Hochschule Mainz

Bachelorarbeit

Entwickeln einer Anwendung zur Erstellung und Visualisierung von Wahrheitsbäumen

Lars Bodewig

Eingereicht am:

11.10.2020

Management Summary

Wahrheitsbäume sind eine effiziente Methode, um die Erfüllbarkeit einer logischen Aussage zu prüfen und deren Lösungsmenge zu bestimmen. Dafür werden die Transformationsregeln zur Zerlegung von Konjunktionen, Disjunktionen und Subjunktionen festgelegt. Unter anderem werden Wahrheitsbäume zur logischen Beweisführung in der Mathematik und den Naturwissenschaft verwendet. Um die Methode auf eine selbst erstellte Anwendung übertragen zu können, wird ein Konzept definiert und beschrieben. Die Eingabe der logischen Aussage per Kommandozeile muss mittels einer Grammatik validiert und analysiert werden. Zur Abbildung der Daten werden passende Datenstrukturen erstellt. Auf Basis der Methode wird ein Algorithmus entwickelt, der die Schritte zur Erstellung des Wahrheitsbaumes aus dem erstellten Datentyp durchführt. Abschließend wird der erzeugte Wahrheitsbaum in einer grafischen Benutzeroberfläche visualisiert und kann als Datei exportiert werden. Die Arbeit zeigt die erfolgreiche Umsetzung einer Methode einer computerfremden Domäne in einem computergestützten Umfeld.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

$\neg$	Negation
$\wedge$	Konjunktion
$\vee$	Disjunktion
$\rightarrow$	Subjunktion
$\triangleq$	Entsprechung
$=$	Gleichheit
$:=$	Definition
$\forall$	Allquantor
$\models$	Semantische Folgerung
$\therefore$	Deduktion
$\top$	Wahr
$\subset$	echte Teilmenge
$\subseteq$	Teilmenge/echte Teilmenge
$\cup$	Vereinigung
$\in, \ni$	Element von
ε	Leeres Wort
$\lvert \ldots \rvert$	Betrag
$\{\ldots\}$	Menge
$(\ldots)$	Tupel

Formelverzeichnis

1 Einleitung

Wahrheitsbäume sind eine Methode des Fachgebiets der Aussagenlogik, einem Teilbereich der Mathematik und Logik. Sie ermöglichen die Erfüllbarkeit einer logischen Aussage zu beweisen mittels einer einfachen grafischen Aufschlüsselung der Aussage in kleinere Terme. Hauptbestandteil ist eine Menge von Transformationsregeln, die die rekursive Ableitung kleinerer Terme ermöglicht, bis ein Widerspruch gefunden wurde oder keine weitere Transformation möglich ist. So kann man logische Aussagen auf ihre Validität überprüfen und ihre Erfüllbarkeit beweisen bzw. widerlegen. Zusätzlich kann die Menge aller validen Kombinationen zur Erfüllung der Aussage bestimmt werden. Dabei ist der Aufwand geringer als bei der Erstellung einer vollständigen Wahrheitstabelle, da nicht erfüllbare Terme früher erkannt und ausgeschlossen werden können. Die Verwendung als Beweismethode führte zur Erfindung verschiedener Automated Theorem Prover und ermöglicht beispielsweise die Verwendung beim Design und Testen von Prozessorschaltkreisen.

1.1 Forschungsfrage

Da die Erstellung eines Wahrheitsbaumes eine händische Methode ist, soll im Rahmen dieser Arbeit die Frage beantwortet werden, wie man eine Anwendung erstellt, die Wahrheitsbäume erzeugen und visualisieren kann. Hierfür soll insbesondere beantwortet werden, wie man die Eingabe durch den Nutzer realisiert und analysiert, wie man die Eingabe als Datentyp abbildet und wie dieser mittels eines geeigneten Algorithmus verarbeitet werden kann. Ziel der Arbeit ist ein Konzept für eine Anwendung zu entwickeln und praktisch umzusetzen.

1.2 Aufbau der Arbeit

Die Arbeit stellt die Theorie der Methode und das Konzept der Anwendung gegenüber. Dafür wird die Herleitung der Methode beschrieben und die theoretische Vorgehensweise mit Beispielen verdeutlicht. Anschließend folgt die Darstellung des Konzepts zur Übertragung der Methode in eine praktische Anwendung. Dies umfasst die Herleitung einer Grammatik zur Analyse der Nutzereingabe, die Definition von Datentypen zur Abbildung bestimmter Objekte, den Algorithmus zur Erstellung des Wahrheitsbaumes und die Beschreibung der grafischen Ausgabe. Abschließend wird die Forschungsfrage zusammenfassend beantwortet.

2 Methode

Wahrheitsbäume (auch Baumkalküle, Tableaukalküle oder Beth-Kalküle genannt) wurden um 1955 vom niederländischen Logiker und Philosophen Evert Willem Beth erfunden [18]. Sein Ziel war die Anwendbarkeit der klassischen Logik auf wissenschaftliche, insbesondere mathematische Fragestellungen zu erweitern [5] und brachte dabei das in diesem Kapitel vorgestellte Verfahren zum Beweisen der Erfüllbarkeit einer logischen Aussage hervor [4]. Beth gilt als Vorreiter, da zu seiner Zeit die Logik noch keine anerkannte akademische Disziplin darstellte, aber er war überzeugt von der fundamentalen Wichtigkeit der Logik als Grundlage für die Mathematik und die Naturwissenschaften [5]. Als Mitglied einer Forschungsgruppe entwickelte er ebenfalls Interesse Theoreme computergestützt zu beweisen, starb jedoch bereits kurze Zeit später im Jahr 1964 [18]. Im Folgenden soll der mathematisch-philosophische Hintergrund zur Entstehung der Methode erläutert, die verschiedenen Anwendungsfälle aufgezeigt und die logische Beweisbarkeit hergeleitet werden. Anschließend wird eine Menge von Regeln zur Transformation von Konjunktionen, Disjunktionen und Subjunktionen definiert und die Erstellung eines Wahrheitsbaumes theoretisch und an Beispielen beschrieben.

2.1 Intuitionismus

Intuitionismus ist eine philosophisch-mathematische Position, die zur Strömung des mathematischen Konstruktivismus zählt [9]. Sie wurde zu Anfang des 20. Jahrhunderts durch den niederländischen Mathematiker L. E. J. Brouwer begründet [9]. Der mathematische Konstruktivismus wendet sich dabei von der Betrachtung der Mathematik als objektive Wahrheit ab und ordnet ihren Konstrukten den Kontext des menschlichen Denkens zu [9]. So ergibt sich auch ein anderes Verständnis der Logik, das von der klassischen Logik abzugrenzen ist. Während die klassische Logik einer Aussage einen Wahrheitswert (wahr/falsch) zumisst, beschäftigt sich die intuitionistische Logik mit der Beweisbarkeit [2]. Die Darstellung gleicht dabei, die Interpretation jedoch ist eine andere.

Klassische Logik: $A \wedge B \; ist \; eine \; Wahrheitsfunktion, beschrieben \; durch$
$"A \; und \; B \; sind \; wahr"$

Intuitionistische Logik: $A \wedge B \; wird \; interpretiert \; als$
$"A \; und \; B \; können \; bewiesen \; werden"$

L. E. J. Brouwer hatte bereits 1920 durch sein Fan-Theorem den Kompaktheitssatz (auch Endlichkeitssatz genannt) der klassischen Logik in der intuitionistischen Logik erfüllt [17]. Dieser besagt, dass jede endliche oder unendliche Formelmenge erfüllbar ist, wenn jede endliche

Teilmenge erfüllbar ist bzw. jede endliche oder unendliche Formelmenge unerfüllbar ist, wenn eine endliche Teilmenge unerfüllbar ist [7]. Beth glaubte auf Basis dessen durch ein geeignetes Verfahren die klassische Logik in der intuitionistischen Logik verwenden zu können, um so Fragestellungen der konstruktiven Mathematik und Physik intuitionistisch zu beantworten [17].

2.2 Anwendungsfall

Wahrheitsbäume können in verschiedenen Anwendungsfällen zum Einsatz kommen. Einerseits kann die Existenz oder Nicht-Existenz einer Lösung für eine Aussage oder einer Menge von mehreren Aussagen bewiesen werden. Dies ist möglich, da eine Menge von Aussagen, bei der jede Aussage gleichzeitig erfüllt sein muss einer Konjunktion der einzelnen Aussagen entspricht. [4]

$$\{A, B, C\} \triangleq A \wedge B \wedge C$$

Formel 1: Konjunktion der Elemente

Andererseits kann, ähnlich einer Wahrheitstabelle, die Lösungsmenge der validen Variablenkombinationen bestimmt werden [6]. Um die vollständige Lösungsmenge zu erhalten muss die Anwendung der Transformationsregeln unter Berücksichtigung der Operatorpräzedenz erfolgen, andernfalls erhält man unter Umständen nur eine Teilemenge[1]. Ein Wahrheitsbaum könnte somit für eine Aussage F verschiedene Teilmengen L' der Lösungsmenge L ergeben:

$$F := A \vee B \wedge \neg B$$
$$L := \{\{A, \neg B\}, \{A, B\}\}$$

Der erste Fall erlaubt eine logische Schlussfolgerung auf ihre Validität zu prüfen. Hierfür wird ein Widerspruchsbeweis verwendet, um die Existenz einer Kombination von Wahrheitswerten für die verwendeten Variablen zur Erfüllung aller Prämissen, bei gleichzeitiger Nicht-Erfüllung der Konklusion zu widerlegen. [1]

$$\textit{Aus einer Menge von Prämissen } P \textit{ folgt eine Konklusion } k,$$
$$\textit{wenn es keinen Fall gibt, in dem die Prämissen erfüllt sind, die Konklusion jedoch nicht.}$$

$$\neg(\forall P \wedge \neg k) \vDash \forall P \therefore k$$

Formel 2: Widerspruchsbeweis der Konklusion

[1] Siehe Beweis der unvollständigen Lösungsmenge im Anhang

Somit dienen Wahrheitsbäume nicht nur dem händischen Test einer Aussage auf Erfüllbarkeit, sondern führten auch zur Entstehung verschiedener Projekte, um mathematische Beweise in einer computergestützten Umgebung zu rekonstruieren und Theoreme zu prüfen – ein möglicher Anwendungsfall, mit dem sich bereits eine Forschungsgruppe um Beth in den 1960er Jahren beschäftigte [5]. Sogenannte Automated Theorem Prover kommen auch beim Designen und Prüfen von elektronischen Schaltkreisen in Prozessoren zum Einsatz, um auf mögliche Fehlkonstruktion zu testen [8][15]. Beispiele hierfür sind das Mizar Project[2], Isabelle[3] und HOL[4].

2.3 Reductio ad absurdum

Ein Widerspruchsbeweis, auch Reductio ad absurdum genannt, bezeichnet eine Form der Beweisführung, bei der eine zur Hypothese gegenteilige Annahme aufgestellt und widerlegt wird. Ein bekannter Anwendungsfall ist Euklids Beweis der Irrationalität der Wurzel aus 2 [19].

