Dynamics in Formal Argumentation

Daniela Vasconcelos Carbogim

Ph.D.
University of Edinburgh
2000

To Zé
To Bosco and Leinad
To Caio

constant sources of inspiration,
motivation,
courage,
and of a humanistic attitude towards life

Abstract

In this thesis we are concerned with the role of formal argumentation in artificial intelligence, in particular in the field of knowledge engineering. The intuition behind argumentation is that one can reason with imperfect information by constructing and weighing up arguments intended to give support in favour or against alternative conclusions. In dynamic argumentation, such arguments may be revised and strengthened in order to increase or decrease the acceptability of controversial positions.

This thesis studies the theory, architecture, development and applications of formal argumentation systems from the procedural perspective of actually generating argumentation processes. First, the types of problems that can be tackled via the argumentation paradigm in knowledge engineering are characterised. Second, an abstract formal framework for dynamic argumentation is proposed, based on an analysis of dynamic aspects of informal argumentation. Formal arguments in this framework are built from an underlying set of axioms, represented here as executable logic programs. Finally, an architecture for dynamic argumentation systems is defined, and domain-specific applications are systematically instantiated from this formalisation. Relevant applications are presented within different domains, thus grounding problems with very distinctive characteristics into a similar source in argumentation.

The methods and definitions described in this thesis have been assessed on various bases, including the reconstruction of informal arguments and of arguments captured by existing formalisms, the relation between our framework and these formalisms, and examples of dynamic argumentation applications in the safety-engineering and multi-agent domains.

Acknowledgements

Someone once told me that writing a thesis can be an enjoyable experience. Well, I would like to second that opinion and gratefully acknowledge all the people who somehow influenced the work described here, helping to make this period a most pleasant one.

First and foremost, my biggest thanks to the best supervisors one could ask for, Dave Robertson and John Lee. Their time, continuous support and precise advice were crucial. Dave has influenced my work in so many aspects that I find it hard to enumerate them all here; in particular, his questions and comments have made me think a lot about my own research, and his sharp sense of humour was a big plus. John was an invaluable source of motivation; his careful comments and constant presence have helped me in directing my project and has given an interdisciplinary flavour to this work.

Special thanks go to my examiners, Corin Gurr and Simon Parsons, for their comments, suggestions and for the most rewarding experience which was my Viva.

I am most grateful to current and past members and co-workers of the Software Systems and Processes (SSP) Group—Jaume Agusti, Virginia Brilhante, Alberto Castro, João Cavalcanti, Luigi Ceccaroni, Jessica Chen-Burger, Flávio Corrêa da Silva, Stefan Daume, Peter Funk, Yannis Kalfoglou, Renaud Lecoeuche, Siu-Wai Leung, Chris Lin, Edjard Mota, Steve Polyak, Dave Robertson, Jon Tonberg, Wamberto Vasconcelos and Chris Walton. These people make the SSP Group the perfect environment for research and collaboration (and thanks to João we have a SSP t-shirt too!). The SSP weekly meetings were an invaluable opportunity for communication and feedback on the various stages of my research. I also had the pleasure of organising these meetings for over two years, so thanks to all of you who have kindly volunteered (or been volunteered) to give a talk on a Wednesday morning.

Big thanks also to Elias Biris, Marcio Brandão, Marco Aurélio Carvalho, Herman Gomes, Patricia Machado, Manuel Marques Pita, Sonia Schulenburg, Josh Singer and Gerhard Wickler. They too deserve a lot of credit for all the help on various matters and particularly for coping with my continuous talking for the past three years. Their patience and disposition is warmly acknowledged—especially Virginia, João, Alberto and Yannis.

I would like to acknowledge all the participants of the Symposium on Argument and Computation, with special thanks to Chris Reed and Tim Norman for organising such an inspirational and stimulating event. A big thanks to the members of my group, Erik Krabbe, Tim Norman and Doug Walton, and also to Peter McBurney.

A hearty thank-you is due to two great women who played a big role in my work and life, Jane Hillston and Virginia Brilhante. Jane as my mentor has helped me to realise a lot of my potential. Thank you both for your serenity, for the long and fruitful talks, and for sharing with me your experiences and expectations.

Thanks to Flávio Corrêa da Silva, my MSc supervisor in Brazil and the main person responsible for me coming to Edinburgh in the first place. Flávio himself was a PhD student at the former AI Department, in fact the first one to be supervised by Dave. A big thank-you also to Wamberto Vasconcelos for all his tips, advice, time and knowledge from the early start.

More generally, I thank the staff of the former AI Department and now Division of Informatics—particularly Janet Lee, Olga Franks, Jean Bunten, Deirdre Burke, Jane Rankin, Michelle Siszczuk, Neil Brown, Craig Strachan, Gordon Reid and John Berry—for the enjoyable and informative chats, and for being so helpful in every possible way. A special acknowledgement goes to Olga for her commitment in providing a most complete service which is the AI Library. Thanks are also due to the Informatics Graduate School for support provided through the Student Travel Grant, and to the Faculty of Science and Engineering for running valuable programmes such as the Transferable Skills Programme and the Science and Engineering Mentoring and Springboard Programme.

For its direct support I would like to acknowledge the Brazilian National Research Council (Conselho Nacional de Desenvolvimento Científico e Tecnológico, CNPq), under grant no. 200074/97-0. Eli Ribeiro and Nelson Prugner in particular have been extraordinarily helpful and efficient.

Finally, a most heartfelt and loving thank-you to my other half, Zé, who has given up so much unselfishly, coming with me to Edinburgh, helping me with everything, reading my drafts, making the right questions. Thanks for your love, and for being there all the time.

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Daniela Vasconcelos Carbogim
Edinburgh
October 20, 2000

Contents

List of Figures

List of Definitions

Part I

Background and Overview

Chapter 1

Context and Motivation

This thesis is concerned with the role of formal argumentation in knowledge engineering. Our motivation is that research into argumentation can provide methods and techniques for tackling the sorts of *wicked problems* that are common in this field, problems which according to Rittel and Webber (1973) have no definitive and correct solutions because criteria for success are often subjective and conflicting.

The intuition behind argumentation is that one can reason with imperfect information and deal with such wicked problems by constructing and weighing up arguments relevant to alternative conclusions. In a recent survey (Carbogim et al. 2000b), we have identified four types of problems in knowledge engineering that have been tackled by argument-based approaches:

- the problem of defeasibility in a knowledge base, where some conclusions might be withdrawn in the presence of new knowledge;
- the problem of decision making based on uncertain knowledge, where we have to decide which alternative to select;
- the problem of negotiation, where autonomous agents communicate and reason about propositions in order to reach an agreement; and
- the problem of design, where it is important to make decisions, communicate decisions and argue that the resulting artifact represents an acceptable solution to a particular problem.

An analysis of the state of the art in argumentation research shows that there are as yet few clear guides to standard practice in this area, and although argumentation gives a generic architecture for a particular style of reasoning, much domain-specific expertise is required to instantiate this architecture to a domain of application. Since argumentation, in automated forms, is relatively new there do not yet exist methods for guiding application of architectures to problems, and the focus has been on more abstract argumentation theory. In many cases specialised solutions have been adopted in order to implement practical systems from theoretical frameworks, and systems have been mostly evaluated in terms of simple benchmark problems.

This present state of affairs reflects an expected direction of development in argument-oriented research in knowledge engineering, summarised in the following two (related) points:

- there is a need for increasing the practical utility of argumentation systems in knowledge engineering by taking more complex arguments into account; and
- there is a need for clear methodologies for the systematic development of systems for argument generation in specific domains.

This thesis looks at both issues.

1.1 Formal Argumentation and Reasoning

One of the assumptions underlying the use of classical methods for representation and reasoning is that the information available is complete, certain and consistent. But often this is not the case. In almost every domain, there will be beliefs that are not categorical; rules that are incomplete, with unknown or implicit conditions; and conclusions that are contradictory. Therefore, we need alternative knowledge representation techniques for dealing with the problem of imperfect information.

There are two reactions to this sort of problem when designing systems. The first is to resolve conflict and restore consistency, as for instance in most research in belief revision. A second view, however, suggests that inconsistency can offer insights into rational processes and therefore should not be eradicated. Argumentation as a reasoning technique is an example of the latter, through which we can construct and compare arguments in order to reach and justify decisions.

Argumentation bears a strong resemblance to certain approaches for inconsistency management, in particular to truth maintenance systems (Doyle 1979). The difference is more about a shift in emphasis than it is technical. Truth maintenance systems keep track of the reasons for deriving conclusions from a knowledge base, so they can deal with conflict by trying to explain *why* it happened. If a belief needs to be retracted (e.g. to restore consistency), truth maintenance systems can identify which are the conclusions that depend on this belief that should also be retracted. On the other hand, in argumentation it is important to make the sources of inconsistency clearer, and also to chart the course of an argument, so we can reason methodically in the face of conflict.

Formal argumentation theories are characterised by representing precisely some features of (informal) argumentation via formal languages and by applying formal inference techniques to these. Although such systems can be of different nature and have distinct aims, the notion of argument adopted by them is usually the same, corresponding to that of logical proof. In fact, the difference between *formal argument* and *logical proof* is not syntactic, but pragmatic in the sense that proofs are certain and arguments can be defeated by or preferred over others. As remarked by Krause et al. (1995), "arguments have the form of logical proof, but they do not have the force of logical proof."

Despite the traditional interest in argumentation in many disciplines, computational frameworks for representing moderately complex arguments have appeared on the scene only recently. Some believe that formal argumentation has many disadvantages, because the study of formal logic can require a great deal of effort (van Eemeren et al. 1987) and its use to model real (natural language) arguments is too restrictive (Reed 1997). However, formal models of argumentation can be applied successfully as a reasoning method in certain contexts, especially if used in a lightweight manner by applying logic to specific parts of a problem in a focused and selective way (Robertson and Agustí 1999). Recent efforts in bringing the communities of philosophy and artificial intelligence together have also resulted in a handbook (Norman and Reed 2000) for identifying problems, issues and a roadmap for research in the interdisciplinary field of argument and computation.[1]

1.1.1 Truth and Acceptability

What is interesting about argumentation is that it explores aspects of practical reasoning that are not always addressed by conventional reasoning theories. For instance, it is based on the notion of acceptability—a proposition is acceptable on the basis of the arguments that are relevant to it. As argued by Prakken and Vreeswijk (1999):

> Argumentation systems are not concerned with the truth of propositions, but with justification of accepting a proposition as true.

Note that this view had already been advocated by Doyle (1979, p.234):

> To say that some attitude (such as belief, desire, intent, or action) is rational is to say that there is some acceptable reason for holding that attitude. Rational thought is the process of finding such acceptable reasons. [...] One consequence of this view is that to study rational thought, we should study justified belief or reasoned argument, and ignore questions of truth.

Being a constructive process for finding acceptable reasons, argumentation is essentially dynamic in nature (Gabbay 1999, 2000), and also intrinsically non-monotonic because a position may be warranted with respect to certain premises but not if other related arguments are also considered. Note that argument processes rely mostly on *conflict* and *disagreement* hence it is important to deal with these types of inconsistency properly. Again, moving away from the notion of truth to that of acceptability gives a way for doing this.

1.2 The Way We View Arguments

The study of argument is traditional in many disciplines, and although the notion of argumentation is common to most of us there is still no consensus as to the correct

[1] *"Call it computational theory of argumentation, or argument-based artificial intelligence (or both)."*— David Hitchcock, e-mail posting to the ARGTHRY list on 3 August 2000.

meaning of the term (Gilbert 1995). The following tries to summarise the ubiquitous character of informal argumentation.

> Argumentation is a verbal and social activity of reason aimed at increasing (or decreasing) the acceptability of a controversial standpoint for the listener or reader, by putting forward a constellation of propositions intended to justify (or refute) the standpoint before a rational judge. (van Eemeren et al. 1996, p. 5)

Note that this definition encompasses two views of an argument:

- a local, static view, in which an argument is intended to give support in favour or against a conclusion; and
- a global, dynamic view, in which an argument is intended to increase or decrease the acceptability of controversial positions.

Most existing formalisms are limited in scope because they describe the shape of an argument but not the mechanisms needed to give dynamics to it. Such formalisms are often characterised as two-step processes in which arguments are first generated and then evaluated in terms of their acceptability. The dynamic counterpart of argumentation is restricted to determining whether an argument is acceptable based on its relations to all existing arguments. This may be defined in dialectical terms via dialogues and debates, but is still a limited view of dynamics because it does not allow arguments to be revised or strengthened in order to change their acceptability with respect to certain positions.

Mechanisms for capturing dynamics involve revising arguments that have been attacked in order to reestablish their validity; and also strengthening arguments by anticipating criticisms and dismissing them. This thesis focuses on whether such mechanisms can be formalised and automated and how argumentation seen from this dynamic perspective can provide an answer to the two research issues stated above. Our position is summarised below:

- Argument dynamics broadens the scope of argument-based applications in the knowledge engineering domain by grounding various problems with very distinctive characteristics into a similar source.

- Certain types of argument dynamics can be formalised and provide a generic methodology supporting the design of domain-specific argument systems in a systematic way.

Although this view of dynamics has not been much explored in the context of formal argumentation, it is a legitimate part of the study of arguments and informal logic. Arguments are based on reasons and assumptions which are not necessarily acknowledged by others, and which can therefore be challenged. Studies in argument analysis include the use of techniques for strengthening an argument so as to reduce chances of attacks and to eliminate the demand for yet more reasons and justifications. Fogelin and Sinnott-Armstrong (1997, p. 40) have identified three such techniques:

Assuring an argument by stating that backup reasons exist, although they are not explicitly presented.

Guarding an argument by weakening the argument claim, thus protecting it from certain attacks.

Discounting an argument by anticipating criticisms and dismissing them.

Among these strategies, we are mostly interested in that of *discounting*, i.e. in ways of considering potential attacks and dismissing them. According to Fogelin and Sinnott-Armstrong (1997), "the general pattern of discounting is to cite a possible criticism in order to reject it" by indicating that the current position is more important than this criticism. We are also concerned with cases in which criticisms can be more important. And to dismiss such criticisms, the argument under attack might need to be restructured: some premises on which it is based may be reviewed, and new ones may be put forward.

1.3 General Questions Addressed in this Thesis

This thesis is about generating arguments. It is a study of theory, architecture and development of formal argumentation systems in the context of knowledge engineering from a computational and procedural perspective. The central contribution is that it is possible to construct an abstract formal framework for argument dynamics, and to systematically instantiate domain-specific applications from this formalisation.

The work in this thesis has been guided by two main, general questions, namely:

- How can knowledge engineers benefit from argumentation-based approaches to knowledge representation and reasoning?
- How can we improve the methodology for building systems for supporting such tasks?

More specific questions are stated in the next section, after we define in more detail the problem of formalising and automating argument dynamics. Before, though, we delineate the structure of the present thesis.

1.3.1 Thesis Overview

The remainder of this thesis is divided as follows:

Part I. In Chapter 2 we identify and define precisely the problem to be addressed in this thesis. Then, in Chapter 3, we characterise the types of problems that can be tackled via the argumentation paradigm in knowledge engineering.

Part II. Chapter 4 introduces the formal concepts underlying our approach, and identifies the subproblems that need to be addressed in order to formalise and automate dynamic argumentation. The rest of the chapters in this part then address these subproblems: Chapter 5 gives an intuitive description of our approach in terms of informal examples and of concepts from informal argumentation theory; then, Chapter 7 introduces the corresponding formal description based on a precise characterisation of possible attacks given in Chapter 6; Chapter 8 gives a worked example illustrating the use of two possible implementations for a dynamic argumentation mechanism; and finally, roles and properties of our theory are discussed in Chapter 9.

Part III. This part is about adapting our abstract theory of dynamic argumentation to domain-specific applications. We do this in Chapter 10 by proposing a generic architecture for argumentation systems which elaborates on the mechanisms defined in Part II. Two areas of application are considered: safety-engineering in Chapter 11, and negotiation in Chapter 12.

Part IV. In Chapter 13 we summarise our contributions, and finally, in Chapter 14, we discuss possible directions and avenues for future work.

Chapter 2

Problem Definition: Dynamic Argumentation

From a procedural perspective, *formal argumentation* is about capturing processes of argument exchange by means of formal languages and inference techniques. Such arguments are often represented by means of logical proofs, generated from an underlying knowledge base—usually composed of facts and rules—via a provability relation. And although argumentation processes can be of different natures and have distinct aims, they are often based on conflict and disagreement between arguments.

Argumentation is sometimes used for *determining* whether a conclusion is acceptable with respect to a static knowledge base (or a set of knowledge bases) assumed to be fixed over time. Note that here *time* does not necessarily correspond to real time, but rather it is related to the sequence of argument moves. Thus, the knowledge base—and consequently the set of all arguments that can be derived from it—remain unchanged as the argumentation develops. Most conventional formal argumentation systems describe only this type of process for organising the relevant arguments (possibly in a dialectical style) in order to specify if a conclusion can successfully defend itself from attacks. Examples are given in Sections 3.1 and 3.2. In this work, however, we are interested in argumentation processes that do account for changes to the underlying knowledge base. We refer to these as *dynamic*.

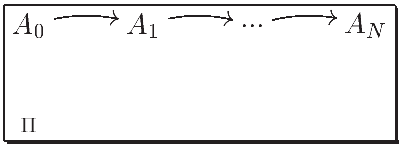
Changes to a knowledge base can be of two broad types: those independent from the argumentation, and those related to it. The first type is said to be *external* in the sense that changes are caused by some outside, not necessarily known, factor. Such changes happen over time, but independently from the sequence of argument moves. Dynamic argumentation systems that account for *external changes* are used to *determine* whether certain conclusions are acceptable given that the available information can change during the argumentation. These are briefly discussed in Section 3.1.2.

The second type of change is said to be *guided* by argumentation, in the sense that changes can allow desired arguments to be generated and undesired arguments to be blocked. These are intrinsically related to the sequence of argument moves—we can deliberately try to increase or decrease the acceptability status of a position by performing changes so as to introduce supporting or attacking arguments, respectively. Therefore, dynamic argumentation systems that account for *guided changes* can be used not only to *determine* if a conclusion is acceptable with respect to a knowledge base, but also to *affect* its acceptability status by performing certain changes to this knowledge base during the argumentation. Examples of such processes are presented later in this chapter, in Section 2.1.

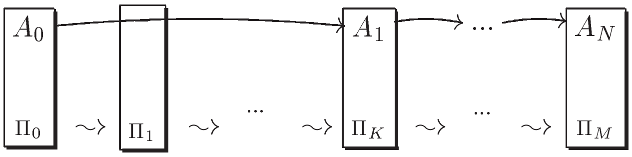
In brief, the nature and purpose of each type of argumentation process can be rather different. Figure 2.1 illustrates the different sorts of processes with respect to the changes allowed. Below we summarise the general concept of dynamic argumentation.

Dynamic argumentation *is about using formal languages and inference techniques for capturing processes of argument exchange where the knowledge base from which arguments are derived is dynamic, i.e. it can be changed during the argumentation process, either via external changes or via guided changes.*

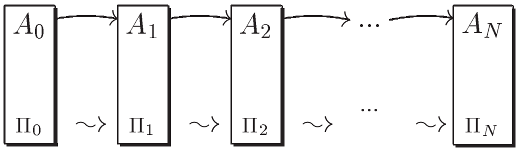
In this thesis we are interested in formally describing dynamic argumentation processes based on guided changes. From now on we refer to these by *dynamic argumentation* or *argument dynamics*, unless there is a risk of ambiguity. We also use the term *revision* to refer to any sort of change to the knowledge base.



(a) Conventional (static) argumentation: argumentation steps assume fixed knowledge base.



(b) Dynamic argumentation with *external* changes: knowledge base may change independently of argumentation step.



(c) Dynamic argumentation with *guided* changes: knowledge base changes as a consequence of argumentation steps.

Figure 2.1: Types of argumentation according to changes in the underlying knowledge base, symbolised here by the possibly indexed letter Π. $A_0, A_1, A_2, ...$ represents the sequence of argument moves, while $\Pi_0, \Pi_1, \Pi_2, ...$ stands for the sequence of knowledge bases obtained as changes (expressed by $\leadsto$) are performed.

2.1 Examples of Dynamic Arguments

One way to think about argument dynamics is that it should be possible to change and revise an argument in order to defend it from attacks. In formal systems, where arguments are derived from a knowledge base, it should be possible to revise this knowledge base so as to defend arguments from attacks, e.g. by adding new information so that new supporting arguments or counter attacks can be derived. From this perspective, dynamic argumentation is a process of knowledge base revision guided by attacks and counter attacks, which is intended to increase—rather than just determine—the acceptability status of a position with respect to this knowledge base.

