Utrecht University
Cognitive Artificial Intelligence

Master's Thesis

Implementation of argument-based practical reasoning

Author:
Wietske Visser
(0511412)

Supervisors:
prof. dr. mr. Henry Prakken
dr. Gerard Vreeswijk

17 March 2008

Contents

1 Introduction **1**
- 1.1 Argument-based reasoning . . . 1
- 1.2 Practical reasoning . . . 2
- 1.3 Research question and method . . . 2
- 1.4 Outline of this thesis . . . 3

2 Theories of practical reasoning **4**
- 2.1 Combining epistemic and practical reasoning . . . 4
- 2.2 The practical syllogism . . . 6
- 2.3 Accrual of arguments . . . 6

3 The formalism **10**
- 3.1 Arguments . . . 10
  - 3.1.1 The language . . . 10
  - 3.1.2 The structure of arguments . . . 11
  - 3.1.3 Inference schemes . . . 11
  - 3.1.4 Strength of arguments . . . 13
  - 3.1.5 Conflicts between arguments . . . 14
- 3.2 Dung's abstract argumentation framework . . . 15
  - 3.2.1 Grounded semantics . . . 15
  - 3.2.2 Preferred semantics . . . 16
  - 3.2.3 E-p-semantics . . . 16
- 3.3 Argument games . . . 17
  - 3.3.1 General properties . . . 17
  - 3.3.2 The G-game . . . 18
  - 3.3.3 The P-game . . . 19
  - 3.3.4 The GP-game . . . 20

4 The Epistemic and Practical Reasoner **22**
- 4.1 Overview . . . 22
- 4.2 User interface . . . 23
- 4.3 Belief base data structure . . . 23
- 4.4 Output . . . 27

5 Algorithms **28**
- 5.1 Construction of arguments . . . 28
  - 5.1.1 Global approach . . . 28
  - 5.1.2 Algorithm . . . 28

5.2 Construction of argument games 30
5.2.1 Global approach 30
5.2.2 Algorithm 31

6 Examples **33**
6.1 E-p-semantics 33
6.1.1 Example 1 33
6.1.2 Example 2 34
6.1.3 Example 3 35
6.1.4 Example 4 36
6.1.5 Example 5 37
6.2 The practical syllogism and accrual 37
6.2.1 Simple example 37
6.2.2 Jogging example 38
6.2.3 Judge example 39

7 Discussion and conclusion **40**
7.1 Related research 40
7.2 Results 41
7.2.1 E-p-semantics 41
7.2.2 The practical syllogism 42
7.2.3 Accrual of arguments 42
7.2.4 Conclusion 43
7.3 Further work 43

A User manual **44**

Bibliography **45**

Chapter 1

Introduction

In this graduation project I have implemented an argument-based practical reasoning system. I will briefly introduce argument-based reasoning and practical reasoning in Sections 1.1 and 1.2 respectively. The main research question is stated in Section 1.3, along with the method that will be used to answer it. Finally, Section 1.4 presents the outline of this thesis.

1.1 Argument-based reasoning

Defeasible reasoning is a way to reason with inconsistent or incomplete belief bases. Its inferences are not absolutely certain, but can still be made, provided that there is no information to the contrary that defeats them. So it can happen that a certain inference can be made from a set of beliefs, but cannot be made anymore if more information becomes known. Hence defeasible reasoning is nonmonotonic. Since this is also the way humans reason in their daily lives, the term commonsense reasoning is often used too. Prakken and Vreeswijk [15] present an extensive overview of defeasible reasoning systems.

Argument-based reasoning systems are useful formalisations of defeasible reasoning. They are based on arguments which may contradict each other, for example because they have conflicting conclusions (rebuttal), or one is an argument for the inapplicability of an inference step made in the other (undercut).

Arguments can be seen as the defeasible counterpart of proofs in classical logic, but their status is quite different. Classical logic aims to determine the truth value of a given proposition, and one proof is sufficient to do that. Defeasible logic does not search for truth, but rather for having some justification for a given proposition, or more exactly, having more justification for it than against it. So whether a proposition is justified is not determined by a single argument, but by the interaction between multiple arguments for and against the proposition.

The basics of defeasible argumentation are well understood. Dung [6] has defined an abstract framework with several semantics that define which arguments are justified (see also Chapter 3, Section 3.2). Proof theories for these semantics in the form of argument games have been developed by Prakken and Sartor [14] and Vreeswijk and Prakken [20] (see also Chapter 3, Section 3.3).

Argument-based reasoning systems provide a middle way between classical

(monotonic) reasoning systems and statistical methods such as Bayesian networks. Classical logic has the problem that all possible exceptions of a rule have to be listed, and moreover that they must be known to be false before the rule can be applied. The problem with Bayesian networks is that the prior and conditional probabilities of all possible situations have to be estimated, which is often impossible. Defeasible argumentation systems do not have these problems, but of course they have their own (for example, because all relevant arguments must be created and considered, they can become quite time-consuming compared to classical logic, where one proof is sufficient). Which approach is most appropriate depends on the context of the problem.

1.2 Practical reasoning

Argument-based reasoning is mostly used to reason about knowledge or beliefs (epistemic or theoretical reasoning). However, more and more attention is payed to practical reasoning, that is, reasoning about goals, desires and actions. Practical reasoning is essential for agents whose actions are not straightforward. For example, when an agent has multiple options for achieving a goal, he must choose one action, preferably one that also achieves some other goal or at least does not prevent too many other goals from being achieved. Or maybe the agent has multiple goals that it cannot achieve simultaneously. Then he must determine which action to undertake to achieve the best set of goals.

Practical reasoning is highly dependent on epistemic reasoning, but it has to deal with its own set of properties and problems, which will be discussed later in this thesis.

Some existing argument-based practical reasoning systems focus on which desires can be defeasibly inferred from a belief base, for example Prakken [13, 12] and Bench-Capon and Prakken [4]. Others decide between actions in a separate decision process, for example the ASPIC project [1]. Still others embed this kind of reasoning in a planning system, which generates not only desires but also plans, for example Pollock [10] and Rahwan and Amgoud [16]. The theories of ASPIC and Pollock have both been implemented. This project focuses on the first type of practical reasoning.

1.3 Research question and method

In this project I will implement an argument-based practical reasoning system, which integrates some existing theories of argument-based practical reasoning. Implementing a theory is relevant and interesting for two reasons. Firstly, it is a good method to check whether a theory is correct and fully specified. In an implementation no aspects can be left unspecified, even when they would seem obvious to humans. This way it will become clear how complete the theories are, and what is left to be specified further. If underspecified aspects are found, a choice must be made how to fill the gap. When a theory has been implemented, it can be tested against examples, which are often provided with the theory. Secondly, an implementation can be used as a prototype for real applications, such as an agent reasoning about his beliefs and desires. This project focuses on the first goal.

Three topics concerning argument-based practical reasoning have been chosen to focus on: the combination of epistemic and practical reasoning, the practical syllogism and accrual of arguments. As a formalisation of these topics some specific theories are chosen: Prakken's [13] combination of epistemic and practical reasoning in the form of e-p-semantics, and Bench-Capon and Prakken's [4] formalisation of the practical syllogism and accrual, based on Prakken's [12] mechanism for accrual of arguments.

The research question can be stated as follows. Are the theories of e-p-semantics, the practical syllogism and accrual of arguments, as presented in the papers mentioned above, correct and completely specified? If not, what changes and additions are needed to make them better or more complete?

First the relevant literature is studied. The theories will be integrated into one formalism, based on Dung's [6] abstract argumentation framework. Then a phase of system analysis and design follows, resulting in a general layout of the classes that model the formalism's concepts and the flow of the program. The formalism is then implemented in the Java language. If any difficulties arise, the previous steps may have to be repeated. When the program is ready, it will be clear which aspects had to be specified further, and which problems have not been solved. Finally, the program will be used to test the theories against examples which are mostly taken from the literature.

1.4 Outline of this thesis

The outline of this thesis is as follows. In Chapter 2 the theories of practical reasoning that will be implemented are discussed. Then Chapter 3 presents the exact formalism that will be used in the implementation, in which these theories are combined. An overview of the program is given in Chapter 4. Chapter 5 discusses the most interesting algorithms of the program in detail. In Chapter 6 the behaviour of the program is illustrated using examples from the literature, and compared to the original discussion of the same examples. Finally, Chapter 7 concludes this thesis with a discussion of related research, results and further work.

Chapter 2

Theories of practical reasoning

Many theories of practical reasoning using defeasible argumentation can be found in the literature, covering a wide range of subjects. In this project the focus lies on three subjects. The first is the combination of epistemic and practical reasoning, for which Prakken's [13] e-p-semantics will be used. The second is the practical syllogism, which will be based on the system presented by Bench-Capon and Prakken [4]. The third subject is accrual of arguments, as formalised by Prakken [12] and Bench-Capon and Prakken [4].

This chapter discusses the three subjects and the chosen theories. In Section 2.1 e-p-semantics is introduced, the practical syllogism is presented in Section 2.2, and Section 2.3 discusses accrual of arguments.

2.1 Combining epistemic and practical reasoning

Prakken [13] presents an argument-based semantics for combined epistemic and practical reasoning, called e-p-semantics. He claims that in certain contexts, reasoning about beliefs is sceptical while reasoning about action is credulous. In the case of beliefs, choosing (randomly) between two propositions that one has equal reason to believe but cannot believe simultaneously seems not rational. In the case of actions, however, choosing between two alternatives that are equally well argumented but exclude each other is perfectly rational.

Prakken illustrates this with an example about John, a university lecturer who needs to finish a paper but also has to go to a remote town Faraway to give a talk. He knows of two ways to get to Faraway, by car and by bus. But because of his car sickness he will not be able to write his paper in either case. Then he cannot fulfill both of his desires at the same time, so he must choose one of them. This can be formalised as credulous reasoning. Then Bob tells John that there is a railway connection to Faraway, enabling him to finish his paper while travelling, so fulfilling both goals. But Mary says that there will be a railway strike, which Bob does not believe. John trusts Bob and Mary equally, so he can only credulously, not sceptically, believe that there will be a

train to Faraway. Then it will be rational for him not to act on this belief.

So it is assumed that a rational agent only considers action alternatives that are based on sceptical beliefs, and makes a credulous choice between them. The formalisation of this idea uses Dung's [6] abstract approach to defeasible argumentation. It is based on grounded semantics for sceptical epistemic reasoning and preferred semantics for credulous practical reasoning. One reason for this is that sceptical grounded and credulous preferred semantics have elegant proof theories in the form of argument games; the G-game for grounded sceptical semantics [14] and the P-game for preferred credulous semantics [20].

It is assumed that the logical language that is used can be divided into two disjoint sublanguages, one of epistemic formulas and one of practical formulas. Furthermore, an inference is called epistemic if all its premises and its conclusion are epistemic formulas, and it is called practical if its conclusion is a practical formula. All inferences in an argument must either be epistemic or practical (so no inference with an epistemic conclusion can have a practical formula as a premise). An argument is called epistemic if all its inferences are epistemic, otherwise it is called practical. An argumentation system is said to be an e-p-argumentation system if its set of arguments consists of two disjoint sets of epistemic and practical arguments.

Now the semantics of combined sceptical epistemic and credulous practical reasoning is defined as follows. Let $\mathcal{H} = (\mathcal{A}, \mathcal{D})$ be an e-p-argumentation system with grounded extension $G_{\mathcal{H}}$. Let $\mathcal{H}_g = (\mathcal{A}_g, \mathcal{D}_g)$ be obtained from $\mathcal{H}$ by removing from $\mathcal{A}$ all arguments that are defeated by an epistemic argument that is not defeated by an argument in $G_{\mathcal{H}}$, and restricting $\mathcal{D}$ to $\mathcal{A}_g$. Then S is an e-p-extension of $\mathcal{H}$ if S is a preferred extension of $\mathcal{H}_g$.

However, first determining the grounded extension of all belief arguments, adding all justified beliefs to the belief base and then constructing the preferred extensions of the new theory is not the right approach. Reasoning about beliefs and planning is interleaved; practical reasoning determines which beliefs are relevant, as is stated by Pollock [11] as follows:

> '...interleaving planning and epistemic reasoning, and performing the epistemic reasoning in a way that is directed by or focused by the practical considerations involved in the planning [is important]. The epistemic reasoner must be interest-driven in the sense that it tries to answer particular queries that are passed to it by practical cognition. These will be factual queries of relevance to the planning problem, and will arise in the course of planning. We cannot assume that the planner comes to the planning problem equipped with precisely the knowledge it needs to solve the planning problem (and without the ability to do further reasoning) without ignoring essential aspects of planning and cognition.'

Therefore, to prove whether a given action can be accepted credulously while relying only on sceptically accepted beliefs, without determining entire extensions, Prakken proposes a new argument game for his semantics: the GP-game. It combines the G-game for epistemic arguments and the P-game for practical arguments. But in contrast to these games, a distinction is made between a player and its dialectical role. If player PRO moves an epistemic argument that defeats a practical argument moved by player CON, then he in fact attacks an epistemic subargument of CON's practical argument and has to show that

it is not in the grounded extension. So pro acts as the opponent in the subgame about the epistemic subargument. The exact rules of the GP-game are presented in the next chapter.

2.2 The practical syllogism

Abduction is a kind of 'backwards' inference. It is often used in diagnosis (finding the cause of an observation). For example, if someone knows that a causes b and that b is the case, he might want to infer a (if a is the only or the best explanation of b). It is also a very natural way to reason about actions. For example, if action a causes state b, and I have the goal to be in state b, then it is rational for me to wish to do a (if I am capable of doing a, if a has no unwanted side-effects, and if there is no better alternative for achieving b). This kind of rule is called the practical syllogism, and it can be traced back to Aristotle. However, its application is not straightforward, as can be seen from the conditions.

