An Argumentation Framework Supporting Evidential Reasoning with Applications to Contract Monitoring

Nir Oren

A thesis presented for the degree of
Doctor of Philosophy
at the
University of Aberdeen.

Department of Computing Science

2007
Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: 2007
Abstract

In this thesis we present a framework for argument which aims to support concepts useful for evidential reasoning. The framework is presented at two levels, an abstract level, in which arguments are not treated as concrete entities, and a concrete level, in which arguments are underpinned by Subjective Logic. At the abstract level, the framework provides the theoretical underpinnings for the concept of support in a manner distinct from other frameworks. Support for an argument can mean that the argument may be inferred from existing arguments, or that the argument is a *prima facie* argument. At the concrete level, support for burden of proof, argument schemes and sensing actions is introduced. A simple dialogue game is also presented, which shows how agents can use the framework to reason about their environment. Finally we introduce a heuristic that allows agents participating in a dialogue to decide, given multiple possible utterances, which utterance to advance.

The second part of the thesis concerns itself with the application of the argument framework to the contract monitoring domain. We show how the concrete form of the framework allows a set of agents, possibly with conflicting goals, to agree on the most likely state of a contract, and to reach a decision regarding what penalties should be imposed if the contract has been violated.

The main contributions of this thesis include an abstract argumentation framework that introduces enhancements over traditional frameworks, and is geared towards evidential reasoning; the framework is instantiated in a novel way using Subjective Logic. This instantiation allows us to represent important concepts such as burden of proof and argument schemes. Another contribution involves the ability to introduce evidence via the sensors within the framework. Sensors are critical for evidential reasoning, but have not received much attention from the argumentation community.

We then introduce two contributions that demonstrate the validity of the model in a practical application, namely in the area of automated contracting. We describe a simple contracting language, and show how agents may use the argument framework to reason about the state of a contract.
Salient Points of the Thesis

- We introduce an evidence based abstract argument framework containing the notions of evidential support and attack between arguments.

- We describe a concrete argument framework based on Subjective Logic opinions. This framework provides support for a number of important concepts including argument schemes, burden of proof, and some instances of accrual of argument.

- A dialogue game allowing agent to introduce evidence into an argument is introduced.

- We show how an agent may decide which argument to advance, and which evidence to introduce into an ongoing dialogue.

- We describe a new contracting language.

- We show how the argument framework may be used in conjunction with the contracting language to allow a set of agents to reason about the state of a contract within a partially observable domain.
Acknowledgements

First and foremost, I’d like to thank Ruth Woods for, well, pretty much everything. This thesis wouldn’t be here today without her support and encouragement. I’m also grateful to my supervisors, Tim Norman and Alun Preece, for all their advice, ideas and help.

Of course, these are not the only people who have had a direct, or indirect hand in the completion of this thesis. So, I’d like to thank, in no particular order, the following people, without whom this thesis would have taken a lot more (or, in some instances, a lot less) time:

Gideon, Hana and Iris Oren, Madalina Croitoru, Frank Guerin, Joey Lam, Gunnar Grimnes, Martin Kollingbaum, Tina Gotschi, and the entire Woods clan.

This research was partly funded by the DTI/EPSRC E-Science Core Program and British Telecom, via a grant for the CONOISE-G project. It is continuing through participation in the International Technology Alliance sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence.
# Contents

1 Introduction.................................................. 1
   1.1 Overview................................................. 1
   1.2 Motivation............................................... 2
   1.3 Research Hypothesis..................................... 7
   1.4 Contributions of this Research........................ 8
   1.5 Thesis Structure......................................... 9
   1.6 Related Publications.................................... 9

2 Background................................................... 11
   2.1 Argumentation.......................................... 11
      2.1.1 Argumentation Frameworks........................ 13
      2.1.2 Accruals of Argument.............................. 20
      2.1.3 Argument Schemes.................................. 21
      2.1.4 Toulmin’s Representation of Argument............ 23
      2.1.5 Burden of Proof.................................... 23
      2.1.6 Dialogue Games.................................... 24
      2.1.7 Heuristics for Argument........................... 25
   2.2 Contracting.............................................. 26
   2.3 Subjective Logic........................................ 29
   2.4 Summary................................................ 35

3 Evidential Argumentation Frameworks.......................... 36
   3.1 Introduction............................................ 36
   3.2 The Need for Support................................... 36
   3.3 Reasoning with Support................................ 38
      3.3.1 Arguments, Attack and Support.................... 40
      3.3.2 E-Preferred Extensions............................ 44
      3.3.3 E-Stable Extensions............................... 45
      3.3.4 E-Grounded Extensions............................. 46
   3.4 Discussion.............................................. 46

4 A Concrete Argumentation Framework............................ 49
   4.1 Introduction............................................ 49
   4.2 The Logic Layer......................................... 50
Contents

4.3 The Dialectic Layer .................................................. 55
4.4 The Procedural Layer ................................................ 63
4.5 The Heuristic Layer ................................................ 67
  4.5.1 Learning from Evidence .................................. 69
4.6 Discussion ............................................................ 70
  4.6.1 The Logical and Dialectic Layers ....................... 70
  4.6.2 The Procedural and Heuristic Layers ............... 74
  4.6.3 Implementation ............................................. 76
4.7 Conclusions ......................................................... 77

5 Contracting ............................................................ 78
  5.1 Introduction ....................................................... 78
  5.2 Contracts .......................................................... 78
  5.3 Contracting Language Features ............................. 80
  5.4 A Simple Contracting Language ......................... 86
  5.5 Using Contracts in a Dialogue ............................... 88
    5.5.1 The Family of Fundamental Argument Schemes for Contracting 88
    5.5.2 Handling Variables in Contracts .................... 89
    5.5.3 Detecting Clause Applicability ..................... 93
    5.5.4 Violations and Norms .................................. 94
    5.5.5 Clauses and Norms .................................... 95
    5.5.6 Dealing with Violations ................................ 96
  5.6 Arguing about Contracts ..................................... 98
  5.7 Discussion ......................................................... 102
  5.8 Conclusions ....................................................... 104

6 Conclusions .......................................................... 106
  6.1 Future Work ....................................................... 106
    6.1.1 The Abstract Argument Framework ................ 106
    6.1.2 The Concrete Argumentation Framework .......... 107
    6.1.3 Contracting ............................................... 109
  6.2 Validating the Research Hypothesis ....................... 110
  6.3 Conclusions ....................................................... 111
List of Tables

5.1 The similarities between an argument scheme and a contract clause. . . . . . . . . 81
5.2 The contracting language syntax in BNF form. . . . . . . . . . . . . . . . . . . . 87
5.3 Representing the contract using the new contracting language. . . . . . . . . 88
List of Figures

2.1 A Summary of the topics covered in the background chapter. ........................................... 12
2.2 A simple argumentation framework. .................................................................................. 14
2.3 The components of an argument according to Toulmin. .................................................. 23
2.4 A frame of discernment. ....................................................................................................... 30

3.1 Possible representations in an argument framework of the sentence $p, p \rightarrow q, q$. ........ 37
3.2 An implicit representation of support in an argument framework containing only the attack relation. .................................................................................................................. 37
3.3 A set of arguments and their interactions in a framework containing support and attack. ........................................................................................................................................ 39
3.4 An illustration of support within the evidential framework. .............................................. 41
3.5 An illustration of a supported attack. .................................................................................... 42
3.6 An argument system showing supported arguments, support attacks, acceptability and admissibility. ......................................................................................................................... 43
3.7 Representing the flying example using the evidential framework. ................................ 46

4.1 An example illustrating the shortcomings of the abstract framework. .............................. 49
4.2 An argument graph with evidence and warrant nodes. ....................................................... 51
4.3 An illustration of how evidential nodes and argument schemes may interact with each other. ........................................................................................................................................ 54
4.4 An algorithm to compute conclusions given a set of argument schemes, instantiated arguments, and, optionally, some opinions. .................................................................................. 61
4.5 A “translated” argument graph containing the mutually exclusive pieces of evidence $a$ and $b$. .......................................................................................................................................... 63
4.6 Screenshot of the user interface to the implemented concrete framework. ................... 76

5.1 An English-language contract. .............................................................................................. 79
5.2 A possible, but faulty, formalisation of an argument scheme representing contract clauses. ................................................................................................................................. 83
5.3 The corrected argument scheme representation of a contract clause. ............................. 83
5.4 The algorithm used to determine which variables may be bound within a clause. ............. 91
5.5 A graphical representation of an utterance. .......................................................................... 100
5.6 A graphical representation of how one type of accrual may be dealt with. ..................... 101
6.1 A sample set of arguments showing why valuations and admissibility computations are interdependent. ................................. 107
Chapter 1

Introduction

1.1 Overview

The study of argument is a popular research area spanning disciplines ranging from philosophy to computer science. It examines both the structure of argument, as well as how arguments should be constructed. Great strides have been made in both areas, and artificial intelligence researchers consider argumentation a powerful reasoning technique. Argumentation has been applied to great success in the legal domain, for example in analysing and generating legal arguments (Ashley, 1990). While some attention has been paid to the notion of evidence as part of an argument, its treatment as part of a unified argument framework has been limited at best.

In this thesis, we describe two related argument frameworks designed for evidential reasoning. The first system utilises the notion of support between arguments, and corrects many of the criticisms levelled at another framework which makes extensive use of support, namely Bipolar Argumentation Frameworks (Cayrol and Lagasque-Schiex, 2005b). The criticisms are centred around the lack of distinction between support and inference within these frameworks. By introducing a number of new extensions aimed at evidential reasoning, and using support to aid us in distinguishing between standard, and default arguments, we overcome these criticisms. The second system consists of a more concrete framework, and is intended to allow one to associate an opinion with the conclusions of arguments, rather than just determine which arguments are, and are not, acceptable. We demonstrate how the concrete framework can be embedded within a dialogue, and present a heuristic that allows agents to decide which arguments to advance at any stage in the dialogue.

The frameworks we present are useful in a variety of settings where evidence is required to be considered before some decision is reached. This situation most often arises in partially observable environments, where an agent must determine what the actual state of the environment is most likely to be. This type of deliberative reasoning can also arise when multiple agents are present; here, agents may cooperate and undertake information seeking or enquiry to determine what the environmental state is, or they may have their own agendas and try to persuade each other that the environment is in a certain state. One concrete example of such a scenario, investigated in Chapter 5, occurs when agents are participants in a contract, and attempt to reason about the state of the contract, either by undertaking internal deliberation, or by engaging in debate regarding the state of the contract. For example, one agent may initiate the debate by claiming that the contract has been violated, while another may attempt to defend its position with regards to the contract.
The ability to carry out such debate is particularly useful in cases where contract monitoring or contract enforcement must take place within a partially observable domain. In such a case, each agent would like to persuade the others that the contract is in a state that garners it maximal utility.

The framework may also be used for practical reasoning, that is, reasoning about which action to undertake. Here, an agent would decide what action to perform by trying to determine what state the environment is in. Consider, for example, the case where an agent is attempting to decide where to situate a bridge. By calling upon various sensors (such as GIS systems, people on the ground, and satellite information), the agent is able to gather data which will allow it to justify its decision.

The abstract framework, while unable to assign opinions to conclusions, can be used to differentiate between different consistent sets of choices, and can handle complex interactions between arguments that the concrete framework cannot deal with. It is thus also useful in some situations where practical, or epistemic, reasoning must take place, and both framework are thus complementary.

1.2 Motivation

We envision our framework being used to drive reasoning within an intelligent, autonomous, agent. To motivate our research, we must thus examine the properties of agents and autonomous systems, the different types of reasoning they may perform, how they interact with each other, and how their behaviour may be regulated. We can then describe how argumentation, and evidence, may contribute to these processes.

If multi-agent systems (MASs) are to ever reach their true potential, groups of agents must be able to operate and interact within a variety of complex environments. The central tenant behind MASs revolves around having multiple agents operate within some environment or system. While anything can, philosophically, be considered an agent (Shoham, 1990), most MASs research focuses on autonomous agents. Indeed, many would argue that it is precisely autonomy that allows us to differentiate between an agent and a part of the environment. Autonomy allows an agent to operate by selecting actions based on its own goals, and the behaviour of the entire system is thus an emergent property of individual agent actions.

The emergent behaviour of multiple, interacting agents is what gives MASs its power. However, finding ways to harness this power is difficult, and the techniques developed for this task form the core of MASs research. The areas investigated by MAS researchers can be broadly subdivided into the intra-, and inter-agent levels (van der Torre, 2003).

At the intra-agent layer, research questions form around how to construct a complete, intelligent agent. The main problems at this level are thus cognitive. Researchers working on this layer are likely to borrow ideas from psychology, neuroscience, and artificial intelligence. Many attempts at building cognitive agents have taken concepts from folk psychology. Indeed, it is hard to imagine describing an agent without referring to concepts such as its beliefs and goals. The main questions facing researchers attempting to build cognitive agents include which concepts should be represented within the agent, and how such a representation should take place; how to translate

---

\(^1\) Some state that an agent must also have reactivity and proactivity (Wooldridge and Jennings, 1995), we class these, as well as the notion of "intelligence" under autonomy.
from the concepts to some concrete action, and finally, how changes in the environment, often due to the agent’s actions, influence the agent’s attitudes.

One of the most popular attitude-based agent architectures is the BDI model Rao and Georgeff (1991). In this model, an agent has a set of Beliefs regarding what holds true, a set of Desires representing the agent’s goals, and a set of Intentions. Intentions are desires to which the agent has, in some sense committed to achieving. Typical implementations have beliefs (and desires) represented by some formal language such as a modal logic. A deliberation procedure is used to select a set of non-conflicting desires to become intentions, and an action is picked from a plan library so as to satisfy the agent’s intention. This process of translating from concepts to concrete action is known as practical reasoning, and involves deciding which action to perform given an agent’s cognitive state. The dual of practical reasoning is epistemic reasoning. Here, the question that needs answering is how an agent should update its cognitive state based on its current state and its observations (for example, epistemic reasoning would be used to determine which desires to instantiate as intentions).

Most agents perform both practical and epistemic reasoning, and it is difficult to have one without the other within an intelligent autonomous agent. In many situations, loops between practical and epistemic reasoning may form (Pollock, 1998a). For example, assume that we would like to get to London before 3pm today. We form a plan based on our beliefs about the world (practical reasoning), and decide that we may either catch a plane, or a train to achieve this goal. However, we are unsure whether a train will get us there on time (epistemic reasoning), and must thus form a plan to find out (more practical reasoning). The results of this sensing action will cause further epistemic reasoning to occur as we update our beliefs about train times.

At the heart of both types of reasoning lies deliberation (or cognition according to Pollock). Practical reasoning may also include aspects of plan generation and decision theory, while epistemic reasoning may involve sensing, that is, integrating information from the world into the agent’s internal state.

The act of deliberation includes the weighing up of various alternatives. Often, these alternatives may be mutually exclusive (for example, we may either fly, or take the train, but not both), or even in conflict with each other. Techniques such as automated planning (Nau et al., 2004), or the use of experts (for example, within a blackboard architecture, where each expert would provide a suggestion, which other agents may modify, until a decision is reached (Laasri et al., 1988)), may be used to perform deliberation and select a course of action (or, for epistemic reasoning, a state update strategy).

Another strategy, based on the way humans appear to manage this internal deliberation process, involves the use of argument. That is, the agent advances a possible alternative, and then searches for reasons against it. This is repeated by examining the pros and cons for each alternative, until some computational limit is reached, or a possible alternative survives all arguments against it. The argument-based approach appears to hold much promise, and a number of researchers have attempted to use it as part of an agent’s reasoning strategy. Possibly the most famous of these is Pollock’s (1995) OSCAR framework.

Utilising argument as the basis for a reasoning procedure provides us with a number of advantages, including
1.2. Motivation

- Non-monotonicity. An agent’s knowledge base may be inconsistent. The ability to retract inferences provided by non-monotonic reasoning allows an agent to easily reason with such a knowledge base. Argument, by its nature, provides non-monotonic reasoning as the introduction of a new argument may cause the conclusions of an argument to be rendered invalid.

- Defeasibility. Related to the previous point, an argument’s conclusions may be accepted by default, and then, when additional arguments appear, the conclusion may be defeated. Arguments may also be reinstated when arguments defeating them are, in turn, defeated. Defeasible reasoning is critical when reasoning with limited knowledge.

- Ease of handling new information. If an agent changes its beliefs, it is possible to integrate such changes with any reasoning that has already taken place, while still taking advantage of any reasoning that has taken place until the new information had appeared.

- Simpler knowledge engineering. Arguments are natural tools for people to reason with. It is thus easier for a knowledge engineer to create an agent that reasons with arguments, at least when compared to more formal methods.

- Enhanced understanding of the output. Similar to the previous point, humans are naturally able to follow a line of reasoning presented as an argument, allowing one to understand the way in which an agent arrived at a certain conclusion.

An agent operates within an environment, and reacts to changes within it. An agent must thus incorporate sensing into its reasoning process. In all but the simplest of environments, this is a difficult task. For example, the environment may change while the agent reasons, or the sensors utilised by the agent to determine the environmental state may be fallible. Some portions of the environment may not be accessible to the agent, meaning that it has to infer their state by indirect observations. To operate in such an environment requires techniques for reasoning under uncertainty. Bayesian networks (Pearl, 1988) are probably the most popular technique for performing such reasoning. It is, however, difficult to use Bayesian networks in many domains. Reasons include the networks’ computational complexity, the need for a prespecified network structure, and problems when dealing with inconsistent probabilities\(^2\). Machine learning techniques can help overcome some of these issues (Mitchell, 1997), but it is clear that other techniques are more suitable in certain domains. Since reasoning about the environment must occur as part of epistemic reasoning, it would be useful to represent both using a single formalism. In this thesis, we propose using Subjective Logic (Jøsang, 2001), which we describe in detail in Chapter 2, as this underlying formalism. Subjective logic allows us to deal with the notions of belief, as well as uncertainty, and provides us with a calculus for combining these notions.

So far, we have looked at only the intra-agent level of a multi-agent system. Even at this level, it should be apparent that argumentation is a useful reasoning tool, and that some sort of uncertainty representation for dealing with evidence is required to perform reasoning in any sort

\(^2\)Pollock (1995) points out some other fundamental difficulties in using the Bayesian approach for epistemic reasoning. Most, but not all, of his criticisms can be reduced back to the problems of dealing with inconsistent probabilities.
of realistic domain. We will now briefly examine the inter-agent level. Here, an agent must interact with other agents present in the environment, and argument theory may appear in many guises.

Any sort of communication between agents could be thought of as a dialogue. Walton and Krabbe (1995) have identified a typology of dialogues, and most agent communication protocols can easily be subsumed into one (or more) of these types. For example, the act of one agent attempting to obtain information from another could be viewed as a type of information seeking dialogue, while agents undertaking distributed planning can be seen as participating in a combination of persuasion, deliberation and negotiation dialogues. While many of these dialogues are simple, argument based techniques are often applicable when agents are faced with a choice of possible responses to an utterance. While simple prescripted dialogues are sometimes used as part of a protocol, the use of argument based protocols allow for more robust autonomous agents to be created (as agents are able to reason about unexpected situations). If agents have conflicting information, or are able to further their own goals by having the dialogue proceed in a specific manner, argument based techniques are even more applicable. Many researchers have looked at the use of argument between agents in the context of multi-agent systems in roles as diverse as negotiation (Rahwan et al., 2003), persuasion (Reed et al., 1997), and as a powerful group decision making technique (McBurney and Parsons, 2000).

It should also be noted that an agent operating within a society must often reflect on the society when performing reasoning. This is because the agent must take into account other agent’s actions when deciding what to do, and because the agent may have to curtail its actions based on societal norms. These societal norms include obligations, permissions and prohibitions that are imposed on the agent. Norms provide a powerful declarative control mechanism for agent societies. Kollingbaum (2005) identifies two types of norm mechanisms. The first, referred to as internalised control, has the norms programmed into the agent, limiting the agent’s choices, but simplifying its reasoning procedure. The actions of the resulting agents are predictable, but this type of approach is unsuitable for many environments since access to the agents’ internal program is required. Thus, the internalised control approach is most applicable to closed systems, where a single group is responsible for all agents.

Many multi-agent systems are expected to be open environments – that is, they contain agents that were created by different organisations. Agents may enter these environments at any time and interact with other agents (usually according to some published protocol). In such domains, it is impossible to verify that an agent conforms to a certain set of behaviours. To operate in such a domain, an agent must be norm-aware, but must be able to reason about norm violations. A normative agent is one that is able to perform such reasoning, and decide whether to honour, or ignore, a set of norms. A normative agent has a further reasoning advantage over a non-normative agent, namely in that it is able to reason about conflicting norms.

There are two broad approaches to categorisations for specifying the behaviour of normative agents, namely offline, and online approaches. In the offline approach, exemplified by Shoham and Tennenholtz’s 1995 work on “social laws”, a set of rules is imposed upon the system when it is created. If agents are norm-aware, they may ignore these laws (but suffer a penalty if they do). Much work has been done investigating the issues surrounding the development and use of social

---

3The possibility of programming errors makes this a problem even in closed systems.
laws, including their computational complexity and their automatic synthesis. The advantage of the offline approach is that norms are created only once, saving valuable processing time as the system executes. However, publishing a set of norms in this way results in an inflexible system. Very often, norms should be dynamically constructed. For example, one agent may agree to buy a service from another agent, and agree that the service be delivered in a specific way. If the system designer had not foreseen the possibility of such an interaction (which, in an open system, may easily be the case), it should be possible for the agents to agree upon a set of norms that will govern this transaction. A more extreme example of this scenario, and one which is attracting much real world interest, occurs when a set of agents identify a market need and decide to form a virtual organisation to service this need (Patel et al., 2005). While the online approach creates more flexible agents, it introduces additional complexity into the system, making verification difficult and places more reasoning requirements on the agents.

Contracts are probably the most popular approach for performing online behaviour regulation (social laws can also be viewed as contracts, albeit offline ones). The first problem encountered when using contracts in a multi-agent system involves how they should be created. Various techniques have been proposed, including combining a number of contract template fragments from a contract library to form a final contract (Cardoso et al., 2006), negotiating the values of variables in a contract template, after which it can be instantiated (Kollingbaum and Norman, 2002), and, utilising argumentation to negotiate a contract (Carborgim and Robertson, 1999).

Once a contract has been negotiated, agents must reason about the norms imposed on them due to the contract, and alter their actions as appropriate. Apart from performing the reasoning itself, the main difficulty at this stage involves contract parsing, and a myriad of machine understandable contracting languages have been proposed, each with different advantages and disadvantages (Oren et al., 2005). The stage at which a contract is active is often referred to as contract execution.

While the contract is executing, it is also being monitored. An agent performs contract monitoring by determining which norms the contract imposes upon them at the present time. If a norm is broken, sanctions may come into play.

A range of sanctions may be applied to an agent violating a contract. One possibility involves imposing additional norms on the agent, with the idea that these additional norms cause the agent to correct its behaviour. These are often referred to as contrary to duty obligations (Prakken and Sergot, 1996). Another way of sanctioning an agent involves reducing its utility. Normally, this requires that some sort of centralised authority exist in the system. This authority could then impose a financial penalty on the agent. In a distributed, decentralised system, enforcing sanctions is difficult. Another technique for punishing an agent could occur via a reputation mechanism (Teacy et al., 2006). Agents with a low reputation would find it more difficult to enter into agreements in the future, thus indirectly leading to a reduction in their utility.

As can be seen from the preceding discussion, all phases of the contracting lifecycle have been investigated. However, many open questions still remain, especially in the contract monitoring stage. The notion of evidence plays a key role here, and we thus examine this phase in more detail, explaining why the approach that this thesis proposes is particularly relevant for contract monitoring.
Given a fully observable domain and a machine interpretable contract, contract monitoring reduces to theorem proving: all an agent must do is show that the environment directly maps onto some contract state. When some portions of the environment are hidden, the problem becomes much more difficult. The agent must now gather evidence so as to deduce the values of the hidden states. This may be further complicated by the fact that contracts often rely on a large amount of background knowledge, some of which may very well be inconsistent or in conflict. Lastly, it should be noted that since contracts occur between (at least) two parties, the violation of the contract is usually beneficial to at least one party, and harmful to at least one other party. It is thus sometimes in an agent’s best interest to attempt to hide the fact that a contract violation occurred. Therefore, simply “asking” an agent whether it has violated a contract may not yield the desired results, as the agent may lie.

Humans perform contract monitoring by advancing arguments for their position and introducing evidence to reinforce it (Walton, 2002). Evidence may be uncertain, and thus reasoning about, and arguing with, evidence is a critical part of argument based contract monitoring.

In this section, we have shown that argument is a powerful tool for building reasoning agents. Argument may be used at both the intra-agent, and inter-agent levels. At the intra-agent level, an agent may engage in a deliberative argument so as to decide the best course of action. At the inter-agent level, arguments between agents may occur for many reasons, including persuasion, information seeking and negotiation. In all these cases, arguments must be backed up by some sort of evidence. One domain in which argument with evidence is particularly useful is contract monitoring. It should be clear, from the preceding discussion, that solutions to the problem of arguing with evidence will bring the promise of robust multi-agent societies one step closer to reality.

1.3 Research Hypothesis

One of the long term goals of multi-agent system researchers is to have large groups of autonomous agents interact with each other within rich, complex, agent societies. To do so, these agents must be able to reason about the environment they inhabit, and, more specifically, to undertake practical reasoning, that is, to decide which actions they should undertake in order to achieve their own goals.

Argumentation has been recognised as a powerful tool for practical reasoning, and in this thesis, we focus on one aspect of the problem, namely reasoning with, and about, evidence. This type of reasoning appears in many realistic domains, where only partial information regarding the state of the environment is available. It may even arise in fully observable domains, where the internal state of other agents is not accessible. Here, conclusions must be reached using evidence based on an agent’s actions.

We thus claim as our research hypothesis that

*A formal model of evidential reasoning can be produced that will allow agents to undertake inference in complex, partially observable domains.*

Here, a “complex domain” may be one in which agents attempt to hide knowledge from each other, and in which fallible sensors must be used whenever an attempt is made to determine the state of an observable portion of the environment.
This hypothesis arose from questioning how norm-aware agents may perform failure analysis and blame detection in the context of virtual organisations. We began this line of enquiry by looking at how humans dealt with the situation, and came up with a model inspired by the Pleadings Game (Gordon, 1993), namely having agents argue their case in front of an adjudicator. Unlike his framework, however, our solution does not have a centralised adjudicator. It was clear that evidence plays a major role in such cases (as can be seen by the wealth of material on the subject within the field of jurisprudence). When evaluating our hypothesis, we return to the legal domain; in Chapter 5, we show how the argumentation model we propose may be used in the field of contract monitoring.

1.4 Contributions of this Research

Apart from showing whether the research hypothesis holds, this thesis makes a number of contributions to the field of argumentation:

- A new set of extensions, the evidential extensions, are proposed which overcome many of the criticisms levelled at Bipolar Argumentation Frameworks. These extensions help justify the notion of support within abstract argument frameworks.

- A new argument framework is proposed. This framework differs from most existing argument frameworks in assigning numeric opinions to the conclusions of arguments. The framework caters for many concepts central to argumentation, including argument schemes, burden of proof, and is able to model many instances of accrual of arguments.

- A simple dialogue game is proposed that takes into account the ability to advance evidence as part of an agent’s move. This dialogue allows agents to make use of the framework when engaging in debate with each other, and explicitly caters for the introduction of evidence into the dialogue.

- A heuristic, corresponding to single-ply lookahead, is proposed, allowing agents to decide which utterance to make, as necessary, during a dialogue. As with the dialogue, the heuristic allows the agent to decide not only what utterance to make, but also what evidence to introduce into the debate.

- We have shown how the concrete and abstract argument frameworks may be combined to provide a more powerful framework, able to cater for situations that neither framework alone can handle.

Apart from the contributions made by introducing the framework, the validation process, carried out by showing how arguments about contracting may be situated within an evidence based dialogue, makes two further contributions:

- A new language for contracting is described, which allows for agents to argue about the state of a contract.

- By situating the contracting dialogue within the concrete framework, a powerful new method for performing reasoning about the state of a contract is introduced.
1.5 Thesis Structure
The remainder of this thesis is structured as follows: in the next chapter, related work is examined. This consists of looking at a number of argument frameworks, an overview of Subjective Logic, and a brief look at contract representation languages. We also look at some approaches that have attacked portions of the problem we examine.

In Chapter 3, we introduce an abstract argument framework which is designed to represent the semantics of reasoning with evidence. The framework is abstract in the sense that arguments are treated as atomic entities, and we focus on the interactions between these arguments, rather than their conclusions. The abstract framework is built upon in Chapter 4, where a full, concrete, argument framework is described for reasoning about evidence. Unlike the abstract framework, the concrete framework allows us to describe an argument and its components (such as premises and conclusions) in detail. At the base of this concrete framework lies a Subjective Logic based representation language in which artifacts in the environment can be represented. After introducing various argumentation concepts, such as argument schemes, we show how an opinion for an environment state may be computed, given various arguments for and against it. We then situate these ideas in a dialogue, allowing agents to make use of the framework to argue. Following this, we introduce a heuristic that may be used to aid agents in deciding what argument to advance within the dialogue.

Chapter 5 illustrates one possible application of the framework, namely in the area of contract monitoring.

We conclude the thesis by summarising our work, showing how our work answers the research hypothesis, and discuss possible areas of future work.

1.6 Related Publications
Portions of this thesis have previously appeared in the following publications:


1.6. Related Publications


Chapter 2

Background

This section both provides the background necessary for understanding the remainder of the thesis, and provides a survey of related work. Figure 2.1 provides a graphical summary of the contents of this chapter. We begin the chapter by introducing argumentation, looking at a number of commonly used approaches. We then focus on those techniques most relevant to the work presented in this thesis, namely bipolar argumentation frameworks, and those techniques which allow for the representation of uncertainty and concepts such as argument schemes. We also briefly examine dialogue games and heuristics for argument.

Following this, we look at some research into automated contracting. After a brief foray into contracting languages, we look at some work related to contract monitoring. We also examine a number of approaches to regulating agent behaviour that may be used in conjunction with, or instead of, contracts. In the last part of this chapter, we examine Subjective Logic, an uncertainty calculus which forms the basis of our concrete framework.

2.1 Argumentation

Argumentation researchers have long recognised that argumentation research can be divided into two main strands (Reed and Norman, 2003). The first involves the analysis of argument, while the second borrows ideas from argumentation theory in an attempt to create powerful reasoning mechanisms. The two strands are clearly intertwined, with progress in one area enriching the other. In this work, we follow the latter strand, creating an argument framework that can be used as part of a reasoning mechanism. While we do not intentionally attempt to model or analyse human argument, the framework that we describe in later chapters allows us to examine some of the properties of arguments between humans.

According to Prakken and Sartor (2002), an argumentation framework can be partitioned into four layers, namely the logical, dialectic, procedural and heuristic layers. This categorisation, while imperfect, is useful in the analysis of disparate frameworks, and we will make extensive use of it when discussing the various existing frameworks. Before examining these frameworks, however, we will examine each layer in more detail.

The logical layer describes the concepts that can be represented in the framework. For example, an argumentation framework grounded in logic might speak of sentences, literals, atoms and the like. It is at this level where the concept of an argument is described, and abstract argumentation frameworks may contain only arguments, as atomic entities, at the logical layer.

No mention is made of how arguments interact within the logical layer. This concept appears
at the dialectic level. In this layer, concepts such as rebutting and undercutting attacks between arguments, argument conflict, argument defeat, and argument admissibility appear. It is at this level that the status of an argument can be computed. The concept of defeat between arguments is a “static” one, independent of which other arguments are being considered. For example the argument “It is a sunny day, and I’ll therefore go for a walk”, may be (unconditionally) defeated by the argument “The windows are wet, therefore it is raining, and thus not sunny”. The status of an argument, on the other hand, is dynamic: given the above two arguments, we may deem the former argument inadmissible (due to its being defeated). However, given the additional argument “the windows are wet because I was cooking carrots”, which defeats the latter argument, the first argument’s status could now be changed to admissible. These concepts were elegantly formalised by Dung (1995), for the purposes of determining which sets of arguments may be considered “consistent”, and will be discussed in the following section. They form the basis of many modern argument frameworks.

As suggested by its name, the procedural layer concerns itself with the procedure that is to be followed when an argument takes place. Dialogue games (McBurney and Parsons, 2002) appear at this level, and appear to be the most popular way to specify procedure in argument. Dialogue games are discussed in detail in Section 2.1.6.

Conceptually, the topmost layer is the heuristic layer. This layer is where techniques for argument are described. For example, the idea that an agent should always advance its strongest argument would be described in the heuristic layer of an argumentation framework.

In the remainder of this section, we examine a number of argumentation frameworks. Following this, we look at some applications of argumentation. we then examine general concepts that
repeatedly arise in argument, describing where they fit into the four-layer model and how existing frameworks cater for them.

### 2.1. Argumentation Frameworks

**Dung’s Abstract Argumentation Framework**

Dung’s (1995) seminal argumentation framework has had a remarkable influence on the field of argumentation, and we thus begin our survey by examining it. The following discussion is taken almost directly from his paper.