Our view is that argumentation seen from a dynamic perspective has a broader role in computational systems. This section gives some scenarios in which formalising and automating the kind of dynamic arguments above could be useful, and it turns out that these are applicable also in domains far removed from the roots of argumentation theory—for instance in describing relationships between fault trees and system models in examples taken from the safety-engineering community.

2.1.1 Model Design

Argumentation can play an important role in design and analysis, especially in safety-critical domains, where safety arguments are normally intended to convince people that the specified system will be safe if implemented appropriately.

Consider for example a system that models the operation of the pressure tank control system in Figure 2.2, as defined in the Fault Tree Handbook (Vesely et al. 1981):

> The pump pumps fluid from an infinitely large reservoir into the tank. We shall assume that it takes 60 seconds to pressurize the tank. The pressure switch has contacts which are closed when the tank is empty. When the threshold pressure has been reached, the pressure switch contacts open, deenergizing the coil of relay K2 so that relay K2 contacts open, removing power from the pump, causing the motor to cease operation. The tank is fitted with an outlet valve that drains the entire tank in an essentially negligible time. [...] When the tank is empty, the pressure

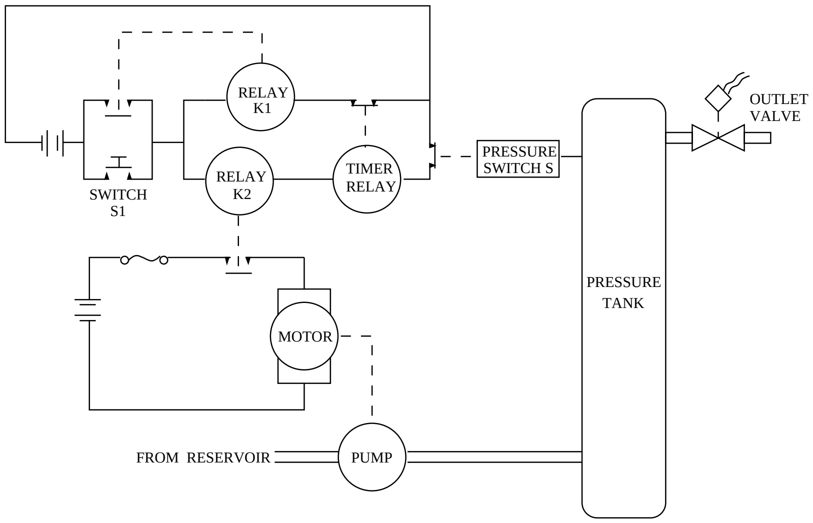


Figure 2.2: A pressure tank system.

switch contacts close, and the cycle is repeated.

Formal arguments for the safety of this system may involve a proof that the system is operational at all times. In safety-critical domains, though, it is also important to show that the system is acceptably tolerant to known faults, and such arguments are often supported by fault tree analysis.

The fault tree technique is a well-established method used in industry for analysing characteristics of systems under development. A fault tree is a model of the faults that can lead to an unsafe event, or *top event*, in such systems. Fault tree analysis evaluates weaknesses of the system by assessing the fault tree qualitatively and quantitatively. It identifies the possible combinations of basic events in a fault tree from which the top event can be derived (namely *minimal cut sets*), and estimates the probability of the top event from the probabilities assigned to the basic events. Thus fault tree analysis not only gives possible points of attack to the system model, but it also provides criteria for priority and relevance of such arguments.

Consider the top event of a fault tree for this system to be the *rupture of the pressure tank after the start of pumping*[1]. One of the minimal cut sets of this fault tree is

[1] See chapter VIII in (Vesely et al. 1981) for the fault tree analysis with respect to this top event.

composed of the basic event *primary failure of k2* . By primary failure of a component we mean that the component fails to work under circumstances in which it should work, so if *k2* contacts fails to open when the coil of *k2* is deenergised, then the tank will rupture.

According to the fault tree analysis in (Vesely et al. 1981), the probability of this minimal cut set is 3×10^{-5}, which is fairly high for safety standards. This represents a strong argument against system safety, but which can be undermined if we add some redundancy to the system; i.e. safety could be considerably improved by adding another relay in parallel to *k2*.

In such a way, the fault tree model is a source of possible arguments against system safety that can guide the revision of a system model in order to increase its acceptability with respect to known faults. Chapter 11 shows how our argumentation framework deals with a fault tree example taken from the safety engineering literature.

2.1.2 Negotiation between Agents

Negotiation is often described as the process of achieving mutually acceptable agreements between agents. Sometimes agreements are about finding acceptable solutions for common problems (rather than deciding on conclusions that are acceptable to all agents involved), which can be achieved in a sort of goal-oriented reasoning where agents take some goal as a starting point and interact in order to agree on how to satisfy it.

In this context, negotiation focuses on the construction of objects as solutions to open problems, and dynamic argumentation can provide means for building such solutions. In contract-based negotiation for instance, contracts are objects that can be adjusted based on reasoned arguments by the agents involved in the agreement so that it is acceptable for all the parties involved.

Assume that contracts are objects which regulate agreements between autonomous agents—*consumers* (or clients) and *producers* (or servers)—about the supply of products and services. The process of contract-based negotiation could be described as follows. Initially, one of the parties proposes a binding contract to regulate the agreement between them; without loss of generality, we can assume that a producer makes this first proposal. This contract is now the *object of negotiation* between producer and consumer, and can be seen as a set of formulae stating the conditions for accomplishing the agreement.

The consumer receives the contract from the producer and analyses it. If it agrees with the clauses, then the process of negotiation is over. More interestingly, the consumer might have reasons to believe that this particular contract will not be successfully completed. In this case, the consumer sends it back to the producer with the appropriate criticisms. The producer then tries to adapt some of the clauses in that particular contract in order to make it more acceptable, sending it back again to the consumer for further analysis. The process of adjusting the contract continues until there are no more criticisms (i.e. it is acceptable for producer and consumer) or until one of the parties withdraws. This process is similar to the kind of negotiation that humans perform in many situations involving contracts.

In such a way, negotiation can be viewed as a dynamic argument where the aim is to increase the acceptability of a contract by revising it in terms of possible objections from participating agents, until all agents commit to it. Chapter 12 shows how our argumentation framework deals with an example of this sort in contract-based negotiation.

2.2 Specific Questions Addressed in This Thesis

There are two main reasons why we believe it is important to formalise and automate argumentation processes like the ones mentioned above. First, argument-based methodologies should be supported by (semi-) automated tools which can both guide knowledge engineers in developing knowledge bases that derive the intended consequences, and also support designers of argument systems in investigating properties and effects of certain attacks and revisions in a domain. Second, automated argument systems can be used by artificial agents that want to employ this technique to solve certain types of problems.

A number of more specific questions has steered the development of such a formalisation of argument dynamics (together with those general questions stated in Section 1.3), such as:

- Which concepts are involved in argument dynamics, and which of these would be interesting to formalise? Can these be defined in a general way or are they (or some of them) domain-specific?
- How to represent and generate an argument? What types of arguments are important to be represented?
- How do arguments relate to each other and what types of relationships can be defined between arguments?
- Where do attacks come from?
- What mechanisms are used to prioritise arguments, and how can contextual (domain) information be incorporated into such mechanisms?
- When do dynamic arguments terminate?

Now, before moving towards a formalism for capturing arguments dynamics, the next chapter presents an overview of the existing work in argumentation in the context of knowledge engineering.

Chapter 3

Argumentation and Knowledge Engineering

One of the contributions of this thesis is to characterise the types of problems in the knowledge engineering domain that have been tackled by formal argumentation. This chapter surveys the state-of-the-art in formal models of argumentation and presents a classification in terms of problems they are meant to solve.

Our goal is to illustrate the use of formal and structured semi-formal approaches to argumentation, evaluating its practical utility in knowledge engineering. Instead of taking the usual path of reviewing different proposals for solving a particular problem, here we analyse different issues that can be tackled by automated argumentation systems, briefly comparing these approaches to other paradigms found in the literature. This is not supposed to be an exhaustive survey, but an analysis of various formal representation styles that are obtained by looking at argumentation from different perspectives.

Because at this point we take such a broad view of argumentation, the systems we describe are diverse. To guide the reader and facilitate comparison, the existing argument-based efforts are analysed in terms of general problems stated at the beginning of each section. The chapter is then organised as follows:

- Section 3.1 discusses how formal argumentation can deal with non-monotonic and defeasible reasoning;
- Section 3.2 reports on some of the argument-based approaches for decision making

and reasoning under uncertainty;

- Section 3.3 reviews some applications of argumentation in distributed settings, paying particular attention to multi-agent negotiation systems;
- Section 3.4 focuses on systems that use argumentation to support the design of an artifact, especially in the software development context.

Because many argument-based systems share similar features and purposes, it is hard (if not impossible) to establish a definitive classification of which research falls into which category. However, an analysis based on our problem-oriented classification helps to highlight strengths and problems in the existing proposals.

Finally, Section 3.5 summarises the current state-of-the-art and speculates on important directions in argument-oriented research in knowledge engineering.

3.1 Argument and Non-monotonic Reasoning

3.1.1 Problem Description

This section considers the problem of drawing conclusions from a knowledge base in the face of incompleteness and inconsistency. Very often, the addition of new propositions into a knowledge base can invalidate previously held conclusions and introduce contradictions. In this case, reasoning is said to be non-monotonic.

Non-monotonic or defeasible reasoning[1] addresses the problem of reasoning under incompleteness and inconsistency in the sense that some conclusions can be taken back in the presence of new information. That is, a proposition can be accepted until a better reason for rejecting it is found. Approaches for dealing with non-monotonic reasoning should then have means for deciding which conclusions are justified and acceptable in a knowledge base. Here we investigate how formal argumentation models can provide this means.

[1] The term *defeasibility* has its origins in the context of Legal Philosophy—see (Prakken and Vreeswijk 1999, p. 10) and (Chesñevar et al. 1999, p. 3). As argued by Pollock (1987), the ideas behind *defeasible reasoning* as it is studied in Philosophy and *non-monotonic reasoning* in Artificial Intelligence are roughly equivalent, hence these terms have often been used interchangeably.

3.1.2 Defeasible Argumentation

Several approaches for formalising non-monotonic reasoning have been proposed in the literature, such as default logics (Reiter 1980; Antoniou 1998). Argumentation provides a different perspective to non-monotonic and defeasible reasoning, in which a claim is accepted or withdrawn on the basis of the arguments for and against it, and on whether these arguments can be attacked and defeated by others. This view has been characterised as *defeasible argumentation*[2] and gained momentum after the publication of the work of Loui (1987) and Pollock (1987). Since then, myriad defeasible argumentation systems have been proposed (Nute 1988, 1994; Lin and Shoham 1989; Simari and Loui 1992; Freeman 1993; Brewka 1994; Dung 1995; Bondarenko et al. 1997; Jakobovits 2000), also motivated by research in the area of legal reasoning (Kowalski and Toni 1996, 1994; Verheij 1996; Prakken 1997a,b; Prakken and Sartor 1997, 1996; Vreeswijk 1997). It is important to note that the field of Artificial Intelligence and Law has proved a fertile domain for defeasible argumentation research and applications. This section, however, does not describe particular approaches to legal argumentation.[3] Instead it concentrates on general techniques for tackling defeasible reasoning based on argumentation, often referred to as argument-based semantics.

In general, defeasible argumentation systems are intended to characterise precisely whether an argument is acceptable based on its relations to other arguments. Prakken (1995) has identified a generic conceptual framework which underlies the majority of existing defeasible argumentation systems. This framework consists of five basic notions that may not always be explicit:

1. an underlying logical language;
2. a concept of argument;

[2] A comprehensive view of logics for defeasible argumentation can be found in (Prakken and Vreeswijk 1999), and this section is partly based on it. For another survey on this topic, including a historical account of argumentation and defeasibility, see (Chesñevar et al. 1999).

[3] An overview of legal applications of defeasible argumentation can be found in (Chesñevar et al. 1999, pp. 12–14). A more recent roadmap paper (Bench-Capon et al. 2000) brings together various strands of research in this area to create a conceptual model for the rational reconstruction of legal argument. For more specific references, the interested reader can refer to the Artificial Intelligence and Law Journal and to the Proceedings of the International Conference on Artificial Intelligence and Law, both accessible from the homepage of the International Association for Artificial Intelligence and Law at `http://ais.gmd.de/iaail/`.

3. a concept of conflict between arguments;
4. a notion of defeat among arguments; and
5. an account of the acceptability status of arguments.

The status of one argument depends on the whole set of arguments, and can be specified in two ways: *declaratively*, by defining a class of acceptable arguments; and *procedurally*, via proof-theoretical mechanisms for determining whether an argument is in this class.

A different view of procedural models was summarised by Loui (1998), who argues that what makes beliefs rational is not only their relations to other beliefs, but also the way in which they are built as the outcome of deliberative processes. In this sense, Loui gives an account of defeasible argumentation as resource-bounded, dialectic disputation protocols. Protocols are procedural models for constructing arguments based on notions such as which parties are involved; what are the possible moves for each party; how moves affect the outcome; how to determine if a disputation has finished; and if it has been won or lost. For the outcome to be rational, such protocols must be *fair* (e.g. parties get the same amount of resources, such as time) and *effective* (e.g. when a conclusion is established, it means that maximum resources were used in unsuccessful criticisms).

More recently, Prakken (2000) has also been focusing on the study of dialectical protocols, but from a slightly different perspective than Loui's. Rather than considering partial computation and limited resources, Prakken (2000) is interested in cases where new information is added during the process, and in characterising the properties that make protocols appropriate in these situations (e.g. if a participant could have advanced an attack, this participant had the chance to do so during the argumentation). In his words, protocols must be *fair* and *sound*. One could think of such protocols as representing dynamic argumentation with external changes (see Chapter 2), in the sense that they do account for changes in the underlying knowledge base but are not concerned with exactly why nor when these happened.

It has been argued that these sorts of procedural models are at a different layer of argumentation, a layer concerned with disputes and dialogue games rather than declarative acceptability of arguments. Bench-Capon et al. (2000) summarise the four types of layers often considered in computational models of argument: a *logical layer*—corresponding to the underlying logic mentioned above—for generating arguments and justifications based on a monotonic logical system; an *argument framework layer*—addressed in this section in terms of the framework above—for dealing with non-monotonic and defeasible reasoning by classifying acceptable arguments based on conflicts and attacks from a fixed set of premises; a *procedural layer* for regulating real disputes in which information can be added or challenged dynamically; and a *heuristic layer* on top of the procedural layer for considering efficient strategies for selection and presentation of arguments during a dispute.

This thesis is mostly concerned with the latter two layers, particularly on how dynamic changes to the set of premises relate to types of attacks that can be generated. The analysis in this section, though, is concerned with non-monotonic and defeasible reasoning, and hence with the *argument framework layer*. We base this analysis on the generic conceptual framework above, so it is possible to identify many similarities and common features between existing systems for defeasible argumentation, and also differences between these systems in terms of variations of these basic concepts. We will be looking at this framework in detail in Section 3.1.3.

We are not presenting the various defeasible argumentation formalisms in detail. A comprehensive account of the most relevant ones can be found in (Prakken and Vreeswijk 1999) and (Chesñevar et al. 1999). Instead, the rest of this section focuses on a particular approach that is viewed as a unifying, abstract account of defeasible argumentation. The *Abstract Argumentation Framework* of Kowalski & Toni (also known as the BDTK approach) is a logic programming-based theory of argumentation that "unifies and generalises many approaches to default reasoning" (Bondarenko et al. 1997; Kowalski and Toni 1994). Most existing defeasible argumentation systems can be understood and described in terms of this formalism, which is discussed in Section 3.1.4.

Finally, Section 3.1.5 compares argument-based semantics approaches to other paradigms for capturing defeasible and non-monotonic reasoning found in the literature.

3.1.3 A Conceptual Framework for Defeasible Argumentation Systems

This section discusses the five main concepts behind formalisms for defeasible argumentation: an underlying *logic* notions of *argument*, *conflict* and *defeat*, and an account of the *possible status of an argument.* Note that these are not always explicit, and the terminology used to designate them may also vary between argumentation systems.

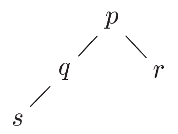
Each element is briefly described below based on the more complete account given in (Prakken 1995; Prakken and Vreeswijk 1999).

Underlying Logic As discussed earlier, formal argumentation systems are characterised by the use of formal knowledge representation and inference techniques. The underlying logic is essentially the formal logic system defining a *monotonic* consequence relation as the basis for deriving arguments. For instance, we might adopt a Horn clause resolution-based system as the underlying logic. Such systems are fundamentally deductive and therefore monotonic.

Arguments Arguments correspond to proofs in the underlying formal system. Consider, for example, the set of Horn clauses below:

$$p \leftarrow q \wedge r \qquad q \leftarrow s \qquad r \leftarrow \mathit{true} \qquad s \leftarrow \mathit{true}$$

Then the following proof of p (depicted as a tree with lower nodes supporting the conclusion above) is said to be an argument for p.

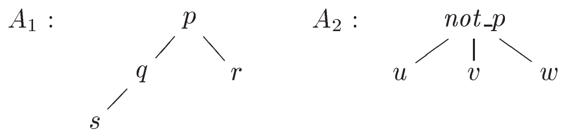


Conflict Intuitively, argumentation presupposes disagreement, which is captured in this framework by the notion of conflict. Also referred to in the literature as *attack* or *counter-argument* (Prakken and Vreeswijk 1999), *conflict* determines which conclusions in a knowledge base can be considered contradictory. For example, the sentences $married(X)$ and $bachelor(X)$ can be seen as conflicting, when instantiated by the same value for X.

It is possible to identify different types of conflict in terms of the underlying system, e.g. *rebuttal*. Arguments are said to be rebutting if they have contradictory conclusions. Assume for instance propositions p and not_p to be conflicting, and suppose the following clauses are added to the small example above:

$$not\_p \leftarrow u \wedge v \wedge w \qquad u \leftarrow true \qquad v \leftarrow true \qquad w \leftarrow true$$

then the arguments A_1 and A_2 below are examples of rebutting arguments.



Defeat Because the underlying logic is monotonic, the addition of new information does not invalidate existing arguments or previously derived conclusions, so conflicting arguments may coexist in a knowledge base. In the item above, for example, we are able to derive arguments for both p and not_p. The non-monotonic character of argumentation arises from the fact that some arguments may be preferred over others, and we should have means to decide which of these arguments are acceptable.

The notion of defeat is usually based on some comparative measure for arguments and a criterion based on this measure for adjudicating between conflicting arguments. One way to do this is to assign some priority order to certain clauses in a knowledge base, and to use this order to decide between arguments. For instance, if the clause $not\_p \leftarrow u \wedge v \wedge w$ has precedence over $p \leftarrow q \wedge r$, then the argument A_2 for not_p defeats the argument A_1 for p.

It has already been argued that such criteria are usually domain specific (Konolige 1988; Prakken and Sartor 1997), but in some cases it is possible to apply generic, domain independent standards such as the specificity principle[4] (Simari and Loui 1992).

[4] The specificity principle is a priority measure in which rules that deal with specific cases are preferred over generic ones. For example, if we can derive the following conflicting arguments:

Tweety flies because Tweety is a bird
Tweety does not fly because Tweety is a penguin

then by the specificity principle the argument for *Tweety does not fly* is preferred because the fact that *Tweety is a penguin* is more specific than the fact that *Tweety is a bird.*

Status The goal of a defeasible argumentation system is to determine which claims and which arguments are acceptable. The notion of acceptability can vary from formalism to formalism, but intuitively an argument that defeats a conflicting argument but is also defeated by a third one is not acceptable. Therefore it is not enough to just look at the two conflicting arguments alone to decide upon them, but instead all relevant arguments must be considered before making a decision.

For instance take the knowledge base that extends the examples above by the addition of the following clauses.

$$not\_u \leftarrow t \wedge z \qquad t \leftarrow true \qquad z \leftarrow true$$

Let the conflicting propositions be p and not_p; and u and not_u, and assume the following priority ordering is assigned to this knowledge base.

- $not\_p \leftarrow u \wedge v \wedge w$ has precedence over $p \leftarrow q \wedge r$;
- $not\_u \leftarrow t \wedge z$ has precedence over $u \leftarrow true$;
- every other clause has equal precedence.

