UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DIOGO RAPHAEL CRAVO
MATR. 181051

Continuação do desenvolvimento de um provador de teoremas para a lógica clássica de primeira ordem e criação de um raciocinador com o método analítico de Tableau para uma lógica de descrição[1]

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Álvaro Moreira

Porto Alegre
2015

[1] Este trabalho é uma tradução e reapresentação do trabalho entregue no dia 23 de Março de 2015 à Technische Universität Berlin. O trabalho original encontra-se anexo.

DECLARAÇÃO SOB JURAMENTO

Através deste declaro que produzi o presente trabalho por conta própria bem como sem ajuda alheia ilegal e exclusivamente com uso das referidas fontes e recursos.

Porto Alegre,

.....

Diogo Raphael Cravo

AGRADECIMENTOS

Aqui gostaria de agradecer ao Sr. Dr. Fricke pela orientação e todas as reuniões, a todos meus professores pelo conhecimento transmitido, à minha família pelo apoio e à pessoa que corrigiu este trabalho[2].

[2] O original em alemão foi corrigido por um professor de alemão.

RESUMO

Durante o Semestre de Inverno de 2013 a 2014 aconteceu o módulo “Projekt: Symbolische Künstliche Intelligenz”. Neste módulo os estudantes criaram um provador de teoremas para a lógica clássica de primeira ordem. O provador criado na época conseguia através da aplicação de seis diferentes métodos checar a insatisfatibilidade ou validade de fórmulas bem como a consistência de bases de conhecimento. Este Trabalho de Graduação apresenta a continuação do desenvolvimento desse provador de teoremas. Entre as novas funcionalidades do provador estão várias estratégias do método de Tableau, como o Tableau analítico de Smullyan e o Tableau com variáveis livres de Fitting, assim como representações gráficas das provas. Além disto o Software também consegue resolver o problema da subsunção da lógica de descrição ALC. O Software implementado é fácil de usar e vários módulos podem ser reutilizados. Finalmente todas as estratégias utilizadas neste trabalho foram avaliadas por 68 testes, através do que a ineficiência da implementação foi constatada. Melhorias são sugeridas no fim.

Palavras-Chave: Tableau. Provador de Teoremas. Lógica Clássica. Lógicas de Descrição. FOL.

Further development of a theorem prover for classical first-order logic and design of an analytic tableau reasoner for a description logic

ABSTRACT

During the Winter Semester from 2013 to 2014 the Project “Projekt: Symbolische Künstliche Intelligenz” took place. During that project the students have created a theorem prover for the first-order logic. The prover created in this time could check the unsatisfiability or validity of formulas as well as the consistency of knowledge bases by applying six different approaches. This Bachelor’s Thesis presents the further development of that theorem prover. The new prover offers new functionalities, such as the analytical Smullyan Tableau and the free Variable Tableau by Fitting, as well as graphical representations of proofs. Furthermore the Software can solve the subsumption problem of the ALC Description Logic. The implemented Software is easy to use and many of its modules are reusable. Finally all approaches implemented in this work were evaluated by the application of 68 Tests, which revealed the bad performance of the implementation. Improvements are suggested in the end.

Keywords: Tableau. Theorem prover. Classical logic. Description logics. FOL.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

TUB	Technische Universität Berlin
UFRGS	Universidade Federal do Rio Grande do Sul
AOT	Agententechnologien in betrieblichen Anwendungen und der Telekommunikation
FOL	*First-Order Logic* / Lógica de primeira ordem
DL	*Description Logic* / Lógica de descrição
FNC	Forma Normal Conjuntiva
CNF	*Conjunctive Normal Form*
FOF	*First-Order Form* / Forma de Primeira Ordem
sse	se e somente se
DFID	*Depth-First Iterative Deepening search* / Busca em profundidade iterativa

SUMÁRIO

1 INTRODUÇÃO

Mesmo que atualmente um provador de teoremas não consiga resolver sozinho problemas importantes da matemática, provadores de teoremas são muito úteis em aplicações concretas. Sistemas automáticos de prova são bem sucedidos na verificação de certas propriedades de um software. Por exemplo, o estudante ocupou no semestre de inverno entre 2013 e 2014 a cadeira "Software Engineering eingebetteter Systeme" na TUB. Nesta cadeira os estudantes utilizaram o UPPAAL[3], que permite o desenho de sistemas de tempo real com o uso de autômatos temporizados. O estudante participou do desenvolvimento de um simples elevador através do UPPAAL. A relação com a lógica: Através da utilização de lógica temporal os estudantes puderam provar várias propriedades do desenho implementado.

O provador do UPPAAL, mais especificamente um *Model Checker*, carece porém de interatividade. Ele não necessita disso. O provador exposto neste trabalho é por outro lado muito interativo, dado que o usuário pode participar tanto da configuração do processo utilizado como da realização da prova.

Existem vários métodos de prova. Como o título já revela, o método de Tableau é o foco deste trabalho. O Tableau é um método conhecido que pode provar a satisfatibilidade de um conjunto de fórmulas da lógica de primeira ordem. Neste trabalho será ampliado um provador de teoremas interativo. O método de Tableau será implementado para resolver não só o problema da satisfatibilidade da lógica de primeira ordem como também o problema da subsunção de uma lógica de descrição. Inicialmente será dada uma história curta do Tableau. A maior parte das informações foi retirada de D'Agostino *et al*. (1999).

Harrison (2009) toma E. W. Beth e J. Hintikka por conta de seus artigos publicados em 1955 por inventores do método de Tableau semântico aplicável à lógica clássica de primeira ordem. No entanto, Tablôs não eram intuitivos naquela época conforme D'Agostino *et al*. (1999). O Tableau analítico publicado em 1968 por Smullyan foi o primeiro método de Tableau fácil de se aplicar.

Desde então o método de Tableau tornou-se um dos métodos de prova mais utilizados.

[3] Disponível em www.uppaal.org

Tableaus são aplicados segundo D'Agostino *et al*. (1999) à lógica modal, às lógicas multivaloradas, à lógica intuicionista, às lógicas de alta ordem e a muitas outras lógicas. Naturalmente Tableaus também são aplicados a vários tipos de lógicas de descrição (BAADER *et al*. 2003), como a ALC.

Tableaus podem ser refinados de várias formas. O primeiro grande refinamento foi a aplicação da unificação (COHEN *et al*. 1974). Depois veio segundo D'Agostino *et al*. (1999) por Fitting no ano de 1986 uma aplicação precoce da skolemização, que poupa a aplicação da regra δ (veja o capítulo 3). Outros refinamentos acontecem segundo Hähnle (2001) na forma clausal. Hähnle menciona o *Connection Tableau*, no qual a criação de novos ramos efetua-se segundo certas regras. Outras restrições da ramificação são alcançadas pelos métodos conhecidos como *Hypertableau* e *Model Generation*. Ainda mais refinamentos são descritos por Hähnle (2001). Neste trabalho serão contemplados no tocante à lógica de primeira ordem o Tableau de Smullyan, o KE Tableau e dois tipos de Tableau para a FNC (veja capítulo 3). Essas abordagens serão além disso aprimoradas através da unificação. Este trabalho amplia um provador de teoremas, que foi escrito no semestre de inverno entre 2013 e 2014.

1.1 O provador de teoremas escrito no SI 13/14

Durante o semestre de inverno entre 2013 e 2014 realizou-se o evento "Projekt: Symbolische Künstliche Intelligenz" na TU Berlin. Neste evento criou-se um provador de teoremas para a lógica clássica de primeira ordem. Em um grupo de onze pessoas e com a orientação do Dr. Stefan Fricke os estudantes, entre os quais o diplomando estava, pesquisaram o tema, discutiram a estrutura da implementação, implementaram o desenho adequado em código, refinaram este através de heurísticas e finalmente avaliaram o resultado através de *Benchmarks*.

O programa implementado é um provador, que através de diferentes métodos e com uma base de conhecimento fornecida pelo usuário consegue provar ou contradizer expressões lógicas. Os seguintes métodos e heurísticas foram implementados:

- ★ Resolução, refinado através de conjunto de suporte com uma função heurística para seleção de cláusulas, assim como subsunção, fatorização e literais de resposta;
- ★ Encadeamento para trás, refinado através de uma fila de prioridades para preferência de regras pobres em premissas;
- ★ Encadeamento para adiante, que levou ao algoritmo *rete*;
- ★ E *rete*, refinado através da redução da quantidade de uniões de substituições.

Além disso os métodos existentes foram combinados, através do que o encadeamento para adiante e para trás e o encadeamento para trás e *rete* apareceram. O software oferece ainda um conjunto de funcionalidades básicas: uma interface gráfica (GUI), uma gramática e um parser para a lógica de primeira ordem, conversão de fórmulas para a forma normal conjuntiva (FNC) assim como para cláusulas de Horn e unificação e substituição de fórmulas. A implementação aconteceu em Java. O software é livremente expansível através de uma licença Open-Source.

1.2 Objetivos e fins

O objetivo deste Trabalho de Conclusão é expandir o provador de teoremas escrito no semestre de inverno entre 2013 e 2014 através da implementação, refinamento e avaliação de vários métodos de Tableau para a lógica de primeira ordem, assim como de um método de Tableau para a lógica de descrição ALC e da realização de todas as modificações necessárias do provador de teoremas, bem como da implementação de representações gráficas de provas. Para atingir este objetivo uma pesquisa detalhada deve ser realizada.

Durante a elaboração do trabalho o autor aprenderá a projetar e implementar melhor um software. Almeja-se também um melhor entendimento da lógica. Além disso, apesar das abordagens implementadas não serem eficientes, o software produzido poderia ser utilizado em um curso de introdução em lógica.

1.3 Estrutura

Este trabalho é estruturado da seguinte maneira: Primeiro será dada no capítulo 2 uma definição formal da lógica clássica assim como uma curta definição da lógica de descrição ALC. O capítulo 2 esclarece a notação, que será usada neste trabalho. O capítulo 3 apresenta a ideia de um método de Tableau, descreve várias abordagens e compara essas abordagens entre si. O capítulo 4 esclarece o software implementado, inclusive estrutura de classes e algoritmos, e justifica as decisões tomadas. Em seguida o capítulo 5 avalia o software através da realização de testes detalhados e apresenta um resumo dos resultados. Finalmente o capítulo 6 explica tudo que não é considerado neste trabalho. O anexo A apresenta os resultados de todos os testes e o anexo B esclarece o uso do software.

2 A LÓGICA

Neste capítulo será explicada a lógica. Isto é, todas informações necessárias sobre a lógica clássica e as lógicas de descrição serão apresentadas aqui, bem como a correspondente literatura, que lida com cada tema em específico. Se o leitor já estiver familiarizado com a lógica clássica e com a lógica de descrição ALC, este capítulo pode ser pulado. Neste caso este capítulo servirá somente para apresentar a notação utilizada neste trabalho.

Nós começamos com a lógica clássica. **Definições desta lógica são amplamente conhecidas, veja por exemplo Russel e Norvig (2012), Hähnle (2001) e a primeira seção do artigo de Letz (D'AGOSTINO *et al*., 1999).** Essas são as fontes para este capítulo. Com base nessas definições o autor reproduzirá a seguir a definição da lógica de primeira ordem, ressaltando somente os termos e enunciados que forem importantes para este trabalho.

2.1 A lógica clássica

Uma lógica é uma linguagem. Isto é, dadas uma determinada sintaxe e uma determinada semântica frases podem ser construídas. A lógica nos permite atribuir valores a essas frases: os valores-verdade. Através de uma interpretação podem ser criadas regras, que são sempre verdadeiras ou falsas. Quando se aplicam propositadamente regras a conjuntos de expressões lógicas, resultam novas expressões. Este processo é chamado de método de prova e é exatamente o que objetiva-se neste trabalho. Esta definição carece de precisão. Em consequência disso nos dedicaremos agora à formalização dessa definição.

2.1.1 A lógica de primeira ordem: Sintaxe

Uma sintaxe determina que componentes uma linguagem tem e como pode-se combinar esses componentes. Em seguida será dada uma definição do conceito mais abstrato ao mais específico.

DEFINIÇÃO 2.1.1. O Alfabeto. Os símbolos que compõe as fórmulas na lógica são os seguintes:

- ★ O conjunto de todas as variáveis: **V**.
- ★ O conjunto de todas as funções: **F**. A cada elemento f de **F** é atribuída uma quantidade de parâmetros.
- ★ O conjunto de todos os predicados: **P**. A cada elemento p de **P** é atribuída uma quantidade de parâmetros.
- ★ Os conectivos: $\wedge$ (Conjunção), $\vee$ (Disjunção), $\rightarrow$ (Implicação material), $\leftrightarrow$ (Bicondicional)
- ★ Os quantificadores: $\forall$[4] (Quantificador universal), $\exists$[5] (Quantificador existencial)
- ★ A negação ($\neg$[6]), a igualdade (=) e outros símbolos: (,)

DEFINIÇÃO 2.1.2. Fórmula. A lógica de primeira ordem lida com fórmulas. O conjunto de todas as fórmulas bem-formadas **MF** é definido da seguinte maneira: (i) se φ é um literal, então $\varphi \in$ **MF**; (ii) se $\varphi_1, \varphi_2 \in$ **MF** então φ_1 OP $\varphi_2 \in$ **MF**, onde OP $\in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$; (iii) se $\varphi \in$ **MF** e x $\in$ **V** então $\forall$x.$\varphi \in$ **MF** e $\exists$x.$\varphi \in$ **MF**.

DEFINIÇÃO 2.1.3. Literal. Um Literal é ou um átomo (φ) ou um átomo negado através da negação ($\neg\varphi$). Um átomo às vezes é chamado de literal positivo, enquanto um átomo negado é chamado de literal negativo.

DEFINIÇÃO 2.1.4. Átomo. Um átomo é um predicado ou uma equação.

---

[4] No software um quantificador universal é representado através de um @.
[5] No software um quantificador existencial é representado através de um #.
[6] No software a negação é representada através de um ~.

DEFINIÇÃO 2.1.5. Predicado. Um predicado tem um nome, que geralmente é escrito em letras maiúsculas, e um número qualquer (≥ 1) de parâmetros, que são apresentados entre parênteses, e.g. Predicado(p_1, p_2). Na lógica de primeira ordem só podem aparecer termos nos parâmetros de um predicado. Todos os predicados pertencem a **P**.

DEFINIÇÃO 2.1.6. Termo. Termos são todas as constantes, variáveis e funções.

DEFINIÇÃO 2.1.6.1. Constante. Todos e somente os elementos $K \in \mathbf{F}$ com 0 parâmetros são chamados de constantes.

DEFINIÇÃO 2.1.6.2. Variável. Todos e somente os elementos $v \in \mathbf{V}$ são chamados de variáveis.

DEFINIÇÃO 2.1.6.3. Função. Todos e somente os elementos $f \in \mathbf{F}$ com pelo menos um parâmetro são chamados de função. Na lógica de primeira ordem somente termos podem aparecer nos parâmetros de uma função.

DEFINIÇÃO 2.1.7. Equação. Uma equação é a ligação de dois termos através do símbolo da igualdade, e.g. v=f(x).

DEFINIÇÃO 2.1.8. Variáveis livres e atadas. Uma variável atada é uma variável que segue ou a um quantificador universal ou a um quantificador existencial. Toda variável atada é escrita diretamente ao lado do quantificador que a amarra. Se não existirem quantificadores ou se uma variável não é escrita diretamente ao lado de um quantificador, então esta variável é dita variável livre. Essa é no entanto uma definição relaxada, que não esclarece o escopo de uma variável. Fitting (1990, P. 99-100) define o conjunto de todas as variáveis livres **FV** da seguinte maneira: (i) se φ é uma fórmula atômica, então $\mathbf{FV}(\neg\varphi) = \mathbf{FV}(\varphi) = \{v \mid v \in \mathbf{V}$ e v aparece em $\varphi\}$; (ii) se φ, $\psi \in \mathbf{MF}$, então $\mathbf{FV}(\varphi \text{ OP } \psi) = \mathbf{FV}(\varphi) \cup \mathbf{FV}(\psi)$, onde OP $\in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$; (iii) se $\varphi \in \mathbf{MF}$ e $x \in \mathbf{V}$, então $\mathbf{FV}(\forall x.\varphi) = \mathbf{FV}(\exists x.\varphi) = \mathbf{FV}(\varphi) \setminus \{x\}$. Uma variável é atada ou livre, mas nunca ambos.

DEFINIÇÃO 2.1.9. Complemento. Sejam φ, ψ, $\alpha \in \mathbf{MF}$, onde $\psi = \neg\alpha$ e φ é uma fórmula não negada, então o complemento de φ (representado por φ^c) é a fórmula $\neg\varphi$ e o complemento de ψ (representado por ψ^c) é a fórmula α.

Assim foi apresentada a sintaxe da lógica de primeira ordem.

2.1.2 A lógica de primeira ordem: Semântica

A sintaxe define todas as fórmulas bem formadas, mas sem semântica as expressões da lógica não têm relação com a realidade, não fazem sentido. A semântica atribui a cada elemento sintático uma interpretação. Há um domínio U, que contém todas as interpretações, e uma função, que cuida da atribuição. **As fontes para estas definições são Hähnle (2001) e Letz (D'AGOSTINO *et al*., 1999).**

DEFINIÇÃO 2.2.1. Estrutura. Uma estrutura é um par (U,I), onde U é um conjunto e I é uma interpretação. O conjunto U contém todos os elementos que podem ser utilizados. A função I é definida da seguinte maneira: (i) Se $f \in \mathbf{F}$ então $I(f):U^n \rightarrow U$; (ii) Se $P \in \mathbf{P}$ então $I(P) \subseteq U^n$.

DEFINIÇÃO 2.2.2. Atribuição. Uma atribuição é uma função $B:\mathbf{V} \rightarrow U$. Além disso B(v) é a atribuição da variável v. Aqui é ressaltado que a atribuição é independente da estrutura.

DEFINIÇÃO 2.2.3. Verdade. A seguir será utilizada a notação de Hähnle (2001). A lógica clássica tem dois valores-verdade: $\top$ (verdadeiro) e $\bot$ (falso). A cada fórmula φ é atribuído com uma estrutura S e atribuição B exatamente um valor. Se a fórmula φ for verdadeira, escreve-se $(S,B) \vDash \varphi$. No entanto, se ela for falsa, escreve-se *não* $(S,B) \vDash \varphi$ ou simplemente $(S,B) \nvDash \varphi$. É absolutamente possível, mas não necessário, que dadas duas estruturas quaisquer $S_1 \neq S_2$ seja

verdade que $(S_1,B) \models \varphi$, mas $(S_2,B) \not\models \varphi$. O mesmo é verdade para a atribuição, quer dizer $B_1 \neq B_2$, $(S,B_1) \models \varphi$, mas $(S,B_2) \not\models \varphi$. **A fonte para estas definições é Hähnle (2001).**

- ★ Se $P \in \mathbf{P}$ então $(S,B) \models P$ sse. $P' \in I(P)$, onde P' resulta da substituição[7] de todas as variáveis (livres) em P por suas atribuições em B considerando-se o escopo de cada variável.
- ★ Se $\alpha \in \mathbf{MF}$ então $(S,B) \models \neg\alpha$ sse. $(S,B) \not\models \alpha$
- ★ Se $\alpha, \beta \in \mathbf{MF}$ então $(S,B) \models \alpha \wedge \beta$ sse. $(S,B) \models \alpha$ e $(S,B) \models \beta$
- ★ Se $\alpha, \beta \in \mathbf{MF}$ então $(S,B) \models \alpha \vee \beta$ sse. $(S,B) \models \alpha$ ou $(S,B) \models \beta$ ou $(S,B) \models \alpha \wedge \beta$
- ★ Se $\alpha, \beta \in \mathbf{MF}$ então $(S,B) \models \alpha \rightarrow \beta$ sse. $(S,B) \models \neg\alpha$ ou $(S,B) \models \beta$
- ★ Se $\alpha, \beta \in \mathbf{MF}$ então $(S,B) \models \alpha \leftrightarrow \beta$ sse. $(S,B) \models \alpha \wedge \beta$ ou $(S,B) \models \neg\alpha \wedge \neg\beta$
- ★ Se $\alpha \in \mathbf{MF}$, $v \in \mathbf{V}$ então $(S,B) \models \forall v.\alpha$ sse. $(S,B) \models \alpha_1'$, ... , $(S,B) \models \alpha_n'$, onde α_i' resulta da substituição de v pelo elemento $k_i \in U$.
- ★ Se $\alpha \in \mathbf{MF}$, $v \in \mathbf{V}$ então $(S,B) \models \exists v.\alpha$ sse. houver um elemento $k \in U$, tal que $(S,B) \models \alpha'$, onde α' resultado da substituição de v por k.

DEFINIÇÃO 2.2.4. Modelo. Cada par M = (U,I) é chamado de modelo para $\{\varphi_1,...,\varphi_n\}$, desde que U seja um conjunto, I seja um interpretação, $\varphi_1, ..., \varphi_n \in \mathbf{MF}$ e dada qualquer atribuição B $(M, B) \models \varphi_1, ..., (M, B) \models \varphi_n$ valem.

DEFINIÇÃO 2.2.5. Satisfatibilidade e insatisfatibilidade. Se houver um modelo para $W = \{\varphi_1, ..., \varphi_n\}$, então W é chamada de satisfatível. Se não houver modelo, então W é chamada de insatisfatível.

DEFINIÇÃO 2.2.6. Validade. Se todos os pares (U,I) forem modelos para $W = \{\varphi_1, ..., \varphi_n\}$, então W é chamada de válida.

[7] Isto é na verdade uma simplificação. Na literatura é definido aqui o valor de um termo (veja por exemplo FITTING 1990, P. 104-5).

DEFINIÇÃO 2.2.7. Consequência lógica. Segundo Fitting (1990, P. 119) $\psi \in \mathbf{MF}$ é uma consequência lógica de $\{\varphi_1,...,\varphi_n\}$ com $\varphi_1,...,\varphi_n \in \mathbf{MF}$ (representado por $\{\varphi_1,...,\varphi_n\} \vDash \psi$) sse. para todo modelo M, tal que M é um modelo para cada fórmula $\varphi_1,...,\varphi_n$, M também for um modelo para ψ.

2.1.3 A lógica de primeira ordem: Unificação

DEFINIÇÃO 2.3.1. Substituição. Uma substituição $\sigma:\mathbf{V}\rightarrow\mathbf{F} \cup \mathbf{V}$ é uma função, que substitui algumas variáveis de uma fórmula por outros termos. Um exemplo é {A / x, f(x) / y}, uma substituição, que substitui x por A e y por f(x).

DEFINIÇÃO 2.3.2. Unificador. Um unificador σ é uma substituição, tal que $\sigma(F_1)=\sigma(F_2)$, onde F_1 e F_2 são fórmulas e '=' significa, que ambos operandos são sintaticamente idênticos.

DEFINIÇÃO 2.3.2.1. Unificador mais geral. Um unificador mais geral σ^A para duas fórmulas F_1 e F_2 é um unificador, tal que dado um outro unificador σ' com $\sigma'(F_1)=\sigma'(F_2)$, há sempre uma substituição s, tal que $s(\sigma^A(F_1))=\sigma'(F_2)$.

Um bom algoritmo para a computação do unificador mais geral pode ser encontrado em Russel und Norvig (2012).

2.1.4 A lógica de primeira ordem: Notação de Smullyan

No entanto é preciso mais uma ideia: a notação de Smullyan. Smullyan apresenta em seu livro (SMULLYAN, 1968) um tipo de Tableau analítico para a lógica clássica de primeira

ordem. Além disso ele introduz também um tipo de apresentação de fórmulas, que é muito útil na apresentação de regras de dedução. Abaixo (SMULLYAN, 1968) estão as fórmulas α, β, δ e γ, onde A é uma constante e φ, ψ ∈ **MF**:

Figura 2.1 — Notação de Smullyan

α	α_1	α_2
$\varphi \wedge \psi$	φ	ψ
$\neg(\varphi \vee \psi)$	$\neg\varphi$	$\neg\psi$
$\neg(\varphi \rightarrow \psi)$	φ	$\neg\psi$
$\neg\neg\varphi$	φ	φ
$\varphi \leftrightarrow \psi$	$\varphi \rightarrow \psi$	$\psi \rightarrow \varphi$

β	β_1	β_2
$\neg(\varphi \wedge \psi)$	$\neg\varphi$	$\neg\psi$
$\varphi \vee \psi$	φ	ψ
$\varphi \rightarrow \psi$	$\neg\varphi$	ψ
$\neg(\varphi \leftrightarrow \psi)$	$\neg(\varphi \rightarrow \psi)$	$\neg(\psi \rightarrow \varphi)$

δ	δ(A)
$\exists x.\varphi$	$\{A / x\}(\varphi)$
$\neg\forall x.\varphi$	$\{A / x\}(\neg\varphi)$

γ	γ(A)
$\forall x.\varphi$	$\{A / x\}(\varphi)$
$\neg\exists x.\varphi$	$\{A / x\}(\neg\varphi)$

Fonte: Smullyan (1968)

2.1.5 A lógica de primeira ordem: Forma Normal Conjuntiva

A FNC (Forma Normal Conjuntiva) é uma forma particular de expressões lógicas, que engloba um subconjunto de **MF**. Provou-se, que toda fórmula φ ∈ **MF** pode ser transformada em uma fórmula na FNC. Para entender o algoritmo é preciso a definição de função de Skolem.

DEFINIÇÃO 2.5.1. Função Skolem. Uma função f, que tem um nome novo, isto é $f \notin \mathbf{F}$. Uma função Skolem f substitui a variável atada de uma quantificação existencial δ e os parâmetros de f são todas as variáveis atadas a quantificações universais que são ancestrais de δ na árvore sintática.

Um bom algoritmo para a computação da FNC pode ser encontrado em Russel e Norvig (2012). Ao fim obtemos um conjunto de cláusulas. Cada fórmula na FNC é chamada de cláusula.

DEFINIÇÃO 2.5.2. Cláusula. O resultado da transformação na FNC é um conjunto de cláusulas. Cada cláusula é uma fórmula que só contém literais. Os literais de uma cláusula só podem ser ligados através da disjunção. Todas as variáveis de uma cláusula são universalmente quantificadas. As cláusulas são representadas aqui como conjuntos de fórmulas (por exemplo {P(h(A,y)), Q(f(x))}). Neste trabalho as cláusulas serão às vezes abreviadas com $\kappa(x_1, ..., x_n)$, onde $x_1, ..., x_n$ são todas as variáveis na cláusula (por exemplo $\kappa(x,y)$ = {P(h(A,y)), Q(f(x))}). O k-ésimo[8] literal em κ será chamado de κ_k (por exemplo κ_1 = P(h(A,y)), κ_2 = Q(f(x))).

2.2 As lógicas de descrição

Através de lógicas de descrição (abreviatura DL, de *Description Logics*) entende-se um conjunto de lógicas, que basicamente possuem três elementos: indivíduos, classes e papéis. Indivíduos são do ponto de vista semântico como constantes na lógica de primeira ordem, classes são conjuntos de indivíduos e papéis estabelecem relações entre classes, i. e., são conjuntos de pares de indivíduos (BAADER *et al*. 2003). DLs satisfazem uma *open world assumption* (Baader *et al*. 2003, P. 72), isto é, um modelo em uma DL nunca descreverá o mundo inteiro. Assume-se que hajam mais indivíduos, classes, papéis.

[8] A ordem não é importante nesta notação.

Fórmulas de uma lógica de descrição são organizadas em duas caixas, para criar uma base de conhecimento: a TBox (*Terminological Box*), que descreve o mundo através de classes bem como papéis e suas dependências, e a ABox (*Assertional Box*), que descreve os indivíduos e suas respectivas classes e papéis com base na TBox (BAADER *et al*. 2003).

Ao contrário da lógica de primeira ordem, na qual sempre checava-se a satisfatibilidade de uma base de conhecimento e um objetivo, há no caso das lógicas de descrição um conjunto de tipos de inferência. Russel e Norvig (2012) tomam a subsunção (de classes) e a classificação (de indivíduos para classes) como principais.

No total há nas lógicas de descrição oito tarefas de inferência: satisfatibilidade (ou consistência), subsunção, equivalência e disjunção quanto às TBoxes e checagem de consistência, checagem de instância, consulta e classificação quanto às ABoxes. Dado que subsunção, disjunção e equivalência no fundo consistem na tarefa de checagem da satisfatibilidade (BAADER *et al*. 2003, P. 67), um *Reasoner* para essas linguagens com respeito às TBoxes só precisa realizar a tarefa da checagem de satisfatibilidade. Uma descrição mais precisa das tarefas de inferência da ABox está em Baader *et al*. (2003, P. 70-72).

2.2.1 A lógica de descrição ALC

Os três componentes das lógicas de descrição (indivíduos, classes e papéis) são combinados na ALC (***A****ttributive* ***C****oncept Description* ***L****anguage with Complements*) com base nos seguintes elementos sintáticos (BAADER *et al*. 2003): A (um conceito atômico), $\top$ (a tautologia), $\bot$ (a contradição), $\neg$A (a negação de um conceito atômico), $\neg$C (o complemento), C $\sqcap$ D (a interseção de dois conceitos), $\forall$R.C (a quantificação universal), $\exists$R.$\top$ (a quantificação existencial limitada).

Nota-se que a união (C $\sqcup$ D) de conceitos bem como a quantificação existencial geral ($\exists$R.C) não fazem parte da ALC. Estes podem no entanto ser expressos através da combinação do complemento e da conjunção (respectivamente do complemento e da quantificação universal, o que por sua vez permite o uso desses elementos sintáticos (BAADER *et al*. 2003, P. 53-54).

