Java-Programm zur interaktiven Bearbeitung von $\mathcal{JALC}$-Herleitungen

als
Bachelorarbeit
an der Naturwissenschaftlichen Fakultät
der Universität Bern

von
Roger Peter Kohler
von Landiswil
im 7. Semester
Matrikelnummer: 08-114-969

Studienadresse
Eggiwilstrasse 11
3535 Schüpbach
(Tel. 079 813 42 73)
(E-Mail: ropeko@students.unibe.ch)

Bern, 09.01.2012

Betreuer
PD Dr. Thomas Studer
Institut für Informatik und angewandte Mathematik
Abteilung Theoretische Informatik und Logik

Zusammenfassung

Während das World Wide Web Daten verbindet, aber deren Inhalt nicht weiter interpretiert, versucht das Semantic Web ähnliche Inhalte miteinander zu verknüpfen.[1] Oft wird aber nur das Resultat einer Suche nach Informationen dargestellt. Wie dieses entstand, ist für den Betrachter schwer zu verstehen. Deshalb wird hier ein Programm vorgestellt, das diese Herleitung vollständig anzeigt und interaktiv vom Benutzer untersucht werden kann. Zusätzlich gibt das Programm für die einzelnen Herleitungsschritte neben der Formel der Rechtfertigungslogik $\mathcal{JALC}$[2] auch eine besser verständliche Erläuterung. Diese wird dynamisch angepasst, je nachdem, wie viel von der Herleitung ersichtlich ist.

[1] Vgl. [4], S. 1
[2] Basierend auf [8]

Inhaltsverzeichnis

Kapitel 1

Einleitung

In diesem Kapitel wird erläutert, was die Ziele dieser Arbeit sind und auf welchen Problemen diese basieren. Weiter wird auf den Aufbau der vorliegenden Arbeit eingegangen und erklärt, mit welchen Hilfsmitteln das Ziel erreicht werden soll.

1.1 Ausgangslage

Heutzutage bildet das World Wide Web ein zentrales Hilfsmittel um an Informationen zu gelangen. Während das WWW auf Standards wie XML[1] und HTTP[2] basiert und die Daten miteinander verknüpft, ist beim Semantic Web der Inhalt von Objekten von zentraler Bedeutung.[3] Es versucht die Informationen auszuwerten und ähnliche miteinander zu verknüpfen. Zum Beispiel soll das Semantic Web zwischen Vornamen und Nachnamen unterscheiden können (was offensichtlich relativ schwierig ist). Dazu werden sogenannte Ontologien verwendet, die dem Begriff sagen, dass dieser ein Vorname ist. Um diese in einer definierten Form zu schreiben und Verknüpfungen zu erstellen, werden entsprechende Logiken benötigt, mit deren Hilfe die gesuchten Informationen herausgefiltert werden können.[4]

1.2 Problemstellung

Es stellt sich die Frage, wie eine solche Logik aufgebaut sein muss um möglichst gute Informationen zu gewinnen. An Logiken, die zu diesem Zweck definiert wurden, mangelt es keineswegs. Oft wird aber nur das Resultat angezeigt und es ist schwierig von den Ausgangsinformationen auf dieses Resultat schliessen zu können. Deshalb wäre es nützlich, wenn die Zwischenschritte angezeigt werden. Es stellen sich weitere Fragen. Falls die Zwischenschritte nun dementsprechend

[1] XML: Extensible Markup Language
[2] HTTP: Hypertext Transfer Protocol
[3] Vgl. [4], S. 1
[4] Vgl. [3], S. 4

angezeigt werden können, sind einige offensichtlich und dem Betrachter wäre es lieber nur die komplexeren Schritte zu untersuchen. Eine Schwierigkeit besteht darin, dass die Komplexität eines Schrittes schwer zu bestimmen ist. Deshalb wäre es von Vorteil, wenn der Betrachter die Herleitung interaktiv nach eigenem Ermessen untersuchen kann. Unwesentliche Schritte soll er schliessen und interessante tiefgründiger anschauen können.
Ein Problem besteht auch in der Anschaulichkeit von den Herleitungsschritten einer Logik. Oft sind sie durch die definierte Schreibweise schwierig zu verstehen. Eine dazugehörende für den Nutzer besser lesbare Erläuterung des Schrittes wäre von grossem Nutzen.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist, ein Programm zu schreiben, das die Zwischenschritte einer Herleitung anzeigen kann. Diese sollen vom Benutzer eigenständig geöffnet werden können, damit nur die Teile der Herleitung aufgeführt werden, die er untersuchen will. Als zusätzliche Eigenschaft sollen die Herleitungsschritte nicht nur in der Sprache der Logik, sondern verständlicher erläutert werden. Diese ergänzenden Erklärungen müssen je nachdem, was von der Herleitung ersichtlich ist, dynamisch angepasst werden.

1.4 Aufbau der Arbeit, Methodisches Vorgehen

Die Arbeit wird in zwei Teile aufgeteilt.
Im ersten Teil werden die theoretischen Grundlagen der Logik übermittelt. Die zu benützende Logik wird $\mathcal{JALC}$[5] sein. Anschliessend sollen verschiedene Beispiele die Anwendung der Rechtfertigungslogik $\mathcal{JALC}$ verdeutlichen und eventuelle Schwierigkeiten aufzeigen.
Als zweiter Teil folgt das Programm. Anfangs werden verschiedene einführende Überlegungen gemacht, die für das Erstellen des Programmes notwendig sind. Danach wird das Programm mithilfe von einigen Screenshots selbst vorgestellt und erklärt. Als Programmierungssprache wird Java verwendet. Abschliessend folgen einige Gedanken zu dieser Arbeit.

[5] Der theoretische Teil über $\mathcal{JALC}$ basiert auf [8]

Kapitel 2

Die Logik $\mathcal{JALC}$

In diesem Kapitel werden die theoretischen Grundlagen übermittelt. Neben der Erklärung, was $\mathcal{JALC}$ bedeutet, werden auch grundlegende Definitionen aufgeführt, die für die späteren Beispiele und das Programm wichtig sind. Das heisst, dass die Definitionen teilweise verkürzt sind. So werden zum Beispiel bei Definition 5 nur Axiome angegeben, die auch im späteren Verlauf dieser Arbeit benutzt werden.

