UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

Matthias Thimm*

Verteilte logikbasierte Argumentation unter Berücksichtigung unsicheren Wissens mit Anwendung auf ein Beispiel aus dem Rechtswesen

Diplomarbeit

27. März 2007

*matthias.thimm@uni-dortmund.de, Matr. Nr. 96021

INTERNE BERICHTE
INTERNAL REPORTS

Lehrstuhl 6 (Information Engineering)
Fachbereich Informatik
Universität Dortmund

Gutachter:
Gabriele Kern-Isberner, Universität Dortmund
Christoph Beierle, FernUniversität Hagen

GERMANY · D-44221 DORTMUND

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich bei der Anfertigung dieser Diplomarbeit unterstützt haben. Im Besonderen bedanke ich mich bei Fr. Prof. Dr. Kern-Isberner, die durch ihre Anregung diese Diplomarbeit initiiert hat sowie bei Hr. Prof. Dr. Beierle für seine Bereitschaft als Zweitgutachter zu dienen.

Weiterhin bedanke ich mich für allgemeine Unterstützung bei Dipl. Inf. Manuela Mark und bei Isabel Breiter, die mir außerdem auch in Rechtsfragen zur Seite stand.

Außerdem bedanke ich mich bei Christian Bredemeier, Fiona Scheiwe, Margarete Geier, Stefan Tittel und Rainer Bressel, die die Arbeit korrekturgelesen und mir nützliche Tipps gegeben haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Algorithmen und Listings

1 Einleitung

1.1 Motivation

Die Realisierung oder Nachahmung von *commonsense-reasoning* ist einer der wichtigsten Forschungszweige innerhalb der Künstlichen Intelligenz (KI) [CnML00]. Software-Agenten sollen in der Lage sein, Entscheidungen möglichst auf dieselbe Art und Weise zu treffen wie der menschliche Geist. Auch wenn eine tatsächliche Realisierung solcher Denkprozesse noch in weiter Zukunft liegt, können bestimmte Eigenschaften des menschlichen Schlußfolgerungsprozesses helfen, Software-Agenten zu „realistischeren“ Methoden der Deliberation zu verhelfen. Eine wichtige Eigenschaft des menschlichen Schlußfolgerungsprozesses ist die Nicht-Monotonie: Regeln können außer Kraft gesetzt werden, wenn neues Wissen gegen die Anwendung dieser Regel spricht. Hier sei das bekannte Pinguin-Beispiel genannt:

> Tweety ist ein Pinguin.
> Pinguine sind Vögel.
> Vögel fliegen normalerweise.
> Pinguine fliegen nicht.

Zur Beantwortung der Frage „Fliegt Tweety?“ reicht eine Darstellung in einer klassischen Logik nicht aus. Die Regel „Pinguine fliegen nicht“ soll die Anwendung der Regel „Vögel fliegen normalerweise“ für einen Spezialfall von Vögeln (Pinguine) überschreiben. Realisiert werden kann dies durch eine Unterscheidung zwischen immer geltenden „sicheren Regeln“ (*strict rules*) und „unsicheren“ oder „anfechtbaren Regeln“ (*defeasible rules*), die stets dann angewendet werden dürfen, wenn es keine Gründe gibt, etwas anderes zu glauben. Wird in dem obigen Beispiel die Regel „Vögel fliegen normalerweise“ als eine unsichere Regel und „Pinguine fliegen nicht“ als eine sichere Regel modelliert, so fliegt ein Vogel nur dann, wenn nichts dagegen spricht, er also z. B. kein Pinguin ist. Wie das Beispiel zeigt ist Inkonsistenz der Wissensbasis auf klassisch-logischer Ebene eine Folge der Verwendung unsicherer Regeln. In der KI wurden bisher viele Formalismen erarbeitet, die sich mit Verarbeitung und Behandlung von unsicherem Wissen sowie der Auflösung der daraus entstehenden Inkonsistenzen beschäftigen. Der formale Ansatz der logikbasierten Argumentation entstand als ein solcher Formalismus zur Realisierung von nicht-monotoner Inferenz [CnML00].

In dieser Arbeit werden zwei verschiedene Ansätze zur Realisierung logikbasierter Argumentation vorgestellt. Während der Ansatz von García und Simari [GS02] die explizite

Verwendung der oben genannten unsicheren Regeln unterstützt, benutzt der Ansatz von Besnard und Hunter [BH06] eine klassisch-propositionallogische Syntax. In beiden Ansätzen ist ein Argument eine Menge von Wissensfragmenten, die die Inferenz – entweder mit Hilfe klassisch-logischer Deduktion oder durch Anwendung von sicheren und unsicheren Regeln – eines weiteren Wissensfragments (die Konklusion) erlauben. Ein Argument kann von anderen Argumenten *angegriffen* werden, die Gründe für eine Gegenaussage in der Inferenzkette des ersten Arguments liefern. Argumente können schlechter oder besser als andere Argumente sein und durch einen Vergleich von sich widersprechenden Argumenten kann das „beste“ Argument bestimmt und dessen Konklusion von dem Agenten geglaubt werden. Wird im Pinguin-Beispiel die Regel „Pinguine fliegen nicht“ als eine unsichere Regel modelliert (weil man sich vielleicht nicht sicher ist, ob es nicht doch fliegende Pinguine gibt), so existiert jeweils ein Argument für „Tweety fliegt“ und für „Tweety fliegt nicht“. Durch einen Vergleich der unsicheren Regeln dieser Argumente kann bestimmt werden, welches Argument besser als das andere ist und welche Aussage somit geglaubt wird. Die Spezifizität von Argumenten kann als ein Vergleichskriterium dienen. Während das Argument für „Tweety fliegt“ lediglich die Tatsache benutzt, dass es sich bei Tweety um einen Vogel handelt, wird bei dem Argument für „Tweety fliegt nicht“ die „stärkere“ Information, dass Tweety auch ein Pinguin ist, benutzt (Diese Information ist „stärker“, da Pinguine ein Spezialfall von Vögeln sind). In diesem Fall wird das Argument „Tweety fliegt nicht“ als das stärkere Argument angesehen und die Konklusion somit geglaubt.

Argumentation kann sowohl als ein interner Deliberationsmechanismus eines Agenten verstanden werden, der zur Ermittlung einer Wissensmenge dient, als auch als ein Kommunikationsprozess zwischen verschiedenen Agenten eines Multiagentensystems, bei dem Agenten Argumente für oder gegen eine Anfrage an das System austauschen. Während bei dem ersten Fall die Bildung einer „Meinung“ (über eine logische Aussage) eines Agenten im Vordergrund steht, geht es in dem zweiten Fall darum, „Meinungen“ verschiedener Agenten zueinander in Beziehung zu setzen. Bisher wurde in den meisten argumentativen Systemen, wie in dem von García und Simari [GS02], Argumentation als ein lokaler Deliberationsmechanismus angewandt. Diese Diplomarbeit beschäftigt sich mit der Konzeption und Realisierung eines verteilten Systems für argumentative Prozesse. Es soll ein System modelliert werden, in dem Agenten Argumente und Gegenargumente austauschen, um Anfragen an das System zu beantworten. Ein Anwendungsszenario aus dem Rechtswesen soll den Prozess der Argumentation in einem verteilten System verdeutlichen. Ausgezeichnete Agenten übernehmen dabei die Rolle des Für- bzw. des Gegensprechers in einem Rechtsfall. Durch den Austausch von Argumenten, die auf der Grundlage einer gesetzmäßigen Verfassung beruhen, soll über den Ausgang des Rechtsfalls entschieden werden. Eine den Agenten übergeordnete Instanz – der *Moderator* – koordiniert den Argumentationsprozess und entscheidet anschließend auf der Grundlage der ausgetauschten Argumente, ob die dem System gestellte Anfrage bejaht oder verneint wird.

1.2 Problemstellung

Das Thema dieser Diplomarbeit ist es, logikbasierte Argumentation in einem Multiagentensystem zu realisieren. Dabei sollen die in der Literatur [GS02, BH06] beschriebenen Ansätze geeignet kombiniert werden, um effizient logikbasierte Argumentation zu ermöglichen. Dazu ist es notwendig, ein formales Konzept für verteilte logikbasierte Argumentation aufzustellen und Kriterien für die Akzeptanz von Gegenargumenten festzulegen. Das System soll explizit die Verwendung von unsicherem Wissen unterstützen. Aus diesem Grund soll das Wissen in sicheres und global verfügbares Wissen sowie in unsicheres und lokales Wissen der einzelnen Agenten, aufgeteilt werden. Die Agenten sollen in der Lage sein, Argumente und Gegenargumente auf Grundlage ihres Wissens zu generieren und es ist zu prüfen, ob der Algorithmus von Besnard und Hunter [BH06] für diese Aufgabe adaptiert werden kann. Ein neutraler Moderator koordiniert den Argumentationsprozess und befragt die einzelnen Agenten nach neuen Argumenten. Weiterhin ist es nötig, verschiedene Präferenzrelationen zum Vergleich von Argumenten zu betrachten und diese geeignet in eine verteilte Umgebung zu integrieren. Ein Beispiel aus dem Rechtswesen soll die Funktionsweise des Systems verdeutlichen und entsprechend formalisiert werden.

Neben der konzeptuellen Arbeit ist die Implementierung eines Programms zur logikbasierten Argumentation in einem Multiagentensystem vorgesehen. Das Programm soll in der Lage sein, das Szenario aus dem Rechtswesen zu realisieren und mit Hilfe der zuvor konzipierten Argumentationsprozesse eine Entscheidung über Akzeptanz oder Widerlegung einer Anfrage zu geben. Das System soll ein Multiagentensystem mit beliebig vielen Agenten simulieren können. Die globale Wissensbasis des Systems und die lokalen Wissensbasen jedes Agenten sollen editierbar sein und geeignet sicheres bzw. unsicheres Wissen darstellen können.

1.3 Übersicht

Der Rest dieser Diplomarbeit ist wie folgt gegliedert: Zur Einführung in das Thema der logikbasierten Argumentation werden in den Kapiteln 2 bzw. 3 die Grundlagen der argumentativen Systeme von García und Simari [GS02] bzw. Besnard und Hunter [BH01] vorgestellt und kurz beschrieben. Die folgenden Kapitel bilden den Hauptteil dieser Diplomarbeit und deren Inhalt wurden in eigener Arbeit zu diesem Thema entwickelt. In Kapitel 4 werden die beiden Ansätze miteinander verglichen und es wird eine geeignete logische Sprache für die Verwendung in einem argumentationsfähigen Multiagentensystem erarbeitet. Zudem wird die Verwendbarkeit des Algorithmus von Besnard und Hunter zur Generierung akzeptierter Argumentationsfolgen geprüft. In Kapitel 5 wird ein argumentationsfähiges Multiagentensystem auf der Grundlage der Ergebnisse aus Kapitel 4 konzipiert und formalisiert. Dazu wird die Struktur des Systems funktional beschrieben

und es werden Algorithmen zur Realisierung des Argumentationsprozesses entwickelt. Zur Veranschaulichung des Argumentationsprozesses in einem verteilten System, wird in Kapitel 6 ein Beispiel aus dem Rechtswesen geeignet modelliert und anschließend an dem System demonstriert. In Kapitel 7 wird eine Implementierung, das DISTRIBUTED ARGUMENTATION SYSTEM (DAS), des in Kapitel 5 konzipierten Systems vorgestellt und die Verwendung mit Hilfe des Rechtsbeispiels aus Kapitel 6 veranschaulicht. Kapitel 8 enthält einen Vergleich des verteilten argumentationsfähigen Systems mit dem Ansatz von García und Simari [GS02]. Anschließend wird in Kapitel 9 ein Ausblick auf mögliche Erweiterungen des Systems zur verteilten logikbasierten Argumentation gegeben und ein Fazit gezogen.

2 Logikbasierte Argumentation nach García und Simari

In diesem Kapitel wird der Ansatz logikbasierter Argumentation von García und Simari [GS02] vorgestellt. Die diesem Ansatz zugrundeliegende logische Sprache DeLP wird die Basis des zu konzipierenden verteilten Systems bilden.

2.1 Einführung

Der Ansatz logikbasierter Argumentation von García und Simari basiert auf der Sprache DeLP (Defeasible Logic Programming) [GS02]. DeLP unterstützt die Möglichkeit, Information in Form von „unsicheren“ oder „anfechtbaren“ Regeln (*defeasible rules*, [Pol95]) zu repräsentieren und verfügt über einen unsicheren Argumentationsmechanismus, um bestimmte Literale als „begründet“ zu verifizieren. Diese unsicheren Regeln sind ein Schlüsselelement zur Realisierung von Argumentation und werden benutzt, um schlüssiges – aber nicht allgemeingültiges – Wissen eines Agenten zu beschreiben.

Mit Hilfe eines Argumentationsprozesses werden unsichere Ableitungen von Literalen miteinander verglichen, um das plausibelste Wissen eines Agenten zu ermitteln. Damit ein Agent ein Literal für plausibel hält, ist es nötig, dass ein nicht-widerlegtes Argument für dieses Literal existiert. Ein Argument ist dabei im weitesten Sinne eine Menge von Regeln, die dieses Literal ableiten. Auf der Grundlage der Anfechtbarkeit können Regeln existieren, die den Regeln des Arguments widersprechen. So können Gegenargumente zu dem Argument konstruiert werden und Gegenargumente zu Gegenargumenten. Eine dialektische Analyse ist notwendig, die Folgen von Argumenten und Gegenargumenten untersucht und entscheidet, welche Argumente widerlegt werden und welche bestehen bleiben.

2.2 Formale Wissensrepräsentation für argumentationsfähige Agenten

2.2.1 Sprachdefinition

DeLP ist eine logische Programmiersprache, die unsicheres Wissen berücksichtigt, um logikbasierte Argumentation zu realisieren. Die folgenden Definitionen der Sprachbestandteile von DeLP sind auf der Grundlage einer gegebenen Grundmenge atomarer

Propositionen gestellt. Auf Grund der üblichen Konventionen aus [Lif96] werden im Folgenden aber auch Beispiele mit mehrstelligen Prädikaten und Regelschemata benutzt.

Definition 2.1 (Fakt)**.** Ein *Fakt* ist ein Literal, also eine atomare Proposition oder die strikte Negation einer atomaren Proposition. Strikte Negation wird in dieser Arbeit durch ein der Proposition vorangestelltes „¬“ dargestellt.

Definition 2.2 (Sichere Regel)**.** Eine *sichere Regel* ist ein geordnetes Paar $h \leftarrow B$, wobei h ein Literal und B eine Menge von Literalen ist. h heißt *Kopfliteral* und jedes $b \in B$ heißt *Rumpfliteral* von $h \leftarrow B$.

Definition 2.3 (Unsichere Regel)**.** Eine *unsichere Regel* ist ein geordnetes Paar $h \prec B$, wobei h ein Literal und B eine Menge von Literalen ist. h heißt *Kopfliteral* und jedes $b \in B$ heißt *Rumpfliteral* von $h \prec B$.

Syntaktisch gesehen ist das Symbol „ $\prec$ “ das einzige, was eine unsichere von einer sicheren Regel unterscheidet. Semantisch beschreibt eine unsichere Regel jedoch Wissensableitungen, die nicht gesichert sind und anderen Ableitungen widersprechen können. Im Allgemeinen werden unsichere Regeln dazu verwendet, *Default*-Wissen abzuleiten, wie beispielsweise *„Vögel können normalerweise fliegen“*.

Sichere und unsichere Regeln sind eigene syntaktische Konstrukte von DeLP und nicht mit der logischen Implikation zu vergleichen. Insbesondere existiert zu einer Regel $a \leftarrow b$ keine Kontraposition $\neg b \leftarrow \neg a$.

Die Wissensbasis eines Agenten besteht aus einer Menge von Fakten, sicheren und unsicheren Regeln und wird DeLP-Programm genannt:

Definition 2.4 (DeLP-Programm)**.** Ein *DeLP-Programm* $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ besteht aus einer Menge Π_G von Fakten, einer Menge Π_R von sicheren Regeln und einer Menge Δ_R von unsicheren Regeln.

Beispiel 2.1 ([GS02], Beispiel 2.1)**.** Sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ gegeben durch

$$
\begin{aligned}
\Pi_G &= \left\{ \begin{array}{l} chicken(tina), \\ scared(tina), \\ penguin(tweety) \end{array} \right\}, \\
\Pi_R &= \left\{ \begin{array}{l} (bird(X) \leftarrow chicken(X)), \\ (bird(X) \leftarrow penguin(X)), \\ (\neg flies(X) \leftarrow penguin(X)) \end{array} \right\}, \\
\Delta_R &= \left\{ \begin{array}{l} (flies(X) \prec bird(X)), \\ (\neg flies(X) \prec chicken(X)), \\ (flies(X) \prec chicken(X), scared(X)), \\ (nests\_in\_trees(X) \prec flies(X)) \end{array} \right\}.
\end{aligned}
$$

Dieses Beispiel enthält als Fakten die Informationen, dass Tina ein Huhn ist und Angst hat, sowie, dass Tweety ein Pinguin ist. Durch sichere Regeln repräsentiert, gilt des Weiteren, dass alle Hühner und alle Pinguine Vögel sind und Pinguine nicht fliegen können. Durch unsichere Regeln repräsentiert gilt, dass Vögel normalerweise fliegen, Hühner normalerweise nicht fliegen (außer sie haben Angst) und etwas, das fliegt, normalerweise in Bäumen nistet.

2.2.2 Unsichere und sichere Ableitungen

Ein DeLP-Programm $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ beschreibt die Wissensbasis eines Agenten und stellt somit nicht das gesamte Wissen des Agenten dar. Mit Hilfe von sicheren und unsicheren Regeln ist es möglich, weitere Literale abzuleiten, die ebenfalls zum Wissen des Agenten dazugehören könnten.

Definition 2.5 (Unsichere Ableitung)**.** Sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ ein DeLP-Programm und h ein Literal. Eine *unsichere Ableitung* von h aus $\mathcal{P}$, dargestellt durch $\mathcal{P} \mathrel{|\!\sim} h$, besteht aus einer endlichen Folge $h_1, \ldots, h_n = h$ von Literalen, so dass für jedes Literal h_i $(1 \leq i \leq n)$ gilt:

1. h_i ist ein Fakt oder
2. es existiert eine sichere oder unsichere Regel in $\mathcal{P}$ mit einem Kopfliteral h_i und Rumpfliteralen $b_1, \ldots, b_k$, wobei jedes Element des Rumpfes b_l $(1 \leq l \leq k)$ gleich einem Element h_j mit $j < i$ ist.

Beispiel 2.2. Aus dem DeLP-Programm von Beispiel 2.1 sind mehrere Ableitungen ersichtlich. Beispielsweise ist die Sequenz

$$chicken(tina), bird(tina), flies(tina)$$

eine unsichere Ableitung für das Literal $flies(tina)$, wobei die Regeln

$$(bird(tina) \leftarrow chicken(tina)) \quad \text{und} \quad (flies(tina) \prec bird(tina))$$

benutzt wurden.

Unsichere Ableitungen weisen ein monotones Ableitungsverhalten auf, d. h. wenn $\mathcal{P}$ ein DeLP-Programm mit $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ ist, aus dem ein Literal h abgeleitet werden kann, und R_1, R_2 bzw. R_3 Mengen von Fakten, sicheren bzw. unsicheren Regeln sind, so hat h auch eine unsichere Ableitung aus $\mathcal{P}' = (\Pi_G \cup R_1, \Pi_R \cup R_2, \Delta_R \cup R_3)$.

Für ein Literal h kann es mehr als nur eine unsichere Ableitung geben. Werden zur Ableitung eines Literals nur sichere Regeln verwendet, so heißt die Ableitung *sicher*.

Definition 2.6 (Sichere Ableitung)**.** Sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ ein DeLP-Programm und h ein Literal mit einer unsicheren Ableitung $h_1, \ldots, h_n = h$. h hat eine *sichere Ableitung* aus $\mathcal{P}$, dargestellt durch $\mathcal{P} \vdash h$, falls entweder h ein Fakt ist oder alle Regeln zur Bestimmung der Folge $h_1, \ldots, h_n$ sichere Regeln sind.

2.3 Argumente, Gegenargumente und gültige Argumentationsfolgen

2.3.1 Unsichere Argumentation

Notation. Wie üblich wird das Symbol $^-$ benutzt, um das Komplement eines Literals zu bezeichnen. Somit gilt $\overline{p} = \neg p$ und $\overline{\neg p} = p$ für eine atomare Proposition p. Zwei Literale heißen *widersprüchlich*, wenn sie komplementär sind.

Definition 2.7 (Widersprüchliche Menge)**.** Sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ ein DeLP-Programm. Eine Menge $\mathcal{C} \subseteq \Pi_G \cup \Pi_R \cup \Delta_R$ heißt *widersprüchlich*, g. d. w. es Ableitungen für zwei komplementäre Literale aus dieser Menge gibt. Falls alle diese Ableitungen sicher sind, so wird dies $\mathcal{C} \vdash \bot$ geschrieben, ansonsten $\mathcal{C} \nvdash \bot$.

Um logikbasierte Argumentation in einem DeLP-Programm $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ zu betreiben, ist es nötig, dass $\Pi_G \cup \Pi_R \cup \Delta_R$ widersprüchlich ist. Gäbe es in $\mathcal{P}$ keinerlei Inkonsistenz, so könnten auch keine Gegenargumente zu vorhandenen Ableitungen aufgestellt werden. Um allerdings „vernünftig“ argumentieren zu können, ist auch eine gewisse Konsistenz bei Π_G und Π_R notwendig. Da Π_G und Π_R sicheres Wissen darstellen wird für jedes DeLP-Programm $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ gefordert, dass $\Pi_G \cup \Pi_R$ nicht widersprüchlich ist. Ist nun aber $\Pi_G \cup \Pi_R \cup \Delta_R$ widersprüchlich, so können komplementäre Literale abgeleitet werden. Um zu entscheiden, welches dieser Literale geglaubt werden soll, ist ein Formalismus notwendig, der zwischen beiden entscheidet. DeLP nutzt dazu unsichere Argumentation, dessen Kernstruktur das *Argument* ist. Informell ausgedrückt ist ein Argument eine minimale und nicht widersprüchliche Menge von unsicheren Regeln, die die Ableitung eines bestimmten Literals ermöglichen.

Definition 2.8 (Argument)**.** Sei h ein Literal und $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ ein DeLP-Programm. $\langle \mathcal{A}, h \rangle$ ist ein *Argument* für h, g. d. w. $\mathcal{A} \subseteq \Delta_R$ ist und

1. eine unsichere Ableitung von h aus $\mathcal{P}' = (\Pi_G, \Pi_R, \mathcal{A})$ existiert,
2. die Menge $\Pi_G \cup \Pi_R \cup \mathcal{A}$ nicht widersprüchlich und
3. $\mathcal{A}$ minimal bezüglich Mengeninklusion ist.

Das Literal h heißt die *Konklusion*, die vom *Support* $\mathcal{A}$ gestützt wird.

Beispiel 2.3. Für das DeLP-Programm aus Beispiel 2.1 lässt sich das folgende Argument für das Literal $\neg flies(tina)$ ableiten:

$$\langle \{\neg flies(tina) \prec chicken(tina)\}, \neg flies(tina) \rangle.$$

Weiterhin besitzt das Literal $flies(tina)$ die beiden Argumente:

$$\langle \{flies(tina) \prec bird(tina)\}, flies(tina) \rangle,$$
$$\langle \{flies(tina) \prec chicken(tina), scared(tina)\}, flies(tina) \rangle \quad .$$

Existiert für ein Literal h eine sichere Ableitung aus $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$, so besitzt h genau ein Argument $\langle \emptyset, h \rangle$, da keine unsicheren Regeln für die Ableitung von h benötigt werden und $\emptyset$ bezüglich Mengeninklusion stets minimal ist. Da $\Pi_G \cup \Pi_R$ nicht widersprüchlich ist, kann es keine sichere Ableitung für $\overline{h}$ geben und daraus folgt auch direkt, dass es kein Argument für $\overline{h}$ geben kann, da für jedes potenzielle Argument $\langle \mathcal{A}, \overline{h} \rangle$ die Menge $\Pi_G \cup \Pi_R \cup \mathcal{A}$ widersprüchlich ist [GS02].

Definition 2.9 (Sub-Argument)**.** Ein Argument $\langle \mathcal{B}, q \rangle$ heißt *Sub-Argument* von einem Argument $\langle \mathcal{A}, h \rangle$, g. d. w. $\mathcal{B} \subseteq \mathcal{A}$.

Beispiel 2.4 ([GS02], Beispiel 3.2)**.** Sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ gegeben durch

$$\begin{aligned} \Pi_G &= \{c, d\}, \\ \Pi_R &= \left\{ \begin{array}{l} (h \leftarrow h_1, h_2), \\ (p \leftarrow h_1), \\ (\neg p \leftarrow h_2), \\ (h_1 \leftarrow b), \\ (h_2 \leftarrow b) \end{array} \right\}, \\ \Delta_R &= \left\{ \begin{array}{l} (b \prec c), \\ (b \prec d), \\ (f \prec b) \end{array} \right\}. \end{aligned}$$

$\mathcal{P}$ enthält das Argument

$$\langle \{(f \prec b), (b \prec c)\}, f \rangle.$$

Dabei ist $\langle (b \prec c), b \rangle$ ein Sub-Argument von $\langle \{(f \prec b), (b \prec c)\}, f \rangle$.

Zudem ist ersichtlich, dass Vereinigungen von Argumenten nicht wieder Argumente sein müssen. $\mathcal{P}$ enthält die Argumente

$$\begin{aligned} \langle \mathcal{A}_1, h_1 \rangle &= \langle \{(b \prec c)\}, h_1 \rangle \text{ und} \\ \langle \mathcal{A}_2, h_2 \rangle &= \langle \{(b \prec d)\}, h_2 \rangle. \end{aligned}$$

Es ist allerdings $\Pi_G \cup \Pi_R \cup \mathcal{A}$ mit $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2 = \{(b \prec c), (b \prec d)\}$ widersprüchlich, da sowohl p als auch $\neg p$ abgeleitet werden können. Somit ist $\langle \mathcal{A}, l \rangle$ für kein l ein Argument.

Notation. Argumente können durch Dreiecke grafisch repräsentiert werden. Der obere Eckpunkt eines Dreiecks stellt dabei die Konklusion des Arguments dar und die Menge der unsicheren Regeln des Arguments wird mit der Dreiecksfläche identifiziert. Sub-Argumente können dann als kleinere Dreiecke in dem Dreieck des entsprechenden Arguments dargestellt werden. Abbildung 2.1 zeigt die grafische Repräsentation eines Arguments $\langle \mathcal{A}, h \rangle$ mit einem Sub-Argument $\langle \mathcal{B}, q \rangle$.

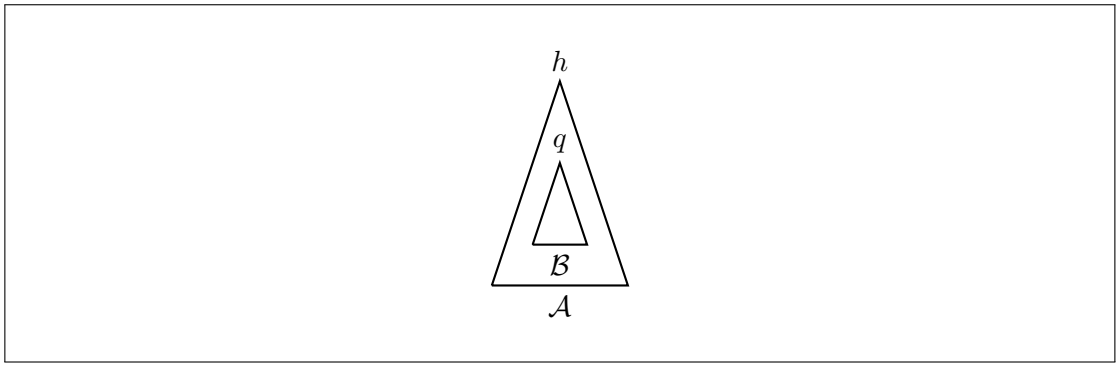

Abbildung 2.1: Ein Argument $\langle \mathcal{A}, h \rangle$ und ein Sub-Argument $\langle \mathcal{B}, q \rangle$

2.3.2 Gegenargumente

In DeLP werden Antworten zu Anfragen durch Argumente gestützt. Wie das Beispiel 2.4 zeigt, kann es allerdings Argumente für ein Literal und sein Komplement geben.

Definition 2.10 (Unstimmigkeit)**.** Sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ ein DeLP-Programm. Zwei Literale h und h_1 sind *unstimmig*, g. d. w. die Menge $\Pi_G \cup \Pi_R \cup \{h, h_1\}$ widersprüchlich ist.

Zwei komplementäre Literale p und $\neg p$ sind somit stets unstimmig, da für jedes DeLP-Programm $\mathcal{P}$=(Π_G,Π_R,Δ_R) die Menge $\Pi_G \cup \Pi_R \cup \{p, \neg p\}$ widersprüchlich ist. Allerdings können auch zwei nicht-komplementäre Literale unstimmig sein. Beispielsweise sind für $\Pi_G = \emptyset$ und $\Pi_R = \{(\neg h \leftarrow b), (h \leftarrow a)\}$ die Literale a und b unstimmig, da $\Pi_G \cup \Pi_R \cup \{a, b\}$ widersprüchlich ist.

Definition 2.11 (Gegenargument)**.** Ein Argument $\langle \mathcal{A}_1, h_1 \rangle$ ist ein *Gegenargument* zu einem Argument $\langle \mathcal{A}_2, h_2 \rangle$ an einem Punkt h, g. d. w. es ein Sub-Argument $\langle \mathcal{A}, h \rangle$ von $\langle \mathcal{A}_2, h_2 \rangle$ gibt, so dass h und h_1 unstimmig sind.

Falls $\langle \mathcal{A}_1, h_1 \rangle$ ein Gegenargument zu $\langle \mathcal{A}_2, h_2 \rangle$ an einem Punkt h ist, so heißt h auch *Angriffspunkt* und das Sub-Argument $\langle \mathcal{A}, h \rangle$ heißt das *unstimmige Sub-Argument*. Falls $h = h_2$, so heißt $\langle \mathcal{A}_1, h_1 \rangle$ *direktes Gegenargument* zu $\langle \mathcal{A}_2, h_2 \rangle$ und ansonsten *indirektes Gegenargument*. In Abbildung 2.2 ist diese Unterscheidung grafisch repräsentiert.

Beispiel 2.5. In dem DeLP-Programm aus Beispiel 2.1 ist

$$\langle \{\neg flies(tina) \prec chicken(tina)\}, \neg flies(tina) \rangle$$

ein direktes Gegenargument zu $\langle \{flies(tina) \prec bird(tina)\}, flies(tina) \rangle$. Sei ein weiteres Argument $\langle \mathcal{A}_4, nests\_in\_trees(tina) \rangle$ gegeben durch

$$\mathcal{A}_4 = \{(nests\_in\_trees(tina) \prec flies(tina)), (flies(tina) \prec bird(tina))\}.$$

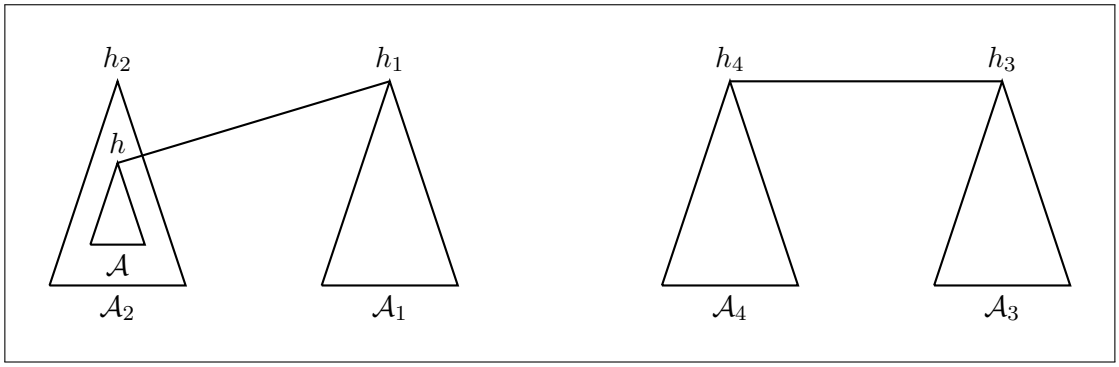

Abbildung 2.2: Indirektes Gegenargument (links) und direktes Gegenargument (rechts)

Dann ist das Argument $\langle\{\neg flies(tina) \prec chicken(tina)\}, \neg flies(tina)\rangle$ ein indirektes Gegenargument zu $\langle\mathcal{A}_4, nests\_in\_trees(tina)\rangle$ mit dem unstimmigen Sub-Argument $\langle\{(flies(tina) \prec bird(tina))\}, flies(tina)\rangle$.

2.3.3 Vergleich von Argumenten

Ein zentraler Aspekt der logikbasierten Argumentation ist die Definition eines formalen Kriteriums zum Vergleich von Argumenten. Zwei Argumente sollten auf eine semantisch schlüssige Art und Weise syntaktisch miteinander vergleichbar sein, damit der Argumentationsprozess intuitiv korrekte Ergebnisse liefert. Im Allgemeinen ist ein Präferenzkriterium für Argumente eine binäre Relation auf der Menge der Argumente. Sie sollte wenigstens antisymmetrisch und transitiv und im besten Fall total sein. Jedoch ist das Präferenzkriterium für Argumente eine modulare Komponente und kann durch jede beliebige Relation ausgetauscht werden, ohne auf den Mechanismus des Argumentationsprozesses einzuwirken.

Notation. Eine Präferenzrelation zwischen Argumenten wird durch das Symbol $\succ$ dargestellt. Gegeben zwei Argumente $\langle\mathcal{A}_1, h_1\rangle$ und $\langle\mathcal{A}_2, h_2\rangle$, so heißt $\langle\mathcal{A}_1, h_1\rangle$ *strikt besser* als $\langle\mathcal{A}_2, h_2\rangle$, g. d. w. $\langle\mathcal{A}_1, h_1\rangle \succ \langle\mathcal{A}_2, h_2\rangle$.

Es folgen zwei Beispiele für Präferenzrelationen unter Argumenten.

Generalized Specificity

Generalized Specificity wurde zuerst in [Lou87] und [Poo85] vorgestellt, in [SL92] und [SCnG94] erweitert und später in [SGCnS03] angepasst, um in das DeLP-Framework zu passen. Hier wird die Definition aus [SGCnS03] auf der Grundlage von DeLP-Programmen benutzt. Informell bevorzugt dieses Kriterium Argumente, die mehr Informationen oder weniger unsichere Schlussfolgerungen benutzen.

Definition 2.12 (Generalized Specificity)**.** Sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ ein DeLP-Programm und $\mathcal{F}$ die Menge aller Literale, die eine sichere oder unsichere Ableitung aus $\mathcal{P}$ haben. Ein Argument $\langle \mathcal{A}_1, h_1 \rangle$ heißt *mindestens so spezifisch* wie ein Argument $\langle \mathcal{A}_2, h_2 \rangle$ (geschrieben $\langle \mathcal{A}_1, h_1 \rangle \succeq_{spec} \langle \mathcal{A}_2, h_2 \rangle$), g. d. w. für alle $H \subseteq \mathcal{F}$ gilt

$$\Pi_R \cup H \cup \mathcal{A}_1 \mathrel{|\!\sim} h_1 \text{ und } \Pi_R \cup H \nvdash h_1 \text{ implizieren } \Pi_R \cup H \cup \mathcal{A}_2 \mathrel{|\!\sim} h_2.$$

Wie in [Poo85] definiere weiterhin: $\langle \mathcal{A}_1, h_1 \rangle$ heißt *strikt spezifischer* als $\langle \mathcal{A}_2, h_2 \rangle$ (geschrieben $\langle \mathcal{A}_1, h_1 \rangle \succ_{spec} \langle \mathcal{A}_2, h_2 \rangle$), g. d. w. $\langle \mathcal{A}_1, h_1 \rangle \succeq_{spec} \langle \mathcal{A}_2, h_2 \rangle$ und $\langle \mathcal{A}_2, h_2 \rangle \not\succeq_{spec} \langle \mathcal{A}_1, h_1 \rangle$.

Die Mengen H mit $\Pi_R \cup H \cup \mathcal{A}_1 \mathrel{|\!\sim} h_1$ und $\Pi_R \cup H \nvdash h_1$ heißen auch *nicht-triviale Aktivierungsmengen* von $\mathcal{A}_1$ [SGCnS03]. $\langle \mathcal{A}_1, h_1 \rangle$ ist also mindestens so spezifisch wie $\langle \mathcal{A}_2, h_2 \rangle$, g. d. w. jede nicht-triviale Aktivierungsmenge von $\mathcal{A}_1$ auch Aktivierungsmenge von $\mathcal{A}_2$ ist. Das heißt, dass $\langle \mathcal{A}_2, h_2 \rangle$ höchstens soviel Information benutzt wie $\langle \mathcal{A}_1, h_1 \rangle$ und in keinem Fall mehr.

Beispiel 2.6. Im DeLP-Programm aus Beispiel 2.1 ist das Argument

$$\langle \mathcal{A}_1, h_1 \rangle \quad = \quad \langle \{\neg flies(tina) \prec chicken(tina)\}, \neg flies(tina) \rangle$$

strikt spezifischer als das Argument

$$\langle \mathcal{A}_2, h_2 \rangle \quad = \quad \langle \{flies(tina) \prec bird(tina)\}, flies(tina) \rangle,$$

da $\langle \mathcal{A}_1, h_1 \rangle$ „direkter“ als $\langle \mathcal{A}_1, h_1 \rangle$ ist ($\langle \mathcal{A}_2, h_2 \rangle$ benutzt zur Ableitung von $bird(tina)$ noch die sichere Regel $bird(tina) \prec chicken(tina)$). Jede nicht-triviale Aktivierungsmenge H von $\langle \mathcal{A}_1, h_1 \rangle$ ist auch nicht-triviale Aktivierungsmenge von $\langle \mathcal{A}_2, h_2 \rangle$, da H auf jeden Fall das Literal $chicken(tina)$ enthalten muss. Jedoch ist die Menge $H' = \{bird(tina)\}$ nur nicht-triviale Aktivierungsmenge für $\langle \mathcal{A}_2, h_2 \rangle$ und nicht für $\langle \mathcal{A}_1, h_1 \rangle$. In demselben Programm ist das Argument

$$\langle \mathcal{A}_3, h_3 \rangle \quad = \quad \langle \{flies(tina) \prec chicken(tina), scared(tina)\}, flies(tina) \rangle$$

strikt spezifischer als $\langle \mathcal{A}_1, h_1 \rangle$, da $\langle \mathcal{A}_3, h_3 \rangle$ mehr Information benutzt. Jede nicht-triviale Aktivierungsmenge H von $\langle \mathcal{A}_3, h_3 \rangle$ muss die Literale $chicken(tina)$ und $scared(tina)$ enthalten und ist wegen $chicken(tina) \in H$ somit auch nicht-triviale Aktivierungsmenge für $\langle \mathcal{A}_1, h_1 \rangle$. Jedoch ist die Menge $H' = \{chicken(tina)\}$ nur nicht-triviale Aktivierungsmenge für $\langle \mathcal{A}_1, h_1 \rangle$ und nicht für $\langle \mathcal{A}_3, h_3 \rangle$.

Für weitere Eigenschaften der *Generalized Specificity* siehe [SGCnS03].

Possibilistische Präferenzrelation aus P-DeLP

Die in [CnSAG04] vorgestellte Präferenzrelation benötigt zunächst eine Erweiterung der Elemente von DeLP um quantitative Abschätzungen der „Sicherheit“. Jede unsichere

Regel $h \prec B$ wird zu einem Paar $(h \prec B, \alpha)$ erweitert, wobei $\alpha \in [0,1]$ eine quantitative Abschätzung der „Sicherheit“ dieser Regel ist. Um in der Notation konform zu bleiben, erhalten Fakten und strikte Regeln einen Sicherheitsfaktor von 1, der sicheres Wissen repräsentiert. Die erzeugte Sprache heißt dann P-DeLP (*Possibilistic Defeasible Logic Programming*).

Die Semantik von P-DeLP [CnSAG04] ist auf der Grundlage der Sprache PGL [AG00, AG01] definiert, die wiederum auf Fuzzy Logik basiert. In PGL erhalten Formeln, die aus Fuzzy-Propositionen konstruiert werden, eine Zusatzinformation in Form eines Notwendigkeitsmaßes (*necessity measure*). Fuzzy-Propositionen liefern eine geeignete Struktur für Wissensrepräsentation in Situationen, in denen ein Agent nur über unsicheres und unpräzises Wissen verfügt.

Beispiel 2.7 ([CnSAG04])**.** Die Fuzzy-Aussage „Die Motorgeschwindigkeit ist niedrig“ kann als Fuzzy-Proposition $engine\_speed(low)$ modelliert werden, wobei low eine Fuzzy-Menge $\mu_{low} : [0, 6000] \rightarrow [0, 1]$ ist. Dabei bedeute $\mu_{low}(x) = \alpha$, dass x Umdrehungen des Motors pro Minute mit einem Sicherheitsfaktor von mindestens α als eine niedrige Geschwindigkeit aufgefasst werden kann. Die Proposition $engine\_speed(low)$ kann dann interpretiert werden durch:

> Für jedes $\alpha \in [0, 1]$ existiert ein $x \in [\mu_{low}]_\alpha$, so dass die Motorgeschwindigkeit mit einem Sicherheitsfaktor von $1 - \alpha$ gleich x ist.

Dabei sei $[\mu_{low}]_\alpha$ der α-Schnitt von μ_{low}.

Da Formeln in P-DeLP auf Fuzzy-Propositionen basieren, ist die unterliegende Semantik mehrwertig. Für eine genaue Beschreibung der Semantik siehe [CnSAG04]. Um Argumente – also Mengen von unsicheren Regeln – miteinander zu vergleichen, genügt es, als Sicherheitsfaktor für jedes Argument das Minimum der Sicherheitsfaktoren seiner Regeln anzunehmen und diese beiden Faktoren miteinander zu vergleichen (dies entspricht der üblichen Fuzzy-Und-Operation). Somit wird das Argument als besser angesehen, dessen unsicherste Regel sicherer ist als die unsicherste Regeln des anderen Arguments.