Uma descrição mais precisa da sintaxe e semântica da família das linguagens AL pode ser encontrada em Baader *et al*. (2003, P. 51-52).

2.3 Recapitulação

Este capítulo explicou a lógica de primeira ordem e a lógica de descrição ALC. Agora devem estar claros tanto conceitos e algoritmos como por exemplo satisfatibilidade, unificação e FNC, como também a notação utilizada neste trabalho. Em particular foi apresentada a notação de Smullyan, que será muito útil no próximo capítulo. O capítulo 3 esclarecerá os vários métodos de Tableau.

3 OS MÉTODOS DE TABLEAU

Esta seção explica em geral o que é um algoritmo de tableau, já que serão apresentados vários algoritmos neste trabalho. As definições exatas e observações sobre cada algoritmo serão dadas nas seções subsequentes.

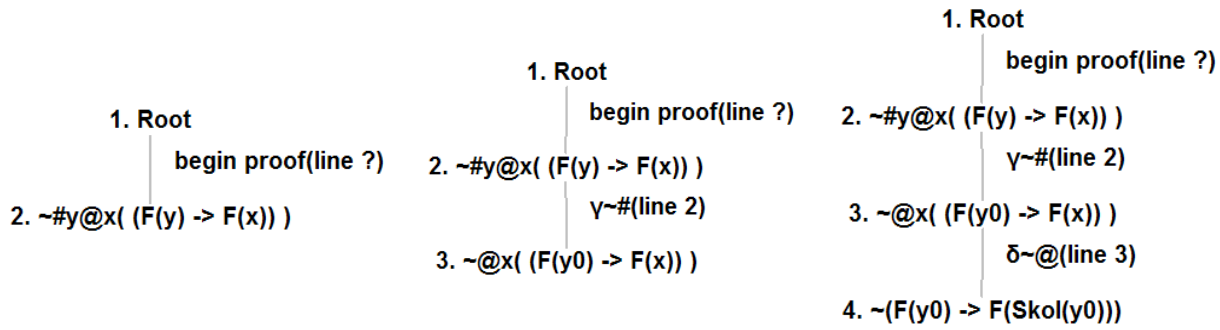
Quando fala-se em método de Tableau, trata-se de um algoritmo, que constrói uma árvore de prova (o Tableau), para checar a satisfatibilidade de um conjunto de fórmulas. Diz-se que Tablôs são uma variante do cálculo de sequentes (D'AGOSTINO *et al*. 1999, S. 22).

Tablôs usam uma técnica de prova semelhante à *reductio ad absurdum*, isto é, eles provam algo através da prova de uma contradição. Mais especificamente, dado um conjunto de fórmulas W (a base de conhecimento) e uma fórmula Z (o objetivo) o Tableau tenta provar a insatisfatibilidade de $W \wedge \neg Z$. Através disso prova-se a validade de $W \rightarrow Z$ (também $\neg W \vee Z$), então $W \models Z$. Em outras palavras o Tableau procura uma prova de que não há modelo M e atraibuição B tal que $(M,B) \models W$ sem que $(M,B) \models Z$.

A árvore começa como uma simples lista de nodos, isto é, no início a árvore tem exatamente um galho. Esses nodos são cada um uma fórmula do conjunto de fórmulas que será checado e que contém tanto fórmulas da base de conhecimento como também o objetivo negado. Um exemplo de um tal Tableau é o primeiro Tableau da próxima figura. A árvore será depois construída passo-a-passo através da aplicação de certas regras a seus nodos. O conjunto exato de regras aplicáveis depende do tipo de Tableau (veja as próximas seções). A figura seguinte mostra um Tableau aberto para o problema #18[9] de Pelletier (1986) e o resultado da aplicação de duas regras a este Tableau:

[9] O problema é $\models_{Tableau} \exists y \forall x(F(y) \rightarrow F(x))$. Neste problema, o Tableau não contém base de conhecimento, somente o objetivo negado.

Figura 3.1 — Exemplo de um Tableau.



Fonte: autoria própria.

Na literatura (HÄHNLE, 2001) as regras são geralmente classificadas em 4 classes: α, β, δ e γ, exatamente como a classificação de fórmulas apresentada em 1963 por Smullyan (veja seção 2.1.4). As regras α não criam novos galhos, mas apenas adicionam nodos a um galho já existente, e são formadas a partir de fórmulas conjuntivas. As regras β podem criar vários galhos e são formadas a partir de fórmulas disjuntivas. As regras δ e as regras γ são formadas respectivamente a partir de quantificações existenciais e universais. As regras δ pressupõe a criação de uma função de Skolem, enquanto as regras γ pressupõe a atribuição de uma variável. Como existem diferentes atribuições para uma variável, uma regra γ pode ser aplicada normalmente uma quantidade infinita de vezes. O algoritmo tem duas condições de parada:

I. **Tableau fechado**: Se o algoritmo encontrar em um galho da árvore uma fórmula e seu complemento, então o galho está fechado e a galhos fechados não podem ser aplicadas regras. Quando todos os galhos da árvore estiverem fechados, então o algoritmo pára e a insatisfatibilidade de W ∧ ¬Z está provada.

II. **Falta de regras de dedução**: Se não for mais possível aplicar alguma regra, então o algoritmo pára. Caso todos os galhos estiverem fechados, então a insatisfatibilidade de W ∧ ¬Z está provada, desde que o algoritmo aplicado seja correto e completo. Este estado

sem regras aplicáveis só pode ser atingido, quando a regra γ for aplicada conscientemente ou quando ela não for aplicada.

A corretude e completude de um método de Tableau são muito importantes. Corretude significa que sempre que um Tableau fechado for encontrado, o objetivo é uma consequência lógica da base de conhecimento. A completude também é importante. Um algoritmo é completo, quando para cada objetivo que é consequência de uma base de conhecimento, existir um Tableau fechado correspondente. Os algoritmos de Tableau são em geral corretos. A completude de cada algoritmo depende no entanto do conjunto de regras aplicáveis. Algumas abordagens utilizam poucas regras de dedução. A restrição das regras de inferência faz com que essas abordagens sejam incompletas (HÄHNLE, 2001).

Como a lógica de primeira ordem **não é decidível**, não existe contudo um algoritmo de Tableau, que sempre termine depois de uma quantidade finita de passos[10]. Como mencionado anteriormente, esse problema pode ser explicado através da aplicação infinita da regra γ. Para que um Tableau fechado seja encontrado em tempo aceitável, uma boa estratégia tem que ser desenvolvida, que procure por esse Tableau. Hähnle (2001) confronta-se com esse problema e sugere fechar os galhos somente quando a árvore inteira puder ser fechada. Isso requer algumas modificações nas regras de dedução, que serão esclarecidas em breve. Nas próximas seções serão apresentados alguns tipos de Tableau.

3.1 Os métodos de Tableau para a lógica de primeira ordem

O primeiro método apresentado aqui é o Tableau de Smullyan. Por causa de sua regra γ este método não é eficiente. O KE Tableau modifica as regras do Tableau de Smullyan, de modo que somente através da aplicação da regra de corte o Tableau ramifica. Em seguida serão apresentadas variantes da regra γ com variáveis livres e finalmente o Tableau clausal.

[10] Às vezes vale $(S,B) \models \varphi$ para algum φ se e somente se U na estrutura S = (U,I) for infinito. Smullyan (1968, P. 63) comenta este caso e alerta que neste caso o método de Tableau não termina.

3.1.1 Tableau de Smullyan

O Tableau de Smullyan (também conhecido como Tableau sistemático ou analítico) utiliza as seguintes regras (SMULLYAN, 1968):

Figure 3.2 — O Tableau de Smullyan.

α	β
α_1	$\beta_1 \mid \beta_2$
α_2	
δ	γ
$\delta(A)$	$\gamma(A)$
onde A é um novo parâmetro[11]	onde A é um termo

Fonte: Smullyan (1968)

No Tableau de Smullyan é importante notar que nenhuma unificação acontece. Ao invés disso na aplicação de uma regra γ as variáveis são substituídas por um termo. O Tableau de Smullyan é completo (SMULLYAN 1968, P. 57-9).

3.1.2 KE Tableau

O KE Tableau (D'AGOSTINO UND MONDADORI, 1994) parece-se muito com o Smullyan. A diferença mais importante entre os dois é que no KE exclusivamente a regra de corte (aqui abreviada SR) cuida da ramificação. Os autores afirmam que por causa disso o KE Tableau é capaz de produzir provas mais curtas. As regras δ e γ do Tableau de Smullyan podem

[11] Smullyan menciona parâmetros na sua definição de lógica de primeira ordem. Parâmetro significa na sua definição o mesmo que constante. Esse fato é destacado por BECKERT *et al.* (1993). Então a noção de parâmetro aqui não deve ser confundida com os parâmetros de uma função ou de um predicado.

ser aplicadas também no KE Tableau (e portanto serão omitidas). O KE Tableau[12] utiliza as seguintes regras:

Figura 3.3 — O KE Tableau.

$$\frac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{}{\varphi \mid \neg\varphi}\ SR \qquad \frac{\begin{array}{c}\beta\\ \beta_1^{\,c}\end{array}}{\beta_2} \qquad \frac{\begin{array}{c}\beta\\ \beta_2^{\,c}\end{array}}{\beta_1}$$

Fonte: D'Agostino e Mondadori (1994)

Além disso os autores recomendam em seu artigo (D'AGOSTINO e MONDADORI, 1994), que a regra de corte seja aplicada somente a fórmulas β, como na próxima figura.

Figura 3.4 — As regras de dedução $SR_{\beta 1}$ e $SR_{\beta 2}$.

$$\frac{\beta}{\beta_1 \mid \beta_1^{\,c}}\ SR_{\beta 1} \qquad \frac{\beta}{\beta_2 \mid \beta_2^{\,c}}\ SR_{\beta 2}$$

Fonte: D'Agostino e Mondadori (1994)

Tanto a variante do KE Tableau com a regra SR como também a variante com as regras $SR_{\beta 1}$ e $SR_{\beta 2}$ são completas (D'AGOSTINO und MONDADORI, 1994).

3.1.3 Tableau com variáveis livres

Hähnle (2001) recomenda para o Tableau da lógica de primeira ordem o método apresentado por Fitting em seu livro (FITTING, 1990). Fitting reconhece a dificuldade com as

[12] Os autores utilizam fórmulas anotadas. Cada fórmula é anotada ou com um T ou com um F. Uma fórmula anotada com um T é verdadeira. Uma fórmula anotada com um F é por outro lado falsa. As anotações servem para a proximidade das fórmulas com o cálculo de sequentes. As regras serão no entanto aqui apresentadas sem anotações, para que sejam mais fáceis de se comparar com os outros métodos.

aplicações infinitas da regra γ e modifica ela para que a quantidade de aplicações diminua. Ele sugere inserir uma nova variável livre ao invés de utilizar termos diretamente. Esta nova variável inserida pode unificar com qualquer termo, dado que ela é nova no Tableau.

Essa modificação da regra γ pressupõe contudo uma modificação da regra δ, como mencionado por Fitting (1990, P. 139). As constantes que forem inseridas por uma regra δ têm que ser sempre constantes novas. Elas não podem aparecer em nenhum outro lugar no Tableau. Fitting explica que a constante inserida através da aplicação da regra δ só será certamente nova, quando as atribuições das variáveis livres já inseridas através de regras γ forem consideradas. Isto é fácil de se entender, quando imaginamos que essas variáveis livres podem unificar com qualquer termo. Se elas não fossem consideradas, uma unificação posterior poderia fazer com que uma daquelas constantes não fosse mais única no Tableau. O Tableau com variáveis livres e essa regra δ é completo (FITTING, 1990).

Figura 3.5 — O Tableau com variáveis livres.

$$\frac{\delta}{\delta(f(x_1, ..., x_n))}$$

onde f é uma nova função de Skolem e $x_1,..., x_n$ são todas as variáveis livres inseridas através de γ

$$\frac{\gamma}{\gamma(x)}$$

onde x é uma nova variável livre

Fonte: Fitting (1990).

No entanto, a condição da regra δ pode ser melhorada (BECKERT *et al.*, 1993) utilizando-se somente as variáveis livres da fórmula δ ao invés de todas as variáveis livres inseridas. A regra resultante[13] dessa nova condição chama-se δ^+. Esta é a proposta de Hähnle, cujo algoritmo é correto (HÄHNLE 2001, P. 114) e completo (HÄHNLE 2001, P. 121).

[13] O artigo lida principalmente com a regra δ^{++}. Como essa regra não foi implementada neste trabalho, mas a regra δ^+ foi, somente a regra δ^+ será mencionada.

3.1.4 Tableau clausal

Tableau clausal designa um método de Tableau cuja base de conhecimento e cujo objetivo são cláusulas. Hähnle (2001) menciona vários tipos de Tableau clausal. Como cláusulas não contém quantificações existenciais, a regra δ pode ser poupada neste tipo de Tableau. Caso a entrada seja na FNC, a regra α também é poupada e a única regra de dedução é a seguinte, onde κ é uma cláusula segundo Def. 2.5.2:

Figura 3.6 — O Tableau da FNC.

$$\frac{\kappa(x_1, ..., x_n)}{\sigma\kappa_1 \mid ... \mid \sigma\kappa_k}$$

onde $\sigma = \{t_1 / x_1, ..., t_n / x_n\}$ é uma substituição segundo Def. 2.3.1
e $t_1, ..., t_n$ são novas variáveis livres.

Fonte: Hähnle (2001) e autoria própria[14]

Hähnle (2001) menciona também *hypertableaus*. Esses são um tipo especial de Tableau clausal, que aplicam somente uma regra. A regra exata depende da abordagem. Aqui será apresentada somente a regra planejada para este trabalho. A hiperregra tem a seguinte aparência, onde cada φ_i é um literal positivo e cada ψ_i é um literal negativo:

Figura 3.7 — O *hypertableau*.

$$\frac{\begin{array}{c}\{\varphi_1, ..., \varphi_k, \psi_{k+1}, ..., \psi_n\} \\ \psi_{k+1}{}^c \\ ... \\ \psi_n{}^c\end{array}}{\varphi_1 \mid ... \mid \varphi_k}$$

Fonte: Hähnle (2001) e autoria própria[15]

[14] Esta é a regra que Hähnle utiliza em seu exemplo na página 127. Hähnle explica essa regra no entanto através de uma descrição textual.

[15] Muitas variantes do *hypertableau* são apresentadas por Hähnle nas páginas 138-9.

3.1.5 Outros Tablôs

Muitos métodos, como o *Connection Method*, *Block Tableau* ou o *Model Generation* não são considerados neste trabalho. Também outros tipos de *hypertableau* não são aplicados, mas somente um tipo limitado. Na literatura há também melhorias, formas de se planejar um Tableau mais eficiente (veja GIESE 2003). O método de Tableau é particularmente eficiente nas lógicas de descrição. O Tableau para a ALC é explicado na seção 3.2.

3.2 O método de Tableau para a ALC

Um Tableau para a ALC deve checar se um conceito subsume outro. Queremos saber, se indivíduos de uma certa classe C também são indivíduos de outra classe C, ou seja se $C \sqsubseteq D$. Isso prova-se criando-se um tableau para a fórmula $(C \sqcap \neg D)(x)$ (BAADER e SATTLER 2000).

Assim como no Tableau para a lógica de primeira ordem, são aplicadas regras no tableau da ALC para construir a árvore (BAADER e SATTLER 2000):

Figura 3.8 — O Tableau para a ALC.

$$\frac{(C_1 \sqcap C_2)(x)}{\begin{array}{c} C_1(x) \\ C_2(x) \end{array}} \rightarrow_{\sqcap} \qquad \frac{(C_1 \sqcup C_2)(x)}{C_1(x) \mid C_2(x)} \rightarrow_{\sqcup}$$

$$\frac{(\exists r.C)(x)}{\begin{array}{c} C(y) \\ r(x,y) \end{array}} \rightarrow_{\exists} \qquad \frac{\begin{array}{c}(\forall r.C)(x) \\ r(x,y)\end{array}}{C(y)} \rightarrow_{\forall}$$

Fonte: Baader e Sattler (2000) e autoria própria[16]

[16] As regras são apresentadas através de uma descrição.

Apesar de ser impossível entrar em um laço infinito na ALC (BAADER e SATTLER 2000, Lema 1.1), não se abre mão da busca em profundidade iterativa[17] em função do uso de memória. O verdadeiro algoritmo é por causa disso com exceção da aplicação das novas regras parecido com o algoritmo para a lógica de primeira ordem. Por este motivo ele é omitido neste trabalho.

3.3 Estratégia de busca e fechamento do Tableau

Como já mencionado no início deste capítulo a estratégia de busca é uma parte importante de um algoritmo de Tableau. A tarefa consiste em aplicar tão poucas regras quanto possível, a fim de obter o menor Tableau fechado tão rápido quanto possível. Pressupõe-se uma função, que escolhe as regras que serão aplicadas de um universo de todas as regras aplicáveis. Se houvesse um algoritmo perfeito, esta função de escolha selecionaria sempre a regra certa dentre todas as regras disponíveis. Escolher a regra certa no entanto não é tão fácil. Por via de regra as próximas heurísticas são úteis:

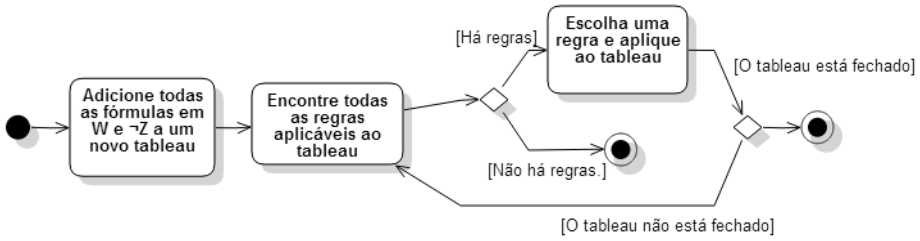
- ★ **α, δ e γ antes de β[18]:** Regras que não criam novos galhos, mas que criam somente novos nodos, têm prioridade sobre regras que ramificam. Particularmente as regras α de Smullyan têm prioridade sobre regras β de Smullyan. Isto é fácil de perceber, quando pensamos nas aplicações dessas regras. Quando aplica-se primeiro uma regra β, regras α têm que ser aplicadas a vários galhos. Quando todas as regras α forem aplicadas primeiro, então haverá somente um galho ao qual as regras β podem ser aplicadas.
- ★ **Regras excedentes que não ramificam:** Quando todas as consequências de uma regra que não ramifica já estiverem em um galho, então a aplicação dessa regra não faz sentido. As consequências precisam no entanto ser sintaticamente idênticas às fórmulas nos galhos, uma unificação não é suficiente.
- ★ **Regras excedentes que ramificam:** Quando pelo menos uma consequência de uma regra que ramifica estiver em um galho, então a aplicação desta regra a esse galho não faz

[17] Baader e Sattler (2000) mencionam uma busca em profundidade.
[18] Aqui entende-se β como a regra β de Smullyan, as regras SR, $SR_{\beta 1}$ e $SR_{\beta 2}$ mas não a regra β do KE Tableau.

sentido. A consequência no entanto deve ser sintaticamente idêntica às fórmulas do galho, uma unificação não é suficiente.

O seguinte diagrama de atividades representa o algoritmo.

Figura 3.9 — A estratégia de busca.

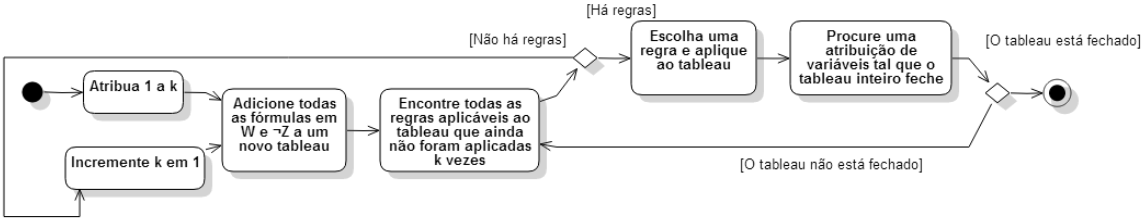


Fonte: autoria própria.

O estado sem regras aplicáveis não será atingido, a menos que a aplicação da regra γ seja limitada. Para limitar a aplicação da regra γ será utilizada uma busca em profundidade iterativa (inglês DFID), como sugerido por Hähnle (2001) e Harrison (2009). Essa estratégia de busca limita as aplicações da regra γ a um número k **por galho**. Em seguida o algoritmo é executado e, caso no fim da execução nenhuma Tableau seja encontrado, k será incrementado.

O fechamento do Tableau também é importante. Diz-se que o fechamento é um problema NP-completo (GIESE, 2003). O fechamento de um Tableau de Smullyan no entanto não é NP-completo: Assim que o par φ e seu complemento são encontrados em um galho, esse galho pode ser fechado e não aplicam-se mais regras a esse galho. A dificuldade para fechar um Tableau encontra-se no método de Fitting. Como as variáveis livres podem ter várias atribuições, nenhum galho pode ser fechado sozinho, senão seria possível que a atribuição que fecha um galho impedisse o fechamento de outro galho. Este fato é chamado por Hähnle (2001) de Tableau destrutivo. A solução para este problema é sempre fechar o Tableau inteiro através do uso de uma única atribuição, ou seja, resolver um problema NP-completo. O algoritmo é representado na figura seguinte.

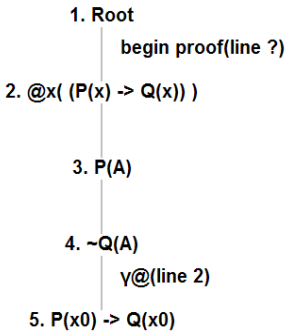
Figura 3.10 — A estratégia de busca com DFID.



Fonte: autoria própria.

Há no entanto um outro problema, que surge da combinação de regras com mais de uma premissa e as regras δ e γ para o ableau com variáveis livres de Fitting. Digamos que haja uma regra com premissas φ, ψ ∈ **MF** e alguma conseqüência. Isto é, devemos encontrar no Tableau duas fórmulas φ' e ψ' tal que haja um unificador mais geral σ e σ(φ) = σ(φ') e σ(ψ) = σ(ψ'). Como mencionado há pouco, a unificação só é aplicada no fechamento de um Tableau, o que faz com que regras com mais de um parâmetro que pressupõe unificações nunca sejam aplicadas, dado que as fórmulas no Tableau simplesmente nunca unificam (exemplo na próxima figura). Na figura, a regra β do KE Tableau com conseqüência $\{A / x_0\}\ Q(x_0)$ não pode ser aplicada à linha 5, dado que x_0 não pode unificar com A.

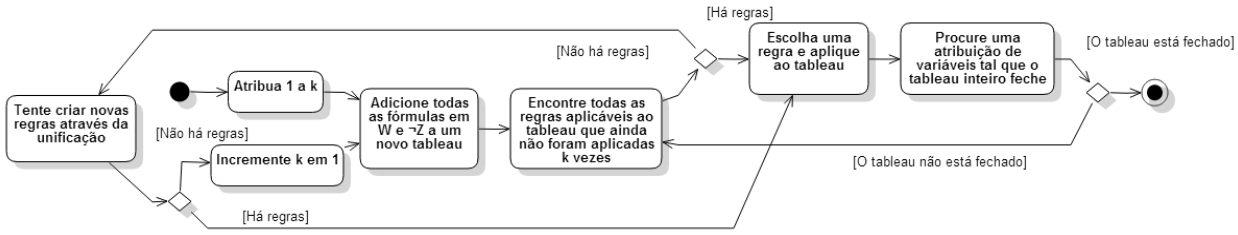
Figura 3.11 — Aplicação da regra β do KE.



Fonte: autoria própria.

No exemplo exibido é possível encontrar uma prova sem a aplicação da regra β. No entanto há exemplos mais complicados, nos quais a prova não é tão fácil de se encontrar[19]. Portanto uma última modificação deve ser feita no algoritmo. Foi adicionado um passo que procura por regras novas. Essas novas regras só são possíveis através do uso da unificação.

Figura 3.12 — A estratégia de busca DFID com unificação.



Fonte: autoria própria.

3.4 Igualdade

A igualdade é uma relação especial que possui três propriedades: reflexividade, simetria e transitividade. Para usar a igualdade também neste trabalho serão utilizadas as seguintes regras (FITTING, 1990):

Figura 3.13 — As regras de dedução para a igualdade.

$\dfrac{t = u \quad \varphi(t)}{\varphi(u)}$ onde $\varphi(t)$ e $\varphi(x)$ são fórmulas atômicas	$\dfrac{}{x = x}$ onde x é uma variável livre	$\dfrac{}{f(x_1, ..., x_n) = f(x_1, ..., x_n)}$ onde $x_1, ..., x_n$ são variáveis livres e f é uma função de Skolem

Fonte: Fitting (1990).

[19] Deve haver um exemplo para o qual não haja prova. Contudo não consegui encontrar prova para a existência de um exemplo assim. Caso ache-se um exemplo assim, então o algoritmo de KE com variáveis livres sem a aplicação da unificação na criação de regras novas é incompleto.

As condições para a criação da função de Skolem são iguais às condições da regra δ^{+}.

3.5 Recapitulação

Neste capítulo tratamos do método de Tableau. Vimos que o Tableau com variáveis livres de Fitting é por via de regra melhor do que o Tableau original de Smullyan. Além disso uma busca em profundidade iterativa foi sugerida para limitar as aplicações da regra γ. O fechamento de um Tableau no entanto continua neste trabalho um problema difícil. O próximo capítulo apresenta a implementação.

4 O PROVADOR DE TEOREMAS

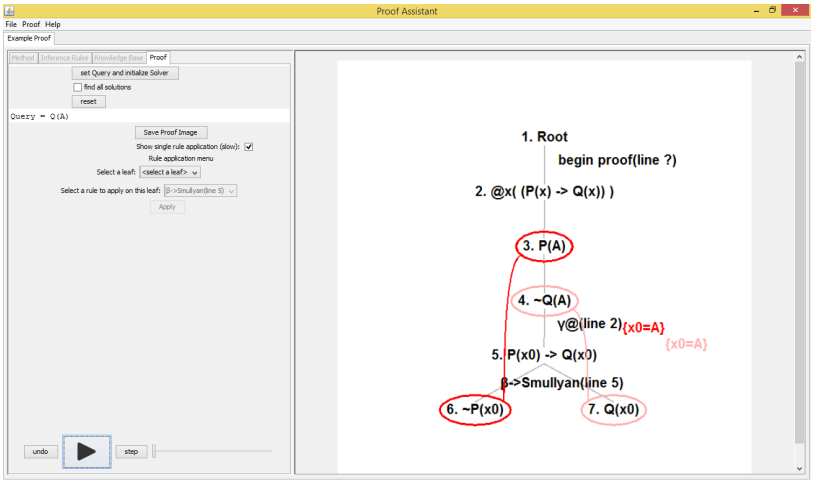
Neste capítulo serão apresentadas todas as novas funcionalidades, assim como os algoritmos implementados, testes e a hierarquia de classes final.

4.1 As novas funcionalidades

4.1.1 A representação gráfica da prova

Um novo tipo de representação das provas foi implementado. Essa é uma representação gráfica, muito parecida com a representação de Letz (D'AGOSTINO *et al.*, 1999). A implementação toma uma árvore como entrada para criar uma imagem. A aplicação da unificação e de várias regras de dedução pode ser exibida na imagem. Além disso a implementação oferece a possibilidade de utilizar este tipo de representação também nos outros métodos de prova (não só o Tableau), desde que esses métodos utilizem a mesma estrutura de dados, i. e., uma árvore. A próxima imagem mostra um exemplo da representação gráfica para a prova de $\forall x(P(x) \rightarrow Q(x)), P(A) \vDash_{Tableau} Q(A)$.

Figura 4.1 — Representação gráfica da prova.



Fonte: autoria própria.

Para uma descrição mais exata da GUI, veja por favor o anexo B - manual do usuário.

4.1.2 As lógicas de descrição

Uma árvore sintática abstrata da lógica de descrição ALC foi criada. O *parser* também foi ampliado para reconhecer também fórmulas da ALC. O Tableau que resolve o problema da subsunção da lógica ALC também foi evidentemente implementado.

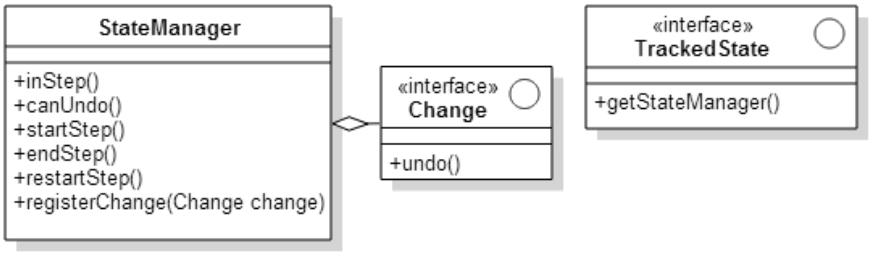
4.1.3 Testes

Muitos testes foram utilizados para encontrar erros no Tableau. Todos os 68 primeiros testes no artigo de Pelletier (1986) foram implementados com JUnit. Esses testes podem agora ser reutilizados para encontrar erros em outros métodos de prova.

4.1.4 StateManager

O fato de que o usuário pode guiar a prova arbitrariamente traz grandes dificuldades consigo. O estado de várias classes tem que ser armazenado e deve ser possível desfazer mudanças. Como algumas modificações dependem de outras, e portanto estas modificações devem ser todas desfeitas de uma só vez em um *undo*, é razoável que essas mudanças sejam armazenadas em um passo inteiro. O armazenamento e a anulação desses passos são executados pela classe StateManager.