2.1 Bedeutung von $\mathcal{JALC}$

$\mathcal{JALC}$ steht für "Justification Attributive Concept Language with Complements", wobei $\mathcal{C}$ die Abkürzung für $\mathcal{UE}$ ist. Mit $\mathcal{U}$ ist die Vereinigung (engl.: union) von Konzepten (geschrieben C⊔D) gemeint. Das $\mathcal{E}$ erlaubt die Benutzung des Existenzquantors (∃R.C).[1]
$\mathcal{JALC}$ erweitert die Beschreibungslogik $\mathcal{ALC}$. Der wesentliche Punkt von $\mathcal{ALC}$ ist das Definieren von Konzepten, auf denen die restlichen Definitionen aufbauen. Für die Rechtfertigungslogik $\mathcal{JALC}$ werden zusätzlich Terme (siehe Definition 3) verwendet. Diese helfen eine Aussage zu beweisen. So bedeutet [t]ϕ, dass t eine Rechtfertigung für die Aussage ϕ ist.[2] Somit dient eine Rechtfertigungslogik bei der Repräsentation von Wissen in einer formalisierten Art und Weise. Dabei können unter anderem auch Rechtfertigungen für Common Knowledge[3] und Public Announcements[4] behandelt werden.

2.2 Definitionen der Syntax

Es folgen die Definitionen, die für das Programm wesentlich sind.[5]

[1] Vgl. [2], S. 51-53
[2] Vgl. [1], S. 477
[3] S. dazu [6]
[4] S. dazu [7] und [5]
[5] Die Definitionen basieren auf [8], S. 2-4

Definition 1 ***(Konzept):*** *Für abzählbar unendlich viele Konzeptnamen und Rollennamen gilt:*

1. *Jeder Konzeptname ist ein Konzept.*
2. *Seien C und D Konzepte und R ein Rollenname. So sind $\neg C$, $C \sqcap D$ und $\forall R.C$ auch Konzepte.*

Weiter gilt zur Vereinfachung (C, D Konzepte, R Rollenname, A Konzeptname):

$$C \sqcup D := \neg(\neg C \sqcap \neg D)$$
$$\exists R.C := \neg\forall R.\neg C$$
$$\top := A \sqcup \neg A$$

Definition 2 ***($\mathcal{L}_\mathcal{A}$ Formel):*** *Seien C und D Konzepte, ϕ und ψ $\mathcal{L}_\mathcal{A}$ Formeln. Dann sind*

$$C \sqsubseteq D,\ \neg\phi,\ \phi \wedge \psi$$

$\mathcal{L}_\mathcal{A}$ Formeln.

Definition 3 ***(Term)***[6]***:*** *Seien die Konstanten Con und die Variablen Var unendlich abzählbare Mengen. Weiter seien $c \in Con$, $x \in Var$, sowie t_1, t_2 und t Terme. So sind*

$$x,\ c,\ t_1 \cdot t_2,\ t_1 + t_2,\ !t$$

auch Terme.

Definition 4 ***($\mathcal{L}_\mathcal{J}$ Formel):*** *Jede $\mathcal{L}_\mathcal{A}$ Formel ist eine $\mathcal{L}_\mathcal{J}$ Formel.*
Weiter seien t ein Term, ϕ und ψ $\mathcal{L}_\mathcal{J}$ Formeln. Dann sind

$$\neg\phi,\ \phi \wedge \psi,\ [t]\phi$$

$\mathcal{L}_\mathcal{J}$ Formeln.

Definition 5 ***($\mathcal{JALC}$ Deduktives System)***[7]***:*** *Die Axiome in $\mathcal{JALC}$ haben folgende Schemen (t, s Terme, ϕ, ψ $\mathcal{L}_\mathcal{J}$ Formeln):*

1. *Alle gültigen $\mathcal{L}_\mathcal{A}$ Formeln*[8]
2. *$[t](\phi \rightarrow \psi) \rightarrow ([s]\phi \rightarrow [t \cdot s]\psi)$*
3. *$[t]\phi \rightarrow \phi$*

[6] Für das Programm werden nur die ersten drei Termformen x, c und $t_1 \cdot t_2$ benötigt.

[7] Es seien hier nur die Schemen von Axiomen aufgeführt, die in dieser Arbeit von Bedeutung sind. Vgl. [8], S. 4, Definition 7

[8] Gültig bedeutet, dass eine Aussage immer wahr ist.

2.3 Anwendungsbeispiele

Nachdem die Theorie definiert wurde, folgen jetzt zwei Beispiele, die die Anwendung der Beschreibungslogik $\mathcal{JALC}$ aufzeigen sollen.

2.3.1 Beispiel 1: Computertastaturen

Es wird von den gebräuchlisten Computertastaturen ausgegangen. Für einzelne spezielle Tastaturen kann das folgende Beispiel falsch sein. Mit Eingabetasten sind die Tasten gemeint, bei denen ein Zeichen eingegeben wird, sobald sie gedrückt werden (z.B. A, B, ..., Z, 0, 1, ..., 9, ., $, usw.). Keine Eingabetasten sind z.B. Ctrl, Alt, Enter, F1, usw. Seien folgende $\mathcal{L}_{\mathcal{A}}$ Formeln gegeben:

1. Alle Tasten, die doppelt auf der Tastatur vorkommen und Eingabetasten sind, gehören zum NumPad (Ziffernblock):

$$2x \sqcap eingabe \sqsubseteq numPad \qquad (\phi_1)$$

2. Die Taste 1 ist eine Eingabetaste:

$$taste1 \sqsubseteq eingabe \qquad (\phi_2)$$

3. Die Taste 1 gehört zu den Tasten, die doppelt vorkommen:

$$taste1 \sqsubseteq 2x \qquad (\phi_3)$$

Aus diesen Formeln kann hergeleitet werden, dass die Taste 1 im NumPad vorkommt:

$$\phi_1,\ \phi_2,\ \phi_3 \vdash \text{taste1} \sqsubseteq \text{numPad}$$

Obwohl diese Formel offensichtlich gilt, könnte sich der Betrachter fragen, wie die Herleitung mithilfe von $\mathcal{JALC}$ erstellt wird oder welche Axiome verwendet werden.
Natürlich gibt es verschiedene Möglichkeiten auf die Lösung zu schliessen. Nachfolgend eine der möglichen Herleitungen. Diese wird später automatisch vom erstellten Anwendungsprogramm konstruiert.

