias para a Lógica Ł$_3$. As fórmulas passadas para o programa precisam estar no seguinte formato:

```
tf(formula, [t]).
```

onde `formula` é a fórmula que deve ser passada para avaliação, basicamente com a mesma sintaxe exigida pelo provador por nós implementado, e `[t]` é o valor designado da Lógica Ł$_3$ definido no arquivo *luka*.

Capítulo 6

Avaliação Experimental

Uma vez construído o tradutor, podemos agora realizar uma série de testes necessários para comparar o desempenho entre o provador construído e o MUltseq.

6.1 Gerador de Fórmulas

Para fins de teste, foi preciso desenvolver um pequeno programa em C capaz de gerar fórmulas dos mais variados tamanhos nos formatos adequados para serem lidos pelo nosso provador e pelo MUltseq. O tamanho de uma fórmula é definido pelo número de conectivos e variáveis proposicionais que nela ocorrem.

Como parâmetros de entrada definimos o tamanho das fórmulas e o número de fórmulas que se deseja gerar. Como saída, o programa gera dois arquivos texto para cada fórmula, um no formato a ser lido pelo meu provador, e outro no formato a ser lido pelo MUltseq. As fórmulas geradas possuem formatos e número de variáveis completamente aleatórios.

A execução dos programas seguem o fluxograma da Figura 6.1.

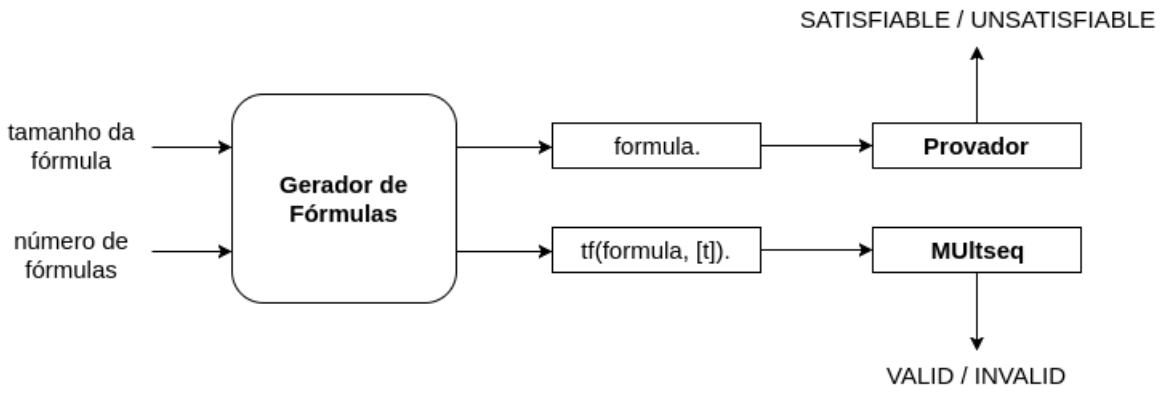

Figura 6.1: Fluxograma de Execução

6.2 Ambiente de Execução

Todos os testes a seguir foram executados na seguinte arquitetura:

- **SO**: Linux elementary OS 0.4.1 Loki [Construído sobre Ubuntu 16.04.3 LTS]
- **CPU**: Dual-Core Intel Core 2 Duo CPU E4500 @ 2.20GHz
- **GPU**: NVIDIA Corporation G84 [GeForce 8600 GT] (rev a1)
- **RAM**: 6,1 GB

6.3 Comparação de Resultados

Os dois provadores fornecem dois tipos de informações diferentes acerca da fórmula fornecida para avaliação. O MiniSAT checa a satisfatibilidade de um fórmula enquanto o MUltseq checa a sua validade. Portanto, é necessário criar-se um paralelo de comparação entre os dois resultados.

- Se uma fórmula é dita *válida* no MUltseq, sua negação no MiniSAT precisa ser *insatifatível*;
- Se uma fórmula é dita *inválida* no MUltseq, sua negação no MiniSAT precisa ser *satisfatível*.

O tradutor foi construído de forma a negar automaticamente qualquer fórmula fornecida. Dessa maneira, podemos comparar o resultado entre os dois provadores, desde que atendam a regra acima.

6.4 Script de Execução

Foi criado um *shell script* para execução automática dos dois programas. Além de automatizar o processo, o *script* fornece também três tipos de tempo de execução necessários para a avaliação. Para cada fórmula testada é possivel obter o tempo *Elapsed*, *System* e *User*.