Figura 4.2 — O pacote StateManager.



Fonte: autoria própria[20].

A classe StateManager e as interfaces Change e TrackedState são reutilizáveis.

4.1.5 Outras melhorias

Pequenas melhorias foram realizadas por todo o software. A classe da GUI foi dividida em partes menores e a possibilidade de executar várias provas simultaneamente foi implementada. O *parser* reconhece também igualdades.

4.2 Projeto e algoritmos

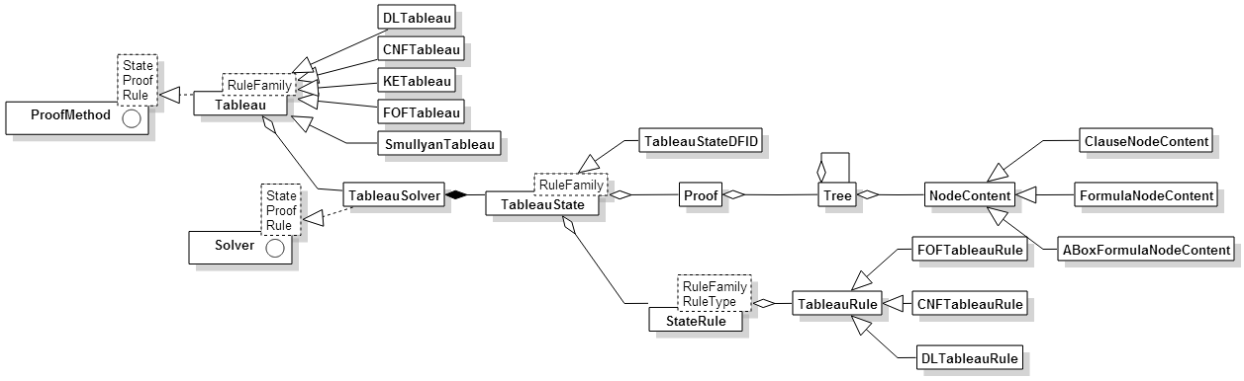
Esta seção explica e justifica o projeto do software além de expor os algoritmos implementados.

4.2.1 Hierarquia de classes

Na figura seguinte está representada a hierarquia de classes do software. Uma explicação do projeto segue.

[20] Ferramenta UML: StarUML, disponível em http://staruml.io/.

Figura 4.3 — A hierarquia de classes.



Fonte: autoria própria[21].

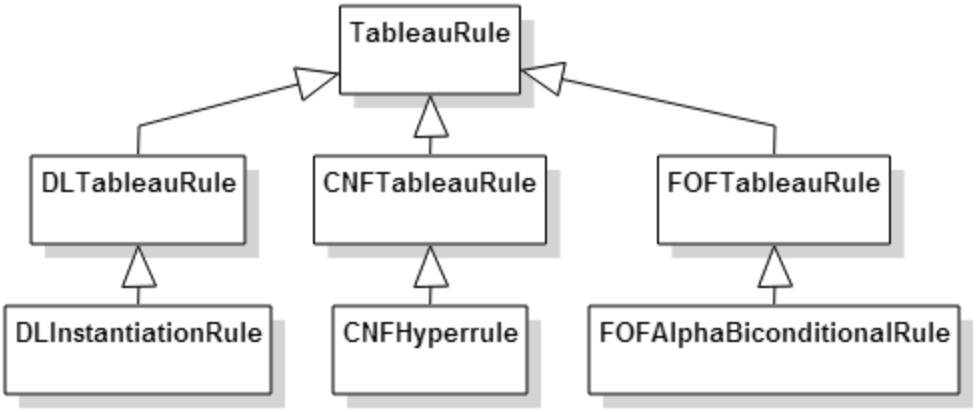
Como pode ser visto na figura, a classe Tableau implementa a interface ProofMethod, enquanto a classe TableauSolver por sua vez implementa a interface Solver. ProofMethod e Solver são duas interfaces, que foram desenvolvidas pelo time durante o projeto para que a GUI pudesse apresentar todos os métodos de prova diferentes. Através do reuso dessas interfaces a integração do Tableau na GUI é facilitada.

Há também vários tipos de Tableau. SmullyanTableau e KETableau utilizam uma lista pré-definida de regras de inferência para que o usuário não precise ajustar as regras de inferência sempre que for executar uma prova. O usuário pode no entanto ajustar as regras de inferência como desejar. As classes DLTableau, CNFTableau e FOFTableau permitem que o usuário utilize as regras que desejar, desde que essas regras pertençam ao conjunto de regras correspondente (respectivamente DLTableauRule, CNFTableauRule e FOFTableauRule).

Toda as regras de inferência herdam os métodos e propriedades da classe abstrata TableauRule. Há três tipos de regras de inferência: FOFTableauRule, CNFTableauRule e DLTableauRule. As verdadeiras regras de inferência foram omitidas da figura e elas herdam métodos e propriedades das classes abstratas FOFTableauRule, CNFTableauRule e DLTableauRule.

[21] Ferramenta UML: StarUML, disponível em http://staruml.io/.

Figura 4.4 — As classes das regras de dedução.



Fonte: autoria própria[22].

As regras também podem criar outras regras depois de aplicadas. Quando por exemplo aplica-se uma regra α à fórmula $P(f(x)) \wedge (P(h(y)) \rightarrow Q(A))$, através do que as fórmulas $P(f(x))$ e $P(h(y)) \rightarrow Q(A)$ originam-se, uma regra β para a fórmula $P(h(y)) \rightarrow Q(A)$ também deve ser criada. Esta é a responsabilidade da classe TableauRule. O método *getGeneratedRulesOfType()* cria novas regras. Além disso TableauRule determina se uma regra pode ser aplicada a uma folha específica.

A prova (Proof) é basicamente uma árvore. Todos os nodos dessa árvore carregam um objeto NodeContent. Como há vários tipos de NodeContent, é possível representar também várias lógicas através dessa estrutura de dados.

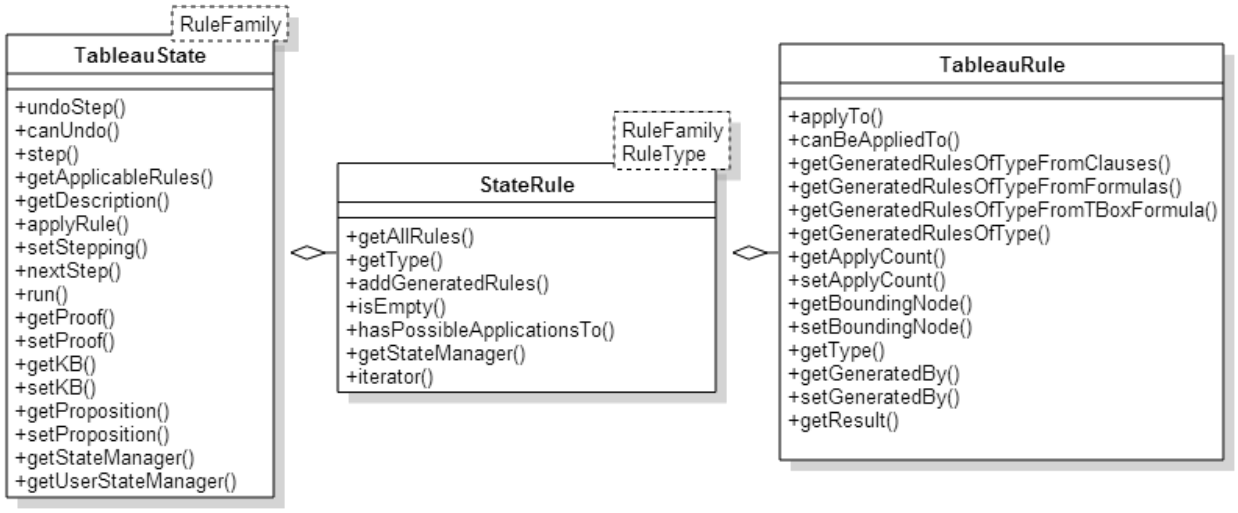
A classe TableauSolver utiliza a classe TableauState, que executa o verdadeiro algoritmo. A classe TableauState tem uma lista de classes StateRule. Cada objetivo StateRule compõe exatamente um tipo de regra de inferência, mais precisamente a classe StateRule tem dois parâmetros: o RuleFamily e o RuleType. RuleFamily é uma das classes FOFTableauRule, CNFTableauRule e DLTableauRule. RuleType é uma classe que herda da classe RuleFamily.

TableauState oferece um conjunto de funções. O método *step()* executa um passo inteiro, checando todas as árvores com um determinado número de aplicações de regras. O método

[22] Ferramenta UML: StarUML, disponível em http://staruml.io/.

applyRule() permite ao usuário aplicar uma regra. Além disso o método *setStepping()* inicia um estado, no qual o usuário e o sistema constroem juntos a prova.

Figura 4.5 — TableauState, StateRule e TableauRule.



Fonte: autoria própria[23].

A classe StateRule é um tipo de fila de regras. Ela contém uma lista com todas as regras de inferência de seu tipo, disponibiliza essas regras através de um iterador em fila e cuida da aplicação das regras. Cada regra pode ser aplicada uma quantidade restrita de vezes. Quando essa quantidade é atingida, então a regra não é mais oferecida pela classe StateRule. A regra pode ser oferecida novamente assim que um *undo* for realizado ou a quantidade restrita de aplicações for aumentada. Através do uso da classe StateRule é muito fácil utilizar um conjunto arbitrário de regras de inferência. Basta criar um objeto StateRule por regra.

A TableauState é além disso uma classe abstrata. A classe TableauStateDFID herda da classe TableauState e implementa uma busca em profundidade iterativa (DFID). Outras estratégias podem também ser implementadas, escrevendo-se outras classes herdeiras de TableauState. Finalmente a relação TableauState-StateRule permite uma priorização dos tipos de regras, isto é, algumas regras serão aplicadas sempre, assim que disponíveis, antes de outras regras. A prioridade exata pode ser ajustada pelo usuário à vontade.

[23] Ferramenta UML: StarUML, disponível em http://staruml.io/.

4.2.2 Algoritmos

A busca em profundidade iterativa da seção 3.3 foi implementada, bem como o fechamento do Tableau inteiro, ao invés de galhos isolados. As regras implementadas foram combinadas com essa busca, de onde quatro algoritmos originaram-se:

- ★ O Tableau de Smullyan utiliza as regras α e β da seção 3.1.1 e as regras com variáveis livres δ^+ e γ da seção 3.1.3.
- ★ O KE Tableau utiliza as regras α e β da seção 3.1.2, bem como as regras $SR_{\beta 1}$ e $SR_{\beta 2}$, mas não as regras SR, e as regras com variáveis livres δ^+ e γ da seção 3.1.3.
- ★ O Tableau da FNC utiliza a primeira regra da seção 3.1.4 e uma versão mais fraca dessa regra, que só lida com fórmulas sem variáveis.
- ★ O *hypertableau* utiliza a hiperregra apresentada na seção 3.1.4.

Todos os algoritmos utilizam também as regras para a igualdade da seção 3.4. Outras abordagens podem ser criadas, basta que o usuário escolha as regras.

4.3 Recapitulação

Este capítulo apresentou a representação gráfica da prova bem como outras funcionalidades. Além disso a hierarquia de classes foi explicada e várias vantagens do projeto foram mencionadas, como a configuração das regras de dedução ou a separação entre estratégia de busca e regras de dedução: A classe TableauState sozinha pode ser combinada com vários tipos de regras de dedução para criar vários algoritmos. O próximo capítulo apresenta os resultados dos testes.

5 AVALIAÇÃO

Neste capítulo o software produzido será avaliado, apresentando-se os resultados dos testes e destacando-se as propriedades do software relativamente à qualidade.

5.1 Corretude e completude

O artigo de Pelletier (1986) apresenta 75 testes. Eles são problemas lógicos, que têm cada um alguma particularidade: Alguns são difíceis de se resolver com alguma abordagem específica, outras são difíceis de se transformar na FNC, outros são simplesmente longos demais. Os primeiros 68 testes foram implementados. Eles checam se o programa é correto. Os testes #71 a #75 determinam a eficiência do programa. Desses testes escolheu-se e implementou-se o teste #71. A tabela com os resultados de todos os testes encontra-se nos anexos.

Como o processo pode levar muito tempo em alguns testes, foi colocado um limite de tempo. Se o problema não for resolvido em um minuto, assume-se que o problema não pode ser resolvido em tempo aceitável. Além disso os seguintes algoritmos foram testados:

- ★ Tableau de Smullyan com unificação;
- ★ KE Tableau com unificação;
- ★ *Hypertableau* com unificação;
- ★ Tableau na FNC com unificação.

O computador que executou os testes é um acer Aspire E 15 (Modelo E5-571G-3188). A duração dos testes foi medida através da classe Timeout[24]. A duração contém também o tempo de criação da representação gráfica da prova. Como nenhuma ferramenta de *profiling* foi utilizada, mas simplemente JUnit, as medições feitas nos testes não são confiáveis. Portanto os resultados serão resumidos aqui. A duração de cada teste é individualmente apresentada no anexo A.

[24] Caminho exato org.junit.rules.Timeout

Tabela 5.1 — Resultados dos testes de correção.

Abordagem	QTBS[25] dentro de 5 Seg.	QTBS dentro de 1 Min.
Smullyan	26 / 68	38 / 68
KE	25 / 68	31 / 68
Hypertableau	29 / 68	31 / 68
Tableau na FNC	30 / 68	30 / 68

Fonte: autoria própria.

Pelletier (1986) organiza seus testes em cinco classes. Cada classe destina-se a um certo subconjunto da lógica de primeira ordem, tal que problemas da primeira classe são muito mais fáceis de resolver-se do que problemas da última classe. As classes são:

- ★ Lógica proposicional (aqui chamada de K_1) contém problemas #1 a #17;
- ★ Lógica de primeira ordem monádica (aqui chamada de K_2) contém problemas #18 a #34;
- ★ Lógica de primeira ordem sem equações e sem funções (aqui chamada de K_3) contém problemas #35 a #47;
- ★ Lógica de primeira ordem com equações e sem funções (aqui chamada de K_4) contém problemas #48 a #55;
- ★ Lógica de primeira ordem com equações e funções (aqui chamada de K_5) contém problemas #56 a #70;

Nota-se que a maioria dos testes bem sucedidos foram também os testes mais fáceis, como percebe-se na próxima figura.

[25] Quantidade de testes bem sucedidos.

Tabela 5.2 — Testes bem sucedidos por lógica.

Abordagem	K_1	K_2	K_3	K_4	K_5
Smullyan	17 / 17	11 / 17	5 / 13	2 / 8	3 / 15
KE	17 / 17	7 / 17	5 / 13	2 / 8	0 / 15
Hyper	17 / 17	7 / 17	5 / 13	1 / 8	0 / 15
FNC	17 / 17	7 / 17	3 / 13	2 / 8	2 / 15

Fonte: autoria própria.

Do fato de que alguns testes não foram bem sucedidos pode-se concluir que o software é incompleto[26]. O motivo disso é o limite de tempo. Pode ser que os testes fossem bem sucedidos com um limite maior de tempo. Isso é indicado pelos resultados dos testes para a abordagem de Smullyan.

Finalmente os mesmo testes de Pelletier foram executados com uma pequena diferença. O objetivo de cada teste individual foi negado nesta fase antes do início da prova, de modo que os problemas não poderiam mais ser resolvidos. Por exemplo o problema #59 é o seguinte (PELLETIER, 1986):

★ $\forall x.\ F(x) \leftrightarrow \neg F(f(x)) \vDash \exists x.\ F(x) \wedge \neg F(f(x))$

O problema com o objetivo negado seria então:

★ $\forall x.\ F(x) \leftrightarrow \neg F(f(x)) \vDash \forall x.\ \neg F(x) \vee F(f(x))$

Todos os quatro métodos foram aplicados a esses problemas, mas para nenhum[27] desse 67 novos problemas encontrou-se uma prova. Isso indica a correção do software.

[26] Um método completo termina depois de uma quantidade finita de passos e encontra um Tableau fechado, desde que haja um Tableau fechado.

[27] Para o teste #25 encontrou-se uma prova, mas isso deve-se ao fato de que as premissas do teste #25 são contraditórias.

No caso da ALC testou-se exclusivamente a correção do método através de um teste disponível na internet. Esse teste será omitido neste trabalho.

5.2 Testes de eficiência

Como mencionado na seção anterior, o teste #71 (PELLETIER, 1986) determina a eficiência do programa. O problema que deve ser resolvido é o seguinte:

Tabela 5.3 — Teste de Pelletier #71.

Problema 1	$\models_{\text{Tableau}} P_1 \leftrightarrow P_1$
Problema 2	$\models_{\text{Tableau}} P_1 \leftrightarrow (P_2 \leftrightarrow (P_1 \leftrightarrow P_2))$
Problema 3	$\models_{\text{Tableau}} P_1 \leftrightarrow (P_2 \leftrightarrow (P_3 \leftrightarrow (P_1 \leftrightarrow (P_2 \leftrightarrow P_3))))$
Problema n	$\models_{\text{Tableau}} P_1 \leftrightarrow (P_2 \leftrightarrow (P_3 \leftrightarrow ... (P_n \leftrightarrow (P_1 \leftrightarrow (P_2 \leftrightarrow (P_3 \leftrightarrow ... (P_{n-1} \leftrightarrow P_n))))$

Fonte: Pelletier (1986)

A tarefa consiste em deduzir como o algoritmo comporta-se com entradas que ficam cada vez maiores. Na tabela seguinte os dados ocorrem em segundos.

Tabela 5.4 — Resultados dos testes de eficiência.

Método	P1	P2	P3	P4	P5	P6	P7
Smullyan	< 1	~2	~14	~219	F[28]	-	-
KE	< 1	< 1	~4	~17	~55	~901	F[29]
Hyper	< 1	< 1	~11	F[30]	-	-	-
FNC	< 1	< 1	~10	F[31]	-	-	-

Fonte: autoria própria.

Há aqui uma diferença clara entre os métodos da FNC e os métodos da FPO (Forma de Primeira Ordem). Como todas as entradas são na FPO, os métodos da FNC precisam executar a transformação na FNC primeiro. Essa transformação no entanto é muito difícil e leva tanto tempo e consome tanta memória que os testes falham rápido. A transformação na FNC é realmente uma fraqueza desses métodos.

Por outro lado no Tableau de Smullyan e no KE Tableau a eficiência é um pouco melhor. O Tableau de Smullyan ainda consegue resolver P4. Isso dá-se devido à entrada na FPO. O KE Tableau pode resolver até P6 dentro de menos de uma hora. Isso certamente está ligado ao fato de que o KE evita a ramificação e portanto cria provas mais curtas, como os autores dizem (D'AGOSTINO e MONDADORI, 1994).

5.3 Resultados

Não só a eficiência e correção do software são importantes. Há também um conjunto de propriedades que são tão importantes quanto, como a usabilidade e a manutenibilidade. Na tabela

[28] Depois de uma hora nenhum resultado foi encontrado e o teste por este motivo foi interrompido.
[29] Depois de uma hora nenhum resultado foi encontrado e o teste por este motivo foi interrompido.
[30] Depois de 20 Minutos uma exceção OutOfMemoryError foi jogada. A transformação na FNC ainda não estava pronta.
[31] Depois de 26 Minutos uma exceção OutOfMemoryError foi jogada. A transformação na FNC ainda não estava pronta.

seguinte será dada uma curta análise crítica do software relativamente à qualidade. Como nenhuma análise detalhada foi feita, esta análise serve somente para descrever o software. Nenhuma métrica de software foi utilizada. As propriedades foram retiradas do caderno de requisitos do projeto do semestre de inverno entre 2013 e 2014.

Tabela 5.5 — Qualidade dos resultados.

Correção	A correção do software foi provada através dos testes da seção 5.1.
Robustez	Os testes escritos checam exclusivamente a correção das classes do tableau. Nenhum teste foi criado para checar cada módulo separadamente. Portanto a robustez do software não está garantida.
Eficiência	O software não é eficiente. Isso foi provado através dos testes que falharam. Como a maioria dos testes era tão simples, o software também deveria ter conseguido resolvê-los dentro de um minuto.
Usabilidade	A representação gráfica das provas facilita o entendimento. Além disso o software é interativo.
Manutenibilidade	A hierarquia de classes melhora a manutenibilidade e extensibilidade. Por exemplo as classes TableauState e TableauRule podem ser facilmente modificadas. A implementação das regras de dedução no entanto poderia ser mais curta. As regras apresentadas no capítulo 3 foram todas implementadas, através do que um conjunto de 41 classes originou-se. Essa quantidade gigante poderia ser menor, se as classes fossem mais genéricas. Assim o software seria mais fácil de manter-se.
Documentação	Javadoc foi utilizado em muitas classes para tornar o código compreensível. Além disso o usuário pode folhear o anexo B deste trabalho para entender melhor a GUI.

Fonte: autoria própria.

5.4 Recapitulação

Neste capítulo os resultados dos testes foram apresentados. Vimos que o software é correto, embora ineficiente. Além disso muitas propriedades do software foram discutidas. O capítulo 6 apresenta possíveis aperfeiçoamentos.

6 CONCLUSÃO E PERSPECTIVAS

O software desenvolvido no semestre de inverno entre 2013 e 2014 era muito facilmente modificável. Por esse motivo a integração aconteceu sem problemas. No entanto a implementação não se deu sem dificuldades. Durante o trabalho o diplomando reconheceu vários desafios e procurou suas soluções ou desenvolveu-as sozinho. Muitos desses desafios infelizmente não foram resolvidos.

Um dos desafios existentes é o fechamento do Tableau. Como já mencionado, encontrar o unificador certo com o qual o Tableau fecha é um problema NP-completo. Esse problema é resolvido através de força bruta neste trabalho. Uma melhor alternativa seria a abordagem de Giese (2003). Giese armazena o resultado da última tentativa de fechar-se o Tableau para que a próxima tentativa seja muito mais fácil.

Um outro desafio foi o projeto da estratégia de busca. Nem todas as melhorias possíveis foram implementadas, por exemplo o algoritmo checa depois de cada regra aplicada a aplicabilidade de cada regra a todas as folhas da prova mais uma vez. Isso poderia ser melhor, se somente a aplicabilidade da regra às novas folhas fosse checada. Além disso, a limitação da regra γ, da maneira que foi implementada, não é conveniente. Imaginemos que haja uma fórmula com várias quantificações, por exemplo $\forall x \exists y \exists z.\ \neg((P(y) \rightarrow Q(z)) \rightarrow (P(x) \rightarrow Q(x)))$ (o problema negado #19 de Pelletier, 1986). Mesmo que a regra γ só possa ser aplicada duas vezes, cada aplicação da regra γ cria uma nova regra δ, já que há várias quantificações. Essas novas regras δ podem ser aplicadas cada uma duas vezes e na sua aplicação serão criadas ainda mais regras δ. É evidente que com poucas quantificações a quantidade de regras cresce depressa. As quantificações são presumivelmente o motivo pelo qual as abordagens da FPO, a saber o Tableau de Smullyan e o KE Tableau, não conseguiram resolver o problema #19 dentro de um minuto, enquanto as abordagens da FNC conseguiram em pouco tempo.

Na estratégia de busca a escolha da regra que deve ser aplicada também é importante. Essa decisão acontece neste trabalho através de uma priorização das regras de dedução ajustada pelo usuário. Às vezes essa priorização no entanto está simplesmente errada. Por exemplo as

regras de reflexividade para a igualdade foram aplicadas[32] muitas vezes pela abordagem do *hypertableau* no problema #60, apesar de essas regras não contribuirem para o fechamento do Tableau. Ainda, quanto mais regras são aplicadas, maior fica o tableau e assim mais difícil fica o problema de checar-se a aplicabilidade de uma regra. Uma boa solução para esse problema seria o *Connection Tableau* (HãHNLE, 2001). O *Connection Tableau* permite uma aplicação de regra somente quando essa aplicação contribuir para o fechamento do Tableau.

Na avaliação foi mecionado que o código precisa ser reestruturado. Particularmente ruim é o estado das classes de regras de dedução, há simplesmente demais. Seria possível aplicar a notação de Smullyan, algo do tipo *Formula.getFirstAlphaComponent*, *Formula.getSecondAlphaComponent* etc. Através disso não seria mais necessário criar uma classe para cada tipo de fórmulas α, β, δ e γ, ou seja 13 classes, mas somente uma classe para as α, uma para as β, uma para as δ e uma para as fórmulas γ. Além disso outras classes são simplesmente longas, complicadas e incompreensíveis. Essas classes precisam de alguma maneira ser divididas e reestruturadas.

Finalmente, há ainda depois da correção de todas as falhas e implementação de todas as melhorias sugeridas muito que pode ser feito. O presente trabalho implementou e comparou quatro abordagens, mas há numerosas outras abordagens não só para a lógica clássica, como também para lógicas não-clássicas. Além disso há outras formas de representar a prova. Se a representação em árvore for grande demais e portanto incômoda, talvez as matrizes de Hãhnle (P. 121-2, 2001) sejam uma boa ideia.

[32] Todas as provas criadas através dos testes serão entregue junto com este trabalho em um disco.

REFERÊNCIAS

BAADER, F.; MCGUINESS, D. L.; NARDI, D; PATEL-SCHNEIDER, P. F. **The Description Logic Handbook: Theory, implementation, applications**. Cambridge: Cambridge University Press, 2003.

BAADER, F; SATTLER, U. Tableau Algorithms for Description Logics. In: DYCKHOFF, R. (Hrsg.). **Automated Reasoning with Analytic Tableaux and Related Methods: International Conference, TABLEAUX 2000**. St Andrews, Schottland, UK, Juli 3-7. Berlin: Springer, 2000.

BECKERT, B; HÄHNLE, R; SCHMITT, P. H. **The Even More Liberalized δ-Rule in Free Variable Semantic Tableaux**. doi:10.1007/BFb002255. 1993.

D'AGOSTINO, M.; GABBAY, D. M.; HÄHNLE, R.; POSEGGA, J. (Hrgs). **Handbook of Tableau Methods**. doi:10.1007/978-94-017-1754-0. 1999.

D'AGOSTINO, M.; MONDADORI, M. **The Taming of the Cut. Classical Refutations with Analytic Cut.** doi: 10.1093/logcom/4.3.285. 1994.

FITTING, M. **First-Order Logic and Automated Theorem Proving.** New York u.a.: Springer. 1990. ISBN: 3-540-97233-1

GALMICHE, D.; LARCHEY-WENDLING, D. (Hrgs.). **Automated Reasoning with Analytic Tableaux and Related Methods: 22nd International Conference, TABLEAUX 2013**, Nancy, Frankreich, 16-19 September 2013. Berlin: Springer-Verlag Heidelberg, 2013.

GIESE, M. **Incremental Closure of Free Variable Tableaux**. Verfügbar über http://www.uio.no/studier/emner/matnat/ifi/INF5170/h03/filer/giese_handout.pdf. Der 14. November 2003.

HÄHNLE, R. Tableau and Related Methods. In: Robinson, A., Voronkov, A. **Handbook of Automated Reasoning I** (S. 103-137). Elsevier Science Publishers B. V, 2001.

HARRISON, J. **Handbook of Practical Logic and Automated Reasoning**. Cambridge: Cambridge University Press, 2009.

PELLETIER, F. J. **Seventy-five Problems for Testing Automatic Theorem Provers**. doi:10.1007/BF02432151. 1986.

PELLETIER, F. J. **Errata**. Journal of Automated Reasoning. ISSN: 0168-7433. 1988.

PELLETIER, F. J.; SUTCLIFFE, G. **An Erratum for Some Errata to ATP Problems**. 10.1023/A:1005764705033. 1997.

RUSSEL, S.; NORVIG, P. **Künstliche Intelligenz: Ein moderner Ansatz**. München: Pearson Deutschland GmbH. 2012.

SCHMIDT-SCHAUBß, M.; SMOLKA, G. **Attributive Concept Descriptions with Complements**. doi:10.1016/0004-3702(91)90078-X. Feb. 1991.

SMULLYAN, R. M. **First-Order Logic**. New York: Springer. 1968.

SUTCLIFFE, G.; SUTTNER, C. The State of CASC. **AI Communications, 19**. Abgerufen von http://iospress.metapress.com/content/aymnb5mq1fqy69c9/. 2006.

SUTCLIFFE, G.; SUTTNER, C. **The TPTP Problem Library for Automated Theorem Proving**. http://www.cs.miami.edu/~tptp/. Zugegriffen: Oktober 2014. 2014.

ANEXO A - TESTES

O diplomando utilizou os testes de um artigo científico (PELLETIER, 1986) para avaliar a própria implementação. Alguns dos testes no dado artigo no entanto estavam mal escritos. Correções foram publicadas pelo mesmo autor (PELLETIER, 1988; PELLETIER e SUTCLIFFE, 1997). Em resumo, as fontes para todos os testes são dadas na próxima tabela.

Tabela 1 — Fontes dos testes.

Fonte	Número do teste
Pelletier 1986	1-14, 16-27, 29-33, 35-39, 41-53, 56-59, 60, 61, 63-68, 71
Pelletier 1988	15, 28, 34, 40, 54, 55
Pelletier e Sutcliffe 1997	62
Testes não utilizados	69, 70, 72-75

Fonte: autoria própria.

Na tabela seguinte estão os resultados de todos os 67 testes de Pelletier (1986) implementados. Nessa tabela estão representados os resultados de cada problema individualmente. Todas as representações gráficas das provas, que foram criadas através dos testes, serão entregues em um disco junto com este trabalho.