Bench-Capon and Prakken [4] present a formalisation of reasoning with the practical syllogism. They try to handle the issues related to the practical syllogism, such as its abductive nature, and hence the need to consider alternative actions and possible negative side effects, with an argumentation system that is based on Dung's [6] abstract framework of defeasible argumentation. It is instantiated with a tree-style structure of arguments, and incorporates Prakken's [12] accrual mechanism of arguments. This accrual mechanism will be attended to in the next section; this section only focuses on the practical syllogism.

The language is assumed to be a propositional modal logic with a single modality D standing for desire. The propositional part of the language can be divided into controllable and uncontrollable formulas. The logic of D is of type KD, which validates $\neg(\mathrm{D}\phi \wedge \mathrm{D}\neg\phi)$. Occurrences of D cannot be nested and defaults cannot contain the modality D.

On top of this language, defeasible conditionals or defaults are defined: $\phi \Rightarrow \psi$ where ψ is a single propositional literal and ϕ is a conjunction of propositional literals. Defaults occur in three forms: $a \wedge r \Rightarrow p$ (realising a in circumstance r achieves p), $a \Rightarrow p$ (realising a achieves p), and $r \Rightarrow r'$ (one circumstance typically implies another circumstance).

Next to a defeasible modus ponens rule, two new inference rules are introduced: the positive practical syllogism (from $a \wedge r \Rightarrow p$, Dp and r, infer Da) and the negative practical syllogism (from $a \wedge r \Rightarrow \neg p$, D$p$ and r, infer $\neg$Da).

The negative practical syllogism formalises the need to consider possible negative side effects. The need to consider alternative ways to achieve the same goal is formalised by a new way of defeat. An argument can defeat another argument if they both apply the positive practical syllogism to the same desire premise but their conclusions are different action desires. The exact definition will be given in the next chapter.

2.3 Accrual of arguments

Consider a case in which one has two arguments for performing action a, and one argument for not performing a. The last argument is stronger than the first

two arguments individually, but taken together, the first two are stronger. In that case one would want to accrue the first two arguments in the argumentation system. Another application of accrual in practical reasoning is the accrual of positive (wanted) effects of a particular action on the one hand, and negative (unwanted) effects on the other hand.

Bench-Capon and Prakken [4] describe a system for practical reasoning which makes use of Prakken's [12] accrual mechanism. In their system they apply accrual to the practical syllogism (abductive reasoning), but in general accrual can be applied to any kind of defeasible reasoning.

Prakken [12] takes as starting point of any formalisation of accrual that adding more reasons can make one's case stronger. He defines three principles that are meant to constrain formalisations of this idea.

1. Accruals are sometimes weaker than their elements. Also: in general the strength of an accrual cannot be calculated from the strengths of its elements.

2. An accrual makes its elements inapplicable (this restriction is not needed if accruals are never weaker than their elements).

3. Flawed reasons or arguments may not accrue.

Prakken [12] discusses the knowledge representation approach to this idea. In probabilistic networks accrual is enforced, because they are antecedent-complete. In nonmonotonic logic antecedent-completeness is not wanted, because it is meant to deal with incomplete information. But it is possible to combine statements about the presence or absence of reasons in the antecedent of a conditional, or to state exceptions to default rules. In this case accrual boils down to defining strength relations between conflicting rules. However, Prakken argues that an inference approach, in which accrual is modelled as an inference rule, has certain advantages to the knowledge representation approach. First, in the knowledge representation approach more rules need to be formulated, although these could be generated automatically. Second, the knowledge representation approach requires that each reason can be expressed with the same kind of conditional operator, while in reality reasons of different types may accrue. And third, in the knowledge representation approach undercutters of individual reasons must be represented as undercutters of all accruals in which the reason takes part, which seems not natural.

Prakken's inference system can be summarized with the following changes and additions to a system like that of Bench-Capon and Prakken [4] as described in the previous section. The existing defeasible inference rules are altered such that the conclusion is labelled with the premises (the premises themselves must be unlabelled); a new defeasible inference rule called accrual is introduced which takes any set of labelled versions of a certain formula and produces the unlabelled version; rebuttal is only allowed between arguments with unlabelled conclusions, and an undercutter's conclusion must be unlabelled; and an undercutter scheme called accrual undercut is formulated to the effect that an accrual undercutter A undercuts an accrual B if B's elements are a proper subset of A's elements. It follows that in any argument the application of labelling inference rules strictly alternates with the application of accrual.

This system satisfies the three principles defined above. It does not make any assumptions on the (relative) strength of accruals, so the first principle is

trivially satisfied. The second principle is satisfied by the accrual undercutter and the restriction on the defeat relations. The third principle is also satisfied, since an accrual that has left out a flawed reason will be undercut by the maximal accrual, but is reinstated by the defeater of the flawed reason.

One difficulty relating to accrual is the definition of strength of accrued arguments. Whereas the strength of simple arguments can be straightforwardly computed from the strengths of their premises (for example with the weakest link principle), the definition of the strength of an accrual is less straightforward. The idea behind accrual is that having multiple arguments for a conclusion can make one's case stronger than the individual arguments. But in some cases an accrual may be weaker than its constituting arguments (Prakken's first principle). So in general the strength of an accrual cannot be calculated from the strengths of its elements. Prakken leaves the strength of an accrual unspecified, but in the implementation that is not possible; the strength is needed to determine whether a rebutting or alternative argument is strong enough to defeat its target.

Bench-Capon and Prakken [4] define a preference ordering on arguments based on the goals that are reached and prevented if the argument's final desire is carried out. (Actually, this is a simplification which applies under the assumption that each goal's only promoted value is the goal itself. Since no other values will be modelled here, this assumption is safe.) If the desire conclusion is positive, the goals reached are all positive practical formulas occurring in the argument whose epistemic versions can be derived from the premises and the epistemic version of the conclusion of the argument. The goals prevented are all positive practical formulas occurring in a rebutting argument whose negated epistemic versions can be derived from the premises of that rebutting argument and the epistemic version of the conclusion of the argument. If the desire conclusion is negative, the goals reached/prevented are the union of the goals reached/prevented of all maximal proper subarguments for which the set of goals reached/prevented is defined. So every argument has an associated pair of a set of reached goals and a set of prevented goals. There exists a partial preorder on such pairs, and the ordering of arguments is the same.

However, it is problematic to use this definition in practice for the following reasons.

- The ordering only applies to practical arguments, since epistemic arguments do not contain any practical formulas. However, accrual also applies to epistemic arguments.
- The ordering of arguments does not take into account the relative strengths of rules and facts from the belief base. In itself this is not a problem, but when accrual is incorporated into a system that does rely on these strengths, like the one that will be used here, the two will have to be combined somehow.
- For some arguments, the set of prevented goals is defined outside the argument: its definition is based on the rebutting arguments. This means that the strength of an individual argument cannot be determined independently from other arguments.
- The ordering on pairs of sets of reached and prevented goals, and hence the argument ordering, is only partial, so not all practical arguments can

be compared on their strength. In itself this is not a problem, but it is if the ordering of arguments is modelled using a numeric strength, whose ordering is complete.

- The ordering must be defined somewhere. It is not dependent on the ordering of individual goals. However, it is not directly clear which goals must be considered; not only goals from the goal base can be reached or prevented, but also derived goals. To determine these, either the arguments must be generated first and then ordered by the user (which is not quite 'automatic reasoning'), or all possible combinations of possible goals must be derived from the belief base and given a strength by the user before arguments are created, but this may result in far too many goals (in a proposition-based language the set of goals will at least be finite; in a predicate language with variables this will be impossible).

Because these problems cannot easily be solved, only some simple ad hoc mechanisms to compute accrual strength will be used, thereby unfortunatley violating Prakken's first principle of accrual. They will be presented in the next chapter.

Chapter 3

The formalism

This chapter presents a structured and detailed overview of the formalism on which the implementation is based. This formalism incorporates the theories of Chapter 2, and Dung's [6] grounded and preferred semantics. Section 3.1 is about arguments; it covers the language that the arguments are based upon, the structure of arguments, inference schemes, strength of arguments and conflicts between arguments. Certain choices have been made where the theories were underspecified (for example the exact definition of the grammar and the strength of arguments). Section 3.2 recapitulates Dung's abstract argumentation framework and three types of semantics for it: grounded semantics, preferred semantics and e-p-semantics. This section shows how e-p-semantics fits into Dung's framework. Finally, Section 3.3 is about argument games as they are used in the implementation; it defines the general properties and three specific argument games: the G-game for grounded semantics, the P-game for preferred credulous semantics, and the GP-game for e-p-semantics. Examples that illustrate the formalism can be found in Chapter 6.

3.1 Arguments

3.1.1 The language

The logic that the arguments are based upon is a propositional modal logic. It has a single modality D, standing for desire, which cannot be nested and is of type KD (insofar as this applies to formulas where D is not nested; this boils down to the DKD inference scheme defined in Section 3.1.3). A belief base consists of facts (beliefs) and rules (defeasible conditionals). The grammar is specified below. Note that the term 'formula' is used in a stricter sense than usual; here, conjuncts, disjuncts and implications are not formulas, but only literals preceded by an optional negation and modal operator D.

```
beliefBase ::= ( fact | rule )+
fact       ::= formula ( strength )? "."
formula    ::= ( ( "~" )? "D" )? literal
literal    ::= ( "~" )? atom
atom       ::= ["a"-"z"] ( ["a"-"z","A"-"Z","0"-"9"] )*
strength   ::= ( "0." ( ["0"-"9"] )+ ) | "1"
```

```
rule       ::= ( ruleName ":" )?  formula "<-" formula ( "," formula )*
               ( strength )? "."
ruleName   ::= atom
query      ::= formula "."
```

Formulas, facts and rules can be of either of two types: epistemic or practical. A formula is epistemic if it does not contain an occurrence of the modal operator D, and practical if it does. A fact's type is the same as the type of its consequent (its single formula). A rule is epistemic if its consequent (the formula left of the arrow) and all its antecedents (the formulas right of the arrow) are epistemic, and practical if its consequent is practical. Facts with an epistemic consequent and one or more practical antecedents are disallowed.

Facts and rules have an associated strength between 0 and 1. If the strength is not specified, it is assumed to be 1 by default.

This language fulfills Prakken's [13] requirement that the language can be divided into two disjoint sublanguages for epistemic and practical formulas. It also contains Bench-Capon and Prakken's [4] modal operator D. The distinction between controllable and uncontrollable literals is not enforced explicitly, but if the distinction is made implicitly in a consistent way, no uncontrollable literals will be desired.

3.1.2 The structure of arguments

Arguments are trees of chained defeasible inferences. This formalism does not model strict inferences. This means that every argument could in principle be defeated by another argument. Every argument has a conclusion (except accrual undercutter arguments, see below), which is the conclusion of its top inference, and zero or more subarguments, whose conclusions are the premises of the top inference. Inferences are instantiations of inference schemes, which are listed below in Section 3.1.3. No circular arguments are allowed; if an argument's conclusion is ϕ, this formula ϕ may not occur anywhere else in the argument (except when the top inference is accrual, see below). Because of this, and because the belief base is finite, only a finite number of arguments can be constructed.

Like formulas, arguments have a type (epistemic or practical). An argument is epistemic if its conclusion and all its subarguments are epistemic. An argument is practical if its conclusion is practical. Arguments with a practical subargument and an epistemic conclusion do not exist. This is the same as Prakken's [13] definition of epistemic and practical arguments.

3.1.3 Inference schemes

All inferences that can be made are instantiations of the following inference schemes.