We know from this ordering that argument A_2 for not_p defeats argument A_1 for p. However, this is not enough to decide that argument A_1 is not acceptable. This is because there might exist an argument A_3 that defeats A_2, thus restoring the validity of A_1. In fact, the following argument for not_u defeats A_2.

$$A_3: \quad \begin{array}{ccc} & not\_u & \\ \diagup & & \diagdown \\ t & & z \end{array}$$

In a sense, the acceptable arguments in a knowledge base can be viewed as one way of settling existing conflicts. Sometimes, e.g. in the example above, there is *exactly one* way of settling conflict according to the way preferences were defined, hence the set of acceptable arguments is unique. There may be cases, however, where conflict can be resolved in alternative ways, and therefore alternative sets of acceptable arguments may exist.

A more refined view identifies three general classes of argument, intuitively described as follows:

- An argument is *justified* if and only if all arguments defeating it are not justified; e.g. A_1 and A_3 above are justified arguments.
- An argument is *overruled* if and only if it is not justified and it is defeated by a justified argument; e.g. A_2 above.
- An argument is *defensible* otherwise.

From this perspective there are two possible attitudes towards acceptance of arguments—*credulous* and *sceptical*. In credulous systems, an argument is accepted if it defensible. On the other hand, in sceptical systems an argument is accepted only if it justified. This distinction between justified and defensible is also possible in cases where there are alternative sets of acceptable arguments, so an argument is defensible if it is in *at least one* of these sets, but for it to be justified it must be in *every* alternative set.

This conceptual sketch is in line with Dung's view that every argumentation system consists of two essential parts: an **Argument Generation Unit** (**AGU**) for generating arguments; and an **Argument Processing Unit** (**APU**) for deciding whether an argument is acceptable. Dung (1995) argues that logic programming and non-monotonic reasoning are types of argumentation which can be formalised in an abstract way via notions of *argument* and *attack*. He proposes a method for generating meta-interpreters for argumentation systems, showing also that argumentation can be seen as logic programming. The method is simple and is described below:

- The **AGU** specifies the attack (or conflict) relationships between arguments. In (Dung 1995), these relations are considered to be primitive and represented in terms of a binary predicate *attack*: if an argument A attacks an argument B, this is expressed by $attack(A, B)$.
- The **APU** is the following logic program with negation as failure that determines whether an argument A is acceptable.

$$\begin{array}{rcl} acceptable(A) & \leftarrow & not\ defeat(A) \\ defeat(A) & \leftarrow & attack(B, A) \wedge acceptable(B) \end{array}$$

Intuitively, an argument is acceptable if it cannot be shown to be defeated, i.e. if there is no acceptable argument that defeats it. This captures the idea that an argument A can be attacked by another argument, which in its turn may also be attacked by a third one, therefore restoring the validity of A, but does not capture the distinction between justified and defensible arguments above.

From the perspective of this conceptual model we now take a closer look at the *Abstract Argumentation Framework*, a logic programming based characterisation of defeasible argumentation which is both generic and oriented towards computation.

3.1.4 An Abstract Account of Defeasible Argumentation

The Abstract Argumentation Framework in (Kowalski and Toni 1994, 1996; Bondarenko et al. 1997) gives a flexible way of dealing with defeasibility in argument. As a language independent formalisation of defeasible argumentation, it can semantically characterise many approaches to default reasoning. This framework is partly based on Dung's Argumentation Framework (Dung 1995), but a fundamental difference is that in Dung's formalism the notions of argument and attack are considered as primitives.

So let $(\mathcal{L}, \vdash)$ be a monotonic deductive system, where $\mathcal{L}$ is a formal language and $\vdash$ is provability relation such that $\Pi \vdash \alpha$ if there is a *deduction* of $\alpha \in \mathcal{L}$ from a theory Π. A theory is any set $\Pi \subseteq \mathcal{L}$.

Definition 3.1 (Abstract Argumentation Framework) *Let $(\mathcal{L}, \vdash)$ be a monotonic deductive system. An Abstract Argumentation Framework $(\Pi, \mathcal{A}, ^{-})$ with respect to $(\mathcal{L}, \vdash)$ is an assumption-based framework defined by:*

- *a theory $\Pi \subseteq \mathcal{L}$ representing facts or beliefs;*
- *a set of assumptions $\mathcal{A} \subseteq \mathcal{L}$, $\mathcal{A} \neq \emptyset$, that can extend any theory; and*
- *a mapping $^{-} : \mathcal{A} \rightarrow \mathcal{L}$ to capture the notion of contrary of an assumption—i.e. $\overline{\alpha} \in \mathcal{L}$ represents the contrary of $\alpha \in \mathcal{A}$.* □

A key motivation is that it should be possible to make explicit the assumptions on which defeasible reasoning is based. For instance, an argument which rests on such assumptions is accepted if there is no evidence to the contrary. Non-monotonicity arises when evidence against these assumptions is provided, thus the arguments based on them are no longer accepted (Bondarenko et al. 1997).

Definition 3.2 (Argument) *If a conclusion* $\alpha \in \mathcal{L}$ *can be derived from* $\Delta \subseteq \mathcal{A}$ *and* $\Pi \subseteq \mathcal{L}$*, then we say that* $\Pi \cup \Delta \vdash \alpha$ *is an argument for* α*.* □

Note that arguments are based on assumptions, and these assumptions can be attacked by others:

Definition 3.3 (Attack) *Let* $(\Pi, \mathcal{A}, ^{-})$ *be an Abstract Argumentation Framework. A set of assumptions* $\Delta \subseteq \mathcal{A}$ *attacks another set of assumptions* $\Delta' \subseteq \mathcal{A}$ *if there is* $\alpha \in \Delta'$ *such that* $\Pi \cup \Delta \vdash \overline{\alpha}$*.* □

The term *attack* used in this framework corresponds to the notion of *conflict* in the conceptual sketch in Section 3.1.3. Because an argument can only be attacked by means of its assumptions, conflicts between arguments are not symmetrical; i.e. if an argument A attacks an argument B, then B does not necessarily attack A. These sorts of attacks are known as *assumption attacks.* In this sense, all relations between arguments in the Abstract Argumentation Framework are reduced to undermining attacks, as illustrated in the following example, adapted from (Kowalski and Toni 1996) and (Robertson and Agustí 1999).

Example 3.1 . *Consider the following theory* Π *of an Abstract Argumentation Framework about inheritance.*

$$
\begin{aligned}
inherits(P, estate(B)) &\leftarrow valid\_will(W, B, P) && (3.1)\\
disinherited(P, estate(B)) &\leftarrow found\_guilty(P, murder(B)) && (3.2)\\
found\_guilty(john, murder(henry)) &\leftarrow && (3.3)\\
valid\_will(doc042, henry, john) &\leftarrow && (3.4)
\end{aligned}
$$

We say that a person P *inherits the estate of* B *if there is a valid will* W *from* B *to that person. On the other hand, we say that a person* P *is disinherited of the estate of* B *if this person has been found guilty of the murder of* B*. In a particular inheritance* *case, John has been found guilty of the murder of Henry, and there exists a valid will identified as doc042 naming John the beneficiary of Henry's estate.*

Intuitively, there is conflict if a person P both inherits and is disinherited of some estate. It should be possible to construct two rebutting arguments here: one supporting the conclusion $inherits(john, estate(henry))$*, and another* $disinherited(john, estate(henry))$*. However, from the formal definition of attack given above, we cannot derive any conflicting argument.*

Attacks are based on assumptions. Therefore, in order to allow arguments to be attacked we need to appropriately extend the expressions in the theory by adding assumptions as extra premises. Let the abducible sentences be represented by a non-provability operator of the form $cannot\_be\_shown(\alpha)$*, which denotes that a sentence* α *is assumed to be false if it cannot be proved to be true. Note that* $\overline{cannot\_be\_shown(\alpha)} = \alpha$*.*

Expressions (3.1) and (3.2) could then be rewritten as follows:

$$\begin{aligned} inherits(P, estate(B)) &\leftarrow valid\_will(W, B, P) \ \wedge \\ &\quad\ cannot\_be\_shown(disinherited(P, estate(B))) \qquad (3.5) \\ disinherited(P, estate(B)) &\leftarrow found\_guilty(P, murder(B)) \ \wedge \\ &\quad\ cannot\_be\_shown(inherits(P, estate(B))) \qquad (3.6) \end{aligned}$$

From Definition 3.3 we now have two undermining arguments corresponding to the intuitive rebutting arguments. □

There is no explicit criterion for deciding between two arguments in an Abstract Argumentation Framework. In fact, the notions of defeat and conflict coincide in the sense that every attack to an argument defeats this argument. Note that defeat can be symmetrical, so it is possible to have two arguments defeating each other. This is illustrated above, where the argument for $inherits(john, estate(henry))$, defeats the argument for $disinherited(john, estate(henry))$, and vice versa. In this sense, there are two ways of solving conflict in this inheritance base, corresponding to the following two alternative sets of acceptable arguments: one containing the argument supporting *inheritance*, and the other containing the argument supporting *disinheritance*. Both conclusions are defensible (but not justified), so in a credulous system both would be acceptable whereas in a sceptical system neither of them would.

It should be possible, however, to represent priorities and preferences in this framework. Actually, there are two ways to prioritise an argument in terms of assumptions without altering the semantics. One way is by removing assumptions so that the argument can no longer be attacked. The second way is by introducing labels to the expressions and adding rules that talk about their priorities. A methodology for doing the latter is described in detail in (Kowalski and Toni 1996). Next we illustrate both cases.

Example 3.2 *Consider again the example 3.1. Intuitively, if John is found guilty of murdering the owner of the estate he is supposed to inherit (Henry), it can be expected that he is disinherited of that estate, even if a valid will exists. Therefore, we would like to prioritise the argument for disinheritance with respect to the one supporting inheritance.*

One way to do this is by removing the assumption in expression (3.6). Therefore, in the theory consisting of expressions (3.5), (3.2), (3.3) and (3.4) there are no arguments attacking the argument for $disinherited(john, estate(henry))$.

Another way to prioritise arguments is by talking about priorities in terms of labels. Consider the following expressions:

$$
\begin{aligned}
r1 : inherits(P, estate(B)) &\leftarrow valid\_will(W, B, P) \ \wedge \\
&\quad\ cannot\_be\_shown(defeated(r1(P))) \qquad (3.7)\\
r2 : disinherited(P, estate(B)) &\leftarrow found\_guilty(P, murder(B)) \ \wedge \\
&\quad\ cannot\_be\_shown(defeated(r2(P))) \qquad (3.8)\\
defeated(r1(P)) &\leftarrow cannot\_be\_shown(defeated(r2(P))) \qquad (3.9)
\end{aligned}
$$

Expression (3.9) intuitively corresponds to the idea of "inherits unless is disinherited of", so the argument for inheritance is defeated in case a person is proved to be disinherited of the estate under consideration. In the theory composed of expressions (3.7), (3.8), (3.3), (3.4) and (3.9), the argument for $disinherited(john, estate(henry))$ *defeats the argument for* $inherits(john, estate(henry))$, *but the reverse does not hold because no clause exists for* $defeated(r2(john))$. □

Having defined the notions of defeat, the arguments in an Abstract Argumentation Framework can be evaluated in terms of their ability to defend themselves against attack (Kowalski and Toni 1994). The way in which the class of *acceptable arguments* is defined can vary according to the semantics that one wants to capture. In the case of admissibility semantics, for instance, an argument is acceptable if and only if it is consistent and it attacks every argument that attacks it.

Definition 3.4 (Acceptability) *An argument $\Pi \cup \Delta \vdash \alpha$ is acceptable if and only if the set of assumptions Δ on which it is based is admissible.* □

Definition 3.5 (Admissibility) *A set of assumptions $\Delta \subseteq \mathcal{A}$ is admissible if and only if, for every $\Delta' \subseteq \mathcal{A}$, if Δ' attacks Δ then Δ attacks $\Delta' - \Delta$.* □

To build an admissible argument for a conclusion α we first need to construct an argument $\Pi \cup \Delta \vdash \alpha$ and then augment the set of assumptions Δ so as to defend it against all possible attacks. Note that this is not trivial because by adding new assumptions to an argument we are also adding new potential points of attack against it.

Many other credulous and sceptical semantics for negation as failure can also be captured by adopting other definitions of acceptability.[5] In particular, different logics for default reasoning can be obtained by considering different notions of acceptability, different sets of assumptions or even by assuming a different underlying logic. The advantage of this framework is that it is both generic and oriented towards computation, since it can be implemented as a logic program. Recently, a parametrisable proof theory has been developed for it (Kakas and Toni 1999), where the different semantics that can be formalised via argumentation can be computed in terms of instances of these parameters.

3.1.5 Relation to Other Paradigms for Non-monotonic Reasoning

By appropriately instantiating the concepts described in Section 3.1.3, argumentation frameworks can provide a characterisation of different formalisms for default reasoning,

[5] It has recently been shown in (Dimopoulos et al. 1999) that credulous reasoning under admissibility semantics is as hard as under stable semantics, but in the case of sceptical reasoning it is actually easier. Other complexity results for some of the semantics captured by the Abstract Argumentation Framework can also be found in that paper.

such as logic programming with negation as failure, default logic and auto-epistemic logic, among others. Reconstructing these formalisms in terms of an Abstract Argumentation Framework means specifying appropriately each one of the elements in Definition 3.1, namely an underlying logic, a set of assumptions, and the notion of *contrary* of an assumption.

For illustrative purposes, consider the case of default logic.[6] A default theory is based on a first-order deductive system $(\mathcal{L}', \vdash')$ and can be defined as a pair (W, D), where W is a set of formulae in the underlying system and D is a set of default rules (Antoniou 1998). Default rules have the general form $\frac{\alpha:\beta_1,...,\beta_n}{\gamma}$, denoting that if α is true and if we can assume $\beta_1, ..., \beta_n$ to be consistent with α, then we can derive γ. Let $M\beta$ represent that *it is consistent to assume* β. A default theory (W, D) can then be described as an instance of an Abstract Argumentation Framework $(W, \mathcal{A}, ^{-})$ based on a deductive system $(\mathcal{L}, \vdash)$ as follows:

- $(\mathcal{L}, \vdash)$ is the underlying first-order deductive system:
  - $\mathcal{L} = \mathcal{L}' \cup \{M\beta \mid \beta \in \mathcal{L}'\}$;
  - $\vdash$ is defined by the set of inference rules R below,

    $$R = R' \cup \{\frac{\alpha, M\beta_1, ..., M\beta_n}{\gamma} \mid \frac{\alpha:\beta_1,...,\beta_n}{\gamma} \in D\},$$

    where R' is the set of inference rules defining $\vdash'$.
- $\mathcal{A}$ is the set of assumptions defined by $\{M\beta \mid \beta \in \mathcal{L}'\}$.
- The notion of the contrary of an assumption is defined as $\overline{M\beta} = \neg\beta$.

Recently, argumentation has also been applied to the problem of belief revision (Carbogim and Wassermann 2000), where an instance of the conceptual model in Section 3.1.3 is used in a resource-bounded belief model to decide whether an incoming belief should be accepted or not.

As summarised by Prakken and Vreeswijk (1999, p. 9), the argumentation paradigm seems to be applicable in areas other than defeasible reasoning:

[6] The interested reader should refer to (Bondarenko et al. 1997) for a more complete account of this reconstruction in terms of the Abstract Argumentation Framework with respect to the various possible semantics.

> [...] argumentation systems have a wider scope than just reasoning with default. Firstly, argumentation systems can be applied to any form of reasoning with contradictory information, whether the contradictions have to do with rules and exceptions or not. For instance, the contradictions may arise from reasoning with several sources of information, or they may be caused by disagreement about beliefs or about moral, ethical or political claims. Moreover, it is important that several argumentation systems allow the construction and attack of arguments that are traditionally called 'ampliative', such as inductive, analogical and abductive arguments: these reasoning forms fall outside the scope of most other non-monotonic logics.

The following sections then explore this *wider scope* of argumentation in other contexts.

3.2 Argument and Decision Making under Uncertainty

3.2.1 Problem Description

As argued by Fox and Krause (1992), decision making is not only about quantitative option selection. Practical reasoning—or reasoning about what is to be done—is a rather complex activity that involves many other functions, such as decision structuring, communication, and representation of values, beliefs and preferences. In particular, Fox and Krause (1992) have identified the following requirements that decision support systems should satisfy: *robustness*, *flexibility*, *accountability* and *soundness*.

What makes the problem of practical reasoning yet more complex is the fact that information on which decisions are based is very likely to be imperfect and uncertain. Below we describe some ways in which uncertainty can arise in a knowledge base.

- We can have degrees of confidence associated with the information in the knowledge base, and these measures should be propagated appropriately as we reason about it.
- Uncertainty may be present in a non-deterministic fashion, where either of two (or more) alternatives can come about, but we do not know which. This type of uncertainty is usually represented in terms of disjunctions in the knowledge base.

- Moreover, uncertainty can arise when we cannot explicitly account for the many conditions that are necessary for a rule or a relation to hold. This is usually called the *qualification problem*.

This section looks at the problem of decision making from the more complex perspective advocated by Fox and Krause (1992), considering cases where the information available is uncertain in one of the three senses described above.

3.2.2 Argumentation and Decision Making

Most standard decision theories do not address all the requirements identified by Fox and Krause (1992) appropriately. On one hand, symbolic approaches such as knowledge based expert systems are usually constructed in an *ad hoc* manner, and often considered to be brittle. On the other hand, probabilistic decision theories are not sufficiently flexible nor accountable with respect to the options considered, and therefore have limited appeal to users. In fact, psychological research indicates that people do not reason *probabilistically* when faced with uncertainty.[7] Moreover, it is not always possible to obtain precise, objective statistics in certain domains (Parsons and Fox 1997).

The argumentation paradigm has been explored as an alternative approach to standard decision making theories, where decisions are made by considering arguments for and against decision options. As stated in (Fox and Krause 1992):

> Argumentation captures a natural and familiar form of reasoning, and contributes to the robustness, flexibility and intelligibility of problem solving, while having a clear theoretical basis.

A recent statement on argumentation and practical reasoning has also elaborated on the roles and issues underlying argument-based decision support systems (Girle et al. 2000).

Argumentation has been applied extensively in domains such as risk assessment (McBurney and Parsons 1999, 2000) and medicine (Fox and Das 2000). The *Logic of Argumen-*

[7] See (Parsons and Fox 1997) for a more extensive discussion, including references to empirical evidence supporting this claim.

tation (Krause et al. 1995) in particular is a well-established formal model for practical reasoning in which a structured argument rather than some summative measure is used for describing uncertainty. That is, the degree of confidence in a proposition is obtained by analysing the structure of the arguments relevant to it. The *Dialectical Argumentation System* (Freeman 1993) is also based on the same ideas and motivations, but it has been less widely used than the Logic of Argumentation. Both will be discussed in Section 3.2.3.

Other argumentation-based decision theories look at decision making from the same perspective, but consider different representations of uncertainty. Section 3.2.4 briefly discusses some of these other approaches, in particular Haenni's *Assumption-based Systems* (Haenni 1998) and an extension of Dung's Argumentation Framework for modelling uncertainty (Ng et al. 1998).

3.2.3 The Logic of Argumentation

The Logic of Argumentation (**LA**) is a qualitative approach to decision making, presented as an alternative to standard formalisms in order to overcome some of the limitations imposed by them. The development of **LA** was largely based on Toulmin's work on informal argumentation (Toulmin 1958; Fox et al. 1992), particularly on his descriptive model of arguments which is summarised in Figure 3.1.

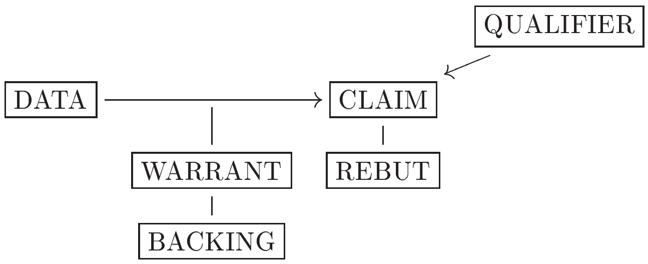


Figure 3.1: Toulmin's argument structure: a *claim* is supported by *data* (or evidence) and by a *warrant*, which is a general rule or principle supporting the step from data to a claim; the *backing* is a justification for the warrant, and the *rebut* is a condition where a warrant does not hold; a *qualifier* expresses the applicability of the warrant.

Fox and Parsons (1998) argue that certain characteristics of this structure make it suitable for practical reasoning in general and for decision making under uncertainty in particular. In contrast to strictly deductive mathematical reasoning, practical reasoning can involve imperfect information and inference relations other than deduction (Elvang-Goransson et al. 1993). In a sense, Toulmin's model accounts for some of these issues: the idea that conclusions are followed by a qualifier suggests that degrees of confidence can be associated with claims; and contradiction can also be represented in terms of the rebut component.