Tempo *Elapsed* é o tempo decorrido desde o início da execução; *User* fornece o tempo de CPU gasto pelo processo atual, ou seja, a sessão do usuário atual; *System* fornece o tempo de CPU gasto pelo *kernel* (o sistema operacional) em nome do processo atual.

Foi definido também um tempo de *timeout* para a execução dos dois provadores. Caso a avaliação de uma fórmula ultrapasse o tempo de 180 segundos (3 minutos), a execução é interrompida e o *script* testará a próxima fórmula na lista. As fórmulas que não foram

avaliadas por motivo de *timeout* foram registradas, mas não entraram para o cálculo de tempo médio de execução ao fim do experimento de forma que se pudesse obter a estatística apenas dos testes que realmente executaram por completo.

6.5 Testes

Foram realizadas quatro baterias de testes diferentes. Cada teste é composto por 100 fórmulas.

- Teste 1: fórmulas de tamanho entre 5-10;
- Teste 2: fórmulas de tamanho entre 15-20;
- Teste 3: fórmulas de tamanho entre 25-30;
- Teste 4: fórmulas de tamanho entre 35-40.

Cada fórmula foi submetida a ambos os programas já mencionados. Além dos tempos de execução, foram registrados o tamanho e o número de variáveis de cada fórmula e a saída fornecida pelos programas.

Foi calculado o tempo médio de execução, o tempo total e a mediana. Foi contabilizado também o número de fórmulas resolvidas (sem *timeout*), o número de fórmulas satifatíveis/insatisfatíveis e o número de fórmulas válidas/inválidas.

Como dito anteriormente, as fórmulas que não foram avaliadas por motivo de *timeout* durante a execução em qualquer um dos provadores foram registradas, mas não entraram para o cálculo de tempo total/médio de execução.

6.6 Resultados Obtidos

Os resultados dos testes foram catalogados em uma planilha [17] e com eles foram gerados os três gráficos informativos dados nas Figuras 6.2, 6.3 e 6.4.

No Gráfico 6.2 temos o tempo médio de execução dos dois provadores nos quatro testes realizados. Para o *Teste 1* a diferença entre os dois provadores é quase imperceptível. Para os três testes seguintes a diferença entre os dois provadores passa a ser consideravelmente maior. É possível notar que o nosso tradutor em conjunto com o MiniSAT nunca ultrapassa dois segundos médios de execução, enquanto o tempo gasto pelo MUltseq cresce exponencialmente à medida que o tamanho das fórmulas avaliadas aumenta.

No Gráfico 6.3 é apresentado o total de fórmulas executadas com sucesso sem *timeout*. Nota-se que a partir do segundo teste o número de fórmulas que ultrapassaram o limite