From fact is a dummy inference scheme where the conclusion is a fact in the belief base. This inference scheme and the next are 'dummy' because they conclude nothing new; they are only needed to transform Fact or Rule objects into Formula objects in the implementation (see Chapter 4, Section 4.3). An argument with this inference has no subarguments.

$$\frac{\phi \textit{ strength } \texttt{"."} \in \text{belief base}}{\phi}\ \text{FF}$$

From rule is a dummy inference scheme where the conclusion is a rule in the belief base. An argument with this inference has no subarguments.

$$\frac{\textit{rule name } \texttt{":"}\ \phi\ \texttt{"<-"}\ \psi_1\ \texttt{","}\ \ldots\ \texttt{","}\ \psi_n \textit{ strength } \texttt{"."} \in \text{belief base}}{\phi\ \texttt{"<-"}\ \psi_1\ \texttt{","}\ \ldots\ \texttt{","}\ \psi_n}\ \text{FR}$$

Rule application is an inference scheme where the conclusion is the consequent of a rule in the belief base. The premises (conclusions of subarguments) are the rule itself and its antecedents.

$$\frac{\phi\ \texttt{"<-"}\ \psi_1\ \texttt{","}\ \ldots\ \texttt{","}\ \psi_n \quad \psi_1 \quad \ldots \quad \psi_n}{\phi}\ \text{RA}$$

DKD is an inference scheme formalisation of the assumption that the logic of the modal operator D is of type KD. This means that for any atom ϕ, $\texttt{~D~}\phi$ can be inferred from $\texttt{D}\phi$, and $\texttt{~D}\phi$ can be inferred from $\texttt{D~}\phi$ (but not the other way round). An argument with this inference has exactly one subargument.

$$\frac{\texttt{"D"}\phi}{\texttt{"~D~"}\phi}\ \text{DKD} \qquad \frac{\texttt{"D~"}\phi}{\texttt{"~D"}\phi}\ \text{DKD}$$

Positive practical syllogism is an abductive inference scheme. The conclusion is the practical version of one of the antecedents of a rule in the belief base. Premises are the rule, the practical version of its consequent, and all other antecedents.

$$\frac{\psi\ \texttt{"<-"}\ \phi_1\ \texttt{","}\ \ldots\ \texttt{","}\ \phi_n \quad \texttt{"D"}\psi \quad \phi_1 \quad \ldots \quad \phi_{i-1} \quad \phi_{i+1} \quad \ldots \quad \phi_n}{\texttt{"D"}\phi_i}\ \text{PPS}$$

Negative practical syllogism is an abductive inference scheme. The conclusion is the negated practical version of one of the antecedents of a rule in the belief base. Premises are the rule, the practical version of the negation of its consequent, and all other antecedents.

$$\frac{\texttt{"~"}\psi\ \texttt{"<-"}\ \phi_1\ \texttt{","}\ \ldots\ \texttt{","}\ \phi_n \quad \texttt{"D"}\psi \quad \phi_1 \quad \ldots \quad \phi_{i-1} \quad \phi_{i+1} \quad \ldots \quad \phi_n}{\texttt{"~D"}\phi_i}\ \text{NPS}$$

Accrual is an inference scheme whose premises and conclusion are all the same formula. Instead of labelling and delabelling formulas, as Prakken [12] and Bench-Capon and Prakken [4] do, it is demanded that the premises are all conclusions of arguments that do not have accrual as their top inference scheme, which is equivalent. A restriction is added that all subarguments must have different top inferences (an explanation for this is given at the end of this section).

$$\frac{\phi \quad \ldots \quad \phi}{\phi}\ \text{ACCR}$$

Accrual undercutter is an inference scheme that resembles accrual, but does not have a formula as conclusion. As with accrual, it is demanded that the premises are all conclusions of arguments that do not have accrual as their top inference scheme, and that all subarguments have different top inferences.

$$\frac{\phi \quad \ldots \quad \phi}{} \text{ AU}$$

All leaves of an argument tree are either from fact or from rule inferences, because they are the only ones without subarguments. The term 'simple argument' is used to denote any argument whose top inference scheme is not accrual, as opposed to 'accrual argument'.

The accrual and accrual undercutter inference schemes are only used if the use of accrual is turned on. The positive and negative practical syllogism inference schemes are only used if the use of the practical syllogism is turned on. If this is the case, rule application may only be used on epistemic rules. This has the same effect as Bench-Capon and Prakken's [4] requirement that defaults (rules) may not contain the modality D.

Refinement of the accrual inference scheme

One might think that accrual can be applied to all possible combinations of simple arguments with the same conclusion. This is also allowed in Prakken's [12] system. However, this will result in more arguments than are needed. The problem is best illustrated with an example. Consider the following argument:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{a<-b}}\,\text{FR}}{\texttt{a<-b}}\,\text{ACCR} \quad \dfrac{\dfrac{\dfrac{\overline{\texttt{-b}}\,\text{FF}}{\texttt{b}}\,\text{RA}}{\texttt{b}}\,\text{ACCR}}{\texttt{b}}\,\text{RA}}{\texttt{a}} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{a<-b}}\,\text{FR}}{\texttt{a<-b}}\,\text{ACCR} \quad \dfrac{\overline{\texttt{-b}}\,\text{FF}}{\texttt{b}}\,\text{ACCR}}{\texttt{b}}\,\text{RA}}{\texttt{a}}}{\texttt{a}}\,\text{ACCR}$$

It is an accrual with two elements. However, the two elements are similar: they have the same top inference. The only difference between them is that one of the subarguments of the second element is a smaller accrual than the corresponding subargument of the first element. Accruing them together makes no sense; the same information is contained in the argument that uses the bigger accrual. In Prakken's [12] original theory this problem can be solved by demanding that accruing elements have different labels (each conclusion of a non-accrual argument is labelled with its premises, so arguments with the same inference have the same label). In the implementation, this problem is solved by ensuring that an accrual argument does not contain two subarguments with the same top inference. To avoid unwanted accrual undercut (see Section 3.1.5), the same restricion must be applied to the accrual undercutter inference scheme.

3.1.4 Strength of arguments

Every argument has a strength, which, in the case of simple arguments, depends in some way on the strengths of its subarguments, and ultimately on the strengths of the facts and rules from the belief base that it uses. Two possible principles to determine the strength of an argument are the weakest link principle [9] and the last link principle [14].

Weakest link An argument is as strong as its weakest subargument, or as strong as the fact or rule that it is based upon if it has no subarguments.

Last link An argument is as strong as the rule that was last used in an inference, or as strong as the fact that is based upon if it contains no rule.

As discussed in Section 2.3, the computation of the strength of accrual arguments is less straightforward. Only three simple ad hoc mechanisms will be used.

All equal All accrual arguments have the same strength (1). The strengths of subarguments are ignored.

Strongest link The strength of an accrual argument is the strength of the strongest of its elements (direct subarguments). The strengths of these subarguments, which are themselves not accruals, are determined by the strength mechanism that is used for simple arguments.

Number of promoted desires The strength of an accrual argument depends on the number of desires that are promoted by the argument. Only desires from the belief base are counted, derived desires are not. If p is the number of promoted desires of an accrual argument, its strength is $p/(p+1)$. This definition was chosen because it is simple, the strength stays within the range [0,1], and the more desires are promoted, the higher the strength.

3.1.5 Conflicts between arguments

There are three types of conflict between arguments: rebuttal, alternative and undercut. The only type of undercut that is used is accrual undercut. Other approaches often incorporate another form of undercut, which is not a general undercutter scheme like accrual undercut, but a user-defined undercutter for one particular inference. For this, the language would have to be extended so that inferences can be expressed too. Since this is not straightforward and this type of undercut is not essential for the project, this has not been done yet.

Definition 1 (Rebuttal)**.** *An argument A* rebuts *an argument B if the conclusion of A is the negation of the conclusion of B. If the use of accrual is turned on, both A's and B's top inference scheme must be accrual.*

Definition 2 (Alternative)**.** *If accrual is not used, an argument A is an* alternative *to an argument B if both A's and B's top inference scheme is the positive practical syllogism, their conclusions are different and they have one practical premise in common.*

If accrual is used, an argument A is an alternative *to an argument B if they have different positive practical formulas as conclusions, both arguments' top inference is an accrual, the top inference scheme of all elements of these accruals is the positive practical syllogism (*PPS*), and at least one* PPS *subargument of A has a practical premise that is also a premise of a* PPS *subargument of B.*

Definition 3 (Accrual undercut)**.** *An argument A* accrual-undercuts *an argument B if A's top inference scheme is the accrual undercutter, B's top inference scheme is accrual, and B's subarguments are a proper subset of A's subarguments.*

Now, defeat between arguments is defined as follows:

Definition 4 (Defeat)**.** *An argument A* defeats *an argument B if*

- *A rebuts B and A is as least as strong as B; or*
- *A accrual-undercuts B; or*
- *A is an alternative to B and A is as least as strong as B; or*
- *A defeats a subargument of B.*

Note that epistemic arguments cannot be defeated by practical arguments, since rebuttal, alternative and accrual undercut only occur between arguments of the same type, and epistemic arguments never have practical subarguments.

3.2 Dung's abstract argumentation framework

Dung [6] defines a general, abstract framework for the semantics of defeasible argumentation systems. It is abstract because it leaves the internal structure of arguments unspecified. The framework only supposes the existence of a set of arguments with a binary defeat relation defined on it.

Definition 5 (Argumentation framework)**.** *An* argumentation framework *AF is a pair* $\langle \mathcal{A}, \text{defeat} \rangle$ *where* $\mathcal{A}$ *is a set of arguments, and* defeat *is a binary relation on* $\mathcal{A}$*:* $\text{defeat} \subseteq \mathcal{A} \times \mathcal{A}$.

Now several semantics are defined. Every semantics defines one or more sets of arguments (subsets of $\mathcal{A}$), so-called extensions, which can be seen informally as sets of 'justifiable' arguments. Two well-known semantics are grounded semantics and preferred semantics. An argument is said to be justified with respect to a semantics if it is a member of all extensions of that semantics. In grounded semantics, an argument is said to be defensible if it is not justified, but not defeated by a justified argument. In preferred semantics, an argument is defensible if it is a member of some, but not all extensions. Sceptical reasoning is concerned with determining whether an argument is justified, credulous reasoning with determining whether an argument is at least defensible.

The grounded and preferred extensions are defined below, but first the notions of a conflict-free set and an acceptable argument are defined.

Definition 6 (Conflict-free set)**.** *A set S of arguments is* conflict-free *if there are no arguments A and B in S such that A defeats B.*

Definition 7 (Acceptable argument)**.** *An argument A is* acceptable *with respect to a set S of arguments if every argument* $B \in \mathcal{A}$ *that defeats A is defeated by an argument in S.*

3.2.1 Grounded semantics

Every argumentation framework has exactly one grounded extension. Dung defines it in terms of the argumentation framework's characteristic function.

Definition 8 (Characteristic function)**.** *The* characteristic function F_{AF} *of an argumentation framework* $AF = \langle \mathcal{A}, \text{defeat} \rangle$ *is defined as follows:*
$F_{AF} : \mathcal{P}(\mathcal{A}) \rightarrow \mathcal{P}(\mathcal{A})$,
$F_{AF}(S) = \{A | A$ *is acceptable with respect to* $S\}$.

Definition 9 (Grounded extension)**.** *The* grounded extension *of an argumentation framework* AF *is the least fixed point of* F_{AF}.

The grounded extension can be found incrementally:

Proposition 10. *Consider the following sequence of arguments.*

- $F^0 = \emptyset$
- $F^{i+1} = \{A \in Args | A$ *is acceptable with respect to* $F^i\}$

Let $F^\omega = \cup_{i=0}^{\infty}(F^i)$. *Then all arguments in* F^ω *are in the grounded extension.*

That is, all arguments that are not defeated are justified, all arguments whose defeaters are all defeated themselves by such a justified argument are justified, and so on. If there is no undefeated argument, the grounded extension is empty.

If there are no arguments with infinitely many defeaters, which is the case here, Definition 9 and F^ω are equivalent.

3.2.2 Preferred semantics

An argumentation framework can have multiple preferred extensions. Dung defines them in terms of an admissible set.

Definition 11 (Admissible set)**.** *A set* S *of arguments is* admissible *if it is conflict-free and every argument in* S *is acceptable with respect to* S.

Definition 12 (Preferred extension)**.** *A* preferred extension *of an argumentation framework* AF *is a maximal (with respect to set inclusion) admissible set of* AF.

3.2.3 E-p-semantics

Prakken's [13] semantics of combined sceptical epistemic and credulous practical reasoning, e-p-semantics, is based on grounded and preferred semantics. It is defined as follows.

Definition 13 (E-p-argumentation framework)**.** *An* e-p-argumentation framework *is an argumentation framework whose set of arguments consists of two disjoint sets of epistemic and practical arguments.*

Definition 14 (E-p-extension)**.** *Let* $AF = \langle \mathcal{A}, \text{defeat} \rangle$ *be an e-p-argumentation framework with grounded extension* G_{AF}. *Let* $AF_g = \langle \mathcal{A}_g, \text{defeat}_g \rangle$ *be obtained from* AF *by removing from* $\mathcal{A}$ *all arguments that are defeated by an epistemic argument that is not defeated by an argument in* G_{AF} *and restricting* defeat *to* $\mathcal{A}_g$. *Then* S *is an* e-p-extension *of* AF *if* S *is a preferred extension of* AF_g.

3.3 Argument games

Semantics of defeasible argumentation systems define what the extensions of an argumentation framework are; they are declarative. Argument games, on the other hand, provide a procedural approach. That is, given an argumentation framework and an argument, an argument game can determine whether that argument is a member of some extension. Two well-known argument games are the G-game for grounded semantics and the P-game for preferred credulous semantics. For e-p-semantics, there is an argument game called the GP-game, which is based on the G-game and the P-game.

3.3.1 General properties

An argument game is a sequence of moves. Two players, PRO and CON, move in turn. In the first move, PRO puts forward the main argument of the game. In every following move, an argument is put forward that defeats the argument of an earlier move made by the other player (the target of this move). Every type of argument game has its own additional rules. If one of the players has no legal moves left, the other player wins the game. If PRO can win the game no matter what moves CON chooses to make, then the argument of the first move is in an extension associated with the type of game that is played and vice versa (soundness and completeness of the game). This was proven for the G-game and grounded semantics by [14], for the P-game and preferred credulous semantics by [20], and for the GP-game and e-p-semantics by [13].

Below are the formal definitions of moves and argument games.

Definition 15 (Move)**.** *A* move *has six properties:*

- *an* identifier
- *a* player*:* PRO *or* CON
- *a* role*: the role of the player of this move,* P *or* O
- *an* argument *put forward in this move*
- *a* target*: the move to which this move replies (sometimes indicated by its identifier)*
- *whether this move has been* backtracked

A move m is said to defeat a move n if the argument put forward in m defeats the argument put forward in n.

Definition 16 (Argument game)**.** *An* argument game *is a sequence of moves obeying the following rules:*

- *The first move is made by player* PRO *with role* P
- PRO *and* CON *move in turn*
- *All moves (except the first) have a previous move in the game made by the other player as their target*
- *All moves (except the first) defeat their target*

- *Two moves with the same target cannot have the same argument*
- *The target of a move is the most recent move of the other player that it can legally target*
- *If the target of a move by* P *is not the directly preceding move, all moves between the target and this move are marked as backtracked*
- *A player wins the game iff the other player has no legal moves left*

The games that are presented here are slightly different versions of the original games. The major difference is that players are now allowed to backtrack, but only if it is necessary (if they cannot target the directly preceding move), and then only to the nearest possible point. Moreover, every time P backtracks the game tree is marked as such from the point where the unsuccessful move (the move with the same target as the current move) was made. Moves that are marked as backtracked are treated as if they were removed from the game; the only thing that must be remembered of them is that the unsuccessful move must not be made again. This backtracking facility is essential for the reasoning algorithm in the implementation.