Diese Beweisform basiert auf verschiedenen Axiomen des Systems der Aussagenlogik. Grundsätzlich existieren zwei Wahrheitswert, die eine Variable oder Aussage annehmen kann: ‚wahr' und ‚falsch'. Dies wird das Prinzip der Zweiwertigkeit genannt. Der Satz vom ausgeschlossenen Dritten legt zugrunde, dass eine Variable oder Aussage dabei nicht gleichzeitig einen Wert und dessen Gegenteil abbilden kann. Zuletzt trifft der Satz vom Widerspruch die Annahme, dass zwei widersprüchliche Aussagen nicht gleichzeitig zutreffen können. Aus der Widerlegung der Nichterfüllbarkeit der logischen Aussage folgt also die Erfüllbarkeit der Aussage, da nach dem Gesetz der doppelten Negation ein doppelt verneinter Satz denselben Wahrheitswert hat, wie der entsprechende unverneinte Satz. [3]

$$A \vee \neg A$$

Formel 3: Satz vom ausgeschlossenen Dritten

$$\neg(A \wedge \neg A)$$

Formel 4: Satz vom Widerspruch

$$\neg\neg A = A$$

Formel 5: Gesetz der doppelten Negation

[2] Siehe http://mizar.org/project/
[3] Siehe https://isabelle.in.tum.de/
[4] Siehe https://hol-theorem-prover.org/

Da das Gesetz der doppelten Negation aufgrund der Nicht-Haltbarkeit des Satzes vom ausgeschlossenen Dritten nicht in der intuitionistischen Logik gilt [9], bedarf es aber einem weiteren Satz, um diese Beweisform anwendbar zu machen: der Doppelten-Negations-Beseitigung.
V. Glivenko gelang es 1929 durch das nach ihm benannte Glivenko-Theorem die notwendige Eigenschaft der klassischen Logik in die intuitionistische Logik zu übertragen [11]. Durch die Anwendung im Beth-Kalkül lässt sich die Erfüllbarkeit einer logischen Aussage oder einer Aussagenmenge per Widerspruch beweisen bzw. widerlegen und dadurch die Konklusion einer logischen Argumentation validieren.

2.4 Transformationsregeln

Ein zentraler Aspekt von Beths Erfindung war die Definition einer Menge von Transformationsregeln, die die Spaltung der Ausgangsaussage(n) ermöglicht. Eine Transformationsregel bezeichnet dabei eine Funktion, die einen Term in eine Menge kleinerer Terme transformiert. Grundsätzlich sind dabei zwei Arten zu unterscheiden: lineare Transformationen und verzweigende Transformationen. Diese leiten sich aus den logischen Verknüpfungsarten ab, auf die sich die jeweilige Regel bezieht. Sowohl lineare als auch verzweigende Transformationen können in mehreren Termen resultieren. Lineare Transformationen implizieren, dass alle resultierenden Terme erfüllt sein müssen, damit der Eingabe-Term erfüllt ist. Verzweigende Transformationen zeigen an, dass nur ein resultierender Zweig diese Prämisse erfüllen muss. [1]
Da die Transformation die Grundlage der namensgebenden Baumstruktur ist, werden Eingabe-Aussage und Ausgabe-Terme im Folgenden auch als Vater- und Kind-Knoten betrachtet. Dies verdeutlicht die entsprechende Abbildung der Transformation als Baum. Im Rahmen dieser Arbeit werden nur einige ausgewählte Verknüpfungen betrachtet, jedoch lässt sich die Menge der Regeln zur Transformation beliebig für weitere logische Verknüpfungen erweitern.

2.4.1 Konjunktion

Die Konjunktion (häufig auch Und-Verknüpfung) lässt mittels der Konjunktionsbeseitigung auf den Wahrheitswert der verknüpften Terme schließen [10]. Die Aussage kann nur wahr sein, wenn beide Terme zu ‚wahr' evaluieren, daher handelt es sich um eine lineare Transformation, aus der mehrere Kind-Terme resultieren. Die Konjunktion hat die höchste Operatorpräzedenz. [12]

$$A \wedge B = \{A, B\}$$

Abbildung 1: Transformation der Konjunktion

Bei der negierten Konjunktion (auch Shefferscher Strich genannt) ergibt sich aus der Wahrheitstabelle eine verzweigende Transformation, da nicht klar ableitbar ist, welcher Term welchen Wert hält, wenn die Aussage falsch ist.

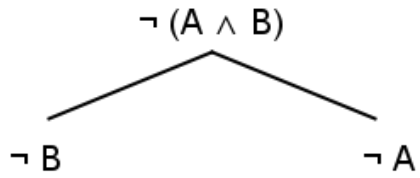


Abbildung 2: Transformation des Shefferschen Strich

2.4.2 Disjunktion

Invers zur Konjunktion gibt die Disjunktion (häufig auch Oder-Verknüpfung) nur Aufschluss über die Werte der verknüpften Terme bei ihrer Negation (auch Peirce-Funktion genannt). Bei Negation resultiert daher eine lineare Transformation, ohne Negation eine verzweigende. Die Disjunktion hat die zweithöchste Operatorpräzedenz nach der Konjunktion.

$$\neg(A \vee B) = \{\neg A, \neg B\}$$

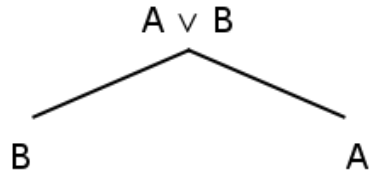


Abbildung 3: Transformation der Disjunktion

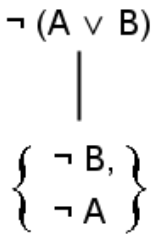


Abbildung 4: Transformation der Peirce-Funktion

2.4.3 Subjunktion

Zuletzt soll die Subjunktion (auch Konditional oder materielle Implikation) betrachtet werden. Die Wahrheitstabelle verdeutlicht die sprachliche Interpretation des Operators als "Wenn A, dann B", wobei diese Interpretation eine Kommutativität impliziert, die allerdings nicht existiert (vgl. Wahrheitstabelle der Subjunktion) [16]. Die Implikation ist äquivalent zur Disjunktion mit Eingangsnegation und bedingt daher eine verzweigende Transformation, dennoch verfügt sie über eine eigene Operatorpräzedenz, die niedriger als die der Disjunktion ist.

$$A \rightarrow B = \neg A \vee B$$

A	B	$A \rightarrow B$
F	F	T
F	T	T
T	F	F
T	T	T

Tabelle 1: Wahrheitstabelle der Subjunktion

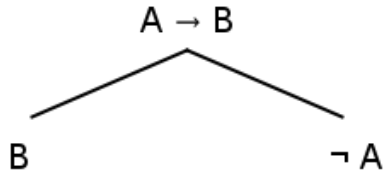


Abbildung 5: Transformation der Subjunktion

Aus der Negation der Subjunktion ergibt sich die Inhibition, die zur Konjunktion mit Eingangsnegation äquivalent ist. Wie bei der Konjunktion resultiert eine lineare Transformation in zwei Kind-Terme.

$$\neg(A \rightarrow B) = A \wedge \neg B$$

A	B	$\neg(A \rightarrow B)$
F	F	F
F	T	F
T	F	T
T	T	F

Tabelle 2: Wahrheitstabelle der Inhibition

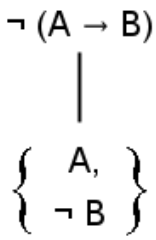


Abbildung 6: Transformation der Inhibition

2.5 Erstellung des Wahrheitsbaumes

Mithilfe dieser Transformationsregeln lässt sich nun eine Baumstruktur aus einer Eingabe-Menge von Aussagen erstellen. Dabei stellt jeder Knoten des Baumes immer eine Menge von einer oder mehreren Aussagen dar. Eine verzweigende Transformation erzeugt also mehrere Kind-Knoten mit jeweils einem Term, während eine lineare Transformation nur einen Kind-Knoten mit mehreren Termen erzeugt.

Um den Wahrheitsbaum zu erstellen, definiert man zuerst die Eingabe-Menge der Aussagen F und die Variablen V. Die Menge der Regeln sei als R definiert.

$$V := \{v_1, v_2, \dots v_n\}$$
$$F := \{f_1, f_2, \dots f_n\}$$
$$R := \{r_1, r_2, \dots r_n\}$$

Soll eine Konklusion k auf Basis der Prämissen einer Menge P bewiesen werden, umfasst F alle Elemente der Menge P und die negierte Konklusion.

$$P := \{p_1, p_2, \dots p_n\}$$
$$F := \{p_1, p_2, \dots p_n, \neg k\}$$

Die Aussagen-Menge F dient als Wurzel unseres Baumes und kann entsprechend notiert werden.

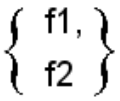


Abbildung 7: Baum-Wurzel bestehend aus F

Jede Aussage f muss nun mittels der Transformationsregeln in kleinere Terme zerlegt werden. Soll die Lösungsmenge der Aussagen-Menge F bestimmt werden, muss die Anwendung der Transformationsregeln in umgekehrter Reihenfolge der Operatorpräzedenz erfolgen, andernfalls ist die Wahl der anzuwendenden Regel nicht deterministisch und lässt unterschiedliche Lösungswege zu[5]. Je Aussage kann nur eine Transformationsregel angewendet werden, die Wahl der Regel kann innerhalb eines Knotens variieren. Im Folgenden wird eine allgemeine, verzweigende Regel r_1 auf f_1 angewendet und resultiert in den Termen t_1 und t_2, die an die Wurzel angehangen werden.

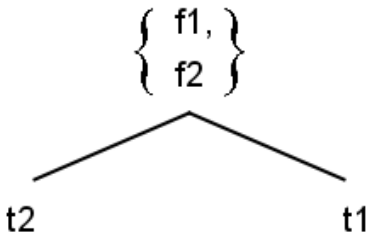


Abbildung 8: Allgemeine verzweigende Transformation

Da jede Aussage f mehr als einen Term umfassen sollte – andernfalls lässt sich die Variable durch den direkten Wahrheitswert ersetzen – ist noch keine Prüfung auf einen Widerspruch zwischen t_1 und f_2 bzw. t_2 und f_2 notwendig. Es muss eine weitere, hier beispielsweise lineare Regel r_2 auf f_2 angewendet werden. Das Ergebnis von r_2, die Terme t_3 und t_4, wird allerdings nicht unter der Wurzel, sondern an alle Blätter, die nicht zu einem geschlossen Zweig gehören, angehangen.