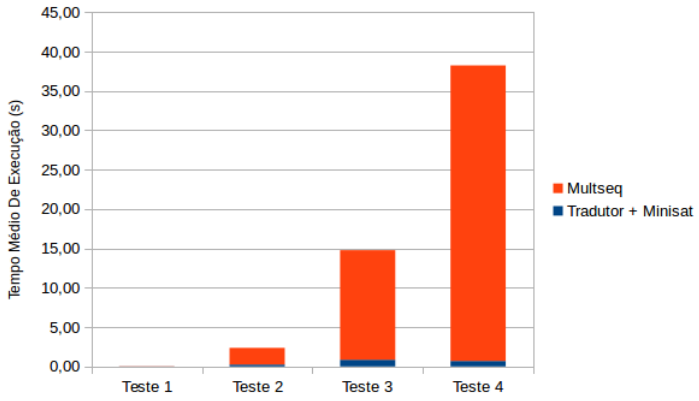

Figura 6.2: Gráfico: Tempo Médio de Execução

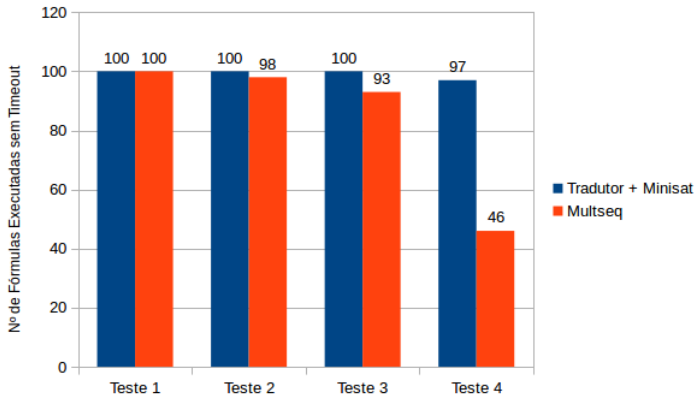

Figura 6.3: Gráfico: Fórmulas Executadas sem Timeout

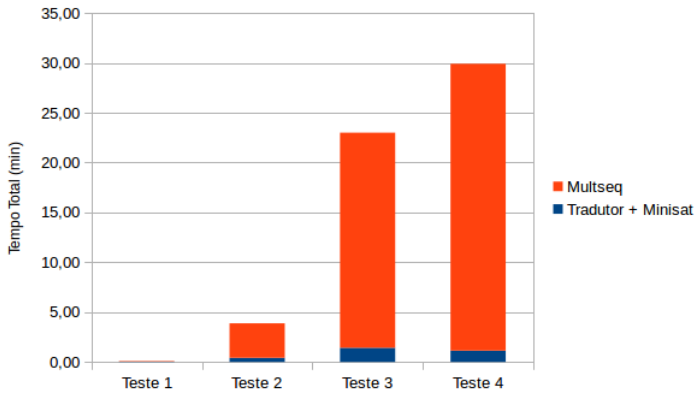

Figura 6.4: Gráfico: Tempo Total de Execução

de três minutos de execução no Multseq aumenta consideravelmente. É só no último teste que o tradutor em conjunto com o MiniSAT começa a apresentar testes com *timeout*. A diferença entre os dois provadores a partir do quarto teste é muito grande e é de se esperar que o aumento do tamanho das fórmulas resulte em um número ainda maior de interrupções por motivo de *timeout* no provador MUltseq.

No Gráfico 6.4 temos o tempo total de execução. Sua estrutura se assemelha bastante ao do primeiro gráfico. O tradutor em conjunto com o MiniSAT não ultrapassa o tempo total de dois minutos enquanto o MUltseq chega a alcançar o patamar de trinta minutos totais no *Teste 4*. É importante notar que no *Teste 4* o MUltseq executou com sucesso menos da metade das fórmulas enviadas para avaliação e mesmo assim registrou trinta minutos totais de execução apenas das fórmulas que foram executadas com sucesso.

Essa grande diferença nos resultados pode ser resultado de diversos fatores. O método de validação utilizado pelo MUltseq poderia ser, em tese, tão rápido quanto o provador por nós implementado. A construção em linguagem Prolog pode ter afetado o desempenho do MUltseq, assim como uma implementação não otimizada por parte dos desenvolvedores. O importante, no entanto, é que nosso provador se mostrou mais eficiente que o MUltseq nos testes realizados. Por esse motivo, consideramos que, entre os dois provadores avaliados, o nosso seja mais recomendado para avaliar fórmulas em Lógica Três-Valorada.

Capítulo 7

Conclusões

Apresentei neste trabalho os fundamentos da lógica infinitamente-valorada de Łukasiewicz, as definições para a formulação da Lógica Três-Valorada, a definição de Tradução Bivalente e o método de Resolução Clausal para fórmulas na Forma Normal Conjuntiva. Com base nessas regras, implementei um programa provador capaz de avaliar a satisfatibilidade de fórmulas em Lógica Três-Valoradas e realizei testes de comparação entre este provador e outro chamado MUltseq.

O método de tradução bivalente e resolução clausal proposto em [1] se verificou bastante eficaz e relativamente mais rápido que o método de validação usado pelo programa MUltseq.

Como se pode observar, o tempo de execução do MUltseq cresce consideravelmente à medida que o tamanho da fórmula avaliada aumenta. Por esse motivo, muitos dos testes alcançaram o limite de *timeout*. O tempo de três minutos definido não foi escolhido por acaso. Em testes isolados feitos antes dos testes definitivos, o MUltseq começava a ultrapassar o limite de memória RAM disponível na máquina de testes após três minutos de execução, o que efetivamente inviabilizava os testes.

Gerenciamento de memória, porém, também é um problema para o nosso tradutor. O processo de tradução bivalente faz com que uma fórmula aumente de tamanho exponencialmente e, consequentemente, o uso da memória RAM também aumenta. É importante notar, também, que nenhum tipo de otimização foi feita durante a implementação do nosso provador. Cláusulas repetidas não são eliminadas e, portanto, nosso provador consome mais memória *RAM* do que deveria. Essa otimização não foi feita pois sua implementação excederia o tempo disponível para completar este trabalho. Cláusulas repetidas são automaticamente ignoradas pelo MiniSAT, o que não compromete o resultado final das avaliações.