Dung’s model is abstract, looking at the relations between arguments without stating anything regarding their content. To him, all that matters is the concept of *defeat* between arguments. Dung’s goal was to link argument with other areas, such as game theory and logic programming. In doing so, he defined a number of important concepts which were reused by other argumentation researchers. These concepts allowed one to determine, given a (possibly conflicting) set of arguments, which subsets of the original set could be considered, in some sense, to be consistent.

To illustrate some aspects of Dung’s framework, we will make use of an example. Consider the set of arguments:

- *a* Since the windows are wet, it is raining, and therefore, it is not sunny.
- *b* The windows are wet because I have been cooking carrots, not because it is raining.
- *c* It is sunny, and I should therefore go for a walk.
- *d* I shouldn’t go for a walk because I’ve got too much work to do.

We may formalise this example as follows:

**Definition 2.1. (Argumentation Framework)** An argumentation framework is a pair

\[ AF = \langle AR, \text{attacks} \rangle \]

where *AR* is a set of arguments, and *attacks* is a binary relation on *AR*.

If \((A, B) \in \text{attacks}\) (more conveniently written as \(\text{attacks}(A, B)\)), then it means that the argument *A* is an attack on the argument *B*.

Given a set of arguments *S*, we say that *S* attacks *B* if some argument in *S* attacks *B*.

Then we may define the following argumentation framework for the example above: \(AF = \langle \{a, b, c, d\}, \{(b, a), (a, c), (c, d), (d, c)\} \rangle\). This may be represented graphically as shown in Figure 2.2.

We can now define a conflict free set of arguments:

**Definition 2.2. (Conflict Free)** A set *S* of arguments is conflict free iff \(\not\exists A, B \in S\) such that \(\text{attacks}(A, B)\).

Clearly, the sets \(\{a\}, \{b\}, \{c\}, \{d\}\{a, d\} and \{b, c\}\) are conflict free.

It is possible to define the concept of acceptability from this. An argument is acceptable if it is defended from attack. Thus,
Definition 2.3. (Acceptability and Admissibility) An argument $A \in AR$ is acceptable with respect to a set of arguments $S$ iff for each argument $B \in AR$, where $attacks(B, A)$, $\exists C \in S$ such that $attacks(C, B)$. A conflict-free set of arguments $S$ is admissible iff each argument in $S$ is acceptable with respect to $S$.

In the example, the argument $c$ is thus acceptable with respect to the set $\{b, c\}$, while the argument $d$ is acceptable with respect to any set $x \subseteq \{a, d\}$. The sets $\{b\}, \{d\}, \{b, c\}$ and $\{b, d\}$ are admissible.

There are a number of ways an observer can interpret a set of arguments. A credulous observer, for example, when presented with a number of possible arguments, all of which are self consistent, would refuse to choose between them, saying that all possibilities are equally likely. Preferred extensions capture this concept:

Definition 2.4. (Preferred Extension) A preferred extension of an argumentation framework $AF$ is a maximal (with regards to set inclusion) admissible set of $AF$.

Thus, the sets $\{b, c\}$ and $\{b, d\}$ are the preferred extensions of the arguments found within the example.

After showing that an admissible set of arguments is monotonic with regards to acceptable arguments, Dung proves the following theorem:

Theorem 2.1. Given an argumentation framework $AF$,

1. The set of all admissible sets of $AF$ form a complete partial order with respect to set inclusion.

2. For each admissible set $S$ of $AF$, there exists a preferred extension $E$ of $AF$ such that $S \subseteq E$.

From this theorem, it can be shown that every argumentation framework has at least one preferred extension.

Dung also introduces the concept of a stable extension:

Definition 2.5. (Stable Extensions) A conflict free set of arguments $S \in AR$ is a stable extension iff $S$ attacks each argument in $AR$ which does not belong to $S$. 
In the example, both preferred extensions are stable extensions. However, while it is easy to see that every stable extension is a preferred extension, the reverse does not hold. For example, in the argumentation framework $\langle \{a, b, c\}, \{\{a, b\}, \{b, c\}, \{c, a\}\} \rangle$, the only preferred extension is the empty set. However, this framework has no stable extensions. Stable semantics are very aggressive, but make sense in some scenarios.

Another viewpoint an agent can adopt when presented with a set of arguments is one of scepticism. That is, when given a set of arguments in conflict, a sceptical reasoner will agree only with those arguments that unequivocally agree with each other. In the case of the example, a sceptical reasoner would that the only conclusion we may draw is that the windows are wet because cooking took place. To formalise the notion of scepticism, a number of definitions and lemmas are needed:

**Definition 2.6. (Characteristic Function)** The characteristic function $F_{AF}$ of an argumentation framework $AF$ is a function $F_{AF}: 2^{AR} \rightarrow 2^{AR}$ defined as follows:

$$F_{AF}(S) = \{ A | A \text{ is acceptable with respect to } S \}$$

It can be shown that a conflict-free set of arguments $S$ is admissible iff $S \subseteq F(S)$. Another lemma Dung proves shows that $F_{AF}$ is monotonic with respect to set inclusion. This means that $F_{AF}$ has a least fixed point, leading to the concept of grounded extensions, which are sceptical in nature:

**Definition 2.7. (Grounded Extensions)** The grounded extension of an argumentation framework $AF$, denoted by $GE_{AF}$, is the least fixed point of $F_{AF}$.

The grounded extension for the above argument framework consists of the set $\{b\}$.

To relate sceptical and credulous semantics, Dung introduces the concept of a complete extension:

**Definition 2.8. (Complete Extensions)** An admissible set $S$ of arguments is called a complete extension iff each argument which is acceptable with respect to $S$ belongs to $S$.

The relation between the various extensions is as follows:

- Each preferred extension is a complete extension (but not vice-versa).
- The grounded extension is the least complete extension.
- The complete extensions form a complete semi-lattice with respect to set inclusion.

While Dung goes on to show the relation between argumentation and game theory, as well as logic programming and argumentation, these links are not applicable in the context of this thesis. A number of criticism have been levelled at Dung’s framework, and we will now examine a number of these, as well as some extensions to his system.
2.1. Argumentation Frameworks

In human argument, preferences are often used to decide between arguments. For example, when presented with the arguments from the Nixon Diamond (which claims that a) Nixon is a warmonger, as he is a Republican, and that b) Nixon is a pacifist as he is a Quaker. Here, both arguments attack each other), a listener might decide that they assign more credence (i.e. prefer) the argument that Nixon is a republican, thus “breaking” the diamond and causing that argument to defeat the other. Amgoud and Cayrol (1997) have extended Dung’s model to cater for preferences between arguments. Their preference based argumentation framework (PAF) is constructed in the same way as Dung’s argumentation framework, with the addition of a preference ordering on arguments. Amgoud goes on to instantiate this argument framework, and shows how heuristics arise given certain types of arguments. She is also able to relate many of Dung’s concepts to concepts within PAFs. Ultimately, the addition of preferences enriches concepts such as the acceptability of arguments, allowing Amgoud to represent arguments which Dung’s framework would struggle to do.

Another, similar enhancement to Dung’s framework are Value Based Argumentation Frameworks (Bench-Capon, 2002). In this work, Bench-Capon proposes a generalisation of PAFs, known as a Value Based Argumentation Framework (VAF). A VAF is a tuple

\[ (AR, \text{attacks}, V, \text{val}, \text{valpref}) \]

Where \( AR \) and \( \text{attacks} \) are the same as those found in Dung’s framework. \( V \) is a non-empty set of values, and \( \text{val} \) is a mapping between arguments in \( AR \) and values \( V \). \( \text{valpref} \) is a preference relation expressing which values are preferred over other values. If each argument maps to a different value, a VAF reduces to a PAF, and if only one value exists, a VAF reduces to a standard Dung-like argumentation framework. As seen earlier, certain arguments can be accepted by credulous or sceptical observers. Bench-Capon then shows how certain arguments in a VAF appear in all extensions (and are thus acceptable to all observers), as well as showing that certain types of VAFs have a unique preferred extension. Such a concept is useful, as all arguments appearing in a unique preferred extension are universally acceptable (or objectively acceptable in his words). He also classifies arguments appearing in some preferred extensions as subjectively acceptable (or defensible), and arguments that appear in no preferred extensions as indefensible.

Consider the arguments

\( a \) Since I don’t have too much work (that is, I don’t have to write a paper and finish my thesis corrections), I’ll go running.

\( b \) I’ve got to finish my thesis corrections, but I can still go running.

\( c \) I’ve got a paper to write, but I can still go running.

Clearly, \( b \) and \( c \) do not individually attack argument \( a \). However, \( a \) is defeated when both \( b \) and \( c \) appear together. While it is possible to represent such attacks by introducing “virtual” arguments and keeping careful track of the attack relation between these arguments and those appearing originally, Nielsen and Parsons (2006b) extended Dung’s framework to allow for such attacks to appear without resorting to the introduction of additional arguments. He did this by
allowing sets of arguments to participate in an attack against an argument. This generalisation of Dung’s framework was shown to satisfy the same results that the original framework satisfied.

**Bipolar Argumentation Frameworks**

Criticism has also been levelled at Dung’s framework for representing only a defeat relation between arguments. Bipolar argumentation systems (Amgoud et al., 2004; Cayrol and Lagasquie-Schiex, 2005b; Cayrol et al., 2006) include an additional support relation in the definition of their argumentation framework. Bipolar argumentation frameworks (also referred to as BAFs) handle a number of issues that we attack in this thesis. We will thus describe them in some detail.

**Definition 2.9. (Bipolar Argumentation Framework)** A Bipolar Argumentation Framework (BAF) is a tuple $\langle A, R_a, R_s \rangle$ where $A$ is a set of arguments, and $R_a, R_s$ are the attacks and supports binary relations respectively.

It is assumed that the $R_a$ and $R_s$ relations are conceptually independent of each other. $R_a$ takes on the same role as Dung’s attack relation. $R_s$ attempts to capture the notion of one argument supporting another. Given this new relation, a number of new concepts and semantics for argument can be defined.

**Definition 2.10. (Support Attack and Indirect Attack)** $A_1$ is said to “support attack” $B$ if there is a sequence $A_1R_1 \ldots R_{n−1}A_n$ where $n \geq 3$ and $A_n = B$ and $\forall i = 1 \ldots n−2, R_i \in R_s$ and $R_{n−1} \in R_a$.

$A_1$ indirectly attacks $B$ if there is a sequence $A_1R_1 \ldots R_{n−1}A_n$ where $n \geq 3$ and $A_n = B$ such that $\forall i = 2 \ldots n−1, R_i \in R_s$ and $R_1 \in R_a$.

Any direct attack (i.e. an element of $R_a$) is also assumed to be a support attack.

**Definition 2.11. (Set Attacks and Set Support)** Let $S \subseteq A$ and $B \in A$. Then $S$ set attacks (also referred to as set defeats) $B$ iff $B$ is support attacked or indirectly attacked by an element of $S$.

$S$ set supports $B$ iff there is a sequence of the form $A_1R_1 \ldots R_{n−1}A_n$ where $n \geq 2$ such that $\forall i = 1 \ldots n−1, R_i \in R_s$ with $A_n = B$ and $A_1 \in S$.

**Definition 2.12. (Collective Defence)** Let $S \subseteq A, B \in A$. Then $S$ collectively defends $B$ iff $\forall C \in A, \{C\}$ set attacks $B$ then $\exists D \in S$ such that $\{D\}$ set attacks $C$.

**Definition 2.13. (Conflict-free Set)** Given $S \subseteq A$, we say that $S$ is conflict free iff $\nexists B, C \in S$ such that $\{B\}$ set attacks $C$.

**Definition 2.14. (Safe Set)** A set $S \subseteq A$ is said to be safe iff $\nexists B \in A$ such that $S$ set-defeats $B$ and either

1. $S$ set-supports $B$ or
2. $B \in S$

Thus, a set of arguments is deemed safe if it does not simultaneously support and attack (possibly through a chain of supports) some other argument outside the set.

As in a Dung style argument framework, we can define stable extensions as:

---

1 It should be noted that in all but Cayrol et al., 2006, the authors assume that a BAF is acyclic. In the latter, the issue is not mentioned.
Definition 2.15. (Stable Extensions in BAFs) Given a set of arguments \( S \subseteq A \), \( S \) is a stable extension (of the argument system) iff \( S \) is conflict free, and, \( \forall A \notin S, S \) set-defeats \( A \).

Since Cayrol’s original paper only considers acyclic systems, it can be shown that there is always a unique stable extension, which is also the unique preferred and grounded extension. Cayrol proves a number of properties regarding safety and stable extensions, but these are not important in the context of this thesis. The addition of the support relation allows her to suggest three possible definitions for admissibility:

Definition 2.16. (D-admissibility) A set \( S \subseteq A \) is d-admissible iff \( S \) is conflict free and defends all its elements.

Definition 2.17. (S-admissibility) A set \( S \subseteq A \) is s-admissible iff it is safe and defends all its elements.

Definition 2.18. (C-admissibility) A set \( S \subseteq A \) is c-admissible iff it is conflict free, closed over \( R_{sup} \) and defends all its elements.

From these, it is possible to define d-, s- and c-preferred extensions:

Definition 2.19. (D-, S-, C-preferred Extension) A set \( S \subseteq A \) is a d-/s-/c-preferred extension if it is maximal among the d-/s-/ or c-admissible subsets of \( A \).

A d-preferred extension is similar in its representational capabilities to Dung’s preferred extensions. Namely, it ensures that its elements are safe from attack.

S-preferred and c-preferred extensions represent different types of acceptability. An s-preferred extension attempts to capture the notion that arguments that defend and attack some other argument outside the set (i.e. an unsafe set) should not be deemed self-consistent. C-preferred extensions model internal coherence by ensuring that no arguments outside the preferred set are supported (and defended and conflict free).

The way in which support appears in BAFs has come under much criticism (Bench-Capon, 2007). Some argue that support simply captures the derivation procedure of the underlying logic, and that with the addition of support, we go back to “the bad old days” where every inference step has to be enumerated. Others claim that support simply makes the concept of sub-arguments more explicit, something which other, less abstract argument frameworks already capture. We too have some concerns regarding the notion of support as used within BAFs, being part of an abstract argument framework. However, as argued later in the thesis, we believe that the concept of support can play an important role when considering the notion of evidence within an argument framework.

The frameworks we have examined until now are abstract in nature: arguments are atomic entities, and, while their status may be determined based on their interactions with other arguments, no clue is given regarding how they originally became part of the argument framework (with the exception of the criticisms levelled at BAFs). We will now shift our attention to more concrete argument frameworks. These frameworks are usually based on some logic, and, by adding certain structures to the rules, allow us to express a rich variety of arguments in varied ways. Often, these expressed arguments are then evaluated using one of the abstract argument frameworks mentioned
above. We will examine a number of these frameworks, but will not attempt to do so exhaustively. The interested reader is referred to (Prakken and Sartor, 2002; Prakken and Vreeswijk, 2002), both of which contain excellent, detailed surveys of argumentation frameworks, and from which some of this discussion draws inspiration.

Pollock’s OSCAR Framework
The philosopher John Pollock has made some significant contributions to argumentation theory while working on his OSCAR system. His argument system, described in detail in (Pollock, 1995), introduced many novel ideas, and we shall therefore examine it here. The atoms of Pollock’s framework consist of standard propositions, such as \textit{bird(t)}. An argument is then built up of a (finite) sequence of lines of argument, where each line of argument is a triple \( \{X_i, p_i, v_i\} \). \( X_i \) is a set of propositions, referred to as the suppositions at line \( i \). \( p_i \) is a proposition (which may be viewed as the conclusion of the line), and \( v_i \) is the strength or degree of justification of the proposition within that line.

The question immediately arises regarding how lines of argument may be combined. According to Pollock, a later line of argument may be derived from earlier lines of arguments based on one of five rules. For example, the input rule is defined as

\textbf{Input} \hspace{1cm} If \( p \) is an input, and \( \phi \) is an argument, then, for any \( X \), it holds that \( \phi, \{X, p, \infty\} \) is also an argument.

In other words, given a known, unquestionable, fact (which Pollock calls an input), \( p \), we may include it in an argument (by adding it as a new line of argument) with an infinite degree of justification (as it is a fact), regardless of preconditions \( X \). Other rules include reason, supposition, conditionalisation and dilemma. We will examine the first rule in more detail.

Pollock defines two types of reasons. Conclusive reasons consist of standard first order inference schemes, and have infinite strength. More interesting are \textit{prima facie} reasons. These can be viewed as introducing domain-specific inference methods into his framework. One commonly used \textit{prima facie} reason is the statistical syllogism:

If \( (r > 0.5) \) then \( [x \text{ is an } F] \text{ and } p(G|F) = r \) is a prima facie reason of strength \( r \) for believing \( [x \text{ is a } G] \)

The symbols \( [\] \) are used as an objectification operator. This operator transforms expressions from the meta-language to the language, allowing the argument process to affect the rules of the system. Thus, we are able to speak about, and change, the strength of reasons during the course of an argument.

To use a reason in an argument, we must make use of the following inference rule:

\textbf{Reason} \hspace{1cm} If \( \sigma \) is an argument which includes \( (X_1, p_1, \eta_1), \ldots, (X_n, p_n, \eta_n) \) as members.

Then if \( \{p_1, \ldots, p_n\} \) is a reason of strength \( \nu \) for a conclusion \( q \), and for each \( i, X_i \subset X \), then a new argument \( \sigma, (X, q, \min(\eta_1, \ldots, \eta_n, \nu)) \) is an argument.

In other words, given that a reason may be applied in a certain situation (by having all its “premises” present), we may deduce the conclusion of the reason. The strength of a reason is the
minimum strength of all its premises and of the reason itself. As will be discussed later, reasons may be seen as a type of argument scheme.

Pollock’s framework contains a number of other innovative concepts, such as the ability to introduce assumptions within a dialogue, but these are not strictly relevant to this thesis, and will thus not be discussed here.

Two types of attack between argument were suggested by Pollock. The first occurs when two arguments have opposing conclusions, while the second occurs when the conclusion of an argument means that a reason found in another argument may not be used. The first case is one of rebuttal between arguments, while the second is called an undercutting attack between arguments. Since Pollock’s arguments have an associated strength, it is possible for one of a pair of rebutting arguments to still be deemed admissible. Once arguments have been defined, it is easy to recast them within an abstract argument framework. Over time, Pollock has wavered between different semantic interpretations of the results of an argument. The latest version of his work evaluates arguments in a manner similar to Dung’s preferred semantics.

Pollock’s theory of argument is extensive, and the result of decades of research into various aspects of logic and philosophy. The work described in Chapter 2 attempts to tackle some of the issues his framework addresses, but does so in a different way. We discuss these differences in Section 4.6.

2.1. Argumentation

2.1.2 Accruals of Argument

One criticism often levelled at Pollock’s framework is its difficulty in dealing with accruals of argument. Informally, an accrual of arguments occurs when arguments with the same conclusion combine to alter the strength of the conclusion. Pollock claims that accruals are, in fact, a fallacy, and should be represented using statistical syllogism and the principle of the weakest link. However, it has been shown that taking this approach is unsatisfactory (Prakken, 2005b). Accruals are, in fact, notoriously difficult to represent formally without hand crafting rules for each possible accrued interaction. Nielsen and Parson’s approach, described previously, may be seen as a simple way of representing accruals. A full theory of accruals, according to Prakken, must be able to cater for the following features:

1. Accruals are sometimes weaker than their elements.
2. An accrual makes its elements inapplicable.
3. Flawed reasons or arguments may not accrue.

Prakken identifies two broad approaches to accruals of argument. The first, which he calls the knowledge representation approach, involves explicitly representing accruals. That is, given that two (or more) arguments \(a_1, a_2, \ldots\) accrue, we must hand code the accrual process, providing rules of the form “if \(a_1\) then deduce \(c_1\),” “if \(a_2\) deduce \(c_2\),” if \(a_1 \land a_2\), deduce \(c_3\)” and so on. He criticises this approach for three reasons. First, too much work is required to enumerate all the rules. Second, even if the rules were automatically generated, the approach requires that all the reasons can be expressed in the same way, so that they can be combined using the same conditional operator. Third, he shows that invalid arguments must be represented as part of the accrual, whereas they should have been dismissed before the accrual takes place.
Instead of approaching accruals this way, Prakken describes a framework wherein accruals can take place as part of the inference process. However, since his approach does not cater for the valuation of arguments, which are a fundamental part of accruals, his work must be viewed as incomplete.

Verheij (1996) suggested a similar treatment of accrual of arguments. As its name suggests, his work on Reason-Based Logic proposes a logic in which reasons (and rules) may be operated on to reach various conclusions. His approach is a predicate based one, with a number of special predicates used to reason about rules. For example, we may say

\[
\text{Valid}\text{(rule}\langle\text{injury,}
\text{injured(}\text{person1, person2)}\text{,}
\text{shouldBePunished(}\text{person1})\text{)}\text{)}
\]

\[
\text{Injured(john, peter)}
\]

To represent a rule that is applicable when a specific person injures a specific other person (i.e. when the variables \text{person1} and \text{person2} are bound). In the case where the rule applies, \text{person1} should be punished. In the concrete case, John injured Peter. As will be seen, this approach is similar to our concrete approach for argument. To represent accruals, Verheij introduces the notion of the Outweighs predicate, allowing us to say that the combination of a number of rules outweighs a single application of a rule. Given the above rule, together with another rule (called “accidentalInjury”) stating that if the injury is accidental, the person causing the injury should not be punished, we could say that the latter rule outweighs the former with the additional rule

\[
\text{Outweighs}\{\text{accidentalInjury(}\text{person1, person2)}\},
\{\text{injured(}\text{person1, person2)}\},
\neg \text{shouldBePunished(}\text{person1})\}
\]

Now, when both rules apply, leading to the conflicting conclusions shouldBePunished and \neg shouldBePunished, the Outweighs predicate will allow us to choose the latter conclusion over the former one.

Verheij also introduced an abstract argument framework called CumuLA. This framework allows for the representation of attacks by an accrual of arguments, that is, even if argument \(a\) and \(b\) are not, individually, strong enough to defeat \(c\), if both are acceptable, then \(c\) may be defeated. Nielsen (2006) pointed out some weaknesses in Verheij’s approach, and suggested an alternate abstract framework, based on Dung’s formalism, in (Nielsen and Parsons, 2006b).

### 2.1.3 Argument Schemes

Verheij’s rules contain variables, and can thus be viewed as a primitive form of argument scheme. Argument schemes are similar to rules or reasons for an inference, and attempt to capture typical patterns of reasoning in argument. Unlike rules though, argument schemes are often not deductive in nature. Historically, many forms of reasoning captured within argument schemes were regarded as fallacies, instead of being viewed as applicable in a narrow set of situations, or of carrying some weight in support of an argument. Some researchers have put forth taxonomies of argument schemes (e.g. Walton (1996) examines twenty-five schemes in detail), but all admit that many argument schemes still await discovery and categorisation.
2.1. Argumentation

Typical computational representations (e.g. Reed and Walton (2005)) associate a name, a set of premises and a conclusion with a scheme. Walton (1996) describes argument schemes in more detail, with the premises of a scheme being divided into major and minor premises, as well as having a set of critical questions associated with the scheme. If any of the critical questions cannot be successfully answered, the scheme is not applicable, even if its premises hold. Thus, critical questions can be viewed as a form of undercutting attack on the inference. A scheme’s major premise represents the inference itself, while the minor premises represent the scheme’s preconditions. To illustrate these concepts, consider the well studied argument from expert opinion (taken from (Walton, 2005)):

Minor premises:  
E is an expert in domain D
E asserts that A is true
A is within D

Major premise:  
When an expert E asserts that A is the case, and
A is within their domain of expertise D, A
may plausibly be taken to be true.

Conclusions:  
A is probably true

Walton identified a number of critical questions identified with this scheme. If any of a scheme’s critical questions are answered in the negative, the scheme may not be applied. This scheme’s critical questions are:

1. How credible is the expert source?
2. How much of an expert is E in domain D?
3. What did E assert that implies A?
4. Could E be biased?
5. Is A consistent with other experts’ testimonies?
6. Is E’s assertion based on evidence?

Prakken et al. (2004) asks whether these critical questions form implicit premises to the argument scheme. Questions 1, 2, 3 and 6 must clearly be shown to hold before the argument scheme is applied. Questions 4 and 5 however may be assumed to hold by default when making use of the scheme. According to Prakken et al., the difference between these critical questions is based on the notion of burden of proof. In the former case, it is up to the party advancing the argument to show that these critical questions are answered correctly. In the latter case, the party arguing against the use of a scheme or its conclusions must provide evidence to back up their assertions. In (Gordon et al., 2007), the categorisation of critical questions is further refined, showing that they may be used to determine whether a premise holds, whether any exceptional circumstances mean a scheme is inapplicable, whether other arguments may be used to attack a scheme, and whether any implicit premises exist. Different levels and types of burden of proof may apply to each of these categories. Finally, we should mention (Verheij, 2003b), which embeds argument schemes in the DefLog framework (Verheij, 2003a).
2.1. Argumentation

2.1.4 Toulmin’s Representation of Argument

While not computational in nature, Toulmin’s representation of argument (1958) has influenced many concrete argument frameworks, as well as the manner in which arguments, and argument schemes are represented. Thus, it bears a brief examination. Figure 2.3 shows how an argument may be deconstructed into its core components in Toulmin’s representation.

Within this figure, data represents facts (e.g. “Bob loved Alice”). These facts provide some sort of support to the argument’s claim (e.g. “Bob did not murder Alice”). The warrant links the data to the claim via some sort of general rule (for example, “People do commonly not murder the person they love”). A warrant is justified by some backing, usually statistical or observational in nature (e.g. “We can see that people do not usually murder the person they love”). The argument’s qualifier provides a strength to the argument’s claim, based on the strength of the warrant (in the case of the example, the qualifier would be “usually”). The rebuttal suggests what exceptional cases may cause the warrant to become inapplicable (e.g. “Bob thought Alice was having an affair”).

Many concrete models of argument utilise some aspect of Toulmin’s framework to represent arguments. Warrants, for example, can be seen as instantiations of argument schemes, while rebuttals are critical questions. One area in which Toulmin’s model has seen a large amount of use is in the graphical representation of argument. Reed’s Araucaria software (2004), has been used to analyse arguments, as well as teach critical reasoning skills, and its representation of argument clearly makes use of (and has been used to further investigate the computational aspects of) Toulmin’s model (Reed and Rowe, 2005; Reed and Walton, 2005). Paglieri and Castelfranchi (2006) framed belief revision as a monologue, and showed how Toulmin’s framework may be used to enhance the belief revision process.

2.1.5 Burden of Proof

So far, we have used the term “burden of proof” without formally defining it. The term may refer to two concepts, both of which are grounded in the legal domain, and both of which have been much discussed within the field of argumentation.

The first aspect of burden of proof relates to the amount of evidence required for an argument to be acceptable. This is sometimes known as the standard of proof. Typical standards of proof
include “on the balance of probabilities”, and “beyond reasonable doubt”. In a criminal case tried under the English legal system, the normal standard of proof states that the prosecution must show its case holds beyond reasonable doubt. If they are unable to do this, then the defendant will not be found guilty.

The term “burden of proof” may also be used to speak about the obligations that one of the parties must meet, so that they may be allowed to advance, or attack, a line of argument.

The standard of proof is not only used at the end of a dialogue to determine the status of a claim. Instead, it may apply to evidence or arguments at any stage of a dispute. The standard of proof is therefore closely linked to burden of proof. For example, a certain standard of proof may be required before evidence is deemed admissible (e.g. it may have to be shown that some DNA belongs to a person on the balance of probabilities). The burden of proof would then lie on the party presenting the evidence to exceed the standard of proof. Once this is done, the evidence may only be dismissed by providing counter arguments, or counter-evidence, lowering the standard of proof below the acceptable level. Thus, the burden of proof shifts to the party trying to dismiss the evidence.

The work of Farley and Freeman (1995) is one of the earliest formal treatments of burden of proof within a computational setting. The authors assign one of five levels of support to any defendable argument. Different inference procedures assign different levels of proof to their conclusions, and the level of burden of proof must be exceeded to allow an argument to be defeated. (Gordon and Karacapilidis, 1997) is another argument framework that operates in a similar manner. Burden of proof has been studied from many other angles, including (Prakken, 2001a), where burden of proof is argued to be procedural in nature, and is assigned by a judge (as a legal setting is assumed) to a party in a dialogue. Another interesting treatment of Burden of proof is (Prakken et al., 2004), where a framework is proposed in which the burden of proof can be allocated by the arguing parties. Burden of proof is treated as another proposition in the dialogue, allowing for embedded dialogues which themselves argue about the assignment of burden of proof.

2.1.6 Dialogue Games
Argumentation can be viewed at the purely logical level as a non-monotonic reasoning technique. Argument normally takes place between multiple parties in a somewhat organised manner, and being able to capture and model this can be useful. Capturing the dialogical aspect of an argument is most commonly done by means of a dialogue game (Walton and Krabbe, 1995; McBurney and Parsons, 2002).

A dialogue game is a formal representation of the communicative interaction between two (or, more rarely, more than two) parties. This communicative interaction takes the form of utterances, and these utterances follow the rules of the dialogue game.

Typically, a dialogue game contains the following elements:

- Commencement rules, which specify the circumstances under which the dialogue may begin.

- Locutions. These specify which utterances may be made, and are based on the type of dialogue that is taking place. These may include speech acts such as questions, assertions and retractions of propositions.
• Combination rules are context dependant, and specify which locutions are permitted (or obligatory) based on a number of factors, including what the entity making the utterance knows, what they have said, what the other party has said and what utterances have been made until that point in the game. For example, an utterance questioning an agent’s commitment to a certain belief may only be made after the agent has asserted its commitment to that belief.

• Commitment rules define the effects of an utterance on the propositions that the arguing parties are committed to defending. Typically, agents taking part in a dialogue game maintain a commitment store, which lists those propositions they are committed to defending. Commitment rules may add and remove elements from this commitment store, and may also affect the strength of commitment an agent has to elements of its store.

• Termination rules define the situations under which the dialogue ends. They may also specify a “winner” of the game, though often any party unable to make an utterance during its turn, leading to the end of the dialogue, is defined as the loser of the game.

The systems defined in previous sections capture the static aspect of argumentation; given that a set of arguments has been advanced, they allow us to decide which sets of arguments are, in some sense consistent. Dialogue games on the other hand, by specifying the protocol agents may use when arguing with each other, capture the dynamic aspects of argument. Walton and Krabbe (1995) suggested a highly influential (but, by no means exhaustive) typology of dialogue types, and most dialogue games fit into one or more of Walton’s categories. Dialogues may be embedded in one another (for example, a negotiation dialogue may shift to a persuasion dialogue when one agent attempts to show another why its bid is useful), and various frameworks have been proposed to model such behaviour (Reed, 1998).

2.1.7 Heuristics for Argument

Agents may reason about the set of consistent arguments using the frameworks described earlier. They may also engage in dialogue by partaking in a dialogue game, as detailed in the previous section. One question that has not yet been answered is how, given a set of possible legal utterances, as constrained by the dialogue game, an agent may choose the utterance that is most beneficial to it.

Argumentation researchers have recognised the need for argument selection strategies for a long time. However, the field has only recently started receiving more attention. Moore (1993), in his work with the DC dialectical system, suggested that an agent’s argumentation strategy should take three things into account:

1. Maintaining the focus of the dispute.

2. Building its point of view or attacking the opponent’s one.

3. Selecting an argument that fulfils the previous two objectives.

The first two items correspond to the military concept of a strategy, i.e. a high level direction and goals for the argumentation process. The third item corresponds to an agent’s tactics. Tactics
allow an agent to select a concrete action that fulfils its higher level goals. While Moore’s work focused on natural language argument, these requirements formed the basis of most other research into agent argumentation strategies.

Amgoud and Maudet (2002) proposed a computational system which captures some of the heuristics for argumentation suggested by Moore. Their system requires very little from the argumentation framework. A preference ordering is needed over all possible arguments, and a level of prudence is assigned to each agent. An argument is assigned a strength based on how convoluted a chain of arguments is required to defend it. An agent can then have a “build” or “destroy” strategy. When using the build strategy, an agent asserts arguments with a strength below its prudence level. If it cannot build, it switches to a destroy strategy. In this mode, it attacks an opponent’s arguments when it can. While the authors note other strategies are reasonable, they make no mention of them. Shortcomings of their approach include its basis on classical propositional logic and the assumption of unbounded rationality; computational limits may affect the arguments agents decide to put forth.

Using some ideas from Amgoud’s work, Kakas et al. (2004) proposed a three layer system for agent strategies in argumentation. The first layer contains “default” rules, of the form $\text{utterance} \leftarrow \text{condition}$, while the two higher layers provide preference orderings over the rules. Assuming certain restrictions on the rules, they show that only one utterance will be selected using their system, a trait they refer to as determinism. While their approach is able to represent strategies proposed by a number of other techniques, it does require hand crafting of the rules. No suggestions are made regarding what a “good” set of rules would be.

Bench-Capon (1998) describes a dialogue game based on Toulmin’s work. He identifies a number of stages in the dialogue in which an agent might be faced with a choice, and provides some heuristics as to what argument should be advanced in each of these cases. Only an informal justification for his heuristics is provided.