Vorerst werden noch die Variablen den Formeln vorangefügt[9]:

$$[v](2x \sqcap eingabe \sqsubseteq numPad)$$
$$[w](taste1 \sqsubseteq eingabe)$$
$$[x](taste1 \sqsubseteq 2x)$$

[9] Vgl. Definition 4

Folgende Axiome werden angewendet (A, B, C Konzepte):

$$[a](A \sqsubseteq B \rightarrow (A \sqsubseteq C \rightarrow A \sqsubseteq B \sqcap C))$$
$$[b](A \sqsubseteq B \rightarrow (C \sqsubseteq A \rightarrow C \sqsubseteq B))$$

Damit kann die Aussage hergeleitet werden:

$$[a \cdot x](taste1 \sqsubseteq eingabe \rightarrow taste1 \sqsubseteq 2x \sqcap eingabe)$$
$$[(a \cdot x) \cdot w](taste1 \sqsubseteq 2x \sqcap eingabe)$$
$$[b \cdot v](taste1 \sqsubseteq 2x \sqcap eingabe \rightarrow taste1 \sqsubseteq numPad)$$
$$[(b \cdot v) \cdot ((a \cdot x) \cdot w)](taste1 \sqsubseteq numPad)$$

2.3.2 Beispiel 2: Definition eines Computers

Dieses Beispiel wird im nächsten Kapitel eine wichtige Rolle spielen. Es wird verwendet um das Programm zu erklären und zu testen.[10]

Seien die folgenden Formeln gegeben:

1. Die Taste 1 gehört zur Tastatur:

$$taste1 \sqsubseteq tastatur \qquad (\phi_1)$$

2. Tastatur, Maus und Scanner sind Eingabegeräte:

$$(tastatur \sqcup maus) \sqcup scanner \sqsubseteq eingabegerät \qquad (\phi_2)$$

3. Bildschirm, Drucker und Lautsprecher sind Ausgabegeräte:

$$(bildschirm \sqcup drucker) \sqcup lautsprecher \sqsubseteq ausgabegerät \qquad (\phi_3)$$

4. Existieren mindestens ein Eingabe- und ein Ausgabegerät, so kann von einem Computer gesprochen werden:

$$\exists hatTeil.eingabegerät \sqcap \exists hatTeil.ausgabegerät \sqsubseteq computer \qquad (\phi_4)$$

Natürlich werden noch andere Bestandteile wie Prozessor oder Speicher benötigt, bis von einem Computer geredet werden kann. Jedoch soll einfachheitshalber davon ausgegangen werden, dass für einen Computer ein Eingabe- sowie ein Ausgabegerät ausreicht.
Aus den obigen Formeln folgt, dass, falls es eine Taste 1 und einen Lautsprecher gibt, bereits von einem Computer gesprochen werden kann.

$$\phi_1, \phi_2, \phi_3, \phi_4 \vdash \exists hatTeil.taste1 \sqcap \exists hatTeil.lautsprecher \sqsubseteq computer$$

[10] Vgl. Abbildung 3.5, Seite 20

Folgende Axiome werden benutzt (seien A, B, C, D Konzepte und R ein Rollenname):

$$[a]((A \sqcup B) \sqcup C \sqsubseteq D \rightarrow A \sqcup (B \sqcup C) \sqsubseteq D)$$
$$[b](A \sqcup B \sqsubseteq C \rightarrow A \sqsubseteq C)$$
$$[c](A \sqsubseteq B \rightarrow (C \sqsubseteq A \rightarrow C \sqsubseteq B))$$
$$[d](A \sqcup B \sqsubseteq C \rightarrow B \sqcup A \sqsubseteq C)$$
$$[e](A \sqsubseteq B \rightarrow \exists R.A \sqsubseteq \exists R.B)$$
$$[f](A \sqsubseteq B \rightarrow (C \sqsubseteq D \rightarrow A \sqcap C \sqsubseteq B \sqcap D))$$

Um die Herleitung erstellen zu können, werden noch die Variablen den vier Formeln hinzugefügt, damit klar ist, welches Axiom auf welche Formel angewendet wird:

$$[v](taste1 \sqsubseteq tastatur)$$
$$[w]((tastatur \sqcup maus) \sqcup scanner \sqsubseteq eingabegerät)$$
$$[x]((bildschirm \sqcup drucker) \sqcup lautsprecher \sqsubseteq ausgabegerät)$$
$$[y](\exists hatTeil.eingabegerät \sqcap \exists hatTeil.ausgabegerät \sqsubseteq computer)$$

Herleitung:

$$[c \cdot y](\exists hatTeil.taste1 \sqcap \exists hatTeil.lautsprecher \sqsubseteq \exists hatTeil.eingabegerät \sqcap \exists hatTeil.ausgabegerät \rightarrow \exists hatTeil.taste1 \sqcap \exists hatTeil.lautsprecher \sqsubseteq computer)$$
$$[a \cdot w](tastatur \sqcup (maus \sqcup scanner) \sqsubseteq eingabegerät)$$
$$[b \cdot (a \cdot w)](tastatur \sqsubseteq eingabegerät)$$
$$[c \cdot (b \cdot (a \cdot w))](taste1 \sqsubseteq tastatur \rightarrow taste1 \sqsubseteq eingabegerät)$$
$$[(c \cdot (b \cdot (a \cdot w))) \cdot v](taste1 \sqsubseteq eingabegerät)$$
$$[e \cdot ((c \cdot (b \cdot (a \cdot w))) \cdot v)](\exists hatTeil.taste1 \sqsubseteq \exists hatTeil.eingabegerät)$$
$$[f \cdot (e \cdot ((c \cdot (b \cdot (a \cdot w))) \cdot v))](\exists hatTeil.lautsprecher \sqsubseteq \exists hatTeil.ausgabegerät \rightarrow \exists hatTeil.taste1 \sqcap \exists hatTeil.lautsprecher \sqsubseteq \exists hatTeil.eingabegerät \sqcap \exists hatTeil.ausgabegerät)$$
$$[d \cdot x](lautsprecher \sqcup (bildschirm \sqcup drucker) \sqsubseteq ausgabegerät)$$
$$[b \cdot (d \cdot x)](lautsprecher \sqsubseteq ausgabegerät)$$
$$[e \cdot (b \cdot (d \cdot x))](\exists hatTeil.lautsprecher \sqsubseteq \exists hatTeil.ausgabegerät)$$
$$[(f \cdot (e \cdot ((c \cdot (b \cdot (a \cdot w))) \cdot v))) \cdot (e \cdot (b \cdot (d \cdot x)))](\exists hatTeil.taste1 \sqcap \exists hatTeil.lautsprecher \sqsubseteq \exists hatTeil.eingabegerät \sqcap \exists hatTeil.ausgabegerät)$$
$$[(c \cdot y) \cdot ((f \cdot (e \cdot ((c \cdot (b \cdot (a \cdot w))) \cdot v))) \cdot (e \cdot (b \cdot (d \cdot x))))] (\exists hatTeil.taste1 \sqcap \exists hatTeil.lautsprecher \sqsubseteq computer)$$