Arguments about Beliefs

In a nutshell, the idea behind **LA** is to analyse the structure of the arguments that are relevant to a proposition in order to obtain a degree of confidence for it. As stated by Krause and Clark (1993), "degrees or states of uncertainty can be viewed as a synthesis of the outcome of reasoning processes (i.e. arguments) germane to the proposition in question."

The Logic of Argumentation is based on a fragment of minimal propositional logic composed of connectives $\wedge$, $\rightarrow$ and $\neg$. In line with most formal frameworks for argumentation, an argument is defined as a proof in this logic, but also with the more pragmatic interpretation of *tentative proof* for indicating support for (or against) a proposition. Each argument in **LA** is represented as the following structure in a Labelled Deductive System style (Gabbay 1996):

$$(St : G : S),$$

where:

- St is any formula of the underlying logic. It corresponds to the conclusion of the argument, or the *claim* in Toulmin's structure.
- G represents the grounds on which the argument is based, i.e. the proof or justification for the argument. The idea is that the sentences and formulae used to derive St in the underlying logical system are explicitly represented in G. G is therefore similar to the *data* and *warrant* supporting the *claim* in Toulmin's model.

- S is a sign, i.e. an element of a dictionary (set) of symbols or numerical values representing possible degrees of confidence in the sentence St, thus capturing the notion of *qualifier* in Toulmin's model.

A number of dictionaries of confidence measures were defined and analysed in (Fox and Parsons 1998), with emphasis on symbolic ones. An example is the so-called *bounded generic* dictionary $\{+, ++\}$, in which $+$ indicates that a claim is supported whereas $++$ denotes that a claim is confirmed and hence cannot be rebutted with respect to the grounds on which it is based. The *delta* dictionary $\{+, -\}$ is another example of a set of symbolic degrees of confidence, where $-$ represents an opposing argument, or any argument that decreases the confidence in a claim. In the delta dictionary for instance the following relation holds:

$$(\neg St : G : +) \Leftrightarrow (St : G : -).$$

In summary, arguments are structures that describe how a sentence is justified. If Δ is a knowledge base composed of such argument structures, then new arguments can be generated from Δ via an argument consequence relation $\vdash_{ACR}$. Figure 3.2 gives some of the rules defining this relation in a consequent style. The interested reader can refer to (Krause et al. 1995; Fox and Parsons 1998) for a complete and detailed definition of $\vdash_{ACR}$.

To illustrate the types of arguments that can be represented in **LA**, consider the following example from a medical domain, adapted from (Fox and Parsons 1998).

Example 3.3 *Suppose that a patient has colonic polyps which could become cancerous. These beliefs can be represented in a knowledge base by the following arguments in terms of the bounded generic dictionary.*

b1:	*The patient has colonic polyps*	$(cp : \{b1\} : ++)$
b2:	*Polyps may lead to cancer*	$(cp \rightarrow ca : \{b2\} : +)$

Here cp stands for "the patient has colonic polyps" and ca for "the patient will develop cancer". The symbols $b1$ *and* $b2$ *are labels for identifying beliefs in the knowledge base. These labels are particularly useful for representing the sentences that are used to prove or justify an argument; i.e. its grounds.*

(Ax) If $(St : G : S)$ is in the knowledge base, then $(St : G : S)$ is an argument itself.

$$\frac{(St{:}G{:}S)\in\Delta}{\Delta \vdash_{ACR} (St{:}G{:}S)}$$

(∧E1) If we can build an argument for $St \wedge St'$ on the grounds of G and with confidence S, then we can eliminate one conjunct and build an argument for St on the same grounds G and the same degree of confidence S associated with it.

$$\frac{\Delta \vdash_{ACR} (St\wedge St'{:}G{:}S)}{\Delta \vdash_{ACR} (St{:}G{:}S)}$$

(→E) If we can build an argument for St and an argument for $St \to St'$, then we can build an argument for St'. The grounds on which St' is based are represented by the union of the grounds on which St and $St \to St'$ were derived. The degree of confidence associated with St' is obtained from a combination function with respect to the elimination of implication.

$$\frac{\Delta \vdash_{ACR} (St{:}G{:}S) \qquad \Delta \vdash_{ACR} (St\to St'{:}G'{:}S')}{\Delta \vdash_{ACR} (St'{:}G\cup G'{:}comb_{imp\_elim}(S,S'))}$$

Figure 3.2: The Argument Consequence Relation $\vdash_{ACR}$.

The argument consequence relation in Figure 3.2 can derive, on the grounds of the arguments above, that this patient may develop cancer.

b: The patient may develop cancer $(ca : \{b1, b2\} : +)$

More specifically, it uses the implication elimination rule **(→E)** *which can be understood as an special application of Modus Ponens in which the grounds and signs have to be propagated appropriately. In this case, the sign propagation function is a minimalisation of the degree of confidence.* □

Thereby **LA** provides a way of building the arguments that are relevant to a sentence. What still needs to be defined is a mechanism for combining every distinct argument in order to obtain a single confidence measure for the sentence in question. This mechanism is also known as *aggregation* or *flattening*, and is defined in terms of flattening functions over the adopted dictionary. If A^{St} is the set of all arguments $(St : G : Sg)$ relevant to a sentence St, then:

$$Flat(A^{St}) = \langle St, v\rangle,$$

where v can be an element of the given dictionary, but can also be drawn from different ones.

The symbolic aggregation procedure defined in (Krause et al. 1995) is an example of the latter case. It combines arguments for (+) and against (−) a proposition into an element of a different dictionary (corresponding to v above) composed of the following linguistic terms:

$$\{certain, confirmed, probable, plausible, supported, open\}.$$

Furthermore these terms closely resemble the qualifiers used by Toulmin. One advantage of this approach is that it can provide a high level summary of the available evidence without going into details of the aggregation procedure.

From the perspective of argumentation, practical reasoning in general and decision making in particular can be characterised as a two-step process in which we first construct arguments for the alternative options and then we select the most acceptable one (Elvang-Goransson et al. 1993). The difference between this approach and the one presented in Section 3.1 is that here *degrees of acceptability* are associated to each sentence, and therefore the argument processing step consists of picking *the most acceptable* argument instead of identifying the *acceptable* ones. It has been shown that the Logic of Argumentation can be related to other systems for non-monotonic reasoning, such as default logic. But unlike the argument-based applications to non-monotonic reasoning, **LA** does not in itself account for the dialectical perspective of argumentation, nor for the possibility of reinstatement. Such aspects are now being explored more broadly in multi-agent negotiation contexts, as described in Section 3.3.3.

A clear mathematical semantics for argumentation and aggregation is provided in terms of category theory (Ambler 1996), so that proofs of soundness can be developed for the systems based on **LA**. Other alternative semantics have also been proposed, for instance the probabilistic semantics in (Parsons and Fox 1997) allows **LA** to represent probabilistic reasoning.

In the context of decision support systems the argumentation paradigm has been proved quite effective (Fox and Das 2000; Fox and Parsons 1998). The Logic of Argumentation has been widely used as the basis of an agents' internal architectures[8] and employed in a number of practical reasoning tasks, especially in medical domains where systems for supporting medical diagnosis are amongst its applications (Parsons and Fox 1997).

Arguments about Actions

Reasoning about beliefs—*what is the case*—is actually different from reasoning about actions—*what we ought to do* (Fox and Parsons 1998). In the first case **LA** can be applied to build arguments (or tentative proofs) supporting a particular conclusion. However, a different notion of support may be needed for reasoning about actions, which may involve values—*what is important or positive*—and expected values—*what is the expected value of doing a certain action.*

Expected values and utilities are traditional ingredients in standard decision theories. In the context of informal argumentation, these concepts were also explored in the New Rethoric (Perelman and Olbrects-Tyteca 1969), a theory that has inspired recent formal approaches such as Daphne (Grasso 1998). Daphne is a system that builds arguments to promote healthy nutrition education based on users' values and preferences.[9]

The following is an informal example extracted from (Fox and Parsons 1998) which extends Example 3.3 and gives an argument-based characterisation of a decision making theory involving both beliefs and actions.

Example 3.4 *Suppose that a patient has colonic polyps which could become cancerous. Since cancer is life-threatening, some action ought to be taken in order to preempt this threat. Surgical excision is an effective procedure for removing polyps, and hence this is an argument for carrying out surgery. Although surgery is unpleasant and has significant morbidity, this is preferable to loss of life, so surgery ought to be carried out.*

Part of this reasoning is about beliefs and could be represented in **LA**-*style as follows:*

[8] Fox and colleagues have developed the DOMINO model, an agent architecture based on the **BDI**—Belief Desire Intention—model (Rao and Georgeff 1991, 1995), and which incorporates procedures for decision making and plan execution based on the Logic of Argumentation (Fox and Das 2000; Das et al. 1996; Fox and Das 1996).

[9] Issues related to argument-based persuasion and guidance are raised in almost every contribution in (Norman and Reed 2000), as for instance in (Gerlofs et al. 2000; Crosswhite et al. 2000).

b1:	*The patient has colonic polyps*	$(cp : \{b1\} : ++)$
b2:	*Polyps may lead to cancer*	$(cp \rightarrow ca : \{b2\} : +)$
b3:	*Cancer may lead to loss of life*	$(ca \rightarrow ll : \{b3\} : +)$
b4:	*Surgery preempts malignancy*	$(su \rightarrow \neg(cp \rightarrow ca) : \{b4\} : ++)$
b5:	*Surgery has some side effect se*	$(su \rightarrow se : \{b5\} : ++)$

Other arguments are about values for representing whether a state is desirable or not.

v1:	*Loss of life is intolerable*	$(\neg ll : \{v1\} : ++)$
v2:	*Side effect of surgery is not desirable*	$(\neg se : \{v2\} : +)$

Arguments about the expected values of actions combine arguments about values with standard **LA** *arguments for reasoning about beliefs.*

ev1:	*Surgery should be carried out*	$(su : \{b1, b2, b3, b4, v1\} : +)$
ev2:	*Surgery should not be carried out*	$(\neg su : \{b5, v2\} : +)$

Furthermore, preferences between decision options and alternative courses of action should be represented, and here this is done in terms of a special predicate pref.

p1:	*Surgery side-effects is preferable to loss of life*	$(pref(se, ll) : \{v1, v2\} : ++)$
p2:	*It is preferable to carry out surgery than to not carry out surgery*	$(pref(su, \neg su) : \{ev1, ev2, p1\} : ++)$

Other types of argument can also be identified: closure arguments, whose grounds might include a proof that all relevant arguments have been considered; and arguments for committing to particular actions and decision options.

cl1:	*No arguments to veto surgery*	$(safe(su) : G : ++)$
co1:	*Commit to surgery*	$(do(su) : \{p2, cl1\} : ++)$

□

To deal with arguments about values—*such as* $v1$ *and* $v2$—and expected values—*such as* $ev1$ *and* $ev2$—Fox and Parsons (1998) have proposed a Logic of Value (**LV**) and a Logic of Expected Value (**LEV**), respectively. Arguments in **LV** and **LEV** have essentially the same format as the arguments in the Logic of Argumentation, explicitly stating the grounds on which they are based. Figure 3.3 summarises the sort of reasoning schema that combines belief arguments in **LA** with value arguments in **LV** to obtain an argument for the expected value of an action in **LEV**.

On the grounds of G, we can argue that action A will lead to condition C with confidence S	$(A \rightarrow C : G : S)$	(**LA**)
On the grounds of G', C has value V	$(C : G' : V)$	(**LV**)
Therefore, on the grounds of $G \cup G'$, action A has expected value E	$(A : G \cup G' : E)$	(**LEV**)

Figure 3.3: Reasoning about beliefs, values and expected values.

Apart from mechanisms for aggregating arguments about values and expected values, we also need a function that combines signs from **LV** and **LA** into a sign in **LEV**. That is, in Figure 3.3 we need a function for deriving an expected value E given a value V and a confidence measure S.

Compared to the Logic of Argumentation, **LV** and **LEV** are still in a preliminary stage of development. The proposal in (Fox and Parsons 1998) concentrates on identifying which behaviour to capture rather than on providing a complete formalisation and analysis of these logics. To our knowledge, systems that involve **LV** and **LEV** have not yet been effectively implemented. The merit of this approach, however, lies in the characterisation of the different aspects of decision making in terms of argumentation. Defining such aspects via separated argumentation systems is rather intuitive and provides a more intelligible account to the problem of decision making under uncertainty.

LA and the Dialectical Argumentation System

Also inspired by Toulmin's argumentation model is the work by Freeman and Farley (1992), namely a formal theory for reasoning, making decisions, and proving and justifying claims in *weak theory domains*, i.e. domains in which knowledge is uncertain, inconsistent or incomplete. Again, the motivation for applying argumentation to deal with incomplete knowledge is that finding an adequate method for attaching numerical values to propositions, and for combining and propagating these values is a difficult task. As stated in (Freeman and Farley 1992), "argumentation can be used as a method for locating, highlighting and organizing relevant information in support of and counter to proposed claims."

In contrast to the Logic of Argumentation, an argument may be viewed not only as a structured entity, but also from a dialectical perspective. This means that an argument is not only described as a structure that organises relevant information for and against a claim, but also as a dynamic process engaged by conflicting parties as in a debate. The argument structures adopted by Freeman and Farley (1992) correspond to a slightly extended version of Toulmin's original schema (see Figure 3.1) together with various qualifiers for capturing uncertainty. The extended Toulmin structures have been implemented as a Dialectical ARgumenTation System—**DART**—that generates arguments in a game-like, dynamic process. **DART** has been used to model simple legal arguments (Freeman and Farley 1996; Freeman 1993), but has not been applied to real world scenarios.

3.2.4 Other Argumentation-based Approaches to Uncertainty

Arguing about beliefs under uncertainty is not fundamentally different from arguing about the acceptability of a claim in a non-monotonic context as discussed in Section 3.1. For instance, Ng et al. (1998) propose a framework for dealing with uncertain and conflicting knowledge that extends the proposals in (Dung 1995) and (Prakken and Sartor 1997).

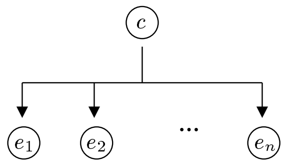
This proposal consists in applying argument-based mechanisms to resolve conflicts in a distributed setting, both within an agent's knowledge base and among different agents. The agents' knowledge bases are represented as extended disjunctive logic programs (Gelfond and Lifschitz 1991), where uncertainty is described by disjunctions in the head of the clauses. The clause below for instance says that *a dog barks when it sees a stranger or a fire*; so if a dog barks then we know that one of these alternatives is true, but we do not know which.

$$stranger \vee fire \leftarrow dog\_barks$$

As in (Prakken and Sartor 1997), two types of attack are considered: *rebuttals*, based on strong (or classical) negation; and *assumption attacks*, based on weak negation (or negation as failure). Defeat is based on an explicit preference hierarchy. In some specific cases, such as a single agent scenario, this framework can be proved equivalent to that of (Prakken and Sartor 1997).

Another approach to argument-based uncertainty has recently been proposed, this time in terms of assumption-based systems. Haenni (1998) incorporates uncertainty as extra assumptions into propositional knowledge, analogously to the idea in the Abstract Argumentation Framework (see Section 3.1.4) of making explicit the assumptions on which defeasible reasoning is based. This connection is not surprising, as the latter is an assumption-based system itself.

Haenni's proposal consists in transforming uncertain causal relations into clauses in an assumption-based propositional logic, and then building arguments for hypotheses based on these assumptions. For instance, the causal relations expressed by the graph below



could be represented by the following clause stating that if cause c is true then at least one effect among e_1, ..., e_n is also true.

$$c \rightarrow e_1 \vee e_2 \vee ... \vee e_n$$

Moreover, because some relations in a causal network can be uncertain, the effects may only come about under certain conditions, or assumptions. These assumptions are introduced as extra premises in the corresponding clauses, as shown below.

$$c \wedge a \rightarrow e_1 \vee e_2 \vee ... \vee e_n$$

An argument for an hypothesis is a set of assumptions that allow this hypothesis to be derived in the underlying propositional logic. An hypothesis is accepted or rejected based on the arguments *for* and *against* it; i.e. on the arguments that allow the hypothesis to be derived, and on the arguments that allow the falsity of the hypothesis to be derived. Note that this differs from the Abstract Argumentation Framework in the sense that counter-arguments are not defined in terms of assumption attacks, but in terms of rebuttals.

Just as in the Logic of Argumentation, it is possible to aggregate the arguments relevant to an hypothesis in order to obtain a confidence measure for it. In Haenni's proposal, however, the aggregation measure is purely *quantitative*, and it can be derived by assigning prior probabilities to the assumptions and propagating them accordingly. Note that the framework also fits in the two-step process characterisation of argumentation systems discussed in the previous sections, since we first build all arguments related to an hypothesis and then, based on these arguments, we evaluate it quantitatively.

The formalism described in (Haenni 1998) has been implemented in **ABEL** (Assumption Based Evidential Language), a modelling language for computing symbolic and numerical arguments for an hypothesis given an expert knowledge base and a set of facts and observations (Anrig et al. 1999). **ABEL** has been applied mostly for reconstructing standard AI examples, in particular in the model-based diagnosis and causal modelling domains.

Hence argumentation can be used to model decision processes under uncertainty in the sense described in Section 3.2.1. Moreover, because the informal notion of argument is naturally connected to that of disagreement between parties, it seems that this paradigm could also be applied in distributed scenarios. This is what we explore next.

3.3 Argument and Multi-Agent Systems

3.3.1 Problem Description

Intelligent software agents should be able to interact with other agents in many different ways. Such interactions usually pose a variety of issues related to information discovery, communication, reasoning, collaboration, coordination of joint approaches and social abilities. Some of these issues may be viewed as a process of achieving mutually acceptable agreements between agents (Parsons and Jennings 1997), where the nature of these agreements varies according to the type of problem to be addressed.

There are in general two types of agreement that can be attempted by agents. On one hand, agreement is about deciding on a conclusion that is acceptable to all agents involved. This sort of interaction usually takes place when there is a conflict that needs to be settled or resolved. On the other hand, agreement may be achieved in a goal-oriented type of reasoning, in which agents take a previously accepted goal as a starting point and interact in order to find an acceptable way of reaching or satisfying it. This sort of interaction arises when there is a common problem to be solved by the agents, who have to agree on a solution. In either case, it is important for agents to reason about their own beliefs, as well as about other agents' beliefs. So it is very likely that these interactions will be based on imperfect information in general and contradictory beliefs and intentions in particular.

Note that this way of looking at multi-agent interactions seems to be in line with the classification of dialogues given by Walton and Krabbe (1995). They have identified six basic types of argumentative dialogues, which can be characterised in terms of an *initial situation*, a *main goal*, and the *aims of the participants*. One systematic way for determining the type of a dialogue is to consider whether it starts from a conflicting situation or from an open problem to be solved, in a similar way as we have characterised the types of multi-agent agreements above. A more detailed discussion on the relation between models in argumentation theory and in multi-agent approaches is given in (Carbogim et al. 2000a), which addresses and identifies issues and open problems that are of interest to both communities.

In the agent community in particular the problem of achieving mutually acceptable agreements between agents has often been described as *negotiation*.[10] In this context, we now consider the problem of negotiation based on the two general types of agreements identified above.

[10] Negotiation is one of the six basic dialogue types identified in (Walton and Krabbe 1995)—it starts with a conflict of interests and has settling, or making a deal, as the main goal. The multi-agent community adopts a broader view of negotiation, usually defined as a general process for achieving agreements. This definition subsumes other types of dialogues such as *deliberation* and *persuasion*, but is still compatible with these: "negotiation dialogues may profit both from inquiries and from persuasion dialogues as sub-dialogues" (Walton and Krabbe 1995, p. 73).

3.3.2 Argumentation-based Negotiation

Research in argumentation in multi-agent settings has been guided by the question of whether it can provide or support intelligent interaction between agents. Recently there has been much interest in applying argumentation systems to capture negotiation, since processes for reaching agreements often involve the exchange of arguments between agents.

Here we present two ways in which negotiation processes can be formalised in terms of argumentation. Section 3.3.3 considers protocol-based argumentation approaches, which focus on the exchange of messages between agents, and therefore are particularly useful for reaching agreements about which conclusion to accept when there is conflict. Section 3.3.4 considers object-based argumentation formalisms. Such formalisms concentrate on the construction of objects as solutions to open problems, and therefore are appropriate for reaching agreements on how to satisfy or achieve certain goals. Note that this classification is not novel, as a similar distinction on argumentation-based negotiation research was presented in (Jennings et al. 1998).