We have outlined heuristics of argument based on an agent attempting to reveal a minimum of information with its utterances (Oren et al., 2006b), and attempting to maximise its utility (Oren et al., 2006a). These are discussed in more detail in Section 4.5.

McBurney and Parsons (2001) treated dialogues as abstract games, and were able to assign families of dialogues game theoretic semantics. By doing so, optimal strategies for agent utterances could be computed.

## 2.2 Contracting

In Chapter 5, we show how our framework can be used to perform automated contract monitoring. To set the scene, we will now briefly examine various aspects related to automated contracting and survey a number of contracting languages.

Automated contracts are probably the most popular technique for normative behaviour specification. It could be argued that research into social laws (Shoham and Tennenholtz, 1995) and even mechanism design (Dash et al., 2003) also fall under contracting albeit while making use of an implicit contract, rather than, as with most contracting approaches, focusing on an explicit document.

---

Social laws and mechanism design also differ from contracts in that the former allow for the system to be designed “offline”, while the latter allows for “online” behaviour regulation.
2.2. Contracting

Broadly, a contract is a legally binding agreement between two or more parties, imposing norms on the contract signatories for the duration of the contract. Many languages for contract representation have been proposed, and we examine a number of these in this section.

A contract passes through a number of distinct phases throughout its life, and various contract representation languages have been developed that focus on one or more of these phases. The first stage involves the \textit{drafting of the contract}. In this phase, languages which aid in contract negotiation and contract validation (i.e. languages which ensure contract consistency) are important.

Once a contract has been drawn up, it comes into effect, and regulates the behaviour of the contracting parties. At this stage, \textit{contract monitoring} becomes important. As the parties act within the environment, their actions are compared to the obligations set forth within the contract, and any violations are raised, leading to the \textit{contract enforcement} stage in which further obligations, and possibly sanctions, come into force. Finally, the \textit{contract termination} stage is reached, in which the contracting parties are freed from their contractual obligations, and the contract is deemed to no longer be in force.

In this thesis, we focus on the contract enforcement and monitoring stages. Some of the languages we examine below are useful in other contracting stages, however, and we will note this where applicable.

To be useful, a contracting language should be sufficiently descriptive to allow us to represent contract-specific concepts, as well as to be able to deal with those parts of the environment which are referred to within the contract (Governatori and Rotolo, 2004). More specifically, a contracting language must

1. Identify the parties involved in the agreement.

2. Refer to the temporal aspects of the contract, including durations and time instants to allow for the description of

   (a) When the contract comes into force.

   (b) When the contract terminates.

   (c) When certain clauses are in effect.

3. Provide support for the concept of an agent undertaking an action.

4. Allow for the description of norms (obligations, prohibitions, permissions and the like) imposed on each party.

5. Provide a way of representing rewards and penalties.

6. Allow for contrary-to-duty obligations (e.g. since rule $a$ was broken, rule $b$ is now in effect).

7. Refer to the state of other agents, contracts, and the environment.

8. Allow for the conditional evaluation of obligations and contract elements, based on temporal, action and state elements.
Contracting languages can be broadly subdivided into two distinct classes. The first, referred to as “ad-hoc” languages, have only informal, or implicitly specified semantics. The most obvious example of such a language is natural language. WS-Agreement (Andrieux et al., 2006) is another, very popular ad-hoc contracting language intended for use in Grid and Web Service based environments. In fact, WS-Agreement is very little more than syntactic sugar used to encapsulate another, more concrete language such as SWRL (Wu and Jin, 2005). DPL (Milosevic and Dromey, 2002) uses ideas from deontic logic, but also has ad-hoc semantics. Nevertheless, it is intended to be used to aid contract consistency checking. It suggests that a monitoring agent can be constructed that can parse a DPL contract and perform contract monitoring.

The second class of contracting languages consists of those with formal underpinnings. Most of these languages incorporate some form of deontic logic, and are then enhanced to provide support for additional contracting concepts. As described in (van der Torre, 2003), “Standard” deontic logic (SDL), is a normal KD modal system that is closed under Modus Ponens and necessitation.

Within a deontic logic, obligations are typically represented using a modal formula of the form $Oa$. Such a sentence can be interpreted as meaning that an obligation exists to achieve the state of affairs $a$. Permissions are usually taken to be the dual of obligations, and can thus be defined as $Pa \equiv ¬O¬a$.

A contrary to duty obligation arises only when another obligation has been violated, and normally takes the form

\[
Oa
\]

\[
¬a → Ob
\]

For example, an obligation to pay a late fee is a contrary to duty obligation based on an original obligation to return a library book on time. While SDL is both simple and elegant, its representational power is insufficient to represent many situations. For example, In SDL contrary to duty obligations quickly lead to paradoxes (Prakken and Sergot, 1996). Dilemmas also lead to paradoxes, and while the latter should not occur within consistent systems, an agent may encounter them when reasoning about multiple, or poorly drafted contracts.

Many extensions have been proposed to SDL. For example, van der Torre (2003) defines a system called Contextual Deontic Logic (CDL). CDL is a dyadic logic, and obligations are written as $O(a|b)_L$ which can be interpreted as “$a$ should be the case if $b$ is the case. $L$ is a label, which is used to overcome various problems with the logic upon which CDL is built. CDL is able to represent contrary to duty obligations. However, it still lacks a number of important features, such as the ability to assign obligations to different parties and the inability to handle temporal concepts.

LCR (Dignum et al., 2002) is a very representationally rich extension of temporal and denotic logic, capable of representing contrary-to-duty obligations, violations, and obligations containing durations and deadlines. Since LCR was designed to operate in a multi-agent environment, obligations are associated with specific agents. Finally, the authors show that LCR does not succumb to the paradoxes that many other deontic logic based approaches suffer from. Given these attributes, LCR could form the basis of a contracting language, and the authors show how reasoning about
the execution of a contract could take place in the language. Essentially, this consists of comparing the environment state to the specified state, and performing theorem proving. Clearly, while LCR could be used to represent a contract, and, given an environment, reason whether a violation has occurred, additional techniques would be needed to determine environmental state within a partially observable system.

While many other formal and informal languages exist (e.g. Governatori’s (2005) BCL, NoA Kollingbaum (2005)), most gloss over the problem of determining the most likely environment state. Exceptions to this, in the context of contracting, are Daskalopulu et al. (2002), and Milosevic et al. (2002), where a framework is presented for contract enforcement built upon Daskalopulu’s work. Both of these approaches represent contracts as finite state machines, with transitions between states occurring due to agents’ actions, and states representing environment and contract states. Entities in the system may assign opinions to the agents’ actions, representing their belief that the action occurred. This approach makes use of a centralised adjudicator to whom opinions are reported, and whose role is to decide when state transitions should occur, as well as what state the system is in. Due to its statistical nature, it is unable to cater for exceptional situations.

Many other systems exist to determine the most likely environmental state, such as Bayesian networks (Pearl, 1988). However, to our knowledge, none have been applied specifically to the contract monitoring problem.

Having briefly surveyed some aspects of contracting, we will now describe Subjective Logic. We utilise Subjective Logic as a core part of our concrete argumentation framework (which we describe in Chapter 4), and will thus examine it in some detail.

### 2.3 Subjective Logic

Subjective Logic (Jøsang, 2001), abbreviated SL, provides a standard set of logical operators (such as negation, conjunction and disjunction), intended for use in domains containing uncertainty, and, more specifically, domains in which there are opinions regarding the truth or falsehood of a (set of) domain elements. Subjective logic also contains a number of other operators, designed to combine opinions in an intuitively correct manner.

Subjective Logic attempts to capture some notions that are difficult to address using probability theory. An informal example\(^3\) is perhaps the best way to illustrate this. Compare a gamble on the whether a head will turn up when a fair coin is tossed to a gamble on the winner of a fight between two expert martial artists of different disciplines. Assuming we don’t know much about martial arts, we would, intuitively, be less likely to take the second gamble than the first. Probabilistically however, we would be forced to represent the two gambles identically. The difference between the two cases lies in our confidence in the probabilities assigned to the events. In the first case, we are confident that we may assign a probability of 0.5 to each outcome. In the second case, we are uncertain as to how to assign the probabilities. By modelling this notion of uncertainty explicitly, we may represent such problems in more detail. We are often forced to reason about situations in which uncertainty of this form exists, and will thus integrate Subjective Logic into the framework described in Chapter 4.

In this section we will briefly describe subjective logic. Most of this description is based on

2.3. Subjective Logic

Jøsang’s original description of SL (Jøsang, 2001).

Since SL is based on Dempster-Shafer evidence theory, it operates on a frame of discernment, denoted by $\Theta$. A frame of discernment contains the set of possible system states, only one of which represents the actual system state. These are referred to as atomic, or primitive, system states.

In many situations, it is difficult to determine what state we are in, and it thus makes sense to talk about non-atomic (or non-primitive) states, consisting of the union of a number of primitive states. If the system is in primitive atomic state $x_i$, it is also in all states $x_j$ such that $x_i \subseteq x_j$. The powerset of $\Theta$, denoted by $2^\Theta$, consists of all possible unions of primitive states. A non-primitive state may contain other states within it. These are referred to as substates of the state.

Consider, for example, the frame of discernment shown in Figure 2.4, obtained when two coins are tossed. In this figure, all four primitive states are shown, but only three of the seven possible non-primitive states are represented. It should be noted that the entire frame of discernment is, itself, a non-primitive state.

An observer assigns a belief mass to various states based on its strength of belief that the state (or, assuming it is non-primitive, one of its substates) is true:

**Definition 2.20. (Belief Mass Assignment)** Given a frame of discernment $\Theta$, we can associate a belief mass assignment $m_{\Theta}(x)$ with each substate $x \in 2^\Theta$ such that

1. $m_{\Theta}(x) \geq 0$
2. $m_{\Theta}(\emptyset) = 0$
3. $\sum_{x \in 2^\Theta} m_{\Theta}(x) = 1$

For a substate $x$, $m_{\Theta}(x)$ is its belief mass.

Thus, for example, we could associate the following belief masses to the states shown in Figure 2.4:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>0.2</td>
<td>HT</td>
<td>0.1</td>
<td>TH</td>
<td>0.1</td>
</tr>
<tr>
<td>HH,TT</td>
<td>0.3</td>
<td>HH,HT</td>
<td>0.1</td>
<td>$\emptyset$</td>
<td>0.1</td>
</tr>
<tr>
<td>Other states</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Belief mass is an unwieldy concept to work with. When we speak of belief in a certain state, we refer not only to the belief mass in the state, but also to the belief masses of the state’s substates.
(in other words, when attempting to determine a level of belief that the world is in state $HH, TT$, we must also take into account the belief mass assignments to states $HH$ and $TT$). Similarly, when we speak about disbelief, that is, the total belief that a state is not true, we need to take substates into account. Finally, subjective logic also introduces the concept of uncertainty, that is, the amount of belief that we might be in a superstate or a partially overlapping state. We can define these concepts formally as:

**Definition 2.21. (Belief, Disbelief and Uncertainty)** Given a frame of discernment $\Theta$ and a belief mass assignment $m_\Theta$ on $\Theta$, we can define the belief function for a state $x$ as

$$b(x) = \sum_{y \subseteq x} m_\Theta(y)$$

where $x, y \in 2^\Theta$.

The disbelief function as

$$d(x) = \sum_{y \cap x \neq \emptyset} m_\Theta(y)$$

where $x, y \in 2^\Theta$.

And the uncertainty function as

$$u(x) = \sum_{\substack{y \cap x \neq \emptyset \\ y \not\subseteq x}} m_\Theta(y)$$

where $x, y \in 2^\Theta$.

In the above example, $b(HH, HT) = 0.3$, $d(HH, HT) = 0.3$, and $u(HH, HT) = 0.6$.

This can be informally interpreted as saying that it is equally likely that we are in a state where either two heads, or a head and a tail (in that order) were flipped, as we are to not be in such a state. However, we are not very confident (due to the high uncertainty) in the strength of our convictions regarding the likelihood of belief and disbelief. This is due to the way in which belief mass was assigned; the assignment of belief mass to the non-primitive $HH, HT$ state, as well as to states whose components intersect it lies at the root of this uncertainty.

These functions have a number of features that should be noted. First, they all range between zero and one. Second, they always sum to one, meaning that it is possible to deduce the value of one function given the other two. If the entire belief mass is assigned to $\Theta$, $u(x) = 1$ if $x \neq \Theta$. This situation is analogous to total uncertainty. Dogmatic beliefs (i.e. those with no uncertainty component) can arise when no belief mass is assigned to $\Theta$.

Another important concept, in addition to belief functions, is the concept of relative atomicity.

**Definition 2.22. (Relative Atomicity)** Given a frame of discernment $\Theta$ and $x, y \in 2^\Theta$, the relative atomicity of $x$ to $y$ is defined as

$$a(x/y) = \frac{|x \cap y|}{|y|}$$

if $y \neq \emptyset$, otherwise, it is undefined. For a state $x$, $|x|$ denotes the number of atomic states it contains.

Imagine a frame of discernment with four atomic states $x_1 \ldots x_4$. If the $m_\Theta(\Theta)$=1, the probability of any state holding (with no prior information) is 0.25. If, however, $m_\Theta(x_3) = 1$
where \( x_5 = \{x_1, x_2\} \), then the probability of \( x_1 \) rises to 0.5. The relative atomicity approximates the role of a prior probability distribution within probability theory, weighting the likelihood of some outcomes over others.

Given the definition of relative atomicity, it is possible to define the probability expectation function:

**Definition 2.23. (Probability Expectation)** Given a frame of discernment \( \Theta \) and a belief mass assignment \( m_\Theta \), the probability expectation function is defined by

\[
E(x) = \sum_{y \in 2^\Theta} m_\Theta(y)a(x/y)
\]

According to this definition, a belief mass assigned to a number of atomic states is split equally among the states. This assumption, based on the principle of insufficient reason, makes sense given no other knowledge regarding the prior likelihood of the various states.

Boolean logic operators have Subjective logic equivalents. It makes sense to use these equivalent operators in frames of discernment containing a state and its complement. A **focused frame of discernment** is a binary frame of discernment containing a state and its complement.

**Definition 2.24. (Focused Frame of Discernment)** Given \( x \in 2^\Theta \), the frame of discernment denoted by \( \tilde{\Theta}^x \), which contains two atomic states, \( x \) and \( \neg x \), where \( \neg x \) is the complement of \( x \) in \( \Theta \), is the focused frame of discernment with focus on \( x \).

**Definition 2.25. (Focused Belief Mass Assignment and Relative Atomicity)** Let \( \tilde{\Theta}^x \) be the focused frame of discernment with focus on \( x \) of \( \Theta \). Given a belief mass assignment \( m_\Theta \) and the belief, disbelief and uncertainty functions for \( x \) (\( b(x) \), \( d(x) \) and \( u(x) \) respectively), the focused belief mass assignment, \( m_{\tilde{\Theta}^x} \) on \( \tilde{\Theta}^x \) is defined as

\[
\begin{align*}
    m_{\tilde{\Theta}^x}(x) &= b(x) \\
    m_{\tilde{\Theta}^x}(\neg x) &= d(x) \\
    m_{\tilde{\Theta}^x}(\tilde{\Theta}^x) &= u(x)
\end{align*}
\]

The focused relative atomicity of \( x \) is defined as

\[
a_{\tilde{\Theta}^x}(x/\Theta) = [E(x) - b(x)]/u(x)
\]

For convenience, the focused relative atomicity of \( x \) is often abbreviated \( A_{\tilde{\Theta}^x}(x) \).

An opinion consists of the belief, disbelief, uncertainty and relative atomicity as computed over a focused frame of discernment:

**Definition 2.26. (Opinion)** Given a focused frame of discernment \( \Theta \) containing \( x \) and its complement \( \neg x \), and assuming a belief mass assignment \( m_\Theta \) with belief, disbelief, uncertainty and relative atomicity functions on \( x \) in \( \Theta \) of \( b(x), d(x), u(x) \) and \( a(x) \), we define an opinion over \( x \), written \( \omega_x \) as

\[
\omega_x \equiv \{b(x), d(x), u(x), a(x)\}
\]
For compactness, Jøsang also denotes the various functions as $b_x, d_x, u_x$ and $a_x$ in places, and we will follow his notation. Similarly, he denotes the probability expectation of an opinion as $E(\omega_x)$ rather than $E(x)$.

Given opinions about two propositions from different frames of discernment, it is possible to combine them in various ways. Combining them to form an opinion about the truth of both propositions holding simultaneously yields the following:

**Definition 2.27. (Propositional Conjunction)** Given two opinions $\omega_x = (b_x, d_x, u_x, a_x)$ and $\omega_y = (b_y, d_y, u_y, a_y)$ where $x$ and $y$ belong to distinct focused frames of discernment, we may compute the propositional conjunction, $\omega_{x \land y}$ as

\[
\begin{align*}
    b_{x \land y} &= b_x b_y \\
    d_{x \land y} &= d_x + d_y - d_x d_y \\
    u_{x \land y} &= b_x u_y + u_x b_y + u_x u_y \\
    a_{x \land y} &= \frac{b_x u_y a_y + u_x a_x b_y + u_x a_x u_y a_y}{b_x u_y + u_x b_y + u_x u_y}
\end{align*}
\]

The propositional disjunction of two propositions computes a joint opinion of the likelihood that at least one of the propositions holds:

**Definition 2.28. (Propositional Disjunction)** Given the opinions $\omega_x = (b_x, d_x, u_x, a_x)$ and $\omega_y = (b_y, d_y, u_y, a_y)$ where $x$ and $y$ belong to distinct focused frames of discernment, we may compute the propositional disjunction, $\omega_{x \lor y}$ as

\[
\begin{align*}
    b_{x \lor y} &= b_x + b_y - b_x b_y \\
    d_{x \lor y} &= d_x d_y \\
    u_{x \lor y} &= d_x u_y + u_x d_y + u_x u_y \\
    a_{x \lor y} &= \frac{u_x a_x + u_y a_y - b_x u_y a_y - b_x a_x b_y - u_x a_x u_y a_y}{u_x + u_y - b_x u_y - b_x b_y - u_x u_y}
\end{align*}
\]

The propositional negation operator calculates the opinion that a proposition does not hold, and is defined as follows:

**Definition 2.29. (Propositional Negation)** Given an opinion $\omega_x = (b_x, d_x, u_x, a_x)$, where $x$ belongs to a focused frame of discernment, we may compute an opinion $\omega_{\neg x}$, known as the propositional negation, as

\[
\begin{align*}
    b_{\neg x} &= d_x \\
    d_{\neg x} &= b_x \\
    u_{\neg x} &= u_x \\
    a_{\neg x} &= 1 - a_x
\end{align*}
\]

Apart from the standard boolean operators, Subjective Logic introduces two operators for dealing with opinions, namely the discounting and independent consensus operators.
Discounting is a model of hearsay. That is, given that an agent $\alpha$ holds an opinion $\omega_\beta^\alpha$ about agent $\beta$’s reliability, and given that $\beta$ has an opinion $\omega_x^\beta$ about $x$, $\omega_\alpha^{\omega_x^\beta \beta}$ gives the opinion $\alpha$ has about $x$. A typical use of discounting involves weighing evidence based on the credibility of the source reporting it. Discounting is defined as follows:

**Definition 2.30. (Discounting)** Given a pair of opinions $\omega_\beta^\alpha = (b_\beta^\alpha, d_\beta^\alpha, u_\beta^\alpha, a_\beta^\alpha)$ and $\omega_x^\beta = (b_x^\beta, d_x^\beta, u_x^\beta, a_x^\beta)$, the discounted opinion $\omega_\alpha^{\omega_x^\beta \beta}$ is computed as follows:

- $b_\alpha^{\omega_x^\beta \beta} = b_\beta^\alpha b_x^\beta$
- $d_\alpha^{\omega_x^\beta \beta} = b_\beta^\alpha d_x^\beta$
- $u_\alpha^{\omega_x^\beta \beta} = d_\beta^\alpha + u_\beta^\alpha + b_\beta^\alpha u_x^\beta$
- $a_\alpha^{\omega_x^\beta \beta} = a_\beta^\alpha$

The independent consensus operator represents the opinion an imaginary agent would have about $x$ if it had to assign equal weighting to the opinions $\omega_x^\alpha$ and $\omega_x^\beta$. For example, when multiple witnesses claim to have seen someone at the scene of a crime, the independent consensus operator could be used to compute a final (compound) opinion about the person’s presence, based on each witness’s testimony. Jøsang uses the following definition for independent consensus:

**Definition 2.31. (Independent Consensus)** Given two independent opinions $\omega_x^\alpha$ and $\omega_x^\beta$ about the same proposition $x$, the independent consensus opinion $\omega_x^{\alpha, \beta}$ is defined as

- $b_x^{\alpha, \beta} = (b_x^\alpha b_x^\beta + b_x^\alpha u_x^\beta) / \kappa$
- $d_x^{\alpha, \beta} = (d_x^\alpha d_x^\beta + d_x^\alpha u_x^\beta) / \kappa$
- $u_x^{\alpha, \beta} = u_x^\beta u_x^\alpha / \kappa$
- $a_x^{\alpha, \beta} = a_x^\beta u_x^\alpha - a_x^\alpha u_x^\beta - (a_x^\alpha + a_x^\beta)u_x^\alpha u_x^\beta /

Where \( \kappa = u_x^\alpha + u_x^\beta - u_x^\alpha u_x^\beta \) such that \( \kappa \neq 0 \) and \( a_x^{\alpha, \beta} = (a_x^\alpha + a_x^\beta) / 2 \) if \( u_x^\alpha = u_x^\beta = 1 \)

We introduce a number of infix operators to simplify notation:

- $\omega_x \land y \equiv \omega_x \land \omega_y$
- $\omega_x \lor y \equiv \omega_x \lor \omega_y$
- $\omega_x \neg \equiv \neg \omega_x$
- $\omega_x^{AB} = \omega_x^A \otimes \omega_x^B$
- $\omega_x^{A \oplus B} = \omega_x^A \oplus \omega_x^B$

Subjective Logic has been criticised due to the fact that quantifying attributes and assigning them opinions can be difficult in many situations, and, in some cases, differs from the way humans appear to reason. Furthermore, additional Subjective Logic operators exist which capture additional ways of utilising opinions (see for example (Jøsang et al., 2005)). The operators defined
here, as well as these additional operators, are thus by no means unique, and were created as they seem to (usually) agree with our intuitions, and satisfy the basic requirements of belief masses. These issues do not hamper our use of SL, for reasons discussed in Section 4.6.

The concrete framework we propose in Chapter 4 makes extensive use of the notion of evidence obtained from fallible sensors. From the discussion above, it should be clear that operators such as discounting and independent consensus prove important when reasoning about such evidence, and Subjective Logic thus forms a natural basis for our framework.

2.4 Summary

In this chapter, we examined various aspects of argumentation that are applicable to this thesis. Dung’s abstract argument framework, and Amgoud et al.’s bipolar argument frameworks were described in detail, while Pollock’s argument framework was also examined. Other issues of argumentation that are relevant to this thesis were described. These include argument schemes, burden of proof, dialogue games and heuristics for argument.

In the next two chapters, we introduce an abstract and concrete argument framework which takes into account evidential reasoning. The abstract framework has many similarities to bipolar argument frameworks, but overcomes many of the criticisms levelled against BAFs, and allows us to reason about arguments supported by some form of evidence.

Our concrete framework is based on Subjective Logic, and supports many important concepts found in argument, such as argument schemes and burden of proof. As part of the framework, we introduce a dialogue game allowing agents to advance both arguments, and evidence from within the environment, so as to support their case. Finally, Chapter 4 also introduces some heuristics for argument, letting the agents decide which arguments and evidence they should advance when they have the opportunity to do so.

In Section 2.2, we examined various aspects of automated contracting, including the desiderata for a contracting language, and how a number of existing contracting languages meet, or fail to meet these desiderata. We also discussed some issues relating to contract monitoring and enforcement.

We make use of contracting in Chapter 5, introducing a new contracting language, and showing how our concrete framework may be used to perform reasoning about the status of a contract within a partially observable domain. This application of our framework illustrates the power of argumentation based techniques in reasoning about situations with partial information, and in performing this reasoning in a distributed manner.
Chapter 3

Evidential Argumentation Frameworks

3.1 Introduction

In this chapter, we introduce an abstract argument framework designed to allow reasoning about evidence to take place. Our framework contains both the notions of attack and support. Unlike other frameworks which cater for both these notions, such as BAFs, our notion of support is rooted in the idea of evidence. This allows us to overcome the criticisms levelled at BAFs, and model interesting interactions between arguments. We begin by examining the notion of support in more detail.

3.2 The Need for Support

A number of researchers have questioned the need for support in a Dung style abstract argument framework (Bench-Capon, 2007). One of the reasons for this is that they argue that support is implicitly built into the argument system in two places:

1. Within a Dung style framework, the existence of an argument means that it is, by default, supported.

2. The logic used when instantiating an argument provides support for the arguments that appear in the argument system.

Furthermore, they claim that an argument framework without a support relation is more concise than one where the relation is present. As an example, consider the following set of statements:

\[ p, q, p \rightarrow q \]

An abstract argument framework does not specify how statements should be translated to arguments. One could represent each statement individually, or group them in some way, perhaps representing inference. Figure 3.1 shows three possible partitions of these statements as arguments. Without the notion of support, only the rightmost partition can be used (as the other partitions have arguments supporting other arguments). When representing all statements in a single argument, we may interpret that single argument as “By default, \( p \) is true, \( p \) implies \( q \), and therefore, \( q \)”. By introducing the notion of support, we could end up with the left hand set of arguments, informally stating “\( p \) is true, and that gives support to the argument that since \( p \) implies \( q, q \)” The middle partition within the graph can be interpreted as “We know that \( p \) holds, and that \( p \) implies \( q \). This argument provides support for the argument that \( q \) holds”.
3.2. The Need for Support

Only if the statement \( \neg p \) is introduced does the original compound argument need to be supported in some way. In this case, argument \( p \) needs to be introduced separately, and, given a framework where no explicit support links exist, we obtain the situation shown in Figure 3.2. When this occurs, the rebutting attacks between \( p \) and \( \neg p \) means that the original argument is reinstated. Thus, an attack based argument framework is able to naturally and elegantly capture the notion of defaults. In such frameworks, an argument is, by default, true unless it is attacked.

Another reason suggested for not using support within abstract argument frameworks is that support between arguments requires too much knowledge about the internal structure of an argument. The multiple possible partitions of a set of arguments such as the one described above, or a more complicated set such as \( a, a \rightarrow b, b \rightarrow c, \ldots y \rightarrow z, z \), is viewed as a weakness, rather than a strength of argument systems containing support. According to critics of support, the notion of attack, due to its similarities with default reasoning, should be captured at the argument level, while support should be relegated to the level of the logic which makes up the arguments due to its similarity to inference.

In summary, the detractors of support in abstract argument frameworks claim that support should not form part of an abstract argument framework because

---

Figure 3.1: Possible representations in an argument framework of the sentence \( p, p \rightarrow q, q \). Solid arrows represent the notion of support (or inference).

Figure 3.2: Representing support implicitly in an argument framework containing only an attack relation.
1. Representing support at the argument level leaves too many choices regarding how to represent argument.

2. Support represents inference, which should be at the level of logic, not argument.

3. Abstract argument frameworks represent support implicitly through the notion of defaults; if an argument is in the framework, it is, somehow, supported.

The first argument against support could equally be deemed an argument for support. While a number of representational choices open up, it is possible to get rid of the redundancy (c.f. the argument $p$ in the example discussed above) that can appear in attack only argument frameworks. Similarly, by adopting the maxim “represent arguments in the simplest form possible”, one can often eliminate a lot of the choices that may appear.

Points 2 and 3 are somewhat related, and assume that support is used to represent the inference mechanism. While this is often the case, particularly in BAFs, support has another role, namely to allow us to distinguish between prima facie and standard arguments. Prima facie arguments do not require any sort of support from other arguments to stand, while standard arguments must be linked with at least one prima facie argument via a support chain. It is this concept that we formalise in our framework below.

Finally, when people reason about argument, they often think of the interactions between support and attack. Therefore, when modelling human argument, it makes sense to have an argument framework that treats the two as equals.

As mentioned in the previous chapter, BAFs, while allowing for support, use it to model inference. Arguments may stand with, or without support, meaning that inference in BAFs may take place both internally to the argument, and outside it. The criticisms described above thus apply directly, and we believe legitimately, to Bipolar Argumentation Frameworks. Our notion of support is subtly but importantly different; we wish to capture the notion of evidential support; we introduce a set of semantics that allow us to reason about arguments that are directly supported by evidence (or are prima facie arguments), and arguments that have been inserted into the framework due to a chain of evidential support from a prima facie argument. This is a new, and important notion of support.

### 3.3 Reasoning with Support

Consider the following set of arguments:

- $a$ It was dark, when it is dark, a witness’s statement could be wrong.

- $b$ Witness $b$ made the statement that the bird flew. A witness’ statement can be viewed as evidence.

- $c$ Witness $c$ made the statement that the bird did not fly. A witness’ statement can be viewed as evidence.

- $x$ We know that birds can normally fly, and thus given some evidence, we may claim that birds fly.
A BAF might represent these arguments as shown in Figure 3.3. Here, and in the rest of the chapter, solid arrows indicate support, while dashed arrows indicate attack. Arguments $b$ and $c$, while resulting in conflicting evidence, do not attack each other, as it is possible for different witnesses to advance both of their claims simultaneously. It should also be noted that including argument $b$ or $c$ with argument $x$ makes little sense, and the figure thus represents a very reasonable way of partitioning the arguments.

Given these arguments, we would intuitively expect the various extensions to contain the arguments $\{a, x\}$. However, only argument $a$ appears in any BAF based extension. This is because $a$ both set-defeats (through $b$), and set-supports (through $c$) argument $x$.

Now assume that instead of arguments above, we were presented with the following arguments:

- $a'$ It was dark, when it is dark, a witness’s statement could be wrong.
- $b'$ Witness $b$ said he saw a man fly by waving his arms. A witness’ statement can be viewed as evidence for a claim.
- $c'$ Witness $c$ said he saw a man waving his arms, but the man didn’t fly. A witness’ statement can be viewed as evidence for a claim.
- $x'$ From the evidence, we can conclude that the man flew.

In this case, the argument graph would remain the same (with primed arguments replacing unprimed arguments). In this case however, we would only want to accept argument $a'$. This time, we have no basis, in the sense of tautological support, default, or prima facie argument for $x'$. Thus, we cannot determine the status of $x'$ one way or another.

In this case, the standard BAF extensions would provide us with the appropriate result. With no notion of support, Dung’s framework would be unable to represent the problem. However, by looking only at the attack links, it would derive arguments $\{a, x\}$ in its extensions.

In this section, we refine BAFs with additional extensions so that these extensions agree with our intuitions. To do so, we introduce two main concepts:

1. We add the notion of support by the empty argument, representing evidence from the environment for a certain argument, or the presence of a prima facie argument.
2. Given this type of support, we introduce a number of new extensions for evidential reasoning.

We also extend BAFs in another way by allowing for cases where multiple arguments are required to support or attack another argument. This enhancement is based on Nielsen and Parsons (2006b)’s similar work in Dung’s argumentation framework.

In a BAF based framework, support is, to some extent, optional. With no support, the framework reduces to a Dung-like framework. In our framework, support is mandatory. Arguments may only be advanced if they are supported. Initially, this support must be provided by what we refer to as the environment. This initial support may arise due to a number of possibilities, including a prima facie argument, a default, a tautology, or some evidence that is beyond question.

3.3.1 Arguments, Attack and Support

In our framework, an argument is accepted only if it supported through a chain of arguments, each of which is themselves supported. At the head of this chain is an argument representing support from the environment (written as support from the special state \( \{ \} \)), and to represent this notion, we must redefine argument frameworks as follows:

**Definition 3.1. (Evidential Argumentation Systems)** An evidential argumentation system is a tuple \( (A, R_a, R_e) \) where \( A \) is a set of arguments, \( R_a \) a relation of the form \( (2^A \setminus \{ \}) \times A \), and \( R_e \) a relation of type \( 2^A \times A \).

Within the argumentation system, \( \exists x \in 2^A, y \in A \text{ such that } x R_a y \text{ and } x R_e y. \)

The \( R_e \) and \( R_a \) relations encode evidential support and attacks between arguments. An element of the support relation of the form \( \{ \} R_e a \) would represent support by the environment for the argument \( a \). Within our argument framework, we are interested in seeing which arguments may eventually be considered to hold. Since any argument attacked by the environment will be unconditionally defeated, we believe that it makes no sense to include such arguments, and therefore prohibit the environment from appearing in the attack relation. Since the environment requires no support, \( \{ \} \) may not appear as the second element of a member of \( R_e \).

For an argument \( a \), if \( \{ \} R_e a \), we say (in a slight abuse of the English language) that \( a \) has environmental support. Environmental support can be used to model defaults (since an argument that has environmental support is true unless attacked), tautologies (if the argument may never be attacked), and arguments that have incontrovertible evidence in their support. We defer in depth discussion regarding the form of such incontrovertible evidence to Chapter 4 and 5. Briefly however, we assume that evidence from the environment that cannot be challenged, such as an observation by an infallible sensor, can be thought of as such evidence\(^1\).