Se um teste é bem sucedido, será dado o tempo de computação. Se um teste no entanto falhar por falta de tempo, será exibido um Z. Todos os dados são apresentados em segundos. Alguns testes não podem ser resolvidos por alguma abordagem, porque por exemplo não há regras aplicáveis. Esses testes serão marcados[33] com um X. Se o teste por algum outro motivo falhar (por exemplo *Stack Overflow*) será exibido um F.

[33] Na verdade é uma tarefa difícil deduzir, se um problema pode ser resolvido por uma certa abordagem incompleta. Ainda mais, se não houver prova para a incompletude. Na seção 3.3 foi apresentada a DFID. Lá foi escrito, que as aplicações das regras são limitadas a um número k. Aqui assume-se, que uma abordagem não pode resolver um problema tão logo k = 10.

Tabela 2 — Resultados dos testes.

Problema	Smullyan	KE	Hypertableau	Tableau da FNC
#1	1.884	1.922	1.385	1.793
#2	0.258	0.286	0.154	0.419
#3	0.172	0.178	0.139	0.126
#4	0.464	0.402	0.229	0.223
#5	0.508	0.381	0.192	0.287
#6	0.159	0.130	0.098	0.097
#7	0.182	0.208	0.065	0.080
#8	0.294	0.167	0.104	0.107
#9	0.822	0.337	0.233	0.502
#10	1.342	0.807	0.507	0.888
#11	0.296	0.237	0.071	0.076
#12	9.128	4.457	1.349	2.167
#13	1.477	1.776	0.350	0.700
#14	1.492	1.258	0.264	0.570
#15	0.664	0.481	0.101	0.293
#16	0.356	0.302	0.148	0.164
#17	2.556	1.074	0.464	1.024

#18	0.355	0.414	0.086	0.103
#19	Z	Z	0.117	0.283
#20	2.380	2.506	0.148	1.244
#21	10.917	Z	0.422	1.154
#22	1.510	0.780	0.475	Z
#23	0.783	0.636	1.343	Z
#24	Z	Z	0.786	1.494
#25	2.457	Z	Z	Z
#26	Z	Z	Z	Z
#27	Z	Z	Z	Z
#28	5.037	Z	Z	Z
#29	Z	Z	Z	Z
#30	1.870	3.864	Z	Z
#31	3.403	12.474	Z	4.735
#32	8.950	Z	Z	Z
#33	7.190	6.105	Z	5.559
#34	Z	Z	Z	Z
#35	0.769	1.260	0.347	0.869
#36	2.884	7.265	X	2.700

#37	Z	Z	X	Z
#38	Z	Z	Z	Z
#39	2.155	1.738	0.445	1.914
#40	Z	5.504	1.537	Z
#41	5.744	3.843	1.953	Z
#42	Z	Z	Z	Z
#43	Z	Z	X	Z
#44	9.245	Z	1.094	Z
#45	Z	Z	X	Z
#46	Z	Z	X	Z
#47	Z	Z	Z	Z
#48	5.294	6.442	2.342	4.550
#49	Z	Z	X	Z
#50	5.885	6.827	X	2.340
#51	Z	Z	Z	Z
#52	Z	Z	Z	Z
#53	Z	Z	Z	Z
#54	Z	Z	Z	Z
#55	Z	Z	Z	Z

#56	9.296	Z	Z	Z
#57	6.574	Z	Z	4.062
#58	Z	Z	Z	X
#59	Z	Z	Z	11.935
#60	6.876	Z	Z	Z
#61	Z	Z	Z	X
#62	Z	Z	Z	Z
#63	Z	Z	Z	Z
#64	Z	Z	Z	Z
#65	Z	Z	Z	Z
#66	Z	Z	Z	Z
#67	Z	Z	Z	Z
#68	Z	Z	Z	Z
QTBS[34] 5 Seg.	26 / 68	25 / 68	30 / 68	29 / 68
QTBS 1 Min.	38 / 68	31 / 68	30 / 68	31 / 68

Fonte: autoria própria.

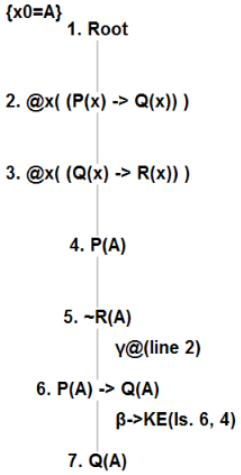
[34] Quantidade de testes bem sucedidos.

ANEXO B - MANUAL DO USUÁRIO

Este manual explica as funções do provador de teoremas. Apesar do título “manual do usuário” o foco aqui não é o uso do software, mas as funções disponibilizadas pelo software. Evidentendente somente o método de Tableau será apresentado.

As provas são representadas graficamente como árvores. Um exemplo é a figura seguinte No canto superior esquerdo de cada imagem é exibida a substituição atual. No exemplo, a substituição é {A / x0}. Perceba, que mesmo se o Tableau ainda estiver aberto, como o Tableau na imagem, será associada uma substituição a esse Tableau. Além disso a substituição será aplicada a todas as variáveis no Tableau. Por exemplo a variável livre “x0” da linha 6 na figura 1 foi substituída por “A”.

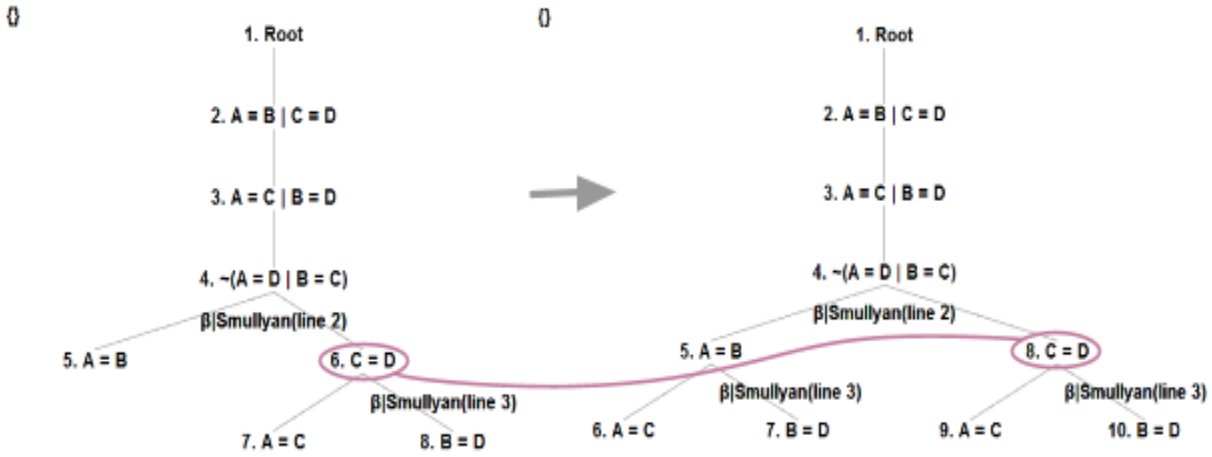
Figura 1 — A representação gráfica.



Fonte: autoria própria.

Na imagem percebe-se também duas outras particularidades da representação gráfica: As linhas são numeradas e as regras aplicadas são dadas. Na numeração das linhas o caminhamento em profundidade é utilizado. O algoritmo é executado para cada árvore de prova. Isso leva ao fato de que os números das fórmulas podem mudar depois de uma aplicação de regra. No exemplo, a aplicação de uma regra β à linha 5 faz com que a linha 6 torne-se linha 8.

Figura 2 — Numeração das linhas.



Fonte: Problema #48 de Pelletier (1986) e autoria própria.

Exemplos de aplicação de regras podem ser encontrados tanto na figura 1 como na figura 2. Ao lado da regra é dada a linha que criou esta regra. Para criar uma regra, todas as premissas desta regra devem estar presentes. Se houver mais de uma premissa, então são dadas todas as linhas. Um exemplo disso é a aplicação da regra β na figura 1. A cada regra é associado um nome que depende das fórmulas que criam essa regra. A tabela completa dos nomes é a seguinte:

Tabela 1 — Nomes das regras de dedução.

Lógica	**Regra de dedução**	**Nome**
Lógica clássica (Forma de Primeira Ordem)	$\dfrac{\varphi \wedge \psi}{\varphi, \psi}$	α&

Lógica clássica (Forma de Primeira Ordem)	$\dfrac{\neg(\varphi \vee \psi)}{\varphi, \psi}$	α~|
	$\dfrac{\neg(\varphi \rightarrow \psi)}{\varphi, \psi}$	α~->
	$\dfrac{\neg\neg\varphi}{\varphi}$	α~~
	$\dfrac{\varphi \leftrightarrow \psi}{\varphi \rightarrow \psi, \psi \rightarrow \varphi}$	α<->
	$\dfrac{\neg(\varphi \wedge \psi)}{\varphi \mid \psi}$	β~&Smullyan
	$\dfrac{\varphi \vee \psi}{\varphi \mid \psi}$	β|Smullyan
	$\dfrac{\varphi \rightarrow \psi}{\neg\varphi \mid \psi}$	β->Smullyan
	$\dfrac{\neg(\varphi \leftrightarrow \psi)}{\neg(\varphi \rightarrow \psi), \neg(\psi \rightarrow \varphi)}$	β~<->Smullyan
	$\dfrac{}{\varphi \mid \neg\varphi}$	Cut
	$\dfrac{\neg(\varphi \wedge \psi) \quad \neg\varphi}{\psi}$	β~&KE
	$\dfrac{\neg(\varphi \wedge \psi) \quad \neg\psi}{\varphi}$	

Lógica clássica (Forma de Primeira Ordem)	$\varphi \vee \psi$ $\neg\varphi$ ——— ψ	β|KE
	$\varphi \vee \psi$ $\neg\psi$ ——— φ	
	$\varphi \rightarrow \psi$ φ ——— ψ	β->KE
	$\varphi \rightarrow \psi$ $\neg\psi$ ——— φ	
	$\neg(\varphi \leftrightarrow \psi)$ $\varphi \rightarrow \psi$ ——— $\neg(\psi \rightarrow \varphi)$	β~<->KE
	$\neg(\varphi \leftrightarrow \psi)$ $\psi \rightarrow \varphi$ ——— $\neg(\varphi \rightarrow \psi)$	
	$\exists x. \varphi$ ——— $\{f(x_1, ..., x_n) / x\} \varphi$	δ#
	$\neg(\forall x. \varphi)$ ——— $\{f(x_1, ..., x_n) / x\} \varphi$	δ~@
	$\forall x. \varphi$ ——— $\{t / x\} \varphi$	γ@
	$\neg(\exists x. \varphi)$ ——— $\{t / x\} \varphi$	γ~#

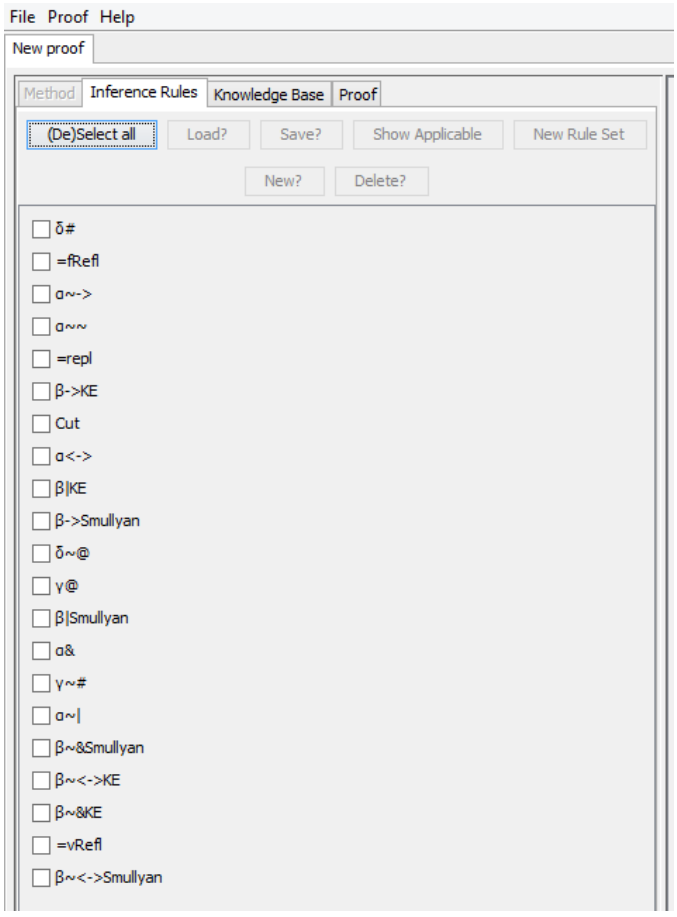
Lógica clássica (Forma de Primeira Ordem)	$\dfrac{}{x = x}$	=vRefl
	$\dfrac{}{f(x_1, ..., x_n) = f(x_1, ..., x_n)}$	=fRefl
	$\dfrac{t = u \quad \varphi(t)}{\varphi(u)}$	=repl
Lógica clássica (Forma Normal Conjuntiva)	$\dfrac{\kappa(x_1, ..., x_n)}{\kappa_1 \mid ... \mid \kappa_k}$	β
	$\dfrac{\kappa(x_1, ..., x_n)}{\sigma\kappa_1 \mid ... \mid \sigma\kappa_k \quad \sigma = \{t_1 / x_1, ..., t_n / x_n\}}$	γ
	$\dfrac{\{\varphi_1, ..., \varphi_k, \psi_{k+1}, ..., \psi_n\} \quad \psi_{k+1}^{c} \quad ... \quad \psi_n^{c}}{\varphi_1 \mid ... \mid \varphi_k}$	hyper
	$\dfrac{}{x = x}$	=CNFvRefl
	$\dfrac{}{f(x_1, ..., x_n) = f(x_1, ..., x_n)}$	=CNFfRefl
	$\dfrac{t = u \quad \kappa(x_1, ..., t, ..., x_n)}{\kappa(x_1, ..., u, ..., x_n)}$	=CNFrepl
Lógica de descrição ALC	$\dfrac{(C_1 \sqcap C_2)(x)}{C_1(x) \quad C_2(x)}$	&
	$\dfrac{(C_1 \sqcup C_2)(x)}{C_1(x) \mid C_2(x)}$	U

Lógica de descrição ALC	$\frac{(\exists r.C)(x)}{C(y)\ r(x,y)}$	#
	$\frac{(\forall r.C)(x)\ r(x,y)}{C(y)}$	@

Fonte: autoria própria. As fontes das regras são todas dadas no capítulo 3.

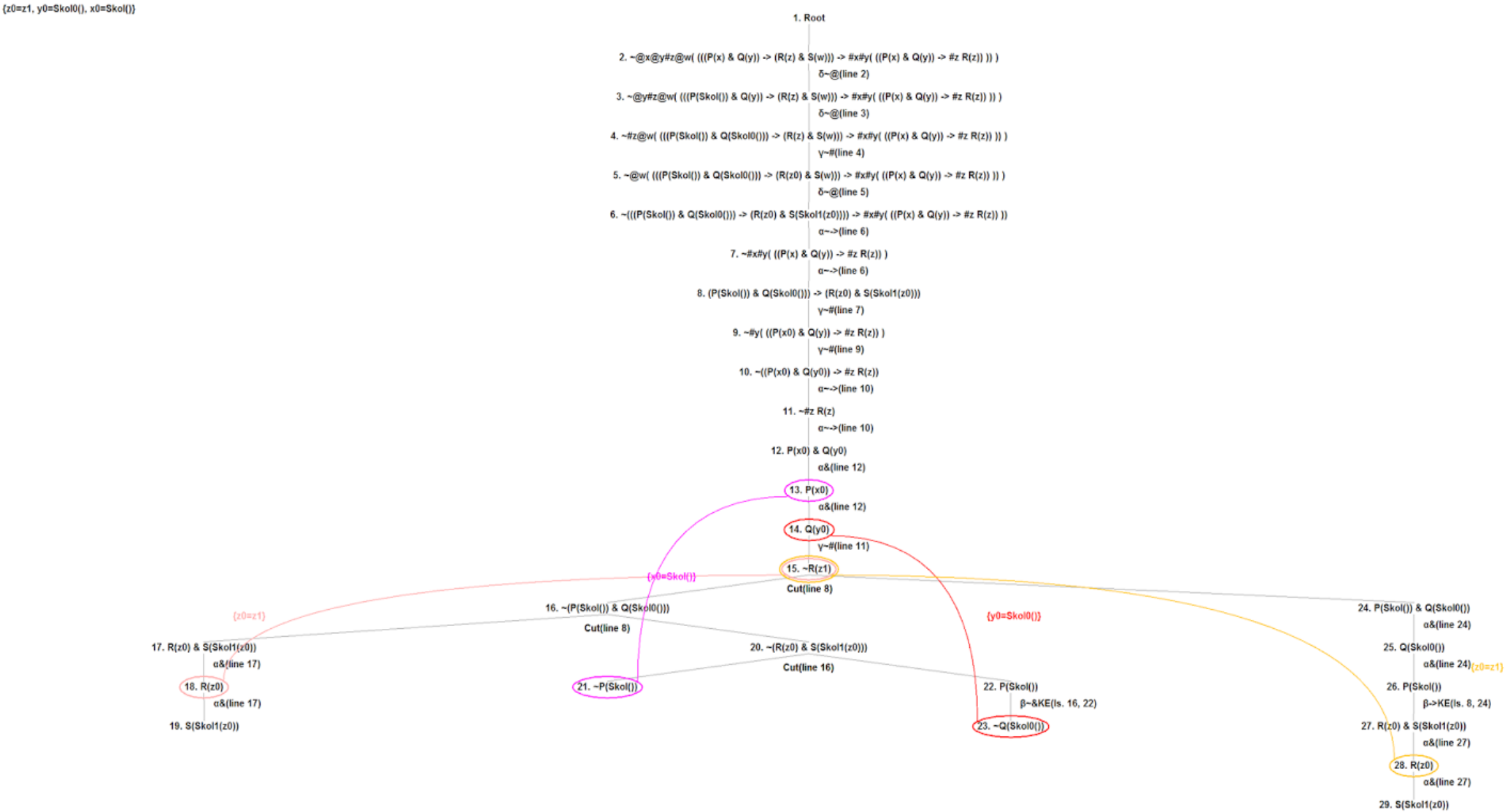
A GUI permite ao usuário além disso criar um método com regras arbitrárias.

Figura 3 — Ajuste do método.



Fonte: autoria própria

Finalmente vê-se na figura quatro (próxima página) um Tableau fechado. As fórmulas que unificam são apontadas. Além disso o subconjunto da substituição do Tableau que é necessário para cada unificação é exibido.


{z0=z1, y0=Skol0(), x0=Skol()}
1. Root
2. ~@x@y#z@w((((P(x) & Q(y)) -> (R(z) & S(w))) -> #x#y(((P(x) & Q(y)) -> #z R(z)))))
δ~@(line 2)
3. ~@y#z@w((((P(Skol()) & Q(y)) -> (R(z) & S(w))) -> #x#y(((P(x) & Q(y)) -> #z R(z)))))
δ~@(line 3)
4. ~#z@w((((P(Skol()) & Q(Skol0())) -> (R(z) & S(w))) -> #x#y(((P(x) & Q(y)) -> #z R(z)))))
γ~#(line 4)
5. ~@w((((P(Skol()) & Q(Skol0())) -> (R(z0) & S(w))) -> #x#y(((P(x) & Q(y)) -> #z R(z)))))
δ~@(line 5)
6. ~(((P(Skol()) & Q(Skol0())) -> (R(z0) & S(Skol1(z0)))) -> #x#y(((P(x) & Q(y)) -> #z R(z))))
α~->(line 6)
7. ~#x#y(((P(x) & Q(y)) -> #z R(z)))
α~->(line 6)
8. (P(Skol()) & Q(Skol0())) -> (R(z0) & S(Skol1(z0)))
γ~#(line 7)
9. ~#y(((P(x0) & Q(y)) -> #z R(z)))
γ~#(line 9)
10. ~((P(x0) & Q(y0)) -> #z R(z))
α~->(line 10)
11. ~#z R(z)
α~->(line 10)
12. P(x0) & Q(y0)
α&(line 12)
13. P(x0)
α&(line 12)
14. Q(y0)
γ~#(line 11)
15. ~R(z1)
Cut(line 8)
{x0=Skol()}
{z0=z1}
16. ~(P(Skol()) & Q(Skol0()))
Cut(line 8)
{y0=Skol0()}
24. P(Skol()) & Q(Skol0())
α&(line 24)
17. R(z0) & S(Skol1(z0))
α&(line 17)
20. ~(R(z0) & S(Skol1(z0)))
Cut(line 16)
25. Q(Skol0())
α&(line 24)
{z0=z1}
18. R(z0)
α&(line 17)
21. ~P(Skol())
22. P(Skol())
β~&KE(ls. 16, 22)
26. P(Skol())
β->KE(ls. 8, 24)
19. S(Skol1(z0))
23. ~Q(Skol0())
27. R(z0) & S(Skol1(z0))
α&(line 27)
28. R(z0)
α&(line 27)
29. S(Skol1(z0))

TECHNISCHE UNIVERSITÄT BERLIN
FAKULTÄT IV - ELEKTROTECHNIK UND INFORMATIK
FACHGEBIET AGENTENTECHNOLOGIEN IN BETRIEBLICHEN ANWENDUNGEN UND DER TELEKOMMUNIKATION (AOT)
BACHELORSTUDIENGANG INFORMATIK

DIOGO RAPHAEL CRAVO
MATR.-NR. 0354288

Weiterentwicklung eines Theorembeweisers für klassische Logik erster Stufe und Gestaltung eines analytischen Tableau-Reasoners für Beschreibungslogik

Zur Erlangung des akademischen Grades eines Bachelor of Science (B.Sc.)

Betreuer: Dr. Ing. Stefan Fricke
Gutachter: Prof. Dr. Dr. h.c Sahin Albayrak
Zweiter Gutachter: Prof. Dr. habil. Odej Kao

Berlin
2015

EIDESSTATTLICHE ERKLÄRUNG

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den

.....

Diogo Raphael Cravo

DANKSAGUNG

Hiermit möchte ich mich bei Herrn Dr. Ing. Fricke für die Betreuung und alle Besprechungen, bei allen meinen Lehrern für das beigebrachte Wissen, bei meiner Familie für die Unterstützung und bei der Person, die das Korrekturlesen dieser Arbeit vorgenommen hat, bedanken.

KURZFASSUNG

Während des Wintersemesters von 2013 auf 2014 fand die Veranstaltung “Projekt: symbolische künstliche Intelligenz” statt. Dabei haben die Studenten einen Theorembeweiser für die klassische Logik erster Stufe erstellt. Der damals angefertigte Beweiser konnte Formeln durch die Anwendung von sechs verschiedenen Ansätzen auf ihre Unerfüllbarkeit oder Allgemeingültigkeit sowie Wissensbasen auf ihre Konsistenz überprüfen. Diese Bachelorarbeit stellt die Weiterentwicklung dieses Theorembeweisers vor. Zu den neuen Funktionalitäten des Beweisers gehören verschiedene Tableau Ansätze, wie das analytische Tableau von Smullyan und das Tableau mit freien Variablen von Fitting sowie graphische Darstellungen der Beweise. Außerdem kann die Software auch das Subsumptionsproblem der ALC Beschreibungslogik lösen. Die implementierte Software ist leicht bedienbar und mehrere Bestandteile können wiederverwendet werden. Schließlich wurden alle in dieser Arbeit implementierten Ansätze durch 68 Tests ausgewertet, wodurch die schlechte Leistungsfähigkeit der Implementierung festgestellt wurde. Dazu werden Verbesserungen am Ende vorgeschlagen.

Schlagwörter: Tableau. Theorembeweiser. Klassische Logik. Beschreibungslogik. FOL.

Further development of a theorem prover for classical first-order logic and design of an analytic tableau reasoner for a description logic

ABSTRACT

During the Winter Semester from 2013 to 2014 the Project “Projekt: Symbolische Künstliche Intelligenz” took place. During that project the students have created a theorem prover for the first-order logic. The prover created in this time could check the unsatisfiability or validity of formulas as well as the consistency of knowledge bases by applying six different approaches. This Bachelor’s Thesis presents the further development of that theorem prover. The new prover offers new functionalities, such as the analytical Smullyan Tableau and the free Variable Tableau by Fitting, as well as graphical representations of proofs. Furthermore the Software can solve the subsumption problem of the ALC Description Logic. The implemented Software is easy to use and many of its modules are reusable. Finally all approaches implemented in this work were evaluated by the application of 68 Tests, which revealed the bad performance of the implementation. Improvements are suggested in the end.

Keywords: Tableau. Theorem prover. Classical logic. Description logics. FOL.

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

TUB	Technische Universität Berlin
UFRGS	Universidade Federal do Rio Grande do Sul
AOT	Agententechnologien in betrieblichen Anwendungen und der Telekommunikation
FOL	*First-Order Logic* / Logik erster Stufe
DL	*Description Logic* / Beschreibungslogik
KNF	Konjunktive Normalform
CNF	*Conjunctive Normal Form*
FOF	*First-Order Form*
gdw	genau dann, wenn
DFID	*Depth-First Iterative Deepening search* / Iterative Tiefensuche

INHALTSVERZEICHNIS

1 EINLEITUNG

Auch wenn ein Theorembeweiser zum jetzigen Zeitpunkt nicht alleine wichtige mathematische Beweise durchführen kann, sind Theorembeweiser sehr nützlich bei konkreten Einsätzen. Erfolg haben automatische Beweissysteme bei der Verifikation bestimmter Eigenschaften einer Software. Beispielsweise hat der Student im WS 13/14 die Vorlesung "Software Engineering eingebetteter Systeme" an der TUB belegt. Im Rahmen dieser Vorlesung haben sich die Studenten mit dem UPPAAL[1] Werkzeug befasst, das die Gestaltung von Echtzeit-Systemen anhand *Timed Automata* ermöglicht. Der Student hat an der Entwicklung eines einfachen Fahrstuhls mittels UPPAAL teilgenommen. Das Verhältnis zur Logik: Durch die Verwendung von temporaler Logik konnten die Studenten verschiedene Eigenschaften der implementierten Gestaltung beweisen.

Dem Beweiser von UPPAAL, genauer gesagt ein *Model Checker*, fehlt es aber an Interaktivität. Er braucht das nicht. Der in dieser Arbeit erklärte Beweiser ist hingegen sehr interaktiv, denn auch der Benutzer darf sowohl an der Einstellung des benutzten Ansatzes als auch an der Durchführung des Beweises teilnehmen.

Es gibt verschiedene Beweisverfahren. Wie der Titel bereits verrät, ist das Tableau Verfahren der Schwerpunkt dieser Arbeit. Das Tableau ist ein berühmtes Verfahren, welches die Erfüllbarkeit einer Menge Formeln der Logik erster Stufe beweisen kann. In dieser Arbeit wird ein interaktiver Theorembeweiser erweitert, indem man das Tableau Verfahren anwendet, um nicht nur das Erfüllbarkeitsproblem der Logik erster Stufe zu lösen, sondern auch das Subsumptionsproblem einer Beschreibungslogik. Zunächst wird eine kurze Geschichte des Tableaus wiedergegeben. Die meisten Informationen wurden aus D'Agostino *et al*. (1999) entnommen.

Harrison (2009) hält E. W. Beth und J. Hintikka wegen ihrer zeitgleichen im Jahr 1955 erschienenen Artikel für die Erfinder des auf die klassische Logik anwendbaren semantischen Tableau Verfahrens. Tableaus waren aber damals laut D'Agostino *et al*. (1999) nicht intuitiv. Das im Jahr 1968 von Smullyan veröffentlichte analytische Tableau war das erste leicht

[1] Verfügbar über www.uppaal.org

anwendbare Tableau Verfahren.

Inzwischen ist das Tableau Verfahren zu einem der am meisten benutzten Beweisverfahren geworden. Tableaus werden nach D'Agostino *et al*. (1999) auf die Modallogik, auf mehrwertige Logiken, auf intuitionistische Logik, auf Logiken höherer Stufe und auf viele andere Logiken angewandt. Selbstverständlich werden Tableaus auch auf verschiedene Arten von Beschreibungslogiken (BAADER *et al*. 2003), wie die ALC, angewandt.

Tableaus lassen sich durch mehrere Ansätze verfeinern. Die erste große Verfeinerung war die Anwendung der Unifikation. Danach kam laut D'Agostino *et al*. (1999) durch Fitting im Jahr 1986 eine frühe Anwendung der Skolemisierung, die den Algorithmen die Verwendung der δ-Regel (siehe Kapitel 3) erspart. Weitere Verfeinerungen erfolgen laut Hähnle (2001) in der Klauselform. Hähnle erwähnt das *Connection Tableau*, bei dem die Erzeugung neuer Zweige nach bestimmten Regeln erfolgt. Weitere Beschränkungen der Verzweigung erreichen die als Hypertableau und *Model Generation* bekannten Verfahren. Noch weitere Verfeinerungen werden von Hähnle (2001) beschrieben. In dieser Arbeit werden hinsichtlich der Logik erster Stufe das Smullyan Tableau, das KE Tableau und zwei Arten Tableau für die KNF berücksichtigt (siehe Kapitel 3). Diese Ansätze werden außerdem durch die Unifikation erweitert. Diese Arbeit erweitert ein Theorembeweiser, der im Winter Semester 2013 / 2014 geschrieben wurde.