Wiederum liegt die hergeleitete Formel eigentlich auf der Hand. So ist es eher überraschend, dass die Herleitung derartig kompliziert ist. Der Grund liegt darin, dass für den Betrachter logische Axiome wie die Assoziativität (siehe Axiom mit Konstante a) als klar erscheinen. Jedoch dürfen diese Herleitungsschritte nicht fehlen, weil die maschinelle Verarbeitung einen solchen Schritt nicht ohne Aufforderung ausführt und zu einem anderen Ergebnis oder einem Fehler führen würde.
Eine andere Schwierigkeit ist repräsentative Namen für die Konzept- und Rollennamen auszuwählen, so dass alleine anhand der Formel klar ist, was damit gemeint ist.
Auf weitere Details und Probleme, die beim Erstellen einer solchen Herleitung entstehen, wird im nächsten Kapitel vertiefter eingegangen und gleich in den Zusammenhang mit dem Programm gebracht.

Kapitel 3

Java-Programm zum Erstellen von Herleitungen in $\mathcal{JALC}$

In diesem Kapitel wird auf die in der Programmierungssprache Java geschriebene Applikation eingegangen. Das Ziel des Programmes ist, aus dem Beweisterm der Endformel die Herleitung automatisch zu rekonstruieren und die Erklärungen für jeden Herleitungsschritt dynamisch anzupassen.
Als erstes muss geklärt werden, was als gegeben angesehen wird. Anschliessend folgen verschiedene Schwierigkeiten, die beim Programmieren beachtet werden müssen. In einem weiteren Teil wird auf das dynamische Erstellen und Anpassen von Erklärungen eingegangen. Anhand von Screenshots wird das Programm präsentiert und aufgezeigt welche Möglichkeiten bei dessen Benutzung zur Verfügung stehen.

3.1 Gegebene Parameter

Zu den gegebenen Parametern gehören:

1. Ausgangsformeln ($\mathcal{L}_{\mathcal{J}}$ Formeln)
2. Axiome
3. Hergeleitete Endformel

Theoretisch kann auch die Endformel weggelassen werden, weil nur der Term davon benötigt wird. Dieser Term muss unter Umständen von zusätzlichen Informationen ergänzt werden. Dies ist der Fall, wenn Axiome noch nicht definierte Konzepte anwenden. Zum Beispiel weiss die Maschine, die das Programm verarbeitet, beim Axiom [a](A$\sqsubseteq$B$\rightarrow$(C$\sqsubseteq$A$\rightarrow$C$\sqsubseteq$B)) nicht, was C ist.

In diesem Programm werden diese Informationen direkt dem Term der hergeleiteten Endformel mitgegeben:
Sei [v]ϕ eine $\mathcal{L}_\mathcal{J}$ Formel. Ist der Term [a·v] so wird er zu [a:C:·v] ergänzt. Unter Umständen können auch mehrere unbekannte Konzepte in einem Axiom vorkommen. Der ergänzte Term sieht dann folgendermassen aus: [a:C,D,E,...:·v].
Zum Beispiel sei [v](tastatur$\sqsubseteq$eingabegerät) eine $\mathcal{L}_\mathcal{J}$ Formel. Dann sieht der Term statt [a·v] nun z.B. [a:taste1:·v] aus und ergibt insgesamt die Formel [a:taste1:·v](taste1$\sqsubseteq$tastatur$\rightarrow$taste1$\sqsubseteq$eingabegerät).

Nach dieser Erläuterung wurden alle Symbole (ausser { und }) eingeführt, die in den Konzept- und Rollennamen nicht vorkommen dürfen: ( ) [ ] { } · . , : $\sqsubseteq$ $\sqcup$ $\sqcap$ $\exists$ $\forall$ $\rightarrow$. { und } wird benötigt, um die Erklärung von der $\mathcal{L}_\mathcal{J}$ Formel zu trennen.

Eine Schwierigkeit ist alle Möglichkeiten an Termen abzudecken. Besteht ein Term nur aus Axiomen (z.B. [a·b]), scheint er auf den ersten Blick als nicht zulässig, weil die Axiome meistens aus allgemein gefassten Konzepten (A, B, usw.) bestehen. Jedoch funktioniert dieser Term z. B. mit den Axiomen:

$$[a](A \sqsubseteq B \rightarrow (C \sqsubseteq A \rightarrow C \sqsubseteq B))$$
$$[b](\mathit{computer} \sqsubseteq \top)$$

einwandfrei und liefert

$$[a \cdot b](\mathit{taste}1 \sqsubseteq \mathit{computer} \rightarrow \mathit{taste}1 \sqsubseteq \top)$$

wobei der Term, wie vorher erläutert, im Programm [a:taste1:·b] wäre.
Weiter kann festgestellt werden, dass diese speziellen Axiome ohne ein anderes Axiom nicht funktionieren. So kann das Axiom mit der Konstante b auf keine Formel angewendet werden, sondern wird von einem anderen Axiom verwendet. Deshalb sollen diese Spezialformen an die Liste der gegebenen Anfangsformeln angefügt werden.