3.3.3 Protocol-based Negotiation via Argumentation

Agent communication models or interaction protocols usually describe dialogues between agents in terms of notions that are relevant to argumentation, and therefore it is possible to look at them from an argumentation perspective. For instance, consider the case of the Knowledge Query and Manipulation Language—**KQML**—an agent communication language that provides a set of *performatives* through which agents can interact (Finin et al. 1997; Labrou et al. 1999). The notion of performatives comes from speech act theory, and essentially is used to convey some action about a message when transmitting it. Some **KQML** reserved performatives are shown in Figure 3.4.

More commonly, however, interaction protocols are only a part of argument-based negotiation models, which is used for dealing with communication issues. Negotiation formalisms normally extend single-agent argumentation frameworks (of the types presented in the previous sections) by using these to generate arguments which will be passed to other agents via some communication protocol, thus providing an argument-

Category	*Name*
Basic query	`evaluate, ask-if, ask-about, ask-one, ask-all`
Multi-response (query)	`stream-about, stream-all, eos`
Response	`reply, sorry`
Generic informational	`tell, achieve, cancel, untell, unachieve`
Generator	`standby, ready, next, rest, discard, generator`
Capability-definition	`advertise, subscribe, monitor, import, export`
Networking	`register, unregister, forward, broadcast, route`

Figure 3.4: Some **KQML** performatives classified into categories (Finin et al. 1997).

based approach for reasoning with imperfect information in a distributed setting.

For instance, the framework proposed by Móra et al. (1998) extends the single-agent declarative argumentation framework in (Prakken and Sartor 1997) to deal with cooperation among agents. Analogously to its single-agent counterpart, the aim is to decide which conclusions are acceptable in this distributed environment, also characterising the semantics of distributed logic programs in terms of argumentation. In this case, however, agents can cooperate by looking for support from other agents when trying to build arguments. Agents are defined as extended logic programs, so they cooperate by asking other agents to infer certain conclusions necessary to complete a proof. The communication process is implemented via an argumentation protocol based on five speech acts: `ask`, `reply`, `propose`, `oppose` and `agree`.

The approach defined in (Schroeder 1999a) is also based on the same declarative framework in Section 3.1.3. The proposal is preliminary, but it goes one step further in the direction of building *effective* operational argumentation systems, as Schroeder touches on issues related to the *heuristic layer*[11] such as the need to define strategies for selecting the best argument in order to reduce the number of exchanged messages and the need to increase general understanding of argumentation and logic, thus undermining some of the most common criticisms of the use of formal logic in modelling arguments. He addresses this need by proposing a graphical language for dynamically visualising argumentation processes (Schroeder 1999b).[12]

[11] See Section 3.1.2.

[12] Information about this language is available at `http://www.soi.city.ac.uk/homes/msch/cgi/viz/`. A system for cooperation between agents in business process modelling is also available at `http://www.soi.city.ac.uk/homes/msch/cgi/aca/aca.html`. This system was motivated by a

In the context of decision making, where it is important to resolve conflicting objectives and to coordinate cooperative actions, negotiation has been characterised in terms of a generic process for exchanging proposals, critiques, counter-proposals, explanations and meta-information. More recently, Wooldridge and Parsons (2000) have been focusing on the study of formal properties that generic logical languages for negotiation can have, as for instance what types of protocols are guaranteed to lead to an agreement. Below we discuss the protocol for negotiation proposed in (Parsons and Jennings 1997), and sketched in Figure 3.5.

Proposal A proposal is the basic element of negotiation, and it usually corresponds to an offer or a request.

Critique Intuitively, to critique a proposal means to reject this proposal, maybe attacking the parts which are not acceptable.

Counter-proposal A counter-proposal is a type of critique where the agent not only rejects a proposal, but also presents another (preferable) one.

Explanation An explanation is a justification or an argument for a proposal, critique or counter-proposal.

Meta-information Any piece of extra information that can be used for guiding the analysis and evaluation of proposals, such as information about preferences or values.

In the protocol outlined in Figure 3.5 there is no explicit indication of exchange of meta-information, as this type of message can be passed at any point by any agent. Arguments (explanations) may be sent together with critiques and proposals, and are represented by the formula ϕ.

This protocol forms the basis of the multi-agent decision making frameworks in (Parsons et al. 1998) and (Sierra et al. 1997b) which, although related, look at argumentation from two different but (maybe) complementary perspectives.

project for developing multi-agent models in the domain of business process management (Jennings et al. 1996), which also inspired the negotiation model in (Sierra et al. 1997b) discussed later in this section.

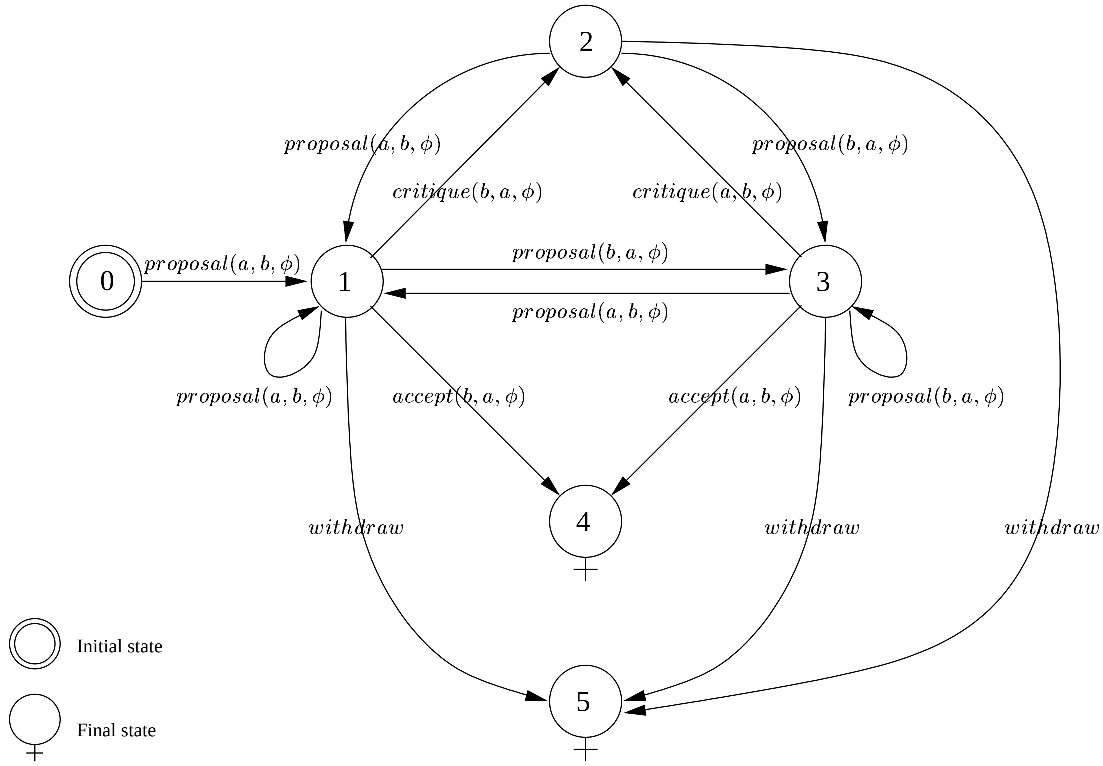


Figure 3.5: Negotiation protocol for two agents a and b (Parsons et al. 1998).

The work in (Sierra et al. 1997b) was motivated by multi-agent applications in business process management domains (Jennings et al. 1996). The emphasis in this proposal is given to the social aspects of negotiation rather than to the actual generation of proposals, so attack relations are assumed to be primitive as in Dung's approach. The model is based on a specific common communication language which deals with elements of persuasion (Sycara 1990)—such as `threat`, `reward` and `appeal`—that agents use to try to change each other's preferences, values and beliefs. Such changes are done in a rather domain specific manner, and some investigation on notions such as values and expected utility in the sense described in Section 3.2.3 might shed some light on how persuasion could be defined in more systematic terms.

While this work focuses on social elements, the framework in (Parsons and Jennings 1997) and (Parsons et al. 1998) is more concerned with providing the necessary mechanisms for implementing the negotiation process in Figure 3.5. More specifically, it uses the Logic of Argumentation to:

- generate proposals, critiques, counter-proposals, meta-information and explanations; and
- evaluate proposals, counter-proposals and meta-information.

The Logic of Argumentation provides means of generating proposals as arguments and of evaluating them in terms of their acceptability. A crucial difference between how **LA** is applied here and in a single-agent scenario is that now an agent has to make explicit not only the rules and facts that it used to generate an argument, but also the inference rules, because different agents might use different logics and therefore would not be able to reconstruct an argument if necessary. This issue is tackled by adopting a uniform underlying agent architecture, the multi-context architecture. An advantage of the multi-context approach is that it is generic enough to capture other architectures, such as the **BDI** framework (see footnote 8).

Although argumentation systems like **LA** give a generic architecture for a particular style of reasoning, much domain-specific expertise is required to instantiate this architecture to a domain of application. One way to define clear methodologies for the development of argumentation systems is to emphasise the problem and domain by identifying classes of problems in which certain evaluation principles would hold and then applying argumentation in these domains (Jackson 1994; Nwana and Ndumu 1999). The sorts of results given in (Wooldridge and Parsons 2000) represent one step in this direction. The Logic of Argumentation also provides a very good example of this, where a number of different symbolic dictionaries and aggregation mechanisms were identified as suitable for medical applications, allowing different argument-based systems to be implemented in this domain (Fox and Parsons 1998).

3.3.4 Object-based Negotiation via Argumentation

Negotiation-based models for decision making can also be seen from the perspective of the object being negotiated, rather than from a communication protocol viewpoint (Jennings et al. 1998). In general, objects are formalised as collections of issues (or variables) over which agreement can be made, and the process of negotiation consists in finding an assignment to the variables that suits every agent. However, it is also possible to consider a wider, constructive view in which the object under negotiation corresponds to an argument that has to be built by agents involved in mixed-initiative tasks. This more generic view subsumes the one where objects correspond to variables, allowing other types of negotiation processes to be characterised.

One example is the contract-based negotiation model initially proposed in (Carbogim and Robertson 1999) and described later in Part III of this thesis. Contracts are objects that are adjusted based on reasoned arguments by the agents involved in the agreement. In this sense, negotiation is about adjusting the terms of an agreement as opposed to the protocol-oriented view of forming an agreement. The same idea is explored in (Ferguson and Allen 1994), now in the context of mixed-initiative planning. Plans are explicitly represented as arguments that can be criticised and revised by the agents in a framework for plan construction and communication. The framework used for generating and evaluating arguments is based on previous work by Pollock (Pollock 1987) and Loui (Loui 1987). Unlike most defeasible argumentation systems, it is not used to derive defeasible conclusions from a plan, but to build a plan which is the defeasible argument itself.

In summary, the idea is to construct an argument (*plan*) supporting a particular conclusion (*goal*) which is acceptable to all agents involved. The example below, adapted from (Ferguson and Allen 1994), illustrates this type of reasoning:

Example 3.5 *Suppose that two agents are cooperating in order to construct a plan for transporting certain supplies (x) to a particular location. To get this done, they first need to move the supplies overland to the port and then carry them by ship. A ship (s) leaves every day between 4h00 and 6h00. If the supplies are shipped by train to the ship, they will arrive at 5h00. If they are shipped by truck, they will arrive at 3h00, but it will cost three times more than if transported by train. One possible interaction between the agents is defined below:*

- *Agent A suggests to ship the supplies by train.*
- *Agent B argues that the supplies will miss the ship if it leaves at 4h00.*
- *Agent A argues that the supplies will not miss the ship if it leaves at 6h00.*

- *Agent B then suggests to ship the supplies by truck.*

- *Agent A accepts this suggestion.*

Note that the agents could go on arguing if for some reason (such as shipping by truck is too expensive) agent A does not find the proposal acceptable. □

In order to build an acceptable plan, agents make proposals, evaluate suggestions and propose alternative course of actions, in a similar way as described in the protocol-based negotiation model of Figure 3.5. In this case, though, reasoning is goal-oriented—in Example 3.5 the goal is to load the ship with the supplies before it leaves the dock.

In (Ferguson and Allen 1994) this sort of reasoning is formalised by means of defeasible rules representing causal knowledge. Intuitively, these rules say that if the preconditions for an action a hold at time t, then attempting a at time t causes an event e_t to happen at the next time point. Defeasibility arises because it is hard (if not impossible) to specify all the preconditions for a rule to hold, and implicit or unknown conditions can invalidate the relation. This is also referred to as the *qualification problem*, already mentioned in Section 3.2.1. Defeasible rules have the following generic form.

$$Holds(precond(a), t) \wedge Try(a, t, e_t) \rightarrow Event(e_t)$$

$$\neg Holds(precond(a), t) \wedge Try(a, t, e_t) \rightarrow \neg Event(e_t)$$

The definition of an event uses material implication (denoted here by $\supset$) instead of defeasible implication to denote that the effects of this event will hold at the next time point.

$$Event(e_t) \supset Holds(effects(e_t), n(t)).$$

This representation can formalise part of the reasoning in Example 3.5.

$$AtDock(x, t) \wedge AtDock(s, t) \wedge Try(load(x, s), t, e_t) \rightarrow Load(e_t, x, s) \tag{3.10}$$

$$Load(e_t, x, s) \supset In(x, s, n(t)) \tag{3.11}$$

Moreover, it is possible to capture uncertainty in terms of disjunctions.

$$AtDock(s,t) \equiv \; t < 4h00 \;\vee\; t < 5h00 \;\vee\; t < 6h00 \tag{3.12}$$

The fact that $\rightarrow$ is a defeasible connective is important. Agents can build arguments for a particular goal and these arguments can be attacked because they involve elements of uncertainty and defeasibility.

What is more interesting about this approach is that it allows the representation of partial plans that do not take all the preconditions of an action into account. To build partial plans, agents can use variants of the existing causal rules obtained by considering only a subset of the preconditions specified in the original relation. As a consequence, a preference criterion can be defined based on the specificity principle: in case of conflicting positions, the position supported by the more specific variant (i.e. the rule in which more preconditions are taken into account) defeats the position that is based on a less specific variant of the same rule.

Example 3.6 *To illustrate this idea, we represent the possible variants of rule (3.10) ordered in a lattice of specificity, where the rule at the top is the most specific one.*

$$(a)\; AtDock(x,t) \wedge AtDock(s,t) \wedge Try(load(x,s),t,e_t) \rightarrow Load(e_t,x,s)$$

$$(b)\; AtDock(s,t) \wedge Try(load(x,s),t,e_t) \rightarrow Load(e_t,x,s) \qquad (c)\; AtDock(x,t) \wedge Try(load(x,s),t,e_t) \rightarrow Load(e_t,x,s)$$

$$(d)\; Try(load(x,s),t,e_t) \rightarrow Load(e_t,x,s)$$

In Example 3.5, agent A presents a proposal for sending the supplies by ship based on a partial plan that disregards whether the ship is in fact at the dock at the time of loading. Such a plan can be supported by variant (c) of the original rule (3.10). □

Other issues are involved in the type of argument described in the example which are not considered in this proposal. In particular, criteria other than specificity for evaluating arguments could be useful in this domain, especially to capture the idea of values and expected values of actions. Again, the work on practical reasoning and arguments about actions[13] is relevant also to this type of application.

The next section explores how this *constructive* view of argumentation has also been applied in a broader context.

3.4 Argument and Design

3.4.1 Problem Description

Design is the process of creating an artifact, but this general definition does not capture the complex, multifaceted nature of design activities. Moran and Carroll (1996) identify four distinct paradigms in the literature which try to portray the nature of design: design as decomposition and re-synthesis; design as search in a design space; design as a process of deliberation and negotiation, in which uncertainty and disagreement is intrinsic; and design as a reflective activity. They also describe a number of issues that must be considered if we are to address the various aspects inherent to the problem of design, some of which are listed below:

- how to represent changes in the problem definition;
- how to keep track of the decisions taken and assumptions made during the design process;
- how to aid communication among different participants in the process.

These sorts of issues are relevant to design processes in a variety of domains, from architectural design to engineering design and software design. This section considers them from the perspective of software design.

3.4.2 Arguing about Software Design

If we look at design as a mixed-initiative process of negotiation, then the object of the negotiation (in the same sense discussed in Section 3.3.4) is the artifact to be designed—

[13] See Section 3.2.3 and (Girle et al. 2000).

in this case, the software system. In this way, we move from the type of multi-agent applications to design support environments that possibly involve many participants.

There are two significant differences between the approaches considered in this section and the multi-agent negotiation models presented earlier in Section 3.3. First, in the context of design, less emphasis has been given to argumentation itself than to the problem being tackled (i.e. the design of a software system involving one or more parties). This is an important point, as argued by Moran and Carroll (1996, p. 7):

> A lot of domain-specific knowledge is needed, and the practices of design are different in different domains [...] Useful design tools need to be domain-specific, but many of the principles behind the tools are generic.

The second difference is that in the software development scenario, argument systems for supporting design have been applied to fairly complex scenarios.

One way to relate the use of argumentation to software design is in terms of *viewpoints* in requirements engineering (Finkelstein et al. 1994, 1992). Though viewpoints are not explicitly characterised as arguments, they involve many ideas germane to the argument paradigm, allowing multiple perspectives to be described and integrated by dealing with inconsistencies just when it is necessary, thus preserving these different perspectives as long as possible.

Also in the context of system requirements, a number of approaches for generating safety arguments have been presented in (Krause et al. 1997). Safety arguments are normally intended to convince people that the specified system will be safe if it is implemented appropriately. According to MacKenzie (1996) there are essentially three types of safety arguments. *Inductive* arguments support that a system is safe by testing it. *Deductive* arguments correspond to mathematical proofs that the system is correct. Finally, *constructive* arguments rely upon the process of design itself, which is argued to be a safe process that results in safe outcomes. This section focuses on the latter form of arguments.

Many problems arise when we try to represent safety arguments formally, although it has been possible to obtain effective and useful results in domain-specific settings. A significant number of these problems stem not from the technicalities of the chosen argumentation system but from assumptions made about its design and deployment, since the entire safety argument cannot be made internal to the formal argumentation system and the fit to its external environment must be carefully shaped. A discussion of these issues appears in (Robertson 1999a; Gurr 1997).

More commonly, argumentation is embedded in design rationale and computer-supported collaborative argumentation (**CSCA**)[14] systems that support the development of design activities. Design rationale is about explicitly recording the reasons why an artifact was designed in a particular way. In argumentation-based design rationale, reasons are generally represented as semi-formal arguments in terms of Issue Based Information System—**IBIS**—models (Conklin and Begeman 1988). Section 3.4.3 discusses another argumentation-based methodology for software design rationale.

Related to design rationale and **CSCA** systems, argument-based mediation systems provide support for deliberative processes involving one or more participants (users), in which the main goal is to reach a decision of some sort. Examples of mediation systems are discussion fora, where it is important to argue and negotiate about different issues, including design issues. The Zeno Argumentation Framework (Gordon and Karacapilidis 1997) is an Internet-based environment that supports structured forms of group decision making, and it has been widely applied across different domains. Zeno is also based on Toulmin's model of argument, and can be thought of as a *formal* version of **IBIS** in the sense that it automatically labels and qualifies positions according to arguments and preferences (i.e. determines a degree of acceptability associated with each position). There is a focus shift between systems like Zeno and the formal approaches for decision making discussed in Section 3.2, as in the first the emphasis is on representing arguments based on different sources and perspectives rather than on generating these arguments from some set of premises.

3.4.3 Argumentation-based Design Rationale

Sigman and Liu (1999) use argumentation to connect software system requirements

[14] See http://kmi.open.ac.uk/~simonb/csca/ for a resource site in **CSCA**.

to the corresponding design dialogue, providing a methodology for capturing design rationale, identifying conflicts and assessing the acceptability of design options. A generic argumentation model is used to relate the components of software design to those of design dialogue. This model allows different perspectives to be represented in terms of requirements, constraints and design features. An overview of the argumentation model is sketched in Figure 3.6.