Die formale Definition der possibilistischen Präferenzrelation nach [CnSAG04] ist:

Definition 2.13 (Possibilistische Präferenzrelation nach [CnSAG04])**.** Sei $\mathcal{P}$ ein P-DeLP-Programm und seien $\langle \mathcal{A}_1, h_1 \rangle$ und $\langle \mathcal{A}_2, h_2 \rangle$ Argumente. Es gilt $\langle \mathcal{A}_1, h_1 \rangle \succeq_{poss} \langle \mathcal{A}_2, h_2 \rangle$, g. d. w.

$$\min \{\alpha | (h \prec B, \alpha) \in \mathcal{A}_1\} \geq \min \{\alpha | (h \prec B, \alpha) \in \mathcal{A}_2\}.$$

Die strikte Relation $\succ_{poss}$ sei wieder wie üblich definiert.

Beispiel 2.8. Sei ein P-DeLP-Programm $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ gegeben durch

$$\begin{aligned} \Pi_G &= \{(a,1),(b,1)\}, \\ \Pi_R &= \left\{\ (c \leftarrow a, 1)\ \right\}, \\ \Delta_R &= \left\{ \begin{array}{l} ((d \prec c),(0,8)), \\ ((e \prec d),(0,7)), \\ ((\neg d \prec a, b),(0,9)) \end{array} \right\}. \end{aligned}$$

$\mathcal{P}$ enthält das Argument

$$\langle \mathcal{A}_1, e\rangle = \langle\{((e \prec d),(0,7)),((d \prec c),(0,8))\}, e\rangle$$

und es ist

$$\min\{\alpha | (H \prec B, \alpha) \in \mathcal{A}_1\} = \min\{(0,7),(0,8)\} = 0,7.$$

Zu $\langle \mathcal{A}_1, e\rangle$ existiert ein Gegenargument

$$\langle \mathcal{A}_2, \neg d\rangle = \langle\{((\neg d \prec a, b),(0,9))\}, \neg d\rangle$$

mit

$$\min\{\alpha | (H \prec B, \alpha) \in \mathcal{A}_2\} = \min\{0.9\} = 0.9.$$

Wegen $0,9 > 0,7$ gilt somit $\langle \mathcal{A}_2, h_2\rangle \succ_{poss} \langle \mathcal{A}_1, h_1\rangle$.

Die Theorie hinter P-DeLP ist etwas umfangreicher als sie hier in diesem Zusammenhang dargestellt werden kann. Da hier die possibilistische Präferenzrelation nur als ein Beispiel für eine Relation unter Argumenten aufgeführt werden soll, sei für weitere Informationen zu P-DeLP auf [CnSAG04] verwiesen.

2.3.4 Gültige Argumentationsfolgen

Sei im Folgenden $\succ$ eine Präferenzrelation unter Argumenten. Da die Totalität von $\succ$ im Allgemeinen nicht gefordert ist, ergeben sich für ein Argument $\langle \mathcal{A}_1, h_1\rangle$ und ein Gegenargument $\langle \mathcal{A}_2, h_2\rangle$ drei verschiedene Anordnungsmöglichkeiten:

- $\langle \mathcal{A}_2, h_2\rangle$ ist „besser“ als $\langle \mathcal{A}_1, h_1\rangle$ bezüglich $\succ$: In diesem Fall heißt $\langle \mathcal{A}_2, h_2\rangle$ ein *ordentlicher Angriff* auf $\langle \mathcal{A}_1, h_1\rangle$.
- $\langle \mathcal{A}_2, h_2\rangle$ und $\langle \mathcal{A}_1, h_1\rangle$ sind nicht vergleichbar bezüglich $\succ$: In diesem Fall heißt $\langle \mathcal{A}_2, h_2\rangle$ ein *blockierender Angriff* auf $\langle \mathcal{A}_1, h_1\rangle$.
- $\langle \mathcal{A}_2, h_2\rangle$ ist „schlechter“ als $\langle \mathcal{A}_1, h_1\rangle$ bezüglich $\succ$: In diesem Fall wird $\langle \mathcal{A}_2, h_2\rangle$ nicht als Angriff auf $\langle \mathcal{A}_1, h_1\rangle$ akzeptiert.

Aufgrund der Antisymmetrie von Präferenzrelationen kann „Gleichheit“ von Argumenten nicht auftreten[1].

Da zum Vergleich von Argumenten zwischen direkten und indirekten Gegenargumenten unterschieden werden muss, bezieht sich die formale Definition eines Angriffs stets auf das unstimmige Sub-Argument.

Definition 2.14 (Ordentlicher Angriff)**.** Seien $\langle \mathcal{A}_1, h_1 \rangle$ und $\langle \mathcal{A}_2, h_2 \rangle$ zwei Argumente. $\langle \mathcal{A}_1, h_1 \rangle$ heißt *ordentlicher Angriff* auf $\langle \mathcal{A}_2, h_2 \rangle$ am Literal h, g. d. w. es ein Sub-Argument $\langle \mathcal{A}, h \rangle$ von $\langle \mathcal{A}_2, h_2 \rangle$ gibt, so dass $\langle \mathcal{A}_1, h_1 \rangle$ ein Gegenargument zu $\langle \mathcal{A}_2, h_2 \rangle$ am Punkt h ist und $\langle \mathcal{A}_1, h_1 \rangle \succ \langle \mathcal{A}, h \rangle$ gilt.

Definition 2.15 (Blockierender Angriff)**.** Seien $\langle \mathcal{A}_1, h_1 \rangle$ und $\langle \mathcal{A}_2, h_2 \rangle$ zwei Argumente. $\langle \mathcal{A}_1, h_1 \rangle$ heißt *blockierender Angriff* auf $\langle \mathcal{A}_2, h_2 \rangle$ am Literal h, g. d. w. es ein Sub-Argument $\langle \mathcal{A}, h \rangle$ von $\langle \mathcal{A}_2, h_2 \rangle$ gibt, so dass $\langle \mathcal{A}_1, h_1 \rangle$ ein Gegenargument zu $\langle \mathcal{A}_2, h_2 \rangle$ am Punkt h ist und $\langle \mathcal{A}_1, h_1 \rangle$ nicht vergleichbar mit $\langle \mathcal{A}, h \rangle$ bezüglich des Präferenzkriterium ist, also $\langle \mathcal{A}_1, h_1 \rangle \not\succ \langle \mathcal{A}, h \rangle$ und $\langle \mathcal{A}, h \rangle \not\succ \langle \mathcal{A}_1, h_1 \rangle$ gilt.

Definition 2.16 (Angriff)**.** Ein Argument $\langle \mathcal{A}_1, h_1 \rangle$ ist ein *Angriff* auf ein Argument $\langle \mathcal{A}_2, h_2 \rangle$, g. d. w. entweder

1. $\langle \mathcal{A}_1, h_1 \rangle$ ein ordentlicher Angriff auf $\langle \mathcal{A}_2, h_2 \rangle$ ist oder
2. $\langle \mathcal{A}_1, h_1 \rangle$ ein blockierender Angriff auf $\langle \mathcal{A}_2, h_2 \rangle$ ist.

Die Definition eines Angriffs ist noch nicht ausreichend, um eine intuitiv korrekte Folge von Argumenten und Gegenargumenten zu beschreiben. Ein Angriff beschreibt nur, wie ein Argument mit seinem Gegenargument in Relation stehen soll, vorangegangene Argumente werden dabei nicht betrachtet. Dies ist jedoch nicht unbedingt sinnvoll, da ein Gegenargument die gesamte vorangegangene Folge von Argumenten und Gegenargumenten berücksichtigen sollte, um ein akzeptierter Angriff zu sein. Beispielsweise sollte kein Argument in einer Argumentationsfolge wiederholt werden, da es sonst zu einer zyklischen Weiterentwicklung der Argumentationsfolge kommen kann. Das ist in den wenigsten Fällen erwünscht, da endliche Wissensbasen nicht zu unendlichen Argumentationsfolgen führen sollten. Um diesen Fall zu unterbinden, ist es sogar notwendig, dass kein Argument in einer Argumentationsfolge ein Sub-Argument eines vorangegangen Arguments sein darf, vgl. [GS02].

Ein weiterer wichtiger Punkt für die Definition von *akzeptierten Argumentationsfolgen* ist die der *Konkordanz* [GS02]. Sei

$$\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_i, h_i \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$$

[1] Es existiert allerdings für die *Generalized Specificity* der Begriff der *Equi-Specificity*. Äquivalent spezifische Argumente können sich aber nicht widersprechen, so dass dieser Fall in einem Argumentationsprozess nicht auftreten kann (siehe [GS02], Proposition 3.4).

eine Folge von Argumenten, so dass jedes Argument $\langle \mathcal{A}_i, h_i \rangle$ ein Angriff auf seinen Vorgänger $\langle \mathcal{A}_{i-1}, h_{i-1} \rangle$ ist $(1 < i \leq n)$. Dann lassen sich in Λ zwei Teilfolgen erkennen: Zum einen die Folge von Argumenten mit ungeradem Index, die das Wurzelargument „unterstützen“, und zum anderen die Folge von Argumenten mit geradem Index, die dem Wurzelargument „widersprechen“. Da sich die Argumente einer dieser Teilfolgen somit „auf der selben Seite befinden“, ist es sinnvoll zu fordern, dass diese Argumente untereinander nicht widersprüchlich sind, vgl. [GS02].

Eine nähere Beachtung muss auch die Definition eines blockierenden Angriffs erfahren. Ein blockierender Angriff $\langle \mathcal{A}_1, h_1 \rangle$ auf ein Argument $\langle \mathcal{A}_2, h_2 \rangle$ ruft eine gewisse Patt-Situation hervor, da $\langle \mathcal{A}_1, h_1 \rangle$ weder besser noch schlechter als $\langle \mathcal{A}_2, h_2 \rangle$ ist. Damit sich solche Patt-Situationen nicht beliebig tief in einer Argumentationsfolge fortführen, wird gefordert, dass auf jeden blockierenden Angriff ein ordentlicher Angriff folgen muss. So wird vermieden, dass Situationen entstehen können, bei denen die Anzahl von Argumenten für ein Literal entscheidender sind als ihre „Qualität“, vgl. dazu [GS02].

Definition 2.17 (Akzeptierte Argumentationsfolge)**.** Sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ ein DeLP-Programm und $\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_i, h_i \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$ eine Folge von Argumenten. Λ heißt *akzeptierte Argumenationsfolge*, g. d. w. die folgenden Bedingungen erfüllt sind:

1. Λ ist eine endliche Folge.
2. Jedes Argument $\langle \mathcal{A}_i, h_i \rangle$ für $i > 0$ ist ein Angriff auf seinen Vorgänger $\langle \mathcal{A}_{i-1}, h_{i-1} \rangle$. Falls $\langle \mathcal{A}_i, h_i \rangle$ ein blockierender Angriff auf $\langle \mathcal{A}_{i-1}, h_{i-1} \rangle$ ist und $\langle \mathcal{A}_{i+1}, h_{i+1} \rangle$ existiert, so ist $\langle \mathcal{A}_{i+1}, h_{i+1} \rangle$ ein ordentlicher Angriff auf $\langle \mathcal{A}_i, h_i \rangle$.
3. Für die Menge der h_1 unterstützenden Argumente $\Lambda_S = \{\langle \mathcal{A}_i, h_i \rangle | i \text{ ungerade}\}$ gilt:
$$\Pi_G \cup \Pi_R \cup \bigcup_{\langle \mathcal{A}, h \rangle \in \Lambda_S} \mathcal{A} \not\vdash \bot$$
4. Für die Menge der h_1 widersprechenden Argumente $\Lambda_I = \{\langle \mathcal{A}_i, h_i \rangle | i \text{ gerade}\}$ gilt:
$$\Pi_G \cup \Pi_R \cup \bigcup_{\langle \mathcal{A}, h \rangle \in \Lambda_I} \mathcal{A} \not\vdash \bot$$
5. Kein Argument $\langle \mathcal{A}_k, h_k \rangle$ ist ein Sub-Argument eines Arguments $\langle \mathcal{A}_i, h_i \rangle$ mit $i < k$.

2.3.5 Begründung von Literalen

In DeLP wird ein Literal h als *begründet* angesehen, wenn es ein Argument $\langle \mathcal{A}, h \rangle$ gibt, welches letztlich nicht widerlegt wird. Um zu entscheiden, ob ein Argument $\langle \mathcal{A}, h \rangle$ nicht widerlegt wird, ist die Menge aller Angriffe auf $\langle \mathcal{A}, h \rangle$ zu betrachten. Da jeder Angriff auch wieder ein Argument ist zu dem Gegenargumente existieren können, sind somit alle akzeptierten Argumentationsfolgen, die mit $\langle \mathcal{A}, h \rangle$ beginnen, zu betrachten. Das führt zu einer Baumstruktur, die den gesamten Argumentationsprozess formal wiedergibt.

Definition 2.18 (Dialektischer Baum)**.** Sei $\langle \mathcal{A}_0, h_0 \rangle$ ein Argument eines DeLP-Programms $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$. Ein *dialektischer Baum* für $\langle \mathcal{A}_0, h_0 \rangle$, geschrieben $\mathcal{T}_{\langle \mathcal{A}_0, h_0 \rangle}$, ist definiert durch die folgenden beiden Bedingungen:

1. Die Wurzel von $\mathcal{T}$ ist $\langle \mathcal{A}_0, h_0 \rangle$.
2. Sei $\langle \mathcal{A}_n, h_n \rangle$ ein Knoten von $\mathcal{T}$ mit $i \geq 0$ und $\Lambda = [\langle \mathcal{A}_0, h_0 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$ die Folge der Knoten des Weges von der Wurzel zu $\langle \mathcal{A}_n, h_n \rangle$. Sei $\{\langle \mathcal{B}_1, q_1 \rangle, \ldots, \langle \mathcal{B}_k, q_k \rangle\}$ die Menge aller Angriffe auf $\langle \mathcal{A}_n, h_n \rangle$. Für jedes Gegenargument $\langle \mathcal{B}_i, q_i \rangle$ mit $1 \leq i \leq k$, für das die Argumentationsfolge $\Lambda' = [\langle \mathcal{A}_0, h_0 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle, \langle \mathcal{B}_i, q_i \rangle]$ akzeptiert wird, erhält der Knoten $\langle \mathcal{A}_n, h_n \rangle$ das Kind $\langle \mathcal{B}_i, q_i \rangle$. Falls es kein Gegenargument zu $\langle \mathcal{A}_n, h_n \rangle$ oder kein $\langle \mathcal{B}_i, q_i \rangle$ gibt, so dass Λ' akzeptiert wird, so ist $\langle \mathcal{A}_n, h_n \rangle$ ein Blatt.

Um zu entscheiden, ob das Wurzelargument widerlegt wird, ist eine *bottom-up*-Analyse des Baumes nötig. Ein Argument wird widerlegt, wenn mindestens ein Angriff auf dieses Argument nicht widerlegt wurde. Da die Blätter des Baumes nicht angegriffen werden, werden sie nicht widerlegt. Formal wird die Analyse durch eine Markierung des dialektischen Baumes beschrieben.

Definition 2.19 (Markierung eines dialektischen Baumes)**.** Sei $\mathcal{T}_{\langle \mathcal{A}, h \rangle}$ ein dialektischer Baum für $\langle \mathcal{A}, h \rangle$. Der zu $\mathcal{T}$ gehörende *markierte dialektische Baum* $\mathcal{T}^*_{\langle \mathcal{A}, h \rangle}$ ist definiert durch die folgenden beiden Bedingungen:

1. Jedes Blatt von $\mathcal{T}_{\langle \mathcal{A}, h \rangle}$ wird in $\mathcal{T}^*_{\langle \mathcal{A}, h \rangle}$ mit „U“ (*undefeated*) markiert.
2. Sei $\langle \mathcal{B}, q \rangle$ ein innerer Knoten von $\mathcal{T}_{\langle \mathcal{A}, h \rangle}$. Der Knoten $\langle \mathcal{B}, q \rangle$ wird in $\mathcal{T}^*_{\langle \mathcal{A}, h \rangle}$ mit „U“ markiert, g. d. w. jedes Kind von $\langle \mathcal{B}, q \rangle$ in $\mathcal{T}^*_{\langle \mathcal{A}, h \rangle}$ mit „D“ (*defeated*) markiert ist. Der Knoten $\langle \mathcal{B}, q \rangle$ wird in $\mathcal{T}^*_{\langle \mathcal{A}, h \rangle}$ mit „D“ markiert, g.d.w. mindestens ein Kind von $\langle \mathcal{B}, q \rangle$ in $\mathcal{T}^*_{\langle \mathcal{A}, h \rangle}$ mit „U“ markiert ist.

Definition 2.20 (Begründung)**.** Ein Literal h heißt *begründet*, g. d. w. es ein Argument $\langle \mathcal{A}, h \rangle$ für h gibt, so dass in dem zu $\langle \mathcal{A}, h \rangle$ gehörenden markierten dialektischen Baum $\mathcal{T}^*_{\langle \mathcal{A}, h \rangle}$ die Wurzel mit „U“ markiert ist. In diesem Fall heißt $\langle \mathcal{A}, h \rangle$ eine *Begründung* für h.

Beispiel 2.9 ([GS02], Beispiel 5.1)**.** Sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ gegeben durch

$$
\begin{aligned}
\Pi_G &= \{c, d, e, g, i, j, k\}, \\
\Pi_R &= \emptyset, \\
\Delta_R &= \left\{ \begin{array}{cccc} (a \prec b), &)b \prec c), & (\neg b \prec c, d), & (\neg b \prec e), \\ (\neg f \prec g, h), & (h \prec j), & (\neg b \prec c, f), & (f \prec g), \\ (\neg f \prec i), & (\neg h \prec k) & & \end{array} \right\}.
\end{aligned}
$$

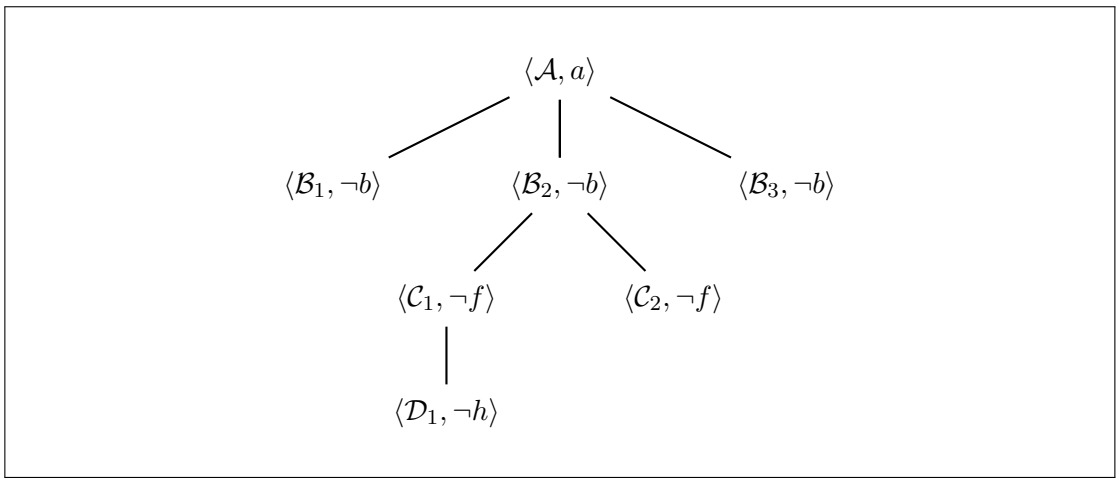

Abbildung 2.3: Dialektischer Baum zu Beispiel 2.9

Als Präferenzrelation für Argumente sei *Generalized Specificity* nach Definition 2.12 angenommen. Dann wird in $\mathcal{P}$ das Literal a durch das Argument

$$\langle \mathcal{A}, a \rangle = \langle \{(a \prec b), (b \prec c)\}, a \rangle$$

unterstützt und es gibt zu $\langle \mathcal{A}, a \rangle$ genau drei Angriffe, die zu jeweils unterschiedlichen Argumentationsfolgen führen:

$$\begin{aligned} \langle \mathcal{B}_1, \neg b \rangle &= \langle \{(\neg b \prec c, d)\}, \neg b \rangle, \\ \langle \mathcal{B}_2, \neg b \rangle &= \langle \{(\neg b \prec c, f), (f \prec g)\}, \neg b \rangle, \\ \langle \mathcal{B}_3, \neg b \rangle &= \langle \{(\neg b \prec e)\}, \neg b \rangle. \end{aligned}$$

Die ersten beiden Angriffe sind ordentliche Angriffe im Sinne der *Generalized Specificity*, der letzte ist ein blockierender Angriff. Zu dem Argument $\langle \mathcal{B}_1, \neg b \rangle$ existiert ein Gegenargument $\langle (b \prec c), b \rangle$, das allerdings nicht als Angriff akzeptiert wird, da $\langle (b \prec c), b \rangle$ ein Sub-Argument von $\langle \mathcal{A}, a \rangle$ ist und die Argumentationsfolge deshalb ungültig würde. Werden für diese Angriffe die darauf folgenden Angriffe ermittelt, entsteht der in Abbildung 2.3 dargestellte dialektische Baum für das Argument $\langle \mathcal{A}, a \rangle$. Dabei sind die Mengen $\mathcal{C}_1, \mathcal{C}_2, \mathcal{D}_1$ definiert durch

$$\begin{aligned} \mathcal{C}_1 &= \{(\neg f \prec g, h), (h \prec j)\}, \\ \mathcal{C}_2 &= \{(\neg f \prec i)\}, \\ \mathcal{D}_1 &= \{(\neg h \prec k)\}. \end{aligned}$$

Eine Markierung des Baumes ergibt, dass das Argument $\langle \mathcal{A}, a \rangle$ beispielsweise von dem nicht-widerlegten Gegenargument $\langle \mathcal{B}_1, \neg b \rangle$ widerlegt wird. Der markierte dialektische Baum für das Argument $\langle \mathcal{A}, a \rangle$ ist in Abbildung 2.4 dargestellt.

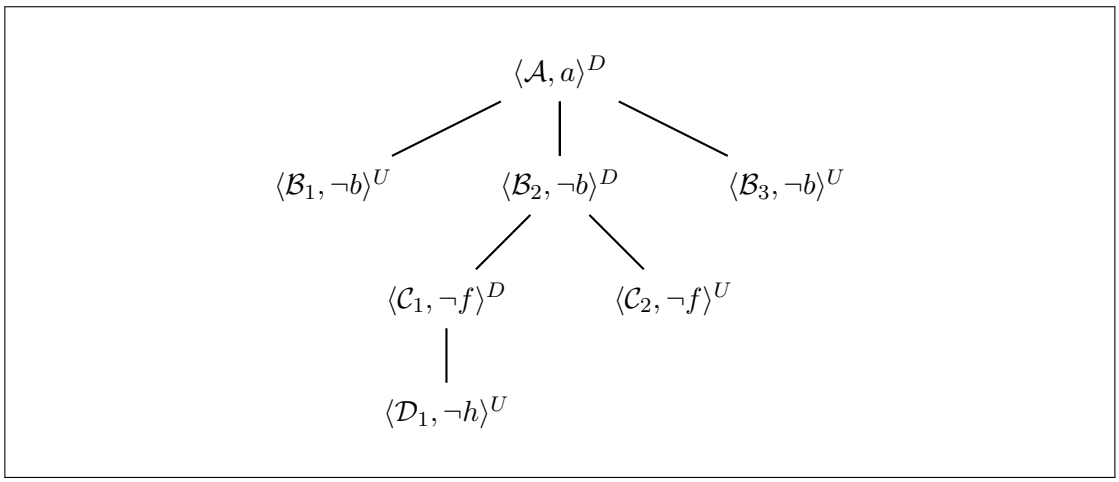

Abbildung 2.4: Markierter dialektischer Baum zu Beispiel 2.9

Basierend auf Begründungen, kann das Antwortverhalten eines DeLP-Interpreters wie folgt definiert werden:

Definition 2.21 (Antworten zu Anfragen)**.** Die Antwort eines DeLP-Interpreters auf eine Anfrage h ist definiert durch:

- YES, g. d. w. h begründet ist.
- NO, g. d. w. das Komplement zu h begründet ist.
- UNDECIDED, g. d. w. weder h noch $\overline{h}$ begründet sind.
- UNKNOWN, g. d. w. h nicht in der betrachteten Sprache liegt.

2.4 Zusammenfassung

In diesem Kapitel wurde der Ansatz der logikbasierten Argumentation von García und Simari am Beispiel von DeLP [GS02] beschrieben. DeLP benutzt die Syntax logischer Programme, um mit Hilfe von Fakten, sicheren und unsicheren Regeln, Argumentation durch Widerlegung von Regeln durch andere Regeln zu realisieren. Mengen von unsicheren Regeln können als Argumente für eine bestimmte Konklusion angesehen werden, die durch diese Regeln abgeleitet wird. Da Konsistenz unter den unsicheren Regeln keine Notwendigkeit ist, können Argumente für widersprüchliche Literale existieren und durch eine dialektische Analyse können Argumente und Gegenargumente in Beziehung zueinander gesetzt werden. Mit der *Generalized Specificity* und der possibilistischen Präferenzrelation nach P-DeLP wurden zwei Beispiele von Vergleichsrelationen unter Argumenten vorgestellt, die bei der dialektischen Analyse benutzt werden können.

3 Logikbasierte Argumentation nach Besnard und Hunter

In diesem Kapitel wird der Ansatz der logikbasierten Argumentation nach [BH01] und [BH06] vorgestellt und beschrieben. Allerdings wird hier im Gegensatz zum vorangegangenen Kapitel der Schwerpunkt auf die Betrachtung des Algorithmus zur Berechnung von Argumentationsbäumen nach [BH06] gelegt und nicht auf eine ausführliche formale Beschreibung der Sprache. Eine Adaptierung dieses Algorithmus für die Verwendung in einem verteilten argumentationsfähigen System ist jedoch nicht gelungen. Die Erläuterung dazu folgt in Abschnitt 4.5.

3.1 Einführung

Der Ansatz von Besnard und Hunter [BH01] benutzt einen klassischen-logischen Rahmen zur Realisierung von logikbasierter Argumentation. Anders als bei dem Ansatz von García und Simari gibt es keine Strukturen für Regeln zur Ableitung weiteren Wissens. Das Wissen eines Agenten wird vielmehr durch eine Menge propositionallogischer Formeln dargestellt und Ableitungen werden durch klassisch-logische Deduktion realisiert, die in diesem Kapitel durch den Operator $\vdash$ dargestellt wird.

Die Wissensbasis eines Agenten werde mit Δ bezeichnet und ist eine potenziell inkonsistente Menge klassisch-logischer Formeln. Für jede Teilmenge $\Phi \subseteq \Delta$ sei eine explizite Ordnung unter den Elementen $\Phi = (\alpha_1, \ldots, \alpha_n)$ gegeben. Diese hat im Weiteren keine semantische Bedeutung, sondern dient nur der eindeutigen Bezeichnung von Teilmengen, da es erforderlich sein wird, einer Menge klassisch-logischer Formeln $\{\alpha_1, \ldots, \alpha_n\}$ die Konjunktion $\alpha_1 \wedge \cdots \wedge \alpha_n$ eindeutig zuzuordnen und somit die exponentiell vielen äquivalenten Konjunktionen, die sich aus verschiedenen Reihenfolgen der Formeln aus $\{\alpha_1, \ldots, \alpha_n\}$ ergeben, aus der Betrachtung auszuschließen, vgl. [BH01].

Erreicht wird diese explizite Ordnung einer Teilmenge, indem der Menge Δ eine totale Ordnung der Elemente hinzugefügt wird, von der die Ordnungen aller Teilmengen induziert werden.

Ein Argument für eine klassisch-logische Formel α ist eine minimale widerspruchsfreie Teilmenge der Wissensbasis Δ, aus der α abgeleitet werden kann.

Definition 3.1 (Argument)**.** Sei Δ eine Wissensbasis. Ein *Argument* ist ein Paar $\langle \Phi, \alpha \rangle$, so dass die folgenden vier Bedingungen erfüllt sind:

1. $\Phi \subseteq \Delta$
2. $\Phi \nvdash \bot$
3. $\Phi \vdash \alpha$
4. Es existiert kein $\Phi' \subset \Phi$ mit $\Phi' \vdash \alpha$.

Φ heißt der *Support* und α die Konklusion des Arguments.

Beispiel 3.1 ([BH01], Beispiel 3.2)**.** Sei

$$\Delta = \{\alpha, \alpha \Rightarrow \beta, \gamma \Rightarrow \neg\beta, \gamma, \delta, \delta \Rightarrow \beta, \neg\alpha, \neg\gamma\}$$

die Wissensbasis eines Agenten. Einige Argumente sind

$$\langle \{\alpha, \alpha \Rightarrow \beta\}, \beta \rangle,$$
$$\langle \{\neg\alpha\}, \neg\alpha \rangle,$$
$$\langle \{\alpha \Rightarrow \beta\}, \neg\alpha \vee \beta \rangle,$$
$$\langle \{\neg\gamma\}, \delta \Rightarrow \neg\gamma \rangle.$$

Eine spezielle Eigenschaft von Argumenten ist die *Zurückhaltung*. Ein Argument ist zurückhaltender als ein anderes, wenn es knapper und allgemeiner ist.

Definition 3.2 (Zurückhaltung)**.** Ein Argument $\langle \Phi, \alpha \rangle$ heißt *zurückhaltender* als ein Argument $\langle \Psi, \beta \rangle$, g. d. w. $\Phi \subseteq \Psi$ und $\beta \vdash \alpha$.

Beispiel 3.2. Bezüglich der Wissensbasis Δ aus Beispiel 3.1 ist das Argument $\langle \{\alpha\}, \alpha \vee \beta \rangle$ zurückhaltender als das Argument $\langle \{\alpha, \alpha \Rightarrow \beta\}, \beta \rangle$.

Da Konsistenz der Wissensbasis Δ nicht vorausgesetzt wird, kann es Argumente für widersprüchliche Formeln geben.

Definition 3.3 (Gegenargument)**.** Ein *Gegenargument* zu einem Argument $\langle \Phi, \alpha \rangle$ ist ein Argument $\langle \Psi, \neg(\varphi_1 \wedge \cdots \wedge \varphi_n) \rangle$ mit $\{\varphi_1, \ldots, \varphi_n\} \subseteq \Phi$

Im Gegensatz zu [GS02] werden allerdings weitere Einschränkungen an die Klasse der zu betrachtenden Gegenargumente gemacht:

Definition 3.4 (Maximal zurückhaltendes Gegenargument)**.** $\langle \Psi, \beta \rangle$ heißt *maximal zurückhaltendes Gegenargument* zu $\langle \Phi, \alpha \rangle$, g. d. w. $\langle \Psi, \beta \rangle$ ein Gegenargument von $\langle \Phi, \alpha \rangle$ ist und kein Gegenargument zu $\langle \Phi, \alpha \rangle$ existiert, das zurückhaltender ist als $\langle \Psi, \beta \rangle$.

Definition 3.5 (Kanonisches Gegenargument)**.** Ein Argument $\langle \Psi, \neg(\varphi_1 \wedge \cdots \wedge \varphi_n)\rangle$ heißt *kanonisches Gegenargument* von $\langle \Phi, \alpha\rangle$, g. d. w. es maximal zurückhaltend ist und $(\varphi_1, \ldots, \varphi_n)$ die kanonische Ordnung von Φ ist.

Durch ausschließliche Betrachtung von kanonischen Gegenargumenten werden nur solche Argumente betrachtet, deren Konklusion so allgemein wie möglich ist. Dies stellt eine einheitliche Struktur von Argumenten sicher.

Wie in [GS02] benutzt auch der Ansatz von [BH01] eine Baumstruktur zur Analyse von Argumentationsfolgen:

Definition 3.6 (Vollständiger Argumentationsbaum)**.** Ein *vollständiger Argumentationsbaum* zu einer klassisch-logischen Formel α ist ein Baum, dessen Knoten mit Argumenten bezeichnet sind, so dass die folgenden drei Bedingungen gelten:

1. Die Wurzel ist ein Argument für α.
2. Für keinen Knoten $\langle \Phi, \beta\rangle$ mit Vorgängerknoten $\langle \Phi_1, \beta_1\rangle, \ldots, \langle \Phi_n, \beta_n\rangle$ ist Φ eine Teilmenge von $\Phi_1 \cup \cdots \cup \Phi_n$.
3. Die Kindsknoten eines Knotens N bestehen aus allen kanonischen Gegenargumenten von N, die Bedingung 2 erfüllen.

Notation. Da im Folgenden nur kanonische Gegenargumente betrachtet werden, deren Konklusion stets die negierte Konjunktion des Supports des vorhergehenden Arguments ist, wird ab sofort das Symbol $\diamond$ als Abkürzung für die Konklusion von Gegenargumenten benutzt. Ist also $\langle \{\varphi_1, \ldots, \varphi_n\}, \alpha\rangle$ ein Argument, so wird ein kanonisches Gegenargument mit $\langle \Psi, \diamond\rangle$ abgekürzt, wobei in diesem Fall $\diamond$ für die Formel $\neg(\varphi_1 \wedge \cdots \wedge \varphi_n)$ steht.

Beispiel 3.3 ([BH06], Beispiel 4)**.** Sei die Wissensbasis Δ gegeben durch

$$\Delta = \{\alpha \vee \beta, \alpha \Leftarrow \gamma, \neg\gamma, \neg\beta, \delta \Leftrightarrow \beta\}.$$

Für die Formel $\alpha \vee \neg\delta$ können die beiden in Abbildung 3.1 dargestellten Argumentationsbäume gebildet werden.

3.2 Ein Algorithmus zur Generierung gültiger Argumentationsfolgen

3.2.1 Übersicht

In diesem Abschnitt wird ein Algorithmus beschrieben, der auf mehr oder weniger effiziente Weise vollständige Argumentationsbäume rekursiv generiert. Der Algorithmus repräsentiert dazu eine logische Wissensbasis in einer Graphstruktur, die auf den minimalen inkonsistenten Teilmengen der Wissensbasis basiert. Aus dieser Graphstruktur können gültige Argumentationsfolgen abgeleitet werden, die zur Kontruktion von vollständigen Argumentationsbäumen benutzt werden können.

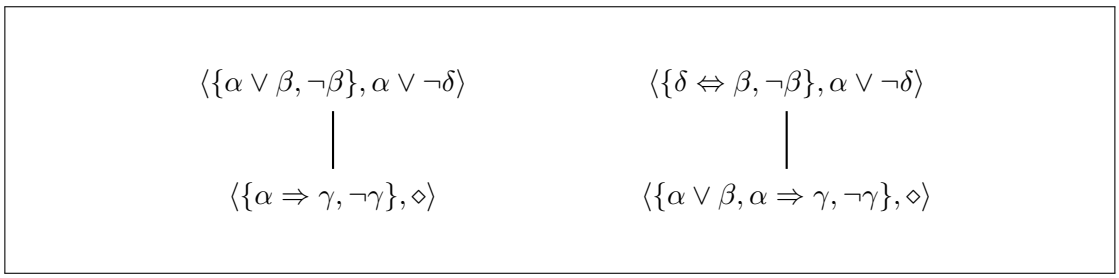

Abbildung 3.1: Vollständige Argumentationsbäume zu Beispiel 3.3

Anmerkung. Die Beweise zu den in diesem Abschnitt gegebenen Propositionen, Theoremen und Korollaren sind in [BH06] nachzulesen.

3.2.2 Minimale inkonsistente Teilmengen

Die Grundlage für die Existenz von Gegenargumenten zu einem Argument ist Inkonsistenz in der Wissensbasis. Diese Inkonsistenz kann durch die Bestimmung der minimalen inkonsistenten Teilmengen einer Wissensbasis strukturell erfasst werden:

Definition 3.7 (Minimale inkonsistente Teilmengen)**.** Sei Δ eine Wissensbasis. Die Menge der *minimalen inkonsistenten Teilmengen* $\text{MinIncon}(\Delta)$ von Δ ist definiert durch

$$\text{MinIncon}(\Delta) = \{X \subseteq \Delta | X \vdash \bot \wedge \forall Y \subset X : Y \nvdash \bot\}.$$

Minimale inkonsistente Teilmengen sind die Grundlage für die Konstruktion von kanonischen Gegenargumenten, wie die folgende Proposition zeigt:

Proposition 3.1 ([BH06], Proposition 1)**.** *Sei $\langle \Psi, \diamond \rangle$ ein kanonisches Gegenargument zu einem Argument $\langle \Phi, \alpha \rangle$. Dann gibt es ein $X \in MinIncon(\Delta)$ mit $\Psi \subset X$.*

Falls ein Argument $\langle \Phi, \alpha \rangle$ die Wurzel eines vollständigen Argumentationsbaumes ist, muss es jedoch nicht notwendigerweise ein $X \in \text{MinIncon}(\Delta)$ mit $\Phi \subseteq X$ geben.

Beispiel 3.4 ([BH06], Beispiel 5)**.** Sei $\Delta = \{\alpha, \neg\alpha, \beta, \neg\beta\}$. Die minimalen inkonsistenten Teilmengen X_1, X_2 von Δ sind

$$\begin{aligned} X_1 &= \{\alpha, \neg\alpha\}, \\ X_2 &= \{\beta, \neg\beta\}. \end{aligned}$$

Für das Wurzelargument eines Argumentationsbaumes $\langle \{\alpha, \beta\}, \alpha \wedge \beta \rangle$ gilt jedoch

$$\{\alpha, \beta\} \not\subset X_1 \quad \text{und} \quad \{\alpha, \beta\} \not\subset X_2.$$

Die nun folgenden Propositionen führen zu einer konkreteren Konstruktionsvorschrift für kanonische Gegenargumente. Es gilt zunächst:

Proposition 3.2 ([BH06], Proposition 2)**.** *Sei $\langle \Phi, \alpha \rangle$ ein Argument mit $\Phi \neq \emptyset$. Für alle $X \in MinIncon(\Delta)$ gilt: Ist Φ eine Teilmenge von X, so ist $\langle X \setminus \Phi, \diamond \rangle$ ein kanonisches Gegenargument zu $\langle \Phi, \alpha \rangle$.*

Beispiel 3.5 ([BH06], Beispiel 6)**.** Sei $\text{MinIncon}(\Delta) = \{\{\beta, \neg\beta \vee \neg\gamma, \gamma\}\}$ und das Argument

$$\langle \{\neg\beta \vee \neg\gamma\}, \neg(\beta \wedge \gamma) \rangle$$

gegeben. Es ist $\{\neg\beta \vee \neg\gamma\}$ eine Teilmenge von $\{\beta, \neg\beta \vee \neg\gamma, \gamma\}$ und somit ist

$$\langle \{\beta, \neg\beta \vee \neg\gamma, \gamma\} \setminus \{\neg\beta \vee \neg\gamma\}, \diamond \rangle$$

ein kanonisches Gegenargument zu $\langle \{\neg\beta \vee \neg\gamma\}, \neg(\beta \wedge \gamma) \rangle$.

Die obige Proposition kann verallgemeinert werden zu:

Proposition 3.3 ([BH06], Proposition 3)**.** *Sei $\langle \Phi, \alpha \rangle$ ein Argument. Es gilt für alle $X \in MinIncon(\Delta)$: Falls $\Phi \cap X \neq \emptyset$ und kein $Y \in MinIncon(\Delta)$ mit $(Y \setminus \Phi) \subset (X \setminus \Phi)$ existiert, so ist $\langle X \setminus \Phi, \diamond \rangle$ ein kanonisches Gegenargument zu $\langle \Phi, \alpha \rangle$.*

Beispiel 3.6 ([BH06], Beispiel 7)**.** Sei $\text{MinIncon}(\Delta) = \{X_1, X_2\}$ mit $X_1 = \{\alpha, \neg\alpha\}$ und $X_2 = \{\beta, \neg\beta\}$. Das Argument $\langle \Phi, \alpha \wedge \beta \rangle$ mit $\Phi = \{\alpha, \beta\}$ besitzt die beiden kanonischen Gegenargumente $\langle X_1 \setminus \Phi, \diamond \rangle$ und $\langle X_2 \setminus \Phi, \diamond \rangle$.

Es gilt weiterhin folgende Umkehrung:

Proposition 3.4 ([BH06], Proposition 4)**.** *Wenn $\langle \Psi, \diamond \rangle$ ein kanonisches Gegenargument zu $\langle \Phi, \alpha \rangle$ ist, so existiert ein X mit $X \in MinIncon(\Delta)$ und $\Psi = X \setminus \Phi$.*

Beispiel 3.7 ([BH06], Beispiel 8)**.** Sei $\Delta = \{\beta, \beta \Rightarrow \alpha, \beta \Rightarrow \neg\alpha\}$. Zu Δ existiert genau eine minimale inkonsistente Teilmenge $X = \{\Delta\}$. Zum Argument $\langle \{\beta, \beta \Rightarrow \alpha\}, \alpha \rangle$ existiert genau ein kanonisches Gegenargument $\langle \{\beta \Rightarrow \neg\alpha\}, \diamond \rangle$. Dabei gilt

$$\{\beta \Rightarrow \neg\alpha\} = X \setminus \{\beta, \beta \Rightarrow \alpha\}.$$

Es lässt sich zeigen, dass die minimalen inkonsistenten Teilmengen einer Wissensbasis Δ nicht notwendigerweise disjunkt sind. Vielmehr lassen sich Gegenargumente zu Argumenten über die Schnitte von minimalen inkonsistenten Teilmengen finden:

Proposition 3.5 ([BH06], Proposition 6)**.** *Sei $\langle \Phi, \diamond \rangle$ ein kanonisches Gegenargument zu einem Argument $\langle \Phi', \alpha \rangle$ und sei $\langle \Psi, \diamond \rangle$ ein kanonisches Gegenargument zu $\langle \Phi, \diamond \rangle$. Dann existiert ein $Y \in MinIncon(\Delta)$ mit $\Psi \subset Y$, so dass für alle $X \in MinIncon(\Delta)$ mit $\Phi \subset X$ gilt: $X \cap Y \neq \emptyset$.*

Die obigen Resultate zeigen, dass aus $\text{MinIncon}(\Delta)$ alle kanonischen Gegenargumente eines beliebigen Arguments gefunden werden können, ohne dass Erfüllbarkeitstests für klassische Logik benutzt werden müssen.

3.2.3 Der Algorithmus im Überblick

Der Algorithmus von Besnard und Hunter [BH06] ermittelt zunächst die minimalen inkonsistenten Teilmengen einer Wissensbasis, aus denen ein Graph – *Kompilierung* genannt – berechnet wird. Aus diesem Graphen kann zu einem gegebenen Argument der entsprechende vollständige Argumentationsbaum abgeleitet werden. Die Kompilierung ist also eine Struktur, die für eine Wissensbasis nur einmalig berechnet werden muss. Verschiedene vollständige Argumentationsbäume können dann effizient daraus abgelesen werden, vgl. [BH06].

Definition 3.8 (Kompilierung)**.** Sei Δ eine Wissensbasis. Dann ist

$$\mathit{Compilation}(\Delta) = (N, A)$$

ein ungerichteter Graph mit $N = \text{MinIncon}(\Delta)$ und

$$A = \{(X, Y) | X, Y \in N \wedge X \cap Y \neq \emptyset)\}$$

und heißt *Kompilierung* von Δ.