[5] Siehe Beweis der unvollständigen Lösungsmenge im Anhang

$$falls\ V = \{A, B\}\ und\ F = \{A \wedge B, B\},$$
$$entspricht\ F = \{A \wedge \top, \top\}\ und\ lässt\ sich\ zu\ F = \{A\}\ reduzieren,$$
$$daraus\ folgt, dass\ f_2\ mehr\ als\ eine\ Variable\ darstellt$$

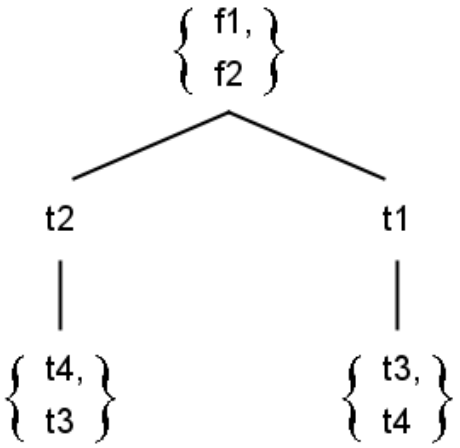


Abbildung 9: Lineare Transformation über mehrere Ebenen

In diesem Beispiel könnte nun der erste Widerspruch zwischen Termen des jeweiligen Zweiges auftreten. Liegt ein Widerspruch vor, muss dieser Zweig geschlossen werden und wird zur Lösungsfindung nicht weiter berücksichtigt. Enthält ein Zweig keine transformierbaren Terme mehr sondern nur noch widerspruchsfreie Variablen, wird er geschlossen und die Lösungsmenge L um die aufgeführte Kombination von Variablen samt vorhandener Negationen erweitert – enthält der Zweig dabei eine Variable nicht, ist diese zur Lösung der Aussagen-Menge F beliebig.

$$falls\ V = \{A, B, C, D\}, t_2 = A, t_3 = B\ und\ t_4 = \neg C,$$
$$erfüllen\ sowohl\ \{A, B, \neg C, D\}\ als\ auch\ \{A, B, \neg C, \neg D\}\ F,$$
$$D\ ist\ daher\ beliebig$$

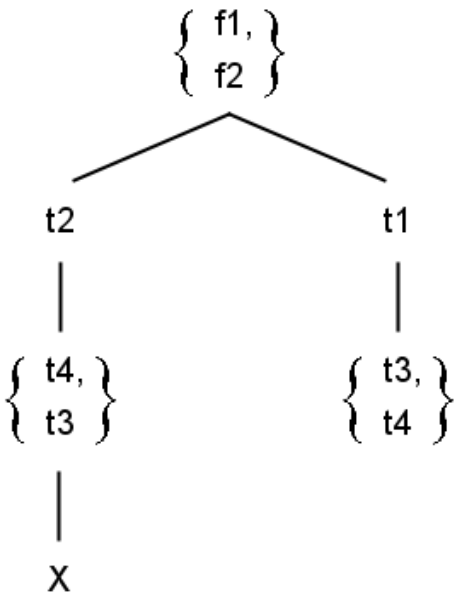


Abbildung 10: Darstellung eines geschlossenen Zweiges mit Widerspruch

$$L = \{\{v_1, v_2, v_3\}\}$$

Sind alle Zweige geschlossen stellt die Lösungsmenge L alle oder eine Teilmenge aller möglichen Kombinationen zur Erfüllung von F an. Ist F die Menge der Prämissen und der negierten Konklusion $\{p_1, p_2, \ldots p_n, \neg k\}$ und die Lösungsmenge L leer, wurde k bewiesen – unabhängig von der Reihenfolge der angewendeten Transformationsregeln –, andernfalls widerlegt.

$$falls\ F = \{p_1, p_2, \ldots p_n, \neg k\}\ und\ |L| > 0, ist\ k\ ung\ddot{u}ltig$$

$$sei\ V\ die\ Menge\ aller\ m\ddot{o}glichen\ Kombinationen\ zur\ Erf\ddot{u}llung\ von\ F,$$
$$ist\ L\ gleich\ V\ bei\ Beachtung\ der\ Operatorpr\ddot{a}zedenz,$$
$$andernfalls\ ist\ L\ eine\ Teilmenge\ oder\ echte\ Teilmenge\ von\ V:$$
$$L \subseteq V$$

2.6 Beispiele

Zur Veranschaulichung der Methode dienen nun folgende Beispiele. Als erstes Beispiel soll die allgemeine Erfüllbarkeit einer einfachen Aussage f_1 bewiesen werden. Die Variablen-Menge sei V.

$$V := \{A, B, C\}$$

$$f_1 := A \vee \neg B \wedge B \wedge C$$

$$F := f_1$$

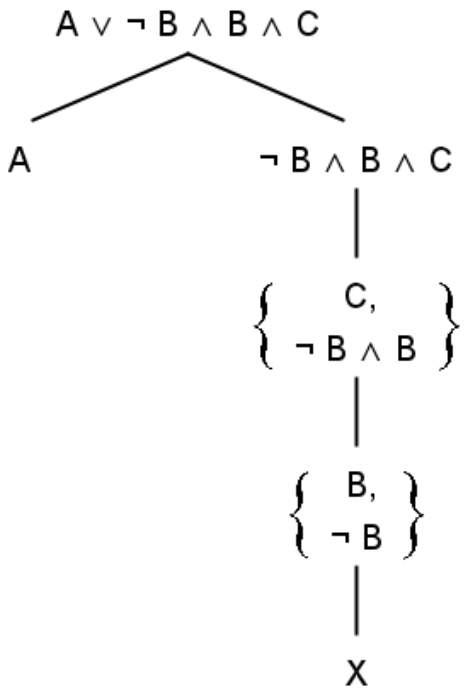


Abbildung 11: Beispiel zur Erstellung eines Wahrheitsbaumes für V

$$L = \{\{A\}\}$$

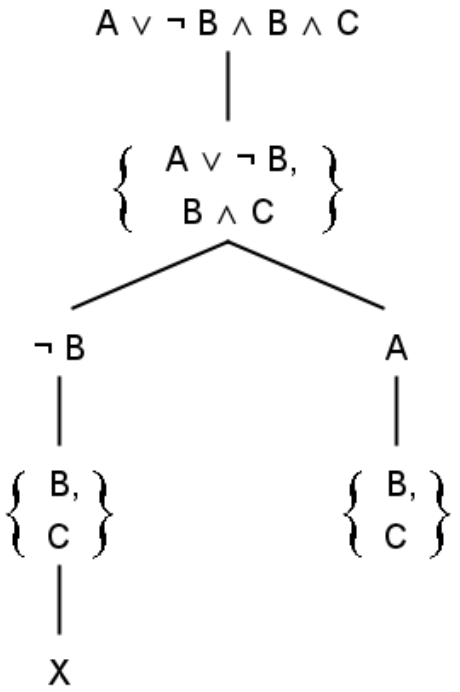


Abbildung 12: Wahrheitsbaum bei Verletzung der Operatorpräzedenz

$$L_2 = \{\{A, B, C\}\}$$

Da die Erstellung des Wahrheitsbaumes nicht deterministisch ist, führen in diesem Fall zwei verschiedene Lösungswege zu den unterschiedlichen Lösungsmengen L_1 und L_2. L_1 stellt dabei die vollständige Lösungsmenge dar, da die Transformationsregeln in invertierter Operatorrangfolge angewendet wurden – L_2 ist eine echte Teilmenge von L_1.

$$L_2 \subset L_1$$

Wie in den vorherigen Kapiteln beschrieben, lässt sich durch einen Wahrheitsbaum auch die Konklusion einer logischen Argumentation beweisen. Im folgenden Beispiel soll daher die Konklusion k bewiesen werden, die besagt, dass L_1 die vollständige Lösungsmenge der vorherigen Aussagen-Menge F ist. F stellt dabei die Prämissen dieses Arguments dar.

$$P := \{A \vee \neg B \wedge B \wedge C\}$$

$$k := A$$

$$F := P \cup \{\neg k\} = \{A \vee \neg B \wedge B \wedge C, \neg A\}$$

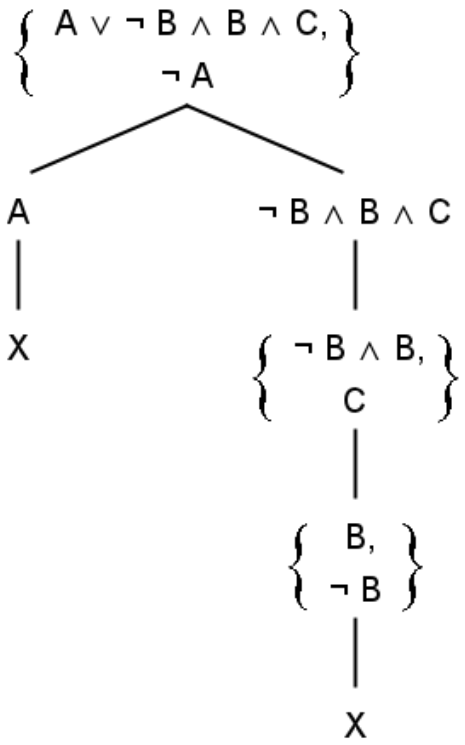


Abbildung 13: Widerspruchsbeweis mittels Wahrheitsbaum

$$L = \{\}, |L| = 0$$

Alle Zweige des Wahrheitsbaumes resultieren in einem Widerspruch, sodass die Lösungsmenge L leer ist. Die Aussagen-Menge F ist also nie erfüllbar, wodurch die Konklusion k per Widerspruch bewiesen wurde.

3 Implementierung

Auf Basis der Beschreibung der zugrunde liegenden Methodik wird folgend das Konzept und die Implementierung der Anwendung zur Automatisierung des Vorgehens beschrieben.
Dabei wird auf die Architektur, die Analyse der Nutzereingabe, das Design der Datenstrukturen, die Erstellung des Wahrheitsbaumes und die grafische Ausgabe eingegangen.

3.1 Architektur

Die Architektur lässt sich in vier logisch trennbare Komponenten abgrenzen: Die Anwendung wird über die Kommandozeile gestartet und kann mit bestimmten Optionen parametrisiert werden, dafür wird die Nutzereingabe zuerst mithilfe einer Bibliothek interpretiert. Die daraus extrahierte Eingabe von logischen Aussagen wird analysiert und in eine passende Datenstruktur übersetzt, die die Transformation zur Bildung des Wahrheitsbaumes ermöglicht. Der erzeugte Wahrheitsbaum wird, als Grafik aufbereitet, ausgegeben und die Anwendung terminiert.

Eine Übersicht kann der folgenden Grafik entnommen werden.