An argument game can be displayed as a tree, where all nodes are moves. The root is the first move by PRO, and child nodes have their parent as target. Each path through the tree from the root to a leaf node is called a line of dispute.

To make some games shorter, the following rule is added:

- *If a player repeats his own argument in the same line of dispute, this line of dispute is ended (the other player may not target this move, only backtrack if he can)*

This rule does not change the outcome of a game. If without this rule the other player has a reply to the last move, he will have the same reply to the move whose argument it repeats, so one of the replies is redundant. The only exception is when the other player has already moved his reply targeting the move whose argument was repeated. However, this situation can only occur if both participants in the game are allowed to repeat their arguments in the same line of dispute, which is never the case in the three specific argument games that are used here.

Note also that all games are finite, since the set of arguments is finite, two moves with the same target cannot have the same argument, and in the three specific argument games that are used always at least one participant is not allowed to repeat his arguments in the same line of dispute.

3.3.2 The G-game

The G-game for grounded semantics is defined as follows:

Definition 17 (G-game)**.** *A* G-game *is an argument game that additionally obeys the following rules:*

- *Player* PRO *always has role* P
- *Player* CON *always has role* O

- *A player with role* P *does not repeat any arguments (his own or the other player's) in the same line of dispute*

Proposition 18. *This game is sound and complete with respect to grounded semantics in the sense that* PRO *wins a game for argument A if and only if A is a member of the grounded extension.*

In the original G-game [14] backtracking is not allowed, and soundness and completeness are defined in the sense of having a winning strategy, as opposed to winning a single game. Because of these differences, the original proofs of soundness and completeness have been adapted to this new definition of the G-game.

Proof of soundness. Suppose PRO has won the game for A. It will be shown that all arguments of PRO's non-backtracked moves, in particular A, are in F^ω and hence in the grounded extension. If PRO won, then CON has no legal moves left, so CON has already moved every defeater of every previous non-backtracked move of PRO. Every non-backtracked branch in the game tree ends with a move of PRO, since PRO has never run out of legal moves, and if he backtracked, that branch was marked as such. Since there are no defeaters of the non-backtracked leaves, these arguments of PRO are acceptable with respect to the empty set, so in F^1 and hence in the grounded extension. Consider now the targets of the targets of the leaf arguments. If they have just the one defeater, or if all defeaters are defeated directly by a leaf argument, they are in F^2. Unless PRO won in one move (in which case his argument A is undefeated and hence in the grounded extension), there is always at least one such argument, namely the target of the target of the leaf argument of the longest branch. This line of reasoning can be repeated until the root argument A is reached, so A is in F^ω and hence in the grounded extension. □

Proof of completeness. Suppose argument A is a member of the grounded extension. Then A is acceptable with respect to the grounded extension, so every defeater of A is defeated by an argument in the grounded extension. So PRO can reply to every move of CON that targets A with an argument from the grounded extension. If he does so, then this line of reasoning can be repeated until PRO moves an undefeated argument which CON cannot target (recall that the set of arguments $\mathcal{A}$ is always finite). If PRO only moves arguments from the grounded extension, which he can, he wins without backtracking. So the only case in which PRO might not be able to end a line of dispute, is if he moves an argument that is not in the grounded extension. But in that case he can always backtrack, since we have seen that he can always reply to a move with at least one argument from the grounded extension (which he did not play before if he replied with another argument first). □

3.3.3 The P-game

The P-game for preferred credulous semantics is defined as follows:

Definition 19 (P-game)**.** *A* P-game *is an argument game that additionally obeys the following rules:*

- *Player* PRO *always has role* P

- *Player* CON *always has role* O
- *A player with role* P *does not repeat the other player's arguments from non-backtracked moves*
- *A player with role* O *does not repeat his own arguments in the same line of dispute*
- *If a player with role* O *repeats an argument from a non-backtracked move of the other player (a so-called eo ipso move), this line of dispute is ended (the other player may not target this move, only backtrack if he can)*

Proposition 20. *This game is sound and complete with respect to preferred credulous semantics in the sense that* PRO *wins a game for argument A if and only if A is a member of some preferred extension.*

Although the P-game is more like its original version than the G-game, there are still some differences. In the original P-game [20] PRO is not allowed to backtrack after an eo ipso move, and soundness and completeness are defined in the sense of having a winning strategy, as opposed to winning a single game. Because of these differences, the original proofs of soundness and completeness have been adapted to this new definition of the P-game.

Proof of soundness. Suppose PRO has won the game for A. Let $\mathcal{A}'$ denote all non-backtracked arguments that PRO moved in this game (in particular, $A \in \mathcal{A}'$). It will be shown that $\mathcal{A}'$ is an admissible set, and hence a subset of a preferred extension. If $\mathcal{A}'$ is not conflict-free, one of the arguments in $\mathcal{A}'$ is defeated by another argument in $\mathcal{A}'$ and CON would have done an eo ipso move, after which either CON would have won, which is not the case, or PRO would have backtracked, which is also not the case. So $\mathcal{A}'$ is conflict-free. If $\mathcal{A}'$ is not admissible, then some argument in $\mathcal{A}'$ is defeated by an argument B that is not defeated by an argument in $\mathcal{A}'$. In that case, CON would have used B as a winning argument, which is also not the case. Hence $\mathcal{A}'$ is admissible, and a subset of a preferred extension, so A is in a preferred extension. □

Proof of completeness. Suppose argument A is a member of a preferred extension $\mathcal{A}'$. Then A is acceptable with respect to $\mathcal{A}'$, so every defeater of A is defeated by an argument in $\mathcal{A}'$. So PRO can reply to every move of CON that targets A with an argument from $\mathcal{A}'$. If he does so, then this line of reasoning can be repeated until CON has moved all defeaters of all arguments in $\mathcal{A}'$ (recall that the set of arguments $\mathcal{A}$ is always finite) and runs out of legal moves because he may not repeat himself (CON always has role O). If PRO only moves arguments from $\mathcal{A}'$, which he can, he wins without backtracking. So the only case in which PRO might not be able to end a line of dispute, is if he moves an argument that is not in $\mathcal{A}'$. But in that case he can always backtrack, since we have seen that he can always reply to a move with at least one argument from $\mathcal{A}'$ (which he did not play before if he replied with another argument first). □

3.3.4 The GP-game

The GP-game for e-p-semantics is defined as follows:

Definition 21 (GP-game)**.** *A* GP-game *is an argument game that additionally obeys the following rules:*

- *A player with role* P *does not repeat any epistemic arguments (his own or the other player's) in the same line of dispute*
- *A player with role* P *does not repeat the other player's practical arguments from non-backtracked moves*
- *A player with role* O *does not repeat his own practical arguments in the same line of dispute*
- *If a player with role* O *repeats a practical argument of the other player from a non-backtracked move (a so-called eo ipso move), this line of dispute is ended (the other player may not target this move, only backtrack if he can)*
- *If an epistemic argument is put forward in a move and a practical argument was put forward in the target of this move, then the role of this move is* O*; otherwise it is* O *if it was* P *in the target and* P *if it was* O *in the target*

Every branch in a GP-game consists of a possibly empty sequence of moves with practical arguments followed by a possibly empty sequence of moves with epistemic arguments. All moves with practical arguments together are a P-game. If there are no moves with practical arguments, the whole GP-game is a G-game. Otherwise all sequences of moves with epistemic arguments whose first moves have the same target, preceded by that target, are a G-game about that target, except that the roles of the players may have switched.

Proposition 22. *This game is sound and complete with respect to e-p-semantics in the sense that* PRO *wins a game for argument A if and only if A is a member of some e-p-extension.*

The proof for this is based on the proofs for the soundness and completeness of the G-game and the P-game.

Chapter 4

The Epistemic and Practical Reasoner

This chapter presents the general layout of the Epistemic and Practical Reasoner. An overview of the program is given in Section 4.1. Section 4.2 presents the user interface. The data structure of a belief base and its elements is discussed in Section 4.3. Finally, Section 4.4 covers the output of the program.

The algorithms for argument and game construction will be discussed in detail in Chapter 5. For more details on the code, the reader is referred to the API documentation. For the user manual of the program, see appendix A.

4.1 Overview

The Epistemic and Practical Reasoner is written in Java 6. Figure 4.1 contains a UML class diagram of the program. This diagram is not exhaustive, but illustrates the core layout of the program. The general flow of the program is as follows. The program starts with the main method of the GUI class, which creates a graphical user interface (see Section 4.2). The user enters or loads a belief base into the editor, enters a query, selects the options he wants, and presses the query button. Clicking the query button starts the Controller. The Controller calls a Parser (which was created from a grammar file with JavaCC [8], a parser generator for use with Java applications) that in turn tries to parse the query into a Formula object, and the facts and rules in the belief base from the editor into Fact and Rule objects, which are then added to a BeliefBase object (see Section 4.3). If this fails, an exception is thrown which is passed back through the Controller to the GUI to notify the user. If the Parser succeeds, it returns a filled BeliefBase and a query Formula to the Controller. Next the Controller starts a Reasoner object, which in turn tries to construct an argument game for the given query with the given semantics. A game is implemented as a TreeMap containing Move objects that are stored under their indentifier. During the construction of the game, ArgumentCreator creates Arguments and stores them in ArgumentArrayLists (for details on these algorithms, see Chapter 5). When the game has been constructed, the Reasoner determines the winner and returns the result to the Controller. Finally, the Controller invokes an OutputWriter, which generates XML and Dot output (see Section 4.4) that is

passed to the GUI and displayed to the user.

The program is a standalone application, but it can also be embedded in another Java program by implementing the UI interface and calling the Conroller.start method.

4.2 User interface

The graphical user interface (GUI) is a standard application window. It has a menu and three tabs, one for input and two for output. The input tab contains a belief base editor, a field where a query can be entered, a button to start the query algorithm, and a log. With the menu, a text file can be loaded into the belief base editor, and the belief base, the log and the output can be saved to files. Also, several query options can be selected in the menu: the semantics to use, whether to use the practical syllogism, whether to use accrual, the strength mechanism to use and the accrual strength mechanism to use. Finally the log level can be set and the documentation can be accessed. The two output tabs display the generated XML file and graph image, respectively. Figures 4.2 and 4.3 show the input tab and graph tab of the graphical user interface with the example from Section 6.1.1.

The Java package java.util.logging provides for the logging mechanism that is used here. There is one Logger object for the entire program. At various points in the code, log messages are sent to the Logger. Whether the message actually gets logged depends on the level of the message and the level of the Logger. If the message level is as least as high as the log level, the message gets logged and will be displayed in the log of the GUI. The log level can be set by the user in the GUI menu; it defaults to info level.

4.3 Belief base data structure

The facts and rules that are parsed from the belief base are modelled as instances of the classes Fact and Rule, which are both subclasses of the abstract class AbstractRule. Their consequents and a Rule's antecedents are instances of the class Formula. A Formula object has a Literal, a Type, and may be negated. A Literal object has an atom string and may be negated.

Two Formula or Literal objects that have the same values for all fields cannot exist. To ensure this, a Formula or Literal object can only be obtained by calling one of the static methods Formula.getFormula or Literal.getLiteral. These methods check whether the desired object already exists; if so, this object is returned, and only if this is not the case, a new object is created. This feature is needed because Formulas are used as keys in HashMaps by BeliefBase and ArgumentCreator; keys are compared with the equals method, which by default only returns true if the parameter object is the same as the object on which the method is applied: object1.equals(object2) iff object1 == object2. It is possible to override this method, but that is tricky and it is discouraged unless it is the only option. Therefore this other solution to the problem has been applied.