1.1 Der im WS 13/14 geschriebene Theorembeweiser

Während des Wintersemesters von 2013 auf 2014 fand die Veranstaltung "Projekt: symbolische künstliche Intelligenz" an der TU Berlin statt. Dabei wurde ein Theorembeweiser für die klassische Logik erster Stufe erstellt. In einer Gruppe von elf Personen haben die Studenten, unter denen der Diplomand war, mithilfe der Betreuung von Dr. Stefan Fricke das Thema recherchiert, die Struktur der Implementation diskutiert, die geeignete Gestaltung in Code umgesetzt, diese dann durch Heuristiken verfeinert und schließlich das Ergebnis durch *Benchmarks* ausgewertet.

Das implementierte Programm ist ein funktionierender Beweiser, der durch

unterschiedliche Verfahren logische Aussagen anhand einer von den Benutzern eingegebenen Wissensbasis widersprechen oder prüfen kann. Die folgenden Verfahren und Heuristiken wurden implementiert:

- ★ Resolution, durch eine Stützmenge mit einer heuristischen Funktion zur Sortierung der Klauseln sowie Subsumption, Faktorisierung und Antwortliterale verfeinert;
- ★ Rückwärtsverkettung, durch eine *PriorityQueue* zum Vorzug der an Prämisse ärmere Regeln verfeinert;
- ★ Vorwärtsverkettung, die zum Rete-Algorithmus geführt hat;
- ★ und Rete, durch Reduktion der Anzahl an Substituionsvereinigungen verfeinert.

Außerdem wurden die vorhandenen Methoden kombiniert, wodurch das Vorwärts-Rückwärtsverkettungsverfahren und das Rete-Rückwärtsverkettungsverfahren entstanden sind. Die Software bietet nebenbei eine Menge von Grundfunktionalitäten: eine grafische Benutzerschnittstelle (GUI), eine Grammatik und einen Parser für Logik erster Stufe, Umwandlung von Formeln in Konjunktive Normal Form (KNF) sowie in Horn-Klauseln und Unifikation und Substitution von Formeln. Die Implementierung ist in Java geschehen. Die Software ist mittels einer Open-Source-Lizenz frei erweiterbar.

1.2 Ziele und Zwecke

Das Ziel der Abschlussarbeit soll sein, den im WS 13/14 geschriebene Theorembeweiser durch die Implementierung, Verfeinerung und Auswertung von mehreren Tableau-Verfahren für die Logik erster Stufe, sowie von einem Tableau-Verfahren für die ALC Beschreibungslogik zu erweitern und die dazu vorausgesetzten Veränderungen und Erweiterungen des Theorembeweisers, sowie die Implementierung von grafischen Beweisdarstellungen durchzuführen. Um dieses Ziel zu erreichen, soll auch eine ausführliche Recherche durchgeführt

werden.

Bei der Bearbeitung lernt der Autor, besser mit der Gestaltung und Implementierung von einer Software umzugehen. Ein besseres Vertrauen mit der Logik ist auch in Sicht. Ferner, obwohl die implementierten Ansätze nicht leistungsfähig sind, kann die angefertigte Software vermutlich in einem Einführungskurs in der Logik verwendet werden.

1.3 Gliederung

Diese Arbeit wird folgendermaßen gegliedert: Zunächst wird im Kapitel 2 eine formelle Definition der klassischen Logik gegeben sowie eine kurze Definition der ALC Beschreibungslogik. Kapitel 2 erklärt außerdem die Notation, die in dieser Arbeit verwendet wird. Kapitel 3 stellt die Idee von einem Tableau Verfahren vor, beschreibt mehrere Ansätze und vergleicht diese Ansätze untereinander. Kapitel 4 erklärt die implementierte Software, einschließlich Klassenstruktur und Algorithmen, und begründet die getroffenen Entscheidungen. Anschließend wertet Kapitel 5 die Software aus, indem ausführliche Tests durchgeführt und die zusammengefassten Ergebnisse solcher Tests präsentiert werden. Letztlich gibt Kapitel 6 einen Ausblick über das, was in dieser Arbeit nicht berücksichtigt wird. Anhang A stellt die Ergebnisse aller Tests vor und Anhang B erklärt die Bedienung der Software.

2 DIE LOGIK

In diesem Abschnitt wird die Logik erklärt. Das heißt, dass alle nötigen Informationen über die klassische und die Beschreibungslogiken hier dargestellt werden sowie die entsprechende Literatur, die jedes Thema genauer betrachtet. Wenn der Leser schon mit der klassischen Logik und mit der ALC Beschreibungslogik vertraut ist, kann dieses Kapitel übersprungen werden. Dieses Kapitel dient in diesem Fall ausschließlich der Vorstellung der in dieser Arbeit verwendeten Notation.

Wir beginnen mit der klassischen Logik. **Definitionen dieser Logik sind weit verbreitet und bekannt, vgl. bspw. Russel und Norvig (2012), Hähnle (2001) und der erste Abschnitt im Artikel von Letz (D'AGOSTINO *et al*., 1999).** Das sind die Quellen für dieses Kapitel. Anhand dieser Definitionen gibt der Autor zunächst die Definition der Logik erster Stufe wieder, wobei nur Begriffe und Aussagen hervorgehoben werden, die wichtig für diese Arbeit sind.

2.1 Die klassische Logik

Eine Logik ist eine Sprache. Das heißt, dass wir anhand einer bestimmten Syntax und einer bestimmten Semantik Sätze bilden können. Die Logik erlaubt uns, logische Werte diesen Sätze zuzuordnen: die Wahrheitswerte. Durch eine solche Interpretation können Regeln erzeugt werden, die immer wahr oder immer falsch sind. Alle diese Ideen sind sehr wichtig für die Verfahren, die in dieser Arbeit erzielt sind. Diese Definition der Logik mangelt aber an Genauigkeit. Infolgedessen widmen wir uns jetzt der Formalisierung dieser Definition.

2.1.1 Die Logik erster Stufe: Syntax

Eine Syntax stellt fest, welche Bausteine eine Sprache hat und wie man diese Bausteine verbinden darf. In dem Folgenden wird eine Definition von dem Allgemeinsten zum spezifischsten Konzept gegeben.

DEFINITION 2.1.1. Das Alphabet. Die Symbole, die die Formeln in der Logik bilden, sind folgende:

- ★ Die Menge aller Variablen: **V**.
- ★ Die Menge aller Funktionen: **F**. Jedes Element f von **F** wird eine Anzahl an Parametern zugeordnet.
- ★ Die Menge aller Prädikate: **P**. Jedes Element p von **P** wird eine Anzahl an Parametern zugeordnet.
- ★ Die Junktoren: $\wedge$ (Konjunktion), $\vee$ (Disjunktion), $\rightarrow$ (materielle Implikation), $\leftrightarrow$ (Bikonditional)
- ★ Die Quantoren: $\forall$[2] (Allquantor), $\exists$[3] (Existenzquantor)
- ★ Die Negation ($\neg$[4]), die Gleichheit (=) und weitere Symbole: (,)

DEFINITION 2.1.2. Formel. Die Logik erster Stufe behandelt Formeln. Die Menge aller gültigen Formeln **MF** wird wie folgend definiert: (i) ist φ ein Literal, so $\varphi \in$ **MF**; (ii) wenn φ_1, $\varphi_2 \in$ **MF** dann φ_1 OP $\varphi_2 \in$ **MF**, wobei OP $\in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$; (iii) wenn $\varphi \in$ **MF** und x $\in$ **V** dann $\forall$x.φ $\in$ **MF** und $\exists$x.φ $\in$ **MF**.

DEFINITION 2.1.3. Literal. Ein Literal ist entweder ein Atom (φ) oder ein durch die Anwendung der Negation negiertes Atom ($\neg\varphi$). Ein Atom wird manchmal positives Literal genannt, während ein negiertes Atom negatives Literal genannt wird.

DEFINITION 2.1.4. Atom. Ein Atom ist entweder ein Prädikat oder eine Gleichung.

---

[2] In der Software wird der Allquantor durch ein @ dargestellt.
[3] In der Software wird der Existenzquantor durch ein # dargestellt.
[4] In der Software wird die Negation durch ein ~ dargestellt.

DEFINITION 2.1.5. Prädikat. Ein Prädikat hat einen Namen, der üblicherweise groß geschrieben wird, und eine beliebige Anzahl (≥ 1) von Parametern, die zwischen Klammern dargestellt werden, e.g. Prädikat(p_1, p_2). In der Logik erster Stufe dürfen ausschließlich Terme in den Parametern eines Prädikates auftauchen. Alle Prädikate gehören zu **P**.

DEFINITION 2.1.6. Term. Terme sind alle Konstanten, Variablen und Funktionen.

DEFINITION 2.1.6.1. Konstant. Alle und ausschließlich die Elemente $K \in \mathbf{F}$ mit 0 Parametern werden Konstant genannt.

DEFINITION 2.1.6.2. Variable. Alle und ausschließlich die Elemente $v \in \mathbf{V}$ werden Variable genannt.

DEFINITION 2.1.6.3. Funktion. Alle und ausschließlich die Elemente $f \in \mathbf{F}$ mit mindestens 1 Parameter werden Funktion genannt. In der Logik erster Stufe dürfen ausschließlich Terme in den Parametern einer Funktion vorkommen.

DEFINITION 2.1.7. Gleichung. Eine Gleichung ist die Verbindung zwischen zwei Terme durch das Symbol der Gleichheit, e.g. v=f(x).

DEFINITION 2.1.8. Gebundene und freie Variablen. Eine gebundene Variable ist eine Variable, die entweder einem Allquantor oder einem Existenzquantor folgt. Jede gebundene Variable wird direkt hinter dem dazugehörenden Quantor geschrieben. Gibt es keinen Quantor oder falls eine Variable nicht direkt hinter einem Quantor geschrieben wird, dann wird diese Variable als freie Variable bezeichnet. Das ist aber eine lockere Definition, die den Gültigkeitsbereich nicht richtig erklärt. Fitting (1990, S. 99-100) definiert die Menge aller freien Variablen **FV** folgendermaßen: (i) ist φ eine atomische Formel, so $\mathbf{FV}(\neg\varphi) = \mathbf{FV}(\varphi) = \{v \mid v \in \mathbf{V}$ und v kommt in φ vor$\}$; (ii) wenn $\varphi, \psi \in \mathbf{MF}$, dann $\mathbf{FV}(\varphi \text{ OP } \psi) = \mathbf{FV}(\varphi) \cup \mathbf{FV}(\psi)$, wobei OP $\in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$; (iii) wenn $\varphi \in \mathbf{MF}$ und $x \in \mathbf{V}$, dann $\mathbf{FV}(\forall x.\varphi) = \mathbf{FV}(\exists x.\varphi) = \mathbf{FV}(\varphi) \setminus \{x\}$. Eine Variable ist entweder gebunden oder frei, aber nicht beides.

DEFINITION 2.1.9. Komplement. Seien φ, ψ, $\alpha \in \mathbf{MF}$, wobei $\psi = \neg\alpha$ und φ eine nicht negierte Formel ist, dann ist das Komplement von φ (dargestellt φ^c) die Formel $\neg\varphi$ und das Komplement von ψ (dargestellt ψ^c) die Formel α.

Damit ist die Syntax der Logik erster Stufe erklärt.

2.1.2 Die Logik erster Stufe: Semantik

Die Syntax definiert alle gültigen Formeln, aber die Sätze der Logik haben ohne Semantik keinen Bezug auf die Wirklichkeit, keinen Sinn. Die Semantik ordnet jedem syntaktischen Element eine Interpretation zu. Es gibt einen Grundbereich U, die alle Interpretationen enthält, und eine Funktion, die für die Zuordnung sorgt. **Die Quellen für diese Definitionen sind wieder Hähnle (2001) und Letz (D'AGOSTINO *et al*., 1999)**.

DEFINITION 2.2.1. Struktur. Eine Struktur ist ein Paar (U,I), wobei U eine Menge ist und I eine Interpretation. Die Menge U enthält alle benutzbaren Elemente. Die Funktion I wird wie folgend definiert: (i) Wenn $f \in \mathbf{F}$ dann $I(f):U^n \rightarrow U$; (ii) Wenn $P \in \mathbf{P}$ dann $I(P) \subseteq U^n$.

DEFINITION 2.2.2. Belegung. Eine Belegung ist eine Funktion $B:\mathbf{V} \rightarrow U$. Darüber hinaus wird B(v) Belegung der Variable v genannt. Hier wird hervorgehoben, dass die Belegung unabhängig von der Struktur ist.

DEFINITION 2.2.3. Wahrheit. In dem Folgenden wird die Notation von Hähnle (2001) verwendet. Die klassische Logik erster Stufe hat zwei Wahrheitswerte: $\top$ (wahr) und $\bot$ (falsch). Jeder Formel F wird anhand einer Struktur S und Belegung B genau einen Wert zugeordnet. Ist die Formel φ wahr, so schreibt man $(S,B) \vDash \varphi$. Ist sie hingegen falsch, so wird *nicht* $(S,B) \vDash \varphi$

oder einfach $(S,B) \nvDash \varphi$ geschrieben. Es ist durchaus möglich, aber nicht erforderlich, dass gegeben zwei beliebige Strukturen $S_1 \neq S_2$ $(S_1,B) \vDash \varphi$ gilt, aber $(S_2,B) \nvDash \varphi$. Dasselbe gilt für die Belegung, also $B_1 \neq B_2$, $(S,B_1) \vDash \varphi$, aber $(S,B_2) \nvDash \varphi$. **Die Quelle für diese Definition ist Hähnle (2001).**

- ★ Wenn $P \in \mathbf{P}$ dann $(S,B) \vDash P$ gdw. $P' \in I(P)$, wobei sich P' daraus ergibt, dass alle (freien) Variablen in P durch ihre Belegungen in B mit Berücksichtigung des Gültigkeitsbereiches jeder Variablen ersetzt[5] werden.
- ★ Wenn $\alpha \in \mathbf{MF}$ dann $(S,B) \vDash \neg\alpha$ gdw. $(S,B) \nvDash \alpha$
- ★ Wenn $\alpha, \beta \in \mathbf{MF}$ dann $(S,B) \vDash \alpha \wedge \beta$ gdw. $(S,B) \vDash \alpha$ und $(S,B) \vDash \beta$
- ★ Wenn $\alpha, \beta \in \mathbf{MF}$ dann $(S,B) \vDash \alpha \vee \beta$ gdw. $(S,B) \vDash \alpha$ oder $(S,B) \vDash \beta$ oder $(S,B) \vDash \alpha \wedge \beta$
- ★ Wenn $\alpha, \beta \in \mathbf{MF}$ dann $(S,B) \vDash \alpha \rightarrow \beta$ gdw. $(S,B) \vDash \neg\alpha$ oder $(S,B) \vDash \beta$
- ★ Wenn $\alpha, \beta \in \mathbf{MF}$ dann $(S,B) \vDash \alpha \leftrightarrow \beta$ gdw. $(S,B) \vDash \alpha \wedge \beta$ oder $(S,B) \vDash \neg\alpha \wedge \neg\beta$
- ★ Wenn $\alpha \in \mathbf{MF}$, $v \in \mathbf{V}$ dann $(S,B) \vDash \forall v.\alpha$ gdw. $(S,B) \vDash \alpha_1'$, ... , $(S,B) \vDash \alpha_n'$, wobei sich α_i' daraus ergibt, dass v durch das Element $k_i \in U$ ersetzt wird.
- ★ Wenn $\alpha \in \mathbf{MF}$, $v \in \mathbf{V}$ dann $(S,B) \vDash \exists v.\alpha$ gdw. es einen Element $k \in U$ gibt, sodass $(S,B) \vDash \alpha'$, wobei sich α' daraus ergibt, dass v durch k ersetzt wird.

DEFINITION 2.2.4. Modell. Jedes Paar M = (U,I) wird ein Modell für $\{\varphi_1,...,\varphi_n\}$ genannt, solange U eine Menge ist, I eine Interpretation ist, φ_1, ..., $\varphi_n \in \mathbf{MF}$ und gegeben irgendeiner Belegung B $(M, B) \vDash \varphi_1$, ..., $(M, B) \vDash \varphi_n$ gelten.

DEFINITION 2.2.5. Erfüllbarkeit und Unerfüllbarkeit. Gibt es ein Modell für $W = \{\varphi_1, ..., \varphi_n\}$, so wird W erfüllbar genannt. Gibt es kein Modell, so wird W unerfüllbar genannt.

[5] Das ist eigentlich eine Vereinfachung. In der Literatur wird hier den Wert eines Termes definiert (siehe bspw. FITTING 1990, S. 104-5).

DEFINITION 2.2.6. Allgemeingültigkeit. Sind alle Paare (U,I) Modelle für $W = \{\varphi_1, ..., \varphi_n\}$, so wird W allgemeingültig genannt.

DEFINITION 2.2.7. Logische Konsequenz. Nach Fitting (1990, S. 119) ist $\psi \in \mathbf{MF}$ eine logische Konsequenz von $\{\varphi_1,...,\varphi_n\}$ mit $\varphi_1,...,\varphi_n \in \mathbf{MF}$ (dargestellt $\{\varphi_1,...,\varphi_n\} \vDash \psi$) gdw. für jedes Modell M, sodass M ein Modell für jede Formel $\varphi_1,...,\varphi_n$ ist, M auch ein Modell für ψ ist.

2.1.3 Die Logik erster Stufe: Unifikation

DEFINITION 2.3.1. Substitution. Eine Substitution $\sigma:\mathbf{V}\rightarrow\mathbf{F} \cup \mathbf{V}$ ist eine Funktion, die einige Variablen einer Formel durch andere Terme ersetzt. Ein Beispiel ist {A / x, f(x) / y}, eine Substitution, die x durch A und y durch f(x) ersetzt.

DEFINITION 2.3.2. Unifikator. Ein Unifikator σ ist eine Substitution, sodass $\sigma(F_1)=\sigma(F_2)$, wobei F_1 und F_2 Formeln sind und '=' bedeutet, dass beide Operanden syntaktisch identisch sind.

DEFINITION 2.3.2.1. Allgemeinster Unifikator. Ein allgemeinster Unifikator σ^A für zwei Formeln F_1 und F_2 ist ein Unifikator, sodass gegeben einen anderen Unifikator σ' mit $\sigma'(F_1)=\sigma'(F_2)$, es gibt immer eine Substitution s, sodass $s(\sigma^A(F_1))=\sigma'(F_2)$.

Ein guter Algorithmus für die Berechnung des allgemeinsten Unifikators kann in Russel und Norvig (2012) gefunden werden.

2.1.4 Die Logik erster Stufe: Notation von Smullyan

Wir benötigen noch eine Idee: die Notation von Smullyan. Smullyan stellt in seinem Buch (SMULLYAN, 1968) eine Art analytischen Tableau für die klassische Logik erster Stufe

vor. Darüber hinaus präsentiert er auch eine Darstellungsart von Formeln, die sehr nützlich bei der Darstellung von Schlussregeln ist. Im Folgenden[6] (SMULLYAN, 1968) werden α-, β-, δ- und γ-Formeln dargestellt, wobei A eine Konstant ist und $\varphi, \psi \in \mathbf{MF}$:

Abbildung 2.1 — Notation von Smullyan

α	α_1	α_2
$\varphi \wedge \psi$	φ	ψ
$\neg(\varphi \vee \psi)$	$\neg\varphi$	$\neg\psi$
$\neg(\varphi \rightarrow \psi)$	φ	$\neg\psi$
$\neg\neg\varphi$	φ	φ
$\varphi \leftrightarrow \psi$	$\varphi \rightarrow \psi$	$\psi \rightarrow \varphi$

β	β_1	β_2
$\neg(\varphi \wedge \psi)$	$\neg\varphi$	$\neg\psi$
$\varphi \vee \psi$	φ	ψ
$\varphi \rightarrow \psi$	$\neg\varphi$	ψ
$\neg(\varphi \leftrightarrow \psi)$	$\neg(\varphi \rightarrow \psi)$	$\neg(\psi \rightarrow \varphi)$

δ	$\delta(A)$
$\exists x.\varphi$	$\{A / x\}(\varphi)$
$\neg\forall x.\varphi$	$\{A / x\}(\neg\varphi)$

γ	$\gamma(A)$
$\forall x.\varphi$	$\{A / x\}(\varphi)$
$\neg\exists x.\varphi$	$\{A / x\}(\neg\varphi)$

Quelle: Smullyan (1968)

2.1.5 Die Logik erster Stufe: konjunktive Normalform

Die KNF (konjunktive Normalform) ist eine besondere Form von logischen Aussagen, die eine Untermenge von **MF** umfasst. Es wurde bewiesen, dass jede Formel $\varphi \in \mathbf{MF}$ in die

[6] Nur die Zeilen mit dem Bikonditional wurden vom Autor hinzugefügt. Alle anderen sind genauso dargestellt wie im Buch von Smullyan.

KNF umgewandelt werden kann. Um den Algorithmus zu verstehen, brauchen wir zuerst die Definition von der Skolem Funktion.

DEFINITION 2.5.1. Skolem Funktion. Eine Funktion f, die einen neuen Namen hat, also $f \notin \mathbf{F}$. Eine Skolem Funktion f ersetzt die gebundene Variable einer existentiellen Quantifizierung δ und die Parameter von f sind alle mit einer universellen Quantifizierung gebundenen Variablen, die die Vorfahren von δ im syntaktischen Baum sind.

Ein guter Algorithmus für die Berechnung der KNF kann in Russel und Norvig (2012) gefunden werden. Am Ende erhalten wir eine sogenannte Klauselmenge. Jede Formel in der KNF wird Klausel genannt.

DEFINITION 2.5.2. Klausel. Das Ergebnis von der Umwandlung in die KNF ist eine Menge von Klauseln. Jede Klausel ist eine Formel, die nur Literale enthält. Die Literalen einer Klausel dürfen nur durch die Disjunktion verbunden werden. Alle Variablen in einer Klausel sind universell quantifiziert. Klausel werden hier wie Mengen von Formeln dargestellt (bspw. {P(h(A,y)), Q(f(x))}). In dieser Arbeit werden manchmal Klauseln abgekürzt mit $\kappa(x_1, ..., x_n)$, wobei $x_1, ..., x_n$ alle Variablen in der Klausel sind (z. B. $\kappa(x,y)$ = {P(h(A,y)), Q(f(x))}). Die k-te[7] Literale in κ wird dann als κ_k bezeichnet (z. B. κ_1 = P(h(A,y)), κ_2 = Q(f(x))).

2.2 Die Beschreibungslogiken

Unter Beschreibungslogiken (abgekürzt DL, aus *Description Logics*) versteht man eine Menge von Logiken, die grundsätzlich drei Elemente besitzen: Individuen, Klassen und Rollen. Individuen sind semantisch betrachtet wie Konstanten in der Logik erster Stufe, Klassen sind Mengen von Individuen und Rollen bauen Relationen zwischen Klassen auf, i. e., sind Mengen

[7] Bei dieser Notation ist die Reihenfolge nicht wirklich wichtig.

von Paaren von Individuen (BAADER *et al*. 2003). DLs entsprechen einer *open world assumption* (Baader *et al*. 2003, S. 72), das heißt, ein Modell in einer DL wird nie die ganze Welt beschreiben. Es wird stattdessen angenommen, es gibt noch weitere Individuen, Klassen, Rollen.

Formeln einer Beschreibungslogik werden zwei Kasten zugeordnet, um eine Wissensbasis zu schaffen: die TBox (*Terminological Box*), die die Welt durch Klassen sowie Rollen und ihre Zusammenhänge beschreibt, und die ABox (*Assertional Box*), die die Individuen und ihre passende Klassen und Rollen anhand der TBox beschreibt (BAADER *et al*. 2003).

Im Gegensatz zu der Logik erster Stufe, bei der es immer auf die Erfüllbarkeit der Wissensbasis und ein Ziel überprüft wurde, gibt es bei den Beschreibungslogiken eine Menge von Inferenzarten. Russel und Norvig (2012) nennen die Subsumption (von Klassen) und Klassifizierung (von Individuen zu Klassen) als die Wichtigste.

Insgesamt gibt es bei den Beschreibungslogiken 8 Inferenzaufgaben: Erfüllbarkeit (oder Konsistenz), Subsumption, Äquivalenz und Disjunktheit hinsichtlich der TBoxen und Konsistenzüberprüfung, Instanzprüfung, Abfrage und Klassifizierung hinsichtlich der ABoxen. Denn Subsumption, Disjunktheit und Äquivalenz im Grunde genommen in der Aufgabe der Überprüfung auf (Un)Erfüllbarkeit bestehen (BAADER *et al*. 2003, S. 67), muss ein *Reasoner* für diese Sprachen im Hinblick auf die TBoxen nur die Aufgabe der Überprüfung auf Erfüllbarkeit erfüllen. Eine präzisere Beschreibung der Inferenzaufgaben der ABox ist in Baader *et al*. (2003, S. 70-72).

2.2.1 Die ALC Beschreibungslogik

Die drei Bausteine der Beschreibungslogiken (Individuen, Klassen und Rollen) werden in der ALC (***A**ttributive **C**oncept Description **L**anguage with Complements*) anhand der folgenden syntaktischen Elemente kombiniert (BAADER *et al*. 2003): A (ein atomisches Konzept), $\top$ (die Wahrheit), $\bot$ (der Widerspruch), $\neg$A (die Negation eines atomischen Konzeptes), $\neg$C (das Komplement), C $\sqcap$ D (der Schnittpunkt zwei Konzepten), $\forall$R.C (die allgemeine Quantifizierung), $\exists$ R.$\top$ (die beschränkte existenzielle Quantifizierung).

Man bemerkt, dass die Vereinigung ($C \sqcup D$) von Konzepten sowie die allgemeine existenzielle Quantifizierung ($\exists R.C$) nicht in der ALC vorkommen. Diese können trotzdem durch die Kombination vom Komplement und von der Konjunktion (bzw. vom Komplement und von der allgemeinen Quantifizierung) ausgedrückt werden, was wiederum die Benutzung dieser syntaktischen Elemente erlaubt (BAADER *et al*. 2003, S. 53-54). Eine präzisere Beschreibung der Syntax und Semantik der Familie der AL Sprachen kann in Baader *et al*. (2003, S. 51-52) gefunden werden.

2.3 Zusammenfassung

Dieses Kapitel hat die Logik erster Stufe und die ALC Beschreibungslogik erklärt. Jetzt sollten sowohl Begriffe und Algorithmen wie Erfüllbarkeit, Unifikation und KNF, als auch die in dieser Arbeit verwendete Notation klar sein. Insbesondere wurde die Notation von Smullyan vorgestellt, die in dem nächsten Kapitel sehr nützlich sein wird. Kapitel 3 erläutert die verschiedenen Tableau-Verfahren.

3 DIE TABLEAU VERFAHREN

Dieser Abschnitt erklärt i. Allg., was ein Tableau Algorithmus ist, denn es werden verschiedene Algorithmen in dieser Arbeit vorgestellt. Die genauen Definitionen und Bemerkungen zu jedem Algorithmus werden in den nachfolgenden Abschnitten gegeben.

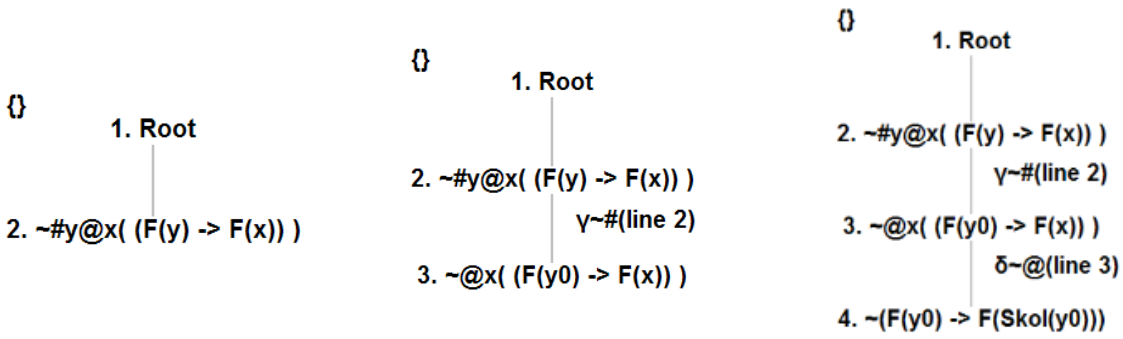
Bei dem analytischen Tableau Verfahren handelt es sich um einen Algorithmus, der einen Beweisbaum (das Tableau) aufbaut, um eine Menge von Formeln auf ihre Erfüllbarkeit zu untersuchen. Es wird behauptet, Tableaus seien eine Variante von Sequenzkalkül (D'AGOSTINO *et al*. 1999, S. 22).

Tableaus benutzen eine Art *reductio ad absurdum* Beweistechnik, d. h., sie beweisen etwas durch das Beweisen eines Widerspruches. Genauer gesagt, anhand einer Formelmenge W (die Wissensbasis) und einer Formel Z (das Ziel) versucht das Tableau die Unerfüllbarkeit von $W \wedge \neg Z$ zu beweisen. Dadurch wird die Allgemeingültigkeit von $W \rightarrow Z$ (auch $\neg W \vee Z$) bewiesen, also $W \models Z$. Mit anderen Worten erzielt das Tableau einen Beweis, dass es kein Modell M und Belegung B gibt, sodass $(M,B) \models W$ ohne dass $(M,B) \models Z$.

Der Baum beginnt als eine einfache Liste von Kanten, das heißt, am Anfang hat der Baum genau einen Zweig. Diese Kanten sind jeweils eine Formel der zu überprüfenden Formelmenge, welche sowohl Formeln der Wissensbasis als auch das negierte Ziel enthält. Ein Beispiel für ein solches Tableau ist das erste Tableau auf der nächsten Abbildung. Der Baum wird danach durch die Anwendung auf seine Kante von bestimmten Regeln schrittweise aufgebaut. Die genaue Menge der anwendbaren Regeln hängt mit der Art von Tableau zusammen (siehe nächste Abschnitte). Folgende Abbildung zeigt ein offenes Tableau für das Problem #18[8] aus Pelletier (1986) und das Ergebnis der Anwendung von zwei Regeln auf dieses Tableau:

[8] Das Problem lautet $\models_{Tableau} \exists y \forall x(F(y) \rightarrow F(x))$. Bei diesem Problem hat das Tableau keine Wissensbasis, nur das negierte Ziel.