3.2 Automatische Rekonstruktion der Herleitung

Um zu erreichen, dass die Herleitung automatisch rekonstruiert werden kann, braucht es die gegebenen Parameter (siehe vorheriger Abschnitt).
Anfangs wird aber nur der Term der Endformel benötigt. Anhand der Anzahl an Klammern kann herausgefunden werden, wie viele Herleitungsschritte es insgesamt braucht. Gibt es keine Klammern, bedeutet dies, dass es bereits die Endformel ist.
Damit die Erläuterungen verständlicher sind, soll der Term der Endformel von Beispiel 2 (Definition eines Computers) auf Seite 7 benutzt werden:

$$(c \cdot y) \cdot ((f \cdot (e \cdot ((c \cdot (b \cdot (a \cdot w))) \cdot v))) \cdot (e \cdot (b \cdot (d \cdot x))))$$

Dieser beinhaltet 11 Klammern. Somit wird die Endformel nach 11 Herleitungsschritten erreicht bzw. gibt es 11 Herleitungsformeln. Wird auch die Endformel für die Anzahl Formeln hinzugezogen, existieren gleich viele Formeln wie Multiplikationszeichen.

Um zu gewährleisten, dass die Erstellung der Herleitung strukturiert abläuft, erfolgt die Herleitung in der Reihenfolge der abschliessenden Klammern (')') von links nach rechts. D.h. im vorherigen Term wird zuerst der Herleitungsschritt (c·y) konstruiert. Danach folgen in der Klammernreihenfolge (a·w), dann (b·(a·w)), usw. Dadurch kann eine übersichtliche Darstellung der Herleitung in einer logischen Reihenfolge erzielt werden. Ausserdem ist es selbsterklärend, dass ohne (a·w) der Herleitungsschritt (b·(a·w)) nicht konstruiert werden kann, was zusätzlich die gewählte Reihenfolge unterstreicht. Somit sind alle Terme der Herleitungsschritte herausfindbar und werden von nun an als gegeben betrachtet.
Diese Terme können in zwei Teile aufgeteilt werden. Links vom Multiplikationszeichen (das in keiner Klammer ist, d.h. beim Term b·(a·w) ersteres) befindet sich das Axiom bzw. der Term davon. Im rechten Teil befindet sich der Term der Anfangsformel (z.B. v) oder der Formel des vorderen Herleitungsschrittes (z.B. (a·w)). Dadurch wissen wir welches Axiom auf welche Formel angewendet wird. Im Beispielsterm b·(a·w) wird das Axiom mit der Konstanten b auf die Formel mit dem Term (a·w) angewendet.

Als nächster Schritt folgt das Ausrechnen der neuen Formel. Dazu muss zuerst auf die Struktur eines Axiomes eingegangen werden.
Ein Axiom ist von der Form $(\phi \rightarrow \psi)$, wobei ϕ und ψ $\mathcal{L}_{\mathcal{J}}$ Formeln sind. Es gibt einige Ausnahmen, die nicht diese Form besitzen. Ein Beispiel wäre wiederum (computer $\sqsubseteq \top$). Jedoch werden diese speziellen Axiome im Programm den gegebenen Formeln angerechnet.[1] Die anzuwendende Formel muss dieselbe Struktur wie ϕ haben, sonst führt es zu einem Fehler. So kann die Formel (tastatur$\sqsubseteq$eingabegerät) nicht auf das Axiom (A$\sqcup$B$\sqsubseteq$C$\rightarrow$A$\sqsubseteq$C) angewendet werden, weil nicht bekannt ist, ob die tastatur A oder B ist. Hier könnte nur eine Formel mit einem $\sqcup$, wie beispielsweise (tastatur$\sqcup$maus$\sqsubseteq$eingabegerät), benutzt werden.
Das Programm orientiert sich anhand der Spezialsymbole $\sqcup$, $\sqcap$, $\sqsubseteq$, $\rightarrow$, $\exists$, $\forall$, '.'. So wird unter dem Konzept A tastatur gespeichert, B=maus und C=eingabegerät. Diese Werte können im zweiten Teil des Axiomes eingesetzt werden und die Formel ist erstellt.
Für das Programm selbst bildet der Modus Ponens einen Spezialfall. In diesem Fall müssen keine Konzepte zugewiesen werden. Anstelle eines Axiomes wird zusätzlich eine Formel benutzt. Es reicht als neue Formel den Teil nach der Implikation ($\rightarrow$) zu nehmen, weil die Terme der beiden verwendeten Formeln Subterme der neuen Formel sind und somit bereits hergeleitet wurden. Dies ist besser in der Darstellung des Modus Ponens ersichtlich: $\frac{\phi \quad \phi \rightarrow \psi}{\psi}$. ϕ und $\phi \rightarrow \psi$

[1] Vgl. letzter Teil von Abschnitt 3.2

sind gegeben oder bereits hergeleitet und somit gilt auch ψ, was der Teil nach der Implikation in $\phi \rightarrow \psi$ ist.
Anhand dieses Verfahrens kann die vollständige Herleitung der Endformel gemacht werden.

3.3 Dynamische Erklärungen der Herleitungsschritte

Damit der Betrachter nicht nur die in der Logik $\mathcal{JALC}$ beschriebenen Ausdrücke, die alle $\mathcal{L}_{\mathcal{J}}$ Formeln sind, verstehen muss, gibt es die Erklärungen. Zu jedem Herleitungsschritt hat der Betrachter die Möglichkeit durch Rechtsklick auf die Formel eine verständlichere Erklärung zu erhalten.
Es stellt sich die Frage, weshalb diese dynamisch sein sollen. Der Grund liegt darin, dass der Nutzer die Herleitungsschritte nach eigenen Belieben aufklappen kann. Ist die ganze Herleitung offen, fällt die Erklärung exakter aus, weil die mit der Formel direkt verbundenen Herleitungsschritte ersichtlich sind. Fehlen jedoch diese Herleitungsschritte vor der Formel, muss die Erklärung angepasst werden. Diese muss in diesem Fall allgemeiner ausfallen und vor allem auf den gegebenen Anfangsformeln basieren.
Weil die Anzahl an Erklärungen pro Formel bzw. Herleitungsschritt direkt mit Subtermen verbunden ist, sollen diese anhand von Beispielen definiert werden, wobei der Begriff Subterm im folgenden Abschnitt alle atomaren Subterme wie a und v ausschliessen soll.
Während [a·v] keinen direkten Subterm besitzt, hat der Term [(a·v)·(b·w)] gleich zwei direkte Subterme: [a·v] und [b·w]. Lautet der Term [(a·(b·v))·w], so ist der direkte Subterm [a·(b·v)], wobei dieser wiederum den direkten Subterm [b·v] besitzt. Hierbei sei auf den Begriff "direkt"hingewiesen. Würde es sich nicht um direkte Subterme handeln, wäre [b·v] auch ein Subterm von [(a·(b·v))·w].
Dementsprechend folgt die Anzahl Erklärungen:

Sei s die Anzahl direkter Subterme. s liegt offensichtlich zwischen 0 und 2, weil jeder Term aus maximal zwei direkten Subtermen besteht.
s=0: 1 Erklärung
s=1: 2 Erklärungen
s=2: 4 Erklärungen

Für s=1 gibt es zwei verschiedene Erklärungen. Eine, wenn der direkte Subterm bzw. die Formel, auf der der jetzige Herleitungsschritt aufbaut, ersichtlich ist, sowie eine, wenn der direkte Subterm nicht geöffnet ist. Bei s=2 existieren Erklärungen, falls beide direkten Subterme offen bzw. geschlossen sind und falls je ein direkter Subterm offen und einer geschlossen ist.

Jede Erklärung besteht aus drei Teilen (ϕ, ψ und α sind die Formeln ohne den Term):

$$Folgt\ aus \begin{cases} \phi \\ \phi,\ \psi \\ \phi,\ den\ gegebenen\ Anfangsformeln \\ den\ gegebenen\ Anfangsformeln \end{cases} \begin{cases} und\ dem\ Axiom\ \alpha. \\ und\ dem\ Modus\ Ponens. \\ und\ logischen\ Axiomen. \end{cases}$$

Diese Teile werden verschiedenartig miteinander kombiniert. Jedoch kommen nicht alle Kombinationen vor. Um die möglichen Erklärungen zu erhalten, ist es am einfachsten nach den Termen zu ordnen.

1. Der Term [a·v] hat keinen direkten (nicht atomaren) Subterm und somit nur eine Erklärung (gilt auch für [a·b]):

   $$Folgt\ aus\ Fm(v)\ und\ dem\ Axiom\ Fm(a).$$

   Mit Fm(v) ist die Formel mit dem Term v gemeint, jedoch ohne den Term auszugeben. Das heisst, bei [v](ϕ) ist Fm(v) gleichbedeutend mit (ϕ). Die Erklärungen sind hier allgemein (z.B. Fm(v)) gefasst. Im nächsten Abschnitt folgen Screenshots, die die Anwendung verdeutlichen. Die verschiedenen Erklärungen für ein und dieselbe Formel sind in Abbildung 3.4 (Seite 19) ersichtlich.

2. Für den Term [a·t] gibt es zwei Erklärungen, wobei t ein direkter Subterm ist, z.B. (b·v).

   (a) Falls der Subterm t geöffnet ist:

   $$Folgt\ aus\ Fm(a \cdot t)\ und\ dem\ Axiom\ Fm(a).$$

   (b) Falls der Subterm t geschlossen ist:

   $$Folgt\ aus \begin{cases} gFm(t) \\ den\ gegebenen\ Anfangsformeln \end{cases} \Big\} \begin{array}{l} und\ logischen \\ Axiomen. \end{array}$$

   Wobei im mittleren Teil zwei verschiedene Erklärungen sind, je nachdem, wie viele der gegebenen Ausgangsformeln von t (gFm(t)) vorkommen. Werden mehr als zwei Ausgangsformeln benutzt, wird die allgemeinere zweite Variante ausgegeben (z.B. bei t=((b·v)·(w·x)) kommen die drei Anfangformeln v, w, x vor). Ist t=(v·w) wird die erste Variante gFm(v·w) ausgegeben, was gleichbedeuted mit Fm(v), Fm(w) ist (vgl. 1.).

3. Zuletzt folgt noch der Fall mit dem Modus Ponens. Hierbei gibt es vier verschiedene mögliche Terme: [v·w], [v·t], [s·w] und [s·t], wobei s und t nicht atomare Subterme sind.
   Jedoch muss nur der Term [s·t] untersucht werden, weil z.B. [v·t] dieselbe Erklärung hat wie [s·t], wenn der Subterm s geöffnet ist, ausser dass Fm(v) statt Fm(s) ausgegeben wird.

(a) Falls die Subterme s und t geöffnet sind:

$$Folgt\ aus\ Fm(s),\ Fm(t)\ und\ dem\ Modus\ Ponens.$$

(b) Falls der Subterm s geöffnet ist:

$$Folgt\ aus\ Fm(s),\ \left\{\begin{array}{l} gFm(t) \\ den\ gegebenen\ Anfangsformeln \end{array}\right\}\ \begin{array}{l} und \\ logischen \\ Axiomen. \end{array}$$

(c) Falls der Subterm t geöffnet ist:

$$Folgt\ aus\ Fm(t),\ \left\{\begin{array}{l} gFm(s) \\ den\ gegebenen\ Anfangsformeln \end{array}\right\}\ \begin{array}{l} und \\ logischen \\ Axiomen. \end{array}$$

(d) Falls die Subterme s und t geschlossen sind:

$$Folgt\ aus\ \left\{\begin{array}{l} gFm(s), gFm(t) \\ den\ gegebenen\ Anfangsformeln \end{array}\right\}\ \begin{array}{l} und \\ logischen \\ Axiomen. \end{array}$$

Bei 3.(d) werden die gegebenen Anfangsformeln, die in den beiden Subtermen s und t vorkommen, zusammengezählt und falls mehr als zwei vorhanden sind, die zweite Variante gewählt.

3.4 Programmfunktionen

Nachdem beschrieben wurde, wie die Herleitung und die Erklärungen konstruiert werden, soll in diesem Abschnitt das Programm anhand von Screenshots vorgestellt werden.