Every time the Parser has parsed a fact or rule, a Fact or Rule object is created and added to a BeliefBase. When a Fact is added, it is stored under its consequent Formula in a HashMap called factBase. The adding of a Rule is

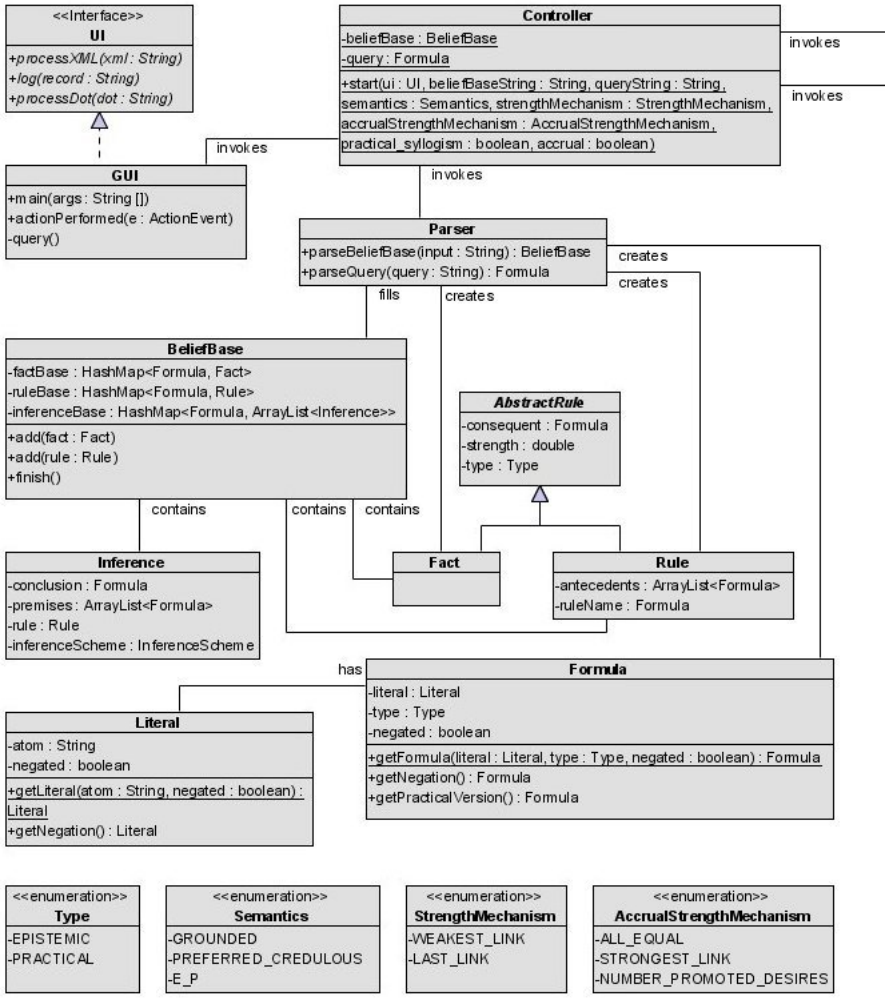


Figure 4.1: Class diagram

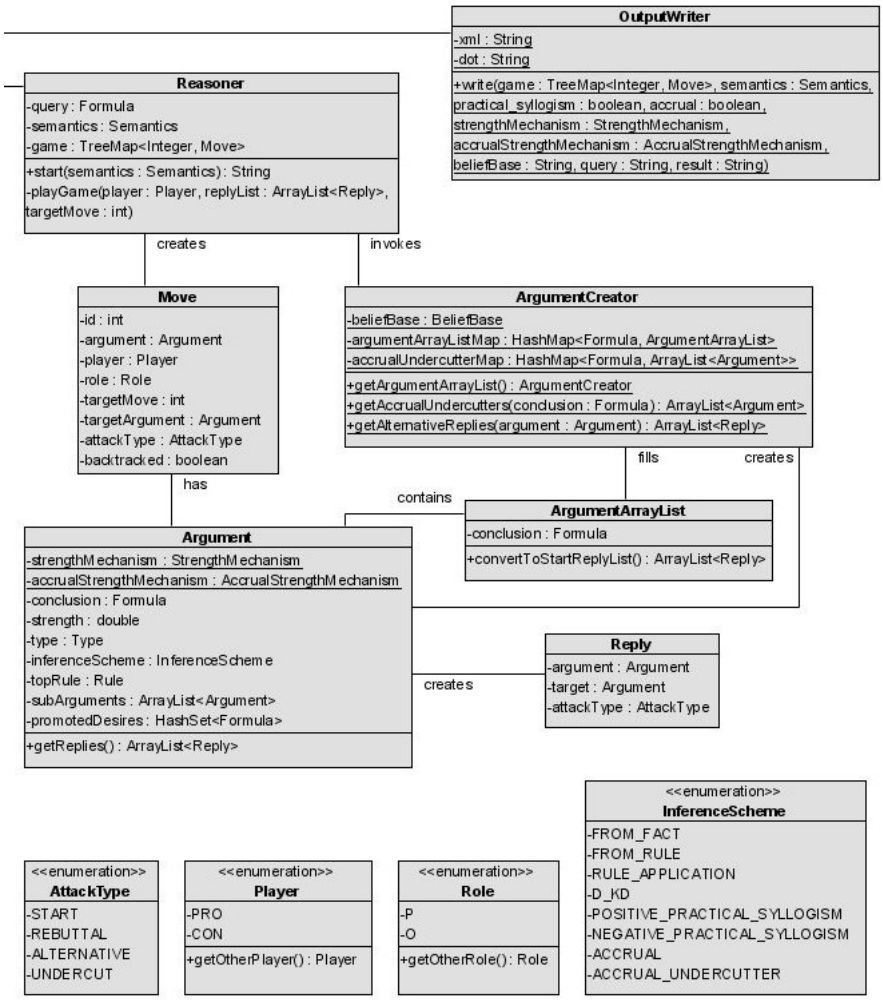


Figure 4.1 (continued)

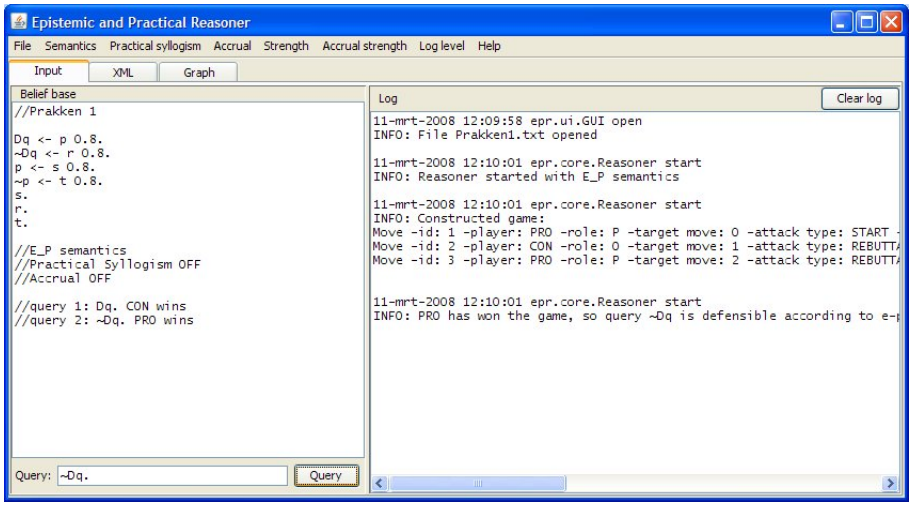


Figure 4.2: User interface, input tab

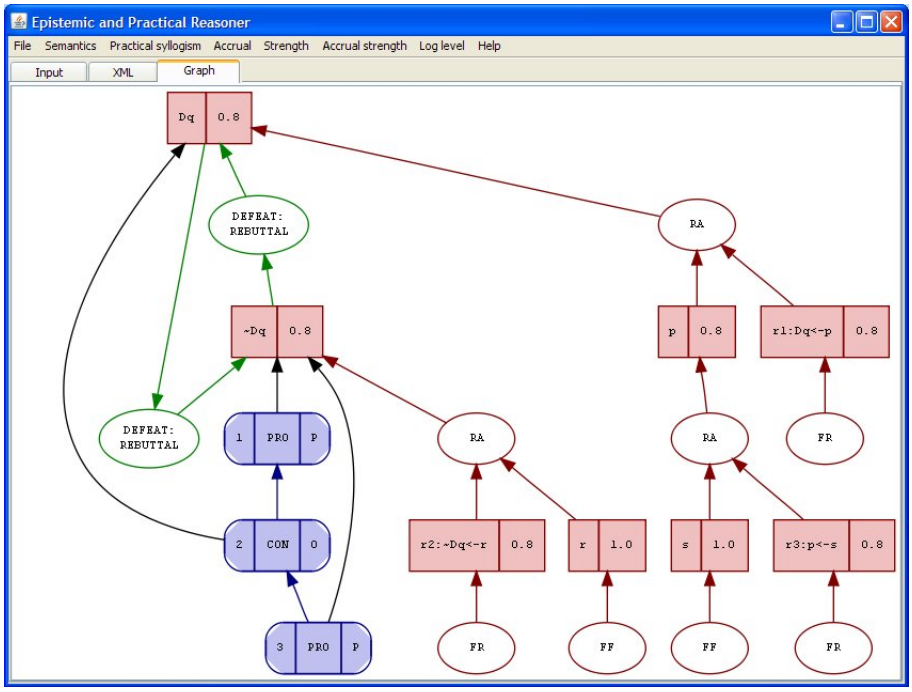


Figure 4.3: User interface, graph tab

more complicated. If the Rule already has a rule name, it is stored under that name in a HashMap called ruleBase. If not, it is stored temporarily in a queue. Later, the finish method will assign it a rule name and store the Rule in the ruleBase. Moreover, when a Rule is added, all inferences that can be made with it are modelled as instances of the class Inference. Inferences that can be made with a rule are instantiations of one of the inference schemes rule application, positive practical syllogism, and negative practical syllogism. The Inferences that are made are stored in a HashMap called inferenceBase; it contains lists of Inferences with the same conclusion stored under that conclusion. The factBase, ruleBase and inferenceBase will be used by ArgumentCreator when Arguments are being constructed.

Besides storing Facts, Rules and Inferences, BeliefBase checks that no fact or rule name occurs more than once in the belief base, otherwise it throws an exception.

4.4 Output

Besides the log, which displays the result of a query and the game that was played, the program generates two kinds of output: an XML file and a graph image.

The XML output conforms to an extension of the AIF Schema, as developed by ASPIC [3]. This schema is a reification of the Argument Interchange Format proposed by Chesñevar et al. [5]. They propose to represent an argument network as a directed graph that consists of two kinds of nodes and edges between them. I-nodes are information nodes and relate to content, S-nodes are scheme application nodes. Multiple types of schemes can be defined, for example inference schemes, preference schemes and conflict schemes.

The ASPIC AIF Schema defines an XML language in which such an argument network can be described. I extended this schema to include some extra features of the program[1]. In the context node the provider, the input, and the result of the query are specified. Within the input, a distinction is made between the belief base, the query, and the chosen values of the query parameters. I-nodes, which essentially contain the formulas that are part of the generated arguments, are extended to contain a strength. A new S-node type, extending the original S-node type, is defined to accommodate moves. S-nodes that represent a move have elements moveID, player, role and backtracked. The argument and target of a move node are related to it with edges.

The graph image is a visualisation of the data in the XML file for human readability. It is created by Graphviz [7] from the Dot output that is generated by OutputWriter.

[1] This extended XML Schema can be found in `epr/xml/EPR.AIF.xsd`.

Chapter 5

Algorithms

In this chapter the more interesting algorithms of the Epistemic and Practical Reasoner are discussed in detail. Section 5.1 starts with the construction of arguments. Then the construction of argument games will be discussed in Section 5.2.

5.1 Construction of arguments

5.1.1 Global approach

Arguments are created by the class ArgumentCreator and stored in ArgumentArrayLists. An ArgumentArrayList contains all arguments that can be constructed for a given conclusion from a given belief base. An ArgumentArrayList can be acquired by calling the static method ArgumentCreator.getArgumentArrayList and passing it the desired conclusion. If an ArgumentArrayList for this conclusion already exists, this ArgumentArrayList is returned (each ArgumentArrayList, and each Argument, is only created once). Else a new ArgumentArrayList for this conclusion is created and filled. All Arguments for a given conclusion are always created simultaneously. If accrual is used, the accrual undercutters corresponding to the accrual arguments (which are arguments with the same subarguments but without a conclusion) are created at the same time and stored in an ArrayList, which can be retrieved by calling ArgumentCreator.getAccrualUndercutters.

Each argument is constructed top down and recursively, from the conclusion to the premises. Given a desired conclusion, the belief base is searched for an inference with this conclusion. Then arguments are tried to be constructed for every premise of the found inference. This goes on recursively until a fact or rule from the belief base is reached in every branch of the argument tree. If this process is successful, an argument is created. For accrual arguments, the procedure is a little more complicated.

5.1.2 Algorithm

The exact algorithm is listed in Listing 5.1. This is the fill method of the ArgumentCreator class, a private method that is called by ArgumentCreator.getArgumentArrayList if the wanted ArgumentArrayList is not created yet.

Listing 5.1: ArgumentCreator.fill

```
1  private static void fill(ArgumentArrayList list, ArrayList<Argument> undercutterList)
2  throws Exception {
3    Formula conclusion = list.getConclusion();
4    ArrayList<ArrayList<Argument>> elements = new ArrayList<ArrayList<Argument>>();
5    ArrayList<Argument> listOrWrapper = list;
6    //FROM_FACT
7    Fact fact = beliefBase.getFactBase().get(conclusion);
8    if (fact != null) {
9      if (accrual) {listOrWrapper = new ArrayList<Argument>();}
10     listOrWrapper.add(new Argument(fact));
11     if (accrual) {elements.add(listOrWrapper);}
12   }
13   //FROM_RULE
14   Rule rule = beliefBase.getRuleBase().get(conclusion);
15   if (rule != null) {
16     if (accrual) {listOrWrapper = new ArrayList<Argument>();}
17     listOrWrapper.add(new Argument(rule));
18     if (accrual) {elements.add(listOrWrapper);}
19   }
20   //D_KD
21   if (conclusion.getType() == Type.PRACTICAL && conclusion.isNegated() == true) {
22     Formula premise = Formula.getFormula(conclusion.getLiteral().getNegation(),
23       Type.PRACTICAL,false);
24     ArgumentArrayList premiseArgumentArrayList =
25       ArgumentCreator.getArgumentArrayList(premise);
26     if (!premiseArgumentArrayList.isEmpty()) {
27       if (accrual) {listOrWrapper = new ArrayList<Argument>();}
28       for (Argument subargument : premiseArgumentArrayList) {
29         ArrayList<Argument> subarguments = new ArrayList<Argument>();
30         subarguments.add(subargument);
31         listOrWrapper.add(new Argument(conclusion,InferenceScheme.D_KD,subarguments,null));
32       }
33       if (accrual) {elements.add(listOrWrapper);}
34     }
35   }
36   //RULE_APPLICATION, POSITIVE_PRACTICAL_SYLLOGISM,
37   //NEGATIVE_PRACTICAL_SYLLOGISM
38   ArrayList<Inference> inferenceList = beliefBase.getInferenceBase().get(conclusion);
39   if (inferenceList != null) {
40     inferenceIteration: for (Inference inference : inferenceList) {
41       ArrayList<ArgumentArrayList> subArgumentArrayLists =
42         new ArrayList<ArgumentArrayList>();
43       for (Formula premise : inference.getPremises()) {
44         ArgumentArrayList premiseArgumentArrayList =
45           ArgumentCreator.getArgumentArrayList(premise);
46         if (premiseArgumentArrayList.isEmpty()) {
47           continue inferenceIteration;
48         } else {
49           subArgumentArrayLists.add(premiseArgumentArrayList);
50         }
51       }
52       subArgumentArrayLists.add(ArgumentCreator.getArgumentArrayList(
53         inference.getRule().getRuleName()));
54       if (accrual) {listOrWrapper = new ArrayList<Argument>();}
55       makeSubargumentCombinations(listOrWrapper,conclusion,inference.getInferenceScheme(),
56         subArgumentArrayLists,inference.getRule(),new ArrayList<Argument>(),0);
57       if (accrual) {elements.add(listOrWrapper);}
58     }
59   }
60   //ACCRUAL, ACCRUAL_UNDERCUTTER
61   if (accrual) {
62     ArrayList<ArrayList<Argument>> combinations = makeAccrualCombinations(elements);
63     for (ArrayList<Argument> subargs : combinations) {
64       list.add(new Argument(conclusion, InferenceScheme.ACCRUAL, subargs, null));
65       undercutterList.add(new Argument(subargs));
66     }
67   }
68 }
```