Unlike BAFs, we allow for attacks and supports by sets of arguments. This allows us to easily differentiate between conjunctive and disjunctive reasons for an argument (e.g. “I will perform action A is it is both sunny and warm” versus “I will perform action A if it is either sunny, or warm”), and will be useful when attempting to represent accrual of arguments.

---

\(^1\)Philosophers such as Pollock (1995; 1998b) have examined the role of defeasible reasoning in perception in detail (using a concrete argument framework), and in their work, all evidence may be challenged. We may still model this in our framework by having environmental support to the challengeable evidence. In such a situation, the evidence is accepted as a default, rather than as being unchallengeable.
A set $S$ is said to attack an argument $a$ iff there is a $S' \subseteq S$ such that $S'R_a a$. $S$ is a minimal attack on $a$ if there is no subset of $S'$ that also attacks $a$. We say that a set $S$ attacks a set $S'$ if $S$ attacks one of the members of $S'$. Similar notions exist for support, but are complicated by the fact that an argument is not supported by simply appearing in the $R_e$ relation. In an evidence based approach, an argument is only acceptable if it is supported by another supported argument, or by evidence from the environment (self support in this framework does not count as support).

Definition 3.2. (Evidential Support) An argument $a$ is supported by a set $S$ iff

1. $SR_e a$ where $S = \{\}$ or
2. $\exists S' \subseteq S$ such that $S'R_e a$ and $\forall x \in S', x$ is supported by $S\setminus\{x\}$

$S$ is a minimum support for $a$ if there is no $S' \subseteq S$ such that $a$ is supported by $S'$.

Figure 3.4 illustrates the notion of support; argument $x$ is a supported argument, as it is supported (shown by bold links) by the chain $\{\}, b, x$.

This notion of support is stronger than the one present in BAFs. It requires evidence at the start of a chain of support (i.e. $\{\}R_e x$ for some argument $x \in S$) which leads, through various arguments to $a$, before the argument $a$ may be used. With this notion, we may define the notion of a supported attack:

Definition 3.3. (Evidence-Supported Attack) A set $S$ carries out an evidence-supported attack on an argument $a$ if

- $XR_a a$ where $X \subseteq S$, and,
- All elements $x \in X$ are supported by $S$.

An evidence-supported attack by a set $S$ is minimal iff there is no $S' \subseteq S$ such that $S'$ carries out an evidence-supported attack on $a$.

Where the context is clear, we will refer to evidence-supported attacks as supported attacks.

A supported attack attempts to represent the notion of an attack backed up by evidence or facts. The concept of an evidence-supported attack is necessary for our definition of acceptability. We illustrate a supported attack in Figure 3.5.

Support for an argument is clearly one requirement for acceptability. However it is not enough. Following Dung, an argument should be acceptable (with respect to some set) if it is defended from attack by that set. The question arises whether all attacks should be defended
Figure 3.5: The set \{\}, \{a\} carries out a supported attack on b (shown by the bold links). This attack undermines support for y, as shown in grey.

against, or only supported attacks. By choosing the latter, we allow an argument to be defended from attack by either having the attack itself attacked, or by having any means of support for the argument attacked by the defending set.

**Definition 3.4. (Acceptability)** An argument a is acceptable with respect to a set S iff

1. S is a support for a.
2. Given a minimal supported attack \(X \subseteq \mathcal{A}\) against a,
   - \(\exists Y \subseteq S\) such that \(Y R_a x\), where \(x \in X\), or
   - \(\exists Y \in S\) such that \(Y R_a x\) where \(x \in X\) so that \(X \setminus \{x\}\) is no longer a supported attack on a.

An argument is thus acceptable with respect to a set of arguments S if any argument that support attacks it is itself attacked (either directly, or by being rendered unsupported) by a member of S. The set S must also support the acceptable argument. It is clear from this definition of acceptability that an asymmetry arises with regards to acceptability, as we ignore elements of S that might themselves be attacked by other arguments, which would prevent S from supporting a. To overcome this problem, the notion of admissibility is required, and for this, we must first define two other notions.

**Definition 3.5. (Conflict free and Self Supporting Sets)** A set of arguments S is conflict free iff \(\forall y \in S, \not\exists X \subseteq S\) such that \(X R_ay\).

A set of arguments S is self supporting iff \(\forall x \in S, S\) supports x.

It should be noted that this definition of a conflict free set is very strong. A weaker definition is possible by defining a conflict free set as one not containing any supported attacks on itself. Since we intend to use the concept of a conflict free set in our definition of admissibility, we will show in Lemma 3.2 that for our purposes, these two definitions of a conflict free set coincide.

The following lemma will be used later:

**Lemma 3.1.** If a set is minimum support for an argument, then the set is self supporting.

**Proof.** Follows directly from point 2 in the definition for evidential support and the definition for minimal support.

The concept of admissibility can now be defined:
Figure 3.6: An argument system showing supported arguments \(a, b, c\) and \(d\), support attacks \(a, b\) attack \(d\), and \(c\) support attacks \(a\). Argument \(d\) is acceptable with respect to \(c\), and \(\{c, d\}\) is an admissible set.

**Definition 3.6. (Admissible Set of Arguments)** A set of arguments \(S\) is said to be admissible iff

1. All elements of \(S\) are acceptable with respect to \(S\).
2. The set \(S\) is conflict free.

Figure 3.6 illustrates these concepts. Here, \(a, b, c, d\) are all supported arguments. The set of arguments \(\{a, b\}\) support attacks \(d\), while \(c\) support attacks \(a\). Argument \(d\) is acceptable with respect to \(\{c\}\), since \(c\) prevents \(b\) from being supported. Finally, the set \(\{c, d\}\) is an admissible set.

The following useful results can then be shown:

**Lemma 3.2.** An admissible set contains no supported attacks on itself.

*Proof.* By contradiction: given an admissible set \(S\), assume that it contains a supported attack on itself. Then there is a \(X \subseteq S, y \in S\) such that \(XR_{a,y}\) and all elements of \(X\) are supported by \(S\). But, by the definition of acceptability, that there is another set \(U \subseteq S\) that (support) attacks an element of \(X\), meaning that \(S\) is not conflict free, and hence not admissible.

**Lemma 3.3.** An admissible set is self supporting.

*Proof.* Since all arguments in the admissible set are acceptable with respect to the admissible set, and since all acceptable arguments are supported, the admissible set must support itself.

**Lemma 3.4.** The empty set is admissible.

*Proof.* The empty set is acceptable as nothing may attack it, and it supports itself. For the same reason, it is conflict free and self supporting.

**Lemma 3.5.** Given an admissible set \(S\), and two arguments \(x, y\) which are acceptable with respect to \(S\),

1. \(T = S \cup \{x\}\) is admissible,
2. \(y\) is acceptable with respect to \(T\).

*Proof.* (1) Since all arguments in \(T\) are acceptable, it is sufficient to show that \(T\) is conflict free. Assume the contrary. Then there exists a set \(U \subseteq T\) and an argument \(z\) such that \(U\) minimal support attacks \(z\). Since all elements of \(T\) are defended by \(S\) (due to the acceptability of the
elements), $S$ must support attack an element of $U$. Let this element be labelled $w$. If $w \in S$, then $S$ is not conflict free, leading to a contradiction.

Otherwise, $w \equiv x$, in which case, since $x$ is admissible with respect to $S$, $S$ must attack itself, thus not being conflict free, and again leading to a contradiction.

Since each argument in $T$ is acceptable, $T$ is an admissible set.

(2) since $S$ supported $y$, $T$ supports $y$. Since $S$ defended $y$ from all attack, $S'$ must also defend $y$ from all attack. Thus, $y$ is acceptable with respect to both $S$ and $T$. □

### 3.3.2 E-Preferred Extensions

We are now in a position to define a modified version of Dung’s preferred extension. We call it the evidential preferred extension, or e-preferred extension, and define it as follows:

**Definition 3.7. (Evidential Preferred Extensions)** An admissible set $S$ is an evidential preferred extension if it is maximal with respect to set inclusion. That is, there is no admissible set $S'$ such that $S \subset S'$.

**Lemma 3.6.** Any argumentation system has at least one e-preferred extension.

*Proof.* Follows trivially from the fact that the empty set is admissible. □

Some e-preferred extensions can be computed by starting with the empty set, and adding arguments while maintaining admissibility. While calculating all preferred extensions for an argument system is still difficult (Dimopoulos et al., 1999), we believe that techniques such as those proposed by Nielsen and Parsons (2006a) are directly applicable to our framework.

It should be noted that Dung’s preferred semantics can be captured in our framework by having support exist only between the empty set, and all other arguments in the system.

There is no trivial transformation between BAF d/s/c-preferred semantics and our e-preferred semantics (or, as will be seen when other semantics for the evidential framework are described, between those and any BAF semantics). The reason for this is that the type of consistency BAF semantics describe is completely different to the consistency we are interested in.

A set of arguments is deemed “compatible”, and thus part of an extension in one of the BAF semantics if, in some sense, none of the arguments in the extension both support, and attack some argument. Two types of compatibility were identified, namely internal, and external compatibility. Internal compatibility occurs when an argument is not accepted into the set if it would lead to the defeat of a member of the set. External compatibility on the other hand means that an argument would not be accepted into a set because its acceptance would mean that an argument outside the set would be both directly, or indirectly, supported and attacked by members of the set.

We do not see this simultaneous attack and defence as a problem, additional evidence may allow us to still have a consistent set of arguments in this case (as shown by the semantics introduced here).

Since Cayrol and Lagasquie-Schiex (2005b) only considers acyclic graphs, preferred, stable and grounded semantics overlap. In such a situation, a sceptical and credulous reasoner would both reach the same conclusion. However, since we are dealing with arbitrary graphs of arguments, we need to differentiate between the two types of reasoners. Preferred extensions capture the
3.3. Reasoning with Support

conclusions a credulous reasoner would reach. Before looking at e-grounded extensions, which represent sceptical reasoning in our framework, we examine how stable semantics would appear in an evidential argument framework.

3.3.3 E-Stable Extensions

Dung defines a stable extension of an argument framework as the set that attacks all arguments not found within the set. While the same definition could be used here, the notion of support allows for the definition of another aggressive extension, which will be referred to as the evidential stable extension (or e-stable extension). Informally, an e-stable extension is one that ensures that nothing but the e-stable set can be derived. This can be achieved by attacking arguments not in the set, or by attacking the support for those arguments.

**Definition 3.8. (E-stable Extensions)** An e-stable extension of an argument framework $A$ is a conflict free and self supporting set $S$ such that for any supported argument $a \notin S$,

1. $S$ (support) attacks $a$ or

2. $\exists S'$ such that $S'$ minimally supports $a$, $S$ (support) attacks $S'$.

As in stable extensions, an argument system may have zero, one, or many stable extensions. The following result allows us to compute a system’s e-stable extensions:

**Lemma 3.7.** A set $S$ is an e-stable extension iff $S = \{a|a$ is not support attacked by $S$ and is supported by $S\}$, and $S$ does not contain only the empty argument.

**Proof. only-if:** Assume that $S$ is an e-stable extension. Since it is self supporting, any argument $a \in S$ must be supported by $S$. Since an e-stable extension is conflict free, none of its arguments may be attacked by $S$. Since the empty argument cannot attack any arguments, it cannot, alone, be a member of an e-stable extension.

Hence if $S$ is an e-stable extension then its arguments are not support attacked by $S$, but are supported by $S$, and do not contain the empty argument alone.

**if:** Since any argument $a \in S$ is supported by $S$, $S$ is self supporting. Similarly, since $a$ is not support attacked by $S$, $S$ is conflict free. Now, for any argument $b \notin S$, $b$ is either

1. attacked by $S$, or
2. not supported by $S$.

If it is attacked by $S$, then it is clearly not part of the e-stable extension. If, however, it is not supported by $S$, then either it is unsupported, in which case it is not part of the e-stable extension, or it is minimally supported by some set $T$. $T$ is not a subset of $S$ (else $b$ would be supported by $S$).

Now there exists a set $U = T - S$. Since $U$ is not in $S$, its elements are either attacked by $S$, or not supported by $S$. If they are attacked, then the second condition of the e-stable extension is fulfilled (and the relevant elements cannot be part of the e-stable extension).

If they are not supported by $S$, we can proceed recursively to determine that their support is, at some stage, attacked by $S$.

Therefore, if $S = \{a|a$ is not support attacked by $S$ and is supported by $S\}$, and $S$ does not contain only the empty argument, then it is an e-stable extension. □
3.3.4 E-Grounded Extensions

Another widely used extension is the grounded extension. A grounded extension represents the arguments a sceptical reasoner would accept as true. Following Dung’s approach, we introduce a similar concept; the evidential grounded extension. To do so, the characteristic function of an argumentation framework must be defined:

**Definition 3.9. (The Characteristic Function)** The characteristic function of an argumentation framework $A$ is the function

\[
F_A : 2^{AR} \to 2^{AR}
\]

\[
F_A(S) = \{a | a \text{ is acceptable with respect to } S\}
\]

The following lemmas then hold:

**Lemma 3.8.** A conflict free set $S$ of arguments is admissible iff $S \subseteq F_A(S)$

*Proof.* Since all elements of $S$ are acceptable, all that needs to be shown is that $F_A(S)$ is conflict free if $S$ is conflict free. By contradiction, assume there is a set $X$ and argument $y$ in $F_A(S)$ such that $X$ attacks $y$. Then there must be a set $X' \subseteq S$ which attacks an element of $X$. Due to the acceptability of $X'$ however, there must be an $X''$ which attacks an element of $X'$, meaning that $S$ is not conflict free. 

**Lemma 3.9.** $F_A$ is monotonic with respect to set inclusion.

*Proof.* This lemma follows directly from Lemmas 3.5 and 3.8.

From this, we can define the evidential grounded extension:

**Definition 3.10. (E-grounded Extensions)** An evidential grounded extension of an argument framework $A$ is the least fixed point of $F_A$.

3.4 Discussion

Referring back to the example at the start of this chapter, we see that both instances could be represented by the left and right hand side (respectively) of Figure 3.7.
Since this argument system is acyclic, the e-preferred, e-stable and e-grounded extensions all overlap. Furthermore, this system has only a single extension, namely \{a\} for the argument with no default support for \(x\), and \{a, x\} when this support exists.

As briefly noted above, when support links exist only between the empty argument and all other arguments, the various extensions we have defined reduce to their Dung equivalents. This is because in this case, it is not possible to undercut support from an argument, and the only way to defeat it is by direct attack.

Similarly, when support for an argument exists through both the empty argument and through another argument, the argument will always be supported, and thus the second support link is redundant. More generally, an argument with an unattacked support chain may only be defeated by direct attack.

In our framework, support may exist between two arguments, or may be present as environmental support, that is support from the \{\} argument to another argument. In the former case, support represents a type of (not necessarily deductive) inference. In the latter case, support may be used to represent a default (or \textit{prima facie}) argument, a tautology (in which case the argument may not be attacked), an argument accepted on unchallengeable evidence, or some other type of argument which, unless attacked, should be considered true. Arguments not supported by the environment cannot be considered acceptable unless supported by a supported argument. While BAFs also have the notion of support, they do not differentiate between arguments as we do. Furthermore, arguments in BAFs do not have to be supported to be judged acceptable. Our differentiation between types of support, allowing us to distinguish between all types of default arguments and “standard” arguments, together with our requirement that a chain of support exist between arguments, allows us to perform reasoning that other frameworks cannot easily do. Support in our framework is thus a new, distinct and useful concept, and the framework allows us to both represent, and reason about it.

In this chapter, we introduced a new abstract argument framework. This argument framework, while drawing some inspiration from BAFs, refines the notion of support that is common to the former approach in a number of ways. First, we introduced an enhancement allowing us to represent the attacks and support by sets of, rather than individual, arguments. Second, we introduced a number of new extensions, based on Dung’s notion of preferred, grounded and stable extensions. These extensions allow us to define a new notion of acceptability based on chains of supporting arguments, starting from some unattackable argument (grounded in a \textit{prima facie} argument).

The intuition behind this notion of acceptability is two-fold. First, it is possible to think of the unattackable argument as some form of unquestionable evidence provided by the environment. It is also possible to view support from the empty set for an argument as that argument being a \textit{prima facie} argument. Such an argument is considered to hold unless it is explicitly shown to be attacked and defeated. Arguments that are not supported by the empty set are then those which require some justification, by means of a support chain, before they may be considered.

Support thus captures both the notion of inference (as is the case in BAFs), and the type of argument we are dealing with. Argument in our framework thus no longer involves only examining which arguments are consistent with each other in that they do not draw opposite conclusions (or attack each other in some other way), but also encompasses the idea that arguments are consistent...
with each other in that they allow inference to take place.

Others (such as (Verheij, 2007)) have examined the computational complexity of calculating the various extensions for a set of arguments in Dung’s argument frameworks. While we have not examined the issue in detail, many of the algorithms they suggest should be easily modifiable to operate on the framework presented here. We have developed a small proof of concept implementation of our framework written in Prolog. Our implementation calculates the e-preferred extensions of a set of arguments using a clumsy, brute force, approach. It determines which of the elements of the power-set of all arguments is an e-preferred extension.

The purpose of this thesis is to create a model of evidential reasoning that will allow agents to undertake inference in complex domains. Using the work of this chapter, an agent is capable of determining which sets of arguments may be deemed consistent, given that some arguments are supported by evidence, some arguments are, by default, considered true, and some arguments are supported by other, supported arguments. There are, however, a number of aspects we have not yet examined which are required if an agent is to perform reasoning in any sort of realistic domain. For example, we have simply assumed that evidence for certain arguments exists, without looking at how this evidence is obtained.

Abstract argument frameworks often ignore the concept of argument valuations. For example, given two differing opinions from witnesses regarding a certain occurrence, how do we choose which one to trust? The issue of weighting an argument quickly arises when reasoning is undertaken in most domains.

We have also not examined any of the dialogical aspects of reasoning. An agent may engage in an inner monologue to reason, or may be involved in arguing with others in an attempt to persuade them of its views.

In the next chapter, we introduce a concrete argument framework which addresses these, and other related issues. This framework allows an agent to construct arguments and inferences about its environment, obtain evidence regarding environmental state, and engage in dialogue with other agents about evidence, as well as to use evidence to buttress its beliefs. Finally, this framework addresses some issues regarding resource-bounded reasoning, suggesting some heuristics that an agent may use when engaging in dialogue. Afterwards, we show how the framework may be used to perform reasoning in a complex, highly applicable real-world domain where judgements about evidence are central to the reasoning process, namely the domain of contract monitoring.


Chapter 4

A Concrete Argumentation Framework

4.1 Introduction

While the abstract model introduced in the previous chapter is powerful, it has a number of shortcomings. Consider the following example (shown graphically in Figure 4.1):

- **a** Bob claims he saw Alice steal. We know that if $X$ claims they saw $Y$, $Y$ usually happened. Therefore, Alice stole.

- **b** Someone who steals is a crook. Therefore, Alice is a crook.

- **c** Bob might be unreliable. If someone is unreliable, their claims may be wrong.

```
| c ─── a ─── b |
|             |
|             |
|             |
```

Figure 4.1: An example illustrating the shortcomings of the abstract framework; the strength of the link between $c$ and $a$ is what allows us to decide which arguments to accept.

In this case, an evidential extension contains the argument $c$, allowing us to determine only that Bob may be unreliable; the ultimate decision rests on whether argument $a$ is strong enough to resist attack by argument $c$. Even if the arguments, on their own, are sufficiently strong to be acceptable, the decision regarding what to accept in our extension is dependant on the strength of support (and attack) between the arguments.

It has been suggested (Cayrol and Lagasquie-Schiex, 2005b) that the argument process follows three steps\(^1\). The first step consists of the exchange of arguments. Arguments are then weighted, or assigned valuations, after which the most acceptable arguments are selected.

It could thus be argued that, by following this process in the above example, we may accept or eliminate argument $c$ (based on the strength of evidence) before the evaluation process takes place. It would then be possible, based on the presence or absence of $c$, to have the evidential extension reach a conclusion that agrees with our intuition. One of the problems with this approach is

\(^1\)Leila Amgoud has refined this into a five step process (Amgoud, 2007).
that it is difficult to then compute a final strength for our conclusion; not only must we assign a valuation to \(c\), but also to \(a\) based on the strength of \(c\). Intuitively, we would probably say that “Alice is possibly a crook”, but the binary evaluation procedure does not allow us to reach this conclusion. Approaches such as value based argument frameworks (Bench-Capon, 2002) and preference based argument systems (Amgoud and Cayrol, 1997) have attempted to address some of the short-comings of the formal approach, but have difficulty with concepts such as the accrual of arguments.

Another possibility involves abandoning the third step of the argument process, and, instead, propagating argument weights across the argument network and reaching decisions based on these weights. Apart from lacking the formal elegance of the abstract framework, most weighting–based approaches are unable to deal with argument loops. Furthermore, it is difficult to integrate absolute arguments into a weighted framework. Consider for example the additional argument

\[ d \text{ Alice is very popular. Popular people are never crooks. Therefore, Alice is not a crook.} \]

It is clear that arguments \(b\) and \(b\) attack each other in such a way that either one, or the other must be true. In this case, we would, ideally, end up with two preferred extensions, each containing a different hypothesis. A purely weighted argument framework giving a result such as “Alice is 50% crooked”, a poor substitute representation for such a case. There thus appear to be two notions of possibility that must be dealt with. In one, which abstract argument frameworks can handle well, a possible argument is one that appears in some, but not all, preferred extensions. In the other, the “strength of possibility” is based on the weighting assigned to the argument. Thus, all three steps of the argument process are needed in at least some cases.

In this chapter, we introduce a concrete (in the sense that we specify the form arguments must take), weighted argument framework which we then integrate with the abstract framework introduced previously. The framework is built upon a Subjective Logic basis and supports a number of common dialogue features, namely argument schemes, burden of proof, as well as some classes of accrual of argument. An earlier version of this work appears in Oren et al. (2007a).

Following Prakken’s taxonomy, after introducing the logical and dialectic layers, we situate the lower layers within a dialogue at the procedural layer. The goal of this dialogue is to allow agents to argue about the state of a partially observable environment. Once this is done, we give some heuristics (in the aptly named heuristic layer) that allow agents to reason about what utterances they should make for a very specific class of dialogue.

As will be discussed later, the concrete argument framework, while containing some powerful features, also lacks a number of important attributes.

### 4.2 The Logic Layer

Unlike abstract argument frameworks, concrete argument frameworks constrain the form arguments may take. The form of arguments that we adopt is based on Toulmin’s model of argument, and thus, at their core, arguments consist of premises, which are linked to a conclusion through a warrant. Conclusions may then serve as premises to other arguments.

We assume that the framework will be used to argue about the state of an environment (as opposed to, for example, arguing about beliefs), and conclusions and premises thus represent a portion of the environment. Warrants, on the other hand, represent valid ways of combining
4.2. The Logic Layer

Figure 4.2: An argument graph consisting of evidence nodes \((\text{fastWater}(l), \text{rocks}(l), \text{mud}(l))\) and a warrant node.

premises to reach a conclusion, and therefore do not speak about environment state (though refer to the future work chapter for more about this assumption). When represented graphically, two types of nodes exist: evidential nodes, containing premises and conclusions, and inference nodes, containing an object we refer to as a warrant.

Evidence is, by its nature, uncertain. Different witnesses, for example, may claim that different things occurred, and sensor readings are subject to error. Decisions regarding the admissibility of an argument depend on the strength of evidence, and it thus makes sense to assign some form of weighting to evidence. In this framework, the weighting is based on Subjective Logic, and can be viewed as either a belief mass assignment, or as an opinion tuple

\[\langle \text{belief}, \text{disbelief}, \text{uncertainty}, \text{atomicity} \rangle\]

The obvious alternative to a Subjective Logic (SL) based approach is probability theory, especially since anything representable in SL can be represented using standard probability theory. However, much like the argument against the statement that “any program written in Smalltalk can be written in Assembly language”, SL appears to have the correct level of granularity to represent evidence. Notably, the distinction between belief/disbelief and uncertainty is very useful in domains where sensors might not be able to distinguish between different environmental states.

We assume that arguments speak about propositions within a language. The set of all possible sets of strings within the language is referred to as \(\Sigma\). All propositions have an opinion associated with them. Since we will commonly be representing interactions between arguments graphically, we refer to an \(\langle \text{atom}, \text{opinion} \rangle\) pair as an evidential node or evidence node.

The other type of node in the framework, referred to as a warrant node links evidence nodes to each other via directed edges. We are not yet in a position to define these concepts formally. We can, however, illustrate these concepts by means of an example: given the argument “Fast water and rocks mean no mud” and the atoms \text{fastWater}(l), \text{rocks}(l) and \text{mud}(l) which respectively represent fast water, rocks and mud at some location \(l\), we could graphically represent the argument as shown in Figure 4.2. It should be noted that this type of diagram prohibits us from introducing arguments about warrants. While our framework could easily be extended to handle such arguments (and other meta-arguments), we leave these issues for future work.

We can immediately ask what combination of evidential and warrant nodes, as well as the links between them, are permissible? To answer this question, the notion of an argument scheme must be introduced.
An argument scheme represents a commonsense form of reasoning. We will not try to capture all possible types of argument schemes, instead restricting ourselves to only those that can be represented using a specific format. More specifically, we introduce a language to represent argument schemes that allows us to determine what evidence may be linked via a warrant, where a warrant is an instantiation of an argument scheme.

When used in our framework, the term warrant is, in fact, a misnomer. In most work on argumentation (including Toulmin’s original work (Toulmin, 1958)), warrants represent the (general) rule used to make the inference. Warrants are thus normally associated with argument schemes, and are not instantiated. However, within our framework, a warrant is an instantiation of an argument scheme². Our warrants point to the applicable argument scheme used within an argument, and connect premises and conclusions in the same way as Toulmin’s warrants do, hence our use of the term.

Informally, this linking of warrants and evidence is achieved by assigning a type label to evidence. Warrants are then allowed to link evidence nodes based on a matching of their type.

Definition 4.1. (Types) A type is a (name, cardinality) pair, where name is an element from the propositional language of types Θ, and cardinality \( \in \mathbb{N} \).

The set of all possible types will be referred to as Types.

A type with a cardinality of 0 is called a token.

Types allow us to limit the application of argument schemes to certain pieces of data (i.e. to evidence nodes with a specific type). We could, for example, say that a “argument from expert opinion” argument scheme has to have, as one of its inputs, evidence from an expert (i.e. of type “expert”). Tokens represent proper nouns. For example, we may have a token “Bob”. Types with a cardinality greater than 0 represent a class. This class’s cardinality indicates how many field the class may have. Types may be used to represent specific elements from within our domain, in which case they are grounded, or to represent variables with specific properties. The latter normally arises when describing what evidence an argument scheme may operate on.

Definition 4.2. (Instantiated Type)

An instantiated type is a tuple (type, parameters) where type = (name, cardinality) is a type and parameters is an ordered set with cardinality elements. Each of these elements is one of the following:

1. A type
2. An instantiated type
3. The special negation operator “¬” followed by a type or instantiated type.

An instantiated type is grounded if all its parameters are either tokens or are themselves grounded types. Otherwise an instantiated type is called an abstract type.

Given an arbitrary labelling \( L \), we can define a surjective mapping \( M \) between an ordered bag with cardinality equal to that of the non-instantiated parameters of an instantiated types and the labelling. The range of this mapping (i.e. elements of \( L \)) are referred to as free variables.

The set of all grounded types will be referred to as GTypes.

²As shown later, this design decision allows us to easily represent undercuts between arguments.
When moving from argument schemes to warrants, instantiated types will be replaced by grounded types using a unification process. It thus makes sense to use a Prolog-like notation to refer to these types (by assuming that the elements of $L$ are strings beginning with capital letters). For example, given the token $nir$ (which in itself is the type $(nir, 0)$) and the types $(\text{hasThesis}, 1)$ and $(\text{holds}, 1)$, we may have a number of abstract types including $\text{holds}(X)$, $\text{holds}(\text{hasThesis}(X))$, and $\text{hasThesis}(\text{holds}(X))$. Examples of grounded types then include $\text{holds}(\text{hasThesis}(\text{nir}))$, and $\text{hasThesis}(\text{holds}(\text{nir}))$. The definition also allows for types of the form $\text{holds}(\neg X)$ and $\text{holds}(\neg \text{hasThesis}(\text{holds}(\text{nir})))$.

While a Subjective Logic opinion expresses both belief and disbelief regarding a state of affairs, the negation operator is still required. A single opinion can represent both the case that “It is sunny” and “It is not sunny” (i.e. sunny and $\neg$-sunny) through belief and disbelief. It cannot, however, represent concepts such as “If it is sunny, I carry an umbrella” and “If it is not sunny, I carry an umbrella” (i.e. $\text{implies}(\text{sunny}, \text{carriesUmbrella})$ and $\text{implies}(\neg \text{sunny}, \text{carriesUmbrella})$). This is because in the latter case, the states represented are not duals of one another, and it is for this reason that we allow the negation symbol to appear within an instantiated type.

From the definition above, unification of an abstract type may take place by substituting the appropriate grounded type in place of its label. It should be noted that an abstract type may be unified into an infinite number of grounded types (assuming the type has cardinality of 1 or greater). The decision problem of unification, namely whether an abstract type may be unified into a specific grounded type, and whether two abstract types may be unified into a grounded type, is of polynomial complexity (Baader and Snyder, 2001).

Given the above, we can define an evidential node (which we may also refer to as an evidence node) as follows:

**Definition 4.3. (Evidential Node)** An evidential node is a pair $(t, \omega_t)$ where $t \in \text{GT} = \text{GTypes}$ is a grounded type and $\omega_t$ is a $\langle \text{belief, disbelief, uncertainty} \rangle$ opinion triple.

It is important to note that at the level of evidential nodes, types can be viewed as a purely syntactic construct; by representing it as a Prolog string, a grounded type can be thought of as an atom within $\Sigma$. The set of all evidential nodes is referred to as $\text{EN} = \text{ENodes}$.

**Definition 4.4. (Argument Scheme)** An argument scheme $\text{AS} = (P, C)$ where $P$ and $C$ are both sets of abstract types. Any free variable appearing in $C$ must appear in $P$.

This definition of an argument scheme allows us to instantiate a warrant node from the scheme using unification. However, it is incomplete, and will be refined in Definition 4.7.

**Definition 4.5. (Warrant Node)** A warrant node is a tuple $(\omega_w, P, C, \text{AS})$ such that $P$ and $C$ are ordered sets of evidential nodes of the form $(t, \omega_t)$ such that $t$ is obtained by a unification within the warrant node’s argument scheme $\text{AS}$. $\omega_w$ is an opinion associated with the warrant itself representing a combination of its strength in the face of attack by other warrants.

As an example of the process of unification, consider the argument scheme representing Modus Ponens, $\text{MP} = ((\text{holds}(A), \text{implies}(A, B)), (\text{holds}(B))$ and the evidential nodes $(\text{holds}(\text{geologist}(\text{fred})), \omega_1), (\text{implies}(\text{geologist}(\text{fred}), \text{expert}(\text{fred}), \omega_2)$, and
holds(geologist(fred))

implies(geologist(fred), expert(fred)) ← ((holds(A), implies(A,B)), holds(B))

holds(expert(fred))

Figure 4.3: An illustration of how evidential nodes and argument schemes may interact with each other.

(holds\(\text{expert}(\text{fred})\), \(\omega_3\)). By applying unification, we could obtain the warrant node

\[
(((\text{holds}((\text{geologist}(\text{fred})), \text{implies}(\text{geologist}(\text{fred}), \text{expert}(\text{fred}))))), \\
(\text{holds}(\text{expert}(\text{fred}))))), MP
\]

When shown graphically, we will substitute the argument scheme for the warrant. Thus, the above example would result in the graph shown in Figure 4.3.

The language we have defined in this section can represent a subclass of argument schemes. The language, and hence argument schemes that may be expressed, have a number of weaknesses:

1. Only fixed length arguments can be represented. Many argument schemes, need to handle an arbitrary number of premises. For example, Modus Ponens can deal with one premise \((A, \text{implies}(A, B) \text{ therefore } B)\), two parameters \((A, B, \text{implies}(\text{and}(A, B), C) \text{ therefore } C)\), or an arbitrary number of parameters.

2. Expressing cardinality conditions is cumbersome. This relates to the previous point. stating that an argument requires no more, and no less than a certain number of premises requires very careful crafting of the scheme definition.

3. Disjunctions cannot be expressed in argument schemes as defined above. An argument scheme of the form “if \(A\) is the case, or, if \(B\) is the case, reach conclusion \(C\)” must be represented as two separate schemes. Disjunctions in the conclusion cannot be represented.

4. Since any variable in the conclusion must be present in the premises, premises must sometimes be specified that perform no function except to bind the variable in the conclusion\(^3\). For example, the unconditionally true argument stating that a geologist can read maps would have to be written as geologist\((X)\), map\((Y) \rightarrow \text{reads}(X, \text{map}(Y))\). Note also that a singleton object, namely map in this case, has to be represented as a type.