Als erstes muss sichergestellt sein, dass die Formeln (und Axiome) richtig eingetippt wurden. Diese werden in der Java-Klasse Main eingebunden. So sieht die gesamte Endformel im Programmcode folgendermassen aus (aus Beispiel 2, Seite 7):

endFormula = "[(c:"+Char.EXISTS+"hatTeil.taste1"+Char.CAPAND+
Char.EXISTS+"hatTeil.lautsprecher:*y)*((f:"+Char.EXISTS+
"hatTeil.lautsprecher,"+Char.EXISTS+ "hatTeil.ausgabegerät:*(e:hatTeil:
((c:taste1:(b*(a*w)))*v)))*"+"(e:hatTeil:*(b*(d*x))))]
("+Char.EXISTS+"hatTeil.taste1"+Char.CAPAND+Char.EXISTS+
"hatTeil.lautsprecher"+Char.CAPTORIGHT+"computer)";

Diese Formel macht einen komplizierten Eindruck. Jedoch sind alle Spezialsymbole in Worten ausgeschrieben (z.B. Char.CAPTORIGHT statt $\sqsubseteq$), wodurch sich die Formel sehr verkürzen würde. Die ausgeschriebene Variante erleichtert die Eingabe deutlich gegenüber den Hexadezimalwerten der Symbole. Wichtig zu überprüfen ist am Schluss der Eingabe jeweils die Anzahl Klammern, sowie ob die zusätzlichen Informationen für Axiome vorhanden sind[2]. Das Programm überprüft verschiedene Fehlerquellen, wie vergessene Klammern, jedoch kann es nicht alle Eingabemöglichkeiten untersuchen.

Wird das Programm gestartet, sieht das GUI[3] wie auf Abbildung 3.1 aus.

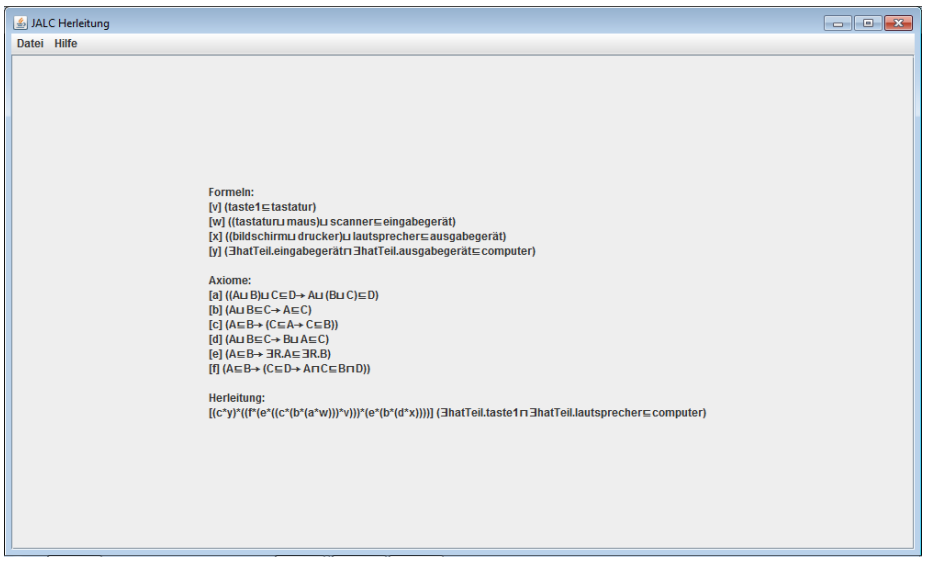

Abbildung 3.1: Screenshot nach dem Start des Programmes

[2] Vgl. Abschnitt 3.1, Seite 10
[3] GUI: Graphical User Interface

Damit der Inhalt besser zu lesen ist, folgt noch eine Vergrösserung der Abbildung 3.1:

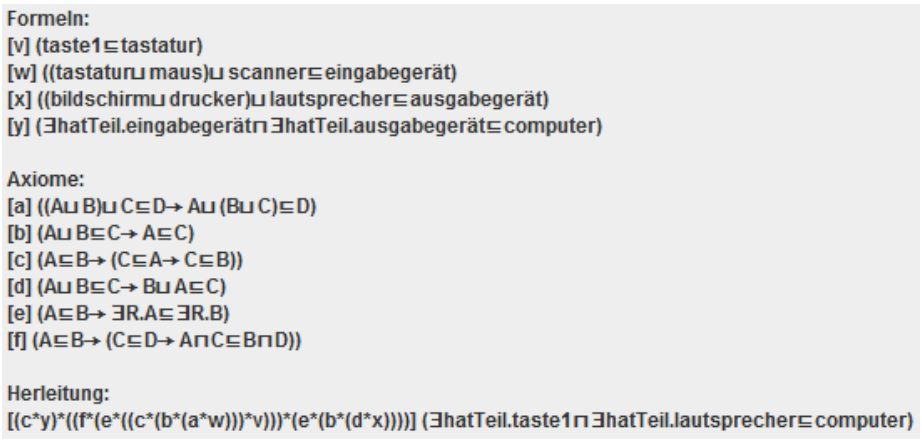

Abbildung 3.2: Vergrösserung der Abbildung 3.1

In Abbildung 3.2 sind nun die drei Teile: Anfangsformeln, Axiome und Herleitung (bzw. anfangs nur die Endformel) ersichtlich.

Wird die Maus über die Terme der Herleitungsformeln gezogen, verfärben sich die Teile blau, die genauer betrachtet werden können. Zum einen, falls ein direkter Subterm geöffnet werden kann, zum anderen, falls es kein Subterm gibt, aber die Erklärung sich öffnen bzw. schliessen lässt (bei Termen der Form [a·v]). Ein Herleitungsschritt wird mit Linksklick auf den blau eingefärbten Subterm geöffnet. Das Öffnen bzw. Schliessen der Erklärungen einer Formel erfolgt durch Rechtsklick auf einen blau eingefärbten Teil des Termes. Die verschiedenen Möglichkeiten sind im Programm unter Hilfe und anschliessend unter "Anleitung"noch einmal erklärt.