The parameters of the fill method are the ArgumentArrayList that is to be filled and the corresponding accrual undercutter list. First some variables are declared. The variable conclusion in line 3 is the conclusion for which arguments are tried to be constructed. The variable elements in line 4 will store lists of possible elements of accrual arguments if accrual is used. The variable listOrWrapper in line 5 is a list to which the generated arguments will be added. If accrual is not used, this list is the ArgumentArrayList that is being filled. If accrual is used, listOrWrapper is a different wrapper ArrayList for every different inference, and will be stored in elements; the arguments in the wrappers in elements will be combined into accrual arguments and at the end of this method these will be added to the ArgumentArrayList that is being filled.

Now the algorithm will search for possible inferences with the desired conclusion. In lines 7 to 12, it is checked whether the conclusion can be derived directly from a fact in the belief base. If so, an argument with inference scheme from fact is added. Lines 14 to 19 do the same with inference scheme from rule. If the conclusion is of the right type to be the conclusion of inference scheme DKD, all possible arguments with this inference scheme are added in lines 21 to 35. In lines 38 to 59, for every Inference with the right conclusion in the inferenceBase of the BeliefBase (recall from Chapter 4, Section 4.3 that this contains inferences with schemes rule application, positive practical syllogism and negative practical syllogism), the ArgumentArrayLists of the premises and the rule (which is strictly speaking also a premise) are collected. If one of them is empty (if there are no arguments for some premise), the Inference is skipped. Otherwise all possible combinations of subarguments are made by the method makeSubargumentCombinations (not listed), and the corresponding arguments are added. If accrual is not used, all arguments have now been added to the ArgumentArrayList, and the method is finished.

If accrual is used, the arguments generated so far will become the elements (subarguments) of accrual arguments. Recall that no two subarguments of an accrual may have the same top inference (see Chapter 3, Section 3.1.3). This is the reason that in the variable elements all arguments with the same inference are stored together in a wrapper. The method makeAccrualCombinations (not listed) makes all possible combinations of subarguments with zero or one occurrence of every inference. The arguments (both with top inference scheme accrual and with top inference scheme accrual undercutter) that can be made with these lists of subarguments are added in lines 61 to 67.

5.2 Construction of argument games

5.2.1 Global approach

Argument games are created by the playGame method of the Reasoner class. Basically, the algorithm is a recursive iteration over lists of replies. A Reply object contains an argument, and it has an attack type and a target argument. For each argument, a list of replies (containing its attackers) can be obtained by calling its getReplies method. An attack type can be rebuttal, alternative, undercut, or the dummy type start, which is used for the first move. Before the first call to playGame, the ArgumentArrayList for the query is turned into a list

of replies with attack type start (except when it is empty, in that case no game can be played because there are no arguments for the query). Then playGame is started with player PRO, the list of start replies, and target 0. The first move contains the argument of one of the start replies. For every next move, the player is switched, and a reply is considered to the last move that has replies left that were not tried before. If the argument of the reply can be used as a legal move according to the rules of the specific game that is played, that move is added to the game. Otherwise the next possible reply is considered, until one of the players has no possible replies left, in which case the game is ended.

5.2.2 Algorithm

The playGame method of the Reasoner class is listed in Listing 5.2. The method starts in line 3 by iterating over the replies. Until line 46, the rules of the specific argument game that is played are applied; if the argument of the reply that is currently considered cannot be used in a legal move, the reply is skipped and the next one will be considered.

If the given player was the last player to move, it is not his turn and the game is stopped (lines 4 to 6). The first move has role P (lines 10-11). In the GP-game, if an epistemic argument is put forward in this move and a practical argument was put forward in the target of this move, then the role of this move is O; otherwise it is O if it was P in the target and P if it was O in the target (lines 12-15). By default, roles alternate (lines 16-18). In the G-game and in the epistemic part of the GP-game, a player with role P may not repeat any arguments in the same line of dispute (lines 19-24). In the P-game and in the practical part of the GP-game, a player with role P may not repeat the other player's arguments (lines 25-29), and a player with role O may not repeat his own arguments in the same line of dispute (lines 30-33). Moves must defeat their target; if the attack type is rebuttal or alternative, the argument must be at least as strong as the target argument (lines 34-40). If the target of a move by P is not the directly preceding move, all moves between the target and this move are marked as backtracked (lines 41-46).

If the argument of the current reply satisfies all requirements, a new move containing this argument is added to the game (lines 47-50). In the P-game and in the practical part of the GP-game, if a player with role O repeats an argument of the other player (a so-called eo ipso move), this line of dispute is ended (lines 51-56). If a player repeats his own argument in the same line of dispute, this line of dispute is ended (lines 57-59). If the line of dispute is not ended, playGame is called again, now with the other player and a list of replies to the argument that was just moved. Since this recursive call is made inside the iteration over replies, games are created depth-first.

Listing 5.2: Reasoner.playGame

```
1  private void playGame(Player player, ArrayList<Reply> replyList, int targetMove)
2  throws Exception {
3    replyLoop: for (Reply reply : replyList) {
4      if (isLastPlayer(player)) {
5        return;
6      }
7      Argument argument = reply.getArgument();
8      AttackType attackType = reply.getAttackType();
9      Role role;
10     if (targetMove == 0) {
11       role = Role.P;
12     } else if (semantics == Semantics.E_P &&
13       game.get(targetMove).getArgument().getType() == Type.PRACTICAL &&
14       argument.getType() == Type.EPISTEMIC) {
15         role = Role.O;
16     } else {
17       role = game.get(targetMove).getRole().getOtherRole();
18     }
19     if (semantics == Semantics.GROUNDED ||
20     (semantics == Semantics.E_P && argument.getType() == Type.EPISTEMIC)) {
21       if (role == Role.P && playedBeforeInThisLine(argument, targetMove)) {
22         continue replyLoop;
23       }
24     }
25     if (semantics == Semantics.PREFERRED_CREDULOUS ||
26     (semantics == Semantics.E_P && argument.getType() == Type.PRACTICAL)) {
27       if (role == Role.P && playedBefore(argument, player.getOtherPlayer())) {
28         continue replyLoop;
29       }
30       if (role == Role.O && playedBeforeInThisLine(argument, targetMove, player)) {
31         continue replyLoop;
32       }
33     }
34     if (attackType == AttackType.REBUTTAL ||
35     attackType == AttackType.ALTERNATIVE) {
36       double strengthNeeded = reply.getTarget().getStrength();
37       if (argument.getStrength() < strengthNeeded) {
38         continue replyLoop;
39       }
40     }
41     int thisMoveID = moveID;
42     if (role == Role.P && thisMoveID != targetMove+1) {
43       for (int i = targetMove+1; i < thisMoveID; i++) {
44         game.get(i).backtrack();
45       }
46     }
47     Move move = new Move(thisMoveID,player,role,argument,targetMove,reply.getTarget(),
48     attackType);
49     game.put(thisMoveID,move);
50     moveID++;
51     if (semantics == Semantics.PREFERRED_CREDULOUS ||
52     (semantics == Semantics.E_P && argument.getType() == Type.PRACTICAL)) {
53       if (role == Role.O && playedBefore(argument, player.getOtherPlayer())) {
54         return;
55       }
56     }
57     if (playedBeforeInThisLine(argument, targetMove, player)) {
58       return;
59     }
60     Player otherPlayer = player.getOtherPlayer();
61     ArrayList<Reply> nextReplyList = argument.getReplies();
62     playGame(otherPlayer,nextReplyList,thisMoveID);
63   }
64 }
```

Chapter 6

Examples

This chapter discusses some examples, mostly taken from the literature, and describes the behaviour of the program compared to the original examples. This also illustrates one benefit of implementation, since any difference between the original examples and the behaviour of the program may indicate a flaw in the original example or theory. Section 6.1 deals with examples that illustrate e-p-semantics, Section 6.2 with examples that illustrate the practical syllogism and accrual[1].

6.1 E-p-semantics

The following examples are taken from Prakken's paper on e-p-semantics [13], section 5. For each example, first the original example is presented and then the program's treatment of the same example is presented and discussed.

6.1.1 Example 1

The first three examples use an inference scheme like rule application. This is the belief base: $p \Rightarrow \mathrm{D}q$, $r \Rightarrow \neg\mathrm{D}q$, $s \Rightarrow p$, $t \Rightarrow \neg p$, s, r, t.

According to Prakken the following GP-games can be played for Dq:

$\textsc{pro}_1[\textsc{p}]$: s, $s \Rightarrow p$, $p \Rightarrow \mathrm{D}q$, so D$q$
$\textsc{con}_1[\textsc{o}]$: r, $r \Rightarrow \neg\mathrm{D}q$, so $\neg\mathrm{D}q$
$\textsc{pro}_2[\textsc{p}]$: repeats $\textsc{pro}_1$

$\textsc{pro}_1[\textsc{p}]$: s, $s \Rightarrow p$, $p \Rightarrow \mathrm{D}q$, so D$q$
$\textsc{con}'_1[\textsc{o}]$: t, $t \Rightarrow \neg p$, so $\neg p$
$\textsc{pro}'_2[\textsc{p}]$: s, $s \Rightarrow p$, so p
$\textsc{con}'_2[\textsc{o}]$: repeats $\textsc{con}'_1$

CON lost the first game with only p-arguments but won the second game with a g-argument, so PRO has no winning strategy for Dq. However, he has one for $\neg\mathrm{D}q$:

[1] All examples of this chapter can be found in the folder `epr/examples`.

PRO'_1[P]: r, $r \Rightarrow \neg \text{D}q$, so $\neg \text{D}q$
CON''_1[O]: s, $s \Rightarrow p$, $p \Rightarrow \text{D}q$, so $\text{D}q$
PRO''_2[P]: repeats PRO'_1

Prakken concludes that the only action alternative with justified support is for $\neg \text{D}q$. This agrees with the semantics: the e-arguments PRO'_2 and CON'_1 defeat each other so they are not in the grounded extension of AF. Then they are not in $\mathcal{A}_g$ so the p-argument PRO_1, which has PRO'_2 as a subargument, is also not in $\mathcal{A}_g$. So AF has a unique e-p-extension, containing $\text{CON}_1 = \text{PRO}'_2$.

The program uses the following belief base (see also Figure 4.2):

```
Dq<-p 0.8.    p<-s 0.8.    s.   t.
~Dq<-r 0.8.   ~p<-t 0.8.   r.
```

With query `Dq` the following game is played:

1. PRO/P: `s`, `p<-s`, `Dq<-p`, so `Dq`
2. CON/O →1: `r`, `~Dq<-r`, so `~Dq`
3. PRO/P →2: repeats 1
4. CON/O →1: `t`, `~p<-t`, so `~p`
5. PRO/P →3: `s`, `p<-s`, so `p`
4. CON/O →4: repeats 4

This game combines the two games that were played in the original example. The first three moves are exactly the same as in the first game. Then CON backtracks, he plays his good move from the second game and wins. Note that CON only backtracked because of the extra rule to keep games shorter, otherwise he would have targeted the third move. Prakken does not use this rule, so it appears that the first game of the original example was not terminated, and PRO did not win at all.

With query `~Dq` the following game is played (see also Figure 4.3):

1. PRO/P: `r`, `~Dq<-r`, so `~Dq`
2. CON/O →1: `s`, `p<-s`, `Dq<-p`, so `Dq`
3. PRO/P →2: repeats 1

This game is exactly the same as in the original example.

6.1.2 Example 2

The second example uses the same belief base as the first, extended with fact p. Now PRO also has a winning strategy for $\text{D}q$, since Prakken assumes that purely factual arguments cannot be defeated, so CON cannot now win as in the second game above:

PRO_1[P]: s, $s \Rightarrow p$, $p \Rightarrow \text{D}q$, so $\text{D}q$
CON'''_1[O]: t, $t \Rightarrow \neg p$, so $\neg p$
PRO'''_2[P]: p

Prakken observes that a choice must still be made what to do since the trivial winning strategy for $\neg \text{D}q$ still stands. Again this agrees with the semantics:

PRO$_1$'s subargument PRO$'_2$ is now in the grounded extension of AF so PRO$_1$ is in $\mathcal{A}_g$. Since PRO$_1$ defeats its only defeater in $\mathcal{A}_g$, which is CON$_1$, there are now two e-p-extensions, one containing PRO$_1$ and the other containing CON$_1$.