Abbildung 3.1 — Beispiel von einem Tableau.



Quelle: eigene Darstellung.

In der Literatur (HÄHNLE, 2001) werden die Regeln üblicherweise in 4 Klassen eingeordnet: α, β, δ und γ, genauso wie die 1963 von Smullyan vorgestellte Einordnung von Formeln (vgl. Abschnitt 2.1.4). Die α-Regeln erzeugen keine neuen Zweige, sondern fügen nur neue Kanten zu einem vorhandenen Zweig hinzu und werden aus konjunktiven Formeln gebildet. Die β-Regeln können mehrere Zweige erzeugen und werden aus disjunktiven Formeln gebildet. Die δ- und γ-Regeln werden aus existenziellen bzw. aus universellen Quantifizierungen abgeleitet. Die δ-Regeln setzen die Erzeugung einer skolem Funktion voraus, während die γ-Regeln die Belegung einer Variable voraussetzen. Da es verschiedene Belegungen für eine Variable gibt, kann normalerweise eine γ-Regel unendliche Male angewendet werden. Der Algorithmus hat zwei Haltebedingungen:

I. **Abgeschlossenes Tableau**: Findet der Algorithmus in einem Zweig des Baums eine Formel und ihr Komplement, so ist der Zweig abgeschlossen und auf abgeschlossene Zweige dürfen keine Regeln mehr angewandt werden. Wenn alle Zweige des Baums abgeschlossen sind, dann hält der Algorithmus und die Unerfüllbarkeit von $W \wedge \neg Z$ ist bewiesen.

II. **Mangelnde Schlussregeln**: Ist es nicht mehr möglich, eine Regel anzuwenden, so hält der Algorithmus. Falls alle Zweige abgeschlossen sind, dann ist die Unerfüllbarkeit von

W ∧ ¬Z bewiesen, solange der angewandte Tableau Algorithmus zuverlässig und vollständig ist. Dieser Zustand ohne anwendbare Regeln kann nur dann erreicht werden, wenn die γ-Regeln vernünftig oder überhaupt nicht benutzt werden.

Die Zuverlässigkeit und die Vollständigkeit eines Tableau Verfahrens sind sehr wichtig. Zuverlässigkeit heißt, dass immer wenn es ein abgeschlossenes Tableau gibt, das Ziel eine logische Konsequenz der Wissensbasis ist. Auch wichtig ist die Vollständigkeit. Bei einem vollständigen Algorithmus gilt die Tatsache, dass immer wenn das Ziel eine logische Konsequenz der Wissensbasis ist, es ein dazu passendes abgeschlossenes Tableau gibt. Die Tableau Algorithmen sind i. A. zuverlässig. Die Vollständigkeit des jeweiligen Algorithmus hängt aber von der Menge der anwendbaren Regeln ab. Manche Ansätze benutzen wenige Regeln, um schneller ein abgeschlossenes Tableau zu finden. Die Einschränkung der Inferenzregeln bewirkt, dass diese Ansätze unvollständig sind (HÄHNLE, 2001).

Da die Logik erster Stufe **nicht entscheidbar** ist, gibt es jedoch keinen Tableau-Algorithmus, der immer nach einer begrenzten Anzahl von Schritten terminiert[9]. Wie vorher erwähnt, lässt sich dieses Problem bspw. durch die unendliche Anwendung einer γ-Regel erklären. Damit ein abgeschlossenes Tableau in akzeptabler Zeit gefunden wird, muss eine gute Strategie entwickelt werden, die nach diesem Tableau sucht. Hähnle (2001) setzt sich mit diesem Problem auseinander und schlägt vor, Zweige nur dann zu schließen, wenn der ganze Baum abgeschlossen werden kann. Das benötigt einige Änderungen an den Schlussregeln, die bald genauer erklärt werden. In den nächsten Abschnitten werden einige Arten von Tableaus vorgestellt.

3.1 Die Tableau Verfahren für die Logik erster Stufe

Das erste hier vorgestellte Verfahren ist das Tableau von Smullyan. Aufgrund seiner γ-Regel ist dieses Verfahren nicht leistungsfähig. Das KE Tableau verändert die Regeln des

[9] Manchmal gilt zwar (S,B) ⊨ φ für irgendeine φ genau dann, wenn U in der Struktur S = (U,I) unendlich ist. Smullyan (1968, S. 63) erwähnt diesen Fall und warnt, dass das Tableau Verfahren bei diesem Fall nicht hält.

Smullyan Tableaus, indem ein Tableau nur durch die Anwendung der Schnittregel verzweigt. Danach werden Varianten der γ-Regel mit freien Variablen und schließlich das Klausel Tableau vorgestellt.

3.1.1 Smullyan Tableau

Das Tableau von Smullyan (auch als systematisches oder analytisches Tableau bekannt) benutzt die folgenden Regeln (SMULLYAN, 1968):

Abbildung 3.2 — Das Tableau von Smullyan.

$\dfrac{\alpha}{\alpha_1 \\ \alpha_2}$	$\dfrac{\beta}{\beta_1 \mid \beta_2}$
$\dfrac{\delta}{\delta(A)}$	$\dfrac{\gamma}{\gamma(A)}$
wobei A ein neuer Parameter[10] ist	wobei A ein Term ist

Quelle: Smullyan (1968)

Bei dem Smullyan Tableau ist wichtig zu bemerken, dass keine Unifikation stattfindet. Stattdessen werden die Variablen bei der Anwendung einer γ-Regel durch einen Term ersetzt. Das Smullyan Tableau ist vollständig (SMULLYAN 1968, S. 57-9).

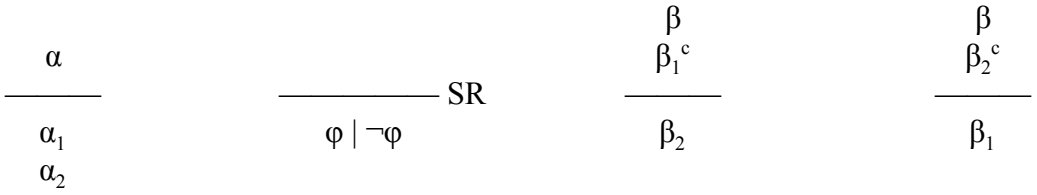
3.1.2 KE Tableau

Das KE Tableau (D'AGOSTINO und MONDADORI, 1994) sieht sehr ähnlich aus wie das Smullyan. Der wichtigste Unterschied zwischen den Beiden ist, dass bei dem KE ausschließlich die Schnittregel (hier abgekürzt SR) für Verzweigung sorgt. Die Autoren

[10] Smullyan erwähnt Parametern in seiner Definition von Logik erster Stufe. Parameter bedeutet in seiner Definition so viel wie Konstant. Diese Tatsache wird von BECKERT *et al.* (1993) hervorgehoben. Also ist der Begriff Parameter hier nicht mit den Parametern einer Funktion oder eines Prädikats zu verwechseln.

behaupten, dass das KE Tableau deswegen kürzere Beweise erstellen kann. Die δ- und γ-Regeln des Smullyan Tableaus können bei dem KE Tableau auch angewendet werden (und werden daher verborgen). Das KE Tableau[11] benutzt folgende Regeln:

Abbildung 3.3 — Das KE Tableau.



Quelle: D'Agostino und Mondadori (1994)

Außerdem empfehlen die Autoren in ihrem Artikel (D'AGOSTINO und MONDADORI, 1994), dass die Schnittregel nur anhand β-Formeln angewandt wird, wie auf der nächsten Abbildung zu sehen ist.

Abbildung 3.4 — Die $SR_{\beta 1}$ und $SR_{\beta 2}$ Schlussregeln.

Quelle: D'Agostino und Mondadori (1994)

Sowohl die Variante des KE Tableaus mit der SR-Regel als auch die mit den $SR_{\beta 1}$- und $SR_{\beta 2}$-Regeln sind vollständig (D'AGOSTINO und MONDADORI, 1994).

[11] Die Autoren benutzen markierte Formeln. Jede Formel wird entweder mit T oder mit F markiert. Eine mit T markierte Formel ist wahr. Eine mit F markierte Formel ist wiederum falsch. Die Markierungen dienen der Ähnlichkeit der Formeln mit dem Sequenzkalkül. Die Regeln werden aber hier ohne Markierung dargestellt, um leichter mit den anderen Verfahren vergleichbar zu werden.

3.1.3 Tableau mit freien Variablen

Hähnle (2001) empfiehlt bei dem Tableau für die Logik erster Stufe das von Fitting in seinem Buch (FITTING, 1990) vorgestellte Verfahren. Fitting erkennt die Schwierigkeit mit den unendlichen Anwendungen der γ-Regel und verändert sie, damit die Anzahl der Anwendungen abnimmt. Er schlägt vor, statt direkt Terme zu benutzen, eine neue freie Variable einzuführen. Diese neue eingeführte Variable, da sie neu in dem Tableau ist, kann mit jedem Term unifizieren.

Diese Veränderung der γ-Regel setzt aber eine Veränderung an der δ-Regel voraus, wie von Fitting erwähnt (FITTING 1990, S. 139). Die Konstanten, die durch die Anwendung einer δ-Regel eingeführt werden, müssen immer neue Konstanten sein. Sie dürfen nirgendwo in dem Tableau vorhanden sein. Fitting erklärt, dass der durch die Anwendung der δ-Regel eingeführte Konstant nur dann sicherlich neu sein kann, wenn die Belegungen der schon durch die γ-Regel eingeführten freien Variablen berücksichtigt werden. Das ist einfach zu verstehen, wenn wir uns vorstellen, dass diese freien Variablen mit jedem Term unifizieren dürfen. Würden sie nicht berücksichtigt, konnte eine spätere Unifikation bewirken, dass so ein Konstant nicht mehr einmalig im Tableau vorhanden sein wird. Das Tableau mit freien Variablen und dieser δ-Regel ist vollständig (FITTING, 1990).

Abbildung 3.5 — Das Tableau mit freien Variablen.

$\dfrac{\delta}{\delta(f(x_1, ..., x_n))}$	$\dfrac{\gamma}{\gamma(x)}$
wobei f eine neue Skolem Funktion ist und $x_1,..., x_n$ alle durch γ eingeführte freie Variablen sind	wobei x eine neue freie Variable ist

Quelle: Fitting (1990).

Die Bedingung der δ-Regel kann aber verbessert werden (BECKERT *et al.*, 1993), indem statt aller eingeführten freien Variablen nur die freien Variablen der δ-Formel benutzt werden.

Die dadurch entstehende Regel[12] nennt sich δ^+. Das ist der Vorschlag von Hähnle, dessen Algorithmus zuverlässig (HÄHNLE 2001, S. 114) und vollständig (HÄHNLE 2001, S. 121) ist.

3.1.4 Klausel Tableau

Klausel Tableau bezeichnet die Tableau-Verfahren, deren Wissensbasis und Ziel Klauseln sind. Hähnle (2001) erwähnt mehrere Arten von Klausel Tableau. Da Klauseln keine existenzielle Quantifizierungen beinhalten, werden die δ-Regeln bei dieser Art von Tableau erspart. Falls die Eingabe in die KNF ist, werden die α-Regeln auch erspart und die einzige Schlussregel ist die Folgende, wobei κ eine Klausel nach Def. 2.5.2 ist:

Abbildung 3.6 — Das KNF Tableau.

$$\frac{\kappa(x_1, ..., x_n)}{\sigma\kappa_1 \mid ... \mid \sigma\kappa_k}$$

wobei $\sigma = \{t_1 / x_1, ..., t_n / x_n\}$ eine Substitution nach Def. 2.3.1 ist
und $t_1, ..., t_n$ neue freie Variablen sind.

Quelle: Hähnle (2001) und eigene Darstellung[13]

Hähnle (2001) erwähnt auch Hypertableaus. Diese sind besondere Arten von Klausel Tableaus, die nur eine Regel anwenden. Die genaue Regel hängt mit dem Ansatz zusammen. Hier wird nur die für diese Arbeit geplante Regel dargestellt. Die Hyper-Regel sieht so aus, wobei φ_i jeweils positive Literale und ψ_i jeweils negative Literale sind:

[12] Der Artikel handelt hauptsächlich von der δ^{++}-Regel. Da diese Regel nicht in dieser Arbeit implementiert wurde, aber die δ^+-Regel schon, wird hier nur die δ^+-Regel erwähnt.

[13] Das ist die Regel, die Hähnle in seinem Beispiel auf Seite 127 verwendet. Hähnle erklärt diese Regel jedoch durch eine textuelle Beschreibung.

Abbildung 3.7 — Das Hypertableau.

$$\frac{\{\varphi_1, ..., \varphi_k, \psi_{k+1}, ..., \psi_n\}}{\begin{array}{c}\psi_{k+1}{}^{c} \\ ... \\ \psi_n{}^{c}\end{array}}$$

$$\varphi_1 \mid ... \mid \varphi_k$$

Quelle: Hähnle (2001) und eigene Darstellung[14]

3.1.5 Weitere Tableau

Viele Verfahren, wie das *Connection Method*, *Block Tableau* oder das *Model Generation* werden in dieser Arbeit nicht berücksichtigt. Auch verschiedene Arten von Hypertableaus werden nicht angewendet, sondern nur eine beschränkte Art. Es gibt auch in der Literatur Verbesserungen, wie man das Tableau noch leistungsfähiger gestalten kann (vgl. GIESE 2003). Insbesondere bei den Beschreibungslogiken hat das Tableau Verfahren gute Leistungen. Das Tableau für die ALC wird im Abschnitt 3.2 erklärt.

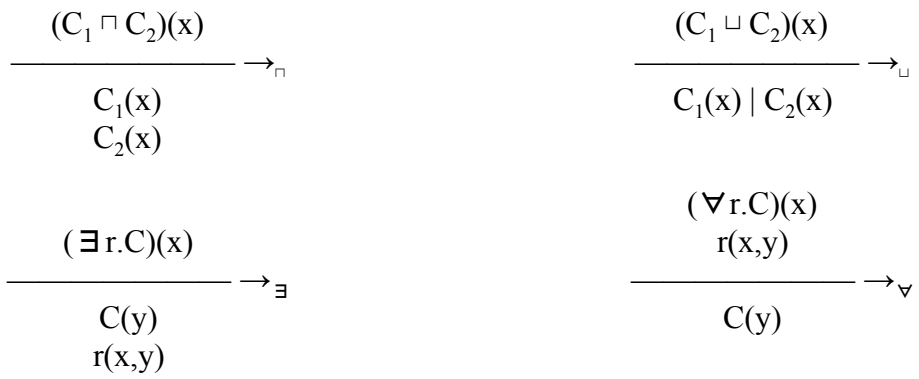
3.2 Das Tableau Verfahren für die ALC

Ein Tableau Verfahren für die ALC soll darauf überprüfen, ob ein Konzept ein anderes Konzept subsumiert. Wir wollen wissen, ob alle Individuen einer bestimmten Klasse C auch Individuen einer anderen Klasse D sind, also ob $C \sqsubseteq D$. Das beweist man, indem ein Tableau für die Formel $(C \sqcap \neg D)(x)$ erstellt wird (BAADER und SATTLER 2000).

Ebenso wie bei dem Tableau für die Logik erster Stufe, werden bei dem ALC-Tableau Regeln angewandt, um den Baum aufzubauen (BAADER und SATTLER 2000):

[14] Viele Varianten von Hypertableau werden von Hähnle auf die Seiten 138-9 vorgestellt.

Abbildung 3.8 — Das Tableau für die ALC.



Quelle: Baader und Sattler (2000) und eigene Darstellung[15]

Obwohl es unmöglich ist, bei der ALC in eine endlose Schleife zu geraten (BAADER und SATTLER 2000, Lemma 1.1), wird auf die iterative Tiefensuche[16] angesichts des Speicheraufwandes nicht verzichtet. Der eigentliche Algorithmus ist infolgedessen bis auf die Anwendung neuer Regeln ähnlich wie der für die Logik erster Stufe. Darum wird dieser Algorithmus in dieser Arbeit verborgen.

3.3 Suchstrategie und Abschließen des Tableaus

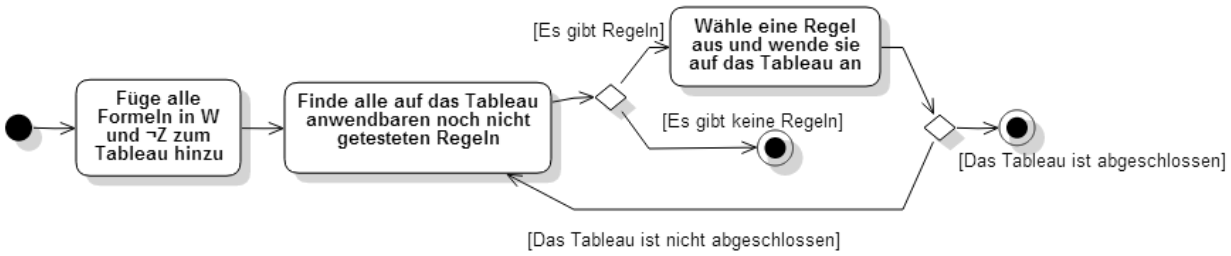
Wie schon am Anfang dieses Kapitels erwähnt, ist die Suchstrategie ein wichtiger Bestandteil eines Tableau Algorithmus. Die Aufgabe besteht darin, so wenige Regeln wie möglich anzuwenden, um so schnell wie möglich das kleinste abgeschlossene Tableau zu erhalten. Vorausgesetzt wird eine Funktion, die die anzuwendenden Regeln aus allen anwendbaren Regeln auswählt. Wenn es einen perfekten Algorithmus gäbe, würde diese Auswahlfunktion immer die richtige Regel aus allen anwendbaren Regeln auswählen. Die richtigen Regeln auszuwählen ist aber nicht so einfach. In der Regel sind folgende Heuristiken nützlich:

[15] Die Regeln werden durch eine Beschreibung präsentiert.
[16] Baader und Sattler (2000) erwähnen eine Tiefensuche.

- ★ **α, δ und γ vor β[17]:** Regeln, die keinen neuen Zweig erzeugen, sondern nur neue Knoten erstellen, haben Vorrang vor verzweigenden Regeln. Insbesondere haben Smullyan α-Regeln Vorrang vor Smullyan β-Regeln. Das ist sehr einfach zu merken, wenn wir uns die Anwendungen dieser Regeln vorstellen. Wenn eine β-Regel zuerst angewandt wird, dann müssen α-Regeln auf mehreren Zweigen angewandt werden. Wenn alle α-Regeln zuerst angewandt werden, dann gibt es nur einen Zweig, auf den wir die β-Regeln anwenden können.
- ★ **Überflüssige nicht verzweigende Regeln:** Wenn alle Folgerungen einer nicht verzweigenden Regel bereits in einem Zweig vorhanden sind, dann ist die Anwendung dieser Regel auf diesen Zweig nicht sinnvoll. Die Folgerungen müssen aber syntaktisch identisch zu den Formeln des Zweiges sein, eine Unifikation reicht nicht aus.
- ★ **Überflüssige verzweigende Regeln:** Wenn mindestens eine Folgerung einer verzweigenden Regel bereits in einem Zweig vorhanden ist, dann ist die Anwendung dieser Regel auf diesen Zweig nicht sinnvoll. Die Folgerung muss aber syntaktisch identisch zu den Formeln des Zweiges sein, eine Unifikation reicht nicht aus.

Folgendes Aktivitätsdiagramm stellt den Algorithmus dar.

Abbildung 3.9 — Die Suchstrategie.



Quelle: eigene Darstellung.

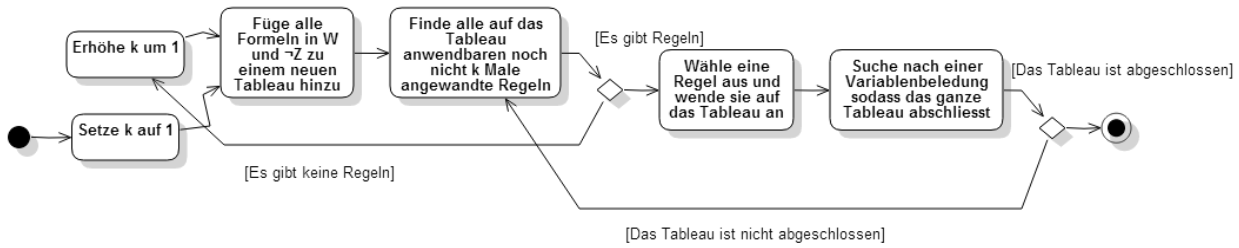
Der Zustand ohne anwendbare Regeln wird nicht erreicht, es sei denn, die Anwendung der γ-Regel wird beschränkt. Um die γ-Regel zu beschränken, wird eine iterative Tiefensuche

[17] Hier sind mit β die Smullyan, die SR, die $SR_{\beta 1}$ und die $SR_{\beta 2}$ β-Regeln gemeint, nicht aber die KE β-Regeln.

(engl. DFID) verwendet, wie von Hähnle (2001) und Harrison (2009) empfohlen. Diese Suchstrategie beschränkt die Anwendungen der γ-Regel **pro Zweig** auf eine Zahl k. Danach wird der Algorithmus ausgeführt und, falls am Ende der Ausführung kein Tableau gefunden wird, wird k erhöht.

Das Abschließen des Tableaus ist auch wichtig. Es wird behauptet, dass das Abschließen ein NP-vollständiges Problem sei (GIESE, 2003). Das Abschließen eines Smullyan Tableaus ist aber nicht NP-vollständig: Sobald das Paar φ und ihr Komplement in einem Zweig gefunden werden, wird der Zweig abgeschlossen und man wendet keine Regel mehr auf diesen Zweig an. Die Schwierigkeit beim Abschließen liegt am Fitting's Tableau Verfahren. Da die freien Variablen verschiedene Belegungen haben dürfen, darf keinen Zweig allein abgeschlossen werden, sonst ist es möglich, dass durch die einen Zweig abschließende Belegung das Abschließen eines anderen Zweigs verhindert wird. Diese Tatsache nennt Hähnle (2001) das *destructive* Tableau. Die Lösung zu diesem Problem ist, immer das ganze Tableau durch eine einzelne Belegung abzuschließen, also ein NP-vollständiges Problem zu lösen. Der Algorithmus wird auf folgender Abbildung dargestellt.

Abbildung 3.10 — Die Suchstrategie mit DFID.

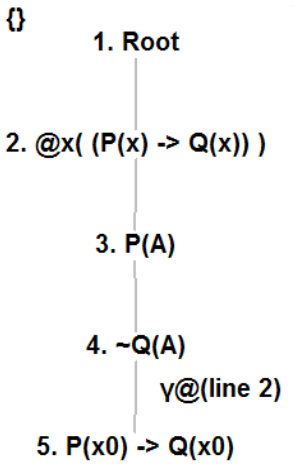


Quelle: eigene Darstellung.

Es gibt aber ein weiteres Problem, das durch die Kombination von Regeln mit mehr als einer Prämisse und die δ- und γ-Regeln für das freie Variablen Tableau von Fitting entsteht. Sagen wir mal, es gibt eine Regel mit Prämissen φ, ψ ∈ **MF** und irgendeine Folgerung. Das heißt, dass wir in dem Tableau zwei Formeln φ' und ψ' finden müssen, sodass es einen allgemeinsten Unifikator σ gibt, damit σ(φ) = σ(φ') und σ(ψ) = σ(ψ'). Wie vor Kurzem erwähnt,

wird die Unifikation nur beim Abschließen eines Tableaus angewandt, was wiederum bewirkt, dass Regeln mit mehreren Parametern, die Unifikationen voraussetzen, niemals angewandt werden, weil die Formeln im Tableau einfach nicht unifizieren (Beispiel auf nächster Abbildung). Auf der Abbildung kann die KE β-Regel mit Folgerung $\{A / x_0\}$ $Q(x_0)$ nicht auf Zeile 5 angewandt werden, da x_0 mit A nicht unifizieren darf.

Abbildung 3.11 — Anwendung der KE β-Regel.

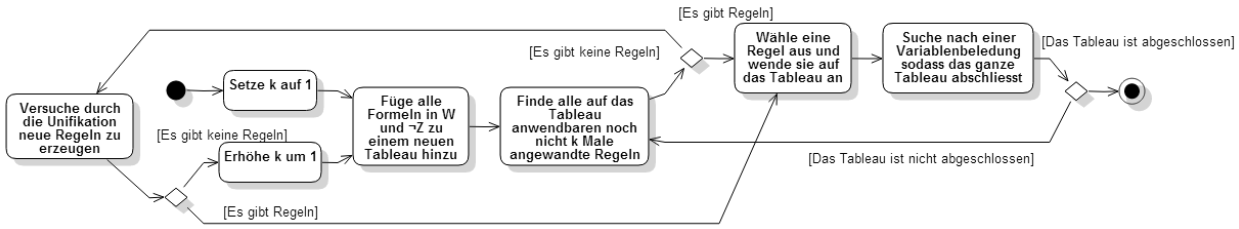


Quelle: eigene Darstellung.

In dem gezeigten Beispiel ist es möglich, einen Beweis ohne die Anwendung der β-Regel zu finden. Es gibt aber kompliziertere Beispielen, bei denen der Beweis nicht so einfach zu finden ist[18]. Deswegen muss eine letzte Änderung an dem Algorithmus vorgenommen werden. Wir fügen einen Schritt hinzu, der nach neuen Regeln sucht. Diese neuen Regeln sind nur durch die Benutzung der Unifikation möglich.

[18] Es muss ein Beispiel geben, bei dem es keinen Beweis gibt. Ich habe jedoch keinen Beweis für das Vorhandensein eines solchen Beispieles. Falls so ein Beispiel gefunden wird, dann ist der KE Algorithmus mit freien Variablen ohne Anwendung der Unifikation bei der Erzeugung neuer Regeln unvollständig.

Abbildung 3.12 — Die Suchstrategie mit DFID und Unifikation.



Quelle: eigene Darstellung.

3.4 Gleichung

Die Gleichung ist eine besondere Relation, die drei Eigenschaften besitzt: Reflexivität, Symmetrie und Transitivität. Um die Gleichung auch in dieser Arbeit zu benutzen, werden die folgenden Regeln verwendet (FITTING, 1990):

Abbildung 3.13 — Die Schlussregeln für die Gleichung.

$\frac{t = u \quad \varphi(t)}{\varphi(u)}$ wobei $\varphi(t)$ und $\varphi(x)$ atomische Formeln sind	$\frac{}{x = x}$ wobei x eine freie Variable ist	$\frac{}{f(x_1, ..., x_n) = f(x_1, ..., x_n)}$ wobei $x_1, ..., x_n$ freie Variablen sind und f eine Skolem Funktion ist

Quelle: Fitting (1990).

Die Skolem Funktionen werden nicht erzeugt, sondern sie sind die Funktionen, die in anderen Formeln vorhanden sind.

3.5 Zusammenfassung

In diesem Kapitel haben wir uns mit dem Tableau Verfahren befasst. Wir haben gesehen, dass das freie Variablen Tableau von Fitting i. d. R. besser als das ursprüngliche Smullyan

Tableau ist. Außerdem wurde eine iterative Tiefensuche vorgeschlagen, um die Anwendungen der γ-Regel zu beschränken. Das Abschließen des Tableaus bleibt jedoch in dieser Arbeit ein schwieriges Problem. Das nächste Kapitel stellt die Implementierung vor.

4 DER THEOREMBEWEISER

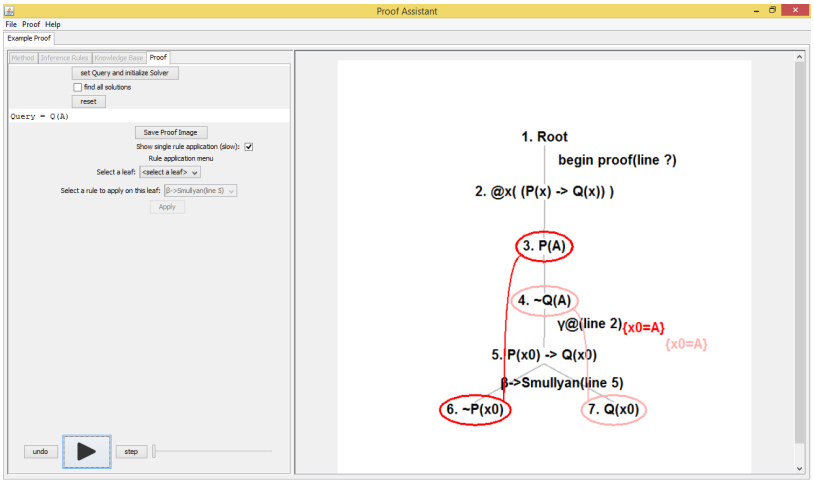
In diesem Kapitel werden alle neue Funktionalitäten vorgestellt, sowie die implementierten Algorithmen, Tests und die endgültige Klassenhierarchie.