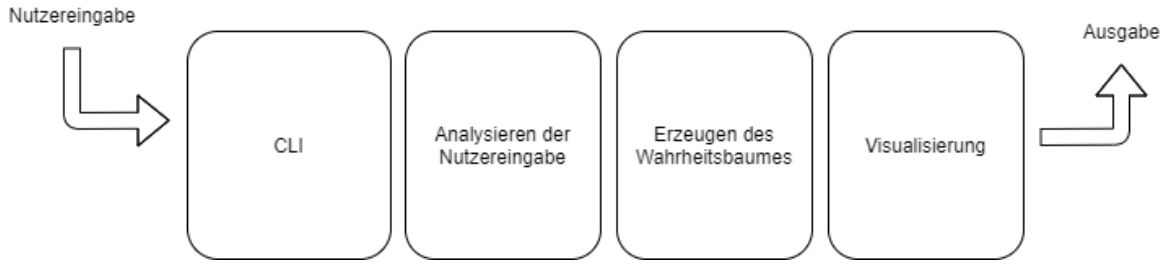


Abbildung 14: Anwendungsarchitektur bestehend aus vier Teilkomponenten

3.2 Analyse der Nutzereingabe

Bei der Implementierung der Nutzereingabe wurde sich für eine Freitexteingabe entschieden, um dem Nutzer die Eingabe von langen Ausdrücken bzw. großer Ausdruck-Mengen zu vereinfachen und den Zeitaufwand zur Entwicklung einer Oberfläche zu minimieren. Da die logischen Aussagen dabei einer gewissen Grammatik unterliegen müssen, muss die Nutzereingabe auf ihre formale Syntax hin überprüft, in ihre Worte zerlegt und in semantische Satzobjekte übersetzt werden. Hierfür wurde eine eigene Grammatik erstellt, die mittels der Programmbibliothek ANTLR[6] zur Anwendung auf die Nutzereingabe verwendet wird. Die Freitexteingabe durch den Nutzer erfolgt als Aufruf-Argument der Anwendung über die Kommandozeile.

[6] Siehe https://www.antlr.org/

3.2.1 Grammatik

Im Folgenden werden die Komponenten der Grammatik nach Noam Chomsky definiert und die Wahl der Produktionsregeln erläutert. G sei dabei die Grammatik bestehend aus den Nicht-Terminalen N, dem Terminalalphabet T, den Produktionsregeln P und dem Start-Symbol S. [14]

$$G = (N, T, P, S)$$
$$S \in N$$

Da ein Wahrheitsbaum aus einer einzelnen oder einer Menge von logischen Aussagen erstellt werden kann, ist die Abgrenzung mehrerer Aussagen durch mindestens ein Terminal notwendig. Um die Nutzerfreundlichkeit zu erhöhen, wurden zwei Terminale zur Abgrenzung und zwei optionale Terminale zur Notation einer Menge gesetzt. Zur Vereinfachung werden Klassen von mehreren Terminalsymbolen hier zum selben Terminal zusammengefasst, um die Darstellung der Produktionsregeln innerhalb dieser Arbeit überschaubar zu halten.

$$
\begin{aligned}
N \ni \quad & statements\_set, statements, statements \\
S := \quad & statements\_set \\
T \ni \quad & STATEMENT\_DELIMIT := [\,;\,], \\
& STATEMENTS\_OPEN := \{\,, \\
& STATEMENTS\_CLOSE := \} \\
P \ni \quad & statements\_set \rightarrow STATEMENTS\_OPEN\ statements\ STATEMENTS\_CLOSE \\
& \quad | \ statements \\
& statements \rightarrow statements\ STATEMENT\_DELIMIT\ statement \\
& \quad | \ statement
\end{aligned}
$$

Ein einzelne logische Aussage besteht aus einer oder mehreren Variablen, optional negiert und in Termen verschachtelt. Die Benennung der Variablen wurde dabei nur auf alphanummerische Zeichenkombination, beginnend mit einem Buchstaben, beschränkt, um dem Nutzer möglichst viel Freiheit bei der Eingabe zu lassen.

$N \ni$ $variable, alpha, digit$

$T \ni$ $DIGIT \coloneqq$ [0-9],

$ALPHA \coloneqq$ [a-zA-Z]

$P \ni$ $variable \rightarrow$ $variable\ DIGIT$

| $variable\ ALPHA$

| $ALPHA$

Wieder zur Vereinfachung der Eingabe wurde neben den in dieser Arbeit verwendeten Symbolen für die logischen Operatoren auch deren gängige Interpretation der englischen Sprache in die Grammatik aufgenommen. Die Gruppierung von Termen und Änderung der Operatorrangfolge ist durch eine einfache Klammerung möglich.

$N \ni$ $term$

$T \ni$ $NOT \coloneqq$ [$\neg\ not$],

$AND \coloneqq$ [$\wedge\ and$],

$OR \coloneqq$ [$\vee\ or$],

$IMPLY \coloneqq$ [$\rightarrow\ imply$],

$PARENTHESIS\_OPEN \coloneqq$ (,

$PARENTHESIS\_CLOSE \coloneqq$)

$P \ni$ $term \rightarrow$ $variable$

| $NOT\ term$

| $term\ AND\ term$

| $term\ OR\ term$

| $term\ IMPLY\ term$

| $PARENTHESIS\_OPEN\ term\ PARENTHESIS\_CLOSE$

Die entstandene Grammatik G ist dabei in der Chomsky-Hierarchie dem Typ 2, den kontextfreien Grammatiken, zuzuordnen [14]. Jede Produktionsregel folgt dem Muster $n \rightarrow v*$, wobei $n \in N$ und $v \in T \cup N$. Dadurch wird die Bedingung einer kontextfreien Grammatik erfüllt, die per Definition auch Teilmenge der kontextsensitiven Grammatiken und damit wiederum der beliebigen formalen Grammatiken ist [14]. Die Erfüllung der Bedingung einer regulären Grammatik $n \rightarrow$

$\varepsilon \mid t* \mid t*n$ (rechtsregulär) bzw. $nt*$ (linksregulär), wobei $t \in T$, scheitert an der Produktionsregel $term \rightarrow PARENTHESIS\_OPEN\ term\ PARENTHESIS\_CLOSE$. Während andere Regeln der Form $n \rightarrow ntn$ in eine reguläre Form übersetzt werden können, ist es nicht möglich mit einer endlichen Anzahl von Nicht-Terminalen eine korrekte Klammerung zu gewährleisten, da die Regel $term \rightarrow PARENTHESIS\_OPEN\ term$ das Wort ‚(A and B' als gültige Eingabe zulassen würde. Dies lässt sich besonders durch die Darstellung der Grammatik als Kellerautomat verdeutlichen.

3.2.2 Kellerautomat

Ein Kellerautomat ermöglicht die grafische Darstellung einer kontextfreien Grammatik. Ein Automat umfasst Zustände, die mittels Übergangsfunktionen gewechselt werden können – diese leiten sich direkt aus den Produktionsregeln ab. Die grafische Darstellung vereinfacht dabei die Prüfung einer Eingabe, da man auf dem Papier dem Verlauf der Zustände folgen kann. So lässt sich die zuvor definierte Grammatik auf ihre Korrektheit prüfen.

Dabei sei K der Kellerautomat bestehend aus der Zustandsmenge Q, dem Eingabealphabet T, dem Kelleralphabet A, den Zustandsübergangsfunktionen Z, dem Start-Zustand q_0, dem Anfangssymbol im Keller # und der Menge der Endzustände F. [14]

$$K = (Q, T, A, Z, q_0, \#, F)$$

$$q_0 \in Q$$

$$F \subseteq Q$$

$$\# \in A$$

Leitet man die Grammatik G nun in einen Automaten über, erhält man folgende Werte für dessen Eigenschaften. Die Zustandsmenge Q und die Zustandsübergangsfunktionen leiten sich direkt aus den Produktionsregeln P der Grammatik ab. Das Eingabealphabet beinhaltet die Terminale der Grammatik, die Symbole des Kelleralphabets sind frei gewählt und austauschbar. Die Menge der Endzustände gibt an, in welchen Zuständen die Eingabe ein Wort der erzeugten Sprache ist, wenn der Automat terminiert.

$$
\begin{aligned}
Q :=\ & \{q_0, q_1, q_2, q_3, q_4\} \\
F :=\ & \{q_4\} \\
A :=\ & \{\#, \{, (\} \\
Z :=\ & \{\ q_0, \varepsilon, \# \rightarrow q_1, \#\ , \\
& \quad q_0, STATEMENTS\_OPEN, \# \rightarrow q_1, \#\{\ , \\
& \quad q_1, PARENTHESIS\_OPEN, \rightarrow q_1, *\ (\ , \\
& \quad q_1, NOT, * \rightarrow q_1, *\ , \\
& \quad q_1, ALPHA, * \rightarrow q_2, *\ , \\
& \quad q_2, ALPHA, * \rightarrow q_2, *\ , \\
& \quad q_2, DIGIT, * \rightarrow q_2, *\ , \\
& \quad q_2, AND, * \rightarrow q_1, *\ , \\
& \quad q_2, OR, * \rightarrow q_1, *\ , \\
& \quad q_2, IMPLY, * \rightarrow q_1, *\ , \\
& \quad q_2, STATEMENT\_DELIMIT, * \rightarrow q_1, *\ , \\
& \quad q_2, PARENTHESIS\_CLOSE, (\rightarrow q_3, \varepsilon\ , \\
& \quad q_2, \varepsilon, \# \rightarrow q_4, \#\ , \\
& \quad q_3, AND, * \rightarrow q_1, *\ , \\
& \quad q_3, OR, * \rightarrow q_1, *\ , \\
& \quad q_3, IMPLY, * \rightarrow q_1, *\ , \\
& \quad q_3, STATEMENT\_DELIMIT, * \rightarrow q_1, *\ , \\
& \quad q_3, STATEMENT\_CLOSE, \{ \rightarrow q_4, \varepsilon\ , \\
& \quad q_3, \varepsilon, \# \rightarrow q_4, \# \quad \}
\end{aligned}
$$

'$*$' sei dabei ein beliebiges Zeichen des Kelleralphabets.

Es handelt sich um einen deterministischen Automaten, da jede Zustandsübergangsfunktion in Z mit einer Eingabe $t \in T$ und dem obersten Kellerelement immer maximal einen möglichen Zustand q_i für jeden Zustand von Q erlaubt – der Automat muss nie eine Entscheidung treffen, welche Zustandsübergangsfunktion er anwenden soll – folglich die Bezeichnung ‚deterministisch'.