For the program, the belief base is also extended with fact `p`. With query `Dq` the following game is played:

1. PRO/P: `p`, `Dq<-p`, so `Dq`
2. CON/O →1: `r`, `~Dq<-r`, so `~Dq`
3. PRO/P →2: repeats 1

The first move of this game contains a different argument than in the original example; it uses the fact `p` as a subargument instead of an argument with the fact `s` and the rule `p<-s`. This is due to the fact that the algorithm searches first for facts, and then for other inferences. Both arguments are generated, but the one that was generated second is not used in the game because PRO has already won and does not need it. Because this other argument is used, CON cannot defeat it with the argument for `~p`, since it is not strong enough.

6.1.3 Example 3

The third example illustrates a role switch. This is the belief base: $p \Rightarrow \mathrm{D}q$, $r \Rightarrow \neg\mathrm{D}q$, $s \Rightarrow r$, $t \Rightarrow \neg r$, p, s, t.

PRO has a winning strategy which includes a role switch:

PRO$_1$[P]: p, $p \Rightarrow \mathrm{D}q$, so $\mathrm{D}q$
CON$_1$[O]: s, $s \Rightarrow r$, $r \Rightarrow \neg\mathrm{D}q$, so $\neg\mathrm{D}q$
PRO$_2$[O]: t, $t \Rightarrow \neg r$, so $\neg r$
CON$_2$[P]: s, $s \Rightarrow r$, so r
PRO$_3$[O]: repeats PRO$_2$

Since CON now has role P in the game on r, CON is not allowed to repeat CON$_2$ and loses.

The program uses the following belief base:

```
Dq<-p 0.7.    r<-s 0.8.    p.   t.
~Dq<-r 0.8.   ~r<-t 0.8.   s.
```

With query `Dq` the following game is played:

1. PRO/P: `p`, `Dq<-p`, so `Dq`
2. CON/O →1: `s`, `r<-s`, `~Dq<-r`, so `~Dq`
3. PRO/O →2: `t`, `~r<-t`, so `~r`
4. CON/P →3: `s`, `r<-s`, so `r`
5. PRO/O →4: repeats 3

This game is exactly the same as the one in the original example. Note that the argument of the first move is less strong than the argument of the second move. If this were not the case, PRO could have repeated the first argument in the third move and would have won the game without a role switch.

6.1.4 Example 4

The next two examples use an inference scheme like the practical syllogism. This is the belief base: $a_1 \wedge s \Rightarrow p$, $a_2 \Rightarrow p$, $r \Rightarrow s$, $a_1 \Rightarrow \neg p$, $a_2 \Rightarrow \neg q$, r, Dp, Dq.

PRO has a winning strategy for an argument for Da_1:

PRO$_1$[P]: r, $r \Rightarrow s$, so s; also $a_1 \wedge s \Rightarrow p$ and Dp, so Da_1
CON$_1$[O]: $a_2 \Rightarrow p$ and Dp, so Da_2
PRO$_2$[P]: $a_2 \Rightarrow \neg q$ and Dq, so $\neg$Da_2
CON$_2$[O]: $a_1 \Rightarrow \neg p$ and Dp, so $\neg$Da_1
PRO$_3$[P]: repeats PRO$_1$

In a similar way there is a winning strategy for Da_2 so Prakken concludes that there are two e-p-extensions, one with arguments for Da_1 and $\neg$Da_2 and one with arguments for Da_2 and $\neg$Da_1. Note also that $AF_g = AF$ so these are also preferred extensions of AF.

The program uses the following belief base:

```
p<-a1,s.    ~p<-a1.    Dp.
p<-a2.      ~q<-a2.    Dq.
s<-r.       r.
```

With query `Da1` the following game is played:

1. PRO/P: `r`, `s<-r`, `Dp`, `p<-a1,s`, so `Da1`
2. CON/O →1: `Dp`, `~p<-a1`, so `~Da1`
3. PRO/P →2: repeats 1
4. CON/O →1: `Dp`, `p<-a2`, so `Da2`
5. PRO/P →4: `Dq`, `~q<-a2`, so `~Da2`

With query `~Da1` the following game is played:

1. PRO/P: `Dp`, `~p<-a1`, so `~Da1`
2. CON/O →1: `r`, `s<-r`, `Dp`, `p<-a1,s`, so `Da1`
3. PRO/P →2: repeats 1

With query `Da2` the following game is played:

1. PRO/P: `Dp`, `p<-a2`, so `Da2`
2. CON/O →1: `Dq`, `~q<-a2`, so `~Da2`
3. PRO/P →2: repeats 1
4. CON/O →1: `r`, `s<-r`, `Dp`, `p<-a1,s`, so `Da1`
5. PRO/P →4: `Dp`, `~p<-a1`, so `~Da1`

With query `~Da2` the following game is played:

1. PRO/P: `Dq`, `~q<-a2`, so `~Da2`
2. CON/O →1: `Dp`, `p<-a2`, so `Da2`
3. PRO/P →2: repeats 1

The first of these games resembles the game from the original example. The only difference is the order in which CON moves the two defeaters of the first argument.

In this case there are indeed two e-p-extensions, one containing arguments for `Da1` and `~Da2` and one containing arguments for `Da2` and `~Da1`. But there is a third e-p-extension that contains the arguments for `~Da1` and `~Da2`.

6.1.5 Example 5

This example uses the same belief base as the fourth, extended with rule $t \Rightarrow \neg s$ and fact t. Then in the argument game for $\mathrm{D}a_1$ CON can win by attacking PRO_1 with an e-argument for $\neg s$, defeating its subargument for s (but also defeated by it). Prakken concludes that the arguments for s and $\neg s$ are both not in $\mathcal{A}_g$, so PRO_1 is also not in $\mathcal{A}_g$ so there is a unique e-p-extension, containing arguments for $\mathrm{D}a_2$ and $\neg\mathrm{D}a_1$.

The program does exactly what is said in the original example: CON adds a move that attacks the argument for `Da1` with an argument for `~s`. Then PRO moves an argument for `s`, but CON repeats his move and wins. Thus PRO has no winning strategy for `Da1` anymore. However, he still has winning strategies for `Da2`, `~Da1` and `~Da2`. Only the arguments for `Da2` and `~Da2` defeat each other, so there is not one e-p-extension, but two: one containing the arguments for `Da2` and `~Da1`, and one containing the arguments for `~Da1` and `~Da2`.

6.2 The practical syllogism and accrual

6.2.1 Simple example

This example illustrates alternative defeat and accrual undercut. This is the belief base:

```
b<-a.   b<-d.   Db.
c<-a.   Dc.
```

Three accrual arguments can be made for `Da`, and one accrual undercutter:

A:
$$\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{Db}}\,\text{FF}}{\texttt{Db}}\,\text{ACCR} \quad \dfrac{\dfrac{}{\texttt{r1:b<-a}}\,\text{FR}}{\texttt{r1:b<-a}}\,\text{ACCR}}{\texttt{Da}}\,\text{PPS} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{Dc}}\,\text{FF}}{\texttt{Dc}}\,\text{ACCR} \quad \dfrac{\dfrac{}{\texttt{r2:c<-a}}\,\text{FR}}{\texttt{r2:c<-a}}\,\text{ACCR}}{\texttt{Da}}\,\text{PPS}}{\texttt{Da}}\,\text{ACCR}$$

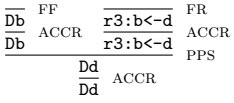
B:
$$\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{Db}}\,\text{FF}}{\texttt{Db}}\,\text{ACCR} \quad \dfrac{\dfrac{}{\texttt{r1:b<-a}}\,\text{FR}}{\texttt{r1:b<-a}}\,\text{ACCR}}{\texttt{Da}}\,\text{PPS}}{\texttt{Da}}\,\text{ACCR}$$

C:
$$\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{Dc}}\,\text{FF}}{\texttt{Dc}}\,\text{ACCR} \quad \dfrac{\dfrac{}{\texttt{r2:c<-a}}\,\text{FR}}{\texttt{r2:c<-a}}\,\text{ACCR}}{\texttt{Da}}\,\text{PPS}}{\texttt{Da}}\,\text{ACCR}$$

D:
$$\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{Db}}\,\text{FF}}{\texttt{Db}}\,\text{ACCR} \quad \dfrac{\dfrac{}{\texttt{r1:b<-a}}\,\text{FR}}{\texttt{r1:b<-a}}\,\text{ACCR}}{\texttt{Da}}\,\text{PPS} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{Dc}}\,\text{FF}}{\texttt{Dc}}\,\text{ACCR} \quad \dfrac{\dfrac{}{\texttt{r2:c<-a}}\,\text{FR}}{\texttt{r2:c<-a}}\,\text{ACCR}}{\texttt{Da}}\,\text{PPS}}{\texttt{Da}}\,\text{AU}$$

The accrual arguments that have `Db` as a premise (A and B) are defeated by an alternative argument:

E:



The following game is played for query `Da` (with grounded semantics):

1. PRO/P: A
2. CON/O →1: E (alternative)
3. PRO/P →2: B (alternative)
4. CON/O →3: E (alternative)
5. PRO/P: B
6. CON/O →5: E (alternative)
7. PRO/P →6: A (alternative)
8. CON/O →7: E (alternative)
9. PRO/P: C
10. CON/O →9: D (accrual undercut)

This game has three lines of dispute. In the first two lines, which only differ in the order of the moves, CON attacks with an alternative argument. In the third line he uses the accrual undercutter. Note that this only happens because grounded semantics is used; with preferred credulous or e-p-semantics PRO would have repeated argument A in the third move and won.

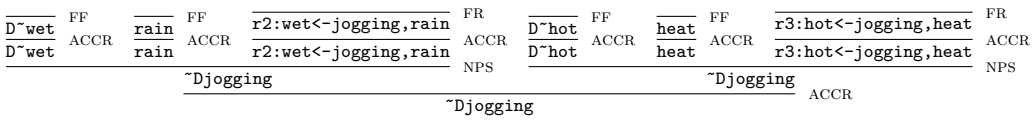
Since in the implementation an accrual can never be weaker than its elements, accrual undercut is never strictly necessary (it does not change the outcome of any games). However, this example shows that the mechanism does work allright.

6.2.2 Jogging example

This example is an adapted version of Prakken's [12] jogging example. It is about a jogger John who wants to stay fit. Jogging fulfills this desire, but it is also hot and raining outside, and John does not want to be hot or wet. The example can be formalised as follows as follows:

```
fit<-jogging.        Dfit.
wet<-jogging,rain.   D~wet.   rain.
hot<-jogging,heat.   D~hot.   heat.
```

With e-p-semantics, the practical syllogism turned on, accrual turned on, and the number of promoted desires as accrual strength mechanism, PRO wins the game for query `~Djogging` in one move with the following argument, which fulfills two desires (`D~wet` and `D~hot`):



There is a rebutting argument, but it is not strong enough to defeat the first argument, because it only fulfills one desire (`Dfit`):

$$\dfrac{\dfrac{\dfrac{}{\texttt{Dfit}}\,\text{FF}}{\texttt{Dfit}}\,\text{ACCR} \qquad \dfrac{\dfrac{}{\texttt{r1:fit<-jogging}}\,\text{FR}}{\texttt{r1:fit<-jogging}}\,\text{ACCR}}{\dfrac{\texttt{Djogging}}{\texttt{Djogging}}\,\text{ACCR}}\,\text{PPS}$$

However, in the original example, the accrual of the two arguments for not wanting to go jogging was supposed to be weaker than each of its elements, so that if it was only hot or only raining, John would not want to go jogging, but if it is both hot and raining, John would want to go jogging anyway. Unfortunately, this implementation cannot handle that.

6.2.3 Judge example

This example is taken from Bench-Capon and Prakken's paper on the practical syllogism and accrual [4]. It is about a judge who must determine the best way to punish (pu) a criminal found guilty. He has three options: imprisonment (pr), a fine (fi) and community service (cs). Besides punishment there are three more goals at stake, deterring the general public (de), rehabilitating the offender (re) and protecting society from crime (pt). The judge must ensure that the offender is punished, and so pu will be the most important goal, but the method of punishment chosen will depend on the other goals that can be achieved by the various methods of punishing the offender. The judge believes that imprisonment promotes both deterrence and protection of society, while it demotes rehabilitation of the offender. He believes that a fine promotes deterrence but has no effect on rehabilitation or the protection of society since the offender would remain free, and he believes that community service has a positive effect on rehabilitation of the offender but a negative effect on deterrence since this punishment is not feared.

This gives the following belief base:

```
pu<-pr.   de<-pr.    de<-fi.    Dpu.   Dre.
pu<-fi.   pt<-pr.    ~de<-cs.   Dpt.
pu<-cs.   ~re<-pr.   re<-cs.    Dde.
```

The interesting queries are `Dpr`, `~Dpr`, `Dfi`, `Dcs`, and `~Dcs`. Arguments can be created for all queries, but the outcome of a game depends on the accrual strength mechanism that is used. If all equal or strongest link is used, PRO wins all games, which can be quite long (up to 125 moves for query `Dfi`). If the number of promoted desires is used, games are shorter (at most 16 moves), and PRO only wins the games for queries `Dpr` and `~Dcs`, since that way the largest number of desires is fulfilled. However, ideally both the promoted and the demoted values would be taken into account, like in the original example. Unfortunately, this implementation cannot do that.