4.1 Die neuen Funktionalitäten

4.1.1 Die grafische Darstellung des Beweises

Eine neue Darstellungsart der Beweise wurde implementiert. Dies ist eine grafische Darstellung, die der Darstellung von Letz (D'AGOSTINO *et al.*, 1999) sehr ähnelt. Die Implementierung nimmt einen Baum als Eingabe, um ein Bild zu erzeugen. Die Anwendung von Unifikation und von verschiedenen Schlussregeln kann anhand des Bildes gezeigt werden. Außerdem bietet die Implementierung die Möglichkeit, dass auch andere Beweisverfahren (nicht nur das Tableau) diese Art Darstellung benutzen, solange diese Verfahren diesselbe Datenstruktur benutzen, i. e., einen Baum. Das nächste Bild zeigt ein Beispiel der grafischen Darstellung für den Beweis von $\forall x(P(x) \rightarrow Q(x)), P(A) \vDash_{Tableau} Q(A)$.

Abbildung 4.1 — Grafische Darstellung des Beweises.



Quelle: eigene Darstellung.

Für eine genauere Beschreibung der GUI, siehe bitte Anhang B - Benutzerhandbuch.

4.1.2 Die Beschreibungslogiken

Eine abstrakte Syntax der ALC Beschreibungslogik wurde erstellt. Der Parser wurde erweitert, sodass er auch ALC Formeln erkennen kann. Das Tableau, das das Subsumptionsproblem der ALC Logik löst, wurde auch selbstverständlich implementiert.

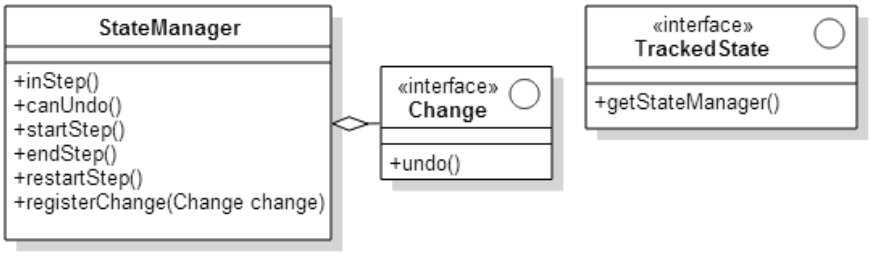
4.1.3 Tests

Viele Tests wurden benutzt, um das Tableau auf Fehler zu überprüfen. Alle ersten 68 Tests im Artikel von Pelletier (1986) wurden mittels JUnit implementiert. Diese Tests können auch jetzt wiederverwandt werden, indem man sie benutzt, um auch Fehler bei anderen Beweisverfahren aufzudecken.

4.1.4 StateManager

Die Tatsache, dass der Benutzer den Beweis beliebig steuern kann, bringt große Schwierigkeiten mit sich. Der Zustand von vielen Klassen muss gespeichert werden und die Änderungen müssen rücksetzbar sein. Da manche Änderungen miteinander zusammenhängen, und deswegen diese Änderungen alle auf einmal bei einem *undo* zurückgesetzt werden, ist es sinnvoll auch diese Regeln in einem ganzen Schritt zu speichern. Die Speicherung und das Zurücksetzen dieser Schritte werden von der StateManager Klasse durchgeführt.

Abbildung 4.2 — Das StateManager Paket.



Quelle: eigene Darstellung[19].

Die StateManager Klasse und die Change und TrackedState Schnittstellen sind wiederverwendbar.

4.1.5 Weitere Verbesserungen

Kleinere Verbesserungen wurden auch überall in der Software durchgeführt. Die GUI Klasse wurde zerlegt und die Möglichkeit, mehrere Beweise gleichzeitig auszuführen, wurde implementiert. Der Parser erkennt außerdem auch Gleichungen.

4.2 Gestaltung und Algorithmen

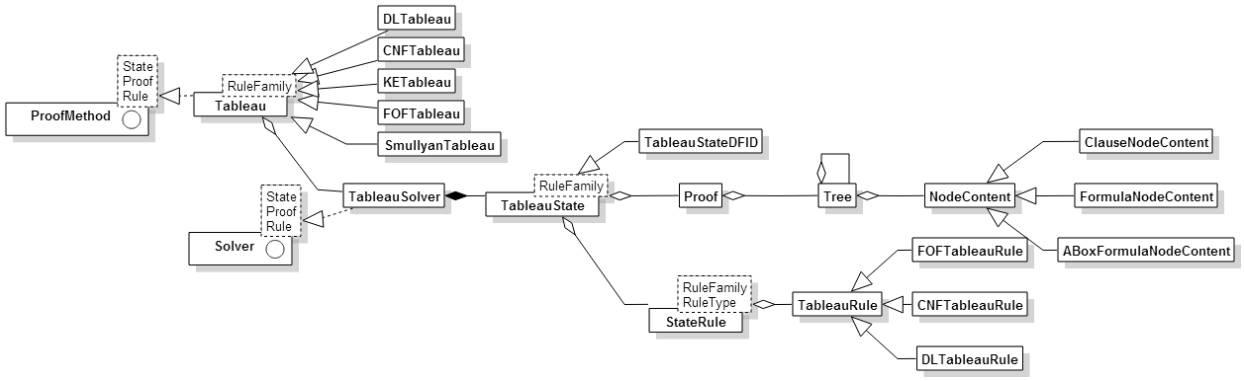
Dieser Abschnitt erklärt und begründet die Gestaltung der Software und zeigt, welche Algorithmen implementiert wurden.

4.2.1 Klassenhierarchie

Auf der folgenden Abbildung ist die Klassenstruktur der Software dargestellt. Eine Begründung der Gestaltung folgt.

[19] UML-Werkzeug: StarUML, verfügbar über http://staruml.io/.

Abbildung 4.3 — Die Klassenhierarchie.



Quelle: eigene Darstellung[20].

Wie auf der Abbildung zu erkennen ist, implementiert die Tableau Klasse die ProofMethod Schnittstelle und die TableauSolver Klasse wiederum die Solver Schnittstelle. ProofMethod und Solver sind zwei Schnittstellen, die während des Projektes von dem Team entwickelt wurden, damit die GUI alle unterschiedlichen Beweisverfahren darstellen konnte. Durch die Wiederverwendung von diesen Schnittstellen wird die Integration des Tableaus in die GUI einfach.

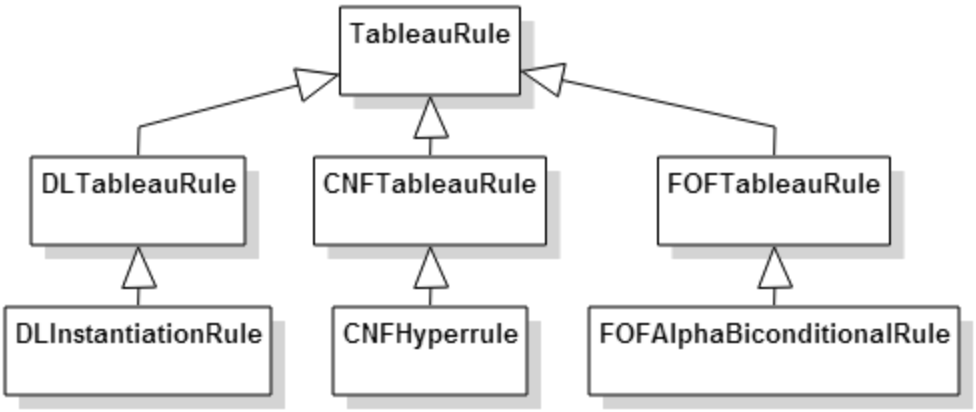
Es gibt auch mehreren Arten von Tableau. SmullyanTableau und KETableau benutzen eine vordefinierte Liste von Inferenzregeln, damit der Benutzer nicht jedes Mal alle Inferenzregeln einstellen muss, wenn er einen Beweis ausführen möchte. Der Benutzer kann aber die Inferenzregeln beliebig einstellen. Die Klassen DLTableau, CNFTableau und FOFTableau erlauben den Benutzer, beliebige Inferenzregeln zu verwenden, solange diese Regeln der entsprechenden Regelmenge gehören (bzw. DLTableauRule, CNFTableauRule und FOFTableauRule).

Alle Inferenzregeln erben die Methoden und Eigenschaften von der abstrakten Klasse TableauRule. Es gibt drei Arten von Inferenzregeln: FOFTableauRule, CNFTableauRule und DLTableauRule. Die eigentlichen Inferenzregeln wurden in der Abbildung verborgen und sie

[20] UML-Werkzeug: StarUML, verfügbar über http://staruml.io/.

erben Methoden und Eigenschaften von den abstrakten Klassen FOFTableauRule, CNFTableauRule und DLTableauRule.

Abbildung 4.4 — Die Klassen der Schlussregeln.



Quelle: eigene Darstellung[21].

Die Regeln können nach ihrer Anwendung auch andere Regeln erzeugen. Wenn bspw. auf die Formel $P(f(x)) \wedge (P(h(y)) \rightarrow Q(A))$ eine α-Regel angewandt wird, woraus die Formeln $P(f(x))$ und $P(h(y)) \rightarrow Q(A)$ entstehen, muss auch eine β-Regel für die Formel $P(h(y)) \rightarrow Q(A)$ erstellt werden. Dafür ist die Klasse TableauRule auch verantwortlich. Die Methode *getGeneratedRulesOfType()* erstellt neue Regeln. Darüber hinaus bestimmt TableauRule, ob eine Regel auf ein bestimmtes Blatt angewandt werden darf.

Der Beweis (Proof) ist im Grunde genommen einfach ein Baum. Alle Knoten von diesem Baum tragen ein NodeContent Objekt. Da es mehrere Arten von NodeContent gibt, ist es möglich, auch mehrere Logiken durch diese Datenstruktur darzustellen.

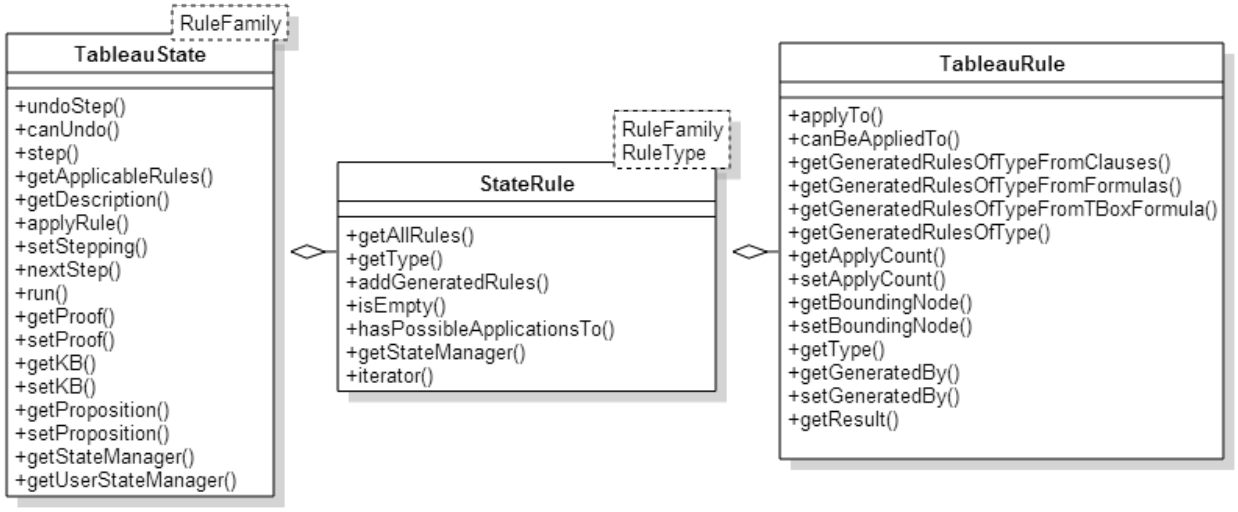
Die TableauSolver Klasse benutzt die TableauState Klasse, die den eigentlichen Algorithmus ausführt. Die TableauState Klasse hat eine Liste von StateRule Klassen. Jedes StateRule Objekt bildet genau eine Art von Inferenzregel, genauer gesagt hat die StateRule Klasse zwei Parametern: das RuleFamily und das RuleType. RuleFamily ist eine von den

[21] UML-Werkzeug: StarUML, verfügbar über http://staruml.io/.

FOFTableauRule, CNFTableauRune und DLTableauRule Klassen. RuleType ist eine Klasse, die von der RuleFamily Klasse erbt.

TableauState bietet eine Menge von Funktionen an. Die Methode *step()* führt einen ganzen Schritt aus, indem alle Beweisbäume mit einer bestimmten Anzahl an Anwendungen von Regeln überprüft werden. Die Methode *applyRule()* erlaubt dem Benutzer, eine Regel anzuwenden. Außerdem beginnt die Methode *setStepping()* einen Zustand, bei dem Benutzer und Beweiser zusammen den Beweis aufbauen.

Abbildung 4.5 — TableauState, StateRule und TableauRule.



Quelle: eigene Darstellung[22].

Die StateRule Klasse ist eine Art Warteschlange von Regeln. Sie enthält eine Liste mit allen Inferenzregeln ihrer Art, stellt diese Regeln durch einen Iterator in einer Reihe zur Verfügung und passt auf die Anwendung der Regeln auf. Jede Regel darf nur eine begrenzte Anzahl von Malen angewandt werden. Wenn diese Anzahl erreicht wird, dann wird die Regel nicht mehr von der StateRule Klasse angeboten. Die Regel kann erst dann wieder angeboten werden, wenn entweder ein *undo* durchgeführt wird oder die begrenzte Anzahl von Anwendungen erhöht wird. Durch die Verwendung von der StateRule Klasse ist es sehr einfach,

[22] UML-Werkzeug: StarUML, verfügbar über http://staruml.io/.

eine beliebige Menge von Inferenzregeln zu benutzen. Es reicht, ein StateRule Objekt pro Regel zu erstellen.

Die TableauState ist außerdem eine abstrakte Klasse. Die TableauStateDFID Klasse erbt von der TableauState Klasse und implementiert die Iterative Tiefensuche (DFID) Strategie. Andere Strategien können auch implementiert werden, indem man andere von dem TableauState erbende Klassen schreibt. Schließlich erlaubt das TableauState-StateRule Verhältnis eine Reihenfolge der Regelarten, d. h., manche Regeln werden immer, sobald sie verfügbar sind, vor anderen Regeln angewandt. Die genaue Reihenfolge kann vom Benutzer beliebig eingestellt werden.

4.2.2 Algorithmen

Die iterative Tiefensuche aus Abschnitt 3.3 wurde implementiert, sowie das Abschließen des gesamten Tableaus, statt einzelner Zweige. Die implementierten Regeln wurden mit dieser Suche kombiniert, wodurch 4 Algorithmen entstanden sind:

- ★ Das Smullyan Tableau benutzt die α- und β-Regeln aus Abschnitt 3.1.1 und die δ^+- und γ-Regeln mit freien Variablen aus Abschnitt 3.1.3.
- ★ Das KE Tableau benutzt die α- und β-Regeln aus Abschnitt 3.1.2, sowie die $SR_{\beta 1}$- und $SR_{\beta 2}$-Regeln, aber nicht die SR-Regeln und die δ^+- und γ-Regeln mit freien Variablen aus Abschnitt 3.1.3.
- ★ Das KNF Tableau benutzt die erste Regel aus Abschnitt 3.1.4 und eine schwächere Version von dieser Regel, die nur Formeln ohne Variablen behandelt.
- ★ Das Hypertableau benutzt die im Abschnitt 3.1.4 vorgestellte Hyperregel.

Alle Algorithmen benutzen auch die Regeln für die Gleichung aus Abschnitt 3.4. Andere Ansätze können erstellt werden, indem der Benutzer die Regeln bestimmt.

4.3 Zusammenfassung

Dieses Kapitel hat die grafische Darstellung des Beweises vorgestellt sowie andere neue Funktionalitäten. Des Weiteren wurde die Klassenhierarchie erklärt und viele Vorteile der Gestaltung erwähnt, wie die Einstellung der Schlussregeln oder die Trennung zwischen Suchstrategie und Schlussregeln: Die Klasse TableauState alleine kann mit verschiedenen Schlussregeln Arten kombiniert werden, um verschiedene Algorithmen zu erstellen. Das nächste Kapitel stellt die Ergebnisse der Tests vor.

5 AUSWERTUNG

In diesem Kapitel wird die angefertigte Software ausgewertet, indem die Ergebnisse der Tests vorgestellt werden und die Eigenschaften der Software hinsichtlich der Qualität hervorgehoben werden.

5.1 Zuverlässigkeit und Vollständigkeit

Der Artikel von Pelletier (1986) stellt 75 Tests vor. Das sind logische Probleme, die jeweils irgendeine Besonderheit haben: Manche sind schwer mittels eines bestimmten Ansatzes zu lösen, manche sind schwer in die KNF umwandelbar, manche sind einfach zu lang. Die ersten 68 Tests wurden implementiert. Sie überprüfen darauf, ob das Programm zuverlässig ist. Die Tests #71 bis #75 stellen die Leistungsfähigkeit des Programms fest. Aus diesen Tests wurde Test #71 ausgewählt und implementiert. Die Tabelle mit den Ergebnissen aller Tests befindet sich in den Anhängen.

Da der Vorgang bei manchen Tests sehr lang dauern kann, wurde eine Zeitgrenze von einer Minute gezogen. Wenn nach einer Minute das Problem nicht gelöst wird, dann wird angenommen, dass das Problem nicht innerhalb akzeptabler Zeit gelöst werden kann. Darüber hinaus wurden die folgenden Algorithmen getestet:

- ★ Smullyan Tableau mit Unifikation;
- ★ KE Tableau mit Unifikation;
- ★ Hypertableau mit Unifikation;
- ★ KNF Tableau mit Unifikation.

Der Rechner, der die Tests ausgeführt hat, ist ein acer Aspire E 15 (Modell E5-571G-3188). Die Dauer der Tests wurde durch die Klasse Timeout[23] gemessen. Die Dauer enthält auch die Zeit der Erstellung der grafischen Darstellung des Beweises. Da kein

[23] Genauer Pfad org.junit.rules.Timeout

Profiling-Werkzeug benutzt wurde, sondern einfach JUnit, ist die gemessene Dauer jedes Tests nicht zuverlässig. Deshalb werden die Ergebnisse hier zusammengefasst. Die Dauer jedes einzelnen Tests wird im Anhang A vorgestellt.

Tabelle 5.1 — Ergebnisse der Zuverlässigkeitstests.

Ansatz	AeT[24] innerhalb 5 Sek.	AeT innerhalb 1 Min.
Smullyan	26 / 68	38 / 68
KE	25 / 68	31 / 68
Hypertableau	29 / 68	31 / 68
KNF Tableau	30 / 68	30 / 68

Quelle: eigene Darstellung.

Pelletier (1986) ordnet seine Tests fünf Klassen zu. Jede Klasse richtet sich nach einer bestimmten Untermenge der Logik erster Stufe, sodass die Problemen der ersten Klasse viel leichter zu lösen sind, als die der letzten Klasse. Die Klassen sind:

- ★ Aussagenlogik (hier als K_1 bezeichnet) beinhaltet Probleme #1 bis #17;
- ★ Monadische Logik erster Stufe (hier als K_2 bezeichnet) beinhaltet Probleme #18 bis #34;
- ★ Logik erster Stufe ohne Gleichungen und Funktionen (hier als K_3 bezeichnet) beinhaltet Probleme #35 bis #47;
- ★ Logik erster Stufe mit Gleichungen und ohne Funktionen (hier als K_4 bezeichnet) beinhaltet Probleme #48 bis #55;
- ★ Logik erster Stufe mit Gleichungen und Funktionen (hier als K_5 bezeichnet) beinhaltet Probleme #56 bis #70;

[24] Anzahl erfolgreicher Tests.

Bemerkenswert ist die Tatsache, dass die meisten erfolgreichen Tests auch die einfachsten Tests waren, wie auf der nächsten Abbildung zu sehen ist.

Tabelle 5.2 — Erfolgreiche Tests pro Logik.

Ansatz	K_1	K_2	K_3	K_4	K_5
Smullyan	17 / 17	11 / 17	5 / 13	2 / 8	3 / 15
KE	17 / 17	7 / 17	5 / 13	2 / 8	0 / 15
Hyper	17 / 17	7 / 17	5 / 13	1 / 8	0 / 15
KNF	17 / 17	7 / 17	3 / 13	2 / 8	2 / 15

Quelle: eigene Darstellung.

Aus der Tatsache, dass manche Tests nicht erfolgreich waren, kann man nicht schließen, dass die Software unvollständig[25] ist. Grund dafür ist das Timeout. Es kann sein, dass die Tests bei längerem Timeout erfolgreich wären. Darauf wird von den Ergebnissen der Tests für den Smullyan Ansatz hingewiesen.

Anschließend wurden diesselbe Pelletier Tests mit einem kleinen Unterschied wieder ausgeführt. Das Ziel von jedem einzelnen Test wurde in dieser Phase vor dem Beginn des Beweises negiert, sodass die Probleme nicht lösbar waren. Beispielsweise ist Problem #59 das Folgende (PELLETIER, 1986):

★ $\forall x.\ F(x) \leftrightarrow \neg F(f(x)) \vDash \exists x.\ F(x) \wedge \neg F(f(x))$

Das Problem mit dem negierten Ziel wäre dann:

★ $\forall x.\ F(x) \leftrightarrow \neg F(f(x)) \vDash \forall x.\ \neg F(x) \vee F(f(x))$

[25] Ein vollständiges Verfahren terminiert nach einer endlichen Anzahl an Schritten und findet ein abgeschlossenes Tableau, solange es ein abgeschlossenes Tableau gibt.

Alle 4 Verfahren wurden auf diese Probleme angewandt, aber für keine[26] von diesen 67 neuen Problemen konnte ein Beweis gefunden werden. Das deutet auf die Zuverlässigkeit der Software hin.

Das Verfahren für die ALC wurde mittels eines über das Internet verfügbaren Test ausschließlich auf ihre Korrektheit geprüft. Dieser Test wird in dieser Arbeit deshalb verborgen.

5.2 Leistungstests

Wie im letzten Abschnitt erwähnt, stellt Test #71 (PELLETIER, 1986) die Leistungsfähigkeit des Programms fest. Das zu lösende Problem sieht so aus:

Tabelle 5.3 — Pelletier Test #71.

Problem 1	$\models_{Tableau} P_1 \leftrightarrow P_1$
Problem 2	$\models_{Tableau} P_1 \leftrightarrow (P_2 \leftrightarrow (P_1 \leftrightarrow P_2))$
Problem 3	$\models_{Tableau} P_1 \leftrightarrow (P_2 \leftrightarrow (P_3 \leftrightarrow (P_1 \leftrightarrow (P_2 \leftrightarrow P_3))))$
Problem n	$\models_{Tableau} P_1 \leftrightarrow (P_2 \leftrightarrow (P_3 \leftrightarrow ... (P_n \leftrightarrow (P_1 \leftrightarrow (P_2 \leftrightarrow (P_3 \leftrightarrow ... (P_{n-1} \leftrightarrow P_n))))$

Quelle: Pelletier (1986)

Die Aufgabe besteht darin, zu erschließen, wie gut sich der Algorithmus bei immer größer werdenden Eingaben verhält. In folgender Tabelle erfolgen die Angaben in Sekunden.

[26] Für Test #25 wurde einen Beweis gefunden, aber das liegt daran, dass die Prämissen von Test #25 widersprüchlich sind.

Tabelle 5.4 — Ergebnisse der Leistungstests.

Ansatz	P1	P2	P3	P4	P5	P6	P7
Smullyan	< 1	~2	~14	~219	F[27]		
KE	< 1	< 1	~4	~17	~55	~901	F[28]
Hyper	< 1	< 1	~11	F[29]			
KNF	< 1	< 1	~10	F[30]			

Quelle: eigene Darstellung.

Es gibt hier einen klaren Unterschied zwischen den KNF Methoden und den FOF (*First-Order Form*) Methoden. Denn alle Eingaben in FOF erfolgen, müssen die KNF Methoden zuerst die KNF Umwandlung ausführen. Diese Umwandlung ist jedoch sehr schwierig und dauert so lang und verwendet so viel Speicher, dass die Tests schnell fehlschlagen. Die Umwandlung in die KNF ist tatsächlich eine Schwäche von diesen Methoden.

Bei den Smullyan und KE Tableau hingegen ist die Leistung ein bisschen besser. Das Smullyan Tableau kann noch P4 lösen. Das ist dank der FOF Eingabe. Das KE Tableau kann erstaunlicherweise sogar P6 innerhalb weniger als einer Stunde lösen. Das hängt bestimmt mit der Tatsache zusammen, dass das KE die Verzweigung vermeidet, und daher kürzere Beweise erstellt, wie die Autoren selbst behaupten (D'AGOSTINO und MONDADORI, 1994).

5.3 Ergebnisse

Nicht nur die Effizienz und die Zuverlässigkeit der Software sind wichtig. Es gibt auch eine Menge an Eigenschaften, die genauso wichtig sind, wie die Benutzbarkeit und die Wartbarkeit. In folgender Tabelle wird eine kurze kritische Analyse der Software hinsichtlich der

[27] Nach einer Stunde wurde kein Ergebnis gefunden und den Test deswegen unterbrochen.
[28] Nach einer Stunde wurde kein Ergebnis gefunden und den Test deswegen unterbrochen.
[29] Nach 20 Minuten wurde eine OutOfMemoryError Ausnahme geworfen. Die KNF Umwandlung war nicht fertig.
[30] Nach 26 Minuten wurde eine OutOfMemoryError Ausnahme geworfen. Die KNF Umwandlung war nicht fertig.

Qualität gegeben. Denn keine ausführliche Analyse wurde gemacht, dient diese Analyse nur der Beschreibung der Software. Keine Softwaremetrik wurde verwendet. Die Eigenschaften wurden dem Leistungsheft des WS 13/14 Projektes entnommen.

Tabelle 5.5 — Qualität der Ergebnisse.

Zuverlässigkeit	Die Zuverlässigkeit der Software wurde durch die Tests aus Abschnitt 5.1 bewiesen.
Robustheit	Die geschriebenen Tests überprüfen ausschließlich die Zuverlässigkeit der Tableau Klassen. Keine Tests wurden erstellt, die die einzelnen Modulen überprüfen. Deshalb ist die Robustheit der Software nicht garantiert.
Leistungsfähigkeit	Die Software ist nicht leistungsfähig. Das wird durch die Tests, die nicht erfolgreich waren, bewiesen. Da die meisten Tests so klein waren, hätte die Software auch in der Lage sein, sie innerhalb einer Minute zu lösen.
Benutzbarkeit	Die grafische Darstellung der Beweise erleichtert das Verständnis. Außerdem ist die Software sehr interaktiv.
Wartbarkeit	Die Klassenhierarchie erleichtert die Wartbarkeit und Erweiterbarkeit. Bspw. können die TableauState und TableauRule Klassen leicht erweitert werden. Die Implementierung der Schlussregeln hätte jedoch kürzer sein müssen. Die im Kapitel 3 vorgestellten Regeln wurden alle implementiert, wodurch eine Menge von 41 Klassen entstanden ist. Diese riesige Anzahl hätte kleiner sein können, wenn die Klassen abstrakter geschrieben würden. So wäre die Software leichter zu warten.
Dokumentation	Javadoc wurde in vielen Klassen verwendet, um den Code verständlicher zu machen. Außerdem kann der Benutzer im Anhang B dieser Arbeit nachschlagen, um die GUI besser zu verstehen.

Quelle: eigene Darstellung.

5.4 Zusammenfassung

In diesem Kapitel wurden die Testergebnisse vorgestellt. Wir haben gesehen, dass die Software zuverlässig ist, obwohl nicht leistungsfähig. Außerdem wurden viele Eigenschaften der Software kritisch diskutiert. Kapitel 6 stellt mögliche Verbesserungen vor.

6 ZUSAMMENFASSUNG UND AUSBLICK

Die im WS 13/14 entwickelte Software war sehr leicht erweiterbar. Daher lief die Integration ohne Probleme. Dennoch geschah die Implementierung nicht ohne Umstände. Bei der Bearbeitung hat der Student mehrere Herausforderungen erkennt und die Lösungen dazu entweder recherchiert oder selber entwickelt. Viele von diesen Herausforderungen wurden leider nicht überwunden.

Eine von den bestehenden Herausforderungen ist das Abschließen vom Tableau. Wie bereits erwähnt, den richtigen Unifikator zu finden, mit dem das Tableau abschließt, ist ein NP-vollständiges Problem. Dieses Problem wird durch reine blinde Suche in dieser Arbeit gelöst. Eine bessere Alternative wäre der Ansatz von Giese (2003). Giese speichert das Ergebnis vom letzten Versuch, das Tableau abzuschließen, damit der nächste Versuch wesentlich leichter wird.

Eine andere Herausforderung war die Gestaltung der Suchstrategie. Nicht alle möglichen Verbesserungen wurden implementiert, bspw. überprüft der Algorithmus nach jeder Regelanwendung die Anwendbarkeit aller Regeln auf alle Blätter des Beweises noch mal. Das hätte besser sein können, wenn nur die Anwendbarkeit auf die neuen Blätter überprüft würde. Darüber hinaus ist die Beschränkung der γ-Regel, so wie sie gestaltet wurde, nicht ganz geeignet. Stellen wir uns vor, es gibt eine Formel mit mehreren Quantifizierungen, z. B. $\forall x \exists y \exists z. \neg((P(y) \rightarrow Q(z)) \rightarrow (P(x) \rightarrow Q(x)))$ (das negierte Problem #19 von Pelletier, 1986). Auch wenn die γ-Regel nur zwei Male angewandt werden darf, erstellt jede Anwendung der γ-Regel eine neue δ-Regel, denn es gibt mehrere Quantifizierungen. Diese neuen δ-Regeln dürfen jeweils zwei Male angewandt werden und bei ihrer Anwendung werden noch mehr δ-Regeln erstellt. Es ist offensichtlich, dass mit wenigen Quantifizierungen die Anzahl der Regeln rasch steigt. Die Quantifizierungen sind vermutlich der Grund, warum die FOF Ansätze, nämlich das Smullyan und das KE Tableau, das Problem #19 nicht innerhalb einer Minute lösen konnten, während die KNF Ansätze schon in sehr kurzer Zeit.