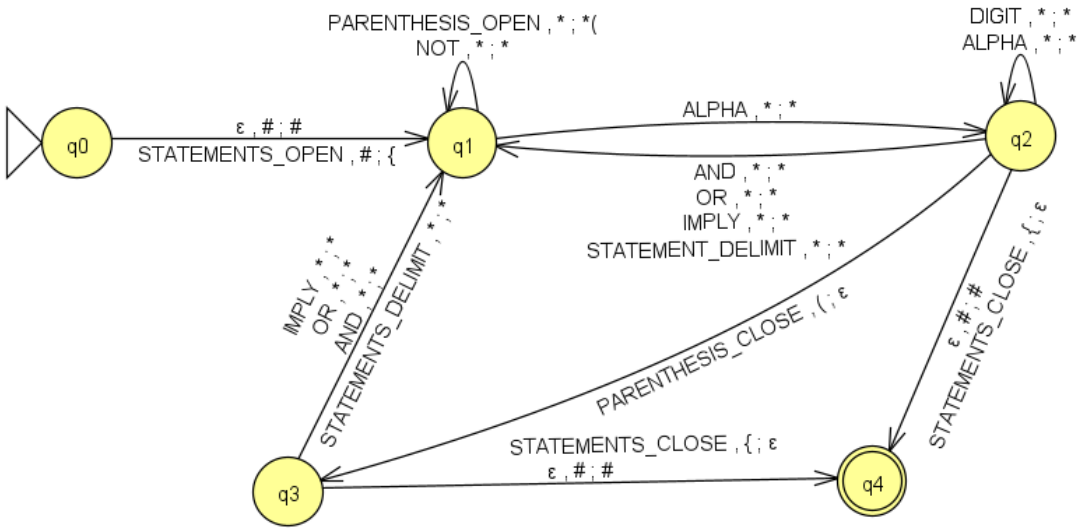


Abbildung 15: Grafische Darstellung des Kellerautomaten

Es ist erkennbar, dass nur in zwei Situationen Elemente in den Keller geschoben bzw. entfernt werden: beim Öffnen bzw. Schließen der Aussagen-Menge und bei der Klammerung von Termen. Das Öffnen und Schließen der Aussagen-Menge kann nur einmalig auftreten und daher ebenso durch die Erweiterung des Automaten um mehrere Zustände ohne Interaktion mit dem Keller realisiert werden, worauf aus Gründen der Übersichtlichkeit jedoch verzichtet wurde. Da die Klammerung von Termen jedoch rekursiv beliebig häufig auftreten kann, ist die Verwendung des Kellers hier unumgänglich und eine Bildung einer links- bzw. rechts-regulären Grammatik (Typ 3) derselben Sprache nicht möglich.

$\{A \wedge B, \neg C\}$:

Eingabe:		$\{$		A		$\wedge$		B		,		$\neg$		C		$\}$	
Zustand:	q_0	$\rightarrow$	q_1	$\rightarrow$	q_2	$\rightarrow$	q_1	$\rightarrow$	q_2	$\rightarrow$	q_1	$\rightarrow$	q_1	$\rightarrow$	q_2	$\rightarrow$	q_4
Keller:	#		{		{		{		{		{		{		{		#
			#		#		#		#		#		#		#		

Tabelle 3: Zustandsverlauf des Kellerautomaten für das Wort $\{A \wedge B, \neg C\}$

$\neg(A \vee B)$:

Eingabe:		ε		$\neg$		(		A		$\vee$		B		)		C	
Zustand:	q_0	$\rightarrow$	q_1	$\rightarrow$	q_1	$\rightarrow$	q_1	$\rightarrow$	q_2	$\rightarrow$	q_1	$\rightarrow$	q_2	$\rightarrow$	q_3	$\rightarrow$	q_4
Keller:	#		#		#		{		{		{		{		#		#
							#		#		#		#				

Tabelle 4: Zustandsverlauf des Kellerautomaten für das Wort $\neg(A \vee B)$

Die beiden Beispiele zeigen den Zustandsverlauf und die Elemente im Keller während der Verarbeitung einer möglichen validen Nutzereingabe und enden jeweils im Final-Zustand q_4 – die Eingaben sind daher Worte der von der Grammatik G erzeugten Sprache.

3.2.3 Integration der Grammatik

Die erzeugte Grammatik wurde mithilfe der Open-Source Programmbibliothek ANTLR (ANother Tool for Language Recognition)[7] in die Anwendung integriert. ANTLR ermöglicht die Eingabe einer Grammatik in der Erweiterten Backus-Naur-Form (EBNF), einer standardisierten Notationsform für Produktionsregeln einer formalen Sprache in der Informatik. Ähnlich der Definition der Grammatik nach Noah Chomsky enthält die EBNF Terminal- und Nicht-Terminal-Symbole, bedient sich jedoch bei den Produktionsregeln statt der Rekursion bestimmter Quantoren. Diese Quantoren ermöglichen Symbole innerhalb einer Produktionsregeln optional zu machen oder beliebig oft zu wiederholen [13]. ANTLR generiert mittels dieser EBNF verschiedene Programm-Klassen zur syntaktischen und semantischen Analyse von Texten. Aus den generierten Klassen kann mittels Vererbung ein eigener Listener implementiert werden, der die Ausführung eigenen Codes beim Verarbeiten der Nutzereingabe ermöglicht. Ein enthaltener Algorithmus ermöglicht das Traversieren über einen aus der Eingabe erzeugten Syntax-Baum nach dem Muster ‚mitte-links-rechts-mitte', wobei ein Knoten betreten, die Kind-Knoten abgearbeitet und der Knoten wieder verlassen wird.

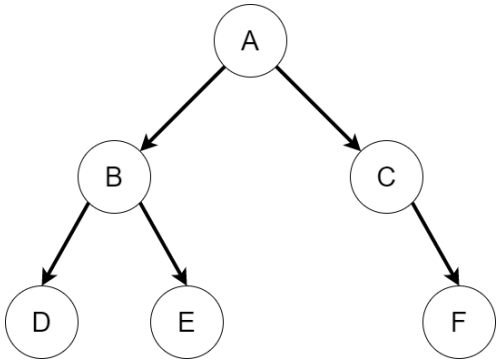


Abbildung 16: Traversier-Reihenfolge: A BDEB CFC A

Da ANTLR keine links-rekursiven Produktionsregeln unterstützt, musste jegliche Links-Rekursion aufgelöst werden. Dabei wurde beispielsweise die Produktionsregel für Variablen mittels den Quantoren der EBNF umgeformt.

Allg. Notation: $variable \rightarrow variable\ DIGIT \mid variable\ ALPHA \mid ALPHA$

EBNF: $variable\colon\ ALPHA\ (ALPHA \mid DIGIT)*$

[7] Siehe https://www.antlr.org

Um die Nutzereingabe einfacher in semantische Java-Objekte überführen zu können, wurden die Regeln der Grammatik um sogenannte ‚Alternative Label' der Programmbibliothek ergänzt, die eine Identifikation der gewählten Produktionsregel bei mehreren Variationen ermöglicht. Die Reihenfolge der Variationen innerhalb einer Produktionsregel ist dabei ausschlaggebend für die Interpretation der Eingabe, da immer die erstmögliche, passende Variation gewählt wird. Dies wurde zur korrekten Interpretation der Präzedenz verwendet, damit die Erzeugung des Wahrheitsbaumes in invertierter Operatorrangfolge geschieht und somit immer die vollständige Lösungsmenge berechnet wird.

$term$: $term\ AND\ term\ \#conjunction$

| $term\ OR\ term\ \#disjunction$

| $term\ IMPLY\ term\ \#subjunction$;

Die vollständige Grammatik in der EBNF-Form kann dem Anhang entnommen werden.

3.3 Datenstrukturen

Zur Verarbeitung des erzeugten Syntax-Baumes wurde ein Datentyp, der den Inhalt der Eingabe strukturiert abbilden kann, erstellt. Da logische Aussagen aus rekursiven Termen bestehen, liegt eine Baumstruktur, die analog zum Traversieren der Eingabe gebaut wird nahe. Um typsicher arbeiten zu können ist dabei eine gemeinsame Klasse oder Oberklasse als Inhaltstyp notwendig. Eine Vererbung eines Terms in verschiedene Satzobjekte könnte dabei wie folgt dargestellt werden:

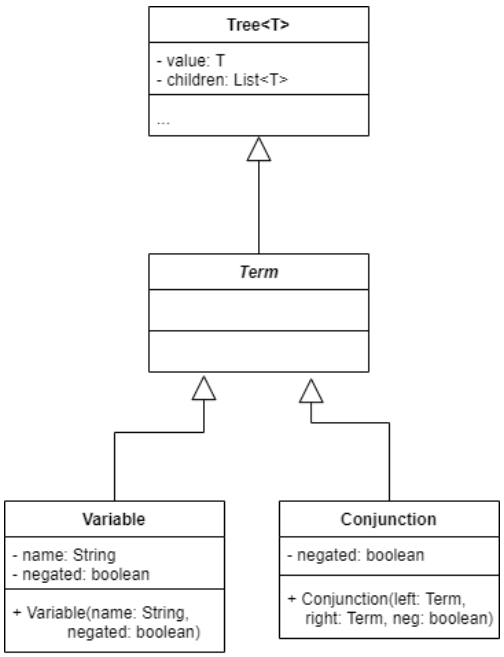


Abbildung 17: UML-Diagramm für Vererbung

Da jedoch in einem ‚m-l-r-m'-Schema über die Eingabe traversiert wird, ist beim Betreten des Knotens dessen Inhalt noch unbekannt, sodass das Objekt erst beim Verlassen konstruiert werden kann. Dann kann der semantische Baum jedoch nicht analog zum Traversieren des Syntaxbaumes aufgebaut werden, sondern es müssen Referenzen zu den jeweiligen semantischen Kind-Knoten zwischengespeichert und beim Verlassen eines Syntaxknotens abgerufen werden. Daher wurde die Datenstruktur angepasst, um einen Term ohne eine Spezifizierung erstellen und diese später gemeinsam mit dem Inhalt zuweisen zu können – dies erlaubt den Knoten bereits beim Betreten der Produktionsregel zu erzeugen und in den Baum einzuhängen. Da somit die Repräsentation durch verschiedene Unterklassen verschwindet, wurde ein Enum zur Unterscheidung zwischen den Inhaltstypen definiert, um die Objekte verschieden verarbeiten zu können.

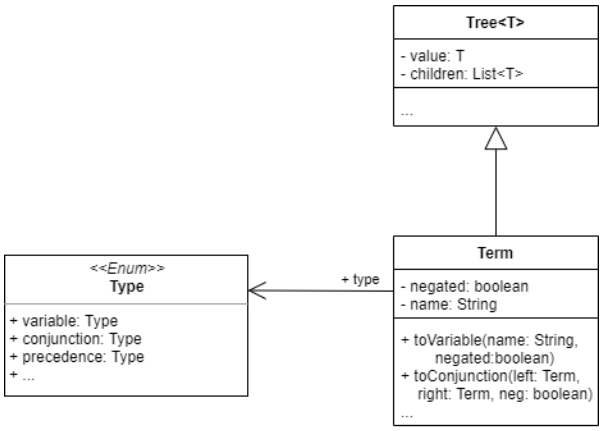


Abbildung 18: UML-Diagramm mit Type-Enum

Zuletzt wurden die Zuweisungen in eine nicht-öffentliche Unterklasse ausgelagert, damit die Terme nach der Erstellung des Baums nicht manipuliert werden können und um den Programmcode übersichtlicher zu gestalten.