Beispiel 3.8 ([BH06], Beispiel 11)**.** Sei $\Delta = \{\delta \wedge \neg\alpha, \alpha, \neg\alpha, \alpha \vee \beta, \neg\beta\}$. Es ist

$$\text{MinIncon}(\Delta) = \{X_1, X_2, X_3, X_4\}$$

mit

$$\begin{aligned} X_1 &= \{\delta \wedge \neg\alpha, \alpha\}, \\ X_2 &= \{\alpha, \neg\alpha\}, \\ X_3 &= \{\neg\alpha, \alpha \vee \beta, \neg\beta\}, \\ X_4 &= \{\delta \wedge \neg\alpha, \alpha \vee \beta, \neg\beta\}. \end{aligned}$$

Dann ist $\mathit{Compilation}(\Delta) = (N, A)$ mit $N = \{X_1, X_2, X_3, X_4\}$ und

$$A = \{(X_1, X_2), (X_2, X_3), (X_3, X_4), (X_1, X_4)\}.$$

Die folgenden beiden Resultate aus [BH06] deuten an, wie Argumentationsbäume aus Teilgraphen einer Kompilierung generiert werden können.

Theorem 3.1 ([BH06], Theorem 1)**.** *Sei $\mathit{Compilation}(\Delta) = (N, A)$, $\langle \Phi, \alpha \rangle$ die Wurzel eines vollständigen Argumentationsbaumes T und $\langle \Psi, \diamond \rangle$ ein Kind von $\langle \Phi, \alpha \rangle$. Dann ist der Subbaum von T mit Wurzel $\langle \Psi, \diamond \rangle$ isomorph zu einem Subgraph von $\mathit{Compilation}(\Delta)$.*

Korollar 3.1 ([BH06], Korollar 1)**.** *Sei $\mathit{Compilation}(\Delta) = (N, A)$ und $\langle \Psi, \diamond \rangle$ ein kanonisches Gegenargument zu einem kanonischen Gegenargument $\langle \Phi, \diamond \rangle$. Dann existiert ein $(X, Y) \in A$ mit $\Phi \subset X$ und $\Psi \subset Y$.*

Durch geeignetes Abwandern der Kanten von $\mathit{Compilation}(\Delta)$ kann also ein vollständiger Argumentationsbaum ermittelt werden.

3.2.4 Generierung von Argumenten

Sei $\langle \Phi, \alpha \rangle$ ein Argument, zu dem ein vollständiger Argumentationsbaum berechnet werden soll. Zunächst werden dazu alle Knoten von *Compilation*(Δ) gesucht, die einen nicht-leeren Schnitt mit Φ haben. Ist X ein Knoten von *Compilation*(Δ) mit $X \cap \Phi \neq \emptyset$, so ist $X \setminus \Phi$ der Support eines Gegenarguments zu $\langle \Phi, \alpha \rangle$ und, falls es keinen Knoten Y von *Compilation*(Δ) gibt mit $\Phi \cap Y \neq \emptyset$ und $(Y \cap \Phi) \subset (X \cap \Phi)$, so ist $\langle X \setminus \Phi, \diamond \rangle$ nach Proposition 3.3 sogar ein kanonisches Gegenargument zu $\langle \Phi, \alpha \rangle$.

Zu jedem Gegenargument wird nun rekursiv nach kanonischen Gegenargumenten gesucht. Sei dazu $\langle \Gamma_j, \diamond \rangle$ ein kanonisches Gegenargument in einem vollständigen Argumentationsbaum und $\langle \Gamma_i, \diamond \rangle$ sein Elternknoten. Dann existiert nach Proposition 3.1 ein Knoten X mit $\Gamma_j = X \setminus \Gamma_i$, also $\Gamma_j \subset X$. Existiert nun eine Kante (X, Y) mit $Y \cap \Gamma_j \neq \emptyset$ und keine Kante (X, Z) mit $(Z \setminus \Gamma_j) \subset (Y \setminus \Gamma_j)$, so ist nach Proposition 3.3 $\langle Y \setminus \Gamma_j, \diamond \rangle$ ein kanonisches Gegenargument zu $\langle \Gamma_j, \diamond \rangle$. Um die Wiederholung von Argumenten zu vermeiden, müssen alle benutzten Kanten von *Compilation*(Δ) protokolliert werden. So kann auf effiziente Weise ein vollständiger Argumentationsbaum berechnet werden.

Der Algorithmus `Undercuts` [BH06] erhält als Parameter die Wurzel eines vollständigen Argumentationsbaumes $\langle \Phi, \alpha \rangle$ und die Knoten- bzw. Kantenmenge des Graphen *Compilation*$(\Delta) = (N, A)$. Als Rückgabe liefert er die Knoten des dazugehörigen vollständigen Argumentationsbaumes. Es folgt eine formale Beschreibung des Algorithmus `Undercuts` in Pseudocode:

```
1  Undercuts(⟨Φ, α⟩, N, A)
2    FirstLevelSets = FirstLevel(⟨Φ, α⟩, N, A)
3    Output = ∅
4    while FirstLevelSets ≠ ∅ do
5      remove X from FirstLevelSets
6      add "⟨X \ Φ, ◇⟩ is an undercut of ⟨Φ, α⟩" to Output
7      add Subcuts(X \ Φ, N \ {X}, A, X, Φ) to Output
8    return Output
```

Algorithmus 3.1: `Undercuts`

Der Algorithmus `FirstLevel` [BH06] erhält als Parameter die Wurzel eines vollständigen Argumentationsbaumes $\langle \Phi, \alpha \rangle$ und die Knoten- bzw. Kantenmenge des Graphen *Compilation*$(\Delta) = (N, A)$. Als Rückgabe liefert er alle Knoten, die für die Konstruktion kanonischer Gegenargumente der Wurzel benutzt werden.

Es folgt eine formale Beschreibung des Algorithmus `FirstLevel` in Pseudocode:

```
1  FirstLevel(⟨Φ, α⟩, N, A)
2    Candidates = {X ∈ N | Φ ∩ X ≠ ∅}
3    Output = ∅
4    while Candidates ≠ ∅ do
5      remove X from Candidates
6      if there is no Y ∈ Candidates ∪ Output
7           such that (Y \ Φ) ⊂ (X \ Φ) then
8        add X to Output
9    return Output
```

Algorithmus 3.2: `FirstLevel`

Der Algorithmus `Subcuts` [BH06] benutzt Tiefensuche, um rekursiv Gegenargumente zu Gegenargumenten zu berechnen. Als Parameter erhält er den Support Γ eines Arguments, die Menge aller Knoten M, die noch nicht besucht wurden, einen Knoten X und die Vereinigung aller Supports der vorhergehenden Argumente S. Es folgt eine formale Beschreibung des Algorithmus `Subcuts` in Pseudocode:

```
1  Subcuts(Γ, M, A, X, S)
2    Output = ∅
3    for all Y ∈ M do
4      if (X, Y) ∈ A and Y ⊈ S and Y ∩ Γ ≠ ∅
5            and there is no (X, Z) ∈ A
6                  such that (Z \ Γ) ⊂ (Y \ Γ)
7      then
8        add "⟨Y \ Γ, ◇⟩ is an undercut of ⟨Γ, ◇⟩" to Output
9        add Subcuts(Y \ Γ, M \ {Y}, A, X, S ∪ Y) to Output
10   return Output
```

Algorithmus 3.3: `Subcuts`

Durch Mitführung der Supports aller vorhergehenden Argumente und Vergleich des Supports eines potenziellen Gegenarguments wird sichergestellt, dass jeder Support eines neuen Arguments nicht Teilmenge der Vereinigung der vorherigen Supports ist. Die Wiederholung von Argumenten wird dadurch vermieden, dass benutzte Knoten aus der Menge M entfernt werden.

Beispiel 3.9 ([BH06], Beispiel 12)**.** Sei

$$\Delta = \{\alpha, \neg\alpha, \beta \wedge \neg\beta, \gamma, \gamma \Rightarrow \delta, \neg\delta, \neg\gamma \vee \delta\}$$

mit der Kompilierung $\mathit{Compilation}(\Delta) = (N, A)$ gegeben, wobei

$$\begin{aligned} N &= \{X_1, X_2, X_3, X_4\} \text{ und} \\ A &= \{X_4, X_3\} \end{aligned}$$

mit

$$\begin{aligned} X_1 &= \{\alpha, \neg\alpha\}, \\ X_2 &= \{\beta \wedge \neg\beta\}, \\ X_3 &= \{\gamma, \gamma \Rightarrow \delta, \neg\delta\}, \\ X_4 &= \{\gamma, \neg\gamma \vee \delta, \neg\delta\} \end{aligned}$$

ist. Betrachte das Argument $\langle\{\gamma, \gamma \Rightarrow \delta\}, \delta \vee \beta\rangle$ als Wurzel eines Argumentationsbaumes. Dann ist `FirstLevelSets` $= \{X_3\}$ und als Support eines kanonischen Gegenarguments ergibt sich

$$X_3 \setminus \{\gamma, \gamma \Rightarrow \delta\} = \{\neg\delta\}.$$

Wegen der Kante (X_4, X_3) ergibt sich als Support eines Gegenarguments zum obigen Gegenargument

$$X_4 \setminus \{\neg\delta\} = \{\gamma, \neg\gamma \vee \delta\}.$$

Da es keine weiteren Kanten gibt, entsteht der in Abbildung 3.2 dargestellte vollständige Argumentationsbaum.

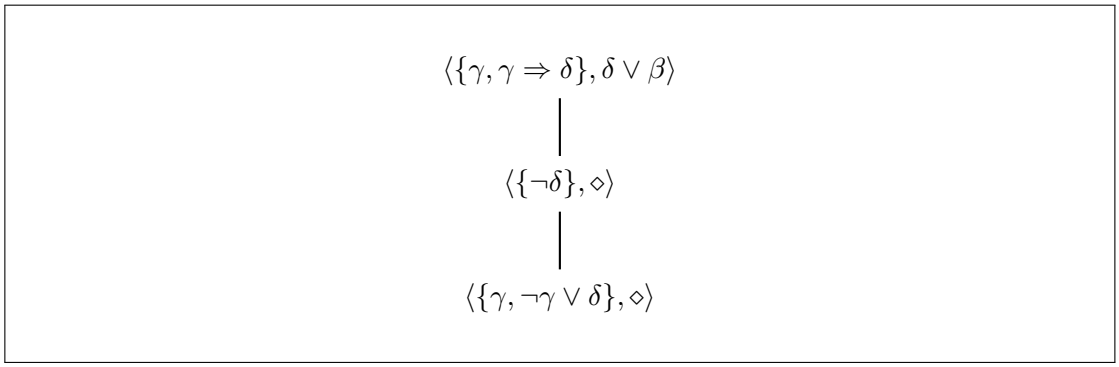

Abbildung 3.2: Vollständiger Argumentationsbaum zu Beispiel 3.9

Die folgenden Theoreme zeigen die Korrektheit und Vollständigkeit der oben beschriebenen Algorithmen.

Theorem 3.2 ([BH06], Theorem 2)**.** *Sei Δ eine Wissensbasis, Compilation$(\Delta) = (N, A)$ die zugehörige Kompilierung und $\langle \Gamma, \alpha \rangle$ ein Argument. Für alle $X \in N$ mit $\Gamma \subset X$ gilt: Falls $\langle \Psi, \diamond \rangle$ ein Gegenargument zu $\langle \Gamma, \alpha \rangle$ ist, dann ist*

- *entweder $\Psi = X \setminus \Gamma$,*
- *oder $\Psi = Y \setminus \Gamma$ für ein $Y \in N$ mit $(X, Y) \in A$ und $Y \cap \Gamma \neq \emptyset$.*

Theorem 3.3 ([BH06], Theorem 3)**.** *Sei Δ eine Wissensbasis, Compilation$(\Delta) = (N, A)$ die zugehörige Kompilierung und $\langle \Phi, \alpha \rangle$ ein Argument. Der Aufruf des Algorithmus `Undercuts(`$\langle \Phi, \alpha \rangle$`,N,A)` liefert die kanonischen Gegenargumente eines vollständigen Argumentationsbaumes T mit der Wurzel $\langle \Phi, \alpha \rangle$ zurück.*

3.2.5 Konstruktion einer Kompilierung

Die Berechnung der Kompilierung *Compilation*$(\Delta) = (N, A)$ einer Wissensbasis Δ findet im Algorithmus `GenerateCompilation` statt, der den Algorithmus `GenerateMinIncons` benutzt. Zur Definition dieser Algorithmen werden die folgenden Funktionen benötigt:

- `Subsets(`Δ`,`C`)`: liefert alle Teilmengen von Δ mit Kardinalität C
- `NotSuperSet(`Φ`,`Θ`)`: liefert *wahr*, falls es kein Element von Θ gibt, das eine Teilmenge von Φ ist, und ansonsten *falsch*
- `Inconsistent(`Φ`)`: liefert *wahr*, g. d. w. $\Phi \vdash \bot$

Der Algorithmus `GenerateCompilation` [BH06] berechnet aus den minimalen inkonsistenten Teilmengen alle Paare, die einen nicht-leeren Schnitt haben. Es folgt eine formale Beschreibung des Algorithmus `GenerateCompilation` in Pseudocode:

```
1  GenerateCompilation(Δ)
2    N = GenerateMinIncons(Δ)
3    M = N
4    A = ∅
5    while M ≠ ∅ do
6      remove an arbitrary element X from M
7      Temp = M
8      while Temp ≠ ∅
9        remove an arbitrary element Y from Temp
10       if X ∩ Y ≠ ∅ then
11         A = A ∪ {(X,Y)}
12   return (N,A)
```

Algorithmus 3.4: `GenerateCompilation`

Der Algorithmus `GenerateMinIncons` [BH06] berechnet alle minimalen inkonsistenten Teilmengen der Wissensbasis Δ, indem mit wachsender Kardinaliät alle Teilmengen von Δ betrachtet werden, die nicht schon eine echte inkonsistente Teilmenge enthalten. Es folgt eine formale Beschreibung des Algorithmus `GenerateMinIncons` in Pseudocode:

```
1  GenerateMinIncons(Δ)
2    Θ = ∅
3    C = 1
4    while C ≤ |Δ| do
5      Ω = Subsets(Δ, C)
6      while Ω ≠ ∅ do
7        let Φ be an arbitrary element of Ω
8        if NotSuperSet(Φ,Θ) and Inconsistent(Φ) then
9          Θ = Θ ∪ {Φ}
10       Ω = Ω \ {Φ}
11     C = C + 1
12   return Θ
```

Algorithmus 3.5: `GenerateMinIncons`

Eine Diskussion zur Effizienz der vorgestellten Algorithmen ist in [BH06] nachzulesen.

3.3 Zusammenfassung

Der Ansatz der logikbasierten Argumentation nach Besnard und Hunter benutzt im Gegensatz zum Ansatz von García und Simari eine klassisch-propositionallogische Syntax zur Wissensrepräsentation und verfügt somit über keine expliziten Strukturen zur Darstellung von sicheren und unsicheren Regeln. Der Algorithmus zur Generierung gültiger Argumentationsfolgen benutzt die minimalen inkonsistenten Teilmengen der Wissensbasis Δ, um eine Graphstruktur zu konstruieren, aus der effizient Folgen von Argumenten und Gegenargumenten hergeleitet werden können. Dieser Algorithmus wurde formal vorgestellt und beschrieben.

4 Vergleich der Ansätze von García/Simari und Besnard/Hunter und Anpassung an ein verteiltes System

In diesem Kapitel werden die Ansätze logikbasierter Argumentation von García/Simari [GS02] und Besnard/Hunter [BH06] miteinander verglichen und an die Verwendung in einem verteilten System angepasst. Dazu wird die Sprache D-DeLP (DISTRIBUTED-DELP) auf der Grundlage der Sprache DeLP entwickelt, die in Kapitel 5 die Basis zur Entwicklung eines verteilten Argumentationsprozesses sein wird. Eine Adaptierung des Algorithmus von Besnard und Hunter für verteilte Systeme ist jedoch nicht möglich.

4.1 Sicheres und unsicheres Wissen

Durch die Verwendung von Strukturen für Fakten, sichere und unsichere Regeln, unterstützt DeLP eine explizite Unterscheidung zwischen sicherem und unsicherem Wissen. Der Ansatz von Besnard und Hunter behandelt jedes Wissensfragment gleich und unterstützt sicheres Wissen somit nicht direkt. Jedes Argument muss dort nur in sich konsistent sein und nicht auch in Bezug auf einen Wissenshintergrund.

Beispiel 4.1. Betrachte die folgende logikbasierte Wissensbasis nach Besnard und Hunter:

$$\Delta = \left\{ \begin{array}{l} A\_hat\_B\_Schaden\_zugefuegt, A\_mag\_B, \\ A\_mag\_B \Rightarrow \neg A\_hat\_B\_Schaden\_zugefuegt \end{array} \right\}.$$

Nach dem Ansatz von Besnard und Hunter wäre

$$\langle \{A\_mag\_B, A\_mag\_B \Rightarrow \neg A\_hat\_B\_Schaden\_zugefuegt\}, \neg A\_hat\_B\_Schaden\_zugefuegt \rangle$$

ein zulässiges Argument. In diesem Beispiel mag es jedoch sinnvoll sein, die logische Aussage $A\_hat\_B\_Schaden\_zugefuegt$ als Faktum anzusehen, da sie vielleicht von einem zuverlässigen Zeugen beobachtet wurde. Die Regel

$$A\_mag\_B \Rightarrow \neg A\_hat\_B\_Schaden\_zugefuegt$$

sollte somit nicht als strikte Implikation, sondern als eine *Default-* oder unsichere Regel behandelt werden, die nur angewendet werden soll, wenn die Konklusion nicht im Widerspruch zum sicherem Wissen steht. Ein derartige Modellierung mit Hilfe des Ansatzes von Besnard und Hunter ist aber nicht möglich, da grundsätzlich jedes Wissensfragment aus der Betrachtung ausgeschlossen werden kann.

Das Beispiel zeigt außerdem, dass die Existenz der Kontraposition für Implikationen nicht immer erwünscht ist. Die Wissensbasis Δ aus Beispiel 4.1 enthält das Literal A_mag_B, das als beobachtetes Faktum gelten könnte. Wegen der Kontraposition

$$A\_hat\_B\_Schaden\_zugefuegt \Rightarrow \neg A\_mag\_B$$

der Implikation

$$A\_mag\_B \Rightarrow \neg A\_hat\_B\_Schaden\_zugefuegt$$

ist aus Δ das Argument

$$\langle \{A\_mag\_B \Rightarrow \neg A\_hat\_B\_Schaden\_zugefuegt, \\ A\_hat\_B\_Schaden\_zugefuegt\}, \neg A\_mag\_B \rangle$$

ableitbar, das ein Argument gegen ein beobachtetes Faktum ist.

Durch die Unterscheidung von sicherem und unsicherem Wissen eignet sich DeLP besser zur Verwendung in einem argumentationsfähigen Multiagentensystem als der Ansatz von Besnard und Hunter. Gerade im Bereich der Rechtswissenschaften sind sowohl Fakten als auch unsichere Regeln die Grundlage für argumentative Prozesse. Argumente in einem Rechtsfall müssen stets konsistent mit den beobachteten Fakten sein, um unsinnige Schlussfolgerungen zu vermeiden.

4.2 Überführung der Ansätze

Im Gegensatz zum Ansatz von Besnard und Hunter aus Kapitel 3 verwendet der Ansatz von García und Simari aus Kapitel 2 eigene Strukturen, um Regeln darzustellen. Eine einfache Überführung von unsicheren Regeln eines DeLP-Programms zu Implikationen einer logischen Wissensbasis nach Besnard und Hunter, führt aufgrund der Möglichkeit der Kontraposition zu unterschiedlichem Antwortverhalten, wie das folgende Beispiel zeigt:

Beispiel 4.2. Sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ ein DeLP-Programm mit

$$\begin{aligned} \Pi_G &= \{a\}, \\ \Pi_R &= \emptyset, \\ \Delta_R &= \left\{ \begin{array}{l} (b \prec a), \\ (c \prec a), \\ (\neg c \prec b) \end{array} \right\}. \end{aligned}$$

In $\mathcal{P}$ ist das Literal b begründet, da es ein Argument $\langle\{(b \prec a)\}, b\rangle$ für b und es zu $\langle\{(b \prec a)\}, b\rangle$ keine Gegenargumente gibt. Wird $\mathcal{P}$ in eine logische Wissensbasis nach Besnard und Hunter überführt, indem unsichere Regeln zu Implikationen transformiert werden, erhält man eine Wissensbasis Δ mit

$$\Delta = \{a, a \Rightarrow b, a \Rightarrow c, b \Rightarrow \neg c\}.$$

Auch in Δ existiert ein Argument für b, nämlich $\langle\{a, a \Rightarrow b\}, b\rangle$. Allerdings existiert dazu auch ein kanonisches Gegenargument

$$\langle\{a, a \Rightarrow c, b \Rightarrow \neg c\}, \neg(a \wedge a \Rightarrow b)\rangle,$$

da die Implikation $c \Rightarrow \neg b$ als Kontraposition der Implikation $b \Rightarrow \neg c$ abgeleitet werden kann. Das Argument $\langle\{a, a \Rightarrow b\}, b\rangle$ für b wird widerlegt.

Somit kann die triviale Überführung eines DeLP-Programms zu einer logischen Wissensbasis nach Besnard und Hunter neue Argumente erzeugen und das Antwortverhalten beeinflussen. Die triviale Überführung einer logischen Wissensbasis nach Besnard und Hunter, bei der nur Literale und Implikationen erlaubt werden, in ein DeLP-Programm kann somit auch dazu führen, dass Argumente verloren gehen.

4.3 Maximal zurückhaltende Argumente in DeLP

Das Konzept der maximal zurückhaltenden Argumente kann unter gewissen Umständen für das DeLP-Framework übernommen werden, ohne dass das Antwortverhalten auf Anfragen davon beeinträchtigt wird. Ein Argument $\langle\mathcal{A}, h\rangle$ heißt zurückhaltender als ein Argument $\langle\mathcal{B}, q\rangle$, falls $\langle\mathcal{A}, h\rangle$ „allgemeiner“ als $\langle\mathcal{B}, q\rangle$ ist, vgl. Definition 3.2. In der DeLP-Terminologie bedeutet dies, dass, wenn nur maximal zurückhaltende Angriffe zugelassen werden, kein Angriff ein Sub-Argument eines anderen Angriffs sein darf. Für ein Argument $\langle\mathcal{A}, h\rangle$ und die Menge seiner maximal zurückhaltenden Angriffe $\{\langle\mathcal{B}_1, h_1\rangle, \ldots, \langle\mathcal{B}_n, h_n\rangle\}$ gilt also:

$$\forall i \in \{1, \ldots, n\} : \neg\exists j \in \{1, \ldots, n\} : i \neq j \wedge \mathcal{B}_i \subseteq \mathcal{B}_j.$$

Anders als bei Definition 2.17, in der gefordert wurde, dass kein Gegenargument ein Sub-Argument eines Vorgängers sein darf, darf nun auch ein Angriff kein Sub-Argument eines Geschwisterarguments sein, also eines Arguments, das dasselbe Argument angreift.

Um das Antwortverhalten nicht zu beeinflussen, ist es nötig, dass nur die Menge der Angriffe und nicht schon die Menge der Gegenargumente auf maximal zurückhaltende Argumente eingeschränkt wird, wie das folgende Beispiel zeigt:

Beispiel 4.3. Sei $\succ_{spec}$ die verwendete Präferenzrelation für Argumente und das folgende DeLP-Programm $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ gegeben:

$$\begin{aligned} \Pi_G &= \{a, b\}\,, \\ \Pi_R &= \emptyset, \\ \Delta_R &= \left\{ \begin{array}{l} (c \prec d), \\ (d \prec a, b), \\ (\neg c \prec \neg d), \\ (\neg d \prec b) \end{array} \right\}. \end{aligned}$$

$\mathcal{P}$ enthält die Argumente

$$\begin{aligned} \langle \mathcal{A}, c \rangle &= \langle \{(c \prec d), (d \prec a, b)\}, c \rangle, \\ \langle \mathcal{B}, \neg c \rangle &= \langle \{(\neg c \prec \neg d), (\neg d \prec b)\}, \neg c \rangle, \\ \langle \mathcal{C}, \neg d \rangle &= \langle \{(\neg d \prec b)\}, \neg d \rangle. \end{aligned}$$

Es sind dabei $\langle \mathcal{B}, \neg c \rangle$ und $\langle \mathcal{C}, \neg d \rangle$ Gegenargumente zu $\langle \mathcal{A}, c \rangle$ und das Argument $\langle \mathcal{C}, \neg d \rangle$ ist ein Sub-Argument von $\langle \mathcal{B}, \neg c \rangle$. Ohne Einschränkung auf maximal zurückhaltende Gegenargumente oder Angriffe, würde $\langle \mathcal{B}, \neg c \rangle$ als blockierender Angriff auf $\langle \mathcal{A}, c \rangle$ akzeptiert. $\langle \mathcal{C}, \neg d \rangle$ würde hingegen nicht akzeptiert, da das dazu unstimmige Sub-Argument $\langle \{(d \prec a, b)\}, d \rangle$ von $\langle \mathcal{A}, c \rangle$ spezifischer ist. Da es zu $\langle \mathcal{A}, c \rangle$ allerdings einen akzeptierten Angriff gibt, wird $\langle \mathcal{A}, c \rangle$ widerlegt.

Werden von vornherein nicht maximal zurückhaltende Gegenargumente ausgeschlossen, so wird $\langle \mathcal{B}, \neg c \rangle$ nicht als Gegenargument akzeptiert, da $\langle \mathcal{C}, \neg d \rangle$ zurückhaltender ist als $\langle \mathcal{B}, \neg c \rangle$. Aus demselben Grund wie oben wird aber auch $\langle \mathcal{C}, \neg d \rangle$ nicht als Angriff akzeptiert und da $\langle \mathcal{A}, c \rangle$ nicht angegriffen wird, wird $\langle \mathcal{A}, c \rangle$ auch nicht widerlegt.

Werden allerdings nur Angriffe auf maximale Zurückhaltung eingeschränkt, wird zunächst $\langle \mathcal{C}, \neg d \rangle$ mit der obigen Begründung gar nicht als Angriff akzeptiert. In der Menge der akzeptierten Angriffe $\{\langle \mathcal{B}, \neg c \rangle\}$ gibt es keine nicht maximal zurückhaltenden Argumente, so dass sich dasselbe Antwortverhalten zeigt wie ohne Einschränkung auf maximal zurückhaltende Angriffe.

Das folgende Theorem zeigt, dass, falls es Teilmengenbeziehungen zwischen Angriffen eines Arguments gibt, die Markierung der entsprechenden Knoten in einer dialektischen Analyse, in einer bestimmten Beziehung zueinander stehen:

Theorem 4.1. *Sei $\langle \mathcal{A}, h \rangle$ ein Argument und seien $\langle \mathcal{B}_1, h_1 \rangle$, $\langle \mathcal{B}_2, h_2 \rangle$ zwei Angriffe auf $\langle \mathcal{A}, h \rangle$. Ist $\langle \mathcal{B}_1, h_1 \rangle$ ein Sub-Argument von $\langle \mathcal{B}_2, h_2 \rangle$, so gilt für jeden markierten dialektischen Baum $\mathcal{T}^*$, in dem $\langle \mathcal{A}, h \rangle$ ein Knoten und $\langle \mathcal{B}_1, h_1 \rangle$, $\langle \mathcal{B}_2, h_2 \rangle$ Nachfolger von $\langle \mathcal{A}, h \rangle$ sind:*

$$\langle \mathcal{B}_1, h_1 \rangle \textit{ ist mit „D“ markiert} \Rightarrow \langle \mathcal{B}_2, h_2 \rangle \textit{ ist mit „D“ markiert}$$

Beweis. Sei $\mathcal{T}^*_{\langle\mathcal{D},g\rangle}$ ein markierter dialektische Baum mit Wurzel $\langle\mathcal{D},g\rangle$ und einem Knoten $\langle\mathcal{A},h\rangle$. Seien $\langle\mathcal{B}_1,h_1\rangle$ und $\langle\mathcal{B}_2,h_2\rangle$ Nachfolgeknoten von $\langle\mathcal{A},h\rangle$ mit $\mathcal{B}_1 \subseteq \mathcal{B}_2$, wobei $\langle\mathcal{B}_1,h_1\rangle$ mit „D“ markiert ist. Dann existiert ein Angriff $\langle\mathcal{C},h_3\rangle$ auf $\langle\mathcal{B}_1,h_1\rangle$, der mit „U“ markiert ist. Da $\mathcal{B}_1 \cup \mathcal{C} \cup \Pi_G \cup \Pi_R$ widersprüchlich ist und $\mathcal{B}_1 \subseteq \mathcal{B}_2$ gilt, ist auch $\mathcal{B}_2 \cup \mathcal{C} \cup \Pi_G \cup \Pi_R$ widersprüchlich. Also ist $\langle\mathcal{C},h_3\rangle$ ein Gegenargument zu $\langle\mathcal{B}_2,h_2\rangle$. $\langle\mathcal{C},h_3\rangle$ ist auch ein akzeptierter Angriff auf $\langle\mathcal{B}_2,h_2\rangle$, da das zu $\langle\mathcal{C},h_3\rangle$ in $\langle\mathcal{B}_1,h_1\rangle$ enthaltene unstimmige Sub-Argument wegen $\mathcal{B}_1 \subseteq \mathcal{B}_2$ auch in $\langle\mathcal{B}_2,h_2\rangle$ enthalten ist und der Angriff von $\langle\mathcal{C},h_3\rangle$ auf dieses Sub-Argument akzeptiert wird. Somit wird die Argumentationsfolge $\Lambda = [\langle\mathcal{D},g\rangle,\ldots,\langle\mathcal{B}_2,h_2\rangle,\langle\mathcal{C},h_3\rangle]$ akzeptiert. $\langle\mathcal{C},h_3\rangle$ ist dabei mit „U“ markiert, da alle akzeptierten Angriffe auf $\langle\mathcal{C},h_3\rangle$ unter $\langle\mathcal{B}_2,h_2\rangle$ wegen $\mathcal{B}_1 \subseteq \mathcal{B}_2$ die Bedingungen 3 und 4 für akzeptierte Argumentationsfolgen nicht verletzen und somit auch akzeptierte Angriffe auf $\langle\mathcal{C},h_3\rangle$ unter $\langle\mathcal{B}_1,h_1\rangle$ wären. Unter $\langle\mathcal{B}_1,h_1\rangle$ ist $\langle\mathcal{C},h_3\rangle$ mit „U“ markiert, d. h. alle Angriffe auf $\langle\mathcal{C},h_3\rangle$ unter $\langle\mathcal{B}_1,h_1\rangle$ sind mit „D“ markiert. Also sind auch alle Angriffe auf $\langle\mathcal{C},h_3\rangle$ unter $\langle\mathcal{B}_2,h_2\rangle$ mit „D“ markiert und somit ist $\langle\mathcal{C},h_3\rangle$ unter $\langle\mathcal{B}_2,h_2\rangle$ mit „U“ markiert. Da wenigstens ein Kind von $\langle\mathcal{B}_2,h_2\rangle$ mit „U“ markiert ist, wird also $\langle\mathcal{B}_2,h_2\rangle$ mit „D“ markiert. □

Korollar 4.1. *Sei $\mathcal{T}^*$ ein markierter dialektischer Baum und $\langle\mathcal{A},h\rangle$ ein innerer Knoten mit den Nachfolgeknoten $M := \{\langle\mathcal{B}_1,h_1\rangle,\ldots,\langle\mathcal{B}_n,h_n\rangle\}$. Sei $N \subseteq M$ die Menge der maximal zurückhaltenden Argumente aus M. Dann gilt:*

$$\begin{aligned}\langle\mathcal{A},h\rangle \textit{ wird mit „D“ markiert} &\Leftrightarrow \textit{ mind. ein } \langle\mathcal{B},g\rangle \in M \textit{ wird mit „U“ markiert}\\ &\Leftrightarrow \textit{ mind. ein } \langle\mathcal{B}',g'\rangle \in N \textit{ wird mit „U“ markiert.}\end{aligned}$$

Beweis. Die erste Äquivalenz entspricht der Definition von markierten dialektischen Bäumen und ist somit korrekt. Zeige nur die zweite Äquivalenz:

„⇒“: Sei ein Knoten $\langle\mathcal{B},g\rangle \in M$ mit „U“ markiert und angenommen alle Knoten $\langle\mathcal{B}',g'\rangle \in N$ seien mit „D“ markiert. Da N alle maximal zurückhaltenden Argumente aus M enthält, gilt für jedes $\langle\mathcal{C},i\rangle \in M \setminus N$, dass es ein zu $\langle\mathcal{C},i\rangle$ zurückhaltenderes Argument $\langle\mathcal{C}',i'\rangle \in N$ gibt. Da nach Annahme $\langle\mathcal{C}',i'\rangle$ mit „D“ markiert ist, gilt nach Theorem 4.1, dass auch $\langle\mathcal{C},i\rangle$ mit „D“ markiert ist. Somit ist jedes Argument in $M \setminus N$ mit „D“ markiert und da nach Annahme jedes Argument in N mit „D“ markiert ist, sind alle Argumente in $(M \setminus N) \cup N = M$ mit „D“ markiert. Das steht im Widerspruch dazu, dass $\langle\mathcal{B},g\rangle \in M$ mit „U“ markiert ist. Also gibt es mindestens ein $\langle\mathcal{B}',g'\rangle \in N$, das mit „U“ markiert sein muss.

„⇐“: Gilt, da $N \subseteq M$.

□

Dieses Korrolar besagt, dass bei der Konstruktion von dialektischen Bäumen nach [GS02] die nicht maximal zurückhaltenden Angriffe für ein Argument $\langle\mathcal{A},h\rangle$ vernachlässigbar sind, da sie die Markierung des Arguments $\langle\mathcal{A},h\rangle$ nicht beeinflussen.

4.4 Anpassung an ein verteiltes System

In diesem Abschnitt wird auf der Grundlage der Sprache DeLP ([GS02]) die Sprache D-DeLP (DISTRIBUTED-DeLP) definiert, welche die Basis für das in Kapitel 5 zu definierende verteilte argumentationsfähige Multiagentensystem bildet.

4.4.1 Wissensrepräsentation und Argumente

Um spezielle Präferenzrelationen, wie die in Unterabschnitt 2.3.3 vorgestellte possibilistische Präferenzrelation, zu unterstützen, erhält in D-DeLP jedes Wissensfragment ein Attribut α für Zusatzinformationen zu diesem Fragment. Im Hinblick auf possibilistische Präferenzrelationen sei α somit eine quantitative Abschätzung der Sicherheit mit $\alpha \in [0, 1]$. Je höher α, desto sicherer ist die α zugeordnete Information im Hinblick auf Revidierbarkeit durch andere Regeln. Die Semantik von α entspreche also den Konventionen in Unterabschnitt 2.3.3 und [CnSAG04]. Da die Angabe von Zusatzinformation optional sein soll, wird die Definition um die Möglichkeit $\alpha = -1$ erweitert, um anzuzeigen, dass keine Zusatzinformation für dieses Fragment vorhanden ist.

Definition 4.1 (Metainformation)**.** Eine Zahl $\alpha \in [0, 1] \cup \{-1\}$ heißt *Metainformation*. Dabei gelte:

- $\alpha = -1$: Für die α zugeordnete Information ist keine Metainformation gesetzt.
- $\alpha = 0$: Die α zugeordnete Information ist maximal unsicher.
- $\alpha = 1$: Die α zugeordnete Information ist sicher.
- $\alpha \in (0, 1)$ Die α zugeordnete Information hat einen Sicherheitsfaktor von α.

Ein argumentationsfähiges Multiagentensystem teilt das vorhandene Wissen in globales und lokales Wissen auf. Das globale Wissen enthält Fakten und sichere Regeln, auf die alle Agenten Zugriff haben, das lokale Wissen eines Agenten enthält unsichere Regeln. Die einzelnen Wissensfragmente sind analog wie in Unterabschnitt 2.2.1 unter Berücksichtigung von Metainformationen definiert durch:

Definition 4.2 (Fakt)**.** Ein *Fakt* ist ein Paar (h, α), so dass h ein Literal und α eine Metainformation zu h mit $\alpha = 1$ ist.

Definition 4.3 (Sichere Regel)**.** Eine *sichere Regel* ist ein Paar $((h \leftarrow b_1, \ldots, b_n), \alpha)$ mit Literalen $h, b_1, \ldots, b_n$ und α ist eine Metainformation zu $(h \leftarrow b_1, \ldots, b_n)$ mit $\alpha = 1$. h heißt *Kopfliteral* und $b_1, \ldots, b_n$ heißen *Rumpfliterale* von $(h \leftarrow b_1, \ldots, b_n)$.

Definition 4.4 (Unsichere Regel)**.** Eine *unsichere Regel* ist ein Paar $((h \prec b_1, \ldots, b_n), \alpha)$ mit Literalen $h, b_1, \ldots, b_n$ und α ist eine Metainformation zu $(h \prec b_1, \ldots, b_n)$ mit $\alpha \in [0, 1] \cup \{-1\}$. h heißt *Kopfliteral* und $b_1, \ldots, b_n$ heißen *Rumpfliterale* von $(h \prec b_1, \ldots, b_n)$.

Diese Definitionen entsprechen den Definitionen der atomaren Wissensfragmente der Sprache P-DeLP [CnSAG04], wobei die Angabe einer Metainformation durch die Möglichkeit $\alpha = -1$ optional ist. Die Semantik dieser Wissensfragmente ist mit der in Unterabschnitt 2.2.1 identisch.

Da D-DeLP in einem verteilten System benutzt werden soll, werden die Wissensfragmente entsprechend in eine globale und mehrere lokale Wissensbasen aufgeteilt. Der Begriff der Widersprüchlichkeit gilt analog zu Definition 2.7.

Definition 4.5 (Globale Wissensbasis)**.** Eine globale Wissensbasis Π ist eine widerspruchsfreie Menge von Fakten und sicheren Regeln.

Definition 4.6 (Lokale Wissensbasis)**.** Sei Δ eine Menge unsicherer Regeln und Π eine globale Wissensbasis. Ist $\Delta \cup \Pi$ widerspruchsfrei, so heißt Δ eine *lokale Wissensbasis* zu Π.

Zu beachten ist, dass lokale Wissensbasen jeweils konsistent mit der globalen Wissensbasis sein sollten. Das ist keine notwendige Einschränkung, aber sie stellt sicher, dass Agenten Gegenargumente nur zu Argumenten anderer Agenten aufstellen. Da in dieser Arbeit allerdings der verteilte und nicht der agenteninterne Argumentationsprozess die zentrale Rolle spielt, ist diese Einschränkung sinnvoll.

Die Definitionen 2.5 und 2.6 von Ableitungen gelten analog auch für D-DeLP:

Definition 4.7 (Unsichere Ableitung)**.** Sei Π eine globale, Δ eine lokale Wissensbasis zu Π und h ein Literal. Eine *unsichere Ableitung* von h aus Π und Δ, dargestellt durch $\Pi \cup \Delta \mathrel{|\!\sim} h$, besteht aus einer endlichen Folge $h_1, \ldots, h_n = h$ von Literalen, so dass für jedes Literal h_i $(1 \leq i \leq n)$ gilt:

1. $(h_i, 1)$ ist ein Fakt $((h_i, 1) \in \Pi)$ oder
2. es existiert eine sichere oder unsichere Regel in Π bzw. Δ mit einem Kopfliteral h_i und Rumpfliteralen $b_1, \ldots, b_k$, wobei jedes Element des Rumpfes b_l $(1 \leq l \leq k)$ gleich einem Element h_j mit $j < i$ ist.

Definition 4.8 (Sichere Ableitung)**.** Sei Π eine globale, Δ eine lokale Wissensbasis zu Π und h ein Literal. L hat eine *sichere Ableitung* aus Π, dargestellt durch $\Pi \vdash h$, falls entweder $(h, 1)$ ein Fakt ist $((h, 1) \in \Pi)$ oder alle Regeln zur Bestimmung der Folge $h_1, \ldots, h_k$ sichere Regeln aus Π sind.

Auf Grundlage dieser Sprachbestandteile werden analog zu den Beschreibungen aus Kapitel 2 Argumente und Gegenargumente definiert. Da jedoch in einem verteilten System eine Trennung von sicherem und unsicherem Wissen stattfindet, muss diese Trennung in den jeweiligen Definitionen formal berücksichtigt werden.

Definition 4.9 (Argument)**.** Sei Π eine globale, Δ eine lokale Wissensbasis zu Π und h ein Literal. $\langle \mathcal{A}, h \rangle$ ist ein *Argument* für h, g. d. w. $\mathcal{A} \subseteq \Delta$ ist und

1. eine Ableitung von h aus Π nach Definition 4.8 oder aus $\Pi \cup \Delta$ nach Definition 4.7 existiert,
2. die Menge $\Pi \cup \mathcal{A}$ nicht widersprüchlich und
3. $\mathcal{A}$ minimal bezüglich Mengeninklusion ist.

Das Literal h heißt die *Konklusion*, die vom *Support* $\mathcal{A}$ gestützt wird.

Definition 2.9 für ein Sub-Argument gilt analog für D-DeLP. Die Definition der Unstimmigkeit wird leicht angepasst:

Definition 4.10 (Unstimmigkeit)**.** Sei Π eine globale Wissensbasis. Zwei Literale h und h_1 sind *unstimmig*, g. d. w. die Menge $\Pi \cup \{h, h_1\}$ widersprüchlich ist.

Die Definition eines Gegenarguments 2.11 wird analog für D-DeLP übernommen:

Definition 4.11 (Gegenargument)**.** Ein Argument $\langle \mathcal{A}_1, h_1 \rangle$ ist ein *Gegenargument* zu einem Argument $\langle \mathcal{A}_2, h_2 \rangle$ an einem Punkt h, g. d. w. es ein Sub-Argument $\langle \mathcal{A}, h \rangle$ von $\langle \mathcal{A}_2, h_2 \rangle$ gibt, so dass h und h_1 unstimmig sind.

4.4.2 Argumentationsfolgen und dialektische Bäume

Im Folgenden gelte die Notation für Präferenzrelationen unter Argumenten aus Unterabschnitt 2.3.3 und die Definition 2.14, 2.15 und 2.16 zu Angriffen analog für D-DeLP.