Some of these limitations may be overcome, or at least mitigated, through careful argument scheme and type design. For example, conjunctions and disjunctions can be represented by creating argument schemes to generate evidence nodes for “and\((X, Y)\)” and “or\((X, Y)\)” (with the appropriate variable bindings). These may then be fed into an argument scheme such as Modus Ponens (which would then have the form \(((\text{and}(A, B), \text{implies}(\text{and}(A, B), C), C))\). The “or\((X, Y)\)” evidence node could be used to handle some instances of disjunction. It should be noted however that

\(^3\)This situation can be seen in some of Walton’s argument schemes, where his minor premises are reflected in a scheme’s major premises or conclusion.
this approach simply shifts the problem one level. Groups of argument schemes must be created to generate the appropriate evidence nodes. For example, the scheme used to create \( \text{and}(A, B) \) could not handle the conjunction of three variables.

Despite its limitations, the language can capture many important and widely employed argument schemes. For example, the argument scheme capturing argument from expert opinion (Walton, 1996) can be represented as⁴:

<table>
<thead>
<tr>
<th>Name</th>
<th>ArgExpOp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Premises</td>
<td>( \text{expert}(E, D), \text{claims}(E, A), \text{inDomain}(A, D) )</td>
</tr>
<tr>
<td>Conclusions</td>
<td>( A )</td>
</tr>
<tr>
<td>Admissibility</td>
<td>true if ( \text{disbelief}(\text{expert}(E, D)) &lt; 0.5 ) and ( \text{disbelief}(\text{inDomain}(A, D)) &lt; 0.5 ) else false.</td>
</tr>
<tr>
<td>Mapping</td>
<td>( \text{claims}(E, A) )</td>
</tr>
</tbody>
</table>

Different admissibility and mapping functions may make more sense, depending on the domain in which the argument scheme will be used. Deciding on an appropriate form for these functions is one of the challenges of using our approach in practice.

Argument schemes give our framework a large amount of flexibility. Rather than depending on predefined inference rules, we may craft domain-dependent rules if the situation warrants. By tying the schemes into our Subjective Logic underpinnings and making use of our mapping and admissibility functions, we may use non-deductive modes of inference when undertaking reasoning using our framework.

So far, we have constructed a graph showing how warrant and evidence nodes are linked. However, notions such as argument strength, attack and support between evidence and warrants, and other concepts critical to the evaluation of argument have not been introduced. Most of these concepts belong to the dialectic layer of argument, which we shall now examine in more detail.

### 4.3 The Dialectic Layer

The sets \( P \) and \( C \) within a warrant represent the premises and conclusions of an argument. The role of the warrant is to provide an opinion about the argument’s conclusions based on the opinions associated with its premises. Since warrants are instantiations of argument schemes, they represent uses of the same argument in different situations (where these different situations are identified by having different premises). Thus, the function mapping opinions between premises and conclusions must be associated with the argument scheme rather than the warrant.

Arguments can both strengthen (by lending support) and weaken (by attacking) a conclusion. An argument may attack a conclusion by providing support for a mutually exclusive condition. An argument may also weaken the likelihood of a conclusion without proposing an alternative. The former case often arises in a binary situation, where support for a mutually exclusive conclusion is given. The latter can occur when multiple alternatives exist. Here, we may be unable to decide

---

⁴The Mapping and Admissibility components of the argument scheme are discussed in Definition 4.7.
between them, but simply be certain that the attacked conclusion is not the case. Using subjective logic, the first type of attack may be modelled by increasing the disbelief for a conclusion, while the second type of attack increases its uncertainty. In both cases, conclusion’s belief is reduced.

Another concept that must be taken into account at the dialectic level is that of accruals. Accrual of arguments can appear in multiple ways, and we attempt to support its most common applications in argument. As mentioned in the background chapter, Prakken has provided three desiderata that an accrual framework should satisfy. We do not attempt to create complete support for all types of accruals within this research, and thus take a different approach: we believe that two different types of accruals exist, and the difficulties researchers have had creating a single unified framework is that they have attempted to model both types of accruals using the same technique.

The first, and most common type of accrual arises when combining evidence. Informally, it can be described as follows:

- By default, \( \neg x \) does not hold.
- Arguments \( a_i \) where \( 0 < i < n \) individually support conclusion \( x \).
- All these arguments together strengthen the likelihood of \( x \).

This type of accrual typically arises when attempting to combine evidence from multiple sources. Examples include witness testimony in a court of law, the results of multiple experiments when attempting to confirm a scientific theory, and philosophical induction.

Another type of accrual which will be referred to in this thesis is one which defeats its members. This type of accrual arises when arguments are combined. Informally,

- Arguments \( a_i \) such that \( 0 < i < n \) individually imply conclusion \( x \).
- If all arguments \( a_i \) hold, then the conclusion \( \neg x \) holds.

This accrual is more unusual. It appears when the combination of arguments making up the accruals members lead to a different conclusion than its individual components. Prakken gives the following example for this accrual:

\[
a \quad \text{If it is hot, I won’t go running.} \\
b \quad \text{If it is rainy, I won’t go running.} \\
c \quad \text{If it is rainy and hot, I will go running.}
\]

In this situation, I will go running when it is hot and rainy, but will not go running if it is only hot (or only raining). Prakken handles accruals by introducing an extra inference step in his reasoning process (Prakken, 2005b). While powerful, his approach does not assign values to arguments, limiting its usefulness in concrete argument frameworks.

The first type of accrual must be addressed when looking at how opinions from different arguments combine. For example, given two pieces of evidence pointing to the fact that it is raining outside, we would have a higher opinion of the veracity of this statement than if only one opinion was given. Both types of accruals should ensure that the accrued arguments are no longer
considered. In the first case this is because the increase in argument strength may not be linear, while in the second it would lead to a contradiction.

However, there are cases where an accrued argument leads to multiple conclusions, only some of which take part in the accrual. Notably, this occurs when an argument has more than one conclusion. In this case, only the accrued conclusions should be affected by the accrual process. For example, given the arguments

1. If it rains, I must take the washing in, and I do not run.
2. If it is hot, I won’t run.

the accrued conclusion (that I do not run), has no influence on the strength of the non-accrued conclusion (that I take the washing in).

We attempt to represent the first type of accrual within evidence nodes. We make the simplifying assumption that arguments for and against the evidential nodes are independent of each other. It thus makes sense to use the subjective logic consensus operator to combine the warrant’s opinions and assign them to an evidential node.

**Definition 4.6. (Evidential Node Opinions)** Given $n$ opinions $\omega_1, \omega_2, \ldots, \omega_n$ for a grounded type $t$, the evidential node $(t, \omega_t)$ is created by assigning $\omega_t = \bigoplus_{i=1}^{n} \omega_i$.

The second type of accrual, as well as some instances of the first, can be captured using attacks between warrants\(^5\). Before describing how to represent this type of accrual, we need to examine warrants in more detail. In the framework, warrants are instantiations of argument schemes. Given two warrants which are instantiations of the same scheme, with premises having the same opinions, the opinions attached to their conclusions should be identical. Thus, it makes sense to attach those properties affecting the opinions of warrants to their argument schemes. In our model, six parameters affect which evidential nodes are affected by an argument scheme (and thus a warrant):

1. **Argument Strength**: Some arguments are more convincing than others. For example, a deductive argument may always be considered correct by many, while arguments from witnesses may sway them less. It thus makes sense to associate an opinion with an argument scheme and use this opinion to weigh any conclusions obtained from the scheme by making use of the subjective logic discounting ($\otimes$) operator.
2. **Premises Strength**: Clearly, a high opinion regarding the veracity of the premises of an argument will lend strength to the opinions resulting from the argument.
3. **Conclusions**: An argument scheme must state what conclusions may be derived from it.
4. **Undercuts**: Undercuts are another type of conclusion which has not yet been mentioned. A warrant undercuts another warrant by attacking its applicability in a certain situation. Rather than lending an opinion to a conclusion, it affects the opinion assigned to the attacked warrant. Undercuts are used to represent the second (and in some instances, the first) class of accruals mentioned earlier.

\(^5\)Attacks between warrants also allow for the representation of undercutting arguments.
5. The Mapping Function: Within warrants, the opinion associated with a conclusion depends on the specific argument scheme used. For example, presenting evidence from multiple witnesses in a trial results in a final opinion very different to one showing that someone is an expert in a field and makes a certain claim at a very high level of confidence. Thus, it appears that the mapping between premises and conclusions is dependant on the argument scheme used, and each argument scheme has its own mapping. This mapping function is used to compute opinions for conclusions based on the opinions associated with the argument scheme’s premises.

6. Applicability: When advancing an argument, we must be satisfied that it is applicable in the circumstances. To model this, every argument scheme has an associated applicability function. A global threshold function can then be applied to decide whether the argument is applicable or not.

To easily support point 4, we need to be able to refer to other argument schemes. Thus, an argument scheme should also be associated with a unique name.

We can now define an argument scheme as follows:

**Definition 4.7. (Argument Scheme)** An argument scheme $AS$ is a tuple

$$AS = (N, P, C, S, U, M, A)$$

Where

- $N$ is the name of the argument scheme.
- $P$ is an ordered set of abstract types reflecting the argument’s premises.
- $C$ is another ordered set of abstract types distinct from $P$, and represents the argument’s conclusions.
- $S$ is an opinion reflecting the inherent strength of the argument.
- $U$ represents the warrants that may be undercut by this argument. $U$ is a set of triples $(P, C, AS)$ where $P$ and $C$ are abstract types and $AS$ is the name of an argument scheme.
- $M$ is a function mapping opinions from the premises $P$, the opinion $S$, and a warrant’s $U$ members to the conclusions and undercutters of the argument. It is discussed in more detail below.
- $A$ is an acceptability function, mapping opinions from the premises $P$, the opinion $S$, and the opinions from a warrant’s $U$ component to a boolean range.

The strength of an argument, $S$ could have been implicitly included within the $M$ function. The separation of the two is somewhat arbitrary, but conceptually useful; by keeping these two parameters distinct, a user of the framework is able to differentiate between inherent belief in the validity of a certain class of arguments, and the strength the class of arguments lends to its conclusions when applied in a particular case (thus, for example, one could objectively say that
Modus Ponens is “stronger”, i.e. has a higher belief allocated to it, than an argument from expert opinion).

Given this definition of an argument scheme, we may derive a warrant from it by instantiating the scheme’s premises, conclusions and undercutting arguments.

Before examining how a complete set of evidence, warrants and argument schemes interact with each other, however, we must examine the $M$ function in detail as briefly mentioned above. In our view, different argument schemes combine evidence in different ways (c.f. Pollock (1995) where he asserts that statistical syllogism represents a general form of reasoning). Each argument scheme, therefore, may have its own way of mapping opinions onto conclusions (and undercutters) from a set of premises, ranging in complexity. For example, given the premises $\text{holds}(A), \text{implies}(A, B)$, one form of Modus Ponens might simply use the subjective logic consensus operator to compute an opinion for the conclusion $\text{holds}(B)$. Another possibility is a function of the form

$$\omega(\text{holds}(B)) = \langle 0, 0, 1 \rangle \text{ if } \text{bel}(\text{holds}(A)) < 0.5 \text{ or } \text{bel}(\text{implies}(A, B)) < 0.5$$

$$\omega(\text{holds}(B)) = \omega(\text{holds}(A)) \text{ if } \text{bel}(\text{holds}(A)) < \text{bel}(\text{implies}(A, B))$$

$$\omega(\text{holds}(B)) = \omega(\text{implies}(A, B)) \text{ otherwise.}$$

As discussed in Jøsang et al. (2005), if more information becomes available, a more complicated representation using the conditional deduction ($\odot$) subjective logic operator might be applicable. This is a ternary operator, computing an opinion for $y$ given opinions about the status of $x$, as well as opinions about $y|x$ (that is, $y$ being true given that $x$ is true), and $y|\neg x$ (i.e. $y$ being true given that $x$ is false). In fact, many other operators have been defined for Subjective Logic (Jøsang and McAnally, 2004; McAnally and Jøsang, 2004; Jøsang et al., 2003), and it appears that many of these capture different ways of combining evidence, and hence different argument schemes. Clearly, there are many ways to combine evidence, and the $M$ function explicitly enables arbitrary operators to be inserted into an argument scheme. Little more can be said about it, however, except that it is a mapping from the opinions of premises to conclusions and undercutts:

**Definition 6. (Argument Mapping Function)**

**Within an argument scheme $(N, P, C, S, U, M, A)$, the mapping function $M$ is defined as**

$$M : (\omega_{p1} \times \ldots \times \omega_{pi}) \rightarrow (\omega_{c1} \times \ldots \times \omega_{cj} \times \omega_{u1} \times \ldots \times \omega_{uk})$$

where $i = 1 \ldots |P|$, $j = 1 \ldots |C|$ and $k = 1 \ldots |U|$.

We can now describe how the status of an argument may be determined$^6$. Informally, given an argument graph together with opinions for some of the evidential nodes, we propagate those

$^6$One simplifying assumption we make is that the argument graph is acyclic. Nothing in the framework prevents an argument graph from becoming acyclic; it is assumed that this restriction is enforced by careful design of the argument schemes. This may be unrealistic in some situations. For example, evidence nodes may be mutually exclusive, and if so, an attack cycle should form between them. This issue is explored in depth later.
opinions forward through the graph. If an warrant’s opinion falls below a threshold applicability value, it is removed from the graph. If we are unable to calculate a warrant’s value, we pause until all of the warrant’s premises (and input undercutting warrants) have values assigned to them. Since our graph is acyclic, all nodes will eventually have an opinion assigned to them, at which point our algorithm will terminate. This algorithm is formalised in Figure 4.4.

Some shorthand is used in the algorithm. $M_{AS}(w)$ represents evaluating argument scheme $AS$’s $M$ function based on the premises and undercutters of warrant $w$. Similarly, $A_{AS}(P, iw)$ represents the evaluation of a warrant’s acceptability based on its premises and undercutters.

The algorithm operates by creating a list of visited warrant and evidence nodes ($VW$ and $VE$), as well as two sets to store opinions as they are generated ($IE$ and $IW$). Then, while some nodes have yet to be visited (line 1), it begins by checking if any evidence nodes are unaffected by opinions from other warrants. If any are found, they are added to the $VE$ set (lines 2–4).

Lines 5 to 9 selects warrants to be evaluated. A Warrant $w$ is chosen if all its evidence nodes have an associated final opinion, as it is found in $VE$ (line 6), and if all warrants that they undercut have also got a final opinion associated with them (lines 7–9).

The warrant is then evaluated for acceptability (line 10). If it fails this test, it is removed from the argument system (line 11). Otherwise, its conclusions are computed (line 14), and a final argument strength is computed based on the strength of arguments that undercut it (line 15). The conclusions that affect the evidence nodes are stored in the set $IE$ (line 16), and those that affect other warrant nodes are stored in the $IW$ set (line 17). Finally, the warrant is marked as visited (line 18).

At this point, evidence nodes can be updated. An evidence node is only updated if there are no warrant nodes that may still affect it that are still to be evaluated (line 21). The update procedure consists of combining all evidence that affects the node’s opinion (line 22). Finally, the evidence node is marked as visited (line 23).

To operate, the algorithm takes in an argument graph, and a starting set of opinions. It is assumed that these opinions are not under dispute, and the associated nodes thus have no edges leading into them. Any other source edges will be assigned an opinion of $\langle 0, 0, 1 \rangle$ (meaning that its real value is completely uncertain), allowing the assumption of closure over the evidence nodes to be fulfilled. The algorithm then propagates these opinions forward through the graph, until all applicable arguments in the graph have been taken into account. This algorithm, and the approach, supports a number of features common to argument:

- Only applicable arguments are used. If a warrant is deemed inapplicable due to falling below a certain opinion threshold, it is removed from the graph. Since the graph is acyclic, and evaluation of a warrant takes place only once all its premises and undercutting warrants have been considered, we are guaranteed that reinstatement cannot occur. Thus, once a warrant is removed, it will never be able to be reinserted into the argument.

- Defaults must be taken into account. By carefully designing the $M$ function, it is possible to ensure that a premise leads to a warrant being used as long as the premise isn’t explicitly false (this is achieved by having the $M$ function not consider the premise if its disbelief component rises above 0.5).
computeConclusions(EN, WN, AS)

Where EN is a set of evidential nodes
WN is a set of warrant nodes
AS is a set of argument schemes

Assumptions:
The set of evidence nodes is closed w.r.t warrants, i.e. ∀w ∈ WN of the form (P, C, AS), P, C ∈ EN.

Define the variables: visited warrant nodes VW = {}
visited evidence nodes VE = {}
A set of evidence node opinion pairs IE = {}
A set of warrant node opinion pairs IW = {}

01 repeat until EN = VE and WN = VW {
02 ∀e ∈ EN such that ∃w = (P, C, AS) ∈ WN where e ∈ C {
03 \quad VE = VE \cup \{e\}
04 }
05 ∀w ∈ WN\VW where
06 \quad w is of the form (P, C, AS), for any p ∈ P, p ∈ VE
07 \quad AS is of the form (N_{AS}, P_{AS}, C_{AS}, S_{AS}, U_{AS}, M_{AS}, A_{AS}). If
08 \quad ∀uc ∈ AS such that uc = (N, P, C, S, U, M, A) and
09 \quad ∃u ∈ U such that u can be unified to w, (P_x, C_x, uc) ∈ VW for any P_x ∈ VE {
10 \quad \text{if } A_{AS}(P, iw) = false where iw = (w, \omega_w) ∈ IW then {
11 \quad \text{WN}\{w\}
12 \quad }
13 \quad else {
14 \quad \text{Compute } M_{AS}(w) = (\omega_{e1}^w \ldots \omega_{en}^w, \omega_{w1}^w \ldots \omega_{wn}^w)
15 \quad \text{Define } Str = S_{AS} \bigoplus \omega_w \text{ for all } (w, \omega_w) ∈ IW
16 \quad IE = IE \cup \{(c_i, \omega_{ci}^w \otimes Str)\} \text{ for all } i = 1 \ldots m
17 \quad IW = IW \cup \{(u_i, \omega_{ui}^w \otimes Str)\} \text{ for all } i = 1 \ldots n
18 \quad \text{VE = VE \cup \{}w\}
19 \quad }
20 \text{∀e = (t, \omega_t) ∈ EN such that ∃w = (P, C, AS) ∈ WN\VW where e ∈ C { }
21 \quad ∀ie = (e, \omega_{ie}) ∈ IE, \omega_i = \bigoplus \omega_{ie}
22 \quad \text{VE = VE \cup \{}e\}
23 }
24 }
25 }

Figure 4.4: An algorithm to compute conclusions given a set of argument schemes, instantiated arguments, and, optionally, some opinions.
Accrual of arguments. As was mentioned above, the application of the consensus \( (\oplus) \) operator when combining warrants into evidence allows us to cater for one class of accruals. As discussed later, the \( (\oplus) \) operator may not be applicable in all cases of accrual, leaving this approach open to some criticism. By having an accruing warrant undercut all its accrued warrants, the second class of accrual of arguments can be represented. In such a case, defeat of the accruing warrant will reinstate the accrued warrants, agreeing with our intuition. Problems remain with this type of accrual; the \( M \) and \( U \) functions need to be carefully crafted to support it, and accruals of an arbitrary number of arguments cannot be represented.

The algorithm is guaranteed to terminate, and all evidence nodes will have an opinion assigned to them. Some warrant nodes will have been removed from the graph as they will have been deemed unacceptable. The algorithm runs in \( O(n) \) time, where \( n \) is the number of edges in the graph.

At this stage, we may be faced with a number of mutually exclusive evidence nodes with similar opinion assignments. One way of resolving this difficulty involves translating the graph into an abstract argument framework, and computing an extension appropriate to the type of reasoning we wish to perform. The resultant graph, unlike the original, may include cycles, and this translation process is thus one way of introducing information that could not have been represented in the original graph.

This approach loosely follows that advocated by a number of argumentation researchers (Amgoud, 2007; Cayrol and Lagasque-Schiex, 2005a), namely assigning valuations to nodes and then computing extensions from nodes after these valuations are assigned. However, without either changing the notion of acceptability, or using an argument framework with explicit support for valuations, this approach will not work. Consider, for example, the case where argument \( x \) is very strongly supported by argument \( a \), and weakly attacked by argument \( b \). In such a case, arguments \( a, b \) will remain in any computed extensions, while our intuitions would want all three arguments to be present.

To overcome this problem, we assume that any arguments remaining at the end of the evaluation procedure, before translation takes place, support any warrants to which they are linked, and that all warrants support their conclusions. In such a situation, the translation process consists of adding support nodes between the empty argument and any source evidence nodes, and the addition of attack nodes between mutually exclusive arguments will lead to e-preferred and e-grounded extensions that support our intuition. Consider, for example, the set of arguments depicted in Figure 4.5. Given that arguments \( a \) and \( b \) are mutually exclusive, we will end up with two e-preferred extensions, namely \( \{a, x, c\} \) and \( \{b\} \). Thus, arguments that “depend” on a certain piece of evidence will not propagate into extensions where this argument is not present. However, this translation process fails where accruals allow a warrant to be established. To perform any better, an abstract argument framework must allow for valuations and be able to model the accrual process accurately.

All that remains is to define the system in which argument takes place. This argument system contains evidence nodes and their associated opinions, together with warrants (and the argument schemes they have been instantiated from). Finally, an argument system contains sets of mutually
exclusive arguments, allowing us to perform the translation step:

**Definition 4.9. (Argument System)** An argument system $\eta$ is a tuple $(E, W, AS, ME)$ where $E$ is a set of evidential nodes, $W$ a set of warrant nodes and $AS$ a set of argument schemes such that for any $(P, C, A) \in W$, $P, C \in W$, $A \in AS$. Similarly, for any $(N, P, C, S, U, M, A) \in AS$, the types represented by $P, C$ and $U$ must be unifiable to those grounded types represented by $E$. $ME$ contains sets of mutually exclusive grounded types\(^7\), i.e. $ME \in 2^Σ$.

An example of the framework in operation is presented in Chapter 5.

At this stage, we are able to statically analyse an argument. That is, given a set of arguments and opinions, we are able to determine which set of arguments represent the most likely environment state, and their likelihood. Given a set of likely, mutually exclusive arguments, we may use the abstract framework to determine what set of arguments is mutually consistent. Domain-specific requirements must then be used to decide whether sceptical or credulous reasoning, together with their associated extensions should be used in generating this admissible set.

We now examine argument in a dynamic sense, that is, examining how agents should go about adding arguments to the argument set, within the context of a dialogue game. This takes place within the procedural layer.

### 4.4 The Procedural Layer

The logical and dialectic layers allow us to compute which sets of arguments represent the most likely environment state (and the arguments’ likelihood) when presented with a set of arguments and opinions. This process is, in a sense, static, as no regard is given as to how these arguments and opinions are generated. We begin to address this issue within the procedural layer.

Within the procedural layer, we introduce rules for a simple dialogue game. These rules describe a valid protocol through which an argumentation-based dialogue may take place. Dialogue games are normally described in terms of commencement, locution, combination, commitment and termination rules (as discussed in Section 2.1.6). Before describing these rules in detail, we provide an informal description of the dialogue game.

\(^7\)It is easily possible to have $ME$ contain abstract types and automatically convert these to grounded types.
The commencement rules for our dialogue are not strictly defined; the game begins whenever 
an agent would like to convince other agents that the environment is in a certain state. Agents 
wanting to make use of the dialogue must have a shared vocabulary of argument schemes. Each 
agent also has an internal knowledge base which keeps track of ways of determining the state 
of a portion of the environment. The action of state determination is hereafter referred to as a 
sensor probe. Sensor probes are needed because within the dialogue, an agent is never aware of 
the real environment state. Instead, all it can do is probe the environment in some way and obtain, 
through a proxy, an opinion regarding environment state. A dialogue begins by having any of 
its participants advancing an utterance. This utterance consists of putting forth any combination 
of warrants and sensor probes. The participants then take turns making utterances. Eventually 
(assuming a finite knowledge base for each participant), no new utterances can be made, and at 
this stage, the dialogue terminates. The algorithm presented in the previous section can then be 
used to determine the most likely state(s) for the environment.

It is assumed that an agent participating in a dialogue is attempting to show that the environ-
ment is in a certain state. It is also assumed that by showing that the environment is in this state, 
the agent gains some utility. Within the context of the dialogue, a participant incurs a utility cost 
when it probes a sensor. This reflects the situation where a probe requires the expenditure of some 
(scarce) resource.

While an agent may make use of the dialogue to perform internal deliberation regarding 
environment state, we will, from now on, refer to each party participating in the dialogue as an 
agent. Utterances are stored in a “commitment store”, and, once made, cannot be retracted. A 
dialogue thus takes place within the environment, defined as follows:

**Definition 4.10. (Environment)** An environment $Env$ is a tuple $(Agents, CS, AS, S, PC)$ where 
Agents is the set of agents operating within the environment and CS is an argument system 
(as per definition 4.9) containing the graph of arguments already introduced via utterances. AS 
contains all argument schemes that may be used in the course of a dialogue. S consists of the set 
of all sensors present in the environment. $PC : 2^S \rightarrow \mathbb{R}$ is a function representing the cost of 
probing a set of sensors.

**Definition 4.11. (Agent)** An Agent $\in Agents$ is a tuple $(Name, KB, G, C)$ where Name uniquely identifies the agent. KB stores a set of evidential nodes which represent the agent’s beliefs about the state of the environment. $G : 2^{GTtypes} \rightarrow \mathbb{R}$ represents the agent’s goal function, and maps “proven” grounded types to utility values. The variable $C \in \mathbb{R}$ keeps track of the utility cost incurred by an agent in the current conversation.

**Definition 4.12. (Sensors)** A sensor $S = (\Omega_p, probed_p)$ is a tuple containing an ordered set $\Omega_p$ of 
grounded evidential nodes of the form sensor($X$) where $X$ is another grounded evidential node. 
This set represents the sensor’s opinions regarding the state of a portion of the environment. A 
sensor contains a second ordered set probed_p of boolean values, capturing whether the sensor 
has already been probed for a specific opinion.

**Definition 4.13. (Sensor Argument Scheme)** AS includes the argument scheme 

$\{SensorScheme, \{sensor(X)\}, X, (1, 0, 0), \}, \omega(sensor(X)), true\}$
This scheme allows us to represent a probing action as the introduction of a sensor probing warrant. The results of a sensor probe can be computed using the $\oplus$ operator on the relevant node $X$ without any additional scaffolding required in the framework (though it should be noted that this is the only way, except via additional argument schemes, to introduce a new opinion into an argument framework).

Agents take turns to advance a line of arguments (consisting of sets of warrants), and, if necessary, probe sensors to obtain more information about the environment. This combination of actions is called an utterance. During each turn, evidence nodes within the environment are updated based on sensor probes, and the commitment store is updated with the appropriate warrants. Any sensors probed are marked as such (since a sensor will always return the same opinion, there is no point in probing it more than once), and costs to the agent are updated. In this dialogue game, once an utterance is made, it may not be withdrawn from the commitment store.

**Definition 4.14. (Utterances)** The utterance function

\[
\text{utterance} : \text{Environment} \times \text{Agent} \rightarrow 2^{\text{Warrants}} \times \text{Probes}
\]

takes in an environment and an agent and returns the utterance made by the agent. The first part of the utterance lists the warrants advanced by the agent, while the second lists the probes the agent undertakes. Here, Probes $\in (2^{\text{GTYPES}} \times 2^S)$. In other words, a probe specifies which grounded types should be probed, as well as through which sensors.

**Definition 4.15. (Turns)** The turn function

\[
\text{turn} : \text{Environment} \times \text{Agent} \rightarrow \text{Environment}
\]

Takes the current environment as well as the agent whose turn it is to make an utterance, and returns a new environment containing the effects of the agent’s utterance.

Only an utterance can change what the environment looks like between turns. Thus, to define what an environment looks like at the end of a turn, i.e. after an utterance $(\text{Warrants}, \text{Probes})$, we must describe how each element of the 5-tuple is changed. Clearly, $AS$ and $PC$ will remain unchanged.

The set of sensors, $S$, will have to be updated based on the contents of $\text{Probes}$. If sensor $s$ is in probes for a grounded type $g$, then the $\text{probed}_g$ entry indexed by the appropriate $\Omega_p$ must be set to true. In other words, the set of sensors is updated so that any probed grounded types are marked as such, so that they are not be probed again (as our framework assumes that a sensor will always return the same opinion in response to a probe for a specific evidential node).

The content of utterances determines what $CS$ looks like. Probed grounded types, whose opinions are exposed for the first time, must either be added to the argument system, or must replace default opinions. New warrants must also be added to the argument system. Finally, grounded types which were already within the argument system must have new values assigned to them. This is achieved by making use of the $\oplus$ operator, operating over all sensors and arguments affecting this node.
The way in which agents are updated is dependant on the exact implementation of the agent. Some agents may be able to update their knowledge bases based on the opinions obtained from sensor probes, while others may be dogmatic in their beliefs. One feature common among all implementations is that the utility cost for the agent making an utterance must be updated. Formally, we can define this procedure as follows:

**Definition 4.16. (Turn Updates)** Given the environment at the start of the turn,

\[ Env_s = (Agents_s, CS_s, ArgS, S_s, PC) \]

and an utterance by Agent, of the form

\[ (\{w_1, \ldots, w_m\}, \{p_1, \ldots, p_n\}) \]

The environment at the end of a turn is \( Env_n = (Agents_n, CS_n, ArgS, S_n, PC) \) where

- While different agents may update differently, at minimum, given

\[ Agents_s = (Name_s, KB_s, G_s, C_s) \]

the new agent will be defined as

\[ Agents_n = (Name_s, KB_s, G_s, C_s + PC(Probes)) \]

The Agents set will be updated according to the rule

\[ Agents_n = Agents_s \setminus \{Agents_s\} \cup \{Agent_n\} \]

- Given \( CS_s = (E_s, W_s, AS_s, ME) \), \( CS_n = (E_n, W_n, AS_n, ME) \) such that \( E_n = (t \bigoplus \omega_i) \) where \( \exists s \) such that \((t, s) \in Probes \) and \( \omega_i \) is the opinion assigned to the evidential node in \( E_s, W_n = W_s \cup \{w_1, \ldots, w_m\}, AS_n = AS_s \cup \{as|\{p, c, as\} \in \{w_1, \ldots, w_m\}\} \)

- \( S_n = S_s \setminus s \) where \( \exists x \) such that \((x, s) \in Probes \cup s_n. \) Here, \( s_n = s \setminus \{(x, false) \cup (x, true)\} \)

For an ordered set \( Agents = \{Agent_0, \ldots, Agent_{k-1}\} \) containing \( k \) agents, we may then define the dialogue game as follows:

**Definition 4.17. (Dialogue Game)** Turns in the dialogue game can be computed according to

\[ turn_0 = turn((Agents, CS_0, ArgS, S, PC), Agent_0) \]

\[ turn_{i+1} = turn(turn_i, Agent_i \mod k) \]

The dialogue game ends when

\[ turn_i = turn_{i-1} = \ldots = turn_{i-k+1} = turn_{i-k} \]
9303 The Heuristic Layer

Often, $CS_0$ consists of the empty argument graph, and is only populated as agents argue. It may however contain knowledge agreed upon by all dialogue participants. It should also be noted that an agent may make a null utterance $\{,\}$ during its turn to (eventually) bring the game to an end. Given a finite number of possible arguments and sensors, it is clear that the dialogue is guaranteed to terminate, as eventually no utterances will be possible that modify the public knowledge base $CS$.

At any time, we may compute an agent’s utility by combining its utility gain (for achieving its goals) with its current costs. Running the reasoning algorithm allows for the computation of a set of literals (which might be dependant on whether credulous or sceptical reasoning is being used), and their associated opinions. Then it is possible, using an admissibility function $Admissible(\omega) \rightarrow [true, false, unknown]$, to group these literals into one of three sets, $f_{true}, f_{false}$ and $f_{unknown}$. From this, it is possible to compute an agent’s net utility gain based on the agent’s goal function $G$, and sensor probe cost amount $C$:

$$agentUtility = G(f_{true}, f_{false}, f_{unknown}) - C$$

Other, more complicated utility functions, based on the strength of an opinion are also possible.

To summarise, our dialogue game is defined by the following dialogue rules:

- **Commencement rules**: Undefined, an agent may begin the dialogue at any stage.
- **Locutions**: Agents may make either make an utterance consisting of a set of arguments and sensor probes, or pass.
- **Combination rules**: Agents may make an utterance or pass at any stage (within their turn).
- **Commitment rules**: As defined previously, utterances are inserted into the public knowledge base. Items may not be removed from the knowledge base. Commitment rules have the effects specified in Definitions 4.16 and 4.17.
- **Termination rules**: The dialogue ends when $n$ passes are uttered in succession, where $n$ is the number of dialogue participants.

At this point, agents may use the dialogue game to participate in a dialogue. However, we have not yet described how an agent may choose what utterance to make. This is done within the heuristic layer.

### 4.5 The Heuristic Layer

In the previous section, we introduced a dialogue game allowing agents to argue about the state of the environment. At any stage of the dialogue, an agent must choose one of many possible utterances to make. In this section, we propose one possible heuristic that will allow an agent to make such a choice.

Informally, the heuristic is built around the idea of performing a one level deep breadth first search of possible warrants to advance and sensors to probe, and selects the combination of these that allows it to maximise its utility if the game were to end after its utterance. This heuristic is
formalised next, after which we suggest a number of other heuristics, and examine their, and this heuristic’s limitations.

To make an utterance, the agent must select

- Which sensors to probe.
- Which argument schemes to instantiate into warrants.