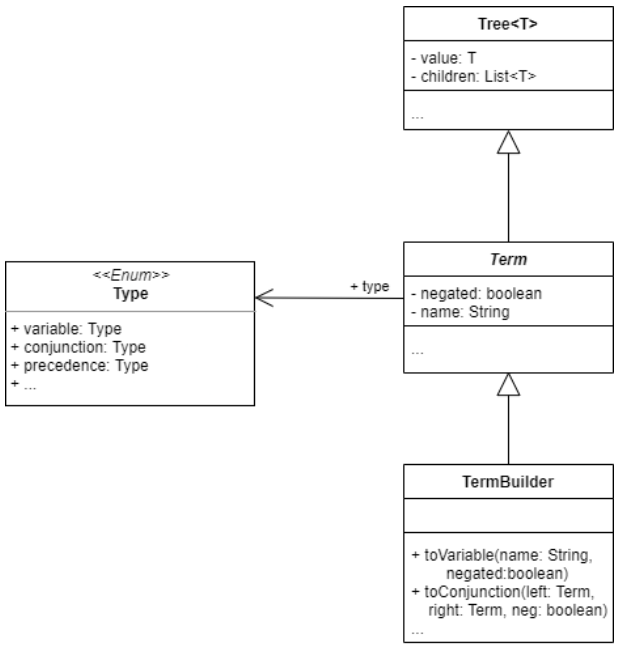


Abbildung 19: UML-Diagramm mit abgeleiteter Builder-Klasse

Die Term-Klasse ist bereits hinreichend, um eine einzelne logische Aussage als Baumstruktur abzubilden. Um einen Wahrheitsbaum abzubilden ist allerdings ein weiterer Baum mit Knoten aus einer oder mehreren Aussagen notwendig. Die erstellte Branch-Klasse speichert zudem, ob der Zweig des Wahrheitsbaumes einen logischen Widerspruch enthält.

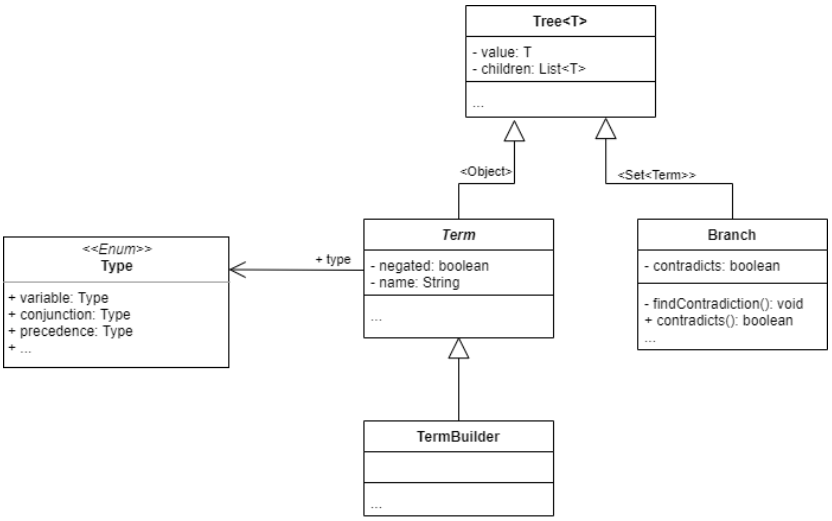


Abbildung 20: UML-Diagramm der Vererbung der Tree-Klasse

Ein vollständiges UML-Diagramm der gesamten Anwendung ist im Anhang zu finden.

3.4 Erstellung des Wahrheitsbaumes

Nachdem aus der Nutzereingabe Term-Objekte erzeugt und zu einem Knoten des Wahrheitsbaumes zusammengefasst wurden, gilt es die Terme rekursiv zu transformieren und in den Baum einzuhängen. Dafür wird eine Queue verwendet, in die die Wurzel des Baumes sowie jeder neu abgeleitete Term eingereiht werden. Gleichzeitig wird eine Zuordnung der Terme zu ihrem jeweiligen Zweig gespeichert, um die neu erzeugten Kind-Knoten dem richtigen Zweig zuzuordnen. Beim Einfügen neuer Elemente wird zudem auf logische Widersprüche geprüft: hierfür werden alle Terme aller Knoten eines Zweiges auf die abgebildeten Variablen gefiltert und abgeglichen, ob eine Variable mehrfach, mit variierender Negation auftritt. Enthält ein Zweig sowohl die negierte und nicht-negierte Form einer Variable, liegt ein logischer Widerspruch vor und der Zweig wird nicht weiter transformiert. Ist die Queue leer, gibt es keine zu transformierenden Terme mehr. Alle Zweige resultieren in einem Widerspruch oder wurden zu einer validen Kombination der Variablen aufgelöst.

Die Transformation erfolgt mithilfe der Spezifizierung des Objekt-Typs. Basierend auf dem Typ und dem Vorhandensein einer Negation wird nach den definierten Transformationsregeln eine Menge von Kind-Knoten mit einem oder mehreren Termen zurückgegeben. Da bei der Transformation eines Knotens mit mehreren Termen ein Einfügen der Kind-Terme in mehrere Zweige auftreten kann, musste zudem ein Copy-Konstruktor hinzugefügt werden – da Java Objekte in einem getrennten Heap-Speicher verwaltet werden und mehrere Verweise auf dasselbe Objekt zeigen, tritt ein Widerspruch in einem Zweig ansonsten in allen Zweigen, die diesen Knoten enthalten, auf.

```
while (!queue.isEmpty()) {
    Term statement = queue.poll();
    Branch parent = parents.get(statement);
    Set<Set<Term>> branches = statement.transform();
    if (!branches.isEmpty()) {
        Set<Branch> children = Util.mapSet(branches, Branch::new);
        List<Branch> successors = parent.getLeafs();
        for (Branch successor : successors) {
            if (!successor.contradicts()) {
                for (Branch child : children) {
                    Branch copy = new Branch(child);
                    successor.addChild(copy);
                    if (!copy.contradicts()) {
                        Set<Term> statements = copy.value();
                        statements.forEach(term -> parents.put(term, copy));
                        queue.addAll(statements);
                    }
                }
            }
        }
    }
}
```

Abbildung 21: Code-Ausschnitt zur Erstellung des Wahrheitsbaumes

Das erneute Iterieren über alle Zweige ohne Widerspruch ermöglicht die Abfrage der Variablenkombinationen, die die Lösungsmenge des Wahrheitsbaumes repräsentiert. Analog zur händischen Durchführung kann diese leer sein, ist jedoch immer vollständig, da die Operatorpräzedenz bei der Transformation berücksichtigt wird.

3.5 Visualisierung

Zur grafischen Ausgabe des Wahrheitsbaumes wird ein Algorithmus zum Zeichnen der Knoten auf einem JPanel verwendet. Dabei wird ausgehend von der Breite der Zeichenfläche eine Teilung für jede Verzweigung im Baum vorgenommen, sodass die Breite für einen vollständigen Binärbaum optimal verteilt wäre. Da der Algorithmus keine optimale Breite für die Grafik selbst bestimmt, kann der Nutzer die gewünschte Breite als Eingabeparameter mit angeben. Das verwendete JPanel kann als Bild exportiert werden, um die erzeugte Grafik beispielsweise innerhalb dieser Arbeit zu verwenden. Zusätzlich kann der Nutzer das erzeugte JPanel in einem JFrame anzeigen lassen, um die Größe der erstellten Grafik zu prüfen, nachträglich anzupassen und die Abbildung per Kontextmenü abzuspeichern.

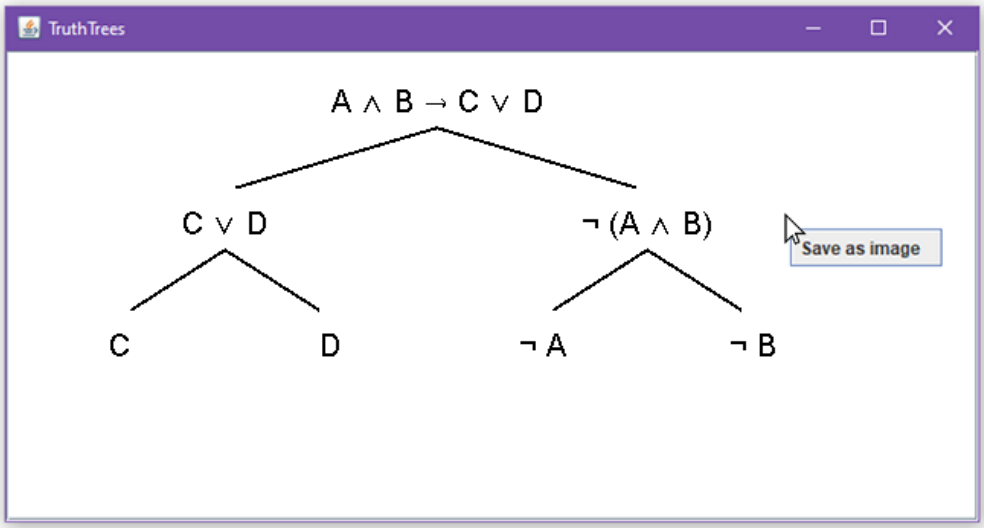


Abbildung 22: Grafische Ausgabe in eigenem Fenster

4 Fazit

Im Rahmen der zentralen Fragestellung dieser Arbeit, wie man eine Anwendung erstellt, die Wahrheitsbäume erzeugen und visualisieren kann, wurde der geschichtliche Hintergrund beschrieben und die Beweisführung per Widerspruch erläutert. Es wurde eine Menge von Transformationsregeln definiert und die Vorgehensweise der Methode theoretisch und anhand von Beispielen dargelegt. Darauf basierend wurde ein Konzept entworfen für eine Anwendung, die die logischen Aussagen als Eingabe akzeptiert, mithilfe einer Grammatik analysiert – die Grammatik mit einem Kellerautomat überprüft – und in geeignete Datenstrukturen übersetzt. Die Vorgehensweise zur Erzeugung des Wahrheitsbaumes wurde auf die Anwendung übertragen und ein Algorithmus zum Zeichnen des Wahrheitsbaumes für die grafische Ausgabe geschrieben.

Dabei lässt sich die Menge der Transformationsregeln beliebig um zusätzliche logische Verknüpfungen erweitern. Die Arbeit zeigt, wie eine Methode einer computerfremden Domäne in eine Anwendung übersetzt und computergestützt durchgeführt werden kann.

Wörterzählung: 3952 Worte

Literaturverzeichnis

[1] H. E. Baber. Truth Trees. University of San Diego. Retrieved September 16, 2020 from http://home.sandiego.edu/~baber/logic/10.TruthTrees.pdf

[2] Nick Bezhanishvili, Dicke de Jongh. 2005. Intuitionistic Logic. Institute for Logic, Language and Computation, Universiteit van Amsterdam. Retrieved from https://www.illc.uva.nl/Research/Publications/Reports/PP-2006-25.text.pdf

[3] Jean-Yves Béziau. 2003. Bivalence, Excluded Middle and Non Contradiction. The Logica Yearbook 2003, L.Behounek (ed), Academy of Sciences, Prague, 2003, pp.73-84.