Da der Argumentationsprozess verteilt abläuft, jeder Agent jedoch lokal Gegenargumente generieren muss, ist es nötig, den Informationsfluss zwischen den einzelnen Komponenten des Systems genauer zu spezifizieren. Jeder Agent verfügt lokal über eine Wissensbasis, die nur aus unsicheren Regeln besteht und hat Zugriff auf das globale Wissen des Systems, das aus Fakten und sicheren Regeln besteht. Definition 2.17 zu akzeptierten Argumentationsfolgen soll zum größten Teil übernommen werden und um geeignete Gegenargumente zu einem gegebenen Argument zu generieren, muss deshalb der Kontext des anzugreifenden Arguments berücksichtigt werden, also die gesamte vorangegangene Argumentationsfolge.

Definition 4.12 (Angriff auf Argumentationsfolgen)**.** Ein Argument $\langle \mathcal{B}, h \rangle$ heißt *Angriff auf eine Argumentationsfolge* $\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$, g. d. w. $\langle \mathcal{B}, h \rangle$ ein Angriff auf $\langle \mathcal{A}_n, h_n \rangle$ ist.

Wie in Abschnitt 4.3 gezeigt, verändert eine Einschränkung auf maximal zurückhaltende Angriffe das Antwortverhalten des Systems auf ein bestimmtes Literal nicht. Im Folgenden werden nur noch dialektische Bäume betrachtet, deren Knoten maximal zurückhaltend in Bezug auf ihre Geschwisterknoten sind. Die angepasste Definition einer akzeptierten Argumentationsfolge ist dann:

Definition 4.13 (Akzeptierte Argumentationsfolge)**.** Sei Π eine globale und Δ eine lokale Wissensbasis zu Π. Sei weiterhin $\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$ eine Folge von Argumenten. Λ heißt *akzeptierte Argumenationsfolge*, g. d. w. die folgenden Bedingungen erfüllt sind:

1. Λ ist eine endliche Folge.
2. Jedes Argument $\langle \mathcal{A}_i, h_i \rangle$ für $i > 0$ ist ein Angriff auf seinen Vorgänger $\langle \mathcal{A}_{i-1}, h_{i-1} \rangle$. Falls $\langle \mathcal{A}_i, h_i \rangle$ ein blockierender Angriff auf $\langle \mathcal{A}_{i-1}, h_{i-1} \rangle$ ist und $\langle \mathcal{A}_{i+1}, h_{i+1} \rangle$ existiert, ist $\langle \mathcal{A}_{i+1}, h_{i+1} \rangle$ ein ordentlicher Angriff auf $\langle \mathcal{A}_i, h_i \rangle$.
3. Für die Menge der h_1 unterstützenden Argumente $\Lambda_S = \{\langle \mathcal{A}_i, h_i \rangle | i \text{ ungerade}\}$ gilt: $\Pi_G \cup \Pi_R \cup \bigcup_{\langle \mathcal{A}, h \rangle \in \Lambda_S} \mathcal{A}$ ist nicht widersprüchlich.
4. Für die Menge der h_1 widersprechenden Argumente $\Lambda_I = \{\langle \mathcal{A}_i, h_i \rangle | i \text{ gerade}\}$ gilt: $\Pi_G \cup \Pi_R \cup \bigcup_{\langle \mathcal{A}, h \rangle \in \Lambda_I} \mathcal{A}$ ist nicht widersprüchlich.
5. Kein Argument $\langle \mathcal{A}_k, h_k \rangle$ ist ein Sub-Argument eines Arguments $\langle \mathcal{A}_i, h_i \rangle$ mit $i < k$.
6. Für kein Argument $\langle \mathcal{A}_j, h_j \rangle$ $(1 < j \leq n)$ existiert ein Argument $\langle \mathcal{A}', h' \rangle$ mit $\mathcal{A}' \subset \mathcal{A}$, so dass $\langle \mathcal{A}', h' \rangle$ ein akzeptierter Angriff auf $\langle \mathcal{A}_{j-1}, h_{j-1} \rangle$ ist ($\langle \mathcal{A}_j, h_j \rangle$ ist maximal zurückhaltend).

Die Definition 2.18 eines dialektischen Baumes wird übernommen:

Definition 4.14 (Dialektischer Baum)**.** Sei Π eine globale Wissensbasis und seien $\Delta_1, \ldots, \Delta_n$ lokale Wissensbasen zu Π. Ist $\langle \mathcal{A}_0, h_0 \rangle$ ein Argument bezüglich Π und einer lokalen Wissensbasis Δ_k $(1 \leq k \leq n)$, so ist der *dialektische Baum* zu $\langle \mathcal{A}_0, h_0 \rangle$, geschrieben $\mathcal{T}_{\langle \mathcal{A}_0, h_0 \rangle}$, definiert durch die folgenden beiden Bedingungen:

1. Die Wurzel von $\mathcal{T}$ ist $\langle \mathcal{A}_0, h_0 \rangle$.
2. Sei $\langle \mathcal{A}_n, h_n \rangle$ ein Knoten von $\mathcal{T}$ mit $i > 0$ und $\Lambda = [\langle \mathcal{A}_0, h_0 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$ die Folge der Knoten des Weges von der Wurzel zu $\langle \mathcal{A}_n, h_n \rangle$. Sei $\{\langle \mathcal{B}_1, q_1 \rangle, \ldots, \langle \mathcal{B}_k, q_k \rangle\}$ die Menge aller Angriffe auf $\langle \mathcal{A}_n, h_n \rangle$, so dass jedes $\langle \mathcal{B}_i, q_i \rangle$ mit $1 \leq i \leq k$ ein Argument bezüglich Π und einem Δ_l $(1 \leq l \leq n)$ ist. Für jedes Gegenargument $\langle \mathcal{B}_i, q_i \rangle$ mit $1 \leq i \leq k$, für das die Argumentationsfolge $\Lambda' = [\langle \mathcal{A}_0, h_0 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle, \langle \mathcal{B}_i, q_i \rangle]$ akzeptiert wird, erhält der Knoten $\langle \mathcal{A}_n, h_n \rangle$ das Kind $\langle \mathcal{B}_i, q_i \rangle$. Falls es kein Gegenargument zu $\langle \mathcal{A}_n, h_n \rangle$ oder kein $\langle \mathcal{B}_i, q_i \rangle$ gibt, so dass Λ' akzeptiert wird, so ist $\langle \mathcal{A}_n, h_n \rangle$ ein Blatt.

Die Definition 2.19 eines markierten dialektischen Baumes gelte analog für D-DeLP und die obige Definition eines dialaktischen Baumes. Ebenso werden die Definitionen 2.20 und 2.21 zur Begründung von Literalen und Antworten auf Anfragen für D-DeLP übernommen.

4.5 Verwendung des Algorithmus von Besnard und Hunter für D-DeLP

Auch wenn die Grundlage zur Konstruktion eines argumentationsfähigen Multiagentensytems DeLP ist, stellt sich die Frage, ob der in [BH06] vorgestellte Algorithmus zur Generierung von Argumentationsfolgen nicht für die Verwendung in einem argumentationsfähigen Multiagentensytem adaptiert werden kann.

Eine Verwendung des Algorithmus von Besnard und Hunter scheidet jedoch in dem in dieser Arbeit beschriebenen argumentationsfähigen Multiagentensystem aus den folgenden Gründen aus:

- Der Algorithmus von Besnard und Hunter konstruiert aus der vorhandenen Inkonsistenz einer Wissensbasis eine Kompilierung, die als Grundlage zur Generierung von Argument-Gegenargument-Beziehungen und von dialektischen Bäumen dient. Nach Definition ist die Wissensbasis, aus der Argumente generiert werden, d. h. die Vereinigung der globalen und der lokalen Wissensbasis eines Agenten, konsistent. Die Kompilierung einer konsistenten Wissensbasis ist nach Definition 3.8 stets leer.

- Inkonsistenz kann einzig durch die Hinzunahme von unsicheren Regeln anderer Agenten entstehen, also durch die Betrachtung eines Arguments von einem anderen Agenten. Dies geschieht jedoch dynamisch bei der Anfrage nach Gegenargumenten. Eine Integration in eine leere Kompilierung ist deshalb ineffizient, insbesondere auch, weil die Lage der entstehenden Inkonsistenz von vornherein bekannt ist. Eine vollständige Kompilierung wird somit gar nicht benötigt und es ist effizienter, nur die Inkonsistenzen mit dem zu betrachtenden Argument zu ermitteln und aus ihnen Gegenargumente zu generieren.

- Die Verwendung einer Kompilierung auf der Vereinigung der globalen und aller lokalen Wissensbasen wäre wieder effizient, jedoch müssten in diesem Fall die Agenten schon zu Beginn ihre Wissensbasen offenlegen, was der Autonomie der einzelnen Agenten widerspräche und keine realistische Annahme für ein Anwendungsszenario wäre.

Die Agenten eines argumentationsfähigen Multiagentensystems werden also Argumente *online* auf Anfrage generieren. Mit Hilfe eines *backward-chaining*-Verfahrens [CnSG93] werden zu den inkonsistenten Literalen eines Arguments iterativ Gegenargumente aufgebaut. Eine genaue Beschreibung des Algorithmus folgt in Abschnitt 5.3.

Allerdings stellt sich noch die Frage, ob der Algorithmus von Besnard und Hunter als Deliberationsmechanismus in einem Singleagentensystem für DeLP adaptiert werden kann. Dort sind die oben aufgeführten Bedingungen eines Multiagentensystems nicht gegeben, da der Agent nicht dynamisch auf neue Information reagieren muss. Eine weitere Verfolgung dieser Fragestellung würde allerdings den Rahmen dieser Arbeit übersteigen, so dass ihr hier nicht weiter nachgegangen wird.

5 Verteilte logikbasierte Argumentation

In diesem Kapitel wird die in Abschnitt 4.4 definierte Sprache D-DeLP benutzt, um logikbasierte Argumentation in einem Multiagentensystem zu konzipieren.

Dieses Kapitel ist wie folgt gegliedert: Zunächst werden in Abschnitt 5.1 die Anforderungen an das zu konzipierende System aufgestellt und mögliche Probleme aufgezeigt. In Abschnitt 5.2 wird ein argumentationsfähiges Multiagentensystem formal repräsentiert und in den Abschnitten 5.3 bzw. 5.4 folgt eine Erarbeitung der algorithmischen Aspekte bei der Generierung bzw. beim Vergleich von Argumenten. Danach wird mit einer Zusammenfassung in Abschnitt 5.5 dieses Kapitel geschlossen.

5.1 Leistungsanforderungen und Problemstellungen

Ein argumentationsfähiges Multiagentensystem soll eine Anfrage in Form einer logischen Aussage entgegennehmen, sie intern mit Hilfe eines logikbasierten Argumentationsprozesses analysieren und eine Aussage über Akzeptanz oder Widerlegung zurückgeben können.

Um eine Menge von Agenten in einer geregelten Weise miteinander agieren zu lassen, ist ein Koordinationsmechanismus nötig. Nach [NLJ96] existieren grundsätzlich vier verschiedene Möglichkeiten, Koordination in einem Multiagentensystem zu realisieren:

- *organisational structuring*: Die Koordination wird durch eine Kontrollinstanz realisiert, die Aufgaben verteilt und die Ergebnisse von Sub-Aufgaben vereinigt.
- *contracting*: Agenten übernehmen die Rollen von *Auftraggebern* und *Auftragnehmern*, teilen ihnen zugewiesene Aufgaben auf und geben Subaufgaben an geeignete Agenten weiter.
- *multi-agent planning*: Vor der eigentlichen Aktionsphase planen die Agenten eines Systems im Vorfeld zusammen alle Aktionen, um möglichst effizient ihre jeweiligen Ziele zu erreichen, ohne sich gegenseitig zu behindern.
- *negotiation*: Agenten verhandeln miteinander über Aktionen und Austausch von Information.

Für weitere Informationen zu diesen Mechanismen siehe [NLJ96]. Da Agenten vor der Generierung von Argumenten nicht nach Gegenargumenten befragt werden können, scheidet die Verwendung von *multi-agent planning* zunächst aus. Auch die Verwendung von

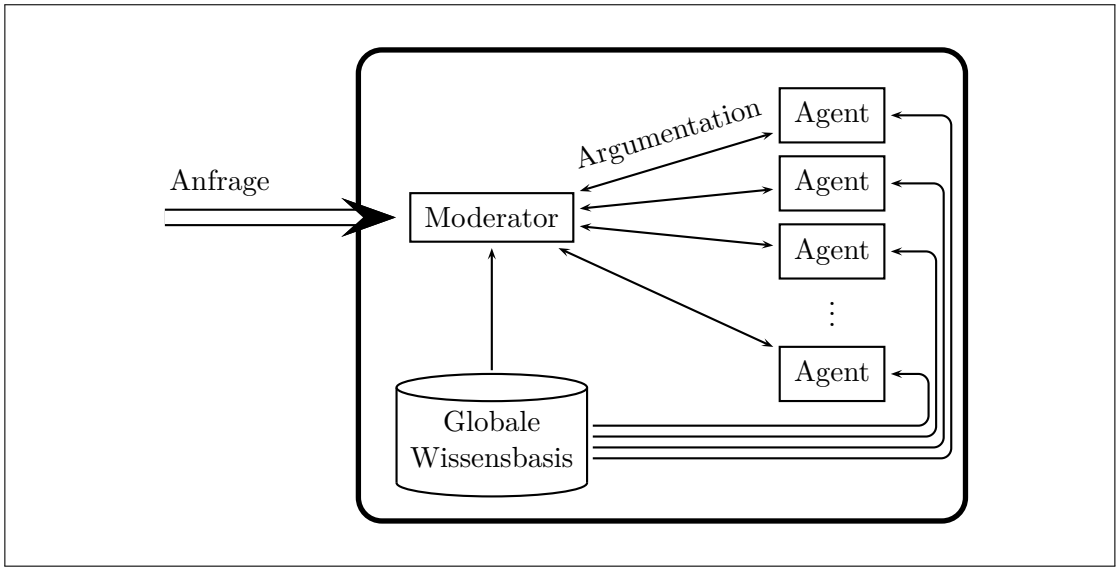

Abbildung 5.1: Schema eines argumentationsfähigen Multiagentensystems

negotiation scheint nicht angebracht, weil im Rahmen dieser Diplomarbeit die individuellen Ziele der Agenten in dem Argumentationsprozess nicht auf den Koordinationsprozess übergreifen sollen[1]. *Contracting* wäre eine geeignete Möglichkeit, logikbasierte Argumentation zu realisieren. Agenten würden die von ihnen generierten Argumente an andere Agenten weitergeben, die daraufhin Gegenargumente erzeugen. Im Hinblick auf die Verwendung eines argumentationsfähigen Multiagentensystems im Rechtswesen wird allerdings ein strukturierterer Koordinationsmechanismus wie *organisational structuring* nahegelegt. Aus diesem Grund und um die Kommunikation nach außen von den Interna des Systems zu trennen, wird ein besonderer Agent als zentrale Kontrollinstanz gewählt und als Moderator bezeichnet. Dieser Moderator ist die Schnittstelle des Systems mit der Außenwelt und fungiert als Ansprechpartner für Anfragen an das System. Weiterhin koordiniert er den Argumentationsprozess zwischen den Agenten des Systems und ist für die anschließende Analyse der erhaltenen Argumentationsstrukturen zuständig. Abbildung 5.1 zeigt ein Schema der Nachrichtenflüsse in einem argumentationsfähigen Multiagentensystem und die besondere Bedeutung des Moderators.

Ein Moderator eines argumentationsfähigen Systems muss somit

1. kommunikationsfähig sein, d. h. sowohl äußere Anfragen empfangen und beantworten können, als auch Anfragen an Agenten des Multiagentensystems schicken und ihre Antworten empfangen können,

[1] Das Ziel des Systems sei also identisch mit den (objektiven) Zielen aller Agenten, nämlich die Entscheidungsfindung über Akzeptanz oder Widerlegung der Anfrage. Es wäre allerdings eine interessante Fragestellung, argumentationsfähige Multiagentensysteme ohne diese Einschränkung zu untersuchen.

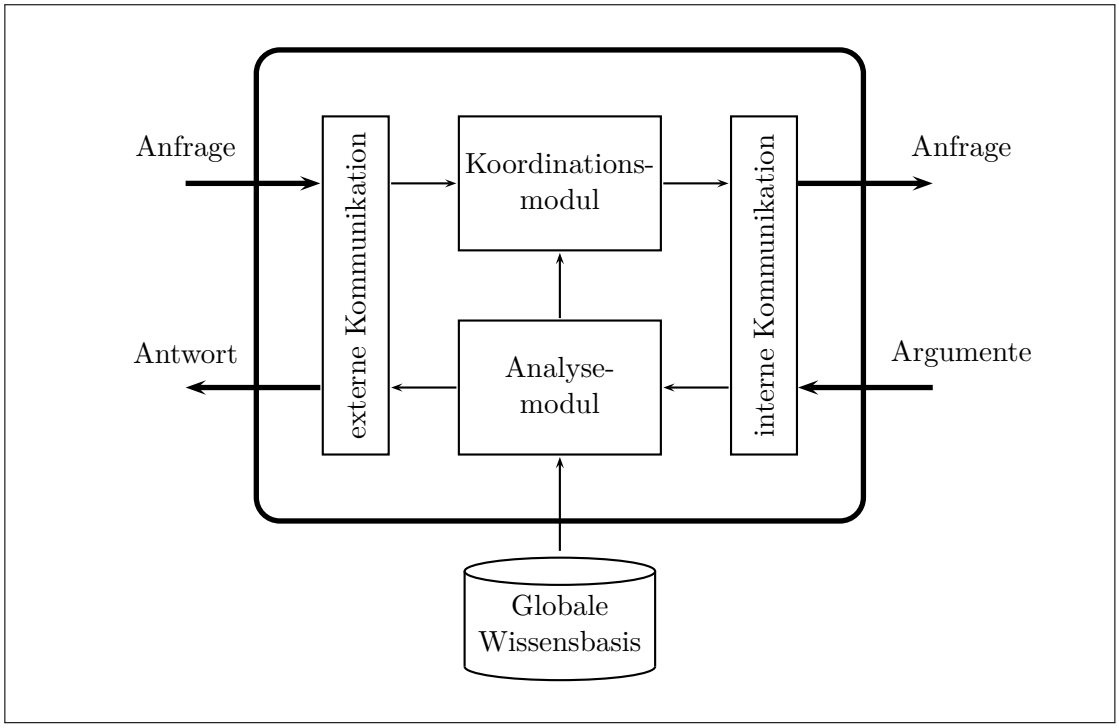

Abbildung 5.2: Schema eines Moderators

2. koordinationsfähig sein, d. h. die Agenten des Multiagentensystems systematisch und koordiniert nach Gegenargumenten befragen können,

3. analysefähig sein, d. h. die erhaltenen dialektischen Bäume auf Begründung der Wurzelliterale hin überprüfen können.

In Abbildung 5.2 ist ein Moderator mit seinen Komponenten schematisch dargestellt.

Den eigentlichen Argumentationsprozess in einem argumentationsfähigen Multiagentensystem leisten jedoch die Agenten. Sie generieren auf Grundlage der globalen und ihrer jeweiligen lokalen Wissensbasis Argumente und reagieren auf Argumente anderer Agenten mit Gegenargumenten. Ein Agent eines argumentationsfähigen Multiagentensystems soll also

1. kommunikationsfähig sein, d. h. Anfragen vom Moderator erhalten können und Argumente an den Moderator schicken können,

2. inferenzfähig sein, d. h. er soll die unsicheren Regeln seiner lokalen Wissensbasis nutzen können, um auf zusätzliches Wissen zu schließen,

3. argumentationsfähig sein, d. h. auf Anfragen des Moderators in Form von Argumentationsfolgen passende Gegenargumente generieren können.

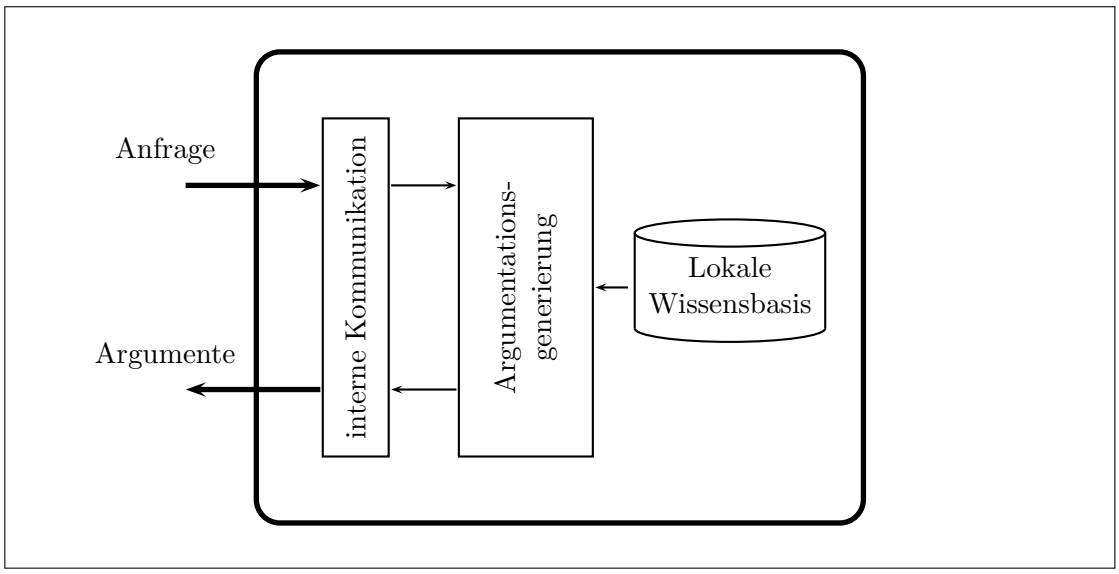

Abbildung 5.3: Schema eines argumentationsfähigen Agenten

In Abbildung 5.3 ist ein argumentationsfähiger Agent mit seinen Komponenten schematisch dargestellt. Dabei ist zu beachten, dass neben der globalen Wissensbasis, auf die jeder Agent des Systems Zugriff hat, jeder Agent auch noch über eine lokale Wissensbasis verfügt, die individuelles Wissen beinhaltet. Da Inkonsistenz nur durch Betrachtung von verschiedenen lokalen Wissensbasen entstehen kann, ist dies für ein argumentationsfähiges Multiagentensystem nötig, damit es überhaupt Argumente und Gegenargumente hervorbringen kann.

Der eigentliche Argumentationsprozess läuft dann folgendermaßen ab:

1. Der Moderator des Systems erhält von außen eine Anfrage in Form eines Literals.
2. Der Moderator befragt jeden Agenten nach initialen Argumenten für oder gegen die Anfrage.
3. Jeder Agent wird zu jedem Argument nach Gegenargumenten befragt. Dieser Prozess wird solange rekursiv wiederholt, bis kein Agent mehr Argumente hervorbringen kann.
4. Der Moderator wertet die entstandenen Argumentationsfolgen aus und gibt die passende Antwort nach außen zurück.

Dieser Ablauf eines Argumentationsprozesses in einem argumentationsfähigen Multiagentensystem wird in den nächsten Abschnitten genauer formalisiert und beschrieben.

5.2 Formale Darstellung eines argumentationsfähigen Multiagentensystems

In diesem Abschnitt werden die formalen Grundlagen zur Beschreibung eines argumentationsfähigen Multiagentensystems gelegt. In Unterabschnitt 5.2.1 werden dazu die Komponenten Moderator und Agenten auf der Grundlage von D-DeLP-Wissensbasen spezifiziert. In Unterabschnitt 5.2.2 wird daraufhin ein argumentationsfähiges Multiagentensystem und schließlich in Unterabschnitt 5.2.3 der Argumentationsprozess formalisiert und das Antwortverhalten des Systems definiert.

5.2.1 Moderator und Agenten

Neben den Strukturen zur Wissensrepräsentation verfügen die Komponenten eines argumentationsfähigen Multiagentensystems über Funktionen zur Realisierung von z. B. Argumentationsgenerierung und Analyse. Der Moderator eines argumentationsfähigen Multiagentensystems muss in der Lage sein, Argumentationsfolgen auf Akzeptanz hin zu prüfen und dialektische Bäume auszuwerten. Aus diesem Grund werden nun die dazu nötigen Funktionen formal spezifiziert. Wie die Funktionen realisiert werden, ist für diese Darstellung zunächst nicht von Belang. Es sind somit auch andere Realisierungen als die in den Beispielen dieses Abschnitts gegebenen denkbar.

Notation. Definiere folgende Abkürzungen:

- $\mathcal{V}$ bezeichne die Menge aller dem System bekannten Literale.
- $\mathcal{R}$ bezeichne die Menge aller unsicheren Regeln, die aus $\mathcal{V}$ gebildet werden können.
- Ω bezeichne die Menge aller Argumente, die mit Regeln aus $\mathcal{R}$ und Konklusionen aus $\mathcal{V}$ gebildet werden können.
- Σ bezeichne die Menge aller endlichen Folgen von Argumenten aus Ω.
- Υ bezeichne die Menge aller dialektischen Bäume nach Definition 4.14 mit Argumenten aus Ω.

Es folgen zunächst formale Beschreibungen der Komponenten des Moderators eines argumentationsfähigen Multiagentensystems.

Definition 5.1 (Analysefunktion)**.** Eine *Analysefunktion* χ ist eine Funktion

$$\chi : \Upsilon \to \{0, 1\},$$

so dass für einen markierten dialektischen Baum $\upsilon \in \Upsilon$ gilt:

$$\chi(\upsilon) = 0 \quad \Leftrightarrow \quad \text{Das Wurzelargument von } \upsilon \text{ ist widerlegt.}$$

Die Definition einer Analysefunktion ist unabhängig von der Definition markierter dialektischer Bäume.

Beispiel 5.1. Der Moderator eines argumentationsfähigen Multiagentensystems benutze die nach Definition 2.19 definierte Auswertung eines dialektischen Baumes. Sei $\upsilon \in \Upsilon$ ein dialektischer Baum und υ^* der nach Definition 2.19 zu υ gehörende markierte dialektische Baum. Dann ist die Funktion χ definiert durch

$$\chi(\upsilon) =_{def} \begin{cases} 0 & \text{, falls die Wurzel von } \upsilon^* \text{ mit „D“ markiert ist} \\ 1 & \text{, sonst} \end{cases}.$$

Eine Akzeptanzfunktion prüft eine gegebene Folge von Argumenten auf Akzeptanz.

Definition 5.2 (Akzeptanzfunktion)**.** Eine *Akzeptanzfunktion* η ist eine Funktion

$$\eta : \Sigma \to \{0, 1\},$$

so dass für eine Argumentationsfolge $\Lambda \in \Sigma$ gilt:

$$\eta(\Lambda) = 0 \quad \Leftrightarrow \quad \Lambda \text{ wird nicht akzeptiert.}$$

Beispiel 5.2. Der Moderator eines argumentationsfähigen Multiagentensystems benutze die durch D-DeLP gegebenen Kriterien für akzeptierte Argumentationsfolgen. Dann ist $\eta(\Lambda) = 1$, g. d. w. die Argumentationsfolge $\Lambda \in \Sigma$ eine akzeptierte Argumentationsfolge nach Definition 4.13 ist.

Definition 5.3 (Entscheidungsfunktion)**.** Eine *Entscheidungsfunktion* μ ist eine Funktion

$$\mu : \mathfrak{P}(\Upsilon) \to \{\text{YES,NO,UNDECIDED,UNKNOWN}\}.$$

Eine Entscheidungsfunktion μ bildet eine Menge von dialektischen Bäumen auf die möglichen Antworten eines argumentationsfähigen Multiagentensystems ab.

Beispiel 5.3. Der Moderator eines argumentationsfähigen Multiagentensystems fälle auf der Grundlage von Definition 2.21 die Antwort auf eine Anfrage p, die Funktion χ sei also wie in Beispiel 5.1 definiert. Sei $Q \subseteq \Upsilon$, die Funktion

$$\mu : \mathfrak{P}(\Upsilon) \to \{\text{YES,NO,UNDECIDED,UNKNOWN}\}$$

ist dann definiert durch

1. $\mu(Q) = \text{YES}$, falls ein dialektischer Baum $\upsilon \in Q$ existiert mit:
   a) die Wurzel von υ ist ein Argument für p und
   b) $\chi(\upsilon) = 1$.

2. $\mu(Q) = \text{NO}$, falls ein dialektischer Baum $\upsilon \in Q$ existiert mit:
   a) die Wurzel von υ ist ein Argument für $\overline{p}$ und
   b) $\chi(\upsilon) = 1$.
3. $\mu(Q) = \text{UNDECIDED}$, wenn $\chi(\upsilon) = 0$ für alle $\upsilon \in Q$.
4. $\mu(Q) = \text{UNKNOWN}$, wenn p nicht in der betrachteten Sprache liegt ($p \notin \mathcal{V}$).

Definition 5.4 (Moderator)**.** Ein *Moderator* ist ein Tupel (μ, χ, η) mit einer Entscheidungsfunktion μ, einer Analysefunktion χ und einer Akzeptanzfunktion η.

Bei der Definition der Funktionen eines Agenten eines argumentationsfähigen Multiagentensystems ist die Realisierung nicht immer so eindeutig wie bei den Funktionen eines Moderators. Eine Beispielrealisierung dieser Funktionen anhand der Definitionen von D-DeLP folgt in Abschnitt 5.3.

Ein Agent eines argumentationsfähigen Multiagentensystems muss zwei Funktionen unterstützen: Zum einen soll er auf Anfragen in Form von Literalen Argumente für und gegen diese Anfragen generieren und zum anderen soll er auf Argumente anderer Agenten mit Gegenargumenten reagieren.

Definition 5.5 (Wurzelargumentsfunktion)**.** Eine *Wurzelargumentsfunktion* φ ist eine Funktion

$$\varphi : \mathcal{V} \to \mathfrak{P}(\Omega),$$

so dass für jedes Literal $h \in \mathcal{V}$ die Menge $\varphi(h)$ eine Menge von Argumenten für h oder für $\overline{h}$ ist.

Für die nächste Definition gelte die folgende Notation:

Notation. Sei $\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$ eine Argumentationsfolge und $\langle \mathcal{A}_{n+1}, h_{n+1} \rangle$ ein Argument. Die Konkatenation

$$[\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle, \langle \mathcal{A}_{n+1}, h_{n+1} \rangle]$$

von Λ und $\langle \mathcal{A}_{n+1}, h_{n+1} \rangle$ werde mit $\Lambda + \langle \mathcal{A}_{n+1}, h_{n+1} \rangle$ bezeichnet.

Um Autonomie der Agenten in Bezug auf die Konstruktion von Gegenargumenten zu gewährleisten, verfügt jeder Agent auch über eine Akzeptanzfunktion η, um seine Gegenargumente auf Akzeptanz innerhalb der Argumentationsfolge hin zu überprüfen. Auf Grundlage von η ist dann die Gegenargumentsfunktion ψ definiert:

Definition 5.6 (Gegenargumentsfunktion)**.** Sei η eine Akzeptanzfunktion. Eine *Gegenargumentsfunktion* ψ ist eine Funktion

$$\psi : \Sigma \to \mathfrak{P}(\Omega),$$

so dass für jede Argumentationsfolge $\Lambda \in \Sigma$ die Menge $\psi(\Lambda)$ eine Menge von Angriffen auf Λ aus Ω ist und für jedes $\langle \mathcal{B}, h \rangle \in \psi(\Lambda)$ gilt: $\eta(\Lambda + \langle \mathcal{B}, h \rangle) = 1$.

Ein Agent eines argumentationsfähigen Multiagentensystems ist dann gegeben durch:

Definition 5.7 (Argumentationsfähiger Agent)**.** Sei Π eine globale Wissensbasis. Ein *argumentationsfähiger Agent* bezüglich Π ist ein Tupel $(\Delta, \varphi, \psi, \eta)$ mit einer bezüglich Π lokalen Wissensbasis Δ, einer Wurzelargumentsfunktion φ, einer Gegenargumentsfunktion ψ und einer Akzeptanzfunktion η.

5.2.2 Argumentationsfähige Multiagentensysteme

Ein argumentationsfähiges Multiagentensystem wird gebildet aus einem Moderator, einer globalen Wissensbasis und einer Menge von argumentationsfähigen Agenten:

Definition 5.8 (Argumentationsfähiges Multiagentensystem)**.** Ein *argumentationsfähiges Multiagentensystem* ist ein Tupel $(M, \Pi, \{A_1, \ldots, A_n\})$ mit einem Moderator M, einer globalen Wissensbasis Π nach Definition 4.5 und bezüglich Π argumentationsfähigen Agenten $A_1, \ldots, A_n$.

5.2.3 Argumentationsprozess und Antwortverhalten

Um die Antwort eines argumentationsfähigen Multiagentensystems auf eine Anfrage $h \in \mathcal{V}$ zu bestimmen, folgt zunächst eine funktionale Beschreibung des Argumentationsprozesses:

Definition 5.9 (Argumentationsprodukt)**.** Sei $h \in \mathcal{V}$ eine Anfrage und T ein argumentationsfähiges Multiagentensystem mit

$$\begin{aligned} T &= (M, \Pi, \{A_1, \ldots, A_n\}), \\ M &= (\mu, \chi, \eta), \\ A_i &= (\Delta_i, \varphi_i, \psi_i, \eta_i) \quad (1 \leq i \leq n). \end{aligned}$$

Ein dialektischer Baum υ heißt *Argumentationsprodukt* von T bezüglich h, g. d. w. die folgenden beiden Bedingungen gelten:

1. Es existiert ein j mit $1 \leq j \leq n$, so dass die Wurzel von υ ein Element aus $\varphi_j(h)$ ist.

2. Für jeden Pfad $\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$ aus υ gilt für die Menge K aller Kindsknoten von $\langle \mathcal{A}_n, h_n \rangle$

$$K = \{\langle \mathcal{B}, h' \rangle | \langle \mathcal{B}, h' \rangle \in \psi_1(\Lambda) \cup \cdots \cup \psi_n(\Lambda) \wedge \eta(\Lambda + \langle \mathcal{B}, h' \rangle) = 1\}.$$

   Die Menge K ist also die Menge aller akzeptierten Angriffe auf Λ.

Das Antwortverhalten eines argumentationsfähigen Multiagentensystems wird dann auf der Grundlage der Argumentationsprodukte von der Entscheidungsfunktion des Moderators bestimmt:

Definition 5.10 (Antwortverhalten)**.** Sei $h \in \mathcal{V}$ eine Anfrage, $T = (M, \Delta, \{A_1, \ldots, A_n\})$ ein argumentationsfähiges Multiagentensystem mit $M = (\mu, \chi, \eta)$ und $\{\upsilon_1, \ldots, \upsilon_n\}$ die Menge aller Argumentationsprodukte von T bezüglich h. Die Antwort $A(T, h)$ von T auf h ist dann

$$A(T, h) = \mu(\{\upsilon_1, \ldots, \upsilon_n\}).$$

Die möglichen Antworten eines argumentationsfähigen Multiagentensystems sind somit YES, NO, UNDECIDED und UNKNOWN. Zusammenfassend ist dann die Interpretation dieser Antworten gegeben durch:

Definition 5.11 (Semantik des Antwortverhaltens)**.** Sei T ein argumentationsfähiges Multiagentensystem und $h \in \mathcal{V}$ eine Anfrage an dieses System. Die Semantik von $A(T, h)$ ist definiert durch:

1. $A(T, h) =$ YES: Die Anfrage h wird von T geglaubt, da nicht-widerlegbare Argumente für h vorliegen.
2. $A(T, h) =$ NO: Die Anfrage h wird von T abgelehnt, da nicht-widerlegbare Argumente für $\overline{h}$ vorliegen.
3. $A(T, h) =$ UNDECIDED: Die Anfrage h kann von T nicht entschieden werden, da alle Argumente für und gegen h widerlegt werden können.
4. $A(T, h) =$ UNKNOWN: Die Anfrage h kann von T nicht beantwortet werden, da kein Wissen über h vorliegt.

5.3 Berechnung von Argumenten und Gegenargumenten

In diesem Abschnitt wird auf die algorithmischen Aspekte eines argumentationsfähigen Multiagentensystems eingangen und es werden die nötigen Algorithmen auf der Basis von D-DeLP formalisiert.

Die Steuerung des Argumentationsprozesses regelt der Moderator des argumentationsfähigen Multiagentensystems. Er befragt zunächst jeden Agenten nach Argumenten für oder gegen die Anfrage. Danach wird jeder Agent nach Gegenargumenten zu den erzeugten Argumenten befragt und dieser Prozess solange rekursiv wiederholt bis kein Agent weitere Gegenargumente zu irgendeinem Argument erzeugen kann. Eine anschließende Analyse der entstandenen dialektischen Bäume durch den Moderator legt dann die Antwort des Systems auf die Anfrage fest.

Sei $h \in \mathcal{V}$ eine Anfrage an ein argumentationsfähiges Multiagentensystem. Der Ablauf der Argumentationsgenerierung gestaltet sich demnach folgendermaßen:

- 1. Schritt: Der Moderator befragt jeden Agenten nach Argumenten für oder gegen die Anfrage. Zu jedem dieser Argumente entsteht ein eigener dialektischer Baum.
- 2. Schritt: Ist $(\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle)$ eine Argumentationsfolge eines dialektischen Baums, so wird jeder Agent nach Gegenargumenten zu $\langle \mathcal{A}_n, h_n \rangle$ befragt.
- 3. Schritt: Kann kein Agent mehr Argumente hervorbringen, so werden die dialektischen Bäume vom Moderator analysiert und die Anfrage wird entsprechend beantwortet.

In diesem Abschnitt werden die Algorithmen zur Generierung von Argumenten und Gegenargumenten in einem argumentationsfähigen Multiagentensystem beschrieben und formalisiert, also mögliche Realisierungen der Wurzelarguments-, Gegenarguments- und Akzeptanzfunktionen aus Unterabschnitt 5.2.1.

5.3.1 Generierung von Wurzelargumenten

Der erste Schritt in einem Argumentationsprozess ist die Generierung von *Wurzelargumenten*, also Argumenten, die an der Wurzel eines dialektischem Baumes stehen und sich direkt auf die Anfrage beziehen. Ist $h \in \mathcal{V}$ eine Anfrage an ein argumentationsfähiges Multiagentensystem, so befragt der Moderator jeden Agenten nach Argumenten zu h und zu $\overline{h}$. Um eine eindeutige Antwort auf eine Anfrage zu liefern, müssen dialektische Bäume für h und für $\overline{h}$ berechnet werden, da die Nicht-Existenz von nicht-widerlegbaren Argumenten für h nicht die Antwort NO impliziert. Dazu muss die Existenz mindestens eines nicht-widerlegten Arguments für die Gegenthese $\overline{h}$ gezeigt werden, siehe Definition 2.21.

Allgemeiner Ablauf

Sei die Anfrage $h \in \mathcal{V}$ an ein argumentationsfähiges Multiagentensystem gegeben. Jeder Agent muss nun Wurzelargumente für h und für $\overline{h}$ generieren. Es reicht jedoch zunächst aus, nur die Generierung von Argumenten für das Literal h zu betrachten, der zweite Fall verläuft analog.

Besitzt h eine strikte Ableitung aus den sicheren Regeln und den Fakten der globalen Wissensbasis, so besitzt h nur ein einziges Argument $\langle \emptyset, h \rangle$ [GS02]. Um diesen Spezialfall zu behandeln, wird zunächst geprüft, ob h eine sichere Ableitung aus der globalen Wissensbasis besitzt und ggf. das zugehörige leere Argument zurückgeliefert. Ist dies nicht der Fall, so kann versucht werden, aus den unsicheren Regeln des Agenten Argumente für h zu konstruieren. Der hier beschriebene Ansatz folgt der Idee des *backward-chaining* aus [CnSG93].

Der Ausgangspunkt des *backward-chaining* ist zunächst die Existenz einer (sicheren oder unsicheren) Regel, die auf h schließt. Diese Regel muss sich in der Vereinigung der sicheren Regeln der globalen Wissenbasis und der unsicheren Regeln der lokalen Wissensbasis des betrachteten Agenten befinden. Eventuell vorhandene Metainformationen zu Regeln können zunächst vernachlässigt werden. Jede Regel mit dem Kopfliteral h liefert einen Ausgangspunkt für die Konstruktion eines Arguments für h. Sei $h \dashleftarrow b_1, \ldots, b_n$ eine (sichere oder unsichere) Regel mit Kopfliteral h. Für jedes Rumpfliteral $b \in \{b_1, \ldots, b_n\}$ dieser Regel wird nun geprüft, ob es sich entweder um ein Faktum der globalen Wissensbasis handelt oder es eine Regel gibt, die auf b schließt. Im zweiten Fall wird rekursiv fortgefahren und überprüft, ob die Rumpfliterale abgeleitet werden können. Gibt es für ein Rumpfliteral $b \in \{b_1, \ldots, b_n\}$ mehrere Regeln, die auf b schließen, so teilt sich die Argumentationsgenerierung für jede vorhandene Regel in einen neuen Zweig auf. Da man bei der Auswahl der Regeln verschiedene Möglichkeiten hat, führt dies zu verschiedenen Argumenten, die alle separat voneinander betrachtet werden.

Existiert ein Rumpfliteral $b \in \{b_1, \ldots, b_n\}$, das kein Fakt ist und für das auch keine Regel mit Kopfliteral b existiert, so schlägt diese Argumentationsgenerierung fehl. Der betrachtete Zweig der Argumentationsgenerierung muss abgebrochen werden, da das Argument nicht vervollständigt werden kann.

Sind alle Rumpfliterale einer Regel ableitbar, so ist die Regel anwendbar und das Kopfliteral somit ableitbar. Gilt dies für die erste Regel einer Argumentationsgenerierung, so sind entsprechend der ableitbaren Verzweigungen in der Generierung entsprechend viele Wurzelargumente erzeugt worden. Die Vereinigung aller erzeugten Wurzelargumente wird dann zurückgeliefert.

Formale Beschreibung des Algorithmus

Wie oben erläutert, muss ein Algorithmus zur Generierung von Wurzelargumenten Argumente für und gegen die Anfrage erzeugen. Der Hauptalgorithmus eines Agenten zur Generierung von Wurzelargumenten ist `RootArguments` mit den Parametern:

- *query*: Die Anfrage an das argumentationsfähige Multiagentensystem, für die und gegen die der Agent Argumente generieren soll.
- Δ: Die lokale Wissensbasis, also die Menge der unsicheren Regeln des Agenten.
- Π: Die globale Wissensbasis, also die Menge der sicheren Regeln und Fakten.

Der Algorithmus `RootArguments` ruft den Algorithmus `Arguments` einmal für *query* selbst und einmal für die Negation von *query* auf. Der Algorithmus `Arguments` enthält den eigentlichen Algorithmus zur Generierung von Argumenten und liefert alle Argumente des Agenten für das angegebene Literal, die in `RootArguments` vereinigt und an die aufrufende Instanz zurückgeliefert werden.