Chapter 7

Discussion and conclusion

7.1 Related research

Within the ASPIC project [2], much work has been done on modelling and implementing argument-based epistemic and practical reasoning. ASPIC Deliverable 2.6 [1] presents two general argumentation systems. The first one is used for epistemic inference. South has implemented this in a Java program called the ASPIC Inference Engine. This program is based on Vreeswijk's Argumentation System [18], which is a prototype implementation of his algorithm that computes minimally grounded and admissible defence sets in argument systems [19]. The Inference Engine allows the user to enter (defeasible) knowledge in a Prolog-like predicate language and to pose a query. The algorithm then tries to construct arguments for and against the query statement and returns their status according to grounded or preferred credulous semantics. The second argumentation system of ASPIC is one for decision making, and contains the epistemic inference system. This second system has also been implemented, as the ASPIC Decision Component, which uses the Inference Engine.

Like the Epistemic and Practical Reasoner, the ASPIC Decision Component is an implementation of argument-based practical reasoning. There are however many differences. Whereas the Epistemic and Practical Reasoner takes one query and determines its status, the ASPIC Decision Component takes a set of possible decisions and recommends some of them, depending on which decision criteria are used. So epistemic and practical reasoning are disjoint, not interleaved as in the Epistemic and Practical Reasoner. The language differs too. ASPIC uses a predicate language without any modal operators, while the language of Epistemic and Practical Reasoner is propositional with a modality D standing for desire. Another difference is that the ASPIC Decision Component only uses grounded or preferred credulous semantics, not e-p-semantics. The ASPIC approach does use some form of the practical syllogism, but since practical arguments are not evaluated according to the semantics of the argument framework, but by separate decision criteria, rebuttal between positive and negative practical syllogism arguments and alternative defeat are not modelled. Accrual of arguments is not modelled at all.

Another approach to argument-based practical reasoning is developed by Pollock [10]. He uses practical reasoning not only to adopt goals, but also to

construct plans. Pollock models plans as ordered sets of goals, subgoals and actions with causal links between actions and (sub)goals. This system has been implemented in the OSCAR architecture for defeasible reasoning [9]. Arguments for plans interact in different ways than arguments for goal or desire adoption, so Pollocks system cannot easily be compared to the Epistemic and Practical Reasoner.

Finally, South et al. [17] have implemented a reasoner that can determine the status of an argument according to grounded or preferred credulous semantics by playing a G-game or a P-game. However, this system does not generate any arguments. Instead, a complete argumentation framework in the sense of Dung [6], with a set of arguments and a defeat relation, must be provided by another application.

7.2 Results

The goal of this implementation was to see whether the implemented theories are correct and completely specified, and if not, what changes and additions are needed to make them better or more complete. This section presents an overview per topic of the various aspects of that topic. It will be discussed which problems arose, how some of these problems were solved, and which problems remain.

7.2.1 E-p-semantics

Like Dung's [6] abstract argumentation framework, Prakken's [13] e-p-argumentation framework is abstract. The internal structure of arguments is unspecified; the only requirement is that the set of arguments can be split into a set of epistemic arguments and a set of practical arguments, based on a distinction between epistemic and practical formulas. Prakken does not specify how this distinction should be made, but in his examples he defines formulas with occurrences of a modal operator D standing for desire as practical, and all other formulas as epistemic. Since such a modal operator is required anyway by Bench-Capon and Prakken's [4] theory, this definition was adopted here too.

As for the internal structure of arguments, a choice has been made to model them as trees of chained defeasible inferences. Basic inference schemes are from fact, from rule and rule application, and practical syllogism and accrual schemes have been added to accommodate the other topics. This again agrees with Prakken's examples: all examples can be modelled in the implemented formalism.

The e-p-semantics and the GP-game are fully specified. A backtracking facility was added to the GP-game to ensure that playing just one game is sufficient to answer a query, instead of needing to find a winning strategy. This was also done for the G-game and the P-game, and since the many similarities between these games and the GP-game, this was quite straightforward.

Some errors were found in Prakken's examples (see Chapter 6, Section 6.1), but these are minor and do not affect the correctness of the theory.

7.2.2 The practical syllogism

A first requirement of Bench-Capon and Prakken's [4] system for practical reasoning with the practical syllogism is the modal operator D. Since the use of this operator is quite restricted (it cannot be nested), it could easily be added to the language. The adoption of the modal operator also provides an easy way to distinguish between epistemic and practical formulas, as was mentioned in Section 7.2.1. The operator D is of type KD insofar as this applies to formulas where D is not nested. This was modelled as an inference scheme (DKD), which works well. Also, defaults (rules) may not contain the modality D. This was modelled as a restriction that the rule application inference scheme may not be used on practical rules. Since other theories do not have this restriction (for example, Prakken [13] uses rule application on practical rules in his examples), it is only enforced when the use of the practical syllogism is turned on.

The positive and negative versions of the practical syllogism are fully specified by Bench-Capon and Prakken and were implemented as the inference schemes positive practical syllogism and negative practical syllogism.

Bench-Capon and Prakken only defined alternative defeat for use with accrual, while the practical syllogism can also be used in systems that do not use accrual. However, the idea behind alternative defeat was very clear, so it was not hard to make an appropriate alternative definition without accrual.

7.2.3 Accrual of arguments

The accrual and accrual undercutter inference schemes are fully specified by Prakken [12], and were implemented as such. Prakken also requires that all conclusions of inferences of other types are labelled with the premises of the applied inference, and that the accrual inference scheme takes any set of labelled versions of a certain formula and produces the unlabelled version. This has been implemented in a different, but equivalent way: all subarguments of an accrual (or accrual undercutter) argument must have a top inference scheme that is not accrual.

A difference between the theory and the implementation is that in the implementation, accrual undercutters do not have a conclusion. This is due to the fact that the language that is used cannot express inferences. This is not very elegant, but it has no consequences for the working of the program.

One point was found upon which the accrual mechanism could be made more efficient: by adding the restriction that accrual cannot be applied to two elements with the same label or top inference (see Chapter 3, Section 3.1.3). If this restriction is used, less arguments can be created, but it has no influence on the result of a query.

The definition of accrual strength turned out to be a major problem (see the discussion in Sections 2.3 and 6.2). Prakken [12] states that in general, the strength of an accrual cannot be computed from the strengths of its elements, but he gives no indication as to how accrual strength could be defined. Bench-Capon and Prakken do give a start of a definition, but do not complete it. To implement accrual strength in a satisfactory way, the way strength is modelled in the current implementation would have to be radically altered. A solution to this problem requires more research.

Furthermore, because so many accrual arguments can be created, argument

games can become very long. This is inherent to the theory and can become a problem if accrual is to be used in real applications. However, since many accrual arguments are similar to each other, some improvement may be gained if argument games are altered to take the internal structure of arguments into account. Further research is needed for this.

7.2.4 Conclusion

No errors were encountered in any of the implemented theories. However, some theories were underspecified. Prakken's theory of e-p-semantics [13] is only underspecified with respect to the internal structure of arguments, and this is explicitly meant to be so. The only thing that was not specified in Bench-Capon and Prakken's theory of the practical syllogism [4] was a definition of alternative defeat without accrual, and this was added without problem. Of the three topics, accrual turned out to be the most problematic. Defining accrual strength may be possible, but this requires more research, and it was not implemented in a satisfactory way. However, implementation did turn up a way to make the accrual system more efficient.

These results can be used in several ways. It shows that the theories of e-p-semantics and the practical syllogism are fully specified (with just one addition in the case of the practical syllogism) and could be used in real applications such as automatic reasoning systems. It also shows that the research of accrual of arguments has not yet reached this stage, and that further research on this topic has to focus on defining preferences on sets of promoted and demoted goals, upon which a definition of accrual strength can be based.

7.3 Further work

As was discussed before, more research is needed to accommodate accrual strength. This requires a radical change in the way strength is modelled in the current implementation. Also, games with accrual arguments may become more efficient if the algorithm takes the internal structure of (often similar) accrual arguments into account.

Another addition to the program that would be nice is a more elaborate notion of undercut. Other approaches often incorporate a form of undercut which is not a general undercutter scheme like accrual undercut, but a user-defined undercutter for one particular inference. As mentioned in Chapter 3, Section 3.1.5, this would require the language to be extended so that inferences can be expressed in it. If this is done, accrual undercutters can have conclusions too, which would be more elegant than the current situation.

Furthermore, it would be nice to make an extension to a first-order predicate language, so that beliefs can be represented in a less restricted way.

Finally, the theories that have been implemented and the topics that they model are only a part of argument-based practical reasoning. The current implementation could be extended with approaches to other topics of argument-based practical reasoning, as long as they can be combined with the currently implemented ones.

Appendix A

User manual

Getting started

1. Download the zip file containing the program from http://www.wietskevisser.nl/research/epr.
2. Unzip this file and save the contents.
3. Doubleclick the jar file; the GUI will open.

Making a query

You can either load an existing belief base into the editor (File > Open), or enter one yourself. Example belief bases can be found in the folder `epr/examples`. A query must always be entered by hand. Both the belief base and the query should adhere to the syntax specified below. All white space and comments (starting with '//' and ending with a newline character) are ignored.

```
beliefBase ::= ( fact | rule )+
fact       ::= formula ( strength )? "."
formula    ::= ( ( "~" )? "D" )? literal
literal    ::= ( "~" )? atom
atom       ::= ["a"-"z"] ( ["a"-"z","A"-"Z","0"-"9"] )*
strength   ::= ( "0." ( ["0"-"9"] )+ ) | "1"
rule       ::= ( ruleName ":" )?  formula "<-" formula ( "," formula )*
               ( strength )? "."
ruleName   ::= atom
query      ::= formula "."
```

Now you can select the query options (semantics, practical syllogism, accrual, strength and accrual strength) and the log level you want in the menu and press the query button. The results or any errors will be shown in the log.

Documentation

The API documentation (Javadoc) can be accessed through the GUI (Help > Javadoc) or directly in the folder `epr/javadoc` (open `index.html`). The Schema definitions that are used for the XML output can be found in the folder `epr/xml`.

Bibliography

[1] Leila Amgoud, Lianne Bodenstaff, Martin Caminada, Peter McBurney, Simon Parsons, Henry Prakken, Jelle van Veenen, and Gerard Vreeswijk. *ASPIC Deliverable D2.6 - Final review and report on formal argumentation systems*. 2006.

[2] Argument Service Platform with Integrated Components (ASPIC). http://www.argumentation.org/.

[3] ASPIC AIFXML Schemas. http://aspic.acl.icnet.uk/.

[4] Trevor J.M. Bench-Capon and Henry Prakken. Justifying actions by accruing arguments. In P.E. Dunne and T.J.M. Bench-Capon, editors, *Proceedings of the First International Conference on Computational Models of Argument (COMMA06)*, number 144 in Frontiers in Artificial Intelligence and Applications, pages 247–258. IOS Press, 2006.

[5] Carlos Chesñevar, Jarred McGinnis, Sanjay Modgil, Iyad Rahwan, Chris Reed, Guillermo Simari, Matthew South, Gerard Vreeswijk, and Steven Willmott. Towards an argument interchange format. *The Knowledge Engineering Review*, 21(4):293–316, 2006.

[6] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.

[7] Graphviz. http://www.graphviz.org/.

[8] JavaCC. https://javacc.dev.java.net/.

[9] John L. Pollock. *Cognitive carpentry. A blueprint for how to build a person*. MIT Press, 1995.

[10] John L. Pollock. The logical foundations of goal-regression planning in autonomous agents. *Artificial Intelligence*, 106:267–335, 1998.

[11] John L. Pollock. Planning agents. In A. Rao and M. Wooldridge, editors, *Foundations of rational agents*. Kluwer, 1999.

[12] Henry Prakken. A study of accrual of arguments, with applications to evidential reasoning. In *Proceedings of the Tenth International Conference on Artificial Intelligence and Law*, pages 85–94. ACM Press, 2005.

[13] Henry Prakken. Combining sceptical epistemic reasoning with credulous practical reasoning. In P.E. Dunne and T.J.M. Bench-Capon, editors, *Proceedings of the First International Conference on Computational Models of Argument (COMMA06)*, number 144 in Frontiers in Artificial Intelligence and Applications, pages 311–322. IOS Press, 2006. (Corrected version, May 14, 2007).

[14] Henry Prakken and Giovanni Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-classical Logics*, 7:25–75, 1997.

[15] Henry Prakken and Gerard A.W. Vreeswijk. Logics for defeasible argumentation. In D.M. Gabbay and F. Günthner, editors, *Handbook of philosophical logic*, volume 4, pages 219–318. Kluwer Academic Publishers, 2nd edition, 2002.

[16] Iyad Rahwan and Leila Amgoud. An argumentation-based approach for practical reasoning. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 347–354, New York, USA, 2006. ACM Press.

[17] Matthew South, Gerard Vreeswijk, and John Fox. Dungine: A Java Dung reasoner. In *Proceedings of the Second International Conference on Computational Models of Argument (COMMA08)*, to appear, 2008.

[18] Gerard Vreeswijk. Algorithms. Unpublished manuscript.

[19] Gerard A.W. Vreeswijk. An algorithm to compute minimally grounded and admissible defence sets in argument systems. In P.E. Dunne and T.J.M. Bench-Capon, editors, *Proceedings of the First International Conference on Computational Models of Argument (COMMA06)*, number 144 in Frontiers in Artificial Intelligence and Applications, pages 109–129. IOS Press, 2006.

[20] Gerard A.W. Vreeswijk and Henry Prakken. Credulous and sceptical argument games for preferred semantics. In *Proceedings of the 7th European Workshop on Logic for Artificial Intelligence (JELIA 2000)*, number 1919 in Springer Lecture Notes in AI, pages 239–253. Springer Verlag, 2000.