Em testes isolados, o nosso tradutor ultrapassou o limite de memória da máquina ao executar fórmulas a partir de tamanho 1000. Mesmo assim, o consumo de memória e o

tempo de execução do nosso provador ainda é mais baixo que o do MUltseq e, portanto, consideramos que seja mais recomendado para avaliar fórmulas em Lógica Três-Valorada.

Por fim, ainda há muito que se otimizar no tradutor. Como dito anteriormente, podemos melhorar o gerenciamento de memória ao eliminar cláusulas repetidas. Podemos também definir parâmetros de execução para limitar a execução de processos intermediários como executar apenas a tradução bivalente ou gerar apenas as cláusulas sem que estas sejam avaliadas pelo MiniSAT.

Tendo em vista o trabalho realizado nesta monografia, considero que os objetivos traçados foram alcançados com sucesso e espero que este trabalho contribua para futuros estudos nas área de Lógicas Multivaloradas e Lógica Três-Valorada.

Referências

[1] Marcos, João e Cláudia Nalon: *Classical resolution for many-valued logics*. Electronic Notes in Theoretical Computer Science, 323:243–270, 2016. v, vi, 3, 4, 9, 12, 25

[2] *MUltseq: Validity of finitely-valued formulas*. `https://www.logic.at/multseq/`, acesso em 2017-06-21. v, vi, 3, 19

[3] Nalon, Cláudia, Ullrich Hustadt e Clare Dixon: *KsP: A resolution-based prover for multimodal k*. Em Olivetti, Nicola e Ashish Tiwari (editores): *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, volume 9706 de *Lecture Notes in Computer Science*, páginas 406–415. Springer, 2016, ISBN 978-3-319-40228-4. 18

[4] Baudry, Laurent, Igor Lukyanchuk e Valerii Vinokur: *Ferroelectric symmetry-protected multibit memory cell*. Em *Scientific Reports*, volume 7, páginas 42–196. Nature Publishing Group, fevereiro 2017. 2

[5] Bofill, Miquel, Felip Manyà, Amanda Vidal e Mateu Villaret: *The complexity of 3-valued łukasiewicz rules*. Em *Modeling Decisions for Artificial Intelligence: 12th International Conference, MDAI 2015, Skövde, Sweden, September 21-23, 2015, Proceedings*, páginas 221–229, Cham, 2015. Springer International Publishing. 3, 4, 5, 6

[6] Bofill, Miquel, Felip Manyà, Amanda Vidal e Mateu Villaret: *Finding hard instances of satisfiability in łukasiewicz logics*. Em *2015 IEEE International Symposium on Multiple-Valued Logic*, páginas 30–35. IEEE Computer Society, May 2015.

[7] Gottwald, Siegfried: *Many-valued and fuzzy logics*. Em Kacprzyk, Janusz e Witold Pedrycz (editores): *Springer Handbook of Computational Intelligence*, páginas 7–29. Springer, 2015, ISBN 978-3-662-43504-5; 978-3-662-43505-2. 2

[8] Hähnle, Reiner, Bernhard Beckert e Stefan Gerberding: *The many-valued theorem-prover*. Relatório Técnico, University of Karlsruhe, Institute for Logic, Complexity and Deduction Systems, 1994.

[9] Robinson, John Alan: *A machine oriented logic based on the resolution principle*. Journal of the ACM, 12:23–41, 1965. 12, 15

[10] Robinson, John Alan: *Automatic deduction with hyper-resolution*. International Jounal of Computer Mathematics, 1:227–234, 1965. 15

[11] Salzer, Gernot e Ventura Verdú: *Generic decision procedures for many-valued logics*. Em *the 6th Kurt Gödel Colloquium*, June 1999. Available at `http://www.logic.at/multseq/kgc99.pdf`. 19

[12] Gensler, Harry J.: *Introduction to Logic*. Routledge, 2016. 1

[13] *GNU Bison*. `https://www.gnu.org/software/bison/`, acesso em 2017-02-22. 16

[14] Paxson, Vern: *FLEX: The fast lexical analyzer*. `https://github.com/westes/flex`, acesso em 2017-02-21. 16

[15] Eén, Niklas e Niklas Sörensson: *The MiniSAT*. `http://minisat.se/`, acesso em 2017-03-05. 16

[16] *SAT competition 2009: Benchmark submission guidelines*. `http://www.satcompetition.org/2009/format-benchmarks2009.html`, acesso em 2017-05-12.

[17] *Table of results*. `http://www.cic.unb.br/~nalon/#software`, acesso em 2017-10-01. 22