Es folgt eine formale Beschreibung des Algorithmus `RootArguments` in Pseudocode:

```
1  RootArguments(query, Δ, Π)
2    queryArguments = Arguments(query, Δ, Π);
3    nqueryArguments = Arguments(¬query, Δ, Π);
4    return queryArguments ∪ nqueryArguments;
```

Algorithmus 5.1: `RootArguments`

Im Algorithmus `Arguments` wird zunächst überprüft, ob der erste Parameter *conclusion* ein Fakt der globalen Wissensbasis ist und im positiven Fall das leere Argument als einzig zulässige Lösung zurückgeliefert. Ist *conclusion* kein Fakt, werden mögliche Argumente iterativ berechnet. Da es grundsätzlich viele verschiedene Ableitungen geben könnte, werden zunächst alle Regeln gesucht, die *conclusion* als Kopfliteral besitzen. Jede dieser Regeln bietet dann einen Ausgangspunkt für die Konstruktion eines Arguments.

Sei nun $r : h \leftarrow b_1, \ldots, b_n$ eine solche Regel. Ein Argument für h kann nun aus der Regel r und Regeln für $b_1, \ldots, b_n$ gebildet werden. Da jedoch jedes b_i $(1 \leq i \leq n)$ mehrere Ableitungen besitzen kann, müssen alle Kombinationsmöglichkeiten berücksichtigt werden. Dazu enthält der Stack S zu jeder Zeit Elemente der Form (R, L), die zu Argumenten ausgebaut werden können. Dabei ist R die Menge der bisher angewandten unsicheren Regeln und L ein Stack von Literalen, die noch abgeleitet werden müssen. Initialisiert wird S mit Paaren (R, L), so dass R genau eine Regel mit Kopfliteral *conclusion* enthält und L die Menge der Rumpfliterale dieser Regel ist. In jedem Iterationsschritt der nachfolgenden `while`-Schleife wird S ein Element entnommen und getestet, ob die Menge der noch abzuleitenden Literale leer ist. Ist das der Fall, so ist R der Support eines Arguments für *conclusion* und $\langle R, conclusion \rangle$ wird der Ergebnismenge hinzugefügt. Ist der Stack L nicht leer, so wird L ein Element l entnommen und die Menge der Regeln mit Kopfliteral l bestimmt. Jede dieser Regeln läßt die Argumentationsgenerierung verzweigen und fügt ein neues Element auf den Stack S hinzu. Dazu wird die Regel – falls es sich um eine unsichere Regel handelt – der aktuellen Menge R von schon genutzten unsicheren Regeln hinzugefügt. Der Stack L der noch abzuleitenden Literale wird um alle Rumpfliterale dieser Regel erweitert, die nicht schon durch die vorhandenen Regeln abgeleitet werden können. Da auf diese Weise auch nicht-minimale Argumente erzeugt werden können, werden am Ende nicht-minimale Argumente aus der Ergebnismenge entfernt.

Es folgt eine formale Beschreibung des Algorithmus `Arguments` in Pseudocode:

```
1  Arguments(conclusion, Δ, Π)
2    if conclusion is a fact in Π then
3      return {⟨∅, conclusion⟩}
4    S = ∅
5    arguments = ∅
6    for each rule r : conclusion <-- b_1, ..., b_n ∈ Δ ∪ Π do
7      if r is a defeasible rule then
8        Push ({r}, {b_1, ..., b_n}) on S
9      else
10       Push ({}, {b_1, ..., b_n}) on S
11   while S not empty do
12     Pop (R, L) from S
13     if L is empty then
14       arguments = arguments ∪ {⟨R, conclusion⟩}
15     else
16       Pop l from L
17       if l is a fact in Π then
18         Push (R, L) on S
19       else
20         for each rule r : l <-- b_1, ..., b_n ∈ Δ ∪ Π do
21           if r is a defeasible rule then
22             R' = R ∪ {r}
23           L' = L
24           for each b_i with 1 ≤ i ≤ n do
25             if b_i is not the head of a rule in R' then
26               L' = L' ∪ {b_i}
27           Push (R', L') on S
28   for each a ∈ arguments do
29     if there exists a' ∈ arguments with a ≠ a'
30                 and a is a subargument of a'
31       arguments = arguments \ {a'}
32   return arguments
```

Algorithmus 5.2: `Arguments`

Auf der Grundlage des Algorithmus `RootArguments` kann nun eine Wurzelargumentsfunktion φ_{D-DeLP} definiert werden.

Definition 5.12 (φ_{D-DeLP})**.** Sei Π eine globale, Δ eine lokale Wissensbasis zu Π und h ein Literal. Die Funktion $\varphi_{D-DeLP} : \mathcal{V} \to \mathfrak{P}(\Omega)$ ist dann definiert durch

$$\varphi_{D-DeLP}(h) =_{def} \texttt{RootArguments}(h, \Delta, \Pi).$$

Vollständigkeit und Korrektheit

Es bleibt zu zeigen, dass der Algorithmus `Arguments` vollständig und korrekt ist. Die Vollständigkeit und Korrektheit des Algorithmus `RootArguments` folgt dann direkt.

Theorem 5.1. *Sei $h \in \mathcal{V}$ ein Literal, Δ die lokale Wissensbasis eines Agenten und Π die globale Wissensbasis des Multiagentensystems. Der Aufruf*

$$N = \texttt{Arguments}(h, \Delta, \Pi)$$

*liefert dann eine Menge von Argumenten N für h zurück, d. h. für jedes $\langle \mathcal{A}, c \rangle \in N$ gilt $c = h$ und $\langle \mathcal{A}, c \rangle$ ist ein Argument nach Definition 4.9 (*Korrektheit des Algorithmus*).*

Beweis. Sei $\langle \mathcal{A}, c \rangle \in N$. Dann ist zunächst $c = h$, da in Zeile 14 von Algorithmus `Arguments` der Menge *arguments* nur Argumente mit Konklusion h hinzugefügt werden. Zeige nun, dass $\langle \mathcal{A}, c \rangle$ ein Argument nach Definition 4.9 ist:

1. h besitzt eine unsichere Ableitung aus $\Pi \cup \mathcal{A}$ nach Konstruktion.
2. Die Menge $\Pi \cup \mathcal{A}$ ist nicht widersprüchlich, da nach Definition $\Pi \cup \Delta$ nicht widersprüchlich ist und $\mathcal{A} \subseteq \Delta$ gilt.
3. $\mathcal{A}$ ist minimal bezüglich Mengeninklusion, da in den Zeilen 28–31 des Algorithmus `Arguments` nicht-minimale Argumentskandidaten, die die Eigenschaften 1. und 2. erfüllen, aus der Menge *arguments* entfernt werden.

□

Theorem 5.2. *Sei $h \in \mathcal{V}$ ein Literal, Δ die lokale Wissensbasis eines Agenten und Π die globale Wissensbasis des Multiagentensystems. Dann ist*

$$N = \texttt{Arguments}(h, \Delta, \Pi)$$

*die vollständige Menge der Argumente für h, d. h. es existiert kein Argument $\langle \mathcal{A}, h \rangle$ mit $\mathcal{A} \subseteq \Delta$ und $\langle \mathcal{A}, h \rangle \notin N$. (*Vollständigkeit des Algorithmus*)*

Beweis. Ist h ein Fakt, so wird das einzig zulässige Argument $\langle \emptyset, h \rangle$ in Zeile 3 von Algorithmus `Arguments` berechnet und zurückgeliefert. Ist h kein Fakt so sei F die Menge der Fakten aus Π und es sei angenommen es existiert ein Argument $\langle \mathcal{A}, h \rangle$ mit $\mathcal{A} \subseteq \Delta$ und $\langle \mathcal{A}, h \rangle \notin N$. Dann existiert eine minimale Menge $K \subseteq \Pi$ von sicheren Regeln, so dass $\mathcal{A} \cup K \cup F \mathrel{|\!\sim} h$. Daraus folgt zunächst die Existenz einer Regel $r : h \leftarrow b_1, \ldots, b_n$ mit $r \in \mathcal{A} \cup K$. Dann gilt, dass jedes b_i $(1 \leq i \leq n)$ entweder ein Fakt aus F oder der Kopf einer Regel $r' \in \mathcal{A} \cup K$ ist. Im Algorithmus `Arguments` wird die Regel r zunächst in Zeile 6 gefunden und ein entsprechend initialisiertes Element (R, L) auf den Stack S gelegt. Induktiv folgt, dass jede weitere Regel $r' \in \mathcal{A} \cup K$ bei der Bearbeitung eines Zweiges des Elements (R, L) gefunden wird und – da $\mathcal{A} \cup K \cup F \mathrel{|\!\sim} h$ – schließlich $(\mathcal{A}, \emptyset)$ auf dem Stack S liegt. $\langle \mathcal{A}, h \rangle$ wird der Ergebnismenge hinzugefügt und auch nicht wieder daraus entfernt, da $\langle \mathcal{A}, h \rangle$ nach Voraussetzung ein Argument und $\mathcal{A}$ somit minimal ist. Also gilt $\langle \mathcal{A}, h \rangle \in N$ im Widerspruch zur Annahme und der Algorithmus `Arguments` ist vollständig. □

5.3.2 Generierung von Gegenargumenten

Gegeben eine Argumentationsfolge $\Lambda = (\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle)$ ist es eine weitere Aufgabe eines Agenten in einem argumentationsfähigen Multiagentensystem, mögliche Gegenargumente $\langle \mathcal{B}, b \rangle$ zu $\langle \mathcal{A}_n, h_n \rangle$ generieren zu können, die nach Definition 4.13 akzeptierte Argumentationsfolgen $\Lambda' = \Lambda + \langle \mathcal{B}, b \rangle$ erzeugen. Als Grundlage für die Erzeugung von Gegenargumenten dient dabei die lokale Wissensbasis.
Ein zentraler Begriff zur Beschreibung von Gegenargumenten ist die *Menge der Angriffspunkte* eines Arguments:

Definition 5.13 (Menge der Angriffspunkte)**.** Sei Π eine globale, Δ eine lokale Wissensbasis zu Π und $\langle \mathcal{A}, h \rangle$ ein Argument. Ist $\mathcal{F}$ die Menge aller Literale, die eine unsichere Ableitung aus $\Pi \cup \Delta$ haben, so ist die *Menge der Angriffspunkte* $AP(\langle \mathcal{A}, h \rangle)$ von $\langle \mathcal{A}, h \rangle$ definiert durch

$$AP(\langle \mathcal{A}, h \rangle) = \{ f \in \mathcal{F} | \Pi \cup \mathcal{A} \cup \{f\} \mathrel{|\!\sim} \bot \}.$$

Jede Konklusion eines Gegenarguments zu $\langle \mathcal{A}, h \rangle$ ist daher ein Literal aus $AP(\langle \mathcal{A}, h \rangle)$. Bei der Konstruktion von Gegenargumenten genügt es also, nur diejenigen potenziellen Gegenargumente zu betrachten, deren Konklusion in $AP(\langle \mathcal{A}, h \rangle)$ liegt. Zu beachten ist, dass Definition 5.13 von Angriffspunkten, genau die Literale beinhaltet, deren Komplemente in DeLP Angriffspunkte genannt werden, vgl. Definition 2.11.

Allgemeiner Ablauf

Bei der Berechnung von Gegenargumenten wird wieder der oben beschriebene Algorithmus `Arguments` benutzt. Eine Verwendung des Algorithmus von Besnard und Hunter

scheidet wegen den in Abschnitt 4.5 genannten Gründen aus und deshalb werden Argumente *online* auf Anfrage generiert. Da manche Argumente mehrfach benutzt werden könnten, empfiehlt es sich einmal generierte Argumente zu speichern, um eine erneute Generierung zu vermeiden.

Sei also $\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_n, h_n \rangle]$ eine Argumentationsfolge und $\langle \mathcal{A}_n, h_n \rangle$ das Argument, zu dem Gegenargumente generiert werden sollen. Da nur Gegenargumente mit Konklusionen in $AP(\langle \mathcal{A}_n, h_n \rangle)$ betrachtet werden müssen, wird zunächst diese Menge berechnet. Für jedes Literal $d \in AP(\langle \mathcal{A}_n, h_n \rangle)$ werden dann mit Hilfe des oben beschriebenen Algorithmus `Arguments` alle Argumente für d berechnet. Jedes generierte Argument wird auf Akzeptanz innerhalb der aktuellen Argumentationsfolge hin getestet und im positiven Fall zu der Menge der Angriffe hinzugefügt. Wie in Abschnitt 4.3 beschrieben, genügt es auch, nur die minimal zurückhaltenden Angriffe zu betrachten. Aus diesem Grund wird im letzten Schritt für jeden Angriff überprüft, ob Sub-Argumente dieses Angriffs auch Angriffe sind und im positiven Fall wird der betrachtete Angriff aus der Ergebnismenge entfernt. Es folgt eine formale Beschreibung des Algorithmus `Attacks` in Pseudocode:

```
1  Attacks((⟨A_1, s_1⟩, ..., ⟨A_n, s_n⟩), Δ, Π)
2    ap = AP(⟨A_n, s_n⟩, Δ, Π)
3    result = ∅
4    for each d ∈ ap do
5      arguments = Arguments(d, Δ, Π)
6      for each ⟨B, d⟩ ∈ arguments do
7        if Acceptable((⟨A_1, s_1⟩, ..., ⟨A_n, s_n⟩), ⟨B, d⟩, Π) then
8          result = result ∪ {⟨B, d⟩}
9    for each ⟨B, a⟩ ∈ result do
10     if there exists a ⟨C, b⟩ ∈ result with C ⊂ B then
11       result = result \ {⟨B, a⟩}
12   return result
```

Algorithmus 5.3: `Attacks`

Auf der Grundlage des Algorithmus `Attacks` kann nun eine Gegenargumentsfunktion ψ_{D-DeLP} definiert werden.

Definition 5.14 (ψ_{D-DeLP})**.** Sei Π eine globale, Δ eine lokale Wissensbasis zu Π und Λ eine Folge von Argumenten. Die Funktion $\psi_{D-DeLP} : \Sigma \to \mathfrak{P}(\Omega)$ ist dann definiert durch

$$\psi_{D-DeLP}(\Lambda) =_{def} \texttt{Attacks}(\Lambda, \Delta, \Pi).$$

Bestimmung von Angriffspunkten

Ist $\langle \mathcal{A}, h \rangle$ ein Argument, so teilt sich die Menge der Angriffspunkte von $\langle \mathcal{A}, h \rangle$ (nach Definition 5.13) in die folgenden beiden disjunkten Mengen auf (eventuell vorhandene Metainformationen der Regeln werden in der Darstellung vernachlässigt):

- Ist $\mathcal{A} = \{(h_1 \prec b_{1_1}, \ldots, b_{n_{n_1}}), \ldots, (h_m \prec b_{m_1}, \ldots, b_{m_{nm}})\}$, so sind zunächst alle Literale $\overline{h_1}, \ldots, \overline{h_m}$ mögliche Angriffspunkte von $\langle \mathcal{A}, h \rangle$, da $\langle \mathcal{A}, h \rangle$ für jede Regel $h_i \prec b_{i_1}, \ldots, b_{i_{n_i}}$ ein Sub-Argument $\langle \mathcal{C}, h_i \rangle$ beinhaltet.
- Durch die Ableitung der Literale $\{h_1, \ldots, h_n\}$ können sichere Regeln aus der globalen Wissensbasis angestoßen werden. Die Negationen von Kopfliteralen solcher Regeln sind weitere Angriffspunkte von $\langle \mathcal{A}, h \rangle$, da Argumente für die Negation dieser Kopfliterale bezüglich der globalen Wissensbasis widersprüchlich zu $\langle \mathcal{A}, h \rangle$ sind.

Bei der Berechnung der zweiten Teilmenge von Angriffspunkten ist zu beachten, dass nur Negationen derjenigen Kopfliterale sicherer Regeln in die Menge der Angriffspunkte mitaufgenommen werden, die nicht schon von der globalen Wissensbasis alleine abgeleitet werden können. Argumente für die Negation solcher Kopfliterale kann es nach Definition 4.6 nicht geben, da die lokale Wissensbasis eines Agenten stets konsistent mit der globalen Wissensbasis ist. Es folgt eine formale Beschreibung des Algorithmus `AP` in Pseudocode:

```
1  AP(⟨A, h⟩, Δ, Π)
2     ap  = {¬h|(h ≺ b_1, ..., b_n) ∈ A}
3     cl_1 = {¬f|Π ∪ Δ |~ f}
4     cl_2 = {¬f|Π ∪ Δ ∪ ap |~ f}
5     ap  = ap ∪ (cl_2 \ cl_1)
6     return ap
```

Algorithmus 5.4: `AP`

Akzeptanz von Gegenargumenten

Nachdem alle potenziellen Gegenargumente zu einem Argument berechnet wurden, ist für jedes dieser Gegenargumente zu prüfen, ob die entstandene Argumentationsfolge nach Definition 4.13 akzeptiert wird.

Es folgt eine formale Beschreibung des Algorithmus `Acceptable` in Pseudocode (sei $\succ$ eine beliebige Präferenzrelation unter Argumenten):

```
1  Acceptable([⟨A_1, h_1⟩, ..., ⟨A_n, h_n⟩], ⟨B, h⟩, Π)
2    let ⟨A', h'⟩ be the disagreement sub-argument
3      of ⟨A_n, h_n⟩ relative to ⟨B, h⟩
4    if B ⊆ A_j for one 1 ≤ j ≤ n then
5      return false
6    if n is even then
7      if A_1 ∪ A_3 ··· ∪ A_{n-1} ∪ B ∪ Π |~ ⊥ then
8        return false
9    if n is odd then
10     if A_2 ∪ A_4 ··· ∪ A_n ∪ B ∪ Π) |~ ⊥ then
11       return false
12   if ⟨A', h'⟩ ≻ ⟨B, h⟩ then
13     return false
14   if n > 1 then
15     if ⟨A_n, h_n⟩ and ⟨A_{n-1}, h_{n-1}⟩ are uncomparable then
16       if not ⟨B, h⟩ ≻ ⟨A', h'⟩ then
17         return false
18   return true
```

Algorithmus 5.5: `Acceptable`

Auf der Grundlage des Algorithmus `Acceptable` kann nun eine Akzeptanzfunktion η_{D-DeLP} definiert werden.

Definition 5.15 (η_{D-DeLP})**.** Sei Π eine globale, Δ eine lokale Wissensbasis zu Π, Λ ein Folge von Argumenten und $\langle \mathcal{A}, h \rangle$ ein Argument. Die Funktion $\eta_{D-DeLP} : \Sigma(\Omega) \to \{0, 1\}$ ist dann definiert durch

$$\eta_{D-DeLP}(\Lambda + \langle \mathcal{A}, h \rangle) =_{def} \begin{cases} 1 & \text{falls } \texttt{Acceptable}(\Lambda, \langle \mathcal{A}, h \rangle, \Pi) = true \\ 0 & \text{sonst} \end{cases} .$$

5.4 Einbindung von Präferenzrelationen

5.4.1 Überblick

Als Vergleichskriterium für Argumente kann ein argumentationsfähiges Multiagentensystem grundsätzlich jede Relation zwischen Argumenten benutzen. Als Beispiel für ein Vergleichskriterium wähle ich den Ansatz, der in [GS02] vorgeschlagen wird: Zwei Argumente werden zunächst mit *Generalized Specificity* miteinander verglichen und falls

die Argumente damit nicht vergleichbar sein sollten, werden eventuell vorhandene Zusatzinformationen der verwendeten Regeln verglichen. Zur Berechnung von *Generalized Specificity* benutze ich die Charakterisierung durch Aktivierungsmengen [SGCnS03].

5.4.2 *Generalized Specificity* und Aktivierungsmengen

Die Definition 2.12 der *Generalized Specificity* legt schon eine Vorgehensweise zur Berechnung von *Generalized Specificity* nahe. Zunächst folgt eine Übertragung der Definition 2.12 in die D-DeLP Terminologie:

Definition 5.16 (Generalized Specificity)**.** Sei Π eine globale, Δ eine lokale Wissensbasis zu Π, $\mathcal{F}$ die Menge aller Literale, die eine unsichere Ableitung aus $\Pi \cup \Delta$ haben und $\Pi_R \subseteq \Pi$ die Menge aller sicheren Regeln aus Π. Ein Argument $\langle \mathcal{A}_1, h_1 \rangle$ heißt *mindestens so spezifisch* wie ein Argument $\langle \mathcal{A}_2, h_2 \rangle$ (geschrieben $\langle \mathcal{A}_1, h_1 \rangle \succeq_{spec} \langle \mathcal{A}_2, h_2 \rangle$), g. d. w. für alle $H \subseteq \mathcal{F}$ gilt:

$$\Pi_R \cup H \cup \mathcal{A}_1 \mathrel{|\!\sim} h_1 \text{ und } \Pi_R \cup H \nvdash h_1 \text{ implizieren } \Pi_R \cup H \cup \mathcal{A}_2 \mathrel{|\!\sim} h_2.$$

Durch Berechnung und Vergleich der Aktivierungsmengen von zwei Argumenten $\langle \mathcal{A}_1, h_1 \rangle$ und $\langle \mathcal{A}_2, h_2 \rangle$ kann somit direkt eine Aussage über die Relation von $\langle \mathcal{A}_1, h_1 \rangle$ zu $\langle \mathcal{A}_2, h_2 \rangle$ gemacht werden. Um die möglichen Aktivierungsmengen zu einem Argument $\langle \mathcal{A}, h \rangle$ zu berechnen, wird zunächst der Begriff der Argumentvervollständigung (*argument completion*) benötigt (siehe [SGCnS03]):

Definition 5.17 (Argumentvervollständigung)**.** Sei Π eine globale, Δ eine lokale Wissensbasis zu Π und $\langle \mathcal{A}, h \rangle$ ein Argument mit $\mathcal{A} \subseteq \Delta$. Eine Argumentationsvervollständigung $\underline{\mathcal{A}}$ von $\langle \mathcal{A}, h \rangle$ ist die Menge aller unsicheren Regeln $r \in \mathcal{A}$ und aller sicheren Regeln $r' \in \Pi$, die bei einer Ableitung von h aus $\Pi \cup \mathcal{A}$ benutzt werden.

Da durch die Verwendung verschiedener sicherer Regeln grundsätzlich mehrere mögliche Ableitungen zu h aus $\Pi \cup \mathcal{A}$ existieren können, können auch verschiedene Argumentationsvervollständigungen zu einem Argument $\langle \mathcal{A}, h \rangle$ existieren.

Beispiel 5.4 ([SGCnS03], Beispiel 15)**.** Seien die globale Wissensbasis Π und die lokale Wissensbasis Δ gegeben durch

$$\begin{aligned} \Pi &= \{d, e, c, (h \leftarrow a), (b \leftarrow d), (b \leftarrow e)\} \text{ und} \\ \Delta &= \{(a \prec b, c)\}, \end{aligned}$$

wobei auf die Angabe von Metainformationen verzichtet wurde. Zu $\mathcal{P}$ existiert das Argument $\langle \mathcal{A}, h \rangle$ mit $\mathcal{A} = \{(a \prec b, c)\}$ und es existieren zwei mögliche Argumentationsvervollständigungen zu $\langle \mathcal{A}, h \rangle$:

$$\begin{aligned} \underline{\mathcal{A}}_1 &= \{(a \prec b, c), (h \leftarrow a), (b \leftarrow d)\}, \\ \underline{\mathcal{A}}_2 &= \{(a \prec b, c), (h \leftarrow a), (b \leftarrow e)\}. \end{aligned}$$

Definition 5.18 ($Lit(\underline{\mathcal{A}})$)**.** Sei $\langle \mathcal{A}, h\rangle$ ein Argument und $\underline{\mathcal{A}}$ eine Argumentvervollständigung zu $\langle \mathcal{A}, h\rangle$. Dann ist $Lit(\underline{\mathcal{A}})$ die Menge aller in den Regeln von $\underline{\mathcal{A}}$ vorkommenden Literale.

Beispiel 5.5. Im Beispiel 5.4 gilt

$$\begin{aligned} Lit(\underline{\mathcal{A}}_1) &= \{a, b, c, h, d\}, \\ Lit(\underline{\mathcal{A}}_2) &= \{a, b, c, h, e\}. \end{aligned}$$

Es folgt nun eine konstruktive Definition von Aktivierungsmengen nach [SGCnS03]:

Definition 5.19 (Aktivierungsmenge)**.** Sei $\langle \mathcal{A}, h\rangle$ ein Argument, $\underline{\mathcal{A}}$ eine Argumentvervollständigung von $\langle \mathcal{A}, h\rangle$ und $Lit(\underline{\mathcal{A}})$ die zugehörige Menge der Literale. Eine Menge $H \subseteq Lit(\underline{\mathcal{A}})$ heißt *Aktivierungsmenge* von $\underline{\mathcal{A}}$, wenn $H \cup \underline{\mathcal{A}} \vdash h$ und H minimal mit dieser Eigenschaft bzgl. Mengeninklusion ist. Es sei $Act\text{-}sets(\mathcal{A})$ die Menge aller Aktivierungsmengen von $\underline{\mathcal{A}}$.

Definition 5.20 (Nicht-triviale Aktivierungsmenge)**.** Sei Π eine globale Wissensbasis, $\langle \mathcal{A}, h\rangle$ eine Argument, $\underline{\mathcal{A}}$ eine Argumentvervollständigung von $\langle \mathcal{A}, h\rangle$ und $Lit(\underline{\mathcal{A}})$ die zugehörige Menge der Literale. Eine Menge $H \subseteq Lit(\underline{\mathcal{A}})$ heißt *nicht-triviale Aktivierungsmenge* von $\underline{\mathcal{A}}$, wenn H eine Aktivierungsmenge von $\underline{\mathcal{A}}$ ist und $H \cup \Pi \nvdash h$. Es sei $NTAct\text{-}sets(\underline{\mathcal{A}})$ die Menge aller nicht-trivialen Aktivierungsmengen von $\underline{\mathcal{A}}$.

Zur Berechnung der *Generalized Specificity* genügt es, sich auf die Menge der nicht-trivialen Aktivierungsmengen zu beschränken [SGCnS03].

Notation. Sei $ArgComp(\langle \mathcal{A}, h\rangle)$ die Menge aller Argumentvervollständigungen von $\langle \mathcal{A}, h\rangle$. Definiere

$$\begin{aligned} Act\text{-}sets(\mathcal{A}) &=_{def} \bigcup_{\underline{\mathcal{A}} \in ArgComp(\langle \mathcal{A}, h\rangle)} Act\text{-}sets(\underline{\mathcal{A}}), \\ NTAct\text{-}sets(\mathcal{A}) &=_{def} \bigcup_{\underline{\mathcal{A}} \in ArgComp(\langle \mathcal{A}, h\rangle)} NTAct\text{-}sets(\underline{\mathcal{A}}). \end{aligned}$$

Mit Hilfe der obigen Definitionen von Argumentvervollständigungen und Aktivierungsmengen lässt sich *Generalized Specificity* folgendermaßen charakterisieren, vgl. dazu [SGCnS03], Theorem 20:

Theorem 5.3. *Sei Π eine globale Wissensbasis, $\Pi_G \subseteq \Pi$ die Menge aller Fakten aus Π und $\langle \mathcal{A}_1, h_1\rangle$, $\langle \mathcal{A}_2, h_2\rangle$ zwei Argumente. Es gilt $\langle \mathcal{A}_1, h_1\rangle \succ_{spec} \langle \mathcal{A}_2, h_2\rangle$, g. d. w. die folgenden beiden Bedingungen gelten:*

1. *Für alle Mengen $H \in NTAct\text{-}sets(\mathcal{A}_1)$ gilt $\Pi_G \cup H \cup \mathcal{A}_2 \vdash h_2$.*
2. *Es existiert eine Menge $H' \in NTAct\text{-}sets(\mathcal{A}_2)$ mit $\Pi_G \cup H' \cup \mathcal{A}_1 \nvdash h_1$.*

Der Beweis dieser Aussage ist nachzulesen in [SGCnS03].

5.4.3 Eine Hybrid-Relation: *DAS Specificity*

Überblick

Als Beispiel für ein Kriterium zum Vergleich zweier Argumente wird in diesem Unterabschnitt die Relation *DAS*[2] *Specificity* vorgestellt, die sich auf den in [GS02] vorgeschlagenen Ansatz stützt. Zwei Argumente werden dabei zunächst mit *Generalized Specificity* verglichen und im Falle der Unvergleichbarkeit werden zusätzliche Metainformationen der in den Argumenten benutzten Regeln nach dem P-DeLP Ansatz aus Unterabschnitt 2.3.3 miteinander verglichen.

Definition 5.21 (Metainformationen von Argumenten)**.** Sei $\langle \mathcal{A}, h \rangle$ ein Argument mit $\mathcal{A} = \{(r_1, \alpha_1), \ldots, (r_n, \alpha_n)\}$. Dann ist die Metainformation $mi(\langle \mathcal{A}, h \rangle)$ des Arguments $\langle \mathcal{A}, h \rangle$ definiert durch

$$mi(\langle \mathcal{A}, h \rangle) =_{def} \min\{\alpha_1, \ldots, \alpha_n\}.$$

Diese Definition entspricht Definition 2.13 zur Berechnung von Metainformationen für Argumente in P-DeLP. Die Semantik sei hier ebenso wie dort definiert.

Verfügt eine Regel $r \in \mathcal{A}$ über keine explizit definierte Metainformation (ist also $\alpha = -1$), so ist automatisch auch $mi(\langle \mathcal{A}, h \rangle) = -1$.

Definition 5.22 (MI-Relation)**.** Seien $\langle \mathcal{A}_1, h_1 \rangle$ und $\langle \mathcal{A}_2, h_2 \rangle$ zwei Argumente mit

$$\begin{aligned} \beta_1 &= mi(\langle \mathcal{A}_1, h_1 \rangle), \\ \beta_2 &= mi(\langle \mathcal{A}_2, h_2 \rangle). \end{aligned}$$

Das Argument $\langle \mathcal{A}_1, h_1 \rangle$ heißt *strikt besser* als $\langle \mathcal{A}_2, h_2 \rangle$ bezüglich der Metainformationen, geschrieben $\langle \mathcal{A}_1, h_1 \rangle \succ_{meta} \langle \mathcal{A}_2, h_2 \rangle$, g. d. w.

$$\beta_1 > \beta_2 \quad \text{und} \quad \beta_2 \neq -1.$$

Die Relation $\succ_{meta}$ heißt *MI-Relation*.

Da die Angabe von Metainformationen bei Regeln optional ist, sollen Argumente, die Regeln ohne Metainformationen benutzen, durch die Relation $\succ_{meta}$ nicht automatisch als „schlechter“ angesehen werden. Allerdings kann eine alternative Definition von $\succ_{meta}$ ohne die Bedingung $\beta_2 \neq -1$ in bestimmten Situationen sinnvoll sein.

Formalisierung

Die Relation *DAS Specificity* verbindet die Relationen $\succ_{spec}$ nach Definition 5.16 und $\succ_{meta}$ nach Definition 5.22 in einer lexikografischen Weise. Dabei erhält die Relation $\succ_{spec}$ die höchste Priorität und nur im Fall der Unvergleichbarkeit mit $\succ_{spec}$ wird die Relation $\succ_{meta}$ verwendet.

[2] DAS bezieht sich auf das Distributed Argumentation System, das eine Implementierung des Konzepts verteilter Argumentation ist und in Kapitel 7 vorgestellt wird.

Definition 5.23 (DAS-Specificity)**.** Seien $\langle \mathcal{A}_1, h_1 \rangle$ und $\langle \mathcal{A}_2, h_2 \rangle$ zwei Argumente. Das Argument $\langle \mathcal{A}_1, h_1 \rangle$ heißt *strikt besser* als $\langle \mathcal{A}_2, h_2 \rangle$ bezüglich *DAS Specificity*, geschrieben $\langle \mathcal{A}_1, h_1 \rangle \succ_{das} \langle \mathcal{A}_2, h_2 \rangle$, g. d. w.

- $\langle \mathcal{A}_1, h_1 \rangle \succ_{spec} \langle \mathcal{A}_2, h_2 \rangle$ oder
- $\langle \mathcal{A}_1, h_1 \rangle$ und $\langle \mathcal{A}_2, h_2 \rangle$ nicht vergleichbar mit $\succ_{spec}$ sind und $\langle \mathcal{A}_1, h_1 \rangle \succ_{meta} \langle \mathcal{A}_2, h_2 \rangle$.

Auf der Grundlage dieser Definition kann nun der Algorithmus `Compare` für *DAS Specificity* formal definiert werden (sei Π eine globale Wissensbasis):

```
1  Compare(⟨A1, h1⟩, ⟨A2, h2⟩, Π)
2    result = CompareSpecificity(⟨A1, h1⟩, ⟨A2, h2⟩, Π)
3    if result = NOT_COMPARABLE then
4      return CompareMetaInformation(⟨A1, h1⟩, ⟨A2, h2⟩)
5    return result
```

Algorithmus 5.6: `Compare`

Der Algorithmus zur Berechnung der *DAS Specificity* benutzt die Charakterisierung der *Generalized Specificity* durch Aktivierungsmengen aus [SGCnS03]. Um zu zwei Argumenten $\langle \mathcal{A}_1, h_1 \rangle$ und $\langle \mathcal{A}_2, h_2 \rangle$ die *Generalized Specificity* zu berechnen, werden die Argumentvervollständigungen jedes Arguments und zu jeder Argumentvervollständigung die möglichen nicht-trivialen Aktivierungsmengen nach Definition 5.20 bestimmt. Anschließend wird getestet, ob alle zu $\langle \mathcal{A}_1, h_1 \rangle$ gehörigen nicht-trivialen Aktivierungsmengen das Argument $\langle \mathcal{A}_2, h_2 \rangle$ aktivieren und vice versa. Entsprechend Theorem 5.3 wird auf Grundlage der Ergebnisse dieser Tests die Relation von $\langle \mathcal{A}_1, h_1 \rangle$ zu $\langle \mathcal{A}_2, h_2 \rangle$ bezüglich *Generalized Specificity* zurückgeliefert. Es folgt eine formale Beschreibung des Algorithmus `CompareSpecificity` in Pseudocode:

```
1  CompareSpecificity(⟨A1, h1⟩, ⟨A2, h2⟩, Π)
2    ntActSets1 = ∅
3    for each A ∈ ArgComp(⟨A1, h1⟩, Π) do
4      ntActSets1 = ntActSets1 ∪ NTActSets(A)
5    ntActSets2 = ∅
6    for each A ∈ ArgComp(⟨A2, h2⟩, Π)
7      ntActSets2 = ntActSets2 ∪ NTActSets(A)
8    activates1 = ActSetTest(ntActSets2, ⟨A1, h1⟩, Π);
9    activates2 = ActSetTest(ntActSets1, ⟨A2, h2⟩, Π);
10   if activates1 and not activates2
11     return IS_BETTER
12   if activates2 and not activates1
13     return IS_WORSE
14   return NOT_COMPARABLE
```

Algorithmus 5.7: `CompareSpecificity`

Zur Berechnung der Argumentvervollständigungen wird wieder der Ansatz des *backward-chaining* aus [CnSG93] benutzt. Es wird allerdings auf eine Darstellung des Algorithmus `ArgComp` verzichtet, weil dieser sich kaum vom Algorithmus `Arguments` unterscheidet. Vielmehr lässt sich der Algorithmus `Arguments` dahingehend erweitern, dass direkt bei der Berechnung von Argumenten auch die jeweiligen Argumentvervollständigungen mitberechnet werden. Das ist keine erhebliche Erweiterung des Algorithmus, weil die sicheren Regeln ohnehin bei der Berechnung eines Arguments berücksichtigt werden und lediglich nicht zum Support des Arguments hinzugefügt werden.

Der folgende Algorithmus zur Berechnung der nicht-trivialen Aktivierungsmengen ist [SGCnS03] entnommen. Der Ansatz des *backward-chaining* wird erneut benutzt, um iterativ die Aktivierungsmengen aus den in der Argumentvervollständigung benutzten Regeln herzuleiten. Es werden dabei sowohl triviale als auch nicht-triviale Aktivierungsmengen berechnet, wobei die nicht-trivialen Aktivierungsmengen zur Bestimmung der *Generalized Specificity* ausreichen, vgl. Theorem 5.3. Für eine nähere Beschreibung des Algorithmus `NTActSets` siehe [SGCnS03]. Es folgt eine formale Beschreibung des Algorithmus `NTActSets` in Pseudocode:

```
1  NTActSets(A,h)
2    S = ∅
3    Push ({h},trivial) on S
4    result = ∅
5    while S not empty do
6      Pop (N,type) from S
7      if type = non − trivial
8        result = result ∪ N
9      for each l ∈ N do
10       if there exists r ∈ A with head l
11         N' = N \ {l}
12         N' = N' ∪ {l'|l' is a body literal of r}
13         if type = trivial and r is a strict rule
14           Push (N',trivial) on S
15         else
16           Push (N',non − trivial) on S
17   return result
```

Algorithmus 5.8: `NTActSets`

Der Algorithmus `ActSetTest` hat als Parameter die Menge $ntActSets$ und das Argument $\langle \mathcal{A}, h \rangle$. Er überprüft, ob jede Aktivierungsmenge $H \in ntActSets$ das Argument $\langle \mathcal{A}, h \rangle$ aktiviert, d. h. ob $H \cup \Pi_R \cup \mathcal{A} \mathrel{|\!\sim} h$ gilt, wenn $\Pi_R \subseteq \Pi$ die Menge der sicheren Regeln einer globalen Wissensbasis Π ist. Er liefert $true$, falls jede Aktivierungsmenge $\langle \mathcal{A}, h \rangle$ aktiviert und $false$ sonst.

Es folgt eine formale Beschreibung des Algorithmus `ActSetTest` in Pseudocode:

```
1 ActSetTest(ntActSets, ⟨A, h⟩, Π)
2   Π_R = {s | s is a strict rule in Π}
3   for each H ∈ ntActSets do
4     if not H ∪ Π_R ∪ A ⊢ h then
5       return false
6   return true
```

Algorithmus 5.9: `ActSetTest`

Sind die beiden zu vergleichenden Argumente mit *Generalized Specificity* nicht vergleichbar, so werden eventuell mit den Argumenten verknüpfte Metainformationen miteinander verglichen. Falls zu einem der Argumente keine Metainformation verfügbar ist, so sollen die beiden Argumente als nicht vergleichbar gelten. Es folgt eine formale Beschreibung des Algorithmus `CompareMetaInformation` in Pseudocode:

```
1 CompareMetaInformation(⟨A_1, h_1⟩, ⟨A_2, h_2⟩)
2   meta1 = mi(⟨A_1, h_1⟩)
3   meta2 = mi(⟨A_2, h_2⟩)
4   if meta1 > meta2 > −1 then
5     return IS_BETTER
6   if meta2 > meta1 > −1 then
7     return IS_WORSE
8   return NOT_COMPARABLE
```

Algorithmus 5.10: `CompareMetaInformation`

5.5 Zusammenfassung

In diesem Kapitel wurde ein argumentationsfähiges Multiagentensystem auf der Grundlage der Sprache D-DeLP aus Abschnitt 4.4 konzipiert und formalisiert. Nach einer informellen Beschreibung der Leistungsanforderungen wurden die Komponenten Moderator und Agent formal spezifiziert. Der Moderator eines argumentationsfähigen Multiagentensystems nimmt Anfragen von außen an das Systems an und koordiniert den Argumentationsprozess zwischen den Agenten. Obwohl ursprünglich eine Verwendung des Algorithmus von Besnard und Hunter [BH06] für die Generierung von Gegenargumenten angedacht war, konnte dieser jedoch wegen der besonderen Natur eines verteilten Systems nicht adaptiert werden. Die Agenten eines argumentationsfähigen Multiagentensystems generieren Argumente und Gegenargumente mit Hilfe des in [CnSG93] vorgeschlagenen Verfahrens des *backward-chainings*. Um spezielle Präferenzrelationen unter den Argumenten, wie die von P-DeLP hergeleitete possibilistische Präferenzrelation, zu erlauben,

wurde die dem Argumentationsprozess unterlegte Sprache um Metainformationen erweitert, die erlauben den einzelnen Wissensfragmenten eine quantitative Abschätzung der Sicherheit hinzuzufügen. Unter Verwendung dieser Metainformationen wurde die Präferenzrelation *DAS Specificity* vorgestellt und formal konzipiert. *DAS Specificity* verwendet als erstes Vergleichskriterium *Generalized Specificity* [Poo85, SGCnS03] und im Falle der Unvergleichbarkeit von Argumenten werden eventuell vorhandene Metainformationen für einen weiteren Vergleich herangezogen.

6 Argumentation am Beispiel

In diesem Kapitel wird ein Beispiel aus dem Rechtswesen dazu verwendet, logikbasierte Argumentation in einem Multiagentensystem zu veranschaulichen. Dazu wird das Beispiel formal in einem argumentationsfähigen Multiagentensystem repräsentiert und anschließend ein Argumentationsprozess simuliert.

Notation. Da die funktionalen Aspekte eines argumentationsfähigen Multiagentensystems in diesem Kapitel nicht von Relevanz sind, werden sie im Folgenden nicht mehr explizit genannt.