An agent may probe any unprobed sensor. That is, the set of possible probes, $PP$, can be defined as follows:

**Definition 4.18. (Possible Probe Set)** Given the environment’s set $S$ of sensors where, each sensor $s \in S$ is of the form $(\{\omega_{s1}, \ldots, \omega_{sn}\}, \{\text{probed}_{s1}, \ldots, \text{probed}_{sn}\})$ and each $\omega$ in turn refers to literal $l$, the set of possible probes an agent may undertake, $PP$, may be defined as

$$PP = \bigcup_{s \in S} \{\omega \mid \text{probed}_s = \text{false}\}$$

By combining the literals from $PP$ with those that appear within evidential nodes in $CS$, the agent may compute which warrants it may advance, namely those which it can instantiate based on the argument schemes, which in turn, are based on the warrants appearing in the combined set. We may define an argument scheme $as$ as instantiatable for an argument system $\eta = (E, W, AS, ME)$ iff a warrant $w$ can be created of the form $(P, C, as)$ such that $P \in E$ and $w \notin W$. Given the set of all instantiable argument schemes for an argument system, we may compute the set of possible warrants $PW(\eta)$ by instantiating those schemes.

Thus, for each element $pp \in PP$, the set of possible arguments that may be advanced is $2^{PW((E \cup pp, W, AS, ME))}$. By simulating a sensor probe (i.e. by assigning it the opinion the agent has for the state of the literal) and then using the algorithm described in Figure 4.4, the agent may compute the outcome of the dialogue if it were to terminate after making the utterance. It could then calculate the utility it would gain from such a situation, and assign it to the possible utterance. The agent would then advance the utterance that maximises its utility.

**Definition 4.19. (The Heuristic)** Given the set of possible probes $PP$, and for each $pp \in PP$, a set of possible arguments $PA = 2^{PW((E \cup pp, W, AS, ME))}$, the agent computes a utility $u(pa, pp)$ for each $pa \in PA$ using the algorithm described in Figure 4.4. The agent then advances the utterance $(pa, pp)$ that maximises its utility.

The presence of powersets within this computation gives the heuristic exponential time complexity. If we were willing to give up the guarantee of finding the best utterance, modifications of this heuristic can be used. For example, by generating utterances by increasing length, we are likely to find good utterances relatively quickly (following the intuition that additional sensor probes cost additional utility).

We have investigated another type of heuristic, namely one in which an agent chooses an utterance that reveals a minimal amount of information, in (Oren et al., 2006b). The heuristic

---

8Some warrants may be pruned from the possible argument set as the algorithm is being run as it may be seen that the evidence for them is not sufficiently strong. The $PW$ set is thus, in a way, optimistic, containing many warrants which may be impossible to use.
4.5. The Heuristic Layer

described in this thesis is an evolved version of that heuristic, but is not identical. In the former heuristic, an agent attempted to minimise the number of new evidence nodes introduced while making an utterance. However, in the heuristic described here, the agent attempts to minimise the cost incurred due to sensor probes. By associating only one sensor with an evidence node, and having all sensor probes cost an equal amount, together with allowing only dogmatic opinions (and using a carefully designed $M$ function for all argument schemes) we can represent the minimal information heuristic in this framework:

**Definition 4.20. (Minimal Information Heuristic)** Given a system where

- For every evidence node, only a single sensor exists which may probe its state.
- Equal costs for every sensor.
- A large utility value (greater than the number of sensors in the system), which an agent is allocated for every state where its goals are neither proved nor disproved (we refer to such states as "draw" states).
- A utility reward greater than the utility gained in the draw states for any state where an agent’s goals are proved.

An agent will, by using the heuristic described in Definition 4.19, and attempting to maximise its utility, probe as few sensors as possible, and thus introduce as little information as possible into the dialogue. We call this strategy the minimal information heuristic.

Other, simpler forms of the breadth first search heuristic were also described in (Oren et al., 2006a,c). Within these papers, it was pointed out that without opponent modelling, it is difficult to perform a search greater than one level deep, since we do not know how an opponent may respond to an utterance. The emphasis in those papers, however, is on an interesting aspect of this type of heuristic: an agent may choose not to achieve all its goals (i.e. maximise gross utility) as the cost of sensor probes would mean that its (net) utility is not maximised.

Further discussion of heuristics, and possible extensions to the argument strategy presented here are discussed in Section 6.1.2.

4.5.1 Learning from Evidence

Assume that an agent is participating in a dialogue and holds a belief that a certain literal holds with an opinion of $\langle 0.9, 0, 0.1 \rangle$. However, as the dialogue progresses, repeated sensor probes of that literal show that it is in fact not in that state. Clearly, the agent should update its beliefs regarding that literal. If this were not the case, the agent would dogmatically attempt to prove its point in the face of mounting evidence against its position.

We propose another simple heuristic to do this, namely to have an agent compute the opinion of the literal based on all available sensor evidence (through the use of the $\oplus$ operator), and then weigh its belief by a further application of the operator. In other words, for a literal $l$,

$$\omega_{agent}(l) = \omega_{env}(l) \oplus \omega_{agent}(l)$$

It can easily be shown that this heuristic satisfies Moore’s criteria (Moore, 1993).
where $\omega_{env}(l)$ is the current opinion for the literal within the environment’s $CS$.

This belief update heuristic is completely ad-hoc. Other, more complicated belief update techniques are possible, with approaches from various areas of statistics and machine learning probably being most useful.

4.6 Discussion

Chesñevar et al. (2000); Prakken and Vreeswijk (2002) provide extensive surveys of approaches to argument, and the work presented in this chapter can be seen as another argument framework in the vein of Pollock (1995); Verheij (2003a); Prakken and Sartor (1996). However, most approaches to argumentation focus on only a subset of the layers described by Prakken and Sartor (2002), and a direct comparison between these, and the work carried out here is, therefore, difficult to perform. We will, however, examine aspects of some systems and differentiate them from what we have achieved here. Since many argument frameworks have been proposed, we will not exhaustively examine them, referring only to those containing features relevant to our research.

4.6.1 The Logical and Dialectic Layers

Pollock’s 1995 framework, instrumental in his OSCAR project, provided a large amount of inspiration for our work. In some ways, his framework is more expressive than the one described in this thesis, as we only deal with what he calls “linear” arguments, that is, reasoning from a finite number of premises to reach certain conclusions. Our framework is unable to perform suppositional reasoning, proof by contradiction and the like. Also, unlike his approach, we are unable to reify our meta-language, meaning that we are unable to reason about argument schemes within our framework. Pollock also provided support for loops, and his work has been used as a foundation for a large body of argumentation research.

Accrual of Arguments

While powerful, Pollock’s approach has difficulties in representing accruals (Prakken, 2005b). While our framework cannot support all forms of accrual, the use of the $\oplus$ operator when combining evidence, together with the ability to use an arbitrary mapping function within an argument scheme, and the ability for an accrued argument to undercut its component arguments, means that we are able to provide a large amount of flexibility when representing accruals.

Frameworks such as Verheij’s (1996) reason based logic also support a model for accrual. In his framework, accruals are represented by allowing sets of arguments to outweigh other arguments. We believe that assigning a specific strength to an argument, as we do in our framework, is more useful, as it is (eventually) possible to compute a probabilistic expectation value for a conclusion, as well as to quantitatively compare the strength of arguments.

Prakken (2005b) identifies two approaches to representing accruals. The first, which he refers to as the knowledge representation approach, involves encoding the accrual within the system. He gives the example of a probabilistic network, where a conclusion $r$, depends on arguments $q$ and $p$. Then the accrual is explicitly encoded by having values in the system for $P(r|q \land \neg p)$, $P(r|p \land \neg q)$, $P(r|p \land q)$ and $P(r|\neg p \land \neg q)$. When not using probabilities, he claims that specifying priorities between the rules is enough to represent accruals. However, the knowledge representation approach requires a large number of rules to be formulated. Furthermore, this
approach needs the ability to represent all accrued arguments in the same way (e.g. as a probability). Finally, he claims that the knowledge representation approach does not represent how humans think about accruals.

Instead, Prakken proposes what he terms the inference approach to accruals. Here, accruals are computed as a step of the inference process. While this approach is elegant, it is unable to easily deal with argument strengths, limiting its application in many situations.

Our handling of accruals shares many of the features of the knowledge representation approach. Priorities between argument schemes participating within accruals are represented by undercuts between arguments (in earlier work (Oren et al., 2007b), we suggested using the idea of one argument subsuming another to handle accruals, but this was only able to handle certain, very specific instances of accrual). The effects of an accrual are represented using the mapping function, and may be customised on a per-accrual basis. All arguments are based on the same Subjective Logic underpinnings, allowing for accruals to take place between them. This approach was chosen as it appears as if there is no general way of combining accruals to compute their final strength, but that each type of accrual must be dealt with on a case-by-case basis.

**Argument Schemes**

Pollock’s “reasons” are similar to argument schemes in our framework. Reasons allow domain-specific inference rules to be added to the argument system. In Pollock’s framework however, the strength obtained from a reason is always as strong as the weakest argument contributing to the reason (or, if the reason is weaker, the reason’s strength). In our framework, the strength of a warrant, i.e. an applied argument scheme, is dependent on the argument scheme itself. Handling argument schemes in this manner allows us to represent argument schemes that do not fit the “weakest link” mold. The most obvious example of such a scheme is accrual of arguments. Schemes based on trust of a witness are another example of a non-weakest link type scheme. Notably, witness testimony is based on the SL discounting operator, and, as suggested by Jøsang’s work, many other combination functions exist, each applicable to a different set of situations. We, therefore, believe it is important to provide support for these by means of a user-defined mapping function within an argument scheme.

While the importance of argument schemes has long been recognised, few frameworks have provided explicit support for them. One possible reason for this could be argumentation researchers’ focus on abstract argument frameworks; argument schemes are more easily represented in concrete frameworks containing inference rules. Much of the existing work on argument schemes has focused on a scheme’s properties (Walton, 1996), on how to argue in a certain way based on a scheme (Atkinson and Bench-Capon, 2006), or on the formalisation and analysis of certain aspects of argument schemes (Bex et al., 2003).

Some argument frameworks (apart from OSCAR) have emerged that have include support for argument schemes. For instance, Verheij (2003b), shows how argument schemes may be represented using his DEFLOG language. As with most other formal approaches, the notion of argument strength is not considered (except in the sense that, when instantiated with Rule Based Logic, the *outweighs* predicate exists). Critical questions are handled as exceptions to an argument, preventing it from being justified.

Before examining how critical questions are handled in our framework, we must examine
how burden of proof may be dealt with.

**Burden of Proof**

As mentioned in Gordon *et al.* (2007), the application of critical questions to an argument scheme is intimately tied into the notion of burden of proof. Two aspects of burden of proof exist: upon which arguing party the requirement for proof falls, and the level of certainty required to support an argument for the burden of proof to shift to the other party.

With a single weight, the level of certainty required for an argument is simple to represent. For example, in OSCAR a typical level of certainty is that an argument be certain above a probability of 0.5. Extending this to different proof standards (i.e. beyond reasonable doubt, on the balance of probabilities, and the like), is relatively simple, and can be done by associating these different levels of proof with different levels of probability. Given that Subjective Logic deals with opinion triples, representing the required level of certainty is more difficult to do. One approach would be to convert an opinion to an expectation value, and then, as usual, require that this expectation exceed a certain threshold value. The lack of a one-to-one mapping between opinions and expectation values may, however, be viewed as a problem if this approach is taken. Another approach would require that the belief component of the triple exceed a certain value, or that either the disbelief, or uncertainty components, remain below a certain level. In our examples, we have tended towards requiring the belief component to remain above a threshold.

To make use of an argument, it must be admissible. By altering the admissibility function, different burden of proof requirements may be set. These requirements may be based on the premises of the argument, or on arguments existing within the system (in the case of arguments undercutting the introduced argument). To set the burden of proof on the agent introducing an argument, the admissibility function must be set so that by default, the argument is not admissible. If, alternatively, the admissibility function is set so that, by default, the argument is admissible (for example, by setting the argument to admissible as long as disbelief is not greater than 0.5), then it is up to the agent who does not agree with the argument to defeat it.

The admissibility and mapping functions thus provide simple, but powerful techniques for both setting proof levels, and assigning burden of proof to one of the arguing parties.

**Handling Critical Questions**

Following Gordon *et al.* (2007), critical questions in our framework may be viewed as a combination of premises for an argument and argument schemes/warrants that may undercut the argument scheme to which the critical questions apply. For example, given the argument scheme for Expert Opinion defined on page 55, we may represent its critical questions as follows:

- **How credible is the expert source?** The credibility of the expert is assumed by default. This critical question should be represented using some credibility argument scheme that may undercut the Argument from Expert Opinion if it is admissible.

- **How much of an expert is E in domain D?** The source’s level of expertise is assumed to hold by default, and is one of the premises (expert(E, D)) to the argument scheme. Thus, the admissibility function applies the condition disbelief(expert(E, D)) < 0.5 to represent this question.
4.6. Discussion

- **What did E assert to imply A?** As presented, we cannot answer this critical question, as claims\((E, A)\) is a primitive in our scheme. However, it is possible to probe a sensor, or somehow alter the opinion of claim\((E, A)\) to represent this critical question.

- **Could E be biased?** Again, bias would require the introduction of another argument scheme that would undercut this scheme.

- **Is A consistent with other expert’s testimony?** This critical question can be represented as an accrual of the experts’ opinions. If we assume that each expert’s testimony results in the assignment of a value to a single evidence node, the \(\oplus\) operator will accrue their statements. In more complex situations, we would need an intermediate argument scheme to combine testimonies. This scheme may range from something as simple as a scheme of the form says\((E, A)\), A (where only the premises and conclusions are shown), in which case the \(\oplus\) accrual will take place within the evidence node for A, to a more complex argument scheme which may perform the accrual within its mapping function based on factors such as the experts level of expertise.

- **Is E’s assertion based on evidence?** To represent this, we would need to further decompose the claim predicate, showing how the expert inferred it. Both undercutting and rebutting attacks are possible to represent this critical question.

**The Use of Subjective Logic as an Underlying Logic**

One of our framework’s core features is its use of a numerical representation of argument strength. Whether this is an appropriate representation depends on whether Subjective Logic (and, at its base, probability theory), is an appropriate technique for performing reasoning. Experiments have shown that humans often reason in a way that is counterintuitive to the way a rational probabilistic agent would reason, and this has been used as an argument against probability theory as a basis for reasoning. It may however similarly be used as an argument against the rationality of humans. However, since it important to be able to model human reasoning (whether it is rational or not), we must address this problem, and we do so through the use of an arbitrarily complicated mapping function within our argument schemes. By using the appropriate mapping function, it should be possible to obtain opinions that “standard” approaches would claim are not reasonable.

We skirt around the philosophical aspects of this discussion (referring the interested reader to books such as (Halpern, 2003)), and instead contrast our approach with a number of other argumentation approaches that make use of a similar underlying formalism. Since we have already discussed Pollock’s work in some depth, we will not examine it further in this section.

Much of argumentation is concerned with default, defeasible and presumptive reasoning. By using Subjective Logic we are able to perform this type of reasoning within our framework. Defaults, as well as presumptive reasons can be given a high uncertainty value. Argument schemes that can make use of these types of arguments can then have their mapping and admissibility functions accept evidence nodes with a high uncertainty. Such argument schemes would only reject default premises if their disbelief value is too high. When these schemes propagate default, or presumptive, conclusions through the argument graph, their belief (and disbelief) values can change, but their uncertainty value should remain high to indicate that their premises have not yet
been shown to hold. Note that this modelling of defaults and presumptive reasoning cannot easily be done using probability theory, as a low probability for $a$ simply reflects a high probability for $\neg a$.

The introduction of evidence via additional arguments and sensor probes, together with the ability of argument schemes to undercut existing warrants allows for defeasible reasoning to take place.

Vreeswijk (2004) showed how the notion of an argument, as well as defeat between arguments could be represented within Bayesian Networks. He then proceeds to map Dung’s semantics onto Bayesian Networks, allowing one to determine which nodes are admissible. While of considerable theoretical interest, only the lowest layers of an argument system were described. Thus, without further work (or embedding within another system), it is impossible to compare this approach to ours. It should be noted that both Bayesian networks, and our approach to argument assumes a feed forward network through which probabilities (or opinions) are propagated. The calculus used in our framework is, however, based on whatever argument scheme is instantiated within the warrant, allowing for more flexibility than the purely Bayesian approach.

Chesnèvvar et al. (2005) and others have attempted to use possibilistic defeasible logic programming, a language for logic programming containing fuzzy knowledge and possibilistic reasoning, as an embedded logic for argument. Again, the work described by Chesnèvvar and others focuses on only small portions of the argumentation problem, and without being embedded in a larger system, a comparison is impossible.

Having examined some of the properties of the lowest two layers of our framework, we now discuss some of the features of the procedural and heuristic layers.

### 4.6.2 The Procedural and Heuristic Layers

#### Dialogue Games

As mentioned earlier, the dialogue game agents use to argue is very simple, containing only two types of utterances. It has been suggested that issues such as the allocation of burden of proof take place at the dialogue level (Prakken, 2001b). Reasons for this include the ability to argue about reasons for the assignment of burden of proof, as well as allowing for the close interlink between a burden of proof assignment and a commitment upon one of the arguing parties. (Prakken et al., 2004) links burden of proof at the dialogue level to the way burden of proof is assigned due to an argument scheme’s critical questions. To support such a feature, our language would need the addition of reflection, that is, the ability of the language to talk about itself.

Another feature of more complicated dialogue games is the differentiation between levels of commitment, together with the ability to withdraw and change commitments. In our simple model, once an utterance is made, it is inserted into the commitment store, and, while both agents are allowed to modify the utterance’s component strengths, they are both bound by whatever the resulting opinion is. It could be argued that this is actually a feature of argument with evidence; until counter-evidence is produced, all arguing parties accept the evidence (and the original evidence remains viewable by everyone, and may affect later utterances). However, it would be useful to separate out the agent’s own beliefs from the state of the environment in a more elegant way than the current private knowledge base/public commitment store partition. Different levels of commitment (for example, an agent advancing, or making use of a predicate, is more committed
to it than an agent simply not attacking the predicate immediately), may also lead to complicated
the heuristics that agents may use in the course of a dialogue.

Sensors
One novel feature of our framework is its treatment of evidence via sensor probes. The use of
evidence allows for agents to engage in argument about environment state within partially ob-
servable domains. Currently, only one type of sensor exists, which may be probed (at a possible
utility cost), and returns an opinion about a subset of the environment. Issues such as the amount
of trust placed in a sensor, together with its exact nature are abstracted into the opinion returned
by the sensor. By using a more complicated dialogue game, and supplying argument schemes for
reasoning about sensors, it may be possible to unpack this process and provide more support for
different sensor types (such as obtaining sensor data over a period of time), as well as allowing for
more fine-grained control over the sensor probing action.

Our treatment of sensors is somewhat primitive. Pollock (1998b) has suggested a large num-
ber of what are, in our framework, argument schemes for dealing with reasoning and perception.
Integrating these into our framework would be interesting.

Our framework allows us to argue about the data obtained from a sensor. It does not however
allow us to question whether the sensor did in fact sense the data that it claims to have sensed.
That is, while we may obtain further evidence for the true value of a sensed item (possibly through
argument, or the probing of additional sensors, or, as suggested above, through a more complicated
dialogue game), the one atomic fact we are unable to disagree with is that the sensor returned a
specific opinion for a specific sensor probe. Philosophically speaking, exchanges of argument
using around our framework must accept representative realism (Brown, 1992).

Heuristics for Argument
The argument heuristic we proposed corresponds to a one step lookahead by the agent. During
its turn, it computes all possible utterances (and sensor probes) it can make, and, based on its
beliefs about the environment, determines what the outcome of each utterance would be. It then
makes the utterance that would maximise its utility. Without opponent modelling, very little can
be done to improve this heuristic while maintaining its optimality. However, the computational
cost of the heuristic makes it impractical for computing utterances within dialogues where many
possible arguments exist. Additional heuristics (such as the application of the Gricean maxim
(Grice, 1975) regarding conciseness), may be used to constrain and guide the search. Applying
arbitrary cut-offs, such as stating that no utterance may make use of more than, for example, three
argument schemes, can be used to limit the amount of search that takes place.

If all sensor probes cost an agent utility, then the heuristic fulfils all of Moore’s requirements
for an argument strategy (Moore, 1993). It maintains the focus of the dispute, as shifting the
focus would require the expenditure of additional utility (via unneeded sensor probes for unneeded
arguments); in an attempt to prove its goals, it will build its point of view (and if the opponent has
mutually exclusive goals, it will attack those), and it will clearly select the utterance that fulfils
these two requirements.

If, however, some sensor probes can be performed with no utility cost, there is nothing stop-
ping the agent from performing these, and thus shifting away from the topic of dispute. In this case,
additional guiding methods, such as requiring the shortest possible utility maximising utterance
4.6. Discussion

Figure 4.6: A screenshot of the implementation of the logical and dialectical layers of our concrete framework. Please refer to the text for a description of the various windows.

...from an agent, are needed to satisfy Moore’s criteria.

By varying the utility cost for sensor probes, or assigning a utility cost to argument schemes, it is possible to generate many different classes of heuristics.

4.6.3 Implementation

We have implemented the algorithm described in Figure 4.4, as well as the supporting machinery to required to perform basic reasoning with our framework. This includes the appropriate Subjective Logic underpinnings and concepts such as evidential nodes, warrants and argument schemes.

Figure 4.6 shows a running instance of this implementation. As can be seen, a graphical representation of the argument graph occupies the centre of the screen. Evidence nodes such as $a$, $\text{implies}(a, b)$ and $b$ have an associated opinion, which can be edited (in the window at the top left of the screen). Argument schemes may be added, and the large window at the bottom shows the form that the Modus Ponens argument scheme may take. In the top right, we can see that when an argument scheme is instantiated into a warrant, variable binding must take place. The user may select what variables should be bound with which evidence nodes. This binding is “smart” in that only types that may be legally bound to the variable are displayed. By selecting a menu option (not shown), the user may cause opinions to be propagated through the network as described in our algorithm.

An earlier version of the heuristic, as well as the dialogue game, were implemented as part of the work described in (Oren et al., 2006b).
4.7 Conclusions

In this chapter we described a concrete argument framework. The framework associates Subjective Logic opinions to literals. We represent evidence using \( \langle \text{literal}, \text{opinion} \rangle \) pairs. Warrants, consisting of premises and conclusions, represent arguments, and are used to modify the strength of evidence. We also formalised the notion of an argument scheme, which allowed us to group warrants using the same form of reasoning together. A language was introduced to allow us to describe how argument schemes interact with warrants and evidence.

At this point, it was possible to generate an argument graph, and we provided an algorithm allowing us to evaluate this graph and assign opinions to all pieces of evidence within the graph. By combining this graph with the work described in Chapter 3, we were able to allow for the representation of some loops between arguments, namely in the case of mutually exclusive conclusions.

To allow agents to use the argument framework, we described a simple dialogue game. The dialogue game was intended to allow agents to argue that their view about the state of a partially observable environment was correct, in cases where any examination of the actual environment state incurred a utility penalty, and could in itself be incorrect.

Finally, we introduced heuristics that allowed the agents to make use of the dialogue game. The heuristic involves a breadth first search of all possible utterances available to the agent, with it selecting the utterance that will maximise its utility. We also suggested that an agent should perform some learning about the environment state during the dialogue game, and showed how this could be done.

One possible application of this type of dialogue game is in the area of contract monitoring and enforcement, and we examine this application in the next chapter.
Chapter 5

Contracting

5.1 Introduction

So far, we have introduced two complimentary models of argument, one abstract (in Chapter 3), and the other concrete (in the previous Chapter). We also showed how the abstract model of argument could be used to represent and resolve some types of loops within the concrete argument model. The concrete model contained a dialogue game that allowed agents to argue about the most likely environment state within a partially observable environment. Access to the environment was provided through a set of fallible sensors, each of which could be asked to reveal a (possibly incorrect) view of a portion of the environment. The process of querying a sensor cost an agent utility, while an agent was able to gain utility by successfully arguing that the environment was in a certain state. Using these concepts, we were able to describe a heuristic agents could use in the course of a dialogue so as to decide which arguments to advance, and which sensors to probe given that their goal involved reaching a maximal utility value.

In this chapter, we examine an application of the entire argument framework, namely determining the most likely state of a contract. In order to perform both contract monitoring and contract enforcement, we must be able to determine what state the system is in, and by using this framework, we are able to do so without the need for a centralised dispute resolution mechanism.

5.2 Contracts

Figure 5.1 shows the contract on which this chapter is based. Our goal is to allow agents to participate in a dialogue with each other with the goal of resolving the issue of this contract has been breached. To do so, they will need to be able to reason about the contract clauses, the obligations imposed by the contract on the contracting parties, and must be able to advance evidence from the environment.

In order to reason about the contract, it must be represented in a machine readable form. In earlier work (Oren et al., 2005), we proposed a simple RDF-based contracting language. However, this language was primarily aimed at service level agreements, rather than general contracts, and had no formal underpinnings, meaning that it is not well suited for representing this contract. A plethora of other contracting languages exist, some with formal underpinnings, such as LCR (Dignum et al., 2002), and others, such as DLP (Milosevic and Dromey, 2002) and NoA²

---

¹This contract is loosely based on a real contract, found at http://users.ox.ac.uk/~juggsoc/contract.shtml, between a student juggling society and an employer.
²See (Vasconcelos et al., 2007) for a first step towards formalising NoA-like contracts.
1. The agreement is entered between the Aberdeen Juggling Club (hereafter referred to as the juggler), and the Aberdeen city council (hereafter referred to as the employer).

2. The juggler will provide the following: 1 performer and all equipment necessary for the performance.

3. The employer will provide the following: Changing room, performance area.

4. Date of Performance: Friday 6 June 2007, Starting at 6:00pm.

5. Duration of event 1.5 hours. Note that short breaks are part of the performance.


7. If the juggler is forced to cancel, all monies will be refunded in full.

8. If the employer is forced to cancel with at least 24 hours notice, all monies will be refunded in full. Otherwise, the deposit will be retained by the juggler.

9. If the event takes place indoors, the juggler will refund $30 in expenses.

10. The employer understands that performances in venues with insufficient heights will hamper the performance.

11. Outside performances cannot take place in bad weather.

12. The employer is responsible for compensating the juggler for damage to the equipment caused by those attending the event unless damage is caused if the performers have left the equipment unguarded.

13. The juggler will be held liable for any injury sustained by a guest at the event if the injury is a result of the provision of services agreed upon in this contract, unless the employer fails to provide a suitable area for performance.

Figure 5.1: The English-language contract which is used as the core of the example.
Contracting Language Features

(Kollingbaum, 2005), containing implicit semantics. Since contract languages are not the focus of this thesis, we will describe the contract in terms of a non-general contract language, complex enough to represent this specific contract, but probably not general enough to represent an arbitrary contract.

Our goal in this chapter is to show how contract monitoring, that is, determining the state of a contract, can be performed using our concrete argument framework. To do so, we need to represent the contract in a way that allows it to be argued about. We thus begin by proposing a number of “strawman” contracting languages. Each of these strawman languages has some features that suggest that they may form the basis of our contracting language. However, we show that certain features are also lacking, tearing down the strawman language. Finally, in Section 5.4, we show how we may create a language by combining the best features from the strawman languages. Once this is done, we show how contract monitoring may take place by using the language and concrete argument framework.

5.3 Contracting Language Features

A contract is built up of clauses, each of which, among other things, imposes a set of norms on some of the contracting parties. At any time, some, or all of a contract’s clauses are “active”. Only the norms associated with active clauses are imposed on an agent. For example, consider a contract with the following three clauses:

1. The seller must pay the buyer $50.

2. The buyer may return the item within 5 days for a full refund.

3. The buyer may return the item within 10 days for a 50% refund.

When the contract is in force, the first clause is active until payment occurs (in fact, without careful translation, the first clause means that a contract violation occurs if the contract comes into force without payment taking place simultaneously). The second clause is active, and provides a permission to the buyer, and imposes a conditional obligation on the seller (dependent upon whether the buyer exercises his privilege) for the first 5 days. The third clause should only become active once the second clause is no longer active, and remains so until 10 days have passed since the contract came into force.

A clause becomes active due to having its preconditions satisfied. In the above example, some of these preconditions were temporal (clauses 2 and 3), and some are true by default (clause 1). Preconditions may refer to

- Environmental states (e.g. if it is rainy, the juggler must perform inside).
- Other clauses (e.g. A clause could state that if the juggler is forced to perform inside due to contractual agreements, they may not use fire).
- Violations of clauses. These can be viewed as a combination of the previous two items.

A clause can have a number of effects including imposing an obligation on one or more agents, invalidating one or more other clauses, or terminating the contract. In the language we will describe below, we primarily focus on only the first of these effects.
There are clearly a number of parallels between the manner in which argument schemes are used in this thesis, and contract clauses. These are summarised in Table 5.1. As shown there, argument schemes and clauses share features such as having a set of activation conditions (namely preconditions for clauses and premises for schemes), and a set of results from having the clause or scheme activated. Similarly, the ability for an argument scheme to have an effect on the argument system when it is instantiated is mirrored by an active clause’s ability to influence the state of a contract. Finally, just as argument schemes may undercut one another, one contract clause may prevent another clause from activating, thereby invalidating it.

These parallels suggest that contract clauses may be represented as argument schemes. To do so would also require the introduction of a number of additional argument schemes to deal with concepts such as the violation of contract clauses and contract termination. Additional schemes would also be required to deal with contract state, norms, and the link between norms and violations. Finally, in any realistic setting, a large number of commonsense argument schemes is also required to deal with basic concepts such as arithmetic, duration and anything else found within the environment to which the contract makes some reference.

As we show below, representing contracts using the language of argument schemes limits the clauses we are able to easily represent. The final language we will introduce is, in large, based on this naive approach, and we thus begin by examining some of the additional argument schemes required to represent a contract. Since we the language we use should allow us to closely mirror the language of our argument framework, we will, while remaining informal, use Prolog-like notation. This will allow concepts such as variable bindings, and premises and conclusions (or, in contracting terms, preconditions and effects), to later be mapped to our language of argument.

One of the simplest inferences any contract reasoning approach needs to be able to make can be written informally as follows:

\[
\text{obliged}(\text{Agent}, \text{State}, \text{Clause}), \neg \text{State} \to \text{violation}(\text{Agent}, \text{Clause})
\]

This states that if, according to Clause, an Agent is required to ensure that a State holds, and it does not hold, then the agent is in violation of the clause\(^3\). The mapping function for this scheme would be proportional to the strength of disbelief in State, and directly proportional to the strength of belief in the obligation. Admissibility may be based on the same opinions, though it could be argued that since results are required to discharge an obligation (i.e. one must ensure that the obligation was in fact discharged, rather than be uncertain, or disbelieve that it is still in force),

\(^3\)In this instance, a violation is associated with a specific clause. Others have linked violations to states and actions (Dignum et al., 2002).
the admissibility function should be dependant on how weak belief is, rather than a combination of the strength of disbelief and uncertainty.

All contract clauses share a similar mapping and admissibility function, defined by the law of the land. Since contracts are normally enforced as civil procedures, the proof requirement on them is that they be admissible “on the balance of probabilities” (Dascalopulu et al., 2001). The admissibility function \( A \) for a contract clause should thus state that a clause is admissible if all its premises \( P \) have a belief greater than 0.5, and we use the principle of the weakest link as a mapping function \( M \):

\[
A = \begin{cases} 
\text{true} & \text{if } \forall p \in P, \text{belief}(p) > 0.5 \\
\text{false} & \text{otherwise}
\end{cases}
\]

\[
M = \omega_p = \langle b, d, u \rangle \text{ such that } b = \min(\text{belief}(p)) \text{ where } p \in P
\]

The second clause in our English language contract, stating that the juggler should provide an entertainer and equipment, could then be represented as the argument scheme

- **Name**: Clause 2
- **Premises**: juggler\( (X) \), equipment\( (Y) \), entertainer\( (Z) \)
- **Conclusions**: obliged\( (\text{juggler}(X), \text{provide}(\text{equipment}(Y))) \)
  
  obliged\( (\text{juggler}(X), \text{provide}(\text{entertainer}(Z))) \)

It should be noted that since the obligation is now part of an argument scheme representing the clause, we may omit the clause label from the obligation itself (assuming, of course, that violations are with respect to states rather than clauses). In the remainder of this chapter, we will, by treating violations as either occurring with regards to a state, or with regards to a norm, be able to treat obligations (and permissions) as having only two parameters.

Predicates such as \( \text{juggler}(X) \) and \( \text{equipment}(Y) \) may be viewed as types. They are used to restrict which instantiated types may be bound within the scheme. Without this restriction, the way in which unification occurs within argument schemes would allow for the binding of any variable, and, within the restrictions of the mapping and admissibility functions, many nonsensical deductions would be allowed within this scheme.