[4] M. D'Agostino, D. M. Gabbay, R. Hähnle and J. Posegga (eds.). 1998. Handbook of Tableau Methods. DOI: https://doi.org/10.1007/978-94-017-1754-0

[5] Arend Heyting. 1966. In memoriam: Evert Willem Beth (1909--1964). Notre Dame J. Formal Logic 7 (1966), no. 4, 289--295. DOI: 10.1305/ndjfl/1093958744. https://projecteuclid.org/euclid.ndjfl/1093958744

[6] Pieter Hofstra. Truth Trees. Department of Mathematics and Statistics, University of Ottawa. Retrieved September 16, 2020 from https://mysite.science.uottawa.ca/phofstra/MAT1348/TruthTrees.pdf

[7] Steffen Hölldobler. 2009. Aussagenlogik. International Center for Computational Logic. Technische Universität Dresden. Retrieved from https://web.archive.org/web/20160417203320/http:/www.wv.inf.tu-dresden.de/Teaching/WS-2009/logik/folien/als2009.pdf

[8] W. A. Hunt Jr, M. Kaufmann, J. S. Moore, A. Slobodova. 2017. Industrial hardware and software verification with ACL2. Phil. Trans. R. Soc. A 375: 20150399. DOI: http://dx.doi.org/10.1098/rsta.2015.0399

[9] Rosalie Iemhoff, Edward N. Zalta (ed.). 2020. Intuitionism in the Philosophy of Mathematics. The Stanford Encyclopedia of Philosophy. Retrieved September 16, 2020 from https://plato.stanford.edu/entries/intuitionism/

[10] Cornelis Menke. 2018. Einführung in die formale Logik. Retrieved September 16, 2020 from https://www.studgen-iful.uni-mainz.de/files/2019/01/Menke_2016_Formale-Logik_v.18.9.pdf

[11] Joan Moschovakis, Edward N. Zalta (ed.). 2018. Intuitionistic Logic. The Stanford Encyclopedia of Philosophy. Retrieved September 16, 2020 from https://plato.stanford.edu/entries/logic-intuitionistic/

[12] Sven Eric Panitz. 2009. Logik, Komplexität und Berechenbarkeit. Hochschule RheinMain. Retrieved September 16, 2020 from https://www.cs.hs-rm.de/~panitz/logik/skript.pdf

[13] Rich Pattis. EBNF: A Notation to Describe Syntax. Department of Computer Science. University of California. Irvine, California. Retrieved September 16, 2020 from https://www.ics.uci.edu/~pattis/ICS-33/lectures/ebnf.pdf

[14] James Power. 2002. Notes on Formal Language Theory and Parsing. Department of Computer Science. National University of Ireland, Maynooth, Co. Kildare, Ireland. Retrieved September 16, 2020 from http://www.cs.may.ie/~jpower/Courses/parsing/parsing.pdf

[15] David M. Russinoff. 2019. Formal Verification of Floating-Point RTL with ACL2. University of Cambridge. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5597723/pdf/rsta20150399.pdf

[16] Peter Suber. 1997. Paradoxes of Material Implication. Department of Philosophy, Earlham College, Richmond, Indiana, USA. Retrieved September 16, 2020 from https://legacy.earlham.edu/~peters/courses/log/mat-imp.htm

[17] A. S. Troelstra, P. van Ulsen. The discovery of E. W. Beth's semantics from intuitionistic logic. University of Amsterdam. Retrieved September 16, 2020 from http://festschriften.illc.uva.nl/j50/contribs/troelstra/troelstra.pdf

[18] P. van Ulsen. 2000. E.W. Beth als logicus. Ph.D. Dissertation. Institute for Logic, Language and Computation, University Of Amsterdam, Amsteram, NL. ISBN: 9057760525

[19] Ideas in Mathematics: The Grammar of Numbers. Retrieved September 16, 2020 from http://fmwww.bc.edu/gross/MT007/NumSqrt2.html