6.1 Ein Beispiel aus dem Rechtswesen

Das Rechtswesen ist ein gutes Anwendungsgebiet für logikbasierte Argumentation. Gerade dort ist es nötig, dass ein Konflikt durch stichhaltige Argumentation gelöst werden kann. Ein generelles Beispiel ist der Rechtsstreit, in dem Argumentation im eigentlichen Sinne zwischen mehreren Parteien stattfindet. Hierzu dient der folgende Rechtsfall aus dem Schuldrecht (leicht abgeändert aus [Bro96]):

> *Der Kunstsammler D aus Berlin will eine Skulptur von Lenoardo Da Vinci kaufen, die im Katalog eines Antiquariats (A) in Hamburg zum Preis von 5 000 Euro angeboten wird. Er beauftragt seinen Freund F, zu diesem Antiquariat zu fahren und die Skulptur für maximal 4 000 Euro zu erstehen. Diesem gelingt es nicht ganz, seinen Auftrag auszuführen, aber immerhin kann er die Skulptur am 3. November nach zähen Verhandlungen im Namen des D für 4 200 Euro erwerben. Entgegen der Erwartung des F ist D nicht begeistert und weigert sich, die ihm von F vorgelegte Rechnung zu bezahlen. A, dem nach einer Woche ohne Zahlungseingang Bedenken kommen, ruft am 11. November bei D an, um sich der Genehmigung des Rechtsgeschäfts zu versichern. Er trifft D in bester Laune und damit verbundener Großzügigkeit an. D teilt A mit, alles sei in Ordnung und er werde das Geld umgehend überweisen. A und D vereinbaren, dass A die Skulptur bis zum 30. November auf seine Kosten nach Berlin versenden soll. Am 24. November tritt A eine Auslandsreise an und vergisst, sich um die Angelegenheit zu kümmern. Erst bei seiner Rückkehr am 4. Dezember erinnert er sich an D und beauftragt seinen Angestellten Hans (H), die Skulptur zum Frachtunternehmen Fahrgut*

> *zu bringen. Auf der Fahrt zu Fahrgut wird H unverschuldet in einen Unfall verwickelt; die Skulptur wird zerstört. Der betrunkene, aber reiche Unfallgegner gibt H sofort 10 000 Euro für das beschädigte Auto und 6 000 Euro für die zerstörte Skulptur, die H umgehend bei A abliefert. D, der sich nach dem Verfliegen der guten Laune doch wieder über den hohen Preis geärgert hat, hatte die Skulptur bereits am 15. November für 4 800 Euro an einen Japaner weiterverkauft.*

Eine Frage, die sich in diesem Szenario stellt, ist, ob D von A eine Zahlung der ihm entgehenden 600 Euro verlangen kann. Um diese Frage zu beantworten, soll ein argumentationsfähiges Multiagentensystem konstruiert werden, bei dem zwei Agenten Für- und Gegensprecher dieser Zahlungspflicht darstellen und Argumente und Gegenargumente auf der Grundlage der Rechtsnormen des Bürgerlichen Gesetzbuchs der Bundesrepublik Deutschland (BGB) austauschen. Die benötigten Rechtsnormen und eine Repräsentation des obigen Szenarios sollen geeignet in einer passenden logischen Sprache dargestellt werden, auf die sich der Argumentationsprozess stützen soll.

Rechtlich gesehen besteht der Anspruch auf Ersatz statt Leistung, falls beispielsweise die folgenden vier Bedingungen gelten [BGB07]:

1. Es ist ein Kaufvertrag zwischen D und A zustande gekommen.
2. Die mit dem Kaufvertrag verbundene Pflicht zur Leistung wurde von A nicht erbracht.
3. Die Pflichtverletzung aus 2. ist aus Unmöglichkeit der Leistung entstanden, d. h. A war nicht in der Lage zu liefern, weil beispielsweise die Skulptur zerstört wurde.
4. A muss die Pflichtverletzung aus 2. vertreten, d. h. Ersatz statt Leistung erbringen.

Ich werde mich in diesem Kapitel genauer mit der Modellierung der Frage, ob ein Kaufvertrag zwischen A und D zustande gekommen ist, beschäftigen. Die Implementierung des vollständigen Beispiels für das in Kapitel 7 vorzustellende DAS ist in Anhang A zu finden.

Das zu formalisierende System soll aus zwei Agenten bestehen: dem Fürsprecher von A (*proponent*), der also gegen eine Zahlung der 600 Euro argumentiert, und einem Gegensprecher von A (*opponent*), der für die Zahlung der 600 Euro argumentiert. Als sicheres Wissen des Systems werden die in dem Beispiel vorkommenden Fakten gelten, wie z. B. die Tatsache, dass D dem F eine Innenvollmacht ausgesprochen hat, um die Skulptur für maximal 4 000 Euro zu erstehen. Die Agenten sollen auf Grundlage der Rechtsnormen des BGB diese Fakten nutzen, um Argumente für bzw. gegen die Leistung der 600 Euro hervorzubringen. Dazu ist es nötig, dass die nötigen Rechtsnormen des BGB als logische Regeln repräsentiert werden. Die Rechtsnormen werden dabei als

unsichere Regeln modelliert und einem (oder beiden) Agenten zugeordnet. Eine Modellierung der Rechtsnormen durch unsichere statt durch sichere Regeln ist hier angebracht, da Rechtsnormen nicht zwingend gelten müssen, sondern durch andere („spezifischere“) Rechtsnormen revidiert werden können. Da die Spezifizität der Rechtsnormen von Belang ist, wurde versucht, diese Spezifizität möglichst genau auf die Formalisierung zu übertragen, so dass als Vergleichskriterium für Argumente die *Generalized Specificity* nahegelegt wird. Der Moderator des Systems übernimmt bei diesem Argumentationsprozess die Rolle einer übergeordneten Rechtsinstanz: Er koordiniert die Argumentation, bewertet die ausgetauschten Argumente und entscheidet anschließend den Rechtsstreit.

6.2 Formale Wissensrepräsentation

Anmerkung. Die folgende Wissensrepräsentation des obigen Rechtsfalls dient nur als Beispiel für die Verwendung eines argumentationsfähigen Multiagentensystem. Aus diesem Grund wird der Fall sehr vereinfacht dargestellt und erhebt keinen Anspruch auf formale Korrektheit im Sinne der allgemeinen deutschen Rechtsprechung. Unter anderem sind rechtswissenschaftliche Fachbegriffe, wie „Bevollmächtiger“, „Vertretungsmacht“ und „Einverständnis“, in ihrer Bedeutung leicht verschieden zu ihrer Bedeutung im Rechtswesen. Diese Anpassungen und Vereinfachungen sind nötig, um ein illustratives Beispiel für ein argumentationsfähiges Multiagentensystem zu erhalten.

Das Wissen des Systems wird in einer einfachen propositionallogischen Form ohne Regelschemata repräsentiert. Eine Übertragung des Szenarios auf einen allgemeinen Fall ist jedoch leicht möglich.

Die Agenten sollen argumentieren, ob ein gültiger Kaufvertrag zwischen D und A zustande gekommen ist. Diese Aussage werde durch die Proposition kv repräsentiert, die also auch die Anfrage an das System darstellt. Der Fürsprecher von A soll so argumentieren, dass kein gültiger Kaufvertrag zwischen D und A zustandegekommen ist. In diesem Fall muss A dann auch keinen Ersatz leisten. Der Gegensprecher von A soll argumentieren, dass ein gültiger Kaufvertrag zwischen D und A zustandegekommen ist, womit ein Anspruch auf Leistung oder Ersatz statt Leistung besteht.

Notation. Für die Wissensrepräsentation in diesem Abschnitt wird die Angabe von Metainformationen nicht benötigt. Es sei also implizit $\alpha = 1$ für alle Fakten und $\alpha = -1$ für alle unsicheren Regeln. Sichere Regeln werden bei der Modellierung nicht benötigt.

Zur Klärung der Frage, ob ein gültiger Kaufvertrag zwischen D und A zustande gekommen ist, werden die folgenden Fakten aus dem Wissen, das im Beispiel enthalten ist, gebildet:

$\neg selbsterklaerung$
D hat keine eigene Erklärung für den Kauf der Skulptur gegeben.

eigene_we_in_fremdem_namen
F hat seine Willenserklärung an A im Namen von D gegeben und das auch entäußert.

innenvollmacht
F hat von D eine Innenvollmacht zur Abwicklung des Kaufgeschäfts erhalten.

$\neg$*einhaltung_der_grenzen*
F hat die Grenzen der ihm verliehenen Innenvollmacht nicht eingehalten, da er die Skulptur zu einem höheren Preis als von D gefordert, erworben hat.

$\neg$*einverstaendnis*
D hat F zu dem Kauf der Skulptur zum erhöhten Preis nicht sein Einverständnis gegeben.

aufforderung_genehmigung
A hat D per Telefon dazu aufgefordert, den von F in Ds Namen abgewickelten Kauf zu genehmigen.

genehmigung
D hat A die Genehmigung zum Kauf durch F am Telefon gegeben.

§ 433 BGB regelt vertragstypische Pflichten beim Kaufvertrag. Findet er im Beispiel Anwendung, so ist ein gültiger Kaufvertrag zustande gekommen. Der Gegensprecher von A erhält auf Grundlage dieser Rechtsnorm somit die unsichere Regel:

$O1 : kv \prec p433$
Findet § 433 BGB zur Regelung vertragstypischer Pflichten beim Kaufvertrag Anwendung, so ist ein Kaufvertrag zustande gekommen.

Der Einfachheit halber gehe ich davon aus, dass § 433 BGB die einzige Rechtsnorm im BGB ist, die sich mit dieser Fragestellung befasst. Findet sie keine Anwendung, so sei kein gültiger Kaufvertrag zwischen D und A zustande gekommen. Der Fürsprecher von A erhalte somit eine unsichere Regel:

$P1 : \neg kv \prec \neg p433$
Findet § 433 BGB zur Regelung vertragstypischer Pflichten beim Kaufvertrag keine Anwendung, so ist kein Kaufvertrag zustande gekommen.

Die Frage, ob ein gültiger Kaufvertrag zwischen D und A zustandegekommen ist, reduziert sich somit auf die Frage, ob § 433 BGB im Beispiel Anwendung findet. Der Fürsprecher von A kann im unspezifischsten Fall zunächst argumentieren, dass § 433 BGB keine Anwendung findet (und somit kein Kaufvertrag zustande gekommen ist), weil D keine eigene Erklärung zur Kaufabsicht geleistet hat:

$P2 : \neg p433 \prec \neg selbsterklaerung$
D hat zunächst keine Selbsterklärung zum Kauf der Skulptur gegeben und somit findet § 433 BGB keine Anwendung.

Mit Hilfe dieser beiden Regeln kann der Fürsprecher von A also das folgende Argument gegen die Wirksamkeit eines Kaufvertrages geben:

$$\langle \{P1, P2\}, \neg kv \rangle$$

§ 433 BGB findet jedoch auch Anwendung, obwohl keine Selbsterklärung von D gegeben wurde, wenn ein nach § 164 BGB mit Vertretungsmacht ausgestatteter Dritter (F) im Namen von D den Kaufvertrag abschließt. Der Gegensprecher von A erhalte somit die unsicheren Regeln:

$O2 : p433 \prec \neg selbsterklaerung, bevollmaechtigter$
§ 433 BGB findet Anwendung, obwohl keine Selbsterklärung von D gegeben wurde, wenn F als bevollmächtiger Dritter in Ds Namen gehandelt hat.

$O3 : bevollmaechtigter \prec p164$
Findet § 164 BGB zur Erklärung eines rechtmäßigen Vertreters Anwendung, so ist F bevollmächtigt in Ds Namen zu handeln.

Es ist zu klären, ob § 164 BGB zur Erklärung eines rechtmäßigen Vertreters Anwendung findet. Dies ist der Fall, wenn F dem A gegenüber die Absicht erklärt, die Skulptur im Namen von D zu erwerben und F mit entsprechender Vertretungsmacht für D handelt. Die Erteilung von Vertretungsmacht wird in § 167 BGB geregelt und findet Anwendung, wenn

- entweder D dem F gegenüber eine Vollmacht erteilt, in seinem Namen zu handeln (Innenvollmacht) oder
- D dem A gegenüber erklärt, dass F in seinem Namen handeln darf (Außenvollmacht).

Somit erhalte der Gegensprecher von A die folgenden unsicheren Regeln:

$O4 : p164 \prec eigene\_we\_in\_fremdem\_namen, vertretungsmacht$
§ 164 BGB findet Anwendung, wenn F dem A gegenüber eine Willenserklärung über eine Kaufabsicht des D ausspricht und F die entsprechende Vertretungsmacht besitzt.

$O5 : vertretungsmacht \prec p167$
Findet § 167 BGB zur Regelung von Befugnissen Dritter Anwendung, so hat F Vertretungsmacht.

$O6 : p167 \prec innenvollmacht$
§ 167 BGB findet Anwendung, wenn F durch D direkt bevollmächtigt wurde.

$O7 : p167 \prec aussenvollmacht$
§ 167 BGB findet Anwendung, wenn D den F indirekt bevollmächtigt, indem er A gegenüber erklärt, dass F in seinem Namen handeln darf.

Auf Grundlage dieser Regeln, kann der Gegensprecher von A das Argument

$$\langle \{O2, O3, O4, O5, O6\}, p433 \rangle$$

als Gegenargument zu $\langle \{P1, P2\}, \neg kv \rangle$ am Angriffspunkt $\neg p433$ und mit der Regel $O1$ das Argument

$$\langle \{O1, O2, O3, O4, O5, O6\}, kv \rangle$$

als eigenes Argument für das Zustandekommen eines Kaufvertrages hervorbringen.

F hat jedoch in dem Beispiel nicht so gehandelt, wie D es ihm bei der Erteilung der Vertretungsmacht aufgetragen hat, sondern den vorgegeben Maximalpreis nicht eingehalten. Das bedeutet, dass F die Grenzen seiner Vertretungsmacht nicht eingehalten hat und somit § 167 BGB keine Anwendung findet, auch wenn dem F eine Innenvollmacht erteilt wurde. Der Fürsprecher von A erhalte somit die Regeln

$P3 : \neg vertretungsmacht \prec \neg p167$
§ 167 BGB regelt Befugnisse Dritter und wenn er keine Anwendung findet, hat F gegenüber A keine Vertretungsmacht für D.

$P4 : \neg p167 \prec innenvollmacht, \neg einhaltung\_der\_grenzen$
§ 167 BGB findet keine Anwendung, wenn einem Dritten zwar Innenvollmacht erteilt wurde, er aber nicht innerhalb seiner Vertretungsmacht gehandelt hat.

$P5 : \neg bevollmaechtigter \prec \neg vertretungsmacht$
Hat F keine Vertretungsmacht, so ist er nicht bevollmächtigt, im Namen von D zu handeln.

Mit Hilfe dieser Regeln kann der Fürsprecher von A zunächst das Argument

$$\langle \{P4\}, \neg p167 \rangle$$

als Gegenargument zu den Argumenten

$$\langle \{O2, O3, O4, O5, O6\}, p433 \rangle \quad \text{und} \quad \langle \{O1, O2, O3, O4, O5, O6\}, kv \rangle$$

mit jeweiligem Angriffspunkt $p167$ hervorbringen.

D hat daraufhin auch F gegenüber sein Einverständnis für den Kauf nicht gegeben und damit den Vertrag implizit verweigert. Der Fürsprecher von A kann somit seine Argumente für das Nichtzustandekommen des Vertrages weiter spezifizieren. Der Fürsprecher von A erhalte die Regeln:

$P6 : \neg p433 \prec \quad \neg selbsterklaerung, \neg bevollmaechtigter, verweigerung\_vertrag$

D hat keine Selbsterklärung für den Kauf der Skulptur gegeben, F war nicht bevollmächtigt den Kauf im Namen von D zu tätigen und D hat den Kaufvertrag zunächst verweigert. Somit findet § 433 BGB keine Anwendung.

$P7 : verweigerung\_vertrag \prec \neg einverstaendnis$

D hat zunächst den Vertrag konkludent verweigert, da er F gegenüber nicht sein Einverständnis für den überteuerten Kauf gegeben hat.

Der Vertrag gilt jedoch nicht mehr als verweigert, wenn eine Aufforderung zur Genehmigung des Vertrags vorliegt und er gilt als akzeptiert, wenn die Genehmigung erteilt wird. A hat D am Telefon dazu aufgefordert, den Vertrag zu genehmigen, und D hat anschließend diese Genehmigung erteilt. Der Gegensprecher von A erhalte also die Regeln:

$O8 : \neg verweigerung\_vertrag \prec \quad \neg einverstaendnis, aufforderung\_genehmigung$

Ein Vertrag gilt nicht als verweigert, obwohl keine Einverständniserklärung abgegeben wurde, wenn eine Aufforderung zur Genehmigung vorliegt.

$O9 : p433 \prec \quad \neg selbsterklaerung, aufforderung\_genehmigung, genehmigung$

§ 433 BGB findet Anwendung, obwohl keine Selbsterklärung von D gegeben wurde, wenn eine Aufforderung zur Genehmigung von A erteilt und die Genehmigung von D gegeben wurde.

6.3 Formalisierung des Systems und des Argumentationsprozesses

Auf der Grundlage der vorangegangenen Wissensrepräsentation kann nun das formale Modell eines argumentationsfähigen Multiagentensystems aufgestellt werden. Nach Notation werden die funktionalen Aspekte des Systems nicht betrachtet, hier seien entsprechende Funktionen wie in den Beispielen aus Unterabschnitt 5.2.1 gegeben. Das argumentationsfähige Multiagentensystem T_{KV} ist definiert durch

$$T_{KV} =_{def} (\Delta, \{proponent, opponent\})$$

mit der globalen Wissensbasis

$$\Pi =_{def} \left\{ \begin{array}{ll} \neg selbsterklaerung, & eigene\_we\_in\_fremdem\_namen, \\ innenvollmacht, & \neg einhaltung\_der\_grenzen, \\ \neg einverstaendnis, & aufforderung\_genehmigung, \\ genehmigung & \end{array} \right\},$$

wobei jedes Faktum aus Π über eine Metainformation $\alpha = 1$ verfüge und den lokalen Wissensbasen

$$\begin{array}{rll} \Delta_{proponent} & =_{def} & \{P1, \ldots, P7\}, \\ \Delta_{opponent} & =_{def} & \{O1, \ldots, O9\} \end{array}$$

der Agenten *proponent* bzw. *opponent*, wobei $\alpha = -1$ für jede Regel aus $\Delta_{proponent} \cup \Delta_{opponent}$ gelte.

Auf die Anfrage kv an T_{KV} generiert der Agent *proponent* die beiden Wurzelargumente

$$\langle \{P1, P2\}, \neg kv \rangle \quad \text{bzw.} \quad \langle \{P1, P3, P4, P5, P6, P7\}, \neg kv \rangle$$

und der Agent *opponent* die beiden Wurzelargumente

$$\langle \{O1, O9\}, kv \rangle \quad \text{bzw.} \quad \langle \{O1, O2, O3, O4, O5, O6\}, kv \rangle.$$

Es werden also insgesamt vier dialektische Bäume rekursiv weiter aufgebaut, wobei als Vergleichskriterium für Argumente *Generalized Specificty* nach Definition 5.16 benutzt wird.

1. Auf das Argument $\langle \{P1, P3, P4, P5, P6, P7\}, \neg kv \rangle$ von Agent *proponent* antwortet der Agent *opponent* mit den beiden Gegenargumenten

   $$\langle \{O9\}, p433 \rangle \quad \text{und} \quad \langle \{O8\}, \neg verweigerung\_vertrag \rangle,$$

   auf die der Agent *proponent* keine weiteren Gegenargumente erzeugen kann. Der entsprechende dialektische Baum ist in Abbildung 6.1 dargestellt.

2. Auf das Argument $\langle \{P1, P2\}, \neg kv \rangle$ von *proponent* kann der Agent *opponent* mit den beiden Gegenargumenten

   $$\langle \{O9\}, p433 \rangle \quad \text{und} \quad \langle \{O2, O3, O4, O5, O6\}, p433 \rangle$$

   antworten. Auf das Argument $\langle \{O2, O3, O4, O5, O6\}, p433 \rangle$ kann *proponent* mit dem spezifischeren Argument

   $$\langle \{P4\}, \neg p167 \rangle$$

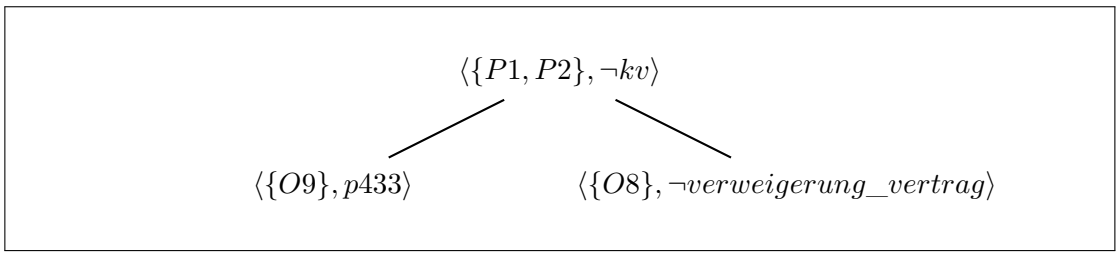

Abbildung 6.1: Erster dialektischer Baum zu T_{KV}.

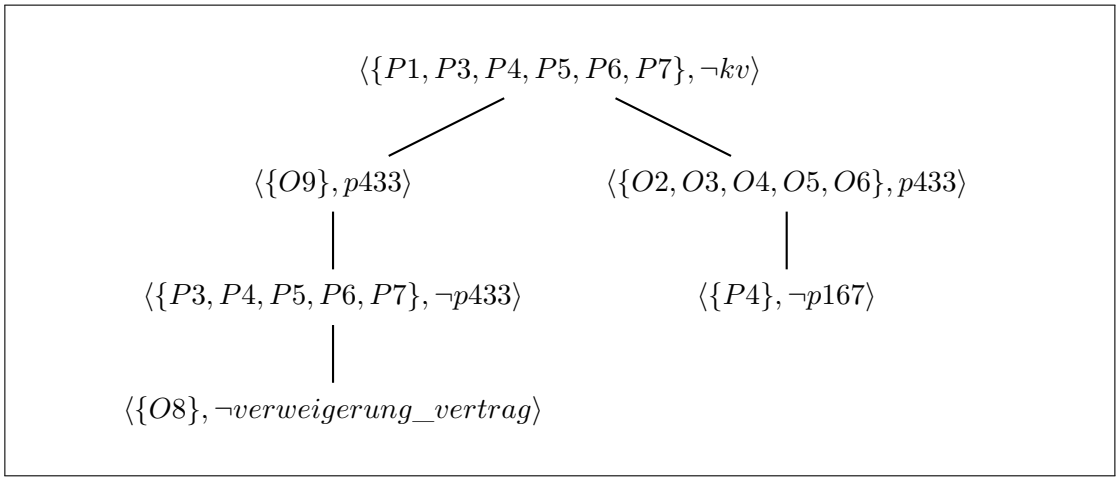

Abbildung 6.2: Zweiter dialektischer Baum zu T_{KV}

antworten, worauf *opponent* keine Gegenargumente generieren kann. Auf das Argument $\langle\{O9\}, p433\rangle$ kann *proponent* mit dem blockierenden Argument

$$\langle\{P3, P4, P5, P6, P7\}, \neg p433\rangle$$

antworten, worauf *opponent* mit dem Gegenargument

$$\langle\{O8\}, \neg verweigerung\_vertrag\rangle$$

antwortet. Weitere Argumente können nicht generiert werden. Der entsprechende dialektische Baum ist in Abbildung 6.2 dargestellt.

3. Auf das Argument $\langle\{O1, O9\}, kv\rangle$ von *opponent* antwortet Agent *proponent* mit dem blockierenden Gegenargument

$$\langle\{P3, P4, P5, P6, P7\}, \neg p433\rangle,$$

worauf Agent *opponent* mit dem spezifischeren Argument

$$\langle\{O8\}, \neg verweigerung\_vertrag\rangle$$

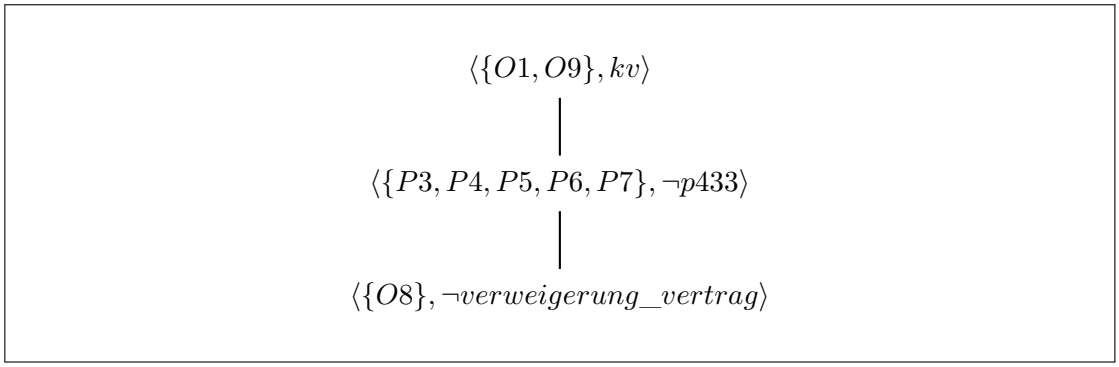

Abbildung 6.3: Dritter dialektischer Baum zu T_{KV}

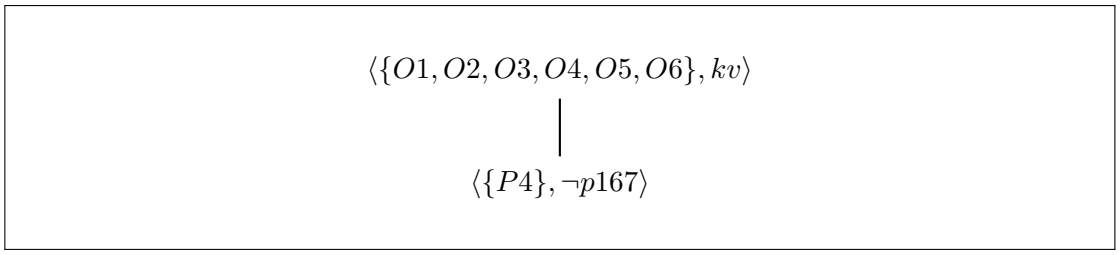

Abbildung 6.4: Vierter dialektischer Baum zu T_{KV}

antwortet. Weitere Argumente können nicht generiert werden. Der entsprechende dialektische Baum ist in Abbildung 6.3 dargestellt.

4. Auf das Argument $\langle\{O1, O2, O3, O4, O5, O6\}, kv\rangle$ von Agent *opponent* antwortet der Agent *proponent* mit dem Gegenargument

   $$\langle\{P4\}, \neg p167\rangle,$$

   worauf der Agent *opponent* keine weiteren Gegenargumente erzeugen kann. Der entsprechende dialektische Baum ist in Abbildung 6.4 dargestellt.

Nach einer dialektischen Analyse der konstruierten dialektischen Bäume gemäß Definition 2.19 zeigt sich, dass die Wurzel des dritten dialektischen Baumes mit „U“ markiert wird. Die Konklusion des entsprechenden Arguments wird also vom System geglaubt. Als Antwort auf die Anfrage kv liefert das System T_{KV} somit YES zurück, d. h. das System hat nicht-widerlegbare Argumente für das Zustandekommen eines Kaufvertrages zwischen D und A.

6.4 Bewertung

Nach Konstruktion verhält sich T_{KV} genauso, wie dies in der allgemeinen deutschen Rechtsprechung der Fall ist [Bro96]. Das Argument $\langle\{O1, O9\}, kv\rangle$ von *opponent* wird nicht widerlegt, da der einzige Angriff $\langle\{P3, P4, P5, P6, P7\}, \neg p433\rangle$ durch das spezifischere Gegenargument $\langle\{O8\}, \neg verweigerung\_vertrag\rangle$ widerlegt wird. Die Verwendung der *Generalized Specificity* als Präferenzrelation unter Argumenten stellt sicher, dass unspezifischere Gegenargumente nicht als Angriffe zugelassen werden. Denn auch im Rechtswesen sind die in einem Fall spezifischsten Argumente die besten. Obwohl D keine eigene Einverständniserklärung gegeben hat, gilt ein Kaufvertrag als zustande gekommen, wenn auf eine Aufforderung zur Genehmigung eines Vertrages die Genehmigung erteilt wurde. Das weniger schlüssige Argument $\langle\{O1, O2, O3, O4, O5, O6\}, kv\rangle$ von *opponent* wird widerlegt, da in diesem Fall F über keine Vertretungsmacht verfügt. Ebenso werden die Argumente von *proponent* gegen das Zustandekommen eines Kaufvertrages widerlegt, da *opponent* über spezifischere Gegenargumente verfügt.

7 Eine Implementierung verteilter logikbasierter Argumentation

Es folgt die Beschreibung einer Implementierung des vorangegangenen Konzepts zur verteilten logikbasierten Argumentation in Java.

7.1 Programmbeschreibung

Distributed Argumentation System (DAS) ist ein System für verteilte logikbasierte Argumentation mit unsicherem Wissen und ist auf der beigelegten CD zu finden. Es unterstützt die Erstellung von argumentationsfähigen Multiagentensystemen nach Definition 5.8 und verfügt über eine Anfrageauswertung, wobei die in einem Argumentationsprozess konstruierten dialektischen Bäume grafisch dargestellt werden können. Globale und lokale Wissensbasen können erstellt und editiert sowie verschiedene Anfragen an das System gestellt werden. Weiterhin verfügt DAS über Lade- und Speicher-Funktionen, so dass erstellte Szenarien und einzelne Agenten importiert und gesichert werden können. Das System ist modular aufgebaut und kann um zusätzliche Strukturen zur Wissensrepräsentation und um Präferenzrelationen erweitert werden.

Das Hauptfenster von DAS ist in Abbildung 7.1 dargestellt und teilt sich in vier Bereiche (siehe Abbildung 7.2):

- Szenarioeinstellungen (Subkomponenten 1 bis 6)
- Agenteneinstellungen (Subkomponenten 7 bis 10)
- Ausgabebereich (Subkomponente 11)
- Menü- und Toolbar-Bereich (Subkomponente 12)

Im Folgenden werden die einzelnen Subkomponenten beschrieben.

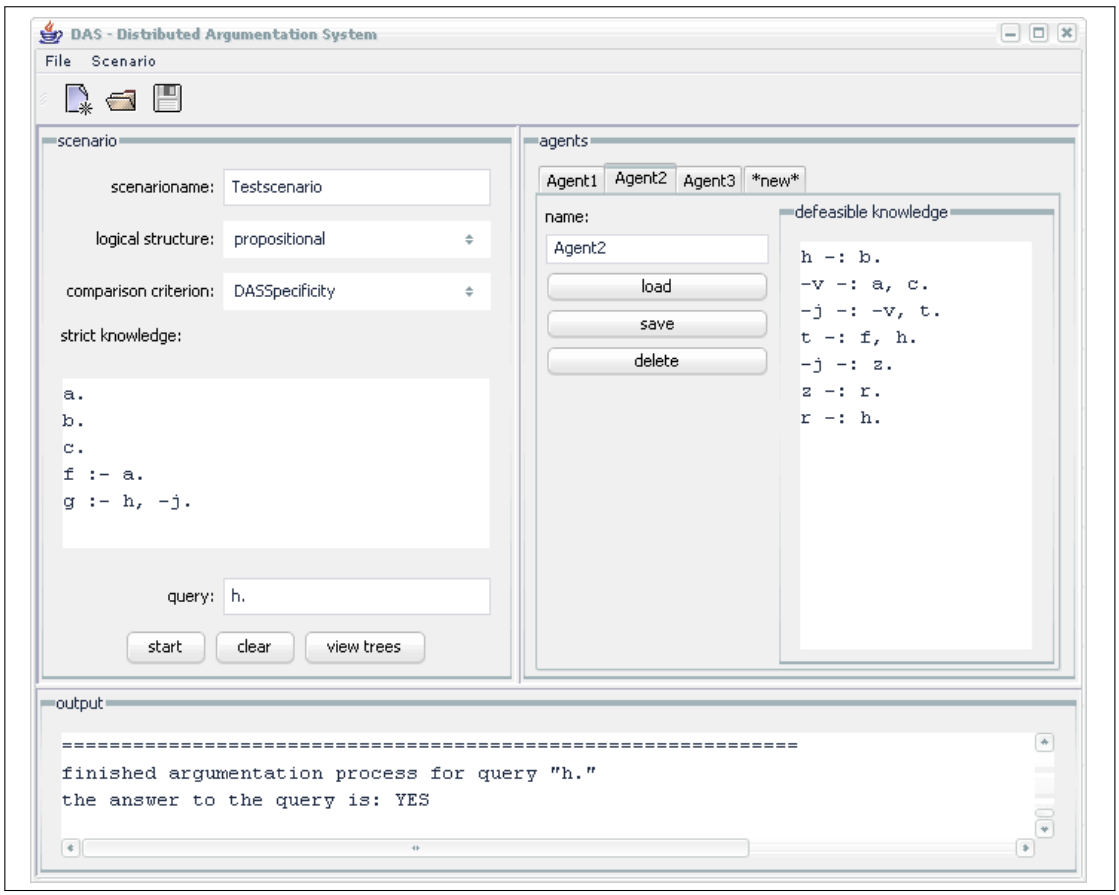

Abbildung 7.1: Das Hauptfenster von DAS

7.1.1 Szenarioeinstellungen

Im Bereich Szenarioeinstellungen werden globale Einstellungen für ein Szenario angezeigt und können modifiziert werden. Die einzelnen Subkomponenten sind:

1. *scenarioname*: Der Name des gerade geladenen Szenarios. Er kann frei gewählt werden und hat keine semantische Bedeutung im Argumentationsprozess.

2. *logical structure*: Die dem System unterlegte logische Struktur. In der derzeitigen Fassung ist folgende Auswahl möglich:

   a) *propositional*: Bei dieser Einstellung werden Fakten und Literale in den Köpfen und Rümpfen der Regeln durch atomare Propositionen oder mit Hilfe des „-“-Negationssymbols durch die Negation einer atomaren Proposition dargestellt.

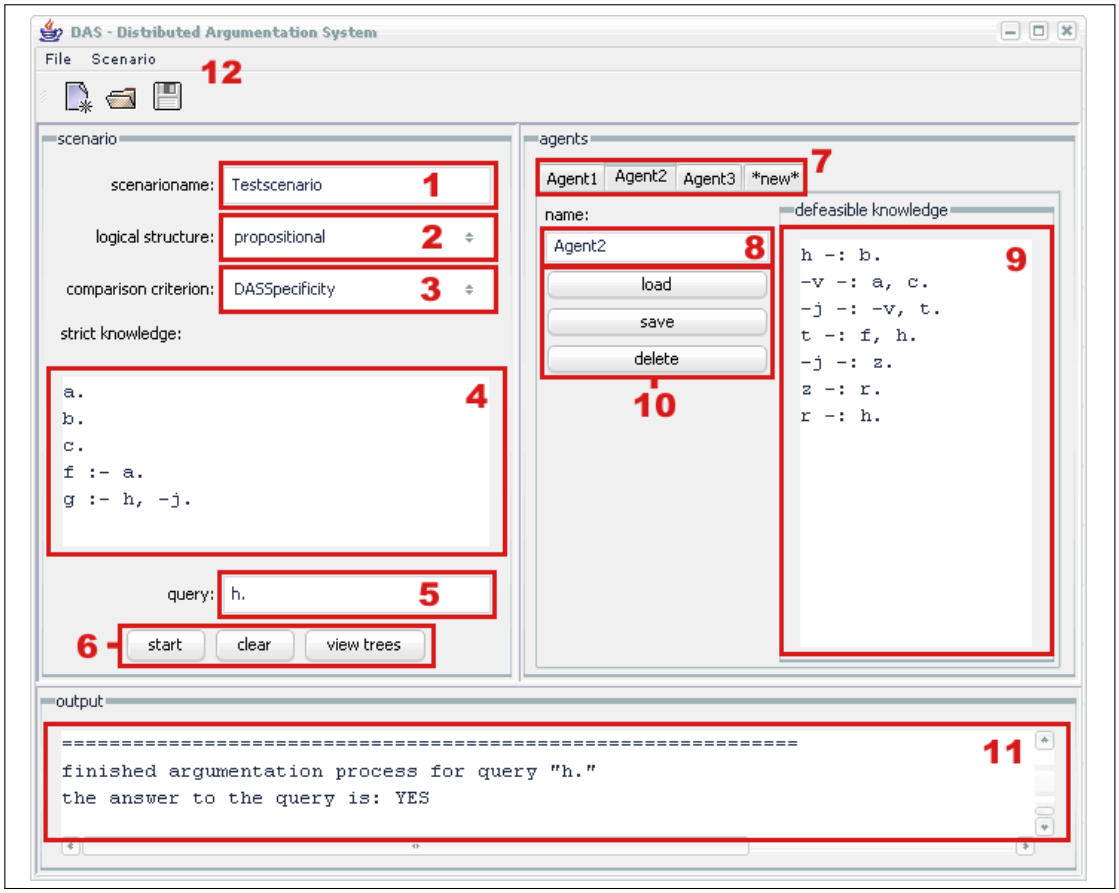

Abbildung 7.2: Die einzelnen Komponenten des Hauptfensters von DAS

b) *propositional (with metainformation)*: Bei dieser Einstellung muss zu jedem Fakt und zu jeder Regel des Systems eine Metainformation in Form einer reellen Zahl $\alpha \in [0,1] \cup \{-1\}$ in eckigen Klammern hinten angestellt werden. Das dient der Realisierung der in Unterabschnitt 5.4.3 vorgestellten *DAS Specificity*. Eine Angabe von $\alpha = -1$ bedeutet, dass keine explizite Metainformation gesetzt wird. Beispiele: `a -: b [0,5].`, `c -: d, e [1].`, `f -: g, h [-1].`

Eine Erweiterung des Systems um die Unterstützung von parametrisierten Prädikaten und Regelschemata ist aber durchaus möglich.

3. *comparison criterion*: Hier wird die Präferenzrelation angegeben, die das System zum Vergleich von Argumenten benutzen soll. In der derzeitigen Fassung von DAS ist nur die Auswahl von *DAS Specificity*, wie sie in Unterabschnitt 5.4.3 beschrieben

wurde, möglich. Wird als logische Struktur *propositional* verwendet, verhält sich *DAS Specificity* genau so wie *Generalized Specificity*. Das System kann aber um andere Präferenzrelationen erweitert werden.

4. *strict knowledge*: Hier wird die globale Wissensbasis des Systems modelliert, die Fakten und sichere Regeln enthalten kann. Jedes Wissensfragment muss mit einem Punkt „.“ abgeschlossen werden, das Folgerungssymbol für sichere Regeln ist „:-“.

5. *query*: Die Anfrage an das System in Form eines Literals. Sie muss mit einem Punkt „.“ terminiert werden.

6. *start-*, *clear-*, *view-trees-Buttons*: Mit diesen Buttons werden grundlegende Aktionen ausgelöst:
   - Mit dem *start-Button* wird der Argumentationsprozess angestoßen, der daraufhin im Ausgabebereich 11 protokolliert wird.
   - Mit dem *clear-Button* wird der Inhalt des Ausgabefeldes 11 gelöscht.
   - Der *view-trees-Button* ist nur nach einem erfolgreichen Argumentationsprozess aktiviert und öffnet ein neues Fenster, in dem die bei dem Argumentationsprozess generierten dialektischen Bäume grafisch dargestellt werden (siehe Abbildung 7.3). Wurde bei dem Argumentationsbaum mehr als ein dialektischer Baum erzeugt, so können über das Auswahlfeld die verschiedenen dialektischen Bäume betrachtet werden. In einem dialektischen Baum werden Knoten, die bei einer dialektischen Analyse mit „D“ markiert wurden, rot dargestellt und die Knoten, die mit „U“ markiert wurden, werden grün dargestellt.

7.1.2 Agenteneinstellungen

In diesem Bereich kann die Anzahl der Agenten des Systems eingestellt und es können die individuellen Wissenbasen konfiguriert werden. Die einzelnen Subkomponenten sind:

7. *agent panels*: Hier können die einzelnen Agenten des Systems angewählt und neue Agenten hinzugefügt werden.

8. *name*: Der Name des Agenten. Wie der Szenarioname kann der Agentenname frei gewählt werden und hat keine weitere semantische Bedeutung innerhalb des Argumentationsprozesses. Allerdings wird dieser Name bei der Protokollierung des Argumentationsprozesses in Ausgabebereich 11 benutzt, um diesen Agenten zu bezeichnen.

9. *defeasible knowledge*: Hier wird die lokale Wissensbasis des gerade angezeigten Agenten in Form von unsicheren Regeln spezifiziert. Jede Regel muss mit einem

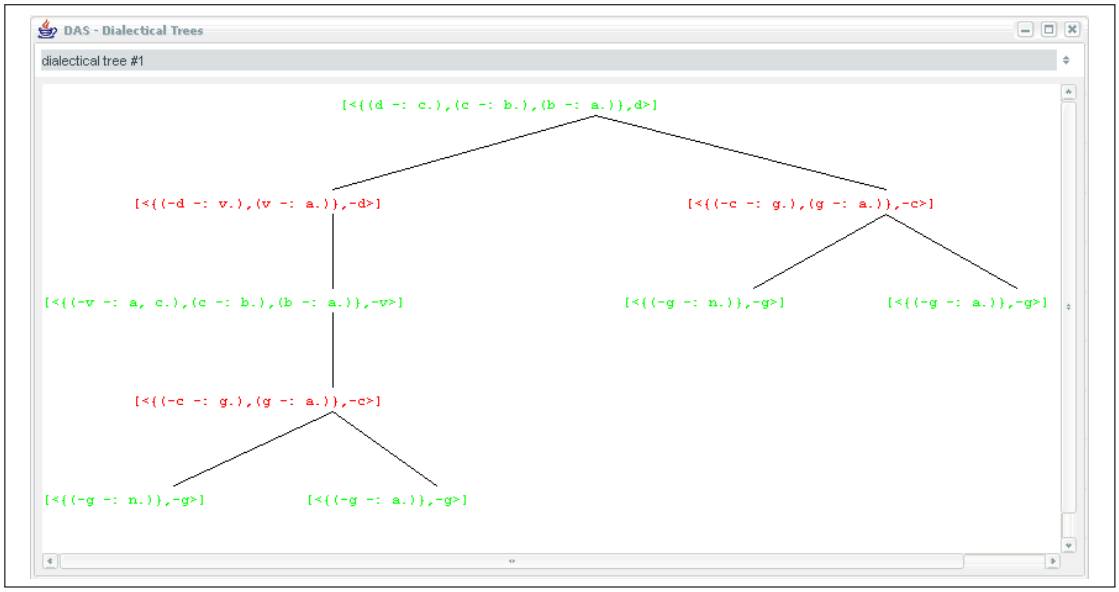

Abbildung 7.3: Ein von DAS generierter dialektischer Baum

Punkt „.“ abgeschlossen werden, das Folgerungssymbol für unsichere Regeln ist „-:“.

10. *load*-, *save*-, *delete-Buttons*: Mit Hilfe dieser Buttons können Agenten importiert, gespeichert und gelöscht werden.

7.1.3 Ausgabebereich

In der Subkomponente 11 werden die Ausgaben des Argumentationsprozesses generiert. Diese umfassen:

- das Protokoll des Koordinationsprozesses durch den Moderator
- das Protokoll der Argumentationsgenerierung der Agenten
- das Protokoll des Analyseprozesses durch den Moderator
- die auf die Anfrage an das System generierte Antwort

7.1.4 Menü- und Toolbar-Bereich

In den Menüs (Subkomponente 12) befinden sich diverse Funktionen zur Dateiverwaltung wie „*Open*“, „*Save*“ und „*New*“ mit den üblichen Bedeutungen sowie die Funktionen

„*start*“ und „*view trees*“ mit denselben Bedeutungen wie die Buttons in Subkomponente 6. Die Toolbar verfügt über Schnellzugriffe auf die Funktionen „*Open*“, „*Save*“ und „*New*“.