A contract built up of argument schemes is in fact a contract template. That is, it may be instantiated in many ways to provide actual contracts. To instantiate a contract, both parties must agree on a shared meaning regarding the contract’s variables. This shared meaning is obtained by means of the insertion of grounded types, which bind to the variables into a public knowledge base. These grounded types are treated as facts, and assigned an opinion of \( \langle 1, 0, 0 \rangle \), or \( \langle 0, 1, 0 \rangle \), as appropriate. Thus, for example, if a contract is instantiated to include the fact that Bob is the juggler, and that juggling balls are part of the equipment that is to be provided, one would include the facts \( \text{juggler(bob)} \) and \( \text{equipment(jugglingBalls)} \) in the public knowledge base. A warrant could then be derived from the argument scheme whose conclusions impose the appropriate obligation on Bob.
Clause 6, regarding the performance fee, is more difficult to represent in this approach. Here, the employer is obliged to pay the juggler for their services. Since the payment involved is a specific amount, it would be ideal if we could represent this clause using a mixture of variables and constants. Informally, this could be written as \( \text{true} \rightarrow \text{obliged}(\text{employer}(E), \text{pay}($200)) \). However, this is clearly invalid as it contains both a free variable \((E)\), and a fact within its conclusions. It is possible to solve these problems by rewriting the clause as \( \text{employer}(X), \text{pay}(Y) \rightarrow \text{obliged}(\text{employer}(X), \text{pay}(Y)) \). However, fulfilling this obligation leads to a loop, as \( \text{pay}($200) \) would form both a premise, and a conclusion of the second argument scheme. Another problem, as shown in Figure 5.2, is that the \( \text{pay}($200) \) evidence node is assigned two separate opinions, one to show that the money must be paid, and the other to show that it was not paid. It is possible to circumvent this problem by rewriting the clause to read \( \text{employer}(X), \text{amount}(Y) \rightarrow \text{obliged}(\text{employer}(X), \text{pay}(Y)) \), and to then introduce \( \text{amount}($200) \) as a fact. The argument scheme could then be instantiated as a warrant, as shown (together with its violation) in Figure 5.3.

As seen in Figure 5.3, an argument scheme to determine whether a violation occurred is needed. One simple representation for this scheme is:
This choice of mapping function is somewhat arbitrary. It does, however, capture the intuition that a very weak violation will result from a weak belief that an action has not been performed together with a weak belief that the action is obligatory.

Apart from the inelegance involved in representing singletons, as well as requiring all variables to appear in the premises, this approach has another significant problem, namely its lack of reflective capability. For example, clause 10 is (at least somewhat) dependant on whether clause 9 is active or not. Many other cases can be imagined where a clause must examine the status of another clause to determine whether it is active or not. Thus, an agent should be able to make use of the argument system to reason about the properties of clauses. Since the argument framework does not provide any way of interacting with argument schemes (except for instantiating them into warrants), an agent cannot perform reflective reasoning with this type of contract representation.

As mentioned previously, additional argument schemes may be required for contract enforcement to take place. As seen in clause 13, the juggler is liable for any injuries sustained during a performance. If this clause comes into effect, agents would have to argue about the concept of liability, how much damages should be paid, and the like. It is useful to distinguish between these additional argument schemes and those argument schemes that emerge from the contract. This cannot be done with the framework as it currently stands.

A more philosophical weakness, partially related to the one described in the previous paragraph, is that even though similarities exist between contract clauses and argument schemes, they are conceptually different entities. Argument schemes are intended to represent general forms of argument, contract clauses are specific entities, applicable only in the context of a single contract. Clause 12 demonstrates another philosophical problem with this approach: if a piece of juggling equipment is damaged, the agent must probe for its value before the clause becomes valid. While it could be argued that this is correct, as the evidence must be gathered before enforcement can take place, the clause is intended to convey that a broken item must be replaced by the employer, regardless of the item’s cost. If clause 12 is interpreted as stated, a disagreement on the item’s cost can invalidate the obligation. Thus, the intermingling of contract clauses and instantiated facts is undesirable.

Given these weaknesses, another way of representing contract clauses is clearly needed. The most obvious option involves representing contract clauses within the object language, that is, the language over which argument takes place, and to then utilise argument schemes to reason about
these contract clauses. Thus, a contract clause could be represented in the form

\[
\text{clause} (\text{clauseName}, \text{preconditions}, \text{effects})
\]

However, as we will now explain, the object language is too weak to directly represent contracts in this manner.

Let us consider clause 2 (regarding the provision of equipment) in the original contract. One way of representing is by using an evidential node with opinion \( (1, 0, 0) \) and the grounded type

\[
\text{clause} (\text{clause2}, \text{effect} (\text{obliged(juggler(bob)), provide(equipment)})\), \\
\text{effect} (\text{obliged(juggler(bob), provide(entertainers)})\})
\]

To reason about contracts in this form requires the introduction of a number of argument schemes. One weakness of the argument framework becomes apparent here, namely its inability to utilise conjunctions of arbitrary length. In other words, instead of having an argument scheme saying “if all the preconditions hold, then the scheme’s effects hold”, we need to introduce argument schemes of the form

\[
\text{clause} (\text{Name}, \text{effect}(A)) \rightarrow A \\
\text{clause} (\text{Name}, \text{effect}(A), \text{effect}(B)) \rightarrow B \\
\text{clause} (\text{Name}, \text{precondition}(A), \text{effect}(B)), A \rightarrow B \\
\text{clause} (\text{Name}, \text{precondition}(A), \text{precondition}(B), \text{effect}(C)), A, B \rightarrow C
\]

These argument schemes are used to determine the effects of a clause. Given only these schemes, this approach appears to be no more useful than the “clauses as argument schemes” approach described above. In fact, given that a large number of schemes need to be introduced to handle multiple preconditions and effects, it is possible to argue that the previous approach is superior to it (though it is possible to generate this family of schemes automatically, somewhat mitigating this weakness). Furthermore, additional argument schemes are required before contract monitoring may proceed. Some of these schemes may be compactly described using this approach, for example, schemes of the form

\[
\text{clause} (\text{N}, \text{precondition}(A), \text{effect}(\text{obliged}(B)))\), A, \neg B \rightarrow \text{viol(N)}
\]

where the mapping function is inversely proportional to the belief within the opinion for \( B \), can be used to determine whether a violation took place.

Reflection using this approach is relatively easy. For example, determining whether a clause is in effect can be done using the argument scheme

\[
\text{clause} (\text{Name}, \text{precondition}(A), \text{precondition}(B)), A, B \rightarrow \text{inEffect}(\text{Name})
\]
While determining whether a violation is, or isn’t handled by the contract, can be computed using
the following argument scheme:

\[
\text{clause}(\text{Name}, \text{precondition}(\text{viol}(N))) \rightarrow \text{handles}(\text{Name}, \text{viol}(N))
\]

Since the parameters of a clause are ordered, we would need a large number of additional argument
schemes to be able to (generally) determine whether a violation is, or is not, handled by the
contract.

While the separation of the object language from the language of argument allows for refl-
ctivity, and overcomes a number of other criticisms levelled at the previous approach (such as the
fact that contract clauses should be viewed as distinct from argument schemes), this approach con-
tains a number of weaknesses. The first, which has already been touched on, is due to the inability
of the argument language to deal with arbitrary length lists of predicates. Therefore, families of
argument schemes must be created to deal with different length contract clauses.

Another weakness can be seen when one examines clauses 12 and 13 from the original con-
tact. One way to represent these clauses in the language is as follows:

\[
\text{clause}(12, \text{precondition}(\text{guarding}(\text{equipment})), \text{precondition}(\text{damaged}(\text{equipment}))), \\
\text{effect}(\text{obliged}(\text{alice}, \text{pay}(\text{price}(\text{equipment}))))) \\
\text{clause}(13, \text{precondition}(\neg \text{viol}(3)), \text{precondition}(\text{injured}(\text{guest}))), \\
\text{effect}(\text{liable}(\text{bob}, \text{injury}(\text{guest}))))
\]

Both of these clauses attempt to refer to a concept, namely equipment, and a guest, which, when the clause is in effect, should be narrowed down to a specific individual. Without some sort of
variable reference, this cannot be represented when using this approach. One way to work around
this limitation, assuming a closed world, is to instantiate a different clause for every possible
individual. This is very cumbersome, and, since most complex domains are open, inapplicable in
most cases.

What is clearly required, and we will now describe, is a technique that takes the best features
of both approaches and combines them. This approach should allow for variables and instantiation,
as was the case in the argument scheme based contract representation approach, but should also
represent the contract clauses using the object language. To achieve this, a new language must be
introduced, which can then be translated into the language used for argument as needed.

### 5.4 A Simple Contracting Language

Our language is intended to closely match the language of argument. It is built around the concept
of a clause, with each clause containing a set of preconditions and effects. Typically, the effects of
a clause are normative in nature. Syntactically, clauses are written in a form similar to that shown
in the previous approach to contract representation. Unlike that technique, however, clauses may
also contain variables, which may then be instantiated to determine whether the clause is in effect
or not. Thus, for example, clause 13 could be written as

\[
\text{clause}(13, \text{precondition}(\neg \text{viol}(3)), \text{precondition}(\text{injured}(\text{guest}(X)))), \\
\text{effect}(\text{liable}(\text{juggler}(Y), \text{injury}(X))))
\]
The unification process for variables $X$ and $Y$ functions very differently here when compared to how unification takes place within argument schemes. Before describing this process, we will describe the syntax and semantics of the contracting language. Table 5.2 summarises the language syntax in BNF form.

**Definition 5.1. Contracts and Clauses**

A contract $C$ is a set containing one or more clauses. A clause $c_i$ is a tuple, containing zero or more preconditions, and one or more effects. Each clause is associated with a set of variable names, taken from some alphabet of symbols $\Sigma$:

$$C = \{c_1, \ldots, c_l\}, |C| \geq 1$$
$$c_i = (P, E, V)$$
$$P = \{p_1, \ldots, p_m\}, |P| \geq 0, p_i \in \text{Pred}$$
$$E = \{e_1, \ldots, e_n\}, |E| \geq 1, e_i \in \text{Pred}$$
$$V \subseteq \Sigma$$

A contract then, is built up of a number of clauses. Each clause consists of a set of preconditions and effects. Each precondition and effect consists of a single predicate.

**Definition 5.2. Predicates**

A predicate $P = (\text{Name}, \text{Parameters})$ is a member of the set of Predicates $\text{Pred}$, and is associated with a name, and an ordered set of parameters. $\text{Name} \in \Delta$, the predicate name, is taken from an alphabet of symbols $\Delta$. This alphabet is distinct from $\Sigma$. Each parameter $q \in \text{Parameters}$ is either a predicate itself, or an element of $\Sigma$. As usual, the arity of a predicate is equal to the size of its parameter set.

The form of predicates within a contract clause is further constrained. Given a contract clause $c_i$ associated with variables $V$, any variables found within the predicate must be elements of $V$. It should be clear that there is a direct mapping between these types of predicates, and instantiated types within the argument language (when no variables exist within the predicate).

Clauses are special types of predicates containing a single literal predicate (the clause name), and an arbitrary number of unary precondition$(\ldots)$ and effect$(\ldots)$ predicates. The latter two predicates list the clauses’ preconditions and effects respectively. For syntactic convenience, elements of $V$ consist of those terms beginning with a capital letter, while any other word within a
Chapter 5.5: Using Contracts in a Dialogue

88

<table>
<thead>
<tr>
<th>Clause</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>clause(c2,effect(obliged(juggler(X),provide(equipment(E))))))</td>
</tr>
<tr>
<td>3</td>
<td>clause(c3,effect(obliged(employer(X),provide(changingRoom))), effect(obliged(employer(X),provide(PerformanceArea)))))</td>
</tr>
<tr>
<td>4,5</td>
<td>clause(c45,effect(obliged(juggler(X),arriveBefore(PerformanceStartTime))), effect(obliged(juggler(X),juggleFor(PerformanceLength)))))</td>
</tr>
<tr>
<td>5</td>
<td>clause(c5,effect(employed(juggler(X),shortBreaks))))</td>
</tr>
<tr>
<td>6</td>
<td>clause(c6,effect(obliged(employer(X),pay(200))))</td>
</tr>
<tr>
<td>7</td>
<td>clause(c7,precondition(cancel(juggler(X))), effect(obliged(juggler(X),refund(30))))</td>
</tr>
<tr>
<td>8</td>
<td>clause(c8,precondition(cancel(employer(X))), precondition(lessThan24Hours(CancelDate,PerformanceStartTime)), effect(obliged(juggler(Y),refund(30))))</td>
</tr>
<tr>
<td>9</td>
<td>clause(c9,precondition(cancel(employer(Y))), precondition(inactive(clause(8))), effect(obliged(juggler(X),refund(200))))</td>
</tr>
<tr>
<td>10</td>
<td>clause(c10,precondition(indoors(PerformanceArea)), effect(obliged(juggler(X),refund(30))))</td>
</tr>
<tr>
<td>11</td>
<td>clause(c11,precondition(lowCeiling(PerformanceArea)), effect(allowed(juggler(X),simplerPerformance))))</td>
</tr>
<tr>
<td>12</td>
<td>clause(c12,precondition(badWeather), effect(obliged(bob,indoors(PerformanceArea))))</td>
</tr>
<tr>
<td>13</td>
<td>clause(c13,precondition(guarding(equipment(E))), precondition(damaged(equipment(E))), effect(obliged(employer(X), pay(price(equipment(E))))))</td>
</tr>
<tr>
<td>14</td>
<td>clause(c14,precondition(¬ viol(3)), precondition(injured(G)), effect(liable(juggler(X),injury(G))))</td>
</tr>
</tbody>
</table>

Table 5.3: Representing the contract using the new contracting language.

The predicate is a member of \( \Delta \). One possible representation of the juggling contract is then shown in Table 5.3.

5.5 Using Contracts in a Dialogue

As mentioned previously, a number of argument schemes are required so as to be able to use a contract within a dialogue. In this section, we describe a number of these schemes, and discuss how to transform a contract into a form that is usable within a dialogue.

5.5.1 The Family of Fundamental Argument Schemes for Contracting

The first set of argument schemes we will examine are the ones involved in the application of a contract clause. That is, they allow us to determine, given that a clause’s preconditions hold, what effects should come into force.

Given a variable-free set of contract clauses \( c_1, \ldots c_n \) where each \( c_i, 1 \leq i \leq n \), is of the form:

\[
c_i = \text{clause}(\text{Name}, \text{precondition}(p_1), \ldots, \text{precondition}(p_a), \text{effect}(e_1), \ldots, \text{effect}(e_b))
\]
We define, for each unique combination of preconditions \((a)\) and effects \((b)\), an argument scheme with

\[
\begin{align*}
\text{Name} : & \quad F_{a,b} \\
\text{Premises} : & \quad \text{clause}(\text{Name}, \\
& \quad \text{precondition}(P_1), \ldots, \text{precondition}(P_a), \\
& \quad \text{effects}(E_1), \ldots, \text{effects}(E_b)), \\
& \quad P_1, \ldots, P_a \\
\text{Conclusions} : & \quad E_1, \ldots, E_b \\
\text{Admissibility} : & \quad \forall P_i, i = 1 \ldots a, \text{belief}(P_i) > 0.5 \\
\text{Mapping} : & \quad \text{if } a > 0, \text{min}(\text{belief}(P_i)), i = 1 \ldots a, \\
& \quad \text{else } \langle 1, 0, 0 \rangle
\end{align*}
\]

Then the family of fundamental argument schemes is the set containing all these schemes.

### 5.5.2 Handling Variables in Contracts

The family of fundamental argument schemes, and all other argument schemes we will discuss, operate on clauses that contain no variables. We must thus be able to convert between clauses with variables, as they appear within a contract, and a variable-free form.

We have already seen that a contract may contain a large number of variables. Some of these variables may be “eliminated” when the contract is instantiated, by having only a single element from the knowledge base be able to be bound to them. For example, a typical contract may contain the English language clause

...This agreement is dated and in effect as of the **19 December 1974**, between **Bob Hodgkins** of Minnesota, hereafter referred to as “Client”...

with the portions in bold filled in by the contracting parties. The Client was originally a variable within the contract, but is now, for the duration of the contract, bound to Bob Hodgkins.

Other variables may only be able to be bound to a clause as the contract is being executed. Consider, for example, clause 13. Assume, that during the performance, a guest is injured. It is only at this time that we may bind the appropriate guest to the variable and determine that the clause is in effect.

A third way to utilise variables in the contract, can be seen in clauses 6, 7, 8, 9 and 10. Here, the amount to pay was directly inserted into the contract. This may, on occasion, lead to some confusion. For example, at the end of the dialogue, both \textit{pay}($30) and \textit{pay}($200) may have a high belief assigned to them. In some cases (such as this one), it is possible to determine what is meant by the contract by looking at which clauses are active, but in many cases, this representation may lead to some confusion. This problem does not arise because of the way in which the amount is inserted directly into the contract, but rather because of the way the \textit{pay} predicate is used in multiple clauses.

The first two types of variable bindings may be encoded as \textit{facts}, that is, as something that is known about the world. In our argument based approach, these facts will be predicates associated with a certain opinion. What we therefore need to do is specify a procedure that will allow us to...
replace these variables with facts in our clauses. These variable free clauses can then themselves be seen as facts, and can be used during the course of an argument, for example, through the fundamental family of argument schemes, or other argument schemes which we introduce below.

The basic idea behind variable binding is that the variable’s enclosing predicate must match an accepted fact so as to allow the binding to take place. Thus, for example, given the facts \textit{juggler(bob)} and \textit{juggler(father(bob))}, both \textit{bob}, and \textit{father(bob)} may be bound in place of \textit{X} within the clause \textit{clause(effect(obliged(juggler(X), perform)))}. As usual, variables with the same symbol must be bound to an identical predicate. One complication is that bindings operate slightly differently within preconditions and within effects.

The preconditions for a clause state what should be the case in the environment for the clause to be in effect. Variables in preconditions may thus only be bound to existing facts, as described in the previous paragraph.

Bindings of variables within effects may refer to existing facts in the knowledge base, or may cause new facts to appear. This is handled in one of two ways. First, if a variable appears, and is bound in a precondition, that binding also occurs in the effect clause. If the variable is free, it may be bound to any fact in the knowledge base, as described above. Only variables that are bound within a precondition may cause new facts to appear using the effects of a clause. The creation of new facts in this manner gives us yet another avenue for performing inference. Figure 5.4 describes this process formally.

The first algorithm in the figure performs binding for a single predicate and fact. That is, given a predicate containing variables, and a fact from the knowledge base, it will determine if there is a mapping between the two based on the unification of the predicate’s variables. The bind function will recurse into a predicate’s parameters, matching variables deep within a predicate if needed.

The \textit{computeClauseBindings(…)} function takes in a clause, containing multiple precondition and effect predicates, as well as a knowledge base, and determines whether some binding is possible. Lines 1–4 perform the bindings over the clause’s preconditions. Here, only existing facts may be bound. The remainder of the algorithm is tasked with binding the effects predicates. If a possible binding already exists (lines 11–16), it is used in the effect nodes. If however, no such binding exists, in other words, if the variable in the affected predicate is independent of the preconditions, we must compute any possible bindings from the knowledge base (lines 6–10). Performing the bindings in this way allows us to introduce new information into the knowledge base if necessary, but only as a last resort.

An agent may use the \textit{computeClauseBindings(…)} function to introduce contract clauses into the argument knowledge base, based on existing arguments within the knowledge base. This algorithm is therefore an argument scheme, albeit one that cannot be represented using the language of Chapter 4. Before looking at reasons for this, we must determine this argument scheme’s mapping and admissibility functions, as well as how this scheme may be used.

This argument scheme, hereafter referred to as the \textit{clause instantiation argument scheme}, can be used at any time. This may seem counterintuitive at first, as an agent could instantiate clauses for which no backing evidence exists. However, since the only effect of this argument scheme is the insertion of a contract clause without variables into the knowledge base, this argument scheme
bind(Predicate, GroundedType, V)

Where GroundedType = \langle (f_n, f_e), (f_{p1}, \ldots, f_{pl}) \rangle
Predicate = \langle p_n, (p_1, \ldots, p_m) \rangle

Assumptions: the number of parameters for the predicate & instantiated type are equal.
VariableBindSet is an array indexed by a variable.

no match results are passed all the way up the call stack.

1 if (p_n ≠ f_n ∨ |f_e| ≠ m)
2 return no match
3 for each i where i = 1 \ldots f_e {  
4 if p_i ∉ V
5 VariableBindSet[p_i] = VariableBindSet[p_i] ∪ bind(p_i, f_{pi}, V)
6 else
7 return f_{pi}
8 }
9 return VariableBindSet

calculateClauseBindings(C, F)

Where: C is a clause with precondition predicates (p_1, \ldots, p_m)
effect predicates (e_1, \ldots, e_n)
variables V

F is a set of facts consisting of evidence nodes f_1, \ldots, f_o

Assumptions: no match results from the bind function are treated as {} in line 03.
line 03 uses bind(p_{i}, f_j, V)[v] if variable v has not been bound before.

01 for each p_i, i = 1 \ldots m {  
02 for each f_j, j = 1 \ldots o {  
03 ∀ v ∈ V, vbs[v] = vbs[v] ∩ bind(p_i, f_j, V)[v]  
04 }  
05 }
06 for each e_i, i = 1 \ldots n {  
07 if e_i contains variable v and vbs[v] does not exist {  
08 for each f_j, j = 1 \ldots o {  
09 ∀ v ∈ V, vbs[v] = vbs[v] ∪ bind(e_i, f_j, V)[v]  
10 }  
11 } else {  
12 for each f_j, j = 1 \ldots o {  
13 ∀ v ∈ V, if bind(e_i, f_j, V) ≠ no match
14 vbs[v] = vbs[v] ∩ bind(e_i, f_j, V)[v]  
15 else do nothing
16 }  
17 }
18 }
19 if ∃ v ∈ V such that vbs[v] = {}, return fail, else, return vbs.

Figure 5.4: The algorithm used to determine which variables may be bound within a clause.
is harmless. The only way non-clausal predicates can be introduced into the knowledge base with this argument scheme is in combination with the family of fundamental argument schemes. Since these require evidence for their preconditions, it is safe to apply this argument scheme in this way.

Since contract clauses are determined by the contract, any clause introduced using this argument scheme is associated with a dogmatic belief, i.e. an opinion of $(1, 0, 0)$.

A number of factors conspire to make this argument scheme impossible to represent in the standard way, and also nicely illustrate the limitations of our framework. The first limitation arises because variables may not encapsulate other variables. In other words, a combination of variables representing any predicate with one parameter, written as $A(B)$ is an illegal construct within our language. This means that it is difficult for argument schemes to talk about, and refer to, other argument schemes, and led directly to the manner in which undercutting between argument schemes was represented.

Related to this is the coarseness of our variable representation. In our framework, any predicate may bind with a variable. It is often useful to be able to specify that only predicates with a certain pattern (probably based on their name, cardinality, or type of parameters), may be bound to a variable.

Finally, the fixed length of predicates makes it difficult to represent such a general argument scheme. In the previous section, we avoided this difficulty by introducing a family of argument schemes, but such a solution is too cumbersome to be used in this context. Overcoming these problems would require the design of a rich and powerful metalanguage, and is discussed in Chapter 6.

Since this argument scheme cannot be represented in our argument language, we may ask how an agent may make use of it in its internal reasoning, and how it may be used within a dialogue.

Internally, an agent holds beliefs about the world, and it is from these beliefs that it may generate instantiated contract clauses. Therefore, an agent must, every time its beliefs are modified, run the algorithm to determine which contract clauses it may utilise in argument.

Introducing the clause into a dialogue can be achieved in multiple ways:

1. As discussed previously, by modifying the argument language, contract clause inference could be treated as yet another argument scheme. Agents could then make use of this scheme to perform the introduction.

2. The dialogue itself could be modified, allowing agents to not only make utterances and probe sensors, but also to introduce the contract clauses as part of their utterance.

3. The clause could be introduced as a zero cost sensor probe. Depending on the manner in which the system is implemented, the sensor probe could then ensure that the clause may indeed be introduced into the dialogue.

4. Simply insert any contract clause introduced by the agent into the public knowledge base.

---

4The computational complexity of this approach is very high, since many useless bindings may be made. Simple optimisations, such as allowing only predicates with some belief or disbelief to be bound will greatly reduce this cost.
The first option would involve a major overhaul of the argument system, and is thus clearly not a realistic option. The second option is a possibility, but adds unneeded complexity to the dialogue. The remaining options can be introduced into the system with no changes to the framework, and are thus both promising approaches. It could be argued that the introduction of a new contract clause into the knowledge base is distinct from a sensor probing action. However, since a contract is a real world artifact, a counterargument exists stating that the sensor probe still reflects an opinion about an environmental entity (namely the contract). One disadvantage of this approach is the duplication of labour; not only does the agent determine that a clause may be instantiated, but the sensor must also ensure it is a valid instantiation. The last approach overcomes this problem, but requires that contract clauses be treated differently to all other predicates in the system (so that they may be introduced without a sensor probe). Since we are trying to minimise the number of special cases the framework must deal with, we will use the third approach in the remainder of this chapter.

5.5.3 Detecting Clause Applicability

The family of fundamental argument schemes can be used to “activate” a clause, that is, to add its effects to the public knowledge base. It is also sometimes useful to be able to determine which clause was activated, and we will now define another family of argument schemes that allows us to do this.

Given a variable-free contract clause $c$ of the form

$$\text{clause}(\text{Name}, \text{precondition}(p_1), \ldots, \text{precondition}(p_a), \text{effect}(e_1), \ldots, \text{effect}(e_b))$$

we define, for each unique combination of the number of preconditions and effects, $(a, b)$, an argument scheme

- **Name**: $\text{active}_{a,b}$
- **Premises**: $\text{clause}(\text{Name},$
  $\text{precondition}(P_1), \ldots, \text{precondition}(P_a),$
  $\text{effect}(E_1), \ldots, \text{effect}(E_b),$
  $P_1, \ldots, P_a$
- **Conclusions**: $\text{active}(\text{Name})$
- **Admissibility**: $\forall P_i, i = 1 \ldots a, \text{belief}(P_i) > 0.5$
- **Mapping**: if $a > 0$, $\text{min}(\text{belief}(P_i)), i = 1 \ldots a$
  else $\{1, 0, 0\}$
5.5. Using Contracts in a Dialogue

Similarly, we define the inactive<sub>a,b</sub> argument scheme as

Name: inactive<sub>a,b</sub>
Premises: clause(Name,<br>precondition(<i>P</i><sub>1</sub>),...,precondition(<i>P</i><sub>a</sub>),<br>effects(<i>E</i><sub>1</sub>),...,effects(<i>E</i><sub>b</sub>)),<br><i>P</i><sub>1</sub>,...,<i>P</i><sub>a</sub>
Conclusions: active(Name)
Admissibility: \( \forall P_i, i = 1 \ldots a, belief(P_i) < 0.5 \)
Mapping: if \( a > 0 \), \( min(belief(P_i)), i = 1 \ldots a \)<br>else \( (0, 1, 0) \)

5.5.4 Violations and Norms

As will be seen in the next section, a contract clause typically imposes an obligation on one of the contracting parties. Some contract clauses may also impose a prohibition on a contracting party, or grant it some sort of permission. We represent all of these normative acts as predicates:

Definition 5.3. (Normative Predicates) We define three special normative predicates representing obligations, permissions and prohibitions:

\[
\text{obliged}(A, X), \text{permitted}(A, X), \text{prohibited}(A, X)
\]

Where \( A \) and \( X \) may be variables or predicates, depending on whether these predicates appear in instantiated or uninstantiated contract clauses.

If an obligation is not met, a violation occurs. To describe a violation, we need to determine what it is that is actually being violated, and in what way. Consider the clause “The juggler is obliged to juggle for one and a half hours”, and assume that the juggler only juggled for an hour. What then was violated? Was it the clause, the obligation, or both?

The main goal of our contracting language is to detect violations and reason about them, and the manner in which we represent violations is thus critical to our language. Since violations must be reasoned about, we will represent violations as a predicate.

Definition 5.4. (Violations) Given a contract clause of the form

\[
c = \text{clause}(\text{Name}, \text{precondition}(p1), \ldots, \text{precondition}(p_a), \text{effect}(e_1), \ldots, \text{effect}(e_b))
\]

Where, \( e_i = \text{obliged}(A, X) \) for some \( 1 \leq i \leq b \), \( A \) and \( X \), we define a special predicate, referred to as the violation predicate as

\[
\text{viol}(\text{Name}, e_i)
\]

We may also define a unary violation predicate for a given obligation, \( \text{obliged}(A, X) \), as

\[
\text{viol}(A, X)
\]
Since the violation predicate encapsulates the clause name, as well as the violated obligation, we can determine exactly where in the contract the violation occurred, and which agent caused it.

### 5.5.5 Clauses and Norms

The most common effect of a contract clause is to impose a norm on some entity. These norms are usually either obligations, permissions, prohibitions. Contract monitoring attempts to detect if these norms were violated, and arguments about the status of norms thus lies at the heart of this thesis.

We can define the following argument scheme to detect violations with regards to obligations:

- **Name**: obligationEnforcement
- **Premises**: obliged(A, X), X
- **Conclusions**: viol(A, X)
- **Admissibility**: belief(obliged(A, X)) > 0.5 ∧ belief(X) < 0.5
- **Mapping**: obliged(A, X) ∧ ¬X

This scheme claims that an agent A is in violation of its obligation to see to it that state X holds if it is believed that such an obligation exists, and it is believed that X does not hold. As was the case with other argument schemes, a number of other possibilities exist for the mapping and admissibility function. For example, the mapping function used above takes a very broad view of the notion of fulfilling a commitment. In many cases, we would want to say an agent is in violation of an obligation unless they can positively show that the obligation has been fulfilled. In other words, a high disbelief or uncertainty value for X in the above argument scheme should lead to a high belief in the violation predicate. In such cases, the following mapping function may be more appropriate:

\[ \text{obliged}(A, X) \land \langle \text{disbelief}(X) + \text{uncertainty}(X), \text{belief}(X), 0 \rangle \]

To link a violation to a contract clause, we may use the following scheme:

- **Name**: contractObligationEnforcement
- **Premises**: active(C), containsEffect(C, obliged(A, X)), viol(A, X)
- **Conclusions**: viol(C, obliged(A, X))
- **Admissibility**: containsEffect(C, obliged(A, X)) = \(\langle 1, 0, 0 \rangle\), belief(active(C)) > 0.5, belief(viol(A, X)) > 0.5
- **Mapping**: viol(A, X)

The predicate \(\text{containsEffect}(\ldots)\) may be instantiated by a family of reflective argument schemes that allow us to detect whether a contract clause contains the relevant effect.

It makes little sense to speak about the violation of a permission. However, a permission to have a certain state of affairs may attack an argument stating that a violation has occurred. we will not propose a general argument scheme dealing with this, instead handling permissions on a
case-by-case basis. Prohibitions are the third element of norms. We may define the following argument scheme to represent prohibitions:

Name: prohibitionEnforcement
Premises: prohibited(A, X), X
Conclusions: viol(A, X)
Admissibility: belief(prohibited(A, X)) > 0.5 \land belief(X) > 0.5
Mapping: prohibited(A, X) \land X

Here, if we believe that something is prohibited, and we believe it has in fact occurred, then we may say that a violation has occurred. Again, we may want to err on the side of caution and require that the agent prove that X has not occurred. In this case, we could, as in the case of an obligation, include uncertainty in the mapping (and admissibility) functions.

In deontic logics, prohibitions are often represented as obligations to not have a certain state hold. This makes sense intuitively, and suggests that we could have simply had an argument scheme for prohibitions of the form

\[ \text{obliged}(A, \neg X), X \rightarrow \text{viol}(A, X) \]

The introduction of the concept of uncertainty makes this infeasible, leading to the more complicated form of prohibitions (and obligations) described above.

### 5.5.6 Dealing with Violations

There are a number of ways of dealing with violations in a contract. The juggling contract does not represent some of these possibilities, and in this subsection we will therefore refer to both it, and a different example.

Consider a contract containing the following clauses:

1. If the juggler is forced to cancel, all monies will be refunded in full.
2. The employer is responsible for guarding their performer’s possessions while the show is taking place.
3. The employer is responsible for compensating the juggler for any damage to their possessions caused by those attending the event unless the possessions were properly guarded.

As usual, the translation process from English language clauses to contracting language clauses is not automatic, and multiple possible representations exist. For example, we could (informally) represent the first clause as

\[ \text{juggler}(X), \text{cancel}(X, \text{performance}) \rightarrow \text{obliged}(X, \text{refund}(\text{Money})) \]

It is important to note that the contract does not state what will happen if this obligation is violated (apart from the viol(…) predicate holding). Compare this to how a violation is handled
in the other contract clauses. One way of representing these is

\[
true \rightarrow \text{obliged}(employer(X), guard(employer(X), Equipment)) \\
\neg guard(employer(X), Equipment), \text{damaged(Equipment)} \rightarrow \text{obliged}(X, \text{compensate(juggler}(Y), \text{cost(Equipment))))
\]

This representation is unsatisfactory, as the link between the two clauses is purely implicit. Unlike the manner in which the first clause was handled, the agents here know how to handle a violation of the second clause. However, without an explicit link, it is hard to allow the agents to do this. Repetition between clauses, as found in the example is also undesirable from a maintenance perspective. It is preferable to represent contrary-to-duty obligations in a form similar to the following:

\[
\text{clause}(1, \text{effect(}\text{obliged}(employer(X), \text{guard(equipment}(Y)))))) \\
\text{clause}(2, \text{precondition(}\text{viol}(1, A)), \text{precondition(}\text{damaged(equipment}(Z))))), \\
\text{effect(}\text{obliged}(employer(X), \text{compensate(juggler}(Y), \text{cost}(Z))))))
\]

It would be useful to be able to refer to variables between clauses in such situations; we assume in clause 2 that only one juggler and employer exists, as well as that only one piece of equipment is guarded. If many instances of these objects exist, duplication of predicates will be required. Extending the language to share variable references between the primary and contrary-to-duty clauses is not difficult, but is beyond the scope of this thesis.