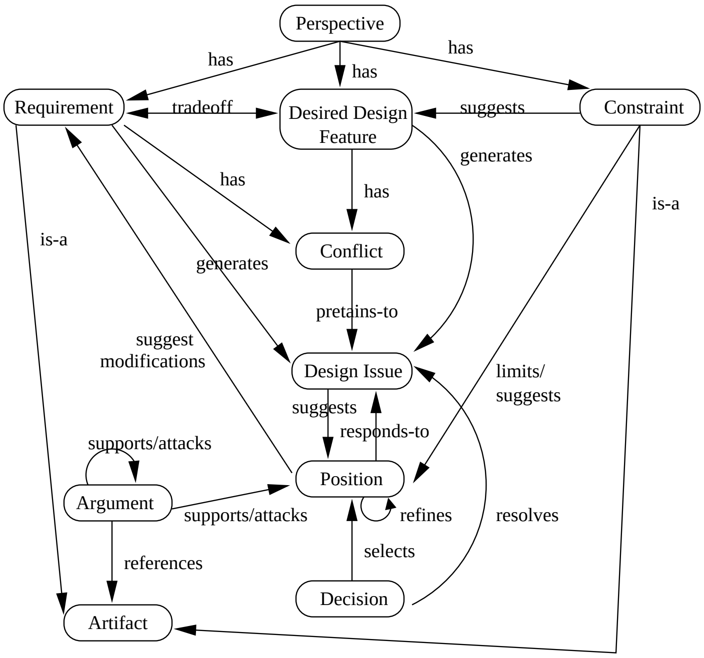


Figure 3.6: A software design argumentation model from (Sigman and Liu 1999).

A dialogue about a design issue is a directed graph in which all relevant arguments for and against each alternative position are organised under the corresponding position node. These structures are referred to as *position dialogue graphs.* Positions and arguments have to explicitly state their owner, i.e. the participant that has advanced them. Moreover, linguistic labels are attached to arguments to indicate their strength. The strength measure used is that of fuzzy sets, represented in terms of the following qualitative labels: `strong attack (SA)`, `medium attack (MA)`, `inconclusive (I)`, `medium support (MS)` and `strong support (SS)`.

Some general argumentation heuristic rules provide means of reducing the position dialogue graphs in a way that all arguments are directly connected to the position node.

This transformation is needed in order to identify inconsistencies as well as to assess the acceptability of the position. One example of such heuristic rules is defined below and illustrated in Figure 3.7 in terms of a simplified version of position dialogue graphs.

Heuristic Rule 1 If an argument A strongly supports a position P and an argument B strongly supports argument A, then argument B strongly supports position P.

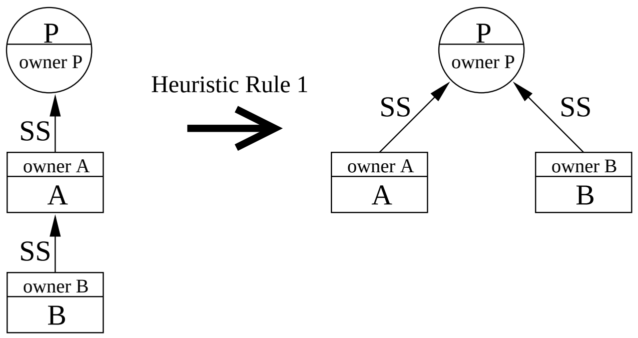


Figure 3.7: An example of a heuristic rule.

The acceptability of a position is then calculated via a *favorability factor*. The favorability factor is a function that assigns a strength measure to the position in question based on two aspects: the strength of the arguments that are relevant to this position; and the priorities previously assigned to participants, representing some idea of hierarchy among them. Comparing the favorability factors of alternative positions provides more information on which decisions can be based.

Note that this model resembles decision making approaches in many ways (see Sections 3.2 and 3.3). First, the idea of using linguistic labels is similar to that proposed in the Logic of Argumentation. Secondly, the calculation of the favorability factor for a position actually corresponds to the notion of aggregation procedures in **LA**. Third, the idea of assigning priorities to participants is in line with the social aspects considered in the negotiation approaches in Section 3.3.3. Finally, this proposal can also be characterised as a two-step argumentation model, because all arguments relevant to a position are first gathered and then analysed in order to provide an acceptability measure for this position.

The mechanisms for manipulating arguments in this framework and in other existing approaches are essentially the same. The difference lies in the notion of argument itself and in the ways arguments are generated. In this case, an argument does not correspond to a proof, but is represented by a piece of text stating the argument. We refer to these approaches as semi-formal, since argumentation is not fully automated; or as lightweight, in the sense that formality is applied only to certain parts of the problem in a focused and selective (e.g. automated argument evaluation). Lightweight uses of logic have already been advocated elsewhere (Robertson and Agustí 1999).

Buckingham Shum and Hammond (1994) have argued that structured semi-formal approaches to design rationale are *useful* and *usable*, and can play several roles in design, such as:

- structuring design problems;
- keeping track of decisions;
- facilitating communication and reasoning;
- assisting the integration of theory into design practice;
- supporting maintenance and reuse;
- exposing all assumptions—which may have been unstated, and conflicts—which may be suppressed; and
- enabling the formal incorporation of diverse types of information.

The approaches considered in this section are lightweight applications of formal argumentation which broaden the role of argumentation by carefully targeted applications of a simple formal method.

3.5 Discussion

The main purpose of this chapter was to analyse the practical use and usefulness of formal and structured semi formal argument-based systems in knowledge engineering. We have done this by classifying the existing efforts in terms of the problems they intend to solve, discussing whether these were actually solved or not, in which case we addressed some of the limitations and the remaining issues that need to be considered.

Four general types of problems have been identified which can be tackled by argument-based methodologies. These are:

- the problem of defeasibility in a knowledge base, where some conclusions might be withdrawn in the presence of new knowledge;
- the problem of decision making based on uncertain knowledge, where we have to decide which alternative to select;
- the problem of negotiation, where autonomous agents communicate and reason about propositions in order to reach an agreement; and
- the problem of design, where it is important to make decisions, to communicate decisions and to argue that the resulting artifact represents an acceptable solution to a particular problem.

One thing that these problems have in common is that they involve knowledge that is far from certain and complete. Potential disagreement and conflict are intrinsic to all four categories above. Therefore, the fact that conflict is the essence of argumentation might explain why the argument paradigm can be applied in these cases.

We have found many common features among the various approaches presented in this chapter. Below we summarise these commonalities:

- In general, formal argumentation can be characterised as a two-step process: first, arguments are generated; then, arguments are evaluated in terms of their acceptability.
- Automated frameworks for argumentation have appeared on the scene only recently. This is probably one reason why most theories are not yet mature enough to allow applications to be developed in a systematic way. In many cases *ad hoc*, specialised solutions have been adopted in order to implement practical systems from theoretical frameworks.

- This is particularly true for argument evaluation. Generic criteria, such as the specificity principle, are not sufficient for effectively capturing the notion of argument defeat across the myriad domains in which argumentation is applicable. Therefore, many theoretical formalisms tend to leave concepts such as preferences and priorities unspecified, but without addressing the issue of how to instantiate these appropriately in order implement practical argument systems from these formalisms.

- Because argumentation is such a broad concept, many already established formalisms can be viewed from an argument perspective. Examples are **KQML** (Section 3.3.3), viewpoints (Section 3.4) and probabilistic reasoning (Section 3.2.3).

- Only a few argument systems have actually been deployed in real, complex domains. Most systems have been evaluated in terms of simple benchmark problems.

There are still open research issues in each of these considerations which can reflect an expected direction of development in argument-oriented research in knowledge engineering.

- The idea of argumentation as a two-step process suggests that all arguments have to be computed before they are evaluated. This may not always be the best strategy if we want to build a constructive theory of argumentation for actually generating arguments, and perhaps more emphasis should be given to the sort of resource-bounded argument discussed by Loui and the *procedural* and *heuristic layers* of argumentation (see Section 3.1.2).

- Section 3.3.3 discussed the need for clear methodologies for the development of argumentation systems. Note that we do not advocate a *one-size-fits-all* approach to argumentation, as we believe that the multifarious nature of argumentation cannot be captured by a uniform method. However, we would like to provide means of implementing argument theories in a systematic way, by trying to identify different methods that allow different types of argument-based systems to be developed. This may be achieved by focusing on domains and problems rather than on tasks, thus specifying domain-specific underlying theories and evaluation criteria instead of generic, domain-independent formalisms for argumentation.

- It was possible to look at certain problems in knowledge engineering from an argumentation viewpoint. This suggests that if we take a more lightweight approach to argumentation formalisms, by using them in a focused and selective way, we might broaden the scope of their applications in the field. This may be achieved by considering more flexible, semi-formal notions of arguments other than that of a proof.

- Finally, to increase the practical utility of these systems, more complex and real arguments need to be taken into account. This again might be possible to achieve by appropriately lightweight applications of argument formalisms.

This summarises the current landscape of argumentation research, which is scattered with tantalising glimpses of problems which may be tackled by this means, yet there are few clear guides to standard practice in this area; nor are there extensive case studies to give maps of fertile domains. The work in this thesis draws a lot from existing work and from the analysis presented here in order to address some of the issues above.

Part II

A Pragmatic Approach

Chapter 4

Basic Concepts and Definitions

As discussed in Chapter 2, one way to think about argument dynamics is that it should be possible to revise the underlying knowledge base so as to defend arguments and positions from attacks, for instance by adding new information so that new arguments or counter-attacks can be derived or by removing certain premises so as to block existing derivations.

To formally describe this sort of argumentation we need to describe precisely *what* types of revisions can be performed, and *when* they can be applied. In comparison to external revisions, guided revisions have some interesting properties: first, we know more about *when* they happen, because they follow the pace of argumentation and are synchronised with argument moves; second, we know more about *what* they are, because they are bound up with and guided by attacks to arguments. In practice, however, there are many ways to attack and defend an argument, and these are essentially domain-specific. We address this problem in a pragmatic way by describing how to capture schemata for argument revision in terms of the structure of attacks, inspired by standard argumentative structures from studies in the fields of informal logic and argumentation theory.

The type of structural classification we present is not complete in itself, but it is based on a complete account of what we mean by dynamic argumentation. But despite its incompleteness, it allows the introduction of both generic and domain-specific revision schemata in a systematic way. We will be carefully examining this approach in the next chapters, where we first present an intuitive description of the classification in terms of informal examples (Chapter 5) before introducing its formal counterpart (Chapter 7) based on a precise characterisation of possible attacks (Chapter 6). We then give a worked example that illustrates how this approach can be used to capture dynamic argumentation (Chapter 8), finally discussing the range of cases covered by our proposal, and analysing how other existing proposals can cope with these cases (Chapter 9).

In the rest of this chapter we present a high level account of dynamic argumentation, before explaining the formal concepts underlying our approach.

4.1 An Abstract View of Dynamic Argumentation

Chapter 1 discussed the informal notion of *argumentation* as usually encompassing two views of an argument. In order to formalise argumentation, we need to account for both of them:

- a local view, in which an argument is intended to give support in favour or against a conclusion; and
- a global view, in which an argument is a process of argument exchange (from a local perspective), often based on disagreement, that is used to determine and to affect the acceptability status of certain controversial positions.

Our notion of dynamic argumentation comprehends both these views. From the local perspective, arguments correspond to formal proofs, and are generated from a knowledge base via a proof mechanism. From the global perspective, argumentation is a process of argument exchange and knowledge base revision guided by attacks. Crucial to both views is the concept of *knowledge base*, which we address more carefully now.

In a sense, this underlying knowledge base constitutes the space of reasons that can be used to justify and refute positions, and which can be challenged and altered as the argumentation process goes along; in short, it constitutes the space of argument premises. From now on, we refer to such knowledge bases or spaces of premises as *sets of axioms* or *theories*, the building blocks of our approach. An axiom set can represent specifications, models, contracts, beliefs or any other theories we might want to argue about with respect to the consequences they support. We can assume that these consequences can be derived from the premises via a logical inference relation. If a conclusion can be derived from a theory, we say that there is an argument for that conclusion in this theory.

Theories usually express someone's view of a problem rather than universal truths, and therefore are intrinsically arguable and refutable. It is likely that unwanted or unpredicted conclusions will follow from a theory, or even that desired conclusions are not supported. Dynamic argumentation is about revising this theory to protect it from such *attacks*; in this sense, it is concerned with *arguing about theories*.

Argument dynamics can then be thought of as a type of goal-oriented reasoning meant to increase the acceptability of a theory as an argument for the position in question by appropriately defending it from attacks. This view is particularly useful if we consider tasks such as argument construction and evaluation, where it is not enough to consider a sole claim, but the *whole* argument—i.e. the theory—supporting the claim.

In this way, we can attack a theory for two reasons: either because it supports a position that we would expect (or want) not to be justified; or because it does not support a position that we would expect (or want) to be justified. How we decide on which are the relevant claims that should or should not be justified in a theory is subject to a deeper discussion, which will be addressed later in Chapter 7. But the intuition behind it is simple:

- if a conclusion can be derived from a theory (if there is an argument for this conclusion in the theory) when we believe it should not be, then we can revise the theory in order to (try to) block this conclusion from being justified (in order to reject the argument supporting it);
- analogously, if a conclusion cannot be derived from a theory (if there is no argument for this conclusion in the theory) when we believe it should be, then we can revise the theory in order to (try to) allow the conclusion to be justified (in order to introduce an argument that justifies it);

As described above, revision of a theory is guided by the intention of either invalidating some existing argument, or adding a new argument to it. So instead of looking to dynamic argumentation as a process of revising a theory, we could consider it as a process for manipulating the arguments in that theory. Let us assume for a moment the notion of argument to be primitive, and consider the set of all arguments in a theory as the starting point of an argumentation process[1]. If we consider arguments to be primitive entities, then dynamic argumentation is about putting forward new arguments and rejecting others in order to attack and defend certain positions. So, as the process develops, new arguments can be added to the initial set, and others can be withdrawn.

An advantage of defining argumentation as manipulation of a set of primitive arguments rather than as revision of an underlying set of premises is that it is more intuitive to talk about introducing and removing arguments than it is to talk about which premises need to be added and removed in order to introduce or remove some argument. The relationship between these views is not straightforward, and it also depends on the choice of logic underlying the generation of arguments. This more abstract approach, however, can be too abstract and also impractical, as accounting for the set of all arguments is likely to be a computationally expensive, if not infinite, task.

Here we take a pragmatic approach by trying to identify ways for capturing this more abstract view of manipulating sets of arguments in terms of guided revisions to the underlying theories that represent the premises of these arguments.

4.2 Formal Definitions

In this section we formally define some general concepts underlying our approach to dynamic argumentation. We start by defining what is meant by *axiom* and by *theory*. Theories and axioms are at the heart of our proposal, as they represent the premises on which arguments are based.

---

[1] A lot of research in formal argumentation is actually based on this assumption, e.g. (Dung 1995) and (Prakken 2000). Jakobovits (2000) also describes how to obtain this set of all arguments from a logic program.

Definition 4.1 (Axiom) *Let $\mathcal{L}$ be a logical language. An axiom is any well-formed formula in $\mathcal{L}$. Π.* $\square$

Definition 4.2 (Theory) *Let $\mathcal{L}$ be a logical language on which a provability relation $\vdash$ is defined. Let $\mathcal{F}_{\mathcal{L}}$ be the set of axioms (formulae) in $\mathcal{L}$. A theory in $\mathcal{L}$ is any consistent subset of $\mathcal{F}_{\mathcal{L}}$, denoted by the possibly indexed symbol Π.* $\square$

Note that at this point we are not making any commitments on the choice of logic underlying an axiom set, nor on the inference rules associated with it; these will be defined in more detail in Chapter 7. For the moment we assume that theories and axiom sets are composed of facts and rules (conditionals).

4.2.1 Arguments

As in most conventional formalisms, arguments are associated with the provability relation in the underlying logical system, and therefore correspond to logical proofs. Such arguments are often used to indicate support and justify positions.

Definition 4.3 (Argument) *Let Π be a theory and φ be a sentence in a logical system $(\mathcal{L},\vdash)$. If φ can be inferred from $\Gamma \subseteq \Pi$ via the provability relation $\vdash$, then $\Gamma \vdash \varphi$ is an argument (or justification) for φ in Π.* $\square$

Arguments are represented by a two-part structure (often denoted by the letter A) comprising an inference $\Gamma \vdash \varphi$ and the corresponding derivation tree, with lower nodes supporting the conclusion above. The generic form of a justification $\Gamma \vdash \varphi$ consists of:

- *a claim φ:* the conclusion of the argument;
- *the grounds, or evidence Γ:* the premises supporting the claim;
- *the reasoning $\vdash$:* the link that relates the conclusion φ (claim) to the premises Γ (evidence); here the reasoning step is based on a logical inference relation $\vdash$, and we often use the term $\vdash_{\Pi}$ to indicate that this relation is restricted to a theory Π.

Note that justification is not the only purpose of an argument. Arguments can play other roles, such as to attack other claims and arguments, for instance in the form of counter-arguments that justify opposing views, or in the form of refutations for rejecting other arguments. These roles are not concerned with individual justifications but with the relationships between them, hence they should be considered from a global perspective.

4.2.2 Dynamic Arguments

Instead of looking at arguments individually, dynamic argumentation is about considering how the relationships between relevant arguments will determine and affect the status of the corresponding claims. Note that having measures of acceptability is not a main part of this thesis. Instead we adopt a simple—yet expressive enough—notion of acceptability: a claim becomes acceptable when an argument supporting it is presented; but it becomes non-acceptable if this argument is attacked; moreover, if this attack is itself attacked, the acceptable status of the claim is restored. In a nutshell, a claim is acceptable if all attacks can be properly dismissed by means of counter-attacks (which are attacks themselves).

The whole idea of attack is based on conflict. An argument is said to attack another argument if they support contradictory conclusions in the underlying language. Moreover, it is also possible to attack and reject the grounds—or premises—on which an argument is based. Yet another type of attack, standard in informal argumentation, consists in rejecting the reasoning underlying an argument by suggesting that the conclusion does not follow from the premises. But in formal systems justifications are generated by means of a sound logical inference method, so we shall assume that the conclusion always follows from the premises.[2] This latter type of attack is therefore not relevant to our approach, and we could say that here argument defeasibility is reduced to premise defeasibility.

We also need to consider the fact that certain arguments may be preferred over others. Preference can sometimes be determined from the logical structure of arguments and claims, but it can also be based on comparative measures for arguments. The notion of preference between contradictory arguments is often referred to as *defeat*, and defined separately in terms of attack. Here we incorporate it in our definition of attack.

[2] An inference or proof method is said to be *sound* if it produces only conclusions that are logical consequences of its premises according to some defined notion of logical consequence. Remember that at this point we have made no commitment on the choice of a particular logical system, or of a logical consequence relation.

Definition 4.4 (Attack) *An argument A' attacks an argument A if and only if A' contradicts a claim supported by A and A is not preferred over A'.* □

Some aspects of this definition are commented below:

- First, an argument can support different types of claims, and a characterisation of what these claims might be can be extremely useful for describing the general format of attacks.
- Second, what it means for claims to be contradictory in a language—as well as what it means for arguments to be preferred over others—can depend on the choice of the underlying logical language itself.
- Finally, criteria for deciding if arguments are preferred may not always exist, in which case any argument is *strong enough* to reject a contradictory argument; but if such criteria exist, they are likely to be domain-specific.

These are important remarks and will be further elaborated mainly in Chapter 6, and later in Part III of this thesis.

Another concept we have to account for is that of revision. Notice that by revision we mean *structural revision*, in which some premises can be retracted from and others can be added to the original theory, allowing for instance for new concepts to be introduced. In the context of argumentation, this intuitively corresponds to the idea of challenging existing premises and bringing in new ones.

Definition 4.5 (Revision) *A structural revision operation ϕ in a language $\mathcal{L}$ is characterised by a pair $(\mathcal{R}, \mathcal{A})$, where:*

- *$\mathcal{R} \subseteq \mathcal{F}_{\mathcal{L}}$ corresponds to the axioms that will be retracted from a theory; and*
- *$\mathcal{A} \subseteq \mathcal{F}_{\mathcal{L}}$ corresponds to the axioms that will be added to a theory.*

The outcome of applying ϕ to a theory Π in $\mathcal{L}$ is a theory Π_ϕ obtained from Π as follows:

$$\Pi_\phi = (\Pi \setminus \mathcal{R}) \cup \mathcal{A}.$$

If $\mathcal{R} = \emptyset$ and $\mathcal{A} = \emptyset$ then ϕ is said to be trivial. If either $\mathcal{R}$ is a singleton and $\mathcal{A} = \emptyset$, or if $\mathcal{A}$ is a singleton and $\mathcal{R} = \emptyset$, then ϕ is said to be elementary. If ϕ is neither trivial nor elementary, then it is said to be complex. □

Observation 4.1 *Note that any non trivial operation can be decomposed into a sequence of elementary operations.* □

In the context of dynamic argumentation, revisions to a theory are performed in order to allow different types of attacks and counter-attacks to be generated. Therefore changes are guided by attacks, so revisions are defined in terms of the argument in a theory that is about to be attacked.