Eine teilweise aufgeklappte Herleitung zeigt Abbildung 3.3[4], wobei der blau eingefärbte Teil der Subterm ist, über dem sich die Maus gerade befindet:

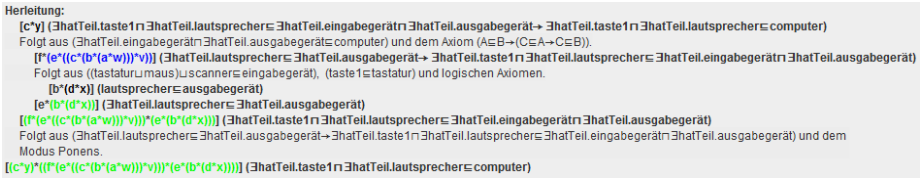

Abbildung 3.3: Beispiel einer teilweise geöffneten Herleitung

[4]In Abbildung 3.5 ist die ganze Herleitung vergrössert und besser lesbar vorzufinden

Die verschiedenen Möglichkeiten an Erklärungen für eine Formel bzw. für einen Herleitungsschritt ist in Abbildung 3.4 wiedergegeben.

Damit die ganze Herleitung betrachtet werden kann ohne alles mit etlichen Mausklicken öffnen zu müssen, gibt es unter Datei den Menüpunkt "Ganze Herleitung". Es zeigt die ganze Herleitung mit ihren Erklärungen (Abbildung 3.5) an. Um die Herleitung wieder zu schliessen, so dass nur noch die Endformel angezeigt wird, existiert der Menüpunkt "Herleitung schliessen". Danach sieht das Frame wieder gleich wie nach dem Starten des Programmes aus.

Abbildung 3.4: Verschiedene Erklärungen für eine Formel (hier ein Beispiel mit dem Modus Ponens)

Abbildung 3.5: Ganze Herleitung (vgl. Beispiel 2, Seite 7)

Kapitel 4

Zusammenfassung

Das Ziel dieser Arbeit war, ein Programm zu erstellen, das automatisch Herleitungen rekonstruiert und dem Benutzer ermöglicht nach eigenem Ermessen gewisse Teile der Herleitung anzuzeigen.

Als Programmiersprache wurde Java verwendet. Nachdem verschiedene Spezialfälle bei der automatischen Rekonstruktion der Herleitung beachtet, sowie vorhergehende Überlegungen und Erweiterungen gemacht werden mussten[1], funktioniert das Herleiten. Möglicherweise möchte der Benutzer anstelle der nackten Endformel die Zwischenschritte betrachten können. Dazu wurde das Programm so aufgebaut, dass der Benutzer interaktiv nach eigenen Belieben die Herleitung aufklappen bzw. genauer betrachten kann. Dadurch wird das Problem gelöst, dass mehrere Betrachter verschiedene Schwierigkeiten beim Verstehen einer Herleitung haben. Jeder kann sich die für ihn unklaren Herleitungsschritte genauer anschauen.
Als zusätzliche Hilfe und damit die Formeln in verständlicheren Worten formuliert werden, gibt es die Erklärungen zu jedem Herleitungsschritt. Diese passen sich dynamisch an, je nachdem wie viel der Herleitung ersichtlich ist.

Zu verbessern wäre eventuell die Eingabemöglichkeit der Formeln für den Nutzer des Programmes. Das Eingeben einer Formel ist ziemlich mühsam, weil viele Spezialzeichen wie $\sqsubseteq$ benutzt werden. Abbildung 4.1 zeigt eine Idee eines Frames, bei dem die Sonderzeichen durch Buttons in einer Formel hinzugefügt werden.

[1] Vgl. Abschnitte 3.1 und 3.2

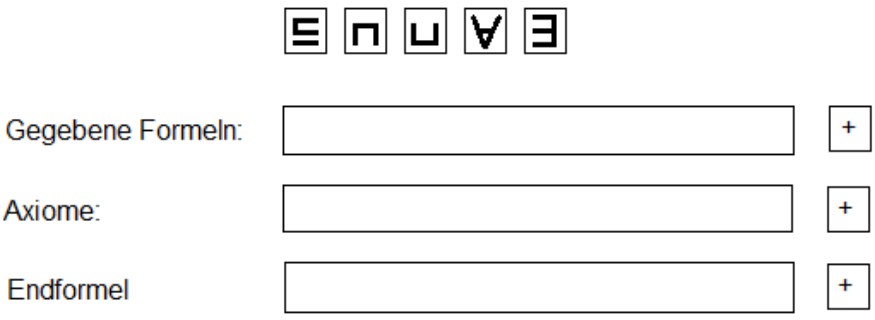

Abbildung 4.1: Frame für einfachere Eingabe der Formeln

Abbildungsverzeichnis

Literaturverzeichnis

[1] S. Artemov. The Logic of Justification. *The Review of Symbolic Logic*, 1(4): 477-513. Graduate Center CUNY, Dezember 2008.

[2] F. Baader, W. Nutt. *Basic Description Logics*. S. 43-95. Cambridge University Press, New York, USA, 2003.

[3] T. Berners-Lee, J. Hendler, O. Lassila. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. In *Scientific American Special Online Issue*, S. 24-30. Scientific American, Inc., 2002.

[4] P. Bouquet, H. Stoermer, M. Vignolo. Web of Data and Web of Entities: Identity and Reference in Interlinked Data in the Semantic Web. In *Philosophy and Technology*, S. 1-22. Springer, Netherlands, 2011.

[5] S. Bucheli, R. Kuznets, B. Renne, J. Sack, T. Studer. Justified Belief Change. In *Proceedings of the Second ILCLI International Workshop on Logic and Philosophy of Knowledge, Communication and Action (LogKCA-10)*, S. 135-155. University of the Basque Country Press, 2010.

[6] S. Bucheli, R. Kuznets, T. Studer. Justifications for Common Knowledge. In *Journal of Applied Non-classical Logics*, 21(35): 35-60. 2011.

[7] S. Bucheli, R. Kuznets, T. Studer. Partial Realization in Dynamic Justification Logic. In *Logic, Language, Information and Computation, 18th International Workshop, WoLLIC 2011, Philadelphia, PA, USA, May 18-20, 2011, Proceedings*, S. 35-51. 2011.

[8] T. Studer. Justified Terminological Reasoning. In *Proceedings of the 8th Andrei Ershov Informatics Conference*, S. 177-185. Ershov Institute of Informatics Systems, Springer, 2011.

Selbständigkeitserklärung

„Ich erkläre hiermit, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 20 Absatz 2a des Gesetzes vom 5. September 1996 über die Universität zum Entzug des aufgrund dieser Arbeit verliehenen Titels berechtigt ist.“

Schüpbach, 09.01.2012 Roger Kohler