Since some violations are handled outside the contract (as in the first and last clause), and some within the contract (as in the second clause), it is important to allow the agents to determine which type of violation they are dealing with. This involves introducing the following simple family of argument schemes:

\[
\begin{align*}
\text{Name} : & \quad \text{violationHandledInContract} \\
\text{Premises} : & \quad \text{clause(Name, precondition(viol(A, B)))} \\
\text{Conclusions} : & \quad \text{violationHandledIn}(A, \text{Name}) \\
\text{Admissibility} : & \quad \text{true} \\
\text{Mapping} : & \quad (1, 0, 0)
\end{align*}
\]

The above scheme represents an argument with one precondition, additional members of this family of schemes would contain preconditions of the form

\[
\text{clause}(\text{Name, precondition(viol(A, B))), X_i)
\]

Where \(X_i\) is an arbitrary length list of variables representing the other preconditions and effects
of the clause.

We have examined a number of contract and obligation related argument schemes. These schemes are far from exhaustive, but cover some of the basic patterns often encountered when reasoning about contracts. Additional argument schemes are however required to handle a number of other concepts, including notions such as time, basic arithmetic, and the like. We will not exhaustively study these, instead introducing them as required by the example. In the next section, we will examine how agents may argue about the status of a contract.

## 5.6 Arguing about Contracts

Assume that a pair of agents have agreed upon the contract shown in Figure 5.3. To operate, the contract must be instantiated, and to do this, the following facts (with opinion \( \{1, 0, 0\} \)) are inserted into both agent’s (and the public) knowledge base:

<table>
<thead>
<tr>
<th>juggle(bob)</th>
<th>employer(alice)</th>
<th>equipment(jugglingBalls)</th>
</tr>
</thead>
<tbody>
<tr>
<td>equipment(devilSticks)</td>
<td>arriveBefore(18:00)</td>
<td>juggler(1.5)</td>
</tr>
</tbody>
</table>

Very few facts are needed to instantiate the contract as much of it is already in concrete form; for example, the cost of the performance and the refund amounts are not treated as variables within this contract.

The agents may now execute the contract, forming opinions about the state of various predicates. Assume that at some point, the employer (Alice) ends up with the following beliefs, and their associated opinions, in her knowledge base:

<table>
<thead>
<tr>
<th>Belief</th>
<th>Opinion</th>
</tr>
</thead>
<tbody>
<tr>
<td>arriveAt(18:30)</td>
<td>(0.7, 0.2, 0.1)</td>
</tr>
<tr>
<td>leaveAt(19:30)</td>
<td>(0.8, 0.1, 0.1)</td>
</tr>
</tbody>
</table>

Therefore, she believes that Bob has not only arrived late, but has also not juggled for a sufficient length of time. Alice also determines that this type of violation is not handled by the contract, and decides to begin a contract enforcement action. As described in Chapter 4, Alice may not simply advance her arguments, but must provide justification for them in the form of evidence (via sensor probes). We assume that Alice obtained her beliefs regarding what time Bob arrived and left by asking someone (Charlie) at the event what they saw. She can thus make the following (rather cumbersome) utterance:
{(}, {C}, sensorScheme), ({}), {arriveAt(18 : 30)},
(sensorScheme), ({}), {later(18 : 30, 18 : 00)}, {arriveAt(18 : 30, O, later(18 : 30, 18 : 00))}, {arriveBefore(18 : 00)},
CS_{1}, ({}), {arriveBefore(18 : 00)}, {viol(juggler(bob), arriveBefore(18 : 00))}
, obligationEnforcement), ({}), {active(c45)}, active_{0,2}, ({}),
{containsEffect(c45, O}), containsEffect_{0,2}, ({}),
containsEffect(c45, O), viol(juggler(bob), arriveBefore(18 : 00)), {viol(c45, O)},
contractObligationEnforcement), {charlie(arriveTime(18 : 30)),
contractSensor(C), mathsSensor(later(18 : 30, 18 : 00))}

where

\[
C = \text{clause(c45, effect(obliged(juggler(bob), arriveBefore(18 : 00))))},
\]

\[
effect(\text{obliged(juggler(bob), juggleFor(1.5)))})
\]

\[
O = \text{obliged(juggler(bob, arriveBefore(18 : 00)))}
\]

This is illustrated graphically in Figure 5.5. Informally, she probes sensors to determine what time Bob arrived, and to introduce the relevant clause into the public knowledge base. She then says that since the clause is active, and that since arriving after the time one was supposed to arrive means that one is late (this is a “commonsense” argument scheme), Bob was in violation of clause c45 in the contract. The only non-dogmatic sensor probe is arriveAt(18 : 30). Assume that Charlie claims that arriveAt(18 : 30) was the case with opinion \langle 0.7, 0.1, 0.2 \rangle. Thus, once the utterance is made, the violation predicates are assigned an opinion of \langle 0.7, 0.1, 0.2 \rangle. Since it adds little to the example, unless explicitly mentioned, we ignore the heuristic aspect of the dialogue; we may assume that the utility gained from winning the dialogue is greater than the sum of the costs of the sensors probed.

Since Bob believes that he arrived on time, he counters Alice’s utterance by performing another sensor probe regarding his arrival time at the party (in other words, he asks another witness, David, for evidence that will contradict Charlie’s evidence). While it is possible to represent this act as a simple sensor probe which, if Bob is correct, weakens belief in arriveAt(18 : 00), a more detailed examination of this action reveals some difficulties inherent in this approach.

Consider the case where Charlie claims Bob arrived at 18:30, while David claims that Bob arrived at 18:29. Since opinions are assigned to the predicates arrive(18 : 30) and arrive(18 : 29), and these two are different, David’s assertion will weaken Charlie’s claim, with no consideration to the fact that 18:29 is “almost the same as” 18:30. Clearly, a fuzzy-logic (Klir and Yuan, 1996) based approach would be able to handle these situations in a more natural way.
5.6. Arguing about Contracts

Figure 5.5: Alice’s first utterance, establishing that the contract was violated because Bob arrived late. The evidence nodes in bold represent information obtained via a sensor probe, while the dotted lines represent negative inferences.
The problem of circularity can also arise in this scenario as \( \text{arrive}(18:30) \) attacks the claim of \( \text{arrive}(18:00) \) and vice-versa. One way to tackle this problem is shown in Figure 5.6. Here, the sensor probe action has been refined. Instead of simply reporting what time the juggler arrived, the sensor reports back on what the witness says. It is then possible to use the following argument scheme to go from what the witness says, to the \( \text{arriveAt}(\ldots) \) predicate:

- **Name**: \( \text{sayAccrual}_n \)
- **Premises**: \( \text{says}(S_1, \text{arriveAt}(A_1)) \ldots \text{says}(S_n, \text{arriveAt}(A_n)) \)
- **Conclusions**: \( \text{arriveAt}(A_1), \ldots, \text{arriveAt}(A_n) \)
- **Admissibility**: \( \forall i, 1 < i < n, \text{says}(S_i, \text{arriveAt}(A_n)) \neq \langle 0, 0, 1 \rangle \)
- **Mapping**: \( \forall i, j, 1 \leq i, j \leq n, \text{arriveAt}(A_i) = \text{says}(S_i, \text{arriveAt}(A_i)) \oplus \neg \text{says}(S_j, \text{arriveAt}(A_j)), \text{where } j \neq i \)
- **Undercuts**: \( \forall i, 0 < i < n, \text{sayAccrual}_i \)

This scheme represents one specific type of accrual, namely the accrual of witness testimony. The specificity of the accrual is a side effect of our argument scheme, we cannot represent the accrual using premises of the form \( \text{says}(X, Y) \). Instead, a precondition of the form \( \text{says}(X, \text{arriveAt}(Y)) \) has to be used so that arbitrary substitutions and accruals cannot occur. As usual, the fixed length nature of our argument schemes means that a different argument scheme must be used for different numbers of witnesses. This scheme takes in evidence from a witness, and uses this evidence to weaken all competing evidence. Similarly, any competing evidence is used to weaken the original witness’ testimony\(^5\).

By carefully crafting the mapping function, it is possible to introduce some form of “fuzziness” into the evidence; if witnesses claim times close to each other, they may mutually reinforce (or weaken each other less), while times that are far apart may weaken each other more.

The undercuts in this argument scheme are similar to those found in other accrual based argument schemes (Prakken, 2005b). Their purpose is to invalidate any form of the scheme that could be contained within a scheme performing a larger accrual.

One weakness of this approach is that it encapsulates the individual inferences from what the witness says to the witness’ claims into one big argument scheme. Thus, it is difficult to introduce an argument which would undercut the inference such as, for example, arguing that the witness is

\(^5\)This discussion glosses over a number of issues related to handling witness testimony, which is in fact still an open problem. For further details, refer to (Walton, 2002).
wrong because their watch has stopped. The easiest way to introduce such an undercut is to use it to attack the say(...) predicate, therefore not allowing it to form part of the accrual.

Since Bob is not going to attempt to show that he arrived at 18:00, but just attempt to prove that he did not arrive at 18:30, we will not further complicate the example by decomposing the sensor probes using the accrual argument scheme. Instead, we will assume that when Bob makes the utterance

\[
\{\{\}, \{\text{arriveTime}(18:30)\}, \text{sensorScheme}\}, \{\text{david(arriveTime(18:30))}\}\}
\]

David responds with an opinion for \text{arriveTime}(18 : 30) of \langle 0.4, 0.4, 0.2 \rangle. This means that \text{arriveTime}(18 : 30) is now associated with an opinion of \langle 0.61, 0.28, 0.11 \rangle. This opinion would carry through to the violation predicate, meaning that the argument has not shifted in Bob’s favour. This allows Alice to pass during her turn, making the utterance,

\[
\{\{\}, \{\}\}\]

Bob then probes a third person at the party (Eric).

\[
\{\{\}, \{\text{arriveTime}(18:30)\}, \text{sensorScheme}\}, \{\text{eric(arriveTime(18:30))}\}\}
\]

This probe returns an opinion of \langle 0.1, 0.8, 0.1 \rangle, meaning that \text{arriveTime}(18 : 30) falls below the threshold required for admissibility, rendering Alice’s claim invalid. Alice is not aware of any further sensors she may probe so as to sway the opinion regarding arrival time. If the dialogue terminates at this stage, Bob will have “won” the discussion, with both parties agreeing that the contract has not been violated.

While Alice could turn her attention to the second part of the clause, arguing that Bob has not juggled for a sufficient length of time, we will not pursue this example further, as we believe it has illustrated our framework in operation.

### 5.7 Discussion

Having introduced the contracting language and showing how it interacts with the argument framework so as to allow for contract monitoring and enforcement to take place, we may compare it to a number of existing frameworks. While a large number of contracting languages exist, many seem to ignore the problem of contract monitoring. Even those languages that do recognise the importance of contract monitoring often assume that the actual task of monitoring is easy, and instead focus on ways of detecting differences between what the environment should look like, as specified by the contract, and what the environment does look like (Grosof and Poon, 2004; Grosof, 2004; Milosevic and Drome, 2002; Xu and Jeusfeld, 2003), providing support for legal and auditing requirements (Gisler et al. (2000), or examining the formal semantics of normative concepts their interactions with contract violations (Governatori and Pham Hoang, 2005; Dignum et al., 2002). The work by Milosevic et al. (2002) and Daskalopulu and Maibaum (2001) (on which the former work appears to be based) is a notable exception to this trend. Both of these approaches make use of Subjective Logic to model the state of the world and determine the probability that
a contract violation has occurred. Unlike our approach however, their technique is centralised, and contains no dialogical aspect, instead representing a contract as a finite state machine, and attempting to reason which node we are in. Within our framework, such an approach could be represented by having evidential nodes stand for state transitions, and allowing these nodes to be probed via sensors. By associating state transitions with contract states, it would then be possible to determine which state the contract is in. Unlike our approach, the only way to introduce domain specific knowledge is via additional sensors. Also, by restricting themselves to only the $\oplus$ and $\otimes$ Subjective Logic operations, the types of interactions that they can represent are limited. Finally, no arguments about normative positions can be made using their approach. All that can be said is what contract state we should be in. It is up to the centralised arbitration mechanism to translate from this to determining which party is at fault, and to decide what actions should be taken to rectify the situation. Therefore, while containing some similarities, the approach to contract monitoring presented here is more powerful and general than the approach presented in Milosevic et al. (2002).

The argumentation based approach to contract monitoring has a number of notable features:

1. It is discretionary. Agents do not have to perform contract enforcement if they believe that the gain from such an action is outweighed by its cost.

2. Many approaches to contract monitoring contain a central authority which decides whether a contract has been violated. Our approach is decentralised, allowing agents to argue between themselves. This attribute is important when many agents inhabit the environment; a central authority might be overloaded if too many simultaneous contract enforcement actions occur at once.

3. An agent may back out of an enforcement action at any point. This may occur due to mounting costs, or due to the agent realising it was wrong. Allowing the process to stop at any time can provide additional computational savings.

4. It is general (via the argument schemes introduced), but domain specific knowledge is trivial to add via additional argument schemes.

5. Humans are very adept at following arguments. Presenting contract enforcement actions as dialogue provides a powerful, intuitive view into the inner workings of the agent system.

6. The approach is particularly well suited to domains where confidentiality is an issue. The heuristic may be adapted so that agents do not reveal certain pieces of information via their arguments at any cost, and sensor costs may be adjusted to reflect the sensitivity of certain bits of knowledge.

In some situations however, argument based contract monitoring is not appropriate. This is due to two main weaknesses. First, the approach is knowledge intensive. Many argument schemes must be included for a dialogue to take place. In complex domains, additional, domain specific argument schemes must be crafted, while “commonsense” argument schemes (and possibly sensors) must be created to reason about concepts such as arithmetic, time and action. Second, the approach is computationally intensive due to the exponential nature of the heuristic. This latter
weakness is however surmountable; improvements to the heuristic can be made. It could also be
argued that the features provided by the approach far outweigh these problems.

The example presented in this chapter was intended to highlight the application of argument-
ation to contract monitoring. We therefore ignored or glossed over many issues that a real
contracting framework would have to face such as issues of contract generation and negotiation (in
which argumentation could play a considerable role, see for example the work of Carbo gim and
Robertson (1999)), contract signing and storage, and contract termination. Additionally, we have
only examined a small number of argument schemes, dealing only with the normative aspects of
contract enforcement. It is easy to add additional argument schemes so as to allow agents to argue
about what actions should be taken to rectify any contract violations, argue about the penalties
that should be imposed, or perform decision making based on the state of the contract and the
environment.

The language for contracting described in this chapter has a number of limitations. For ex-
ample, its semantics are defined only in terms of predicates, with no description of how special
predicates such as obligations, violations and the like should actually influence agent behaviour.
Furthermore, no mention was made of contract level argument schemes (which would provide the
ability to make statements such as “this contract is currently in force”). Another weakness of the
language is that there seem to be many possible ways of translating from an English language
contract to a machine understandable contract. Some of these translations do however introduce
subtle problems. Possible enhancements to the language to overcome some of these issues will be
described in the next chapter.

5.8 Conclusions

In this chapter we introduced a language that can be used to represent a contract. The language was
designed with the intention of allowing agents to use it as a basis for arguing about the status of
a contract, and therefore, in appearance, it is similar to the way in which arguments are described
in the language of argument. However, the way in which variables are used in contracts, requiring
mixing with concrete terms, and often being referred to only in a conclusion, meant that the man-
ner in which unification takes place is different in the contracting language. Once variables are
eliminated from a contract clause, the clause may be used directly in the language of argument.

Introducing a clause into a dialogue is done via the probing of a no-cost contract sensor. Once a clause has been introduced, agents may draw conclusions from it by using a number of
contracting related argument schemes. The fixed length nature of the argument schemes, together
with the way in which variables are represented, means that a single argument scheme can often
not capture the reasoning procedure. In such cases, families of similar argument schemes may be
de fined and used, with each member of the family handling a contract clause containing a different
number of preconditions and conclusions.

Typically, arguing about a contract’s status involves determining which clauses are in effect
(which may be done by sensor probing and the introduction of domain specific arguments), after
which further arguments are introduced to determine what norms the contract imposes on the
parties. Finally, more arguments may be used to identify whether any of the norms have been
violated. Sometimes, these violations may be handled within the contract, via contrary-to-duty
5.8. Conclusions

obligations, and at other times, they may not, in which case the contract has clearly been violated, and additional (legal) argument must take place to determine penalties. Given this pattern of argument, most of the argument schemes described in this chapter dealt with the interactions between clauses, norms and violations.

As was seen in this chapter, the argument framework allows us to reason about the status of contracts and the environment in partially observable domains. Addressing this problem not only illustrates the power of our framework, but deals with an important real-world problem. While our example did not examine it in detail, the heuristics suggested for argument in Chapter 4 are used to drive the dialogue. A number of obvious, and not-so-obvious enhancements can be made to the argument and contracting frameworks so as to enrich their representational capability and their efficiency. In the next chapter, we examine such enhancements, as well as looking at possible ways in which the research described in this thesis can be continued.
Chapter 6

Conclusions

We begin this chapter by examining possible avenues of future work based on the research carried out in this thesis. Due to the thesis' broad nature, some of this work is general in form, attempting to address many issues argumentation researchers currently grapple with. Other parts of the future work are more concrete, focusing on aspects such as improving the link between argumentation and contracting.

After discussing future work, we examine how the research that we have carried out answers the research questions posed in Section 1.3. We then close the thesis by summarising the work contained herein and describing our conclusions.

6.1 Future Work

This work presented in this thesis seems to pose as many questions as it answers. In this section, we examine possible avenues for future research. While we attempt to group related areas together, some of the questions posed cut across a number of areas.

6.1.1 The Abstract Argument Framework

Criticism has been levelled at many models of argument containing the notion of support. Within the abstract argument framework presented in Chapter 3, the notion of support is based on the existence of *prima facie* evidence for the argument. This does overcome some of the reasons against having support within the framework, but a detailed investigation regarding support in arguments may yield more justification for the concept. Apart from the representational enhancements our representation of support yields, we would argue that the inclusion of support aids readability and provides a more compact representation of arguments than could be found in a purely attack based argument framework. It would be ideal if a class of arguments were identified where inconsistencies would arise when they were represented using standard semantics, but which would be easily representable when support is present.

One of the greatest challenges facing argumentation theory today involves finding an acceptable way of representing accruals. While many attempts have been made, all approaches seem to be found wanting. Verheij's abstract CumuLA framework was designed to represent accruals, but does so via defeat rather than support relations. The concept of arguments supporting one another is an integral part of many accruals, it would be interesting to see if the abstract framework could be extended to provide a new, enhanced accrual representation. Of course, since a major component of an accrual is its effect on the strength of an argument, this may be of limited use, unless some abstract way of representing argument strength is also included in the model.
6.1. Future Work

The three step process of argument was briefly mentioned. To recap, agents exchange arguments with each other, after which valuations are assigned to these arguments. Finally, depending on the semantics of the framework, a set, or sets, of acceptable arguments are selected. The separation of these steps is somewhat troubling: while it is clear that weighting affects the acceptability of arguments, in some cases, namely where loops exist, argument acceptability should affect argument weight. For example, considered the situation shown in Figure 6.1.

![Figure 6.1: A sample set of arguments showing why valuations and admissibility computations are interdependent.](image)

Here, the initial lack of evidence for argument $a$ will end up strengthening the argument at a later stage (as it does not attack $b$). In such a situation, it may make more sense to compute acceptability before computing weightings. Other situations may require that the calculation of acceptability and argument strength be interleaved. The recent work of Cayrol and Lagasquie-Schiex (2005a) appears to have begun investigating this issue, but much work on introducing strength into abstract argument frameworks remains to be done.

6.1.2 The Concrete Argumentation Framework

The Logical and Dialectic Layers

Many argument frameworks (e.g. (Pollock, 1995; Prakken and Sartor, 1996; Besnard and Hunter, 2001)) are, in a sense, self referential. That is, an argument within a system described by the framework is able to refer to other arguments within the system, possibly creating new arguments, or changing an argument’s structure. Since argument schemes contain instantiated, but ungrounded types, it is impossible to represent them as literals, meaning that the argument system introduced in this thesis is not self-referential. As was seen in the contracting chapter, overcoming this weakness would be beneficial in many situations, giving the ability to represent arguments that the framework cannot currently handle, and simplifying the representation of some types of arguments.

At the moment, argument schemes have a fixed number of premises and conclusions. Extending the schemes to allow these to take on any number of elements (perhaps via accepting a pattern) should not be particularly difficult, and would greatly enhance the approach’s representational capability.

A third enhancement involves the variables found in argument schemes. By allowing variables to represent parameters and predicates, and permitting the unification process to bind certain variables to only specific predicates, it will be possible to create more powerful argument schemes. Argument schemes such as the `sayAccrual` scheme encountered in the previous chapter would then be able to deal with a variety of predicates, as opposed to the `arriveAt` predicate as shown
in the previous chapter.

The integration between Chapter 3’s abstract argument framework and the concrete argument framework is somewhat tenuous at the moment. It is assumed that, as per Amgoud’s model, the concrete framework would allow us to assign weights to various arguments. By including the support relation, and adding attacks between mutually exclusive evidence nodes, we could then compute which arguments can be combined in admissible sets. However, as mentioned in the previous section, there should be a better way of integrating the two frameworks, and this should be investigated.

Perhaps the most restrictive aspect of the concrete framework lies in its inability to handle loops. Cayrol and Lagasquie-Schiex (2005a) has suggested a novel method of weighing up arguments in systems where loops exist, and it may be possible to apply their approach to our framework. Alternatively, a detailed analysis of the properties of loops may lead to some insights allowing for their use in the framework.

Currently, the framework supports a limited subclass of accruals. Extending this support to represent all three kinds of accruals is a challenging, but promising direction for future work.

Jøsang has defined a large number of operators within Subjective Logic. Many of these seem to map to specific argument schemes (e.g. Jøsang et al. (2005) presents a simple deductive inference scheme), examining whether some argument scheme mapping functions can be used to define some new, standard, SL operators may be worth pursuing.

Finally, creating a domain independent library of argument schemes, in the spirit of (Walton, 1996) would allow agents to reason and argue about a vast range of subjects. Enhancing this library with schemes applicable to common domains of discourse, such as time, knowledge and obligations will further enhance its usefulness. Assuming some sort of reflective capability is added to the framework, it would be interesting to see what argument schemes can be created that themselves deal with argument schemes and arguments.

The Procedural and Heuristic Layers

The dialogue game used as the basis of inter-agent argument is very simple, containing only one type of move. Enhancing it to with additional utterances, as well as using a commitment store rather than a public knowledge base, will allow for more complex dialogues to take place between agents.

The interplay between sensors and arguments is an area in which little formal work has been done (Oren et al., 2007b). While our model is very simple, it elegantly captures the fact that sensor data is inherently unreliable in many situations. Enriching the model of sensors will increase the applicability of this work.

Currently, the heuristic agents use to decide which utterance to make requires them to evaluate all their possible utterances. This means that the heuristic has an exponential computational cost. However, simple optimisations and further tuning of the heuristic would allow us to reduce this cost. One simple example would involve a cut-off regarding the maximum length of an utterance (this is based on the claim that a longer utterance would probably require more sensors to be probed and thus would, on average, be more expensive than a shorter argument). Additional pruning of arguments could further reduce the computation required. Our heuristic is equivalent to one step lookahead; the introduction of opponent modelling could allow further lookahead to
Future Work

Machine learning techniques may also let us enhance the heuristic further. Additional types of heuristics (Oren et al., 2006b), as well as the use of rhetoric are other areas which could be examined.

6.1.3 Contracting

The contracting framework presented in Chapter 4 was intended only to illustrate how argumentation may be applied to contract monitoring. However, it can form the basis of a viable and powerful approach to contracting if extended, and we begin by proposing and examining a few such extensions.

The first area of possible enhancement within the contracting framework is the contracting language itself. The current contracting language was designed to allow for the translation of contract clauses into the argument language. Many, specialist contract language exist, and several authors have attempted to identify the basic requirements for such a language (Neal et al., 2003; Oren et al., 2005). The language in its current form provides limited support for the concept of obligations and permissions, together with the ability to support violations and contrary to duty obligations. This support is however syntactic only, with the relations between these, and other concepts being defined according to the contract related argument schemes. Support for other concepts such as time intervals, the notion of actions, and roles is also based on the argument schemes used to argue about the contract. One possible piece of future work involves extending and formalising the contracting language to support these, and other concepts required for a general contract language. As was seen in the previous chapter, the act of translating from a natural language contract to a formal one is somewhat arbitrary, and many subtle errors can be introduced. Research is already being undertaken regarding the translation process (e.g. (Governatori, 2005)) and formalising such a procedure for our language would be interesting.

WS-Agreement, with its RuleML bindings is fast becoming the de-facto standard language for the representation of contracts. Other, competing languages do however exist, with varying degrees of formality. Given this, another possible avenue of research would involve creating a mapping from one or more of these contract languages to our contracting language, or alternatively a similar language that can be used as the basis for a dialogue in our argument framework. Depending on the nature of the mapping, agents could then undertake a contract enforcement dialogue, or, with different argument schemes, undertake a dialogue about some other aspect of the contract life-cycle, such as contract negotiation.

In the context of contracting, the largest area of opportunity for future research efforts is probably the investigation of additional argument schemes. Even if the contracting language is formalised with regards to concepts such as norms, roles and actions, agents will still have to reason and argue about these concepts, requiring that the appropriate argument schemes be created. As discussed in Chapter 5, the argument schemes we have described are by no means the only way to represent the interactions between contracts, norms and violations, and a detailed examination of whether the proposed mapping functions are indeed appropriate in all situations would be useful. Furthermore, our schemes are by no means exhaustive. For example, it would be useful to introduce some general schemes dealing with normative concepts such as permissions, as well reasoning about how different norms (such as prohibitions and interactions) may interact.

At the moment, we only argue about the status of clauses. Since it is possible to represent an
entire contract using a predicate of the form

$$\text{contract}(\text{clause}_1(\ldots), \ldots, \text{clause}_n(\ldots))$$

It should be relatively easy to introduce argument schemes to deal with the status of a contract as a whole. Similarly, contractual notions such as contract termination must still be dealt with.

Much work has been done on legal reasoning using argumentation (e.g. (Prakken, 2005a)). Since contracts are legal instruments, the integration of laws and legal reasoning into the framework is an area of research which should be pursued. Very often, legal discussions about a contract does not involve the contract’s content, but instead focus on whether a legally binding contract has come into effect. The ability to have agents argue about these types of issues may therefore be useful, especially if they exist within some legal authority (for example, as part of a human-agent team). Similarly, dialogue regarding liability may very quickly descend into matters of law, and extending the framework, via additional argument schemes, to deal with such cases would be useful and interesting.

If agents know they are to be arguing about the status of a contract, it should be possible, in conjunction with the use of unique argument schemes, to introduce heuristics specific to the domain. For example, if agents know that a certain state of affairs was reached, and that one of a number of possible states were to follow, they do not need to reason about the possibility of other, unrelated states being entered into, and may focus their resources into identifying which of the possible valid states was actually the case. In such cases, it may be useful to treat the contract as a workflow and transform it into a graph before reasoning which nodes (states) are connected to the node which the contract was known to have visited. Daskalopulu and Maibaum (2001) hints at, but does not pursue, such an approach.

## 6.2 Validating the Research Hypothesis

Our research hypothesis stated that

*A formal model of evidential reasoning can be produced that will allow agents to undertake inference in complex, partially observable domains.*

Within this thesis, we have shown two broad approaches to creating such a model. The first was the abstract framework of argument. As shown by other authors, it is possible to embed a concrete argumentation within such an abstract framework (Bondarenko et al., 1997; Prakken, 2005b), and then make use of the abstract framework’s semantics to compute sets of acceptable arguments. The main contribution of the abstract framework is its ability to reason about evidence in the form of support from the empty set. Such supported arguments form the basis of evidential reasoning.

The second approach to evidential reasoning was built around the concrete argument framework. Here, we presented a complete system that agents may use to reason within partially observable domains, based on evidence from unreliable and fallible sensors. This approach consisted of the low level logic of the system, based on Subjective Logic, together with a description of how arguments may be presented and how they interact with each other. We then presented a dialogue game which agents may use to undertake argument with each other, and suggested heuristics that
they could use to decide which utterance to advance during their turn.

The domain in which we evaluated the concrete framework was the contracting domain. We showed how agents may undertake a contract monitoring dialogue in situations where only some of the environmental state is available to them.

Given the work of this thesis, we claim that it is possible to answer our research hypothesis positively.

6.3 Conclusions

In this thesis we presented frameworks for argument aimed at supporting concepts useful for evidential reasoning. While the applicability of such reasoning in some domains, such as law, is obvious, we claim that its use extends much further, touching most aspects of reasoning. In particular, we focus on the notion of evidence as a tool to determine environment state within partially observable domains. Here, an agent must be able to draw inferences from whatever evidence it has available to it, and the notion of evidence supporting a certain set of conclusions becomes important.

Other argument based frameworks providing an explicit representation of support have been proposed. However, much criticism has been levelled at such frameworks as it was unclear what additional representational power they provided. In this thesis we introduced a new type of argument framework and associated a new set of evidential semantics to this framework. Within this framework, we were able to distinguish between prima facie and derived arguments based on the notion of support, and showed how the semantics of argument in such a framework agreed with our intuitions in situations where existing argument frameworks produce counter-intuitive results. An agent could use our extensions to determine what sets of arguments are, in a sense, self consistent given that certain arguments support, and attack other arguments. Many applications of argumentation theory in other fields depend on this ability to show consistency of argument.

While clearly both important and useful, our argument framework was abstract in nature. To be implemented, it would have to be populated with some sort of logic, and, depending on its purpose, additional scaffolding might be required. In Chapter 4, we introduced a concrete argument framework for reasoning with evidence. The level of support for an argument in many domains is often highly variable. This means that the level of certainty associated with an argument’s conclusions (and any evidence within the domain) may vary greatly. To represent this notion of argument strength, we based our concrete framework on Subjective Logic. Unlike probabilistic strength representations, the use of Subjective Logic allowed us to model both an argument’s strength, and its level of certainty. In many situations, particularly when dealing with evidence, it is useful to be able to distinguish between these two concepts (for example, one would rather know whether something holds with a high level of certainty, rather than believe something but be uncertain as to how likely that belief is to actually hold).

Our goal when creating the concrete framework was to allow an agent to perform evidence based reasoning. To do so, an agent must often not only associate an opinion with an argument, but must reason about which arguments are relevant, and what evidence it should gather so as to be able to reach a conclusion. To perform such reasoning, an agent could engage in a deliberative dialogue with itself, advancing and criticising its own arguments, and searching for evidence to
support its claims. Clearly, engaging in dialogue is also useful when attempting to convince others of its viewpoints. We described such a dialogue in the form of a dialogue game. Within this game, an agent could make utterances consisting of both arguments for a certain conclusion, and “sensor probes” used to gather evidence in support of the arguments. The interaction between Subjective Logic and the dialogue, as well as our representation of argument schemes, allowed us to represent many facets of argument including default arguments, burden of proof, and some types of accruals of argument.

In our model, sensors are an abstraction representing different ways of obtaining information regarding environment state. They may be fallible (or even malicious), and are associated with an opinion representing their reliability and accuracy. This notion of sensors, and the way an agent may interact with them, is both simple and powerful, allowing one to easily model many different information gathering systems.

Apart from describing how an agent may reason about arguments (the logical and dialectical layers), and how an agent may advance an argument for consideration (the procedural layer), we proposed some heuristics allowing an agent to select which arguments to advance, and what evidence to gather (via sensor probes), in an attempt to prove a certain set of goals. Armed with these heuristics, an agent is able to engage in internal deliberation or in dialogue with another agent, arguing why the environment is most likely within a certain state. The ability to reason with evidence in this way is applicable in almost every aspect of multi-agent systems.

When reasoning about the state of a contract, an agent must often take uncertain evidence into account, and convince others that a certain state of affairs holds. Very little work has been done on evidence based contract monitoring, and this important problem was thus chosen to showcase the power of our approach. As was shown in the previous chapter, by defining a simple contracting language, we were easily able to leverage our framework so as to allow agents to argue about the state of a contract. Our argument based approach is more flexible than existing approaches, and can handle novel situations more easily than probabilistic approaches. However, it does require more detail to function.

Our research proposed both abstract and concrete models addressing the problem of how to argue with evidence. Unlike other work, our approach advanced a unified model, dealing with issues arising at the logical, dialectic, procedural and heuristic levels. While it is able to represent many of the forms of evidential reasoning, additional challenges remain, which we hope to address in future work.
Bibliography