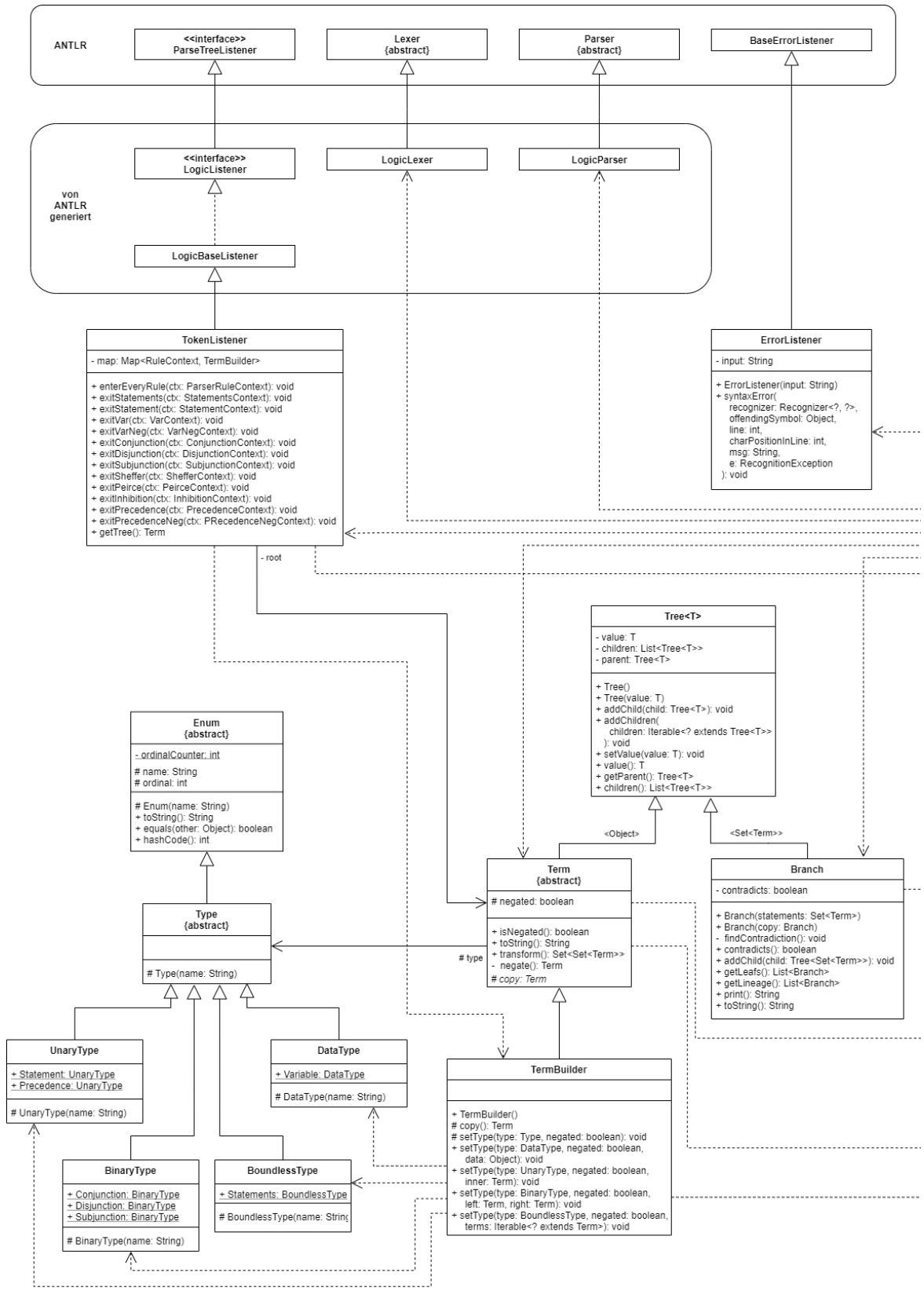
Anhangsverzeichnis

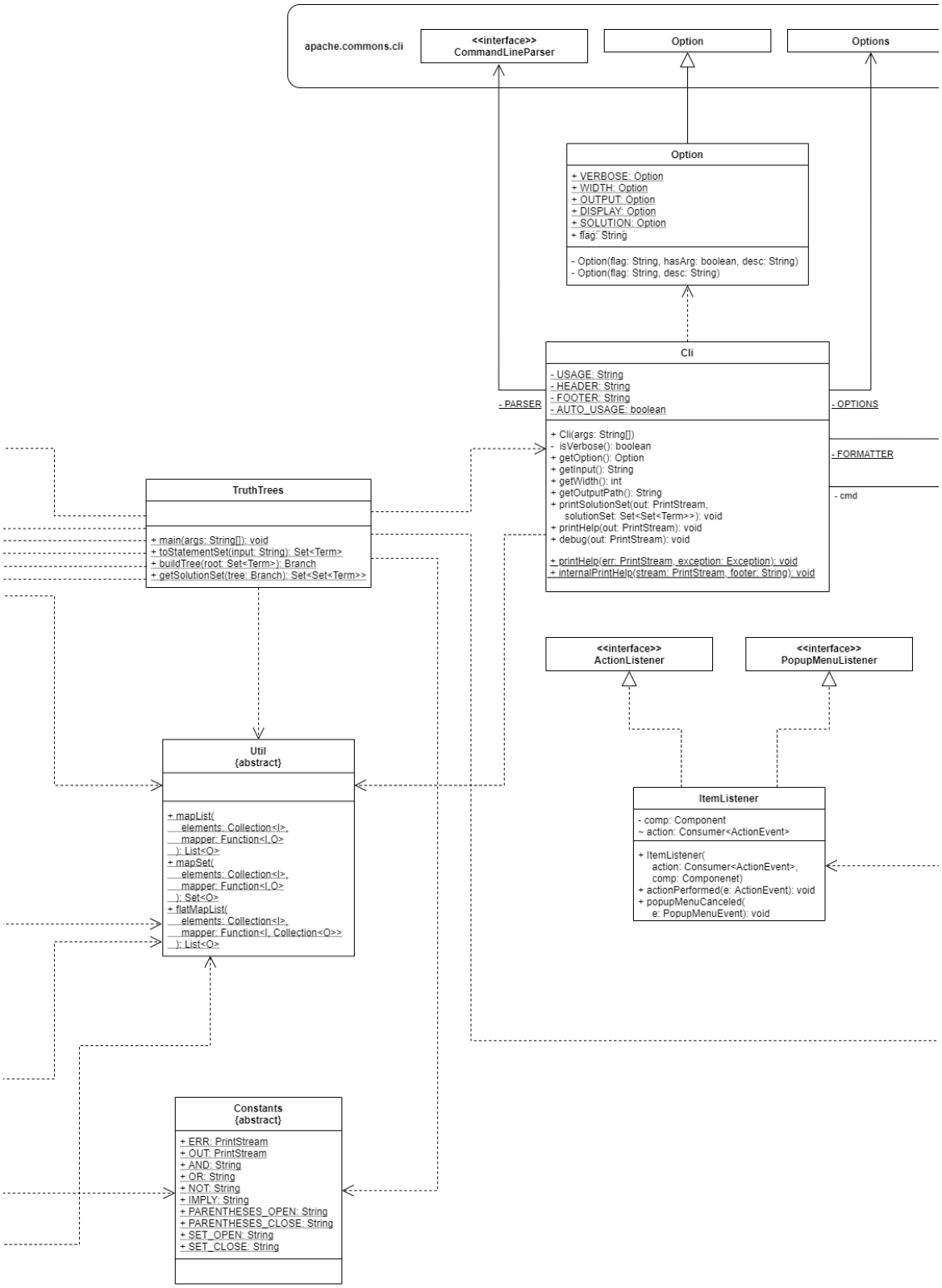
Anhang

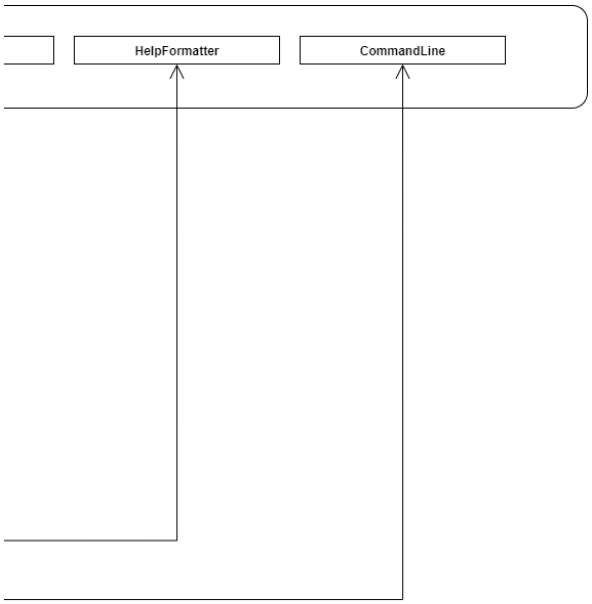
I. Grammatik in der EBNF-Form

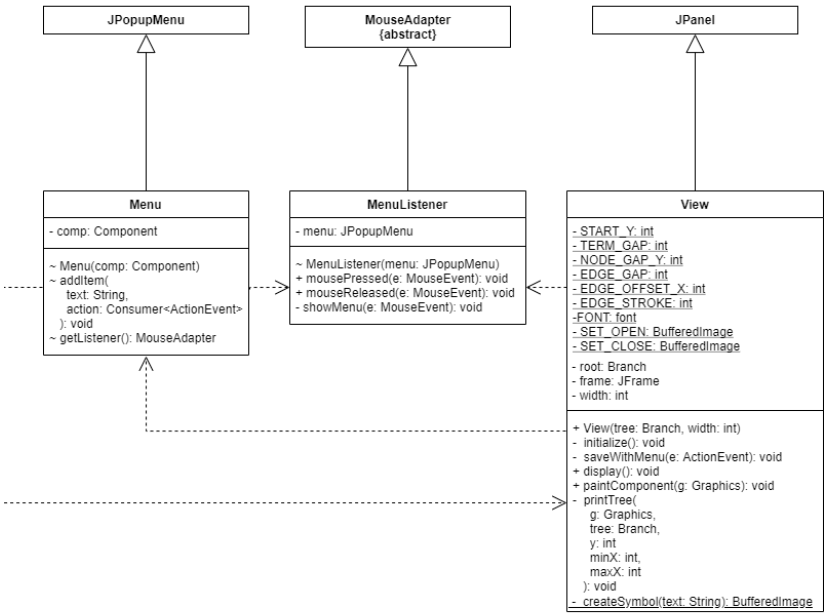
```
// rules
variable:
   ALPHA (ALPHA | DIGIT)*;
term:
     variable #var
   | NOT variable #varNeg
   | term AND term #conjunction
   | NOT PARENTHESES_OPEN term AND term PARENTHESES_CLOSE #sheffer
   | term OR term #disjunction
   | NOT PARENTHESES_OPEN term OR term PARENTHESES_CLOSE #peirce
   | term IMPLY term #subjunction
   | NOT PARENTHESES_OPEN term IMPLY term PARENTHESES_CLOSE #inhibition
   | PARENTHESES_OPEN term PARENTHESES_CLOSE #precedence
   | NOT PARENTHESES_OPEN term PARENTHESES_CLOSE #precedenceNeg;
statement:
     term;
statements:
     statement (STATEMENT_DELIMIT statement)*
   | STATEMENTS_OPEN? statement (STATEMENT_DELIMIT statement)*
STATEMENTS_CLOSE?;


// tokens
ALPHA: [a-zA-Z];
DIGIT: [0-9];

NOT: 'NOT' | 'not' | '¬';
AND: 'AND' | 'and' | '∧';
OR: 'OR' | 'or' | '∨';
IMPLY: 'IMPLY' | 'imply' | '→';

PARENTHESES_OPEN: '(';
PARENTHESES_CLOSE: ')';
STATEMENT_DELIMIT: ',' | ';';
STATEMENTS_OPEN: '{';
STATEMENTS_CLOSE: '}';

WS: [ \t\r\n]+ -> skip; // skip spaces, tabs, newlines
```

II. UML-Diagramm




apache.commons.cli
<<interface>>
CommandLineParser
Option
Options
Option
+ VERBOSE: Option
+ WIDTH: Option
+ OUTPUT: Option
+ DISPLAY: Option
+ SOLUTION: Option
+ flag: String
- Option(flag: String, hasArg: boolean, desc: String)
- Option(flag: String, desc: String)
Cli
- USAGE: String
- HEADER: String
- FOOTER: String
- AUTO_USAGE: boolean
+ Cli(args: String[])
- isVerbose(): boolean
+ getOption(): Option
+ getInput(): String
+ getWidth(): int
+ getOutputPath(): String
+ printSolutionSet(out: PrintStream, solutionSet: Set<Set<Term>>): void
+ printHelp(out: PrintStream): void
+ debug(out: PrintStream): void
+ printHelp(err: PrintStream, exception: Exception): void
+ internalPrintHelp(stream: PrintStream, footer: String): void
- PARSER
- OPTIONS
- FORMATTER
- cmd
TruthTrees
+ main(args: String[]): void
+ toStatementSet(input: String): Set<Term>
+ buildTree(root: Set<Term>): Branch
+ getSolutionSet(tree: Branch): Set<Set<Term>>
<<interface>>
ActionListener
<<interface>>
PopupMenuListener
Util
{abstract}
+ mapList(
elements: Collection<I>,
mapper: Function<I,O>
): List<O>
+ mapSet(
elements: Collection<I>,
mapper: Function<I,O>
): Set<O>
+ flatMapList(
elements: Collection<I>,
mapper: Function<I, Collection<O>>
): List<O>
ItemListener
- comp: Component
~ action: Consumer<ActionEvent>
+ ItemListener(
action: Consumer<ActionEvent>,
comp: Componenet)
+ actionPerformed(e: ActionEvent): void
+ popupMenuCanceled(
e: PopupMenuEvent): void
Constants
{abstract}
+ ERR: PrintStream
+ OUT: PrintStream
+ AND: String
+ OR: String
+ NOT: String
+ IMPLY: String
+ PARENTHESES_OPEN: String
+ PARENTHESES_CLOSE: String
+ SET_OPEN: String
+ SET_CLOSE: String

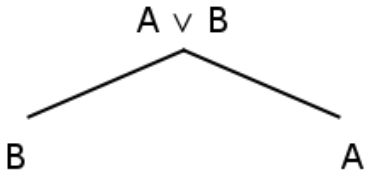

HelpFormatter
CommandLine


JPopupMenu
MouseAdapter
{abstract}
JPanel
Menu
- comp: Component
~ Menu(comp: Component)
~ addItem(
text: String,
action: Consumer<ActionEvent>
): void
~ getListener(): MouseAdapter
MenuListener
- menu: JPopupMenu
~ MenuListener(menu: JPopupMenu)
+ mousePressed(e: MouseEvent): void
+ mouseReleased(e: MouseEvent): void
- showMenu(e: MouseEvent): void
View
- START_Y: int
- TERM_GAP: int
- NODE_GAP_Y: int
- EDGE_GAP: int
- EDGE_OFFSET_X: int
- EDGE_STROKE: int
-FONT: font
- SET_OPEN: BufferedImage
- SET_CLOSE: BufferedImage
- root: Branch
- frame: JFrame
- width: int
+ View(tree: Branch, width: int)
- initialize(): void
- saveWithMenu(e: ActionEvent): void
+ display(): void
+ paintComponent(g: Graphics): void
- printTree(
g: Graphics,
tree: Branch,
y: int
minX: int,
maxX: int
): void
- createSymbol(text: String): BufferedImage

III. Beweis der unvollständigen Lösungsmenge

These: Bei Vernachlässigung der Operatorpräzedenz bei Erstellung eines Wahrheitsbaumes erhält man nicht immer die vollständige Lösungsmenge.

Beweis: Gegeben sei die Menge R mit folgenden Transformationsregeln:



Der Operator ∧ hat eine höhere Präzedenz als der Operator ∨.

Gegeben sei die logische Aussage $f \coloneqq A \vee B \wedge \neg B \wedge C$.

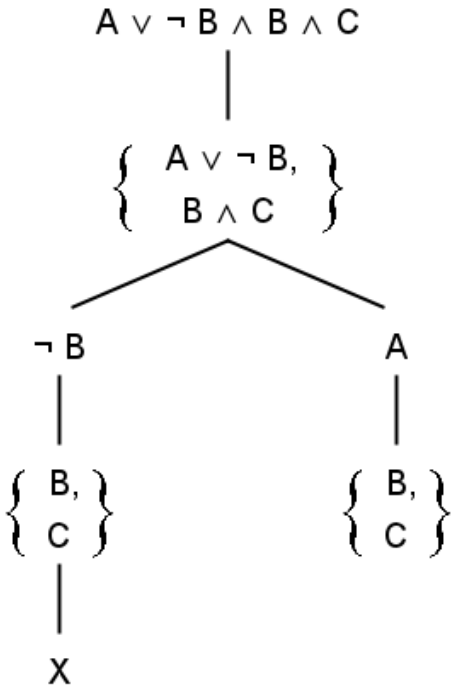
Aus der folgenden Wahrheitstabelle für f lässt sich die vollständige Lösungsmenge L_1 ableiten und zu L_2 vereinfachen:

A	B	C	$A \vee B \wedge \neg B \wedge C$
F	F	F	F
F	F	T	F
F	T	F	F
F	T	T	F
T	F	F	T
T	F	T	T
T	T	F	T
T	T	T	T

$$L_1 = \{\{A, \neg B, \neg C\}, \{A, \neg B, C\}, \{A, B, \neg C\}, \{A, B, C\}\}$$
$$L_2 = \{\{A\}\}$$

Angenommen jeder Wahrheitsbaum ergibt ungeachtet der Operatorpräzedenz immer die vollständige Lösungsmenge. Dann ergibt jeder Wahrheitsbaum aus f die Lösungsmenge L_1 bzw. L_2.

Wird die Operatorpräzedenz bei der Transformation der logischen Aussage f ignoriert, entsteht folgender Wahrheitsbaum:



$$L_3 = \{\{A, B, C\}\}$$

Der Baum gibt nur eine mögliche Kombination der Variablen zur Erfüllung der Aussage f an. L_3 ist eine echte Teilmenge von L_1. Folglich ist die Annahme, dass jeder Wahrheitsbaum für f die vollständige Lösungsmenge L_1 bzw. L_2 ergibt falsch.

Das zeigt, dass man bei Vernachlässigung der Operatorpräzedenz nicht immer die vollständige Lösungsmenge erhält.