Definition 4.6 (Attack-based Revision) *Let Π be a theory and A be an argument about φ in Π. An attack-based revision operation ϕ to Π with respect to A defines a theory Π_ϕ such that in Π_ϕ we can derive an argument that attacks A.*

Attack-based operations are denoted by $\phi^{\Pi,A}$, as they may depend on Π and A (and consequently on φ). The superscript symbols may be omitted when the context is clear. □

Note that neither the argument to be attacked nor the theory need to be fully specified in an attack-based revision operation. Instead, such operations can be described by partially defined structures, like generic schemata for arguments and theories.

In a sense these operations are a bit like actions. They have preconditions that determine when they can be applied, and postconditions that define the outcome of applying them. In the next chapters we analyse the types of revisions that can lead to relevant attacks. We pay special attention to elementary operations, their properties and characteristics, and also how more complex revisions can be defined from them.

We can now formalise the concept of dynamic argument. At this point we would also like to emphasise the procedural nature of argumentation—in fact, argument dynamics can be seen as a mechanism for *proving* whether a position is acceptable with respect to a theory, where this proof process can involve revisions to the theory itself. Each argument that is advanced changes the acceptability status of the initial claim, and for the theory to be acceptable with respect to this claim it has to be revised until all attacks have been appropriately dismissed. Notice also that when we revise a set of axioms to defend it from attacks new points of attacks may be introduced, so the whole resulting theory should be again open to argument. This view is described below and illustrated in Figure 4.1.

Definition 4.7 (Dynamic Argument) *Let Π be a theory and φ be a sentence in a logical system $(\mathcal{L}, \vdash)$, and let Φ be a collection of attack-based revision operations defined in terms of generic schemata for arguments and theories in $\mathcal{L}$.*

A dynamic argument δ about Π with respect to φ is denoted by a sequence:

$$\delta(\varphi, \Pi) = \langle A_0, \phi_1, A_1, ..., \phi_K, A_K, ...\rangle, \textit{ where}$$

- *A_0 is a justification for φ in Π;*
- *$\phi_1, ..., \phi_K, ... \in \Phi$ is a sequence of revision operations to Π;*
- *for $i \geq 1$, A_i is an argument in $\Pi_{\phi_1 ... \phi_i}$; and*
- *for $i \geq 1$, A_i attacks A_{i-1} in the context of the moves $\langle A_0, \phi_1, A_1, ..., \phi_{i-1}, A_{i-1}\rangle$ advanced so far.*

If there is $N \geq 0$ such that no attack-based revision $\phi \in \Phi$ can be applied to $\Pi_{\phi_1 ... \phi_N}$ with respect to A_N, then we say that $\delta(\varphi, \Pi)$ converges to $\Pi' = \Pi_{\phi_1 ... \phi_N}$. Also, if N is even then Π' is said to be acceptable in relation to φ (or yet that φ is acceptable with respect to Π'), as the attacks to φ have been appropriately dismissed. □

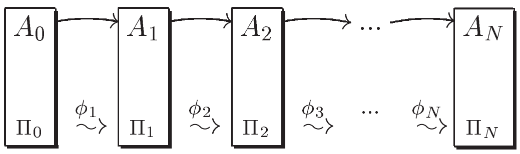


Figure 4.1: Dynamic argumentation: revising sets of premises.

Note that this description accounts for all the concepts in conventional argument frameworks, as identified by Prakken (Prakken 1995) and discussed in Chapter 3:

- an underlying logical language;
- a concept of argument;
- a concept of conflict between arguments;
- a notion of defeat among arguments; and
- an account of the acceptability status of arguments (and in this case, of theories).

Here, however, we have to consider one more notion:

- an account of attack-based revision.

Although we have characterised what properties a dynamic argument should have so that it generates acceptable theories, we have not solved the problem of actually generating them. Instead we have identified exactly the subproblems that need to be tackled:

1. First, we need to characterise the possible attacks at some point $i \geq 1$ in a dynamic argument, considering the moves $\langle A_0, \phi_1, A_1, ..., \phi_{i-1}, A_{i-1} \rangle$ advanced so far.
2. Second, we need to specify the set Φ of possible revision operations; i.e. how attacking arguments can be generated and how they relate to changes and revisions in a theory. Are there any desirable properties for Φ, and what would be their consequences? Is there a systematic way to define Φ?
3. Finally, we need to specify a mechanism for selecting which attack to generate. This selection mechanism is likely to be based on the set Φ (item 2 above) and on the characterisation of attacks (item 1 above).

In the rest of this part we will deal with the first two items, leaving the last—as well as the discussion about preference criteria—for Part III, where we consider control aspects of argument generation in automated dynamic argumentation systems.

Chapter 5

Towards a Classification of Argument Schemata

Chapter 4 gave a characterisation of argument dynamics as a sequence of arguments intended to defend positions from potential attacks, some of which may be put forward only if a structural revision is performed in the underlying theory. In this way we cannot assume that all premises used to generate arguments in a dynamic argument will be available from the start, as some can be added and others withdrawn during the course of the process.

So the aim of this chapter is to identify ways in which axioms in a theory can change as we advance new arguments. Based on examples and ideas from argumentation theory, we move towards a classification of argument schemata for relating the possible changes in a set of premises with the types of attacks we want to put forward. This classification will be used to describe the sorts of revision that characterise dynamic argumentation.

At this point we do not focus on choosing which claim or argument to attack. Instead, we want to explore systematically the types of theory revision that can be performed in order to generate an attack for a given claim. As might be expected, attacks can sometimes be generated from the current set of premises, in which case the theory need not be revised (or is *trivially* revised). However, because here we are interested in classifying *changes*, we can assume for the moment that attack-based revisions are non trivial.

The descriptions in this chapter are informal in order to illustrate the possible sorts of structural revision, but they will also serve to introduce the formal language that will be used in Chapter 7 to define the complete argument schemata classification. To make it easier to understand the idea behind each schema, we will follow the standard description pattern below:

- we first present an informal description of the schema;
- then we present a natural language argument as an example of the schema;
- finally we cast the example by means of the formal schemata description language.

The arguments used to illustrate the schemata are drawn directly from or based on policy debates about the possible carcinogenicity of chemical substances (McBurney and Parsons 1999). We take a close look at the *aflatoxin debate*, which has already been used for investigating argument-based risk assessment (Fox 1994; Robertson 1995) and conflict exploration (Haggith 1996). We set out the context for this debate in Section 5.1, before presenting examples of argument revision schemata in Section 5.2. Finally, in Section 5.3 we briefly discuss some concepts from informal argumentation that have founded the schemata presented here.

5.1 The Aflatoxin Debate: Assessing Cancer Risks

This example concerns a real debate about the carcinogenicity of certain chemical substances called aflatoxins, and about the FDA (US Food and Drugs Administration) policy that restricts aflatoxin levels to 20 parts per billion (ppb). The following are two arguments presented by Rodricks (1992)[1] for different standpoints concerning the question of whether the FDA's position is scientifically defensible.

> (1) Yes. The FDA clearly did the right thing, and perhaps did not go far enough. Aflatoxins are surely potent cancer-causing agents in animals. We don't have significant human data, but this is very hard to get and we shouldn't wait for it before we institute controls. We know from much study that animal testing gives a reliable indication of human risk. We also know that cancer-causing chemicals are

[1] As cited in (Fox 1994).

> a special breed of toxicants—they can threaten health at any level of intake. We should therefore eliminate human exposure to such agents whenever we can, and, at the least reduce exposure to the lowest possible level whenever we're not sure how to eliminate it.
>
> (2) No. The FDA went too far. Aflatoxins can indeed cause liver toxicity in animals and are also carcinogenic. But they produce these adverse effects only at levels far above the limit FDA set. We should ensure some safety margin to protect humans, but 20 ppb is unnecessarily low and the policy that there is no safe level is not supported by scientific studies. Indeed, it is not even certain that aflatoxins represent a cancer risk to humans because animal testing is not known to be a reliable predicator of human risk. Moreover, the carcinogenic potency of aflatoxins varies greatly among the several animal species in which they have been tested. Human evidence that aflatoxins cause cancer is unsubstantiated. There's no sound scientific basis for FDA's position.

The second paragraph gives some reasons for rejecting the argument supporting the FDA's position, which is essentially based on animal testing—or *bioassays.* As argued in (McBurney and Parsons 1999) bioassays are the most common sort of evidence supporting the possible carcinogenicity of a substance, and the authors have identified a number of different assumptions that must hold for this evidence to be considered valid. For instance, to claim that a certain chemical is carcinogenic on the basis of a bioassay on an animal species, the animal physiology and chemistry relevant to the activity of this chemical must be sufficiently similar to human physiology and chemistry.

What we want to illustrate in this chapter is that there might exist standard ways for advancing attacks (e.g. those in paragraph 2) that are based on the structure of the argument being attacked (e.g. the argument in paragraph 1) and which can be instantiated by domain-specific expertise (e.g. the assumptions identified by McBurney and Parsons (1999)). Not all example arguments we present are an accurate reproduction of the aflatoxin debate as stated by Rodricks (1992), as we might alter or introduce information for illustrative purposes only.

In what follows, sets of beliefs related to the aflatoxin debate will be expressed as general logic programs.[2] As expected, axioms (clauses) will be fundamentally arguable, as they represent the essentials of a problem rather than universal truths.

5.2 Argument Schemata for Arguing about Aflatoxins

We now illustrate the use of argument schemata with some examples from the aflatoxin debate. Schemata are used for generating arguments and attacks, specified in terms of revision operations as defined in Section 4.2. Here we depict schemata built upon *elementary* revisions (Sections 5.2.2 and 5.2.3) and upon *updating* revisions, i.e. those composed of two elementary operations and used for updating an axiom by retracting it and subsequently adding a modified version (Sections 5.2.4). First we give a general account of the types of schema we consider and the language used for describing these.

5.2.1 An Overview of the Schemata Description Language

When describing argument schemata we want to represent not only the changes to be performed to the knowledge base, but also the reasons why we can perform them. By looking at concepts studied in argumentation theory—such as *argumentation schemes* and *fallacies*[3]—we have identified a number of possible reasons and motivations for adding, changing and adapting premises in an argument. Here we make use of a formal description language to capture and represent a subset of these, which we feel is relevant to the types of argument in which we are interested.

For instance, when we add a new premise to the theory we might want to say that we are introducing a new *fact*, i.e. something that is taken to be true. In case we are adding a new rule, then we can also specify whether it is a *substantiated rule* for yielding new conclusions, or a *burden shift rule* for reversing the burden of proof.

It should also be possible in this language to represent the reasons for updating and altering premises. We can, for instance, change an axiom in a theory because it should be *specialised*, or *generalised*. Or else, we can replace it with a more *elaborated* version,

[2] See Appendix A for a concise account of logic programming concepts and syntax.

[3] See discussion in Section 5.3.

with extra preconditions; or with a less elaborated version obtained by removing some precondition that is thought to be *irrelevant.* Furthermore, we could *revise the conclusion* of a rule, or *reverse* the relation between the consequent and the antecedent. These descriptions convey the possible reasons for altering and replacing axioms.

It is often the case, though, that these language constructs only summarise what could be guessed from the structure of the updated or added premises—i.e. from the revision operation itself that is associated with the schema. But changes that are different in nature may sometimes yield identical instances of schemata based on identical revision operations. In these cases, such a description language allows us to keep and represent the original distinction.

This is particularly true when we remove premises from a theory. We may have a number of different reasons for withdrawing a premise, but the type of revision associated with these will always be syntactically equivalent. Thus in our representation we use different constructs to distinguish between different reasons for retracting an axiom from a theory, either because it is an *invalid rule*, a *weak rule* or a *misrelation.* The difference is briefly discussed below:

Invalid rule. A rule can be considered to be invalid if there are exceptions to it—cases where the antecedent holds but the consequent does not.

Weak rule. A rule can be considered to be weak if there are instances where the antecedent does not hold, affecting the generality of the relation.

Misrelation. The relation expressed by an axiom is said to be mistaken if there are cases where the antecedent holds and the consequent does not, and instances where the antecedent does not hold but the consequent does, thus compromising the adequacy of the correlation between antecedent and consequent.

Note that we do not require these conditions to be necessarily valid when we apply the corresponding revisions. However, they provide designers with extra information which could be useful in defining domain-specific cases for theory revision.

The terms discussed in this section constitute part of the language we use for describing argument schemata, which we illustrate in the next sections and formally define in Chapter 7.

5.2.2 Adding a New Premise

In this section we look at schemata for deriving new arguments by adding a new axiom to the theory. Added clauses are diagrammatically represented within light gray boxes.

Informal Schema 1 (Adding a New Fact) *A trivial way to present an argument for a sentence is by adding it as a fact in the theory, as facts immediately follow from the theory.*

This is particularly useful if the sentence corresponds to an observation, or to a belief that is taken to be categorically true. For instance, to advance the following argument:

> Aflatoxins are surely potent cancer-causing agents in animals.

it is enough to add it as a fact in the theory, justified by direct observation. Let the sentence $causes(aflatoxin, cancer, animal(X))$ represent the statement above, where $animal(X)$ denotes that X is a non-human animal species. The type of revision necessary for justifying this sentence is depicted below,[4]

$$\{\} \overset{add(fact)}{\rightsquigarrow} \boxed{\texttt{causes(aflatoxin, cancer, animal(X))}}$$

and is represented by the following instantiated schema:

$justify(causes(aflatoxin, cancer, animal(X)))$ if
$add(fact\left(\begin{array}{l} causes(aflatoxin, cancer, animal(X)) \leftarrow \\ \quad true \end{array}\right))$

The following trivial argument can now be derived:

$$\{causes(aflatoxin, cancer, animal(X)) \leftarrow true\} \vdash causes(aflatoxin, cancer, animal(X)) \tag{5.1}$$

[4] For reasons of clarity and space, in the revision diagrams in this section we denote facts of the form $H \leftarrow true$ by the sole expression `H` in Prolog style.

This sort of argument is often regarded as a fallacy in argumentation theory, namely *begging the question* or *circular reasoning.* Although logically sound, it is also "trivially uninteresting" (Fogelin and Sinnott-Armstrong 1997, p. 40), and in our example cannot be considered as a proof that aflatoxins cause cancer in animals.

However, such arguments are indeed valid. It might be uninteresting in this case, but can be useful for generating arguments from more complicated schemata based on complex types of revisions.

Informal Schema 2 (Adding a New Substantiated Rule) *We can justify a sentence by adding a new rule for deriving it such that the rule antecedent is supported.*

For instance, we can advance the following argument supporting the claim that aflatoxins cause cancer in humans.

> Aflatoxins are surely potent cancer-causing agents in animals. We know from much study that animal testing gives a reliable indication of human risk.

So, for this claim to be derived we can add to the theory a rule stating that all agents that cause some pathology in some non-human animal species would cause this pathology in humans. This is a substantiated rule for the case of aflatoxins because its antecedent is satisfied by the *fact* (in the theory) that aflatoxins cause cancer in non-human species. This type of revision is depicted below,

causes(aflatoxin, cancer, animal(X))	$\overset{add(substantiated\_rule)}{\rightsquigarrow}$	causes(aflatoxin, cancer, animal(X)) causes(A, P, human) ← causes(A, P, animal(X))

and is represented by the following instantiated schema:

$$\frac{justify(causes(aflatoxin, cancer, human)) \text{ if}}{add(substantiated\_rule\left(\begin{array}{l} causes(A, P, human) \leftarrow \\ \quad causes(A, P, animal(X)) \end{array}\right))}$$

This rule may not be an universal truth, but it captures the general nature of the domain we are representing. The following argument can now be derived:

$$\begin{array}{c} causes(aflatoxin, cancer, human) \\ \Big| \; {\scriptstyle causes(A,P,human) \leftarrow causes(A,P,animal(X))} \\ causes(aflatoxin, cancer, animal(X)) \end{array} \tag{5.2}$$

Informal Schema 3 (Adding a Burden Shift Rule) *We can shift the burden of proof by adding a rule stating that a sentence is justified if some other (opposing) sentence is not. In this way, we justify a sentence by arguing that its contrary cannot be supported.*

For instance, we can put forward the following argument for sustaining the claim that there is no safe level of exposure for carcinogenic agents.

> We can assume that there is no safe exposure level for an agent unless one can scientifically prove that there is a safe level of exposure for this agent at which it will not cause cancer.

This argument can be derived if we add a general rule stating that there is no safe exposure level for a cancer-causing agent if we cannot justify the existence of a safe level for it. Given that we cannot prove that there is a safe level for aflatoxins, this rule shifts the burden of proof to someone willing to prove that such a level does exist. This type of revision is depicted below,

```
causes(aflatoxin, cancer, animal(X))
causes(A, P, human) ←
     causes(A, P, animal(X))
```

$add(burden\_shift\_rule) \rightsquigarrow$

```
causes(aflatoxin, cancer, animal(X))
causes(A, P, human) ←
     causes(A, P, animal(X))
no_safe_level(A) ←
     not safe_level(A, L)
```

and is represented by the following instantiated schema:

$$\begin{array}{|l|} \hline justify(no\_safe\_level(aflatoxin)) \text{ if} \\ \hline \quad add(burden\_shift\_rule\left(\begin{array}{l} no\_safe\_level(A) \leftarrow \\ \quad not\ safe\_level(A, L) \end{array}\right)) \\ \hline \end{array}$$

Hence the argument below can be derived, supporting the claim that there is no safe exposure level for aflatoxins.

$$\begin{array}{c} no\_safe\_level(aflatoxin) \\ \Big| \scriptstyle no\_safe\_level(A) \leftarrow not\ safe\_level(A,L) \\ not\ safe\_level(aflatoxin, L) \end{array} \tag{5.3}$$

Shifting the burden of proof is sometimes regarded as a fallacy, namely *appeal to ignorance*, whereby a claim is said to be true because there is no evidence that it is false. This type of reasoning, however, can also be used non fallaciously in certain problems and domains.

5.2.3 Retracting an Existing Premise

Revisions in this section are concerned with the quality of the premises used in an argument, in particular with the quality of rules. We focus on rules rather than facts and propositions because the general way for challenging and refuting a proposition is to justify some opposing or contradicting position (i.e. to present a counter-argument). In the case of rules, however, the qualification problem states that it is not always possible to explicitly account for the many conditions necessary for rules to hold, so it is important to investigate whether a rule is in fact germane to the problem in question.

What is interesting about retraction is that it brings into play more of the dynamics of argumentation as opposed to the usual approach of only adding arguments which overcome the weak ones. That allows for instance for previous arguments to be not only defeated but invalidated, e.g. for being fallacious.

As discussed in Section 5.2.1, there may be different reasons for rejecting an axiom, and now we look more closely at some of these ways through which we can withdraw a rule and challenge its validity. Removed clauses are diagrammatically represented within dark gray boxes.

Informal Schema 4 (Retracting an Invalid Rule) *We can refute an argument because the conditional used to derive the argument claim is logically invalid, i.e. there are exceptions to it (cases for which the antecedent holds but the consequent does not).*

For instance, the argument below refutes argument 5.2, suggesting that the claim that aflatoxins cause cancer in humans is unsubstantiated.

> It's not even certain that aflatoxins represent a cancer risk to humans because animal testing is not known to be a reliable predictor of human risk.

This argument rejects the rule that relates animal testing and human risk by questioning its reliability, e.g. because there might be exceptions to this relation (cases of an specific agent known to cause some specific pathology in some animal species, and not causing the same pathology in humans). This type of revision is depicted below,

```
causes(aflatoxin, cancer, animal(X))
causes(A, P, human) ←
        causes(A, P, animal(X))
```

$$\overset{retract(invalid\_rule)}{\rightsquigarrow}$$

```
causes(aflatoxin, cancer, animal(X))
```

and is represented by the following instantiated schema:

$$\begin{array}{l} \underline{refute(causes(aflatoxin, cancer, human))\ \text{if}} \\ \quad retract\left(invalid\_rule\left(\begin{array}{l} causes(A, P, human) \leftarrow \\ \quad causes(A, P, animal(X)) \end{array}\right)\right) \end{array}$$

In this way, argument 5.2 is no longer derivable from the revised set of premises.

Informal Schema 5 (Retracting a Weak Rule) *We can refute an argument because the conditional used to derive the argument claim is logically weak, i.e. there are cases for which the antecedent does not hold, compromising the generality of the relation.*

Let us consider the following argument:

> It's not even certain that aflatoxins represent a cancer risk to humans because animal testing is not known to be a reliable predictor of human risk. Moreover, the carcinogenic potency of aflatoxins varies greatly among the several animal species in which they have been tested.

Again, this argument rejects the rule that relates animal testing and human risk by questioning its reliability. This challenge may not be grounded on explicit denials like in the previous schema, but on weakening the generality and relevance of this relation.