7.2 Benutzung am Beispiel

DAS speichert Agenten und Szenarien in XML-Dateien, die über die integrierten Funktionen importiert, bearbeitet und exportiert werden können. Eine vollständige Implementierung des in Kapitel 6 vorgestellten Beispiels aus dem Rechtswesen in DAS befindet sich im Anhang A und auf der beigelegten CD als XML-Datei „`DAS_Example_01.xml`“.

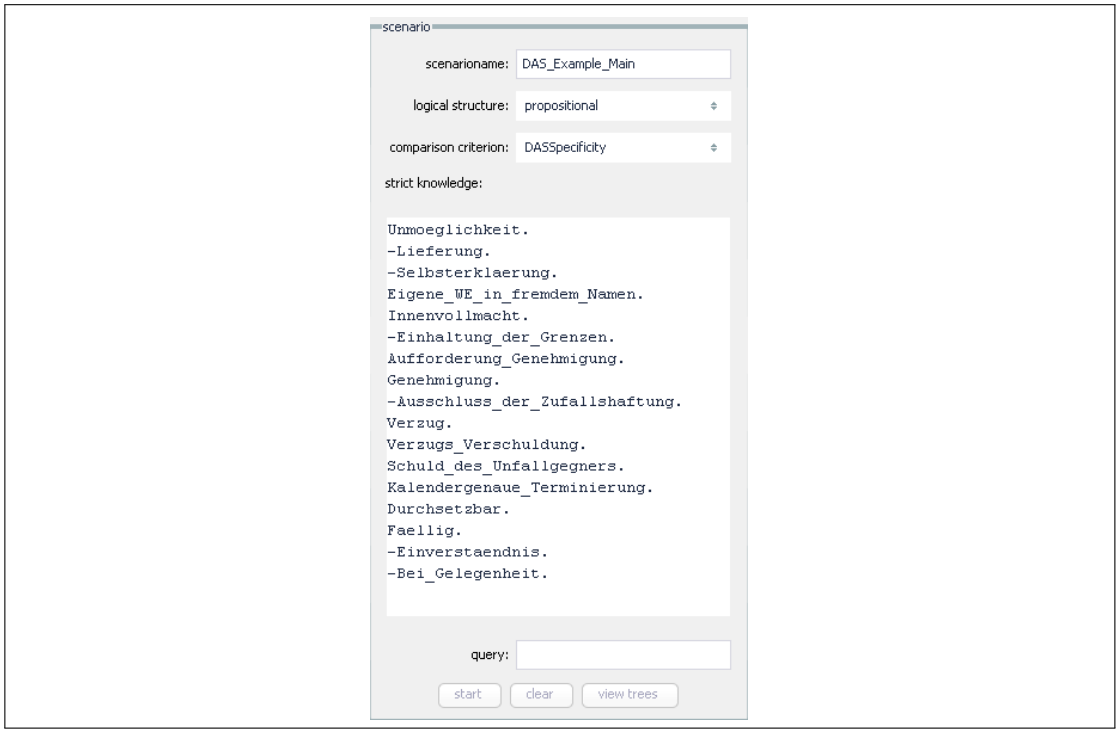

Abbildung 7.4: Die Szenarioeinstellungen für das System aus Kapitel 6

Bei der Modellierung der globalen Wissensbasis muss auf Konsistenz geachtet werden. Ist die globale Wissensbasis nicht konsistent, bricht jeder Argumentationsprozess mit einer entsprechenden Fehlermeldung ab. Zur Modellierung der globalen Wissenbasis können Fakten und sichere Regeln verwendet werden. Die Wahl der unterlegten logischen Struktur beeinflusst die zu verwendende Syntax bei der Spezifikation der globalen und lokalen Wissensbasen. Bei der Verwendung der propositionallogischen Syntax mit Metainforma-

tionen, muss jedem Wissensfragment eine Metainformation in eckigen Klammern angefügt werden. Für Wissensfragmente, die über keine explizite Metainformation verfügen sollen, wird -1 als Metainformation notiert. Die Auswahl der Präferenzrelation für Argumente ist derzeit auf *DAS Specificity* nach Definition 5.23 beschränkt. Abbildung 7.4 zeigt die Szenarioeinstellungen für das Rechtsbeispiel aus Kapitel 6. Als logische Struktur wurde einfache propositionale Syntax gewählt und als Präferenzrelation für Argumente *DAS Specificity*.

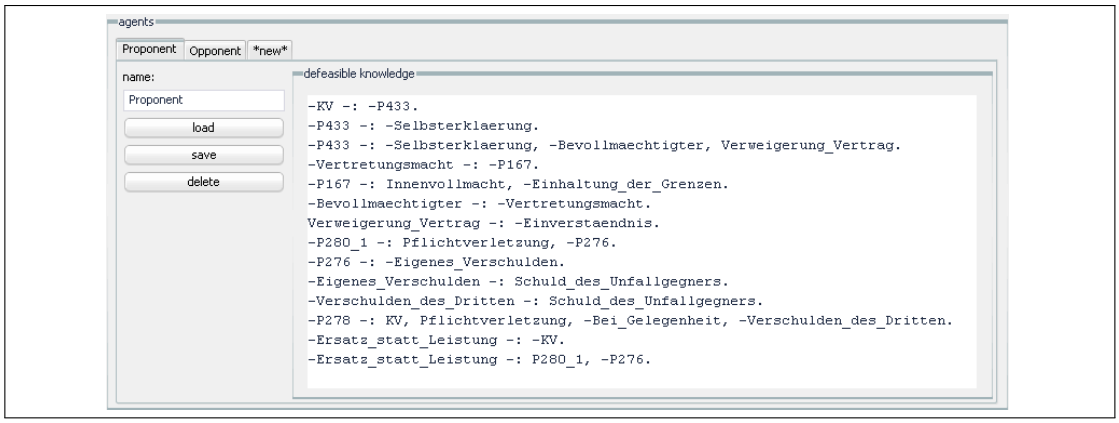

Abbildung 7.5: Der Agent `proponent` in DAS für das System aus Kapitel 6

Bei der Modellierung der lokalen Wissensbasen ist ebenfalls auf Konsistenz bzgl. der globalen Wissensbasis zu achten. Ist eine lokale Wissensbasis bzgl. der globalen Wissensbasis nicht konsistent, so bricht jeder Argumentationsprozess mit einer entsprechenden Fehlermeldung ab. Um einen nicht-trivialen Argumentationsprozess zu erhalten, müssen mindestens zwei Agenten spezifiziert werden. Werden weniger Agenten erzeugt, gibt das System entsprechende Warnhinweise aus. Für das Beispiel aus Anhang A ist die Spezifikation von zwei Agenten `proponent` und `opponent` nötig, deren lokale Wissensbasen in den Abbildungen 7.5 und 7.6 in DAS dargestellt sind.

Ist das System vollständig spezifiert, können verschiedene Anfragen gestellt werden. Der durch eine Anfrage angestoßene Argumentationsprozess wird ausführlich im Ausgabebereich protokolliert und anschließend wird ebendort auch die auf die Anfrage generierte Antwort des Systems gegeben. Um den Argumentationsprozess zu visualisieren, können die einzelnen dialektischen Bäume über den Button „*view trees*“ betrachtet werden. Im Beispielsystem werden nach dem Starten des Argumentationsprozesses auf die Anfrage

`Ersatz_statt_Leistung`

sieben verschiedene dialektische Bäume generiert und die Anfrage wird mit YES beantwortet, da es einen dialektischen Baum gibt, dessen Wurzelargument ein Argument

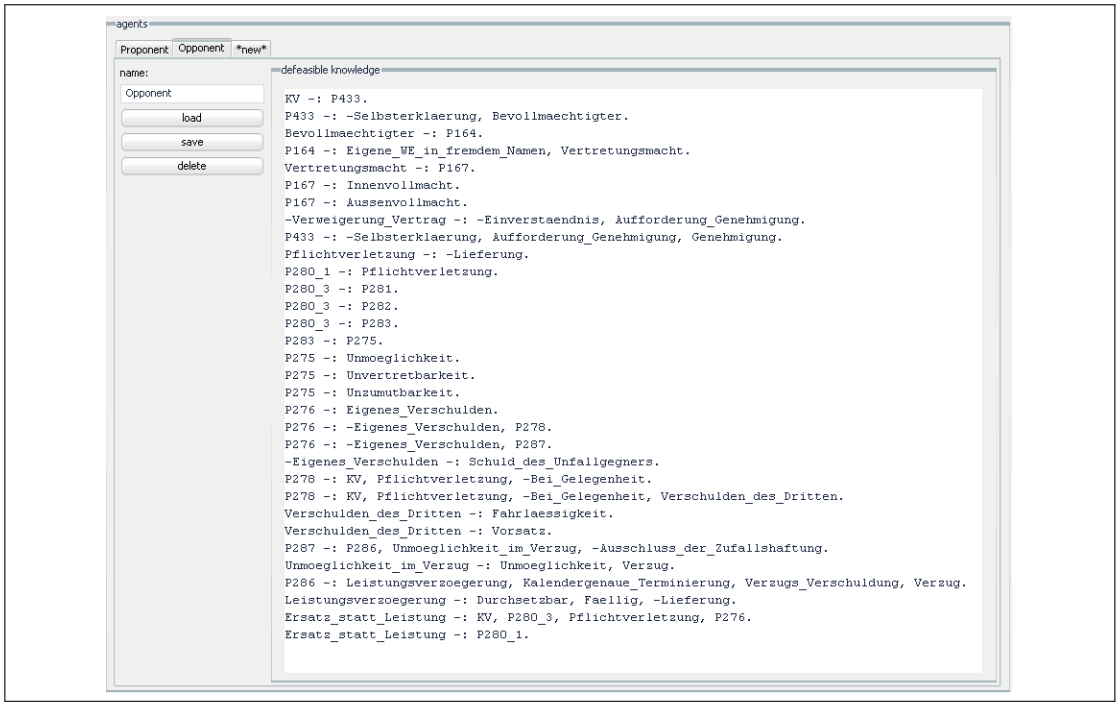

Abbildung 7.6: Der Agent `opponent` in DAS für das System aus Kapitel 6

für die Anfrage ist, das schlussendlich nicht widerlegt wird. Begründung und Widerlegung von Argumenten wird in der Visualisierung der dialektischen Bäume durch eine unterschiedliche farbliche Hervorhebung gekennzeichnet.

7.3 Implementierungsbeschreibung

In diesem Abschnitt wird ein grober Umriss über die Implementierung von DAS gegeben, insbesondere eine Auflistung der Pakete und Klassen. Da die Implementierung von DAS größtenteils aus einer Übertragung der Algorithmen aus den Abschnitten 5.3 und 5.4 besteht, wird auf eine genauere Beschreibung verzichtet. Der vollständige Quellcode von DAS ist auf der beigelegten CD zu finden.

Nach JAVA-Konvention ist DAS in das Paket `edu.udo.cs.ie.das` einsortiert[1] und besteht aus den JAVA-Paketen:

[1] Nach Konvention sollen JAVA-Projekte eine möglichst eindeutige Paketbezeichnung erhalten. Das System DAS wurde dazu dem Lehrbereich (`edu`), der Universität Dortmund (`udo`), dem Fachbereich Informatik (`cs`) und der Arbeitsgruppe *Information Engineering* (`ie`) zugeordnet.

edu.udo.cs.ie.das.argumentation_structures
Das Paket `argumentation_structures` beinhaltet Entity-Klassen für die zur Argumentation notwendigen Strukturen wie Argumente, Fakten und Regeln.

edu.udo.cs.ie.das.comparison_criterions
Das Paket `comparison_criterions` beinhaltet die Präferenzrelation *DAS Specificity* und Hilfsklassen zur Konstruktion eigener Präferenzrelationen.

edu.udo.cs.ie.das.das_structures
Das Paket `das_structures` beinhalt Klassen für Agenten, Moderatoren und Szenarien.

edu.udo.cs.ie.das.gui
Das Paket `gui` beinhaltet die Klassen zur Realisierung der grafischen Benutzeroberfläche.

edu.udo.cs.ie.das.logic
Das Paket `logic` enthält abstrakte Klassen und Schnittstellen zur Realisierung verschiedener logischer Grundbausteine.

edu.udo.cs.ie.das.logic.propositional_logic
Die Klassen des Pakets `propositional_logic` implementieren die Funktionalitäten des `logic`-Paketes zur Realisierung einfacher propositionaler Syntax.

edu.udo.cs.ie.das.logic.propositional_logic_extended
Die Klassen des Pakets `propositional_logic_extended` implementieren die Funktionalitäten des `logic`-Paketes zur Realisierung propositionaler Syntax mit Metainformationen.

edu.udo.cs.ie.das.util
Das Paket `util` enthält diverse Hilfsklassen, wie beispielsweise eine Klasse zum Zeichnen eines dialektischen Baumes.

Es folgt nun einer Auflistung aller Klassen der einzelnen Pakete:

Paket `edu.udo.cs.ie.das.argumentation_structures`

Fact
Die Klasse `Fact` kapselt ein Literal einer beliebigen unterlegten Logik zur Verwendung in Argumenten und Wissensbasen, vgl. Definition 4.2).

`Rule`
Die Klasse `Rule` ist die Superklasse für die Klassen `StrictRule` und `DefeasibleRule` und kapselt die gemeinsame Grundstruktur von Regeln.

`StrictRule`
Die Klasse `StrictRule` ist eine Spezialisierung der Klasse `Rule` nach Definition 4.3.

`DefeasibleRule`
Die Klasse `DefeasibleRule` ist eine Spezialisierung der Klasse `Rule` nach Definition 4.4.

`DialecticalTree`
Die Klasse `DialecticalTree` dient der Realisierung von dialektischen Bäumen nach Definition 4.14. Eine Instanz von `DialecticalTree` realisiert einen Knoten eines dialektischen Baumes, welcher einen Vor- und beliebig viele Nachfahren haben kann, und enthält neben einer Instanz der Klasse `Argument` auch ein Attribut zur Markierung.

`Reasoner`
Die Klasse `Reasoner` kapselt Methoden zur Ableitung, welche u. a. von den Klassen `KnowledgeBase` und `Agent` benötigt werden.

`KnowledgeBase`
Die Klasse `KnowledgeBase` stellt die globale Wissensbasis eines Multiagentensystems nach Definition 4.6 dar.

`Argument`
Die Klasse `Argument` stellt ein Argument nach Definition 4.9 dar und besteht aus einer Menge von Instanzen der Klasse `DefeasibleRule` und aus einer Instanz der Klasse `Fact`.

Paket `edu.udo.cs.ie.das.comparison_criterions`

`ComparisonCriterion`
Diese abstrakte Klasse liefert ein Grundgerüst für die Implementierung von Präferenzrelationen unter Argumenten.

`ArgumentCompletion`
Diese Klasse ist eine Subklasse von `Argument` zur Repräsentation einer Argumentvervollständigung nach Definition 5.17.

`DASSpecificity`
Die Klasse `DASSpecificity` ist eine Implementierung der Klasse `ComparisonCriterion` und realisiert die Relation *DAS Specificity* nach Definition 5.23 und den Algorithmen aus Abschnitt 5.4.

Paket `edu.udo.cs.ie.das.das_structures`

`Agent`
Die Klasse `Agent` ist eine Implementierung eines argumentationsfähigen Agenten nach Definition 5.7 und der Algorithmen aus Abschnitt 5.3.

`Moderator`
Die Klasse `Moderator` ist eine Implementierung eines Moderators nach Definition 5.4 und der Beispielfunktionen aus Abschnitt 5.2.

`Scenario`
Die Klasse `Scenario` kapselt eine Menge von Instanzen der Klasse `Agent`, eine Instanz der Klasse `Moderator` sowie Instanzen der Klassen `KnowledgeBase`, `ComparisonCriterion` und `LogicParser` (s.u.), zur Realisierung eines argumentationsfähigen Multiagentensystems nach Definition 5.8.

Paket `edu.udo.cs.ie.das.gui`

`DAS_Main`
Die Klasse `DAS_Main` ist die Hauptklasse der grafischen Benutzeroberfläche und instantiiert die Klasse `GUI_MainFrame` zur Darstellung der grafischen Benutzeroberfläche.

`GUI_MainFrame`
Die Klasse `GUI_MainFrame` stellt das Hauptfenster von DAS dar.

`AgentPanel`
Die Klasse `AgentPanel` dient der Visualisierung eines Agentenreiters im Hauptfenster.

`Output`
Die Klasse `Output` ist eine Hilfsklasse zur Kapselung des Protokolls eines Argumentationsprozesses.

`TreeFrame`
Die Klasse `TreeFrame` dient der Visualisierung der dialektischen Bäume.

`TreePanel`
Die Klasse `TreePanel` stellt den Zeichenbereich für die Visualisierung eines dialektischen Baumes zur Verfügung.

Paket `edu.udo.cs.ie.das.logic`

`BasicElement`
Die Schnittstelle `BasicElement` stellt das Grundgerüst zur Realisierung von Standardelementen wie Proposition und Negation logischer Sprachen bereit.

`LogicParser`
Die abstrakte Klasse `LogicParser` dient dem Parsen logischer Konstrukte aus den Eingabebereichen von DAS in die entsprechenden logischen Objekte.

Paket `edu.udo.cs.ie.das.logic.propositional_logic`

`Proposition`
Die Klasse `Proposition` realisiert eine atomare Proposition.

`Negation`
Die Klasse `Negation` realisiert die Negation einer atomaren Proposition.

`PL_Parser`
Die Klasse `PL_Parser` ist eine Implementierung der abstrakten Klasse `LogicParser` zum Parsen propositionaler Konstrukte.

Paket `edu.udo.cs.ie.das.logic.propositional_logic_extended`

`PLE_Parser`
Die Klasse `PLE_Parser` ist eine Erweiterung der Klasse `PL_Parser` zum Parsen propositionaler Konstrukte mit Metainformationen.

Paket `edu.udo.cs.ie.das.util`

`Factory`
Die Klasse `Factory` implementiert die Schnittstelle `java.util.ComboBoxModel` und dient der dynamischen Erzeugung von Listen der verfügbaren logischen Sprachen und Präferenzrelationen zur Darstellung im Hauptfenster von DAS.

`VTree`
Die Klasse `VTree` enthält Algorithmen zu einer ansprechenden Visualisierung der dialektischen Bäume.

`XMLSchemaLocation`
Die Klasse `XMLSchemaLocation` enthält die Pfade zu den Schemata der DAS-XML-Spezifikationen.

Abbildung 7.7 zeigt eine schematische Darstellung aller Klassen von DAS und ihrer Relationen in Form eines vereinfachten UML-Diagramms. Um die Übersicht zu wahren, wurde dabei auf die Angabe von Attributen und Methoden verzichtet.

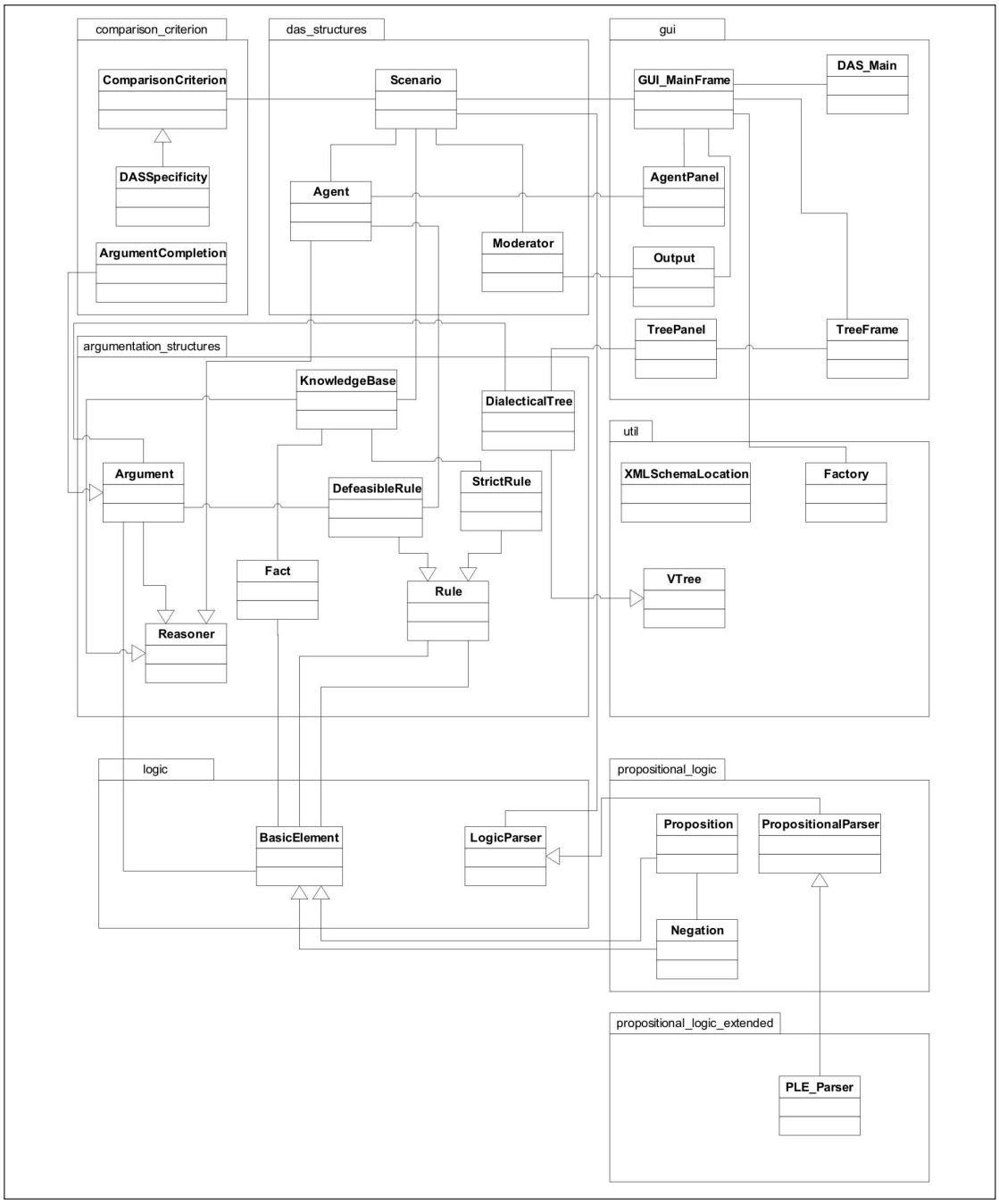

Abbildung 7.7: UML-Diagramm zu DAS

8 Vergleich von D-DeLP und DAS mit dem Ansatz von García und Simari

In den beiden folgenden Abschnitten werden D-DeLP und DAS mit DeLP und einer Realisierung von DeLP, dem *DeLP-Server*, verglichen. Auf einen expliziten Vergleich von D-DeLP zu dem Ansatz von Besnard und Hunter wird hier verzichtet. Da D-DeLP auf DeLP basiert, gelten dieselben Ergebnisse, wie bei dem Vergleich von DeLP mit den Ansatz von Besnard und Hunter (siehe Kapitel 4).

8.1 Vergleich von D-DeLP mit DeLP

In diesem Abschnitt wird das in Abschnitt 5.2 formalisierte argumentationsfähige Multiagentensystem auf Basis der in Abschnitt 4.4 definierten Sprache D-DeLP mit den Realisierungen der Moderator- und Agentenfunktionen durch die Algorithmen aus Abschnitt 5.3 mit dem System DeLP [GS02] verglichen. Da D-DeLP auf der Grundlage von DeLP spezifiziert wurde, erfolgt dies durch Rückführung eines argumentationsfähigen Multiagentensystems in ein DeLP-Programm.

Um ein DeLP-Programm in ein argumentationsfähiges Multiagentensystem zu überführen und konsistente lokale Wissensbasen zu erhalten, muss die Menge der unsicheren Regeln Δ_R eines DeLP-Programms $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ „geeignet“ auf mehrere Agenten aufgeteilt werden. Aufgrund einer gewissen Willkür bei der Aufteilung, sind mehrere gültige Lösungen denkbar. Für den Beweis der nächsten Aussage ist allerdings eine triviale Einteilung von Δ_R auf mehrere Agenten ausreichend:

Definition 8.1 (Argumentbasierte Regelaufteilung)**.** Sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ ein DeLP-Programm und Q die Menge aller Argumente von $\mathcal{P}$. Die *argumentbasierte Regelaufteilung* $AD(\Delta_R)$ von $\mathcal{P}$ ist definiert durch

$$AD(\Delta_R) = \{\mathcal{A} | \langle \mathcal{A}, h \rangle \in Q\}.$$

Für jedes mögliche Argument $\langle \mathcal{A}, h \rangle$ wird die Menge $\mathcal{A}$ als lokale Wissensbasis eines Agenten definiert. Auf diese Weise wird sichergestellt, dass die lokalen Wissensbasen konsistent sind, aber Argumente durch die Aufteilung der unsicheren Regeln nicht verloren gehen. Die Metainformation jeder unsicheren Regel sei dabei für D-DeLP nicht gesetzt ($\alpha = -1$).

Definition 8.2 (Induziertes argumentationsfähiges Multiagentensystem)**.** Sei $AD(\Delta_R)$ die argumentbasierte Regelaufteilung des DeLP-Programms $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$. Definiere $\Pi = \Pi_G \cup \Pi_R$ und $\{\Delta_1, \ldots, \Delta_n\} = AD(\Delta_R)$ als die lokalen Wissensbasen der Agenten $A_1, \ldots, A_n$. Ist M ein Moderator, dessen funktionale Bestandteile gemäß DeLP wie in den Beispielen in Abschnitt 5.2 definiert sind, so heißt $T = (M, \Pi, \{A_1, \ldots, A_n\})$ das von $\mathcal{P}$ *induzierte argumentationsfähige Multiagentensystem.*

Nach Konstruktion der lokalen Wissensbasen gilt zunächst, dass ein induziertes argumentationsfähiges Multiagentensystem ein argumentationsfähiges Multiagentensystem nach Definition 5.8 ist. Weiterhin gilt:

Theorem 8.1. *Sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ ein DeLP-Programm und T das von $\mathcal{P}$ induzierte argumentationsfähige Multiagentensystem. Sei h eine Anfrage an $\mathcal{P}$ und A die entsprechende Antwort. Dann ist A auch die Antwort von T auf h.*

Beweis. Sei $T = (M, \Pi, \{A_1, \ldots, A_n\})$ und zunächst $A =$ YES die Antwort von $\mathcal{P}$ auf die Anfrage h. Dann existiert ein auf $\mathcal{P}$ basierender dialektischer Baum $\mathcal{T}$ mit einem Argument $\langle \mathcal{A}, h \rangle$ als Wurzel und in $\mathcal{T}^*$ ist die Wurzel mit „U“ markiert. Sei $\mathcal{T}'$ aus $\mathcal{T}$ entstanden, indem alle Argumente mit ihren Teilbäumen aus $\mathcal{T}$ entfernt werden, die nicht maximal zurückhaltend bzgl. ihrer Geschwisterknoten sind. Nach Korollar 4.1 ist in $\mathcal{T}'^*$ das Wurzelargument $\langle \mathcal{A}, h \rangle$ mit „U“ markiert. Zeige per Induktion nach der Tiefe t von $\mathcal{T}'$, dass $\mathcal{T}'$ ein dialektischer Baum ist, der bei dem Argumentationsprozess von T auf die Anfrage h generiert wird.

- Induktionsanfang $t = 0$: In T existiert ein Agent A_j $(1 \leq j \leq n)$, dessen lokale Wissensbasis $\mathcal{A}$ ist. Somit existiert auch ein auf T basierender dialektischer Baum mit $\langle \mathcal{A}, h \rangle$ als Wurzel, da der Agent A_j auf die Anfrage h das Wurzelargument $\langle \mathcal{A}, h \rangle$ generiert.

- Induktionsschritt $t \to t + 1$: Sei $\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_t, h_t \rangle]$ ein Pfad in $\mathcal{T}'$ von der Wurzel bis zu einem Argument $\langle \mathcal{A}_t, h_t \rangle$. Sei K die Menge aller Kinder von $\langle \mathcal{A}_t, h_t \rangle$ in $\mathcal{T}'$. Für jedes $\langle \mathcal{B}, g \rangle \in K$ ist $\Lambda + \langle \mathcal{B}, g \rangle$ eine akzeptierte Argumentationsfolge in $\mathcal{T}'$. In T existiert ein Agent A_j $(1 \leq j \leq n)$, dessen lokale Wissensbasis $\mathcal{B}$ ist. Da $\langle \mathcal{B}, g \rangle$ maximal zurückhaltend bzgl. aller Angriffe auf $\langle \mathcal{A}_t, h_t \rangle$ ist, generiert der Agent A_j das Argument $\langle \mathcal{B}, g \rangle$ als gültigen Angriff auf $\langle \mathcal{A}_t, h_t \rangle$ und somit ist $\Lambda + \langle \mathcal{B}, g \rangle$ auch eine akzeptierte Argumentationsfolge bzgl. T.

$\mathcal{T}'$ ist also ein dialektischer Baum, der bei dem Argumentationsprozess von T auf die Anfrage h generiert wird. Da die Wurzel von $\mathcal{T}'^*$ ein Argument für h ist und mit „U“ markiert wird, ist YES die Antwort von T auf h. Für $A =$ NO verläuft der Beweis analog.

Sei $A =$ UNDECIDED, d. h. die Wurzelargumente aller markierten dialektischen Bäume, die auf die Anfrage h von $\mathcal{P}$ generiert werden, sind mit „D“ markiert. Angenommen T generiert auf die Anfrage h einen dialektischen Baum $\mathcal{T}$, dessen Wurzel in

$\mathcal{T}^*$ mit „U“ markiert ist. Sei o. B. d. A. $\langle \mathcal{A}, h \rangle$ die Wurzel von $\mathcal{T}$. Da $\langle \mathcal{A}, h \rangle$ auch ein gültiges Argument in $\mathcal{P}$ ist, existiert ein auf $\mathcal{P}$ basierender dialektischer Baum $\mathcal{T}'$ mit Wurzel $\langle \mathcal{A}, h \rangle$. Sei $\mathcal{T}''$ aus $\mathcal{T}'$ entstanden, indem alle Argumente, die nicht maximal zurückhaltend bzgl. ihrer Geschwisterknoten sind, mit ihren Teilbäumen aus $\mathcal{T}'$ entfernt werden. Mit derselben Argumentation wie oben ist $\mathcal{T}''$ ein dialektischer Baum, der beim Argumentationsprozess von T auf die Anfrage h generiert wird. Da es nicht zwei verschiedene dialektische Bäume mit derselben Wurzel geben kann, gilt $\mathcal{T}'' = \mathcal{T}$. Da $A = \text{UNDECIDED}$ ist, ist in $\mathcal{P}$ die Wurzel des Baumes $\mathcal{T}''^*$ mit „D“ markiert. Das steht im Widerspruch dazu, dass die Wurzel von $\mathcal{T}^*$ nach Annahme mit „U“ markiert ist. Also generiert T auf die Anfrage h nur markierte dialektische Bäume, deren Wurzeln mit „D“ markiert sind, und T antwortet auf h mit UNDECIDED.

Ist h nicht in der Sprache von $\mathcal{P}$, so ist h aufgrund Definition 8.2 auch nicht in der Sprache von T und somit antworten beide Systeme auf die Anfrage h mit UNKNOWN. □

Die triviale Überführung eines argumentationsfähigen Multiagentensystems in ein DeLP-Programm ist jedoch nicht ohne Einschränkung möglich.

Definition 8.3 (Induziertes DeLP-Programm)**.** Sei $T = (M, \Pi, \{A_1, \ldots, A_n\})$ ein argumentationsfähiges Multiagentensystem und seien $\Delta_1, \ldots, \Delta_n$ die lokalen Wissensbasen der Agenten $A_1, \ldots, A_n$. Dann ist das durch T *induzierte DeLP-Programm* $\mathcal{P}$ definiert durch:

$$\begin{aligned} \mathcal{P} &= (\Pi_G, \Pi_R, \Delta_R) \\ \Pi_G &= \{(h \leftarrow b_1, \ldots, b_n) | ((h \leftarrow b_1, \ldots, b_n), 1) \in \Pi\} \\ \Pi_R &= \{a | (a, 1) \in \Pi\} \\ \Delta_R &= \{r | (r, \alpha) \in \Delta_1 \cup \cdots \cup \Delta_n\} \end{aligned}$$

Das durch ein argumentationsfähiges Multiagentensystem induzierte DeLP-Programm weist nicht immer das gleiche Antwortverhalten wie das ursprüngliche DeLP-Programm auf:

Beispiel 8.1. Sei $T = (M, \Pi, \{A_1, A_2\})$ ein argumentationsfähiges Multiagentensystem und seien Δ_1 bzw. Δ_2 die lokalen Wissensbasen der Agenten A_1 bzw. A_2 mit

$$\begin{aligned} \Delta_1 &= \{(b \prec a), (b \prec a, c)\}, \\ \Delta_2 &= \{(\neg b \prec a), (c \prec d)\} \end{aligned}$$

sowie $\Pi = \{a, d\}$, wobei auf die Angabe von Metainformationen verzichtet wurde. Auf die Anfrage b generiert T die zwei in Abbildung 8.1 gezeigten dialektischem Bäume und gibt als Antwort UNDECIDED. Das durch T induzierte DeLP-Programm $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$

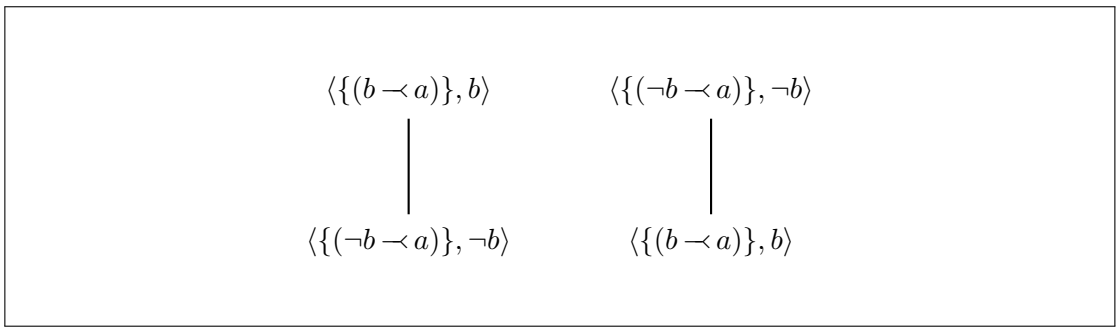

Abbildung 8.1: Von T generierte dialektische Bäume aus Beispiel 8.1

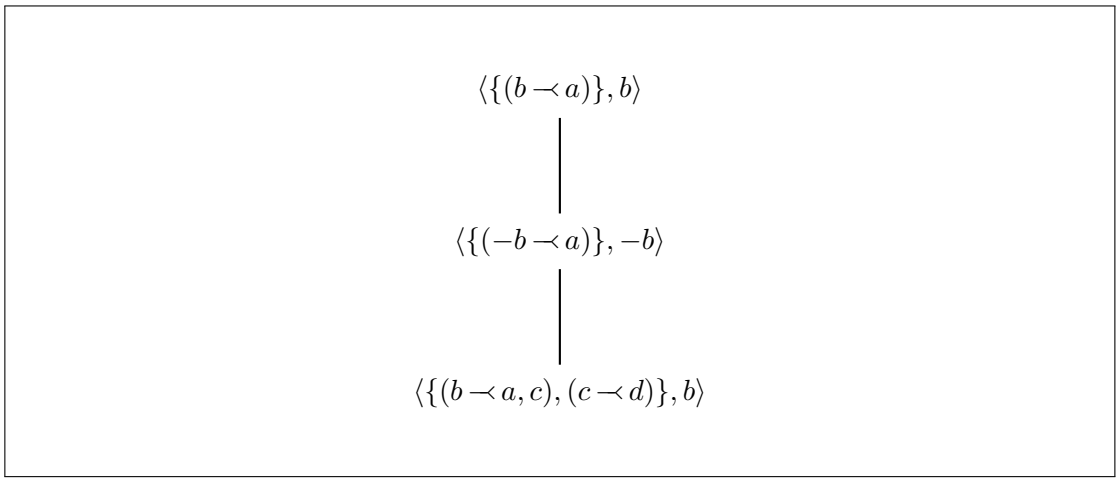

Abbildung 8.2: Ein von $\mathcal{P}$ generierter dialektischer Baum aus Beispiel 8.1

ist definiert durch

$$\begin{aligned} \Pi_G &= \{a, d\}, \\ \Pi_R &= \emptyset, \\ \Delta_R &= \{(b \prec a), (b \prec a, c), (\neg b \prec a), (c \prec d)\} \end{aligned}$$

und generiert auf die Anfrage b unter anderem den in Abbildung 8.2 gezeigten dialektischen Baum. Die Antwort von $\mathcal{P}$ auf die Anfrage b ist YES.

Der Grund für das unterschiedliche Antwortverhalten von DeLP und dem argumentationsfähigen Multiagentensystem in Beispiel 8.1 liegt an der in dem Multiagentensystem „deplatzierten“ Regel $c \prec d$, für die der Agent A_2 in dem Ursprungssystem keine Verwendung hat. Wird die Menge der Multiagentensysteme allerdings auf die Systeme eingeschränkt, in denen keine Regel „deplatziert“ ist, so ist eine Überführung in ein DeLP-Programm unter Wahrung des Antwortverhaltens möglich.

Definition 8.4 (Wohlgeformtes argumentationsfähiges Multiagentensystem)**.** Sei

$$T = (M, \Pi, \{A_1, \ldots, A_n\})$$

ein argumentationsfähiges Multiagentensystem, seien $\Delta_1, \ldots, \Delta_n$ die lokalen Wissensbasen der Agenten $A_1, \ldots, A_n$ und $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ das durch T induzierte DeLP-Programm. T heißt *wohlgeformtes argumentationsfähiges Multiagentensystem* bzgl. $\mathcal{P}$, g. d. w. für jedes Argument $\langle \mathcal{A}, h \rangle$ aus $\mathcal{P}$ ein i $(1 \leq i \leq n)$ existiert mit $\mathcal{A} \subseteq \Delta_i$.

Theorem 8.2. *Sei $T = (M, \Pi, \{A_1, \ldots, A_n\})$ ein wohlgeformtes argumentationsfähiges Multiagentensystem, seien $\Delta_1, \ldots, \Delta_n$ die lokalen Wissensbasen der Agenten $A_1, \ldots, A_n$ und sei $\mathcal{P} = (\Pi_G, \Pi_R, \Delta_R)$ das durch T induzierte DeLP-Programm. Sei h eine Anfrage an T und A die entsprechende Antwort. Dann ist A auch die Antwort von $\mathcal{P}$ auf h.*

Beweis. Seien D_1 bzw. D_2 die Mengen der dialektischen Bäume, die T bzw. $\mathcal{P}$ auf die Anfrage h generieren. Zeige zunächst $|D_1| = |D_2|$. Sei W die Menge der Wurzeln aller dialektischen Bäume aus D_1, d. h. jedes Argument aus W ist ein Argument für h oder $\overline{h}$. In $\mathcal{P}$ gibt es laut Definition eines wohlgeformten argumentationsfähigen Multiagentensystems keine Argumente für h oder $\overline{h}$, die nicht schon in W liegen. Somit ist W auch die Menge der Wurzeln aller dialektischen Bäume aus D_2 und es ist $|D_1| = |D_2|$. Seien $\langle \mathcal{A}, g \rangle \in W$ und $\mathcal{T}_1 \in D_1$ bzw. $\mathcal{T}_2 \in D_2$ die dialektischen Bäume, die T bzw. $\mathcal{P}$ zu der Wurzel $\langle \mathcal{A}, g \rangle$ generieren. Sei $\mathcal{T}_2'$ aus $\mathcal{T}_2$ entstanden, indem alle Argumente mit ihren Teilbäumen aus $\mathcal{T}_2$ entfernt wurden, die nicht maximal zurückhaltend bzgl. ihrer Geschwisterknoten sind. Nach Korollar 4.1 ist die Wurzelmarkierung von $\mathcal{T}_2'^*$ identisch mit der Wurzelmarkierung von $\mathcal{T}_2^*$. Zeige nun per Induktion nach der Tiefe t von $\mathcal{T}_1$, dass $\mathcal{T}_1 = \mathcal{T}_2'$ gilt:

- Induktionsanfang $t = 0$: Die Wurzel von $\mathcal{T}_1$ ist nach Voraussetzung gleich der Wurzel von $\mathcal{T}_2'$.
- Induktionsschritt $t \to t + 1$: Sei $\Lambda = [\langle \mathcal{A}_1, h_1 \rangle, \ldots, \langle \mathcal{A}_t, h_t \rangle]$ ein Pfad in $\mathcal{T}_1$ von der Wurzel bis zu einem Argument $\langle \mathcal{A}_t, h_t \rangle$. Sei K die Menge aller Kinder von $\langle \mathcal{A}_t, h_t \rangle$ in $\mathcal{T}_1$. Für jedes $\langle \mathcal{B}, g \rangle \in K$ ist $\Lambda + \langle \mathcal{B}, g \rangle$ eine akzeptierte Argumentationsfolge in $\mathcal{T}_1$. Jedes $\langle \mathcal{B}, g \rangle \in K$ ist auch ein gültiges Argument in $\mathcal{P}$ und somit ist $\Lambda + \langle \mathcal{B}, g \rangle$ auch eine akzeptierte Argumentationsfolge in $\mathcal{T}_2'$. Angenommen es existiert ein weiterer Angriff $\langle \mathcal{B}', g' \rangle$ auf $\langle \mathcal{A}_t, h_t \rangle$ in $\mathcal{T}_2'$ mit $\langle \mathcal{B}', g' \rangle \notin K$. Nach Definition eines wohlgeformten argumentationsfähigen Multiagentensystems ist $\mathcal{B}' \subseteq \Delta_k$ für ein $k \in \{1, \ldots, n\}$. Dann wäre auch $\langle \mathcal{B}', g' \rangle$ ein akzeptierter Angriff des Agenten A_k auf $\langle \mathcal{A}_t, h_t \rangle$ und somit $\langle \mathcal{B}', g' \rangle \in K$. Das steht im Widerspruch zur Annahme und somit ist die Menge der Angriffe auf $\langle \mathcal{A}_t, h_t \rangle$ in $\mathcal{T}_1$ identisch mit der Menge der Angriffe auf $\langle \mathcal{A}_t, h_t \rangle$ in $\mathcal{T}_2'$.