Bei der Suchstrategie ist die Wahl der anzuwendenden Regeln auch von wesentlicher Bedeutung. Diese Entscheidung erfolgt in dieser Arbeit mittels einer von dem Benutzer eingestellten Priorisierung der Schlussregeln. Manchmal ist diese Priorisierung leider einfach

falsch. Bspw. wurden die Reflexivitätsregeln für die Gleichung vom Hypertableau Ansatz beim Problem #60 unglaublich viele Male verwendet[31], obwohl sie gar nicht zum Abschließen des Tableaus beigetragen haben. Dazu zählt, dass je mehr Regeln angewandt werden, desto größer wird das Tableau und damit schwieriger das Problem, die Anwendbarkeit einer Regel zu überprüfen. Hierzu wäre eine gute Lösung das *Connection Tableau* (HÄHNLE, 2001). Das *Connection Tableau* erlaubt eine Regelanwendung genau dann, wenn diese Anwendung zum Abschließen des Tableaus beiträgt.

In der Auswertung wurde erwähnt, dass der Code restrukturiert werden muss. Besonders schlecht ist der Zustand der Schlussregeln Klassen, es gibt einfach zu vielen. Es wäre möglich, die Notation von Smullyan umzusetzen, irgendwas in der Art *Formula.getFirstAlphaComponent*, *Formula.getSecondAlphaComponent* usw. Dadurch wäre es nicht mehr nötig, eine Klasse für jede Art von α, β, δ und γ-Formeln zu erstellen, also 13 Klassen, sondern nur eine Klasse fur die α-, eine für die β-, eine für die δ- und eine für die γ-Formeln. Außerdem sind andere Klassen einfach zu lang, kompliziert und unverständlich. Diese Klassen müssen irgendwie zerlegt und restrukturiert werden.

Schließlich gibt es nach der Beseitigung aller Fehler und Implementierung aller vorgeschlagenen Verbesserungen immer noch Vieles, was getan werden kann. Die vorliegende Arbeit hat vier Ansätze implementiert und vergleicht, aber es gibt zahlreiche weitere Ansätze nicht nur für die klassische Logik, sondern auch für nicht klassische Logiken. Außerdem gibt es andere Möglichkeiten, den Beweis zu präsentieren. Wenn die Baumdarstellung zu groß und daher unbequem wird, dann wären vielleicht die Matrizen von Hähnle (S. 121-2, 2001) eine gute Idee.

[31] Alle durch die Tests erstellten Beweise werden zusammen mit dieser Arbeit auf einer Festplatte abgegeben.

LITERATURVERZEICHNIS

BAADER, F.; MCGUINESS, D. L.; NARDI, D; PATEL-SCHNEIDER, P. F. **The Description Logic Handbook: Theory, implementation, applications**. Cambridge: Cambridge University Press, 2003.

BAADER, F; SATTLER, U. Tableau Algorithms for Description Logics. In: DYCKHOFF, R. (Hrsg.). **Automated Reasoning with Analytic Tableaux and Related Methods: International Conference, TABLEAUX 2000**. St Andrews, Schottland, UK, Juli 3-7. Berlin: Springer, 2000.

BECKERT, B; HÄHNLE, R; SCHMITT, P. H. **The Even More Liberalized δ-Rule in Free Variable Semantic Tableaux**. doi:10.1007/BFb002255. 1993.

D'AGOSTINO, M.; GABBAY, D. M.; HÄHNLE, R.; POSEGGA, J. (Hrgs). **Handbook of Tableau Methods**. doi:10.1007/978-94-017-1754-0. 1999.

D'AGOSTINO, M.; MONDADORI, M. **The Taming of the Cut. Classical Refutations with Analytic Cut.** doi: 10.1093/logcom/4.3.285. 1994.

FITTING, M. **First-Order Logic and Automated Theorem Proving.** New York u.a.: Springer. 1990. ISBN: 3-540-97233-1

GALMICHE, D.; LARCHEY-WENDLING, D. (Hrgs.). **Automated Reasoning with Analytic Tableaux and Related Methods: 22nd International Conference, TABLEAUX 2013**, Nancy, Frankreich, 16-19 September 2013. Berlin: Springer-Verlag Heidelberg, 2013.

GIESE, M. **Incremental Closure of Free Variable Tableaux**. Verfügbar über http://www.uio.no/studier/emner/matnat/ifi/INF5170/h03/filer/giese_handout.pdf. Der 14. November 2003.

HÄHNLE, R. Tableau and Related Methods. In: Robinson, A., Voronkov, A. **Handbook of Automated Reasoning I** (S. 103-137). Elsevier Science Publishers B. V, 2001.

HARRISON, J. **Handbook of Practical Logic and Automated Reasoning**. Cambridge: Cambridge University Press, 2009.

PELLETIER, F. J. **Seventy-five Problems for Testing Automatic Theorem Provers**. doi:10.1007/BF02432151. 1986.

PELLETIER, F. J. **Errata**. Journal of Automated Reasoning. ISSN: 0168-7433. 1988.

PELLETIER, F. J.; SUTCLIFFE, G. **An Erratum for Some Errata to ATP Problems**. 10.1023/A:1005764705033. 1997.

RUSSEL, S.; NORVIG, P. **Künstliche Intelligenz: Ein moderner Ansatz**. München: Pearson Deutschland GmbH. 2012.

SCHMIDT-SCHAUBß, M.; SMOLKA, G. **Attributive Concept Descriptions with Complements**. doi:10.1016/0004-3702(91)90078-X. Feb. 1991.

SMULLYAN, R. M. **First-Order Logic**. New York: Springer. 1968.

SUTCLIFFE, G.; SUTTNER, C. The State of CASC. **AI Communications, 19**. Abgerufen von http://iospress.metapress.com/content/aymnb5mq1fqy69c9/. 2006.

SUTCLIFFE, G.; SUTTNER, C. **The TPTP Problem Library for Automated Theorem Proving**. http://www.cs.miami.edu/~tptp/. Zugegriffen: Oktober 2014. 2014.

ANHANG A - TESTS

Der Student hat die Tests von einem wissenschaftlichen Artikel (PELLETIER, 1986) verwendet, um die eigene Implementierung auszuwerten. Einige der Tests in dem genannten Artikel waren jedoch leider falsch geschrieben. Korrekturen wurden von demselben Autor auch veröffentlicht (PELLETIER, 1988; PELLETIER und SUTCLIFFE, 1997). Um die ganze Geschichte zusammenzufassen werden die Quellen für alle Tests in folgender Tabelle angezeigt.

Tabelle 1 — Quellen der Tests.

Quelle	Test Nummer
Pelletier 1986	1-14, 16-27, 29-33, 35-39, 41-53, 56-59, 60, 61, 63-68, 71
Pelletier 1988	15, 28, 34, 40, 54, 55
Pelletier und Sutcliffe 1997	62
Nicht verwendete Tests	69, 70, 72-75

Quelle: eigene Darstellung.

In folgender Tabelle stehen die Ergebnisse aller 67 implementierten Tests aus Pelletier (1986). In dieser Tabelle werden die Ergebnisse jedes einzelnen Problems dargestellt. Alle grafischen Darstellungen von Beweisen, die durch diese Tests erstellt wurden, werden auf einer Festplatte zusammen mit dieser Arbeit abgegeben.

Wenn ein Test erfolgreich ist, wird die Dauer der Berechnung angezeigt. Wenn der Test jedoch wegen mangelnder Zeit fehlgeschlagen ist, wird ein Z gezeigt. Alle Angaben erfolgen in Sekunden. Manche Tests können nicht von einem bestimmten Ansatz gelöst werden, weil es bspw. keine anwendbare Regel gibt. Diese Tests werden mit einem X markiert[32]. Wenn der Test aus irgendwelchem sonstigen Grund fehlschlägt (bspw. *Stack Overflow*), wird ein F gezeigt.

[32] Es ist eigentlich eine schwere Aufgabe, zu schließen, ob ein Problem von einem bestimmten unvollständigen Ansatz gelöst werden kann. Besonders, wenn es kein Beweis für die Unvollständigkeit gibt. Im Abschnitt 3.3 wurde die DFID vorgestellt. Dort wurde geschrieben, dass die Anwendungen der Regeln auf eine bestimmte Zahl k begrenzt werden. Hier wird angenommen, dass ein Ansatz ein Problem nicht lösen kann, sobald k = 10.

Tabelle 2 — Ergebnisse der Tests.

Problem	Smullyan	KE	Hypertableau	KNF Tableau
#1	1.884	1.922	1.385	1.793
#2	0.258	0.286	0.154	0.419
#3	0.172	0.178	0.139	0.126
#4	0.464	0.402	0.229	0.223
#5	0.508	0.381	0.192	0.287
#6	0.159	0.130	0.098	0.097
#7	0.182	0.208	0.065	0.080
#8	0.294	0.167	0.104	0.107
#9	0.822	0.337	0.233	0.502
#10	1.342	0.807	0.507	0.888
#11	0.296	0.237	0.071	0.076
#12	9.128	4.457	1.349	2.167
#13	1.477	1.776	0.350	0.700
#14	1.492	1.258	0.264	0.570
#15	0.664	0.481	0.101	0.293
#16	0.356	0.302	0.148	0.164
#17	2.556	1.074	0.464	1.024

#18	0.355	0.414	0.086	0.103
#19	Z	Z	0.117	0.283
#20	2.380	2.506	0.148	1.244
#21	10.917	Z	0.422	1.154
#22	1.510	0.780	0.475	Z
#23	0.783	0.636	1.343	Z
#24	Z	Z	0.786	1.494
#25	2.457	Z	Z	Z
#26	Z	Z	Z	Z
#27	Z	Z	Z	Z
#28	5.037	Z	Z	Z
#29	Z	Z	Z	Z
#30	1.870	3.864	Z	Z
#31	3.403	12.474	Z	4.735
#32	8.950	Z	Z	Z
#33	7.190	6.105	Z	5.559
#34	Z	Z	Z	Z
#35	0.769	1.260	0.347	0.869
#36	2.884	7.265	X	2.700

#37	Z	Z	X	Z
#38	Z	Z	Z	Z
#39	2.155	1.738	0.445	1.914
#40	Z	5.504	1.537	Z
#41	5.744	3.843	1.953	Z
#42	Z	Z	Z	Z
#43	Z	Z	X	Z
#44	9.245	Z	1.094	Z
#45	Z	Z	X	Z
#46	Z	Z	X	Z
#47	Z	Z	Z	Z
#48	5.294	6.442	2.342	4.550
#49	Z	Z	X	Z
#50	5.885	6.827	X	2.340
#51	Z	Z	Z	Z
#52	Z	Z	Z	Z
#53	Z	Z	Z	Z
#54	Z	Z	Z	Z
#55	Z	Z	Z	Z

#56	9.296	Z	Z	Z
#57	6.574	Z	Z	4.062
#58	Z	Z	Z	X
#59	Z	Z	Z	11.935
#60	6.876	Z	Z	Z
#61	Z	Z	Z	X
#62	Z	Z	Z	Z
#63	Z	Z	Z	Z
#64	Z	Z	Z	Z
#65	Z	Z	Z	Z
#66	Z	Z	Z	Z
#67	Z	Z	Z	Z
#68	Z	Z	Z	Z
AeT[33] 5 Sek.	26 / 68	25 / 68	30 / 68	29 / 68
AeT 1 Min.	38 / 68	31 / 68	30 / 68	31 / 68

Quelle: eigene Darstellung.

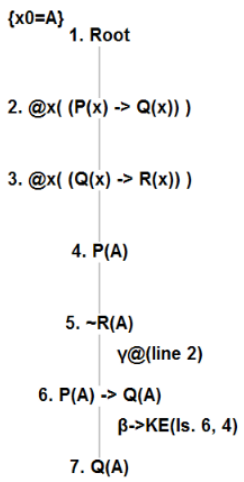
[33] Anzahl erfolgreicher Tests.

ANHANG B - BENUTZERHANDBUCH

Dieses Handbuch erklärt die Funktionen von dem Theorembeweiser. Trotz des "Benutzerhandbuch" Titels ist der Schwerpunkt hier nicht die Bedienung der Software, sondern die von der Software zur Verfügung gestellten Funktionen. Selbstverständlich wird nur das Tableau Verfahren hier vorgestellt.

Die Beweise werden grafisch wie Bäume dargestellt. Ein Beispiel ist die folgende Abbildung. Auf dem oberen linken Rand jedes Bildes wird die aktuelle Substitution gezeigt. In dem Beispiel ist die Substitution {A / x0}. Merken Sie, dass auch wenn das Tableau noch offen ist, wie das Tableau auf dem Bild, wird diesem Tableau eine Substitution zugeordnet. Außerdem wird die Substitution auf allen Variablen in dem Tableau angewandt. Bspw. wurde die freie "x0" Variable von Zeile 6 auf Abbildung 1 durch "A" ersetzt.

Abbildung 1 — Die grafische Darstellung.

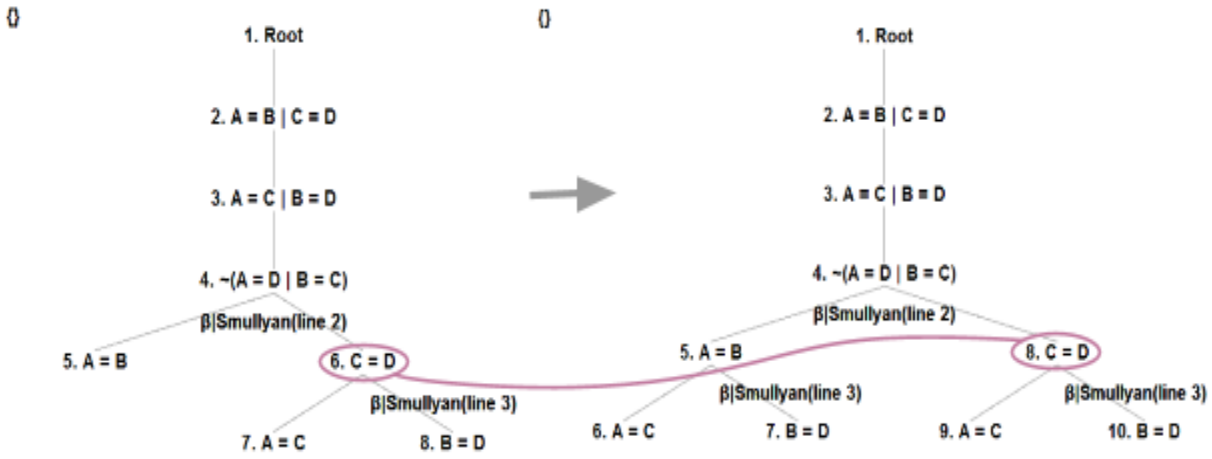


Quelle: eigene Darstellung.

Auf dem Bild erkennt man auch zwei weitere Besonderheiten von der grafischen Darstellung: Die Zeilen werden nummeriert und die angewandten Regeln angegeben. Bei der Nummerierung der Zeilen wird die Tiefensuche verwendet. Der Algorithmus wird für jeden Beweisbaum ausgeführt. Das führt zur Tatsache, dass sich die Zeilennummern von Formeln nach

einer Regelanwendung verändern können. In dem Beispiel bewirkt die Anwendung einer β-Regel auf Zeile 5, dass Zeile 6 zu Zeile 8 wird.

Abbildung 2 — Zeilennummerierung.



Quelle: Problem #48 aus Pelletier (1986) und eigene Darstellung.

Beispielen von Regelanwendung können sowohl auf der Abbildung 1 als auch auf der Abbildung 2 gesehen werden. Hinter dem Namen der Regel wird die Zeile angegeben, die diese Regel erstellt hat. Um eine Regel zu erstellen, müssen alle Prämissen von der Regel vorhanden sein. Wenn es mehr als eine Prämisse gibt, dann werden alle Zeilen angegeben. Ein Beispiel hierfür ist die Anwendung der β-Regel auf Abbildung 1. Jede Regel wird einen bestimmten Namen zugeordnet, der von den Formeln abhängt, die diese Regel erstellt haben. Die vollständige Tabelle der Namen ist die Folgende:

Tabelle 1 — Name der Schlussregeln.

Logik	**Schlussregel**	**Name**
Klassische Logik (*First-Order Form*)	$\dfrac{\varphi \wedge \psi}{\varphi, \psi}$	α&

Klassische Logik (*First-Order Form*)	$\dfrac{\neg(\varphi \vee \psi)}{\varphi, \psi}$	α~|
	$\dfrac{\neg(\varphi \rightarrow \psi)}{\varphi, \psi}$	α~->
	$\dfrac{\neg\neg\varphi}{\varphi}$	α~~
	$\dfrac{\varphi \leftrightarrow \psi}{\varphi \rightarrow \psi, \psi \rightarrow \varphi}$	α<->
	$\dfrac{\neg(\varphi \wedge \psi)}{\varphi \mid \psi}$	β~&Smullyan
	$\dfrac{\varphi \vee \psi}{\varphi \mid \psi}$	β|Smullyan
	$\dfrac{\varphi \rightarrow \psi}{\neg\varphi \mid \psi}$	β->Smullyan
	$\dfrac{\neg(\varphi \leftrightarrow \psi)}{\neg(\varphi \rightarrow \psi), \neg(\psi \rightarrow \varphi)}$	β~<->Smullyan
	$\dfrac{}{\varphi \mid \neg\varphi}$	Cut
	$\dfrac{\neg(\varphi \wedge \psi) \quad \neg\varphi}{\psi}$	β~&KE
	$\dfrac{\neg(\varphi \wedge \psi) \quad \neg\psi}{\varphi}$	

Klassische Logik (*First-Order Form*)	$\varphi \vee \psi$ $\neg\varphi$ ——— ψ	β|KE
	$\varphi \vee \psi$ $\neg\psi$ ——— φ	
	$\varphi \rightarrow \psi$ φ ——— ψ	β->KE
	$\varphi \rightarrow \psi$ $\neg\psi$ ——— φ	
	$\neg(\varphi \leftrightarrow \psi)$ $\varphi \rightarrow \psi$ ——— $\neg(\psi \rightarrow \varphi)$	β~<->KE
	$\neg(\varphi \leftrightarrow \psi)$ $\psi \rightarrow \varphi$ ——— $\neg(\varphi \rightarrow \psi)$	
	$\exists x. \varphi$ ——— $\{f(x_1, ..., x_n) / x\} \varphi$	δ#
	$\neg(\forall x. \varphi)$ ——— $\{f(x_1, ..., x_n) / x\} \varphi$	δ~@
	$\forall x. \varphi$ ——— $\{t / x\} \varphi$	γ@
	$\neg(\exists x. \varphi)$ ——— $\{t / x\} \varphi$	γ~#

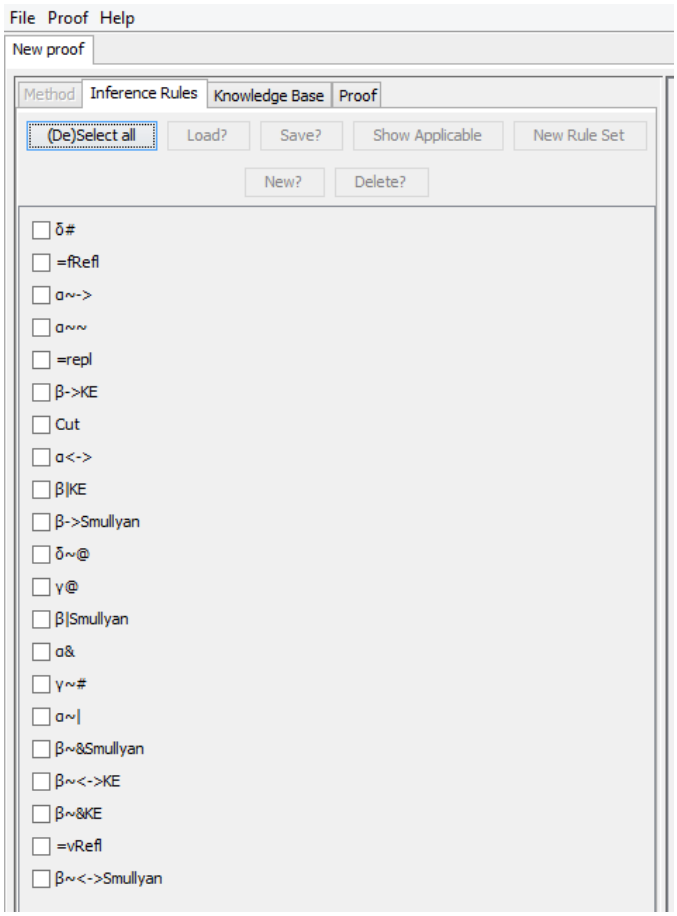
Klassische Logik (*First-Order Form*)	$\dfrac{}{x = x}$	=vRefl
	$\dfrac{}{f(x_1, ..., x_n) = f(x_1, ..., x_n)}$	=fRefl
	$\dfrac{t = u \quad \varphi(t)}{\varphi(u)}$	=repl
Klassische Logik (*Conjunctive Normal Form*)	$\dfrac{\kappa(x_1, ..., x_n)}{\kappa_1 \mid ... \mid \kappa_k}$	β
	$\dfrac{\kappa(x_1, ..., x_n)}{\sigma\kappa_1 \mid ... \mid \sigma\kappa_k \quad \sigma = \{t_1 / x_1, ..., t_n / x_n\}}$	γ
	$\dfrac{\{\varphi_1, ..., \varphi_k, \psi_{k+1}, ..., \psi_n\} \quad \psi_{k+1}^{c} \quad ... \quad \psi_n^{c}}{\varphi_1 \mid ... \mid \varphi_k}$	hyper
	$\dfrac{}{x = x}$	=CNFvRefl
	$\dfrac{}{f(x_1, ..., x_n) = f(x_1, ..., x_n)}$	=CNFfRefl
	$\dfrac{t = u \quad \kappa(x_1, ..., t, ..., x_n)}{\kappa(x_1, ..., u, ..., x_n)}$	=CNFrepl
ALC Beschreibungslogik	$\dfrac{(C_1 \sqcap C_2)(x)}{C_1(x) \quad C_2(x)}$	&
	$\dfrac{(C_1 \sqcup C_2)(x)}{C_1(x) \mid C_2(x)}$	U

ALC Beschreibungslogik	$\frac{(\exists r.C)(x)}{C(y)\ r(x,y)}$	#
	$\frac{(\forall r.C)(x)\ r(x,y)}{C(y)}$	@

Quelle: eigene Darstellung. Die Quelle der Regeln sind alle in Kapitel 3.

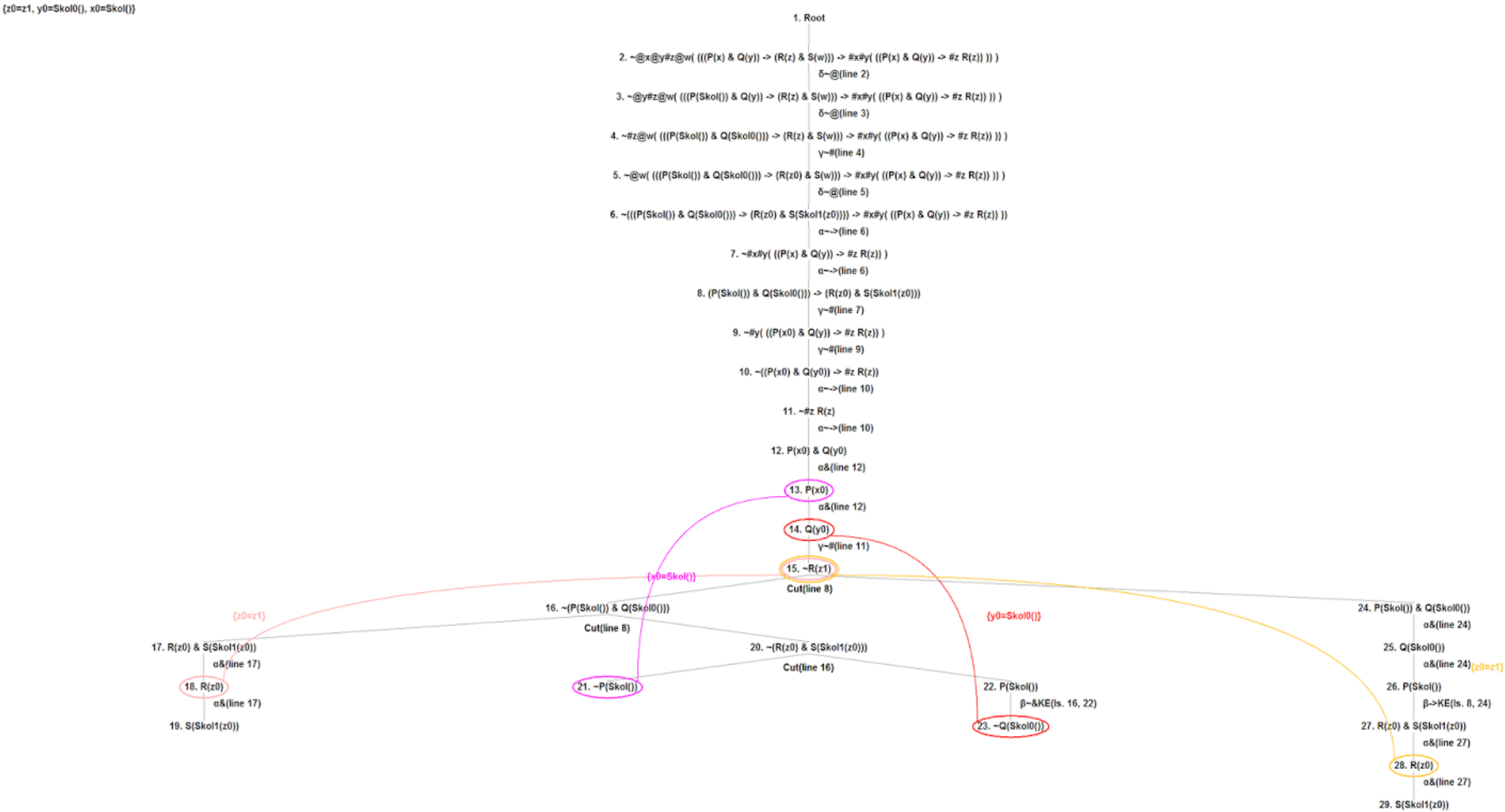
Die GUI erlaubt außerdem der Benutzer, eine Methode mit beliebigen Regeln zu gestalten.

Abbildung 3 — Einstellung der Verfahren



Quelle: eigene Darstellung

Schließlich sieht man auf Abbildung vier (nächste Seite) ein abgeschlossenes Tableau. Die Formeln, die unifizieren, werden angezeigt. Darüber hinaus wird auch die Untermenge der Substitution gezeigt, die für jede Unifikation notwendig war.


{z0=z1, y0=Skol0(), x0=Skol()}
1. Root
2. ~@x@y#z@w((((P(x) & Q(y)) -> (R(z) & S(w))) -> #x#y(((P(x) & Q(y)) -> #z R(z)))))
δ~@(line 2)
3. ~@y#z@w((((P(Skol()) & Q(y)) -> (R(z) & S(w))) -> #x#y(((P(x) & Q(y)) -> #z R(z)))))
δ~@(line 3)
4. ~#z@w((((P(Skol()) & Q(Skol0())) -> (R(z) & S(w))) -> #x#y(((P(x) & Q(y)) -> #z R(z)))))
γ~#(line 4)
5. ~@w((((P(Skol()) & Q(Skol0())) -> (R(z0) & S(w))) -> #x#y(((P(x) & Q(y)) -> #z R(z)))))
δ~@(line 5)
6. ~(((P(Skol()) & Q(Skol0())) -> (R(z0) & S(Skol1(z0)))) -> #x#y(((P(x) & Q(y)) -> #z R(z))))
α~->(line 6)
7. ~#x#y(((P(x) & Q(y)) -> #z R(z)))
α~->(line 6)
8. (P(Skol()) & Q(Skol0())) -> (R(z0) & S(Skol1(z0)))
γ~#(line 7)
9. ~#y(((P(x0) & Q(y)) -> #z R(z)))
γ~#(line 9)
10. ~((P(x0) & Q(y0)) -> #z R(z))
α~->(line 10)
11. ~#z R(z)
α~->(line 10)
12. P(x0) & Q(y0)
α&(line 12)
13. P(x0)
α&(line 12)
14. Q(y0)
γ~#(line 11)
15. ~R(z1)
Cut(line 8)
{x0=Skol()}
{z0=z1}
16. ~(P(Skol()) & Q(Skol0()))
Cut(line 8)
{y0=Skol0()}
24. P(Skol()) & Q(Skol0())
α&(line 24)
17. R(z0) & S(Skol1(z0))
α&(line 17)
20. ~(R(z0) & S(Skol1(z0)))
Cut(line 16)
25. Q(Skol0())
α&(line 24)
{z0=z1}
18. R(z0)
α&(line 17)
21. ~P(Skol())
22. P(Skol())
β~&KE(ls. 16, 22)
26. P(Skol())
β->KE(ls. 8, 24)
19. S(Skol1(z0))
23. ~Q(Skol0())
27. R(z0) & S(Skol1(z0))
α&(line 27)
28. R(z0)
α&(line 27)
29. S(Skol1(z0))