Es folgt $\mathcal{T}_1 = \mathcal{T}_2'$. Da die Markierung der Wurzelargumente aller dialektischen Bäume in D_1 zu denen von D_2' und somit zu D_2 identisch ist, ist auch ist das Antwortverhalten von DeLP und D-DeLP gleich. □

Wegen der Theoreme 8.1 und 8.2 liegt eine (Beinahe-)Äquivalenzbeziehung zwischen DeLP-Programmen und argumentationsfähigen Multiagentensystemen vor. Auch wenn bei der Generierung von Gegenargumenten in argumentationsfähigen Multiagentensystemen nur die maximal zurückhaltenden Gegenargumente berücksichtigt werden, ist wegen Korrolar 4.1 ein identisches Antwortverhalten beider Systeme gewährleistet.

8.2 Vergleich von DAS mit dem *DeLP-Server*

Anmerkung. Der *DeLP-Server* wird an der Universidad Nacional del Sur in Bahía Blanca (Argentinien) von Alejandro J. Garcia, Nicolás Rotstein, Mariano Tucat und Guillermo R. Simari entwickelt. Er befindet sich derzeit noch in der Testphase und ist nicht für die Allgemeinheit verfügbar. Aus diesem Grund werde ich auch nur einen kurzen und oberflächlichen Vergleich von DAS mit dem DeLP-Server führen. Ich beziehe mich in diesem Abschnitt auf die Version 0.1.9 des DeLP-Servers, die mir freundlicherweise von Alejandro J. Garcia[1] und Guillermo R. Simari[2] zu Forschungszwecken zur Verfügung gestellt wurde.

Der DeLP-Server ist eine Implementierung von DeLP in PROLOG und untersützt ebenso wie DAS verteilte Argumentation. Die Art und Weise, wie beim DeLP-Server verteilte Argumentation definiert ist, unterscheidet sich jedoch von der in DAS.

Der DeLP-Server erlaubt die Spezifikation einer globalen Wissensbasis, die Fakten, sichere und unsichere Regeln enthalten darf. Client-Anwendungen können Anfragen an den Server stellen, die von dem DeLP-Server lokal durch einen Argumentationsprozess bearbeitet werden. Anschließend gibt dieser dem Client die Auswertung der Anfrage in Form einer Aussage über Akzeptanz oder Widerlegung der Anfrage sowie eine Menge dialektischer Bäume zurück. Der DeLP-Server unterstützt allerdings auch Anfragen von Clients, die neben der eigentlichen logischen Anfrage auch eine Menge zusätzlicher Fakten, sicherer und unsicherer Regeln enthält. Der DeLP-Server bildet lokal die Vereinigung der globalen und der mitgelieferten „lokalen“ Wissensbasis des Clients und führt den Argumentationsprozess auf dieser Vereinigung aus.

Der Unterschied zwischen dem DeLP-Server und DAS liegt vor allem darin, dass bei dem DeLP-Server keine Interaktion zwischen den Clients stattfindet. Während bei DAS Agenten (Clients) auf Argumente anderer Agenten antworten können, findet zwischen den Clients des DeLP-Servers kein wechselseitiger Informationsaustausch statt. Der DeLP-Server übt also keine Koordination aus, sondern dient einzig der Beantwortung von Anfragen. Logikbasierte Argumentation wird somit bei dem DeLP-Server als Deliberationsmechanismus angewandt und nicht als ein Mechanismus, um „Meinungen“ verschiedener Agenten miteinander zu vergleichen.

[1] ajg@cs.uns.edu.ar
[2] grs@cs.uns.edu.ar

9 Ausblick und Fazit

In diesem Kapitel wird ein Ausblick auf mögliche Erweiterungen von DAS gegeben und mit einem anschließenden Fazit die Arbeit geschlossen.

9.1 Ausblick

9.1.1 Unterstützung von Annahmen

Eine unsichere Regel ohne Rumpfliterale heißt *Annahme* (*Presumption*, [Nut88]). Im DeLP-Framework [GS02] drückt eine Regel „$a \prec$“ eine Annahme in der Form „es existieren unsichere bzw. revidierbare Gründe dafür, a zu glauben“ aus. [GS02] beschreibt eine formale Erweiterung des DeLP-Frameworks um Annahmen, die direkt als Erweiterung von D-DeLP und somit auch als Erweiterung des Konzepts argumentationsfähiger Multiagentensysteme benutzt werden kann. Wie jedoch in [GS02] beschrieben, ergeben sich Schwierigkeiten beim Vergleich von Argumenten mit *Generalized Specificity*, wenn Annahmen unterstützt werden sollen. Weitere Forschung bei der Unterstützung von Annahmen in DeLP führt deshalb auch zu neuen Erkenntnissen bei der Unterstützung von Annahmen in D-DeLP.

9.1.2 Pruning der dialektischen Bäume

Wie in [SCG94] beschrieben, ist es nicht immer nötig, zu einem Argument $\langle \mathcal{A}, h \rangle$ den vollständigen dialektischen Baum zu berechnen. Gibt es einen Angriff auf $\langle \mathcal{A}, h \rangle$, der nicht widerlegt werden kann, ist eine Betrachtung weiterer Angriffe auf $\langle \mathcal{A}, h \rangle$ nicht mehr nötig, da $\langle \mathcal{A}, h \rangle$ in jedem Fall widerlegt wird.

Beispiel 9.1 ([SCG94], Beispiel 3.1)**.** Sei der markierte dialektische Baum aus Abbildung 9.1 gegeben (auf die Angabe der einzelnen Argumente wurde übersichtshalber verzichtet). Die mit „*“ markierten Argumente brauchen für die Bestimmung der Markierung des Wurzelarguments nicht berücksichtigt werden, weil das Wurzelargument in jedem Fall mit „D“ markiert wird.

Auf diese Weise kann auf die Generierung vieler Argumente und die damit einhergehenden Prüfungen zur Akzeptanz der entstehenden Argumentationsfolgen verzichtet werden. Das Antwortverhalten des Systems bleibt dabei erhalten, vgl. [SCG94].

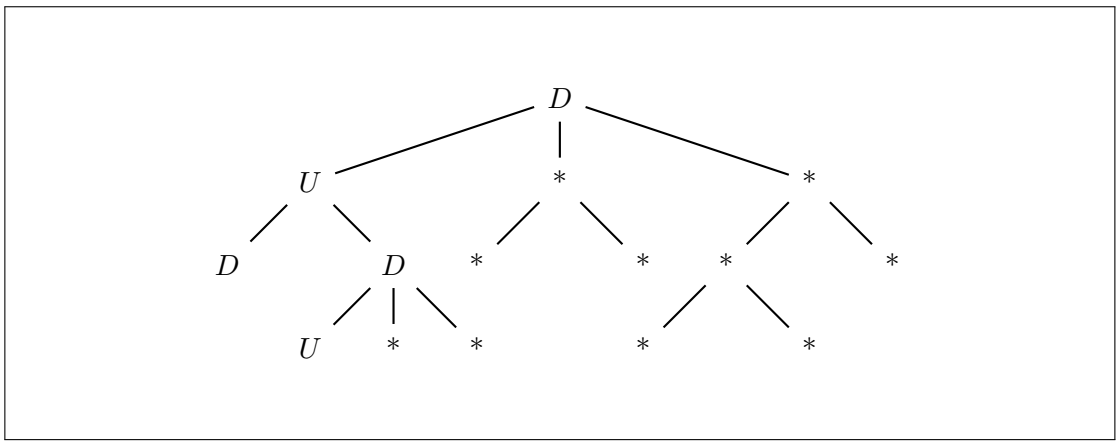

Abbildung 9.1: Markierter dialektischer Baum zu Beispiel 9.1

9.1.3 Effizienzsteigerung durch Anlegen von Archiven

Ein großer Nachteil der derzeitigen Implementierung von DAS ist der Rechenaufwand zur Generierung von Argumenten und vor allem zur Bestimmung von Argumentvervollständigungen. Hier bietet sich allerdings die Möglichkeit der Verbesserung durch das Anlegen von Argument-Archiven, wie in [CCnS04, CnSG93] beschrieben. Ein Agent könnte auf Grundlage seiner lokalen Wissensbasis schon im Vorfeld alle möglichen Argumente generieren und sie in einem Archiv für die spätere Verwendung speichern. Dadurch wird die mehrfache Generierung eines Arguments vermieden. Auf dieselbe Weise können auch Argumentvervollständigungen zu jedem Argument schon im Vorfeld einmalig berechnet werden. Für eine genaue Beschreibung des Verfahrens siehe beispielsweise [CCnS04].

Die Implementierung solcher Mechanismen ist nicht allzu schwierig und sollte bei einer Erweiterung des Systems angedacht werden.

9.2 Fazit

Die Realisierung von verteilter Argumentation eröffnet eine alternative Sicht auf die logikbasierte Argumentation in der KI. Während bisher logikbasierte Argumentation als Deliberationsmechanismus zur Erzeugung einer konsistenten Wissensmenge benutzt wurde, werden durch verteilte Argumentation neue Anwendungsgebiete erreicht. Ein Rechtsstreit wurde als praktisches Beispiel für ein argumentationsfähiges Multiagentensystem aufbereitet und verdeutlicht die Funktionsweise eines verteilten Argumentationsprozesses. Dabei übernehmen zwei Agenten die Rollen von Für- und Gegensprecher und tauschen Argumente in einem Rechtsstreit aus. Auch wenn die Repräsentation des Rechtsbeispiels keinen Anspruch auf formale Korrektheit im Sinne der allgemeinen deutschen Rechtsprechung erhebt, zeigt es wie verteilte Argumentation zur Anwendung kommen kann.

Die Aufgabe dieser Diplomarbeit – die Konzipierung und Realisierung eines verteilten Argumentationsmechanismus – wurde durch die Entwicklung der Sprache D-DeLP und der Implementierung des Distributed Argumentation System gelöst. Auch wenn eine Adaptierung des Algorithmus von Besnard und Hunter zur Generierung gültiger Argumentationsfolgen wünschenswert war, wurde u. a. aus Effizienzgründen darauf verzichtet. Allerdings ist die Argumentationsgenerierung auch nur eine Komponente im Konzept verteilter Argumentation und könnte später durch andere Mechanismen ausgetauscht werden.

Da D-DeLP und das Konzept verteilter Argumentation auf DeLP aufbaut, führt Forschung im Bereich von DeLP auch zu Erkenntnissen, die für D-DeLP genutzt werden können. Auf Grund der Resultate aus Abschnitt 8.1 läßt sich ein verteiltes argumentationsfähiges System auf ein DeLP-Programm zurückführen und erlaubt somit eine Einbettung in die bestehende Theorie um DeLP.

A Beispiel „Antiquariat“

In diesem Anhang findet sich eine vollständige formale Wissensrepräsentation des Beispiels aus Abschnitt 6.1 für die Verwendung in DAS. Auch hier gilt die folgende Anmerkung:

Anmerkung. Die Wissensrepräsentation des Rechtsfalls dient nur als Beispiel für die Verwendung eines argumentationsfähigen Multiagentensystem. Aus diesem Grund wird der Fall sehr vereinfacht dargestellt und erhebt keinen Anspruch auf formale Korrektheit im Sinne der allgemeinen deutschen Rechtsprechung. Unter anderem sind rechtswissenschaftliche Fachbegriffe, wie „Bevollmächtiger“, „Vertretungsmacht“ und „Einverständnis“, in ihrer Bedeutung leicht verschieden zu ihrer Bedeutung im Rechtswesen. Diese Anpassungen und Vereinfachungen sind nötig, um ein illustratives Beispiel für ein argumentationsfähiges Multiagentensystem zu erhalten.

Zur Realisierung werden zwei Agenten `Proponent` und `Opponent` verwendet, die gegen bzw. für die Zahlung der 600 Euro durch A argumentieren. Das in dem Szenario enthaltene Wissen teilt sich also in das sichere Wissen aller Agenten und die lokalen (unsicheren) Wissensbasen der Agenten `Proponent` und `Opponent` auf. Eine Auflistung der verwendeten Rechtsnormen findet sich in Anhang B.

A.1 Sicheres Wissen

Die sicheren Informationen des in dem Szenario enthaltenen Wissens sind repräsentiert durch:

```
Unmoeglichkeit.
-Lieferung.
-Selbsterklaerung.
Eigene_WE_in_fremdem_Namen.
Innenvollmacht.
-Nichteinhaltung_der_Grenzen.
Aufforderung_Genehmigung.
Genehmigung.
-Ausschluss_der_Zufallshaftung.
Verzug.
Verzugs_Verschuldung.
Schuld_des_Unfallgegners.
Kalendergenaue_Terminierung.
Durchsetzbar.
Faellig.
Nichtleistung.
-Einverstaendnis.
-Bei_Gelegenheit.
```

Listing A.1: Das sichere Wissen im Rechtsbeispiel

Zur Erläuterung:

`Unmoeglichkeit.`
A ist es unmöglich, die erforderte Leistung zu erbringen, da die Skulptur zerstört wurde.

`-Lieferung.`
A hat die Lieferung der Skulptur nicht erbracht.

`-Selbsterklaerung.`
D hat keine eigene Erklärung für den Kauf der Skulptur gegeben.

`Eigene_WE_in_fremdem_Namen.`
F hat seine Willenserklärung an A im Namen von D gegeben und das auch entäußert.

`Innenvollmacht.`
F hat von D eine Innenvollmacht zur Abwicklung des Kaufgeschäfts erhalten.

`-Einhaltung_der_Grenzen.`
F hat die Grenzen der ihm verliehenen Innenvollmacht nicht eingehalten, da er die Skulptur zu einem höheren Preis als von D gefordert, erworben hat.

`-Einverstaendnis.`
D hat F zu dem Kauf der Skulptur zum erhöhten Preis nicht sein Einverständnis gegeben.

`Aufforderung_Genehmigung.`
A hat D per Telefon dazu aufgefordert, den von F in Ds Namen abgewickelten Kauf zu genehmigen.

`Genehmigung.`
D hat A die Genehmigung zum Kauf durch F am Telefon gegeben.

`-Ausschluss_der_Zufallshaftung.`
Ein Ausschluss der Zufallshaftung nach § 287 S. 2 Hs. 2 BGB liegt nicht vor, d. h. es kann nicht bewiesen werden, dass der Unfall, durch den die Skulptur zerstört wurde, auch bei termingerechter Lieferung geschehen wäre.

`Verzug.`
A lag bei der Lieferung der Skulptur an D im Verzug.

`Verzugs_Verschuldung.`
Der Verzug der Lieferung war von A selbstverschuldet, da A die Lieferung vergessen hatte.

`-Bei_Gelegenheit.`
Die Lieferung der Skulptur von A nach D durch H war eine verbindliche Auftragserteilung und sollte nicht nur „bei Gelegenheit“ erbracht werden.

`Kalendergenaue_Terminierung.`
Die Lieferung der Skulptur von A nach D wurde kalendergenau auf den 30.11. terminiert.

`Durchsetzbar.`
Die Lieferung der Skulptur war zum 30. November durchsetzbar.

`Faellig.`
Die Lieferung der Skulptur war zum 30. November fällig.

`Schuld_des_Unfallgegners.`
Der Unfall, bei dem die Skulptur zerstört wurde, war von dem Gegner von H verschuldet.

A.2 Wissen des Agenten `Proponent`

Der Agent `Proponent` ist der Fürsprecher von A und argumentiert also gegen die Zahlung der 600 Euro. Das unsichere Wissen des Agenten `Proponent` ist gegeben durch

```
-KV -: -P433.
-P433 -: -Selbsterklaerung.
-P433 -: -Selbsterklaerung, -Bevollmaechtigter,
                              Verweigerung_Vertrag.
-Vertretungsmacht -: -P167.
-P167 -: Innenvollmacht, -Nichteinhaltung_der_Grenzen.
-Bevollmaechtigter -: -Vertretungsmacht.
Verweigerung_Vertrag -: -Einverstaendnis.
-P280_1 -: Pflichtverletzung, -P276.
-P276 -: -Eigenes_Verschulden.
-Eigenes_Verschulden -: Schuld_des_Unfallgegners.
-Verschulden_des_Dritten -: Schuld_des_Unfallgegners.
-P278 -: KV, Pflichtverletzung, -Bei_Gelegenheit,
                             -Verschulden_des_Dritten.
-Ersatz_statt_Leistung -: -KV.
-Ersatz_statt_Leistung -: P280_1, -P276.
```

Listing A.2: Das unsichere Wissen des Agenten Proponent im Rechtsbeispiel

Zur Erläuterung:

`-KV -: -P433.`
§ 433 BGB regelt vertragstypische Pflichten beim Kaufvertrag. Findet er keine Anwendung, so ist kein Kaufvertrag zustande gekommen.

`-P433 -: -Selbsterklaerung.`
D hat zunächst keine Selbsterklärung zum Kauf der Skulptur gegeben und somit findet § 433 BGB keine Anwendung.

```
-P433 -:  -Selbsterklaerung, -Bevollmaechtigter,
          Verweigerung_Vertrag.
```
D hat keine Selbsterklärung für den Kauf der Skulptur gegeben, F war nicht bevollmächtigt den Kauf im Namen von D zu tätigen und D hat den Kaufvertrag zunächst verweigert. Somit findet § 433 BGB keine Anwendung.

`-Vertretungsmacht -: -P167.`
§ 167 BGB regelt Befugnisse Dritter und wenn er keine Anwendung findet, hat F keine Vertretungsmacht für D gegenüber A.

`-P167 -: Innenvollmacht, -Einhaltung_der_Grenzen.`
§ 167 BGB findet keine Anwendung, wenn einem Dritten zwar Innenvollmacht er-

teilt wurde, er aber nicht innerhalb der Grenzen seiner Befugnisse vorgegangen ist.

`-Bevollmaechtigter -: -Vertretungsmacht.`
Hat F keine Vertretungsmacht, so ist er nicht bevollmächtigt im Namen von D zu handeln.

`Verweigerung_Vertrag -: -Einverstaendnis.`
D hat zunächst den Vertrag implizit verweigert, da er F gegenüber nicht sein Einverständnis für den überteuerten Kauf gegeben hat.

`-P280_1 -: Pflichtverletzung, -P276.`
§ 280(1) BGB zur Leistung von Schadensersatz findet keine Anwendung, obwohl eine Pflichtverletzung vorliegt, wenn die Leistung nicht nach § 276 BGB vertreten werden muss.

`-P276 -: -Eigenes_Verschulden.`
Der Unfall, bei dem die Skulptur zerstört wurde, war nicht von A verschuldet und somit muss die Pflichtverletzung nicht vertreten werden.

`-Eigenes_Verschulden -: Schuld_des_Unfallgegners.`
Der Unfall, bei dem die Skulptur zerstört wurde, war von dem Gegner von H verschuldet und somit liegt die Schuld nicht bei A.

`-Verschulden_des_Dritten -: Schuld_des_Unfallgegners.`
Der Unfall, bei dem die Skulptur zerstört wurde, war von dem Gegner von H verschuldet und somit liegt die Schuld nicht bei H.

```
-P278 -:  KV, Pflichtverletzung,
          -Bei_Gelegenheit, -Verschulden_des_Dritten.
```
§ 278 BGB zur Regelung von Verantwortlichkeiten des Schuldners für Dritte findet keine Anwendung, obwohl Kaufvertrag, Pflichtverletzung und erteilter Auftrag vorliegen, wenn der Dritte das Nichterbringen der Leistung nicht verschuldet hat.

`-Ersatz_statt_Leistung -: -KV.`
Besteht kein Kaufvertrag, so muss kein Ersatz statt der Leistung gegeben werden.

`-Ersatz_statt_Leistung -: P280_1, -P276.`
Schadensersatz muss nicht erbracht werden, obwohl eine Pflichtverletzung vorliegt, wenn diese nicht zu vertreten ist.

A.3 Wissen des Agenten `Opponent`

Der Agent `Opponent` ist der Gegensprecher von A und argumentiert also für die Zahlung der 600 Euro. Das unsichere Wissen des Agenten `Opponent` ist gegeben durch:

```
KV -: P433.
P433 -: -Selbsterklaerung, Bevollmaechtigter.
Bevollmaechtigter -: P164.
P164 -: Eigene_WE_in_fremdem_Namen, Vertretungsmacht.
Vertretungsmacht -: P167.
P167 -: Innenvollmacht.
P167 -: Aussenvollmacht.
-Verweigerung_Vertrag -: -Einverstaendnis, Aufforderung_Genehmigung.
P433 -: -Selbsterklaerung, Aufforderung_Genehmigung, Genehmigung.
Pflichtverletzung -: -Lieferung.
P280_1 -: Pflichtverletzung.
P280_3 -: P281.
P280_3 -: P282.
P280_3 -: P283.
P283 -: P275.
P275 -: Unmoeglichkeit.
P275 -: Unvertretbarkeit.
P275 -: Unzumutbarkeit.
P276 -: Eigenes_Verschulden.
P276 -: -Eigenes_Verschulden, P278.
P276 -: -Eigenes_Verschulden, P287.
-Eigenes_Verschulden -: Schuld_des_Unfallgegners.
P278 -: KV, Pflichtverletzung, -Bei_Gelegenheit.
P278 -: KV, Pflichtverletzung, -Bei_Gelegenheit,
                                Verschulden_des_Dritten.
Verschulden_des_Dritten -: Fahrlaessigkeit.
Verschulden_des_Dritten -: Vorsatz.
P287 -: P286, Unmoeglichkeit_im_Verzug,
                               -Ausschluss_der_Zufallshaftung.
Unmoeglichkeit_im_Verzug -: Unmoeglichkeit, Verzug.
P286 -: Leistungsverzoegerung, Kalendergenaue_Terminierung,
                                Verzugs_Verschuldung, Verzug.
Leistungsverzoegerung -: Durchsetzbar, Faellig, -Lieferung.
Ersatz_statt_Leistung -: KV, P280_3, Pflichtverletzung, P276.
Ersatz_statt_Leistung -: P280_1.
```

Listing A.3: Das unsichere Wissen des Agenten Opponent im Rechtsbeispiel

Zur Erläuterung:

`KV -: P433.`
Findet § 433 BGB zur Regelung vertragstypischer Pflichten beim Kaufvertrag Anwendung, so ist ein Kaufvertrag zustande gekommen.

`P433 -: -Selbsterklaerung, Bevollmaechtigter.`
§ 433 BGB findet Anwendung, obwohl keine Selbsterklärung von D gegeben wurde, wenn F als bevollmächtiger Dritter in Ds Namen gehandelt hat.

`Bevollmaechtigter -: P164.`
Findet § 164 BGB zur Erklärung eines rechtmäßigen Vertreters Anwendung, so ist F bevollmächtigt in Ds Namen zu handeln.

`P164 -: Eigene_WE_in_fremdem_Namen, Vertretungsmacht.`
§ 164 BGB findet Anwendung, wenn F dem A gegenüber eine Willenserklärung über eine Kaufabsicht des D ausspricht und F die entsprechende Vertretungsmacht besitzt.

`Vertretungsmacht -: P167.`
Findet § 167 BGB zur Regelung von Befugnissen Dritter Anwendung, so hat F Vertretungsmacht.

`P167 -: Innenvollmacht.`
§ 167 BGB findet Anwendung, wenn F durch D direkt bevollmächtigt wurde.

`P167 -: Aussenvollmacht.`
§ 167 BGB findet Anwendung, wenn D den F indirekt bevollmächtigt, indem er A gegenüber erklärt, dass F in seinem Namen handeln darf.

```
-Verweigerung_Vertrag -:  -Einverstaendnis,
                          Aufforderung_Genehmigung.
```
Ein Vertrag gilt nicht als verweigert, obwohl keine Einverständniserklärung abgegeben wurde, wenn eine Aufforderung zur Genehmigung vorliegt.

```
P433 -:  -Selbsterklaerung, Aufforderung_Genehmigung,
         Genehmigung.
```
§ 433 BGB findet Anwendung, obwohl keine Selbsterklärung von D gegeben wurde, wenn eine Aufforderung zur Genehmigung von A erteilt und die Genehmigung von D gegeben wurde.

`Pflichtverletzung -: -Lieferung.`
Eine Pflichtverletzung liegt vor, wenn die zu erbringende Lieferung nicht erbracht wurde.

`P280_1 -: Pflichtverletzung.`
Liegt eine Pflichtverletzung vor, so ist Schadensersatz statt Leistung nach § 280(1) BGB zu erbringen.

`P280_3 -: P281.`
Schadensersatz statt Leistung nach § 280(3) BGB kann unter den Voraussetzungen des § 281 BGB von D verlangt werden.

`P280_3 -: P282.`
Schadensersatz statt Leistung nach § 280(3) BGB kann unter den Voraussetzungen des § 282 BGB von D verlangt werden.

`P280_3 -: P283.`
Schadensersatz statt Leistung nach § 280(3) BGB kann unter den Voraussetzungen des § 283 BGB von D verlangt werden.

`P283 -: P275.`
Schadensersatz statt Leistung kann von D verlangt werden, wenn die eigentliche Leistung nach § 275 BGB nicht erbracht werden kann.

`P275 -: Unmoeglichkeit.`
Eine Leistung kann nach § 275 BGB nicht erbracht werden, wenn dies unmöglich ist.

`P275 -: Unvertretbarkeit.`
Eine Leistung kann nach § 275 BGB nicht erbracht werden, wenn dies nicht zu vertreten ist.

`P275 -: Unzumutbarkeit.`
Eine Leistung kann nach § 275 BGB nicht erbracht werden, wenn dies unzumutbar ist.

`P276 -: Eigenes_Verschulden.`
Die Pflichtverletzung der Lieferung muss vertreten werden, wenn die Pflichtverletzung durch Verschulden von A zustande gekommen ist.

`P276 -: -Eigenes_Verschulden, P278.`
Die Verletzung der Lieferungspflicht muss vertreten werden, obwohl die Pflichtverletzung nicht durch Verschulden von A zustande gekommen ist, wenn § 278 BGB zur Verantwortlichkeit von A für Hs Verschulden Anwendung findet.

`P276 -: -Eigenes_Verschulden, P287.`
Die Verletzung der Lieferungspflicht muss vertreten werden, obwohl die Pflichtverletzung nicht durch Verschulden von A zustande gekommen ist, wenn die Pflichtverletzung nach § 287 BGB unter Verzug stattgefunden hat.

```
-Eigenes_Verschulden -: Schuld_des_Unfallgegners.
```
Der Unfall, bei dem die Skulptur zerstört wurde, war von dem Gegner von H verschuldet und somit liegt die Schuld nicht bei A.

```
P278 -: KV, Pflichtverletzung, -Bei_Gelegenheit.
```
§ 278 BGB zur Verantwortlichkeit von A für Hs Verschulden findet Anwendung, wenn Kaufvertrag und Pflichtverletzung vorliegen sowie die Lieferung der Skulptur nicht „bei Gelegenheit“ von H erbracht werden sollte.

```
P278 -:  KV, Pflichtverletzung,
         -Bei_Gelegenheit, Verschulden_des_Dritten.
```
§ 278 BGB zur Verantwortlichkeit von A für Hs Verschulden findet Anwendung, wenn Kaufvertrag und Pflichtverletzung vorliegen, die Lieferung der Skulptur nicht „bei Gelegenheit“ von H erbracht werden sollte und H den Unfall, bei dem das Bild zerstört wurde, zu verschulden hat.

```
Verschulden_des_Dritten -: Fahrlaessigkeit.
```
A hat Hs Verschulden zu vertreten, wenn H den Unfall wegen Fahrlässigkeit zu verschulden hat.

```
Verschulden_des_Dritten -: Vorsatz.
```
A hat Hs Verschulden zu vertreten, wenn H den Unfall wegen Vorsatz zu verschulden hat.

```
P287 -:  P286, Unmoeglichkeit_im_Verzug,
         -Ausschluss_der_Zufallshaftung.
```
A hat nach § 287 BGB die Pflicht eine Leistung zu vertreten, wenn diese im Verzug wegen Unmöglichkeit nicht geleistet werden kann und nicht bewiesen werden kann, dass die Pflichtverletzung auch bei rechtzeitiger Leistung aufgetreten wäre.

```
Unmoeglichkeit_im_Verzug -: Unmoeglichkeit, Verzug.
```
Kann eine Leistung im Verzug wegen Unmöglichkeit nicht erbracht werden, findet entsprechend § 287 BGB Anwendung.

```
P286 -:  Leistungsverzoegerung, Kalendergenaue_Terminierung,
         Verzugs_Verschuldung, Verzug.
```
§ 286 BGB zur Regelung des Verzugs findet Anwendung, wenn eine Leistungsverzögerung vorliegt, die Leistung kalendergenau terminiert wurde und die Verschuldung im Verzug passiert.

```
Leistungsverzoegerung -: Durchsetzbar, Faellig, -Lieferung.
```
Eine Leistungsverzögerung liegt vor, wenn eine Leistung zu einem bestimmten Zeitpunkt durchsetzbar und fällig ist, diese aber nicht geleistet wird.

`Ersatz_statt_Leistung -: KV, P280_3, Pflichtverletzung, P276.`
Ersatz statt Leistung kann D verlangen, wenn ein Kaufvertrag zustande gekommen ist, § 280(3) BGB wegen einer Pflichtverletzung Anwendung findet und A die Pflichtverletzung nach § 278 BGB vertreten muss.

`Ersatz_statt_Leistung -: P280_1.`
Ersatz statt Leistung kann D verlangen, wenn A eine Pflicht aus dem Schuldverhältnis nicht erbracht hat.

B Auszug aus dem BGB

Es folgt nun ein Auszug aus dem Bürgerlichen Gesetzbuch (BGB) [BGB07], der die in dieser Arbeit verwendeten Rechtsnormen beinhaltet:

§ 164 Wirkung der Erklärung des Vertreters

(1) Eine Willenserklärung, die jemand innerhalb der ihm zustehenden Vertretungsmacht im Namen des Vertretenen abgibt, wirkt unmittelbar für und gegen den Vertretenen. Es macht keinen Unterschied, ob die Erklärung ausdrücklich im Namen des Vertretenen erfolgt oder ob die Umstände ergeben, dass sie in dessen Namen erfolgen soll.

(2) Tritt der Wille, in fremdem Namen zu handeln, nicht erkennbar hervor, so kommt der Mangel des Willens, im eigenen Namen zu handeln, nicht in Betracht.

(3) Die Vorschriften des Absatzes 1 finden entsprechende Anwendung, wenn eine gegenüber einem anderen abzugebende Willenserklärung dessen Vertreter gegenüber erfolgt.

§ 167 Erteilung der Vollmacht

(1) Die Erteilung der Vollmacht erfolgt durch Erklärung gegenüber dem zu Bevollmächtigenden oder dem Dritten, dem gegenüber die Vertretung stattfinden soll.

(2) Die Erklärung bedarf nicht der Form, welche für das Rechtsgeschäft bestimmt ist, auf das sich die Vollmacht bezieht.

§ 275 Ausschluss der Leistungspflicht

(1) Der Anspruch auf Leistung ist ausgeschlossen, soweit diese für den Schuldner oder für jedermann unmöglich ist.

(2) Der Schuldner kann die Leistung verweigern, soweit diese einen Aufwand erfordert, der unter Beachtung des Inhalts des Schuldverhältnisses und der Gebote von Treu und Glauben in einem groben Missverhältnis zu dem Leistungsinteresse des Gläubigers steht. Bei der Bestimmung der dem Schuldner zuzumutenden Anstrengungen ist auch zu berücksichtigen, ob der Schuldner das Leistungshindernis zu vertreten hat.

(3) Der Schuldner kann die Leistung ferner verweigern, wenn er die Leistung persönlich zu erbringen hat und sie ihm unter Abwägung des seiner Leistung entgegenstehenden Hindernisses mit dem Leistungsinteresse des Gläubigers nicht zugemutet werden kann.

(4) Die Rechte des Gläubigers bestimmen sich nach den §§280, 283 bis 285, 311a und 326.

§ 276 Verantwortlichkeit des Schuldners

(1) Der Schuldner hat Vorsatz und Fahrlässigkeit zu vertreten, wenn eine strengere oder mildere Haftung weder bestimmt noch aus dem sonstigen Inhalt des Schuldverhältnisses, insbesondere aus der Übernahme einer Garantie oder eines Beschaffungsrisikos zu entnehmen ist. Die Vorschriften der §§ 827 und 828 finden entsprechende Anwendung.

(2) Fahrlässig handelt, wer die im Verkehr erforderliche Sorgfalt außer Acht lässt.

(3) Die Haftung wegen Vorsatzes kann dem Schuldner nicht im Voraus erlassen werden.

§ 278 Verantwortlichkeit des Schuldners für Dritte

Der Schuldner hat ein Verschulden seines gesetzlichen Vertreters und der Personen, deren er sich zur Erfüllung seiner Verbindlichkeit bedient, in gleichem Umfang zu vertreten wie eigenes Verschulden. Die Vorschrift des § 276 Abs. 3 findet keine Anwendung.

§ 280 Schadensersatz wegen Pflichtverletzung

(1) Verletzt der Schuldner eine Pflicht aus dem Schuldverhältnis, so kann der Gläubiger Ersatz des hierdurch entstehenden Schadens verlangen. Dies gilt nicht, wenn der Schuldner die Pflichtverletzung nicht zu vertreten hat.

(2) Schadensersatz wegen Verzögerung der Leistung kann der Gläubiger nur unter der zusätzlichen Voraussetzung des § 286 verlangen.

(3) Schadensersatz statt der Leistung kann der Gläubiger nur unter den zusätzlichen Voraussetzungen des § 281, des § 282 oder des § 283 verlangen.

§ 281 Schadensersatz statt der Leistung wegen nicht oder nicht wie geschuldet erbrachter Leistung

(1) Soweit der Schuldner die fällige Leistung nicht oder nicht wie geschuldet erbringt, kann der Gläubiger unter den Voraussetzungen des § 280 Abs. 1 Schadensersatz statt der Leistung verlangen, wenn er dem Schuldner erfolglos eine angemessene Frist zur Leistung oder Nacherfüllung bestimmt hat. Hat der Schuldner eine Teilleistung bewirkt, so kann der Gläubiger Schadensersatz statt der ganzen Leistung nur verlangen, wenn er an der Teilleistung kein Interesse hat. Hat der Schuldner die Leistung nicht wie geschuldet bewirkt, so kann der Gläubiger Schadensersatz statt der ganzen Leistung nicht verlangen, wenn die Pflichtverletzung unerheblich ist.

(2) Die Fristsetzung ist entbehrlich, wenn der Schuldner die Leistung ernsthaft und endgültig verweigert oder wenn besondere Umstände vorliegen, die unter Abwägung der beiderseitigen Interessen die sofortige Geltendmachung des Schadensersatzanspruchs rechtfertigen.

(3) Kommt nach der Art der Pflichtverletzung eine Fristsetzung nicht in Betracht, so tritt an deren Stelle eine Abmahnung.

(4) Der Anspruch auf die Leistung ist ausgeschlossen, sobald der Gläubiger statt der Leistung Schadensersatz verlangt hat.

(5) Verlangt der Gläubiger Schadensersatz statt der ganzen Leistung, so ist der Schuldner zur Rückforderung des Geleisteten nach den §§ 346 bis 348 berechtigt.

§ 282 Schadensersatz statt der Leistung wegen Verletzung einer Pflicht nach § 241 Abs. 2

Verletzt der Schuldner eine Pflicht nach § 241 Abs. 2, kann der Gläubiger unter den Voraussetzungen des § 280 Abs. 1 Schadensersatz statt der Leistung verlangen, wenn ihm die Leistung durch den Schuldner nicht mehr zuzumuten ist.

§ 283 Schadensersatz statt der Leistung bei Ausschluss der Leistungspflicht

Braucht der Schuldner nach § 275 Abs. 1 bis 3 nicht zu leisten, kann der Gläubiger unter den Voraussetzungen des § 280 Abs. 1 Schadensersatz statt der Leistung verlangen. § 281 Abs. 1 Satz 2 und 3 und Abs. 5 findet entsprechende Anwendung.

§ 286 Verzug des Schuldners

(1) Leistet der Schuldner auf eine Mahnung des Gläubigers nicht, die nach dem Eintritt der Fälligkeit erfolgt, so kommt er durch die Mahnung in Verzug. Der Mahnung stehen die Erhebung der Klage auf die Leistung sowie die Zustellung eines Mahnbescheids im Mahnverfahren gleich.

(2) Der Mahnung bedarf es nicht, wenn

a) für die Leistung eine Zeit nach dem Kalender bestimmt ist,

b) der Leistung ein Ereignis vorauszugehen hat und eine angemessene Zeit für die Leistung in der Weise bestimmt ist, dass sie sich von dem Ereignis an nach dem Kalender berechnen lässt,

c) der Schuldner die Leistung ernsthaft und endgültig verweigert,

d) aus besonderen Gründen unter Abwägung der beiderseitigen Interessen der sofortige Eintritt des Verzugs gerechtfertigt ist.

(3) Der Schuldner einer Entgeltforderung kommt spätestens in Verzug, wenn er nicht innerhalb von 30 Tagen nach Fälligkeit und Zugang einer Rechnung oder gleichwertigen Zahlungsaufstellung leistet; dies gilt gegenüber einem Schuldner, der Verbraucher ist, nur, wenn auf diese Folgen in der Rechnung oder Zahlungsaufstellung besonders hingewiesen worden ist. Wenn der Zeitpunkt des Zugangs der Rechnung oder Zahlungsaufstellung unsicher ist, kommt der Schuldner, der nicht Verbraucher ist, spätestens 30 Tage nach Fälligkeit und Empfang der Gegenleistung in Verzug.

(4) Der Schuldner kommt nicht in Verzug, solange die Leistung infolge eines Umstands unterbleibt, den er nicht zu vertreten hat.

§ 287 Verantwortlichkeit während des Verzugs

Der Schuldner hat während des Verzugs jede Fahrlässigkeit zu vertreten. Er haftet wegen der Leistung auch für Zufall, es sei denn, dass der Schaden auch bei rechtzeitiger Leistung eingetreten sein würde.

§ 433 Vertragstypische Pflichten beim Kaufvertrag

(1) Durch den Kaufvertrag wird der Verkäufer einer Sache verpflichtet, dem Käufer die Sache zu übergeben und das Eigentum an der Sache zu verschaffen. Der Verkäufer hat dem Käufer die Sache frei von Sach- und Rechtsmängeln zu verschaffen.

(2) Der Käufer ist verpflichtet, dem Verkäufer den vereinbarten Kaufpreis zu zahlen und die gekaufte Sache abzunehmen.

Literaturverzeichnis

[AG00] Alsinet, T. ; Godo, L.: A complete calculus for possibilistic logic programming with fuzzy propositional variables. In: *Proceedings of the UAI-2000 Conference*, 2000, S. 1–10

[AG01] Alsinet, T. ; Godo, L.: A proof procedure for possibilistic logic programming with fuzzy constants. In: *Proceedings of the ECSQARU-2001 Conference*, 2001, S. 760–771

[BGB07] *Bürgerliches Gesetzbuch (BGB) der Bundesrepublik Deutschland*, 8. Februar 2007

[BH01] Besnard, P. ; Hunter, A.: A logic-based theory of deductive arguments. In: *Artificial Intelligence* 128 (2001), Nr. 1-2, S. 203–235

[BH06] Besnard, P. ; Hunter, A.: Knowledgebase Compilation for Efficient Logical Argumentation. In: *Proceedings of the 10th International Conference on Knowledge Representation (KR'06)*, AAAI Press, 2006, S. 123–133

[Bro96] Brox, H.: *Allgemeiner Teil des BGB*. München, Deutschland : Heymanns, 1996

[CCnS04] Capobianco, M. ; Chesñevar, C. ; Simari, G.: An argument-based framework to model an agent's beliefs in a dynamic environment. In: *Proceedings of the First International Workshop on Argumentation in Multiagent Systems (AAMAS)*, 2004

[CnML00] Chesñevar, C. ; Maguitman, A. ; Loui, R.: Logical Models of Argument. In: *ACM Computing Surveys* 32 (2000), Nr. 4, S. 337–383

[CnSAG04] Chesñevar, C. ; Simari, G. ; Alsinet, T. ; Godo, L.: A Logic Programming Framework for Possibilistic Argumentation with Vague Knowledge. In: *Proceedings of the International Conference in Uncertainty in Artificial Intelligence (UAI 2004)*. Banff, Kanada : AUAI Press, 2004, S. 76–84

[CnSG93] Chesñevar, C. ; Simari, G. ; García, A.: Making Argument System Computationally Attractive. In: *Proceedings of the XIII International Conference of the Chilean Society for Computer Science*, 1993

[GS02] García, A. ; Simari, G.: Defeasible Logic Programming: An Argumentative Approach. In: *Theory and Practice of Logic Programming* 4 (2002), Nr. 1-2, S. 95–138

[Lif96] Lifschitz, V.: Foundations of logic programming. In: Brewka, Gerhard (Hrsg.): *Principles of Knowledge Representation.* CSLI Publications, 1996, S. 69–127

[Lou87] Loui, R.: Defeat Among Arguments: A System of Defeasible Inference. In: *Computational Intelligence* 3 (1987), Nr. 3, S. 100–106

[NLJ96] Nwana, H. S. ; Lee, L. C. ; Jennings, N. R.: Coordination in Software Agent Systems. In: *The British Telecom Technical Journal* 14 (1996), Nr. 4, S. 79–88

[Nut88] Nute, D.: Defeasible reasoning: a philosophical analysis in PROLOG. In: Fetzer, J. H. (Hrsg.): *Aspects of artificial intelligence.* Kluwer, 1988, S. 251–288

[Pol95] Pollock, J.: Cognitive carpentry: A blueprint for how to build a person. In: *MIT Press* (1995)

[Poo85] Poole, D.: On the Comparison of Theories: Preferring the Most Specific Explanation. In: *Proceedings of 9th International Joint Conference on Artificial Intelligence.* IJCAI Inc., San Mateo, Kalifornien, Morgan Kaufmann, Los Altos, Kalifornien, 1985, S. 144–147

[SCG94] Simari, G. ; Chesñevar, C. ; García, A.: Focusing inference in defeasible argumentation. In: *Anales de la Conferencia IBERAMIA'94. Asociacion Venezolana para Inteligencia Artificial.* Caracas, Venezuela, 1994

[SCnG94] Simari, G. ; Chesñevar, C. ; García, A.: The Role of Dialectics in Defeasible Argumentation. In: *Anales de la XIV Conferencia Internacional de la Sociedad Chilena para Ciencias de la Computación.* Universidad de Concepción, Concepción, Chile, 1994

[SGCnS03] Stolzenburg, F. ; García, A. ; Chesñevar, C. ; Simari, G.: Computing Generalized Specificity. In: *Journal of Non-Classical Logics* 13 (2003), Nr. 1, S. 87–113

[SL92] Simari, G. ; Loui, R.: A Mathematical Treatment of Defeasible Reasoning and its Implementation. In: *Artificial Intelligence* 53 (1992), S. 125–157