For instance, by presenting cases where the antecedent does not hold, or a particular animal species to which aflatoxins are not carcinogenic (in line with the assertion that the carcinogenic potency of aflatoxins differs among species). This type of revision is depicted below,

causes(aflatoxin, cancer, animal(X)) causes(A, P, human) ← causes(A, P, animal(X))	$\overset{retract(weak\_rule)}{\rightsquigarrow}$	causes(aflatoxin, cancer, animal(X))

and is represented by the following instantiated schema:

$$\frac{refute(causes(aflatoxin, cancer, human)) \text{ if}}{retract(weak\_rule\left(\begin{array}{l} causes(A, P, human) \leftarrow \\ \quad causes(A, P, animal(X)) \end{array}\right))}$$

Note that this is identical to the instance of Schema 4, the only distinction being the reason for retracting the rule, captured in this representation by the different constructs *invalid_rule* and *weak_rule*. And again, argument 5.2 is no longer supported in the revised set of premises, in which case the claim that aflatoxins cause cancer in humans is unsubstantiated.

Informal Schema 6 (Retracting a Misrelation) *We can refute an argument because the correlation expressed by the rule used to derive the argument claim is mistaken, i.e. the correlation between antecedent and consequent is not adequate.*

Let us consider again the following argument:

> It's not even certain that aflatoxins represent a cancer risk to humans because animal testing is not known to be a reliable predictor of human risk.

This time we could challenge the reliability of the relation between human risk and animal testing on the basis that this relation is mistaken, e.g. because the consequent is not very likely to follow from the antecedent, or simply because there is no correlation at all between the sentences (a particular agent is known to cause some pathology in an animal species but not in humans, and some other agent is known to cause a different pathology in humans but not in certain animal species). Such argument then undermines the general extrapolation of animal risk to human risk. This type of revision is depicted below,

```
causes(aflatoxin, cancer, animal(X))
causes(A, P, human) ←
        causes(A, P, animal(X))
```

$$\overset{retract(misrelation)}{\rightsquigarrow}$$

```
causes(aflatoxin, cancer, animal(X))
```

and is represented by the following instantiated schema:

$$\frac{refute(causes(aflatoxin, cancer, human)) \text{ if}}{retract(misrelation\left(\begin{array}{l} causes(A, P, human) \leftarrow \\ \quad causes(A, P, animal(X)) \end{array}\right))}$$

This argument once again refutes argument 5.2, in which case the claim that aflatoxins cause cancer in humans is again unsupported.

5.2.4 Updating an Existing Premise

It it not always the case that a challenged rule needs to be retracted for good. In fact, according to the qualification problem, it is hard (if not impossible) to specify all the preconditions for a rule to hold, as there might be implicit or unknown conditions that can invalidate the relation. So we can refute a rule by retracting it, and subsequently adding an updated version that accounts for some of these implicit or unknown conditions. In the same way, not all conditions in a rule may be pertinent to the problem we are representing, so we can revise the rule again by dismissing such irrelevant premises.

In this section we look at examples where new arguments are generated on the basis of revised rules. Notice that we can revise an axiom not only to refute an existing argument that is based on it, but also to introduce a new argument that makes use of the updated axiom in order to be inferred. After all, revision is also about strengthening an argument by reviewing the axioms that support it.

Informal Schema 7 (Removing Irrelevance in a Rule) *We can justify a sentence by removing an irrelevant precondition from a rule so that the rule antecedent is now satisfied, and the sentence consequently follows from it.*

For instance, the argument below supports the claim that the maximum required level of intake for aflatoxins should be set to its minimum detectable level, i.e. 20 parts per billion, on the basis that such substances are carcinogenic.

> We know that cancer-causing chemicals are a special breed of toxicants—they can threaten health at any level of intake. We should therefore eliminate human exposure to such agents whenever we can, and, at the least reduce exposure to the lowest possible level whenever we're not sure how to eliminate it. The level of intake for carcinogenic substances should always be restricted, even it is argued that a safe level of intake exists which is far above the minimum detectable level.

Suppose that our theory about carcinogenicity of substances already contains a rule stating that the required level of an agent should be set to its minimum detectable level if it is carcinogenic and if there is no known safe exposure level for it. However, if the theory also contains a fact stating that a safe exposure level s for aflatoxins does exist which is far greater than the minimal detectable level, then the rule above cannot be used as not all its preconditions are satisfied. What we argue, though, is that one can never be too cautious when dealing with carcinogenic substances. So the required level for aflatoxins should still be set to their minimum detectable level, because we must disregard any conditions about safe exposure levels for an agent that can cause cancer. This type of revision is depicted below,

```
causes(aflatoxin, cancer, animal(X))
min_det_level(aflatoxin, 20ppb)
safe_level(aflatoxin, s)
no_safe_level(A) ←
     not safe_level(A, L)
required_level(A, L) ←
        causes(A, cancer, X) ∧
        no_safe_level(A) ∧
        min_det_level(A, L)
```

$\underset{\leadsto}{update(irrelevance)}$

```
causes(aflatoxin, cancer, animal(X))
min_det_level(aflatoxin, 20ppb)
safe_level(aflatoxin, s)
no_safe_level(A) ←
     not safe_level(A, L)
required_level(A, L) ←
        causes(A, cancer, X) ∧
        min_det_level(A, L)
```

and is represented by the following instantiated schema:

$justify(required\_level(aflatoxin, 20ppb)$ if

$$retract(irrelevance\begin{pmatrix} required\_level(A,L) \leftarrow \\ \quad causes(A, cancer, X) \wedge \\ \quad no\_safe\_level(A) \wedge \\ \quad min\_det\_level(A,L) \end{pmatrix})$$

and

$$add(irrelevance\begin{pmatrix} required\_level(A,L) \leftarrow \\ \quad causes(A, cancer, X) \wedge \\ \quad min\_det\_level(A,L) \end{pmatrix})$$

The argument below can then be derived:

$$\begin{array}{c} required\_level(aflatoxin, 20ppb) \\ \scriptstyle required\_level(A,L) \leftarrow causes(A,cancer,Y) \wedge min\_det\_level(A,L) \\ causes(A, cancer, animal(X)) \qquad min\_det\_level(aflatoxin, 20ppb) \end{array} \tag{5.4}$$

Informal Schema 8 (Revising the Consequent of a Rule) *We can revise the consequent of a rule if it does not correspond to what is expected to follow from the preconditions of this rule. This revision can either allow a new argument for a sentence to be derived (if this sentence is now the revised consequent) or refute an existing argument for a sentence (if this sentence was the consequent of the original rule).*

Assume that in our current theory the claim that a safe level of intake for aflatoxins exists is based on a rule stating that the minimum detectable level for a carcinogenic substance is safe, in the sense that it will not cause adverse effects. We can present the following argument for refuting this conclusion.

> We know that cancer-causing chemicals are a special breed of toxicants—they can threaten health at any level of intake. We should therefore eliminate human exposure to such agents whenever we can, and, at the least reduce exposure to the lowest possible level whenever we're not sure how to eliminate it.

In the current theory, we are inferring the wrong conclusion from the right premises. The minimum detectable level of a carcinogenic substance should never be regarded as safe, but as *the best we can do* to eliminate risk (i.e. the maximum acceptable level). This type of revision is depicted below,

```
causes(aflatoxin, cancer, animal(X))
min_det_level(aflatoxin, 20ppb)
safe_level(A, L) ←
        causes(A, cancer, X) ∧
        min_det_level(A, L)
```

$update(misconclusion) \rightsquigarrow$

```
causes(aflatoxin, cancer, animal(X))
min_det_level(aflatoxin, 20ppb)
required_level(A, L) ←
        causes(A, cancer, X) ∧
        min_det_level(A, L)
```

and is represented by the following instantiated schema:

$$\begin{array}{l} refute(safe\_level(aflatoxin, cancer, 20ppb)) \text{ if} \\ \hline \quad retract(misconclusion\left(\begin{array}{l} safe\_level(A, L) \leftarrow \\ \quad causes(A, cancer, X) \wedge \\ \quad min\_det\_level(A, L) \end{array}\right)) \\ \quad \text{and} \\ \quad add(misconclusion\left(\begin{array}{l} required\_level(A, L) \leftarrow \\ \quad causes(A, cancer, X) \wedge \\ \quad min\_det\_level(A, L) \end{array}\right)) \end{array}$$

Informal Schema 9 (Reversing a Rule) *We can invert a rule when the relation between its antecedent and consequent is reversed. This revision can either allow a new argument for a sentence to be derived (the antecedent of the original rule, which is now the consequent of the updated rule) or refute an existing argument for a sentence (the consequent of the original rule, which is now the antecedent of the updated rule).*

Suppose that we still wanted to argue that there is a safe level of exposure for aflatoxins, and that we had done so by introducing a (new substantiated) rule stating that the required level of exposure for an agent is actually a safe exposure level. Below is a counter argument that blocks the safe level conclusion from being derived in this case.

> It is not the case that the maximum acceptable level of intake for a carcinogenic substance is necessarily safe. In fact, it should be restricted by a safe exposure level, if such safe level can ever be proven to exist.

In this way the conclusion that a safe level of intake for aflatoxins exists is no longer supported, as the rule used to derive it can no longer be applied. This type of revision is depicted below,

```
causes(aflatoxin, cancer, animal(X))
min_det_level(aflatoxin, 20ppb)
required_level(A, L) ←
     causes(A, cancer, X) ∧
     min_det_level(A, L)
safe_level(A, L) ←
        required_level(A, L)
```

$\underset{\leadsto}{update(reversion)}$

```
causes(aflatoxin, cancer, animal(X))
min_det_level(aflatoxin, 20ppb)
required_level(A, L) ←
     causes(A, cancer, X) ∧
     min_det_level(A, L)
required_level(A, L) ←
        safe_level(A, L)
```

and is represented by the following instantiated schema:

$refute(safe\_level(aflatoxin, 20ppb))$ if

$$retract(reversion\left(\begin{array}{c} safe\_level(A, L) \leftarrow \\ \quad required\_level(A, L) \end{array}\right))$$

and

$$add(reversion\left(\begin{array}{c} required\_level(A, L) \leftarrow \\ safe\_level(A, L) \end{array}\right))$$

The last two argument schemata were used to block the claim that the minimum detectable level of aflatoxin is a safe exposure level for it. Notice, however, that these are not intended to reject the required level of exposure from being set to this minimum level. Instead, these schemata convey the idea that no safe level of intake for a carcinogenic agent can ever exist, i.e. that cancer-causing substances "can threaten health health at any level of intake."

Informal Schema 10 (Specialising a Rule) *One way to refute an argument is by specialising the rule used to derive the argument claim so that it is no longer applicable to the case under discussion.*

For instance, we can refute argument 5.4 for $required\_level(aflatoxin, 20ppb)$ by advancing the following argument.

> We should not restrict the level of aflatoxin intake to its minimum detectable level unless it is known that aflatoxins cause cancer in humans. In fact, aflatoxins can cause liver toxicity in animals and are also carcinogenic, but it is not even certain that they represent a cancer risk to humans because animal testing is not known to be a reliable predicator of human risk.

The idea behind this argument is that the rule for restricting the level of intake is too general, and should only be applied if an agent is known to be carcinogenic to humans in particular.

This type of revision is depicted below,

```
causes(aflatoxin, cancer, animal(X))
min_det_level(aflatoxin, 20ppb)
required_level(A, L) ←
        causes(A, cancer, X) ∧
        min_det_level(A, L)
```

$update(specialisation)$ $\leadsto$

```
causes(aflatoxin, cancer, animal(X))
min_det_level(aflatoxin, 20ppb)
required_level(A, L) ←
        causes(A, cancer, human) ∧
        min_det_level(A, L)
```

and is represented by the following instantiated schema:

$refute(required\_level(aflatoxin, 20ppb))$ if

$$retract(specialisation \begin{pmatrix} required\_level(A, L) \leftarrow \\ \quad causes(A, cancer, X) \wedge \\ \quad min\_det\_level(A, L) \end{pmatrix})$$

and

$$add(specialisation \begin{pmatrix} required\_level(A, L) \leftarrow \\ \quad causes(A, cancer, human) \wedge \\ \quad min\_det\_level(A, L) \end{pmatrix})$$

In this way, we can no longer derive an argument for $required\_level(aflatoxin, 20ppb)$ in the revised theory.

Informal Schema 11 (Generalising a Rule) *We can justify a sentence by generalising some existing rule so that it now allows this sentence to be derived.*

Suppose we have previously observed that aflatoxins cause cancer in rabbits, and now we want to use this fact to support the claim that aflatoxins are carcinogenic to humans on the basis of a bioassay evidence.

> Aflatoxins are potent cancer-causing substances in rabbits, and we know that animal testing gives a reliable indication of human risk.

Suppose also that the current theory about carcinogenicity of chemical substances contains a rule stating that a chemical agent causes some pathology in humans if it causes this pathology in mice. The existing extrapolation rule cannot be applied because it is specific to the case of mice, so to advance the argument above we need to generalise it. This type of revision is depicted below,

`causes(aflatoxin, cancer, animal(rabbit))`
`causes(A, P, human) ←`
`    causes(A, P, animal(mouse))`
	$\overset{update(generalisation)}{\leadsto}$	`causes(aflatoxin, cancer, animal(rabbit))`
		`causes(A, P, human) ←`
		`    causes(A, P, animal(X))`

and is represented by the following instantiated schema:

$$\boxed{\begin{array}{l} \underline{justify(causes(aflatoxin, cancer, human))\ \text{if}} \\ \quad retract(generalisation\left(\begin{array}{l} causes(A, P, human) \leftarrow \\ \quad causes(A, P, animal(mouse)) \end{array}\right)) \\ \quad \text{and} \\ \quad add(generalisation\left(\begin{array}{l} causes(A, P, human) \leftarrow \\ \quad causes(A, P, animal(X)) \end{array}\right)) \end{array}}$$

The argument below can then be derived:

$$\begin{array}{c} causes(aflatoxin, cancer, human) \\ \Big| {\scriptstyle causes(A,P,human) \leftarrow causes(A,P,animal(X))} \\ causes(aflatoxin, cancer, animal(rabbit)) \end{array} \tag{5.5}$$

Informal Schema 12 (Elaborating Preconditions in a Rule) *One way to refute an argument is by elaborating the preconditions in the rule used to derive the argument claim so that its antecedent is no longer satisfied.*

The following is an argument that refutes argument 5.2 (or similarly, argument 5.5) by elaborating on the conditions for applying the general extrapolation rule.

> To claim that an agent is carcinogenic on the basis of animal testing, the animal physiology and chemistry relevant to the activity of this agent must be sufficiently similar to human physiology and chemistry.

At this point, the claim that aflatoxins cause cancer in humans is unsubstantiated because there is no indication of whether the type of animal that is considered is in fact similar enough to humans. This type of revision is depicted below,

```
causes(aflatoxin, cancer, animal(X))
causes(A, P, human) ←
        causes(A, P, animal(X))
```

$\underset{\leadsto}{update(elaboration)}$

```
causes(aflatoxin, cancer, animal(X))
causes(A, P, human) ←
        causes(A, P, animal(X)) ∧
        similar_physiology(human, X)
```

and is represented by the following instantiated schema:

$$\frac{refute(causes(aflatoxin, cancer, human))\ \text{if}}{\begin{array}{l} retract(elaboration\left(\begin{array}{l} causes(A, P, human) \leftarrow \\ \quad causes(A, P, animal(X)) \end{array}\right)) \\ \text{and} \\ add(elaboration\left(\begin{array}{l} causes(A, P, human) \leftarrow \\ \quad causes(A, P, animal(X)) \wedge \\ \quad similar\_physiology(human, X)) \end{array}\right)) \end{array}}$$

Note that this sort of refutation is not as damaging as those discussed in the previous section. In fact, to reestablish the conclusion that aflatoxins cause cancer in humans, we just need to explicitly account that the animal used in the bioassay was sufficiently similar to humans in what matters. Furthermore, this elaborated rule may be better protected from the types of refutations in Section 5.2.3, as it better specifies what has to be considered as relevant in this domain.

5.3 Relationship with Informal Argumentation Theory

The schemata illustrated here all required some sort of domain-specific expertise to be instantiated in a relevant way. But notice that we do not want to focus on a domain-specific solution. Our point is that an analysis of formal argument structure can shed some light on how justifications and refutations are generated in any particular domain, thus providing *rough sketches* to which domain-specific knowledge can be applied.

To define a classification of schemata we have then looked into traditional argumentation theory. In fact, one of the main problem areas in the study of informal logic consists in identifying, analysing and evaluating arguments (van Eemeren et al. 1996), so argumentation theorists are often interested in developing models and tools for supporting these tasks. One example is the notion of *argumentation schemes* (Walton 1996).

As argued by van Eemeren et al. (1996), argumentation schemes are concerned with the internal structure of arguments, and "to the kind of relation established in a single argument between its premises and the standpoint the argument aims to justify or refute." In summary, they are used for classifying and modelling various types of argument forms. Their use dates back to Aristotle, who discussed the idea of argumentation schemes (or techniques, or moves) being selected and instantiated by an attacker during dialectical debates (van Eemeren et al. 1996, p.38).

More recent approaches (Walton 1996; Perelman and Olbrects-Tyteca 1969) propose lists and catalogues of argumentation schemes that represent acceptable ways for connecting premises and conclusions. The conclusion of a schema is then said to be *presumptively* (or *defeasibly*) valid if the associated premises and conditions hold. In the New Rhetoric (Perelman and Olbrects-Tyteca 1969)[5], for instance, schemes represent logical as well as rhetorical arguments, and characterise inference mechanisms that can be used to convince an audience in persuasive argumentation. Furthermore, *critical questions* are asked in relation to an argumentation schema to determine whether it can in fact be applied.

In our case, however, we adopt a slightly different position. We want to define generic structures of logical arguments rather than different types of inference linkages. This is because our arguments are generated based on a logical system, and on a formal and sound logical inference mechanism. So defeasibility is related not with the *reasoning* step but with the types of *premises* that can be used, added, removed or updated.

In the cases where premises are removed or updated, we have looked at the idea of fallacies, i.e. arguments that appear to be valid but are actually not. The study of fallacies constitutes another major area in argumentation theory, which provided us with rich material for analysing the quality of premises in an argument, and for indicating when these were not really well-grounded. Below we describe some informal fallacies

[5] As cited by van Eemeren et al. (1996) and by Warnick and Kline (1992).

that we have considered and identified as being relevant to our analysis, relating them to the schemata in the previous sections. In particular, we refer to possible revision-based schemata that could have been applied in order to improve the quality of the fallacious argument. The literature on fallacies is vast, and we have based our descriptions mainly on general resources such as (van Eemeren et al. 1996) and (Fogelin and Sinnott-Armstrong 1997).

Slippery Slope. When a claim is said to be caused by a sequence of events, but there is not enough evidence of such relationship.

In this case, the rule representing this relationship may be removed for being invalid **(Informal Schema 4)***, weak* **(Informal Schema 5)***, or mistaken* **(Informal Schema 6)***; or its conclusion may be revised* **(Informal Schema 8)***.*

False cause. When there is not enough evidence that one event caused another.

Similarly to the case above, we can apply **Informal Schemas 4, 5, 6, 8***.*

Hasty conclusion. When we jump to a conclusion not based on enough grounds.

As above, here we could apply **Informal Schemas 4, 5, 6, 8***.*

False criteria. When false or irrelevant criteria are used in the argument.

In this case, **Informal Schema 7** *can be used to disregard this criteria.*

Wrong direction. When the relation between cause and effect is reversed.

Here **Informal Schema 9** *could be applied to reverse the relation.*

Hasty generalisation. When the generalisation is not based on enough cases or samples.

In this case, we could apply **Informal Schema 10** *to specialise the rule.*

Composition. When a property that is valid for a part is assumed to be valid for the whole entity containing it.

Again, **Informal Schema 10** *could be applied to specialise the relation to consider the part, and not the whole.*

Division. When a property that is valid for a whole entity is assumed to be valid for each of its parts.

Again, **Informal Schema 11** *could be applied to generalise the relation to consider the whole, and not the parts.*

Complex Cause. When the cause identified is simpler than the actual cause of the effect.

In this case, **Informal Schema 12** *could be used to elaborate the relation and introduce other relevant conditions.*

By looking at existing accounts from informal argumentation theory, we were then able to combine domain-independent knowledge about arguments to describe general logical forms of arguments and attacks in terms of the premises used. The next chapters formally describe a classification of argument schemata and discuss some of its properties. To these generic structures we can then apply domain-specific knowledge so that we instantiate and determine the contents of an attack to be advanced in an dynamic argumentation process.

First, though, we need to describe exactly how arguments attack each other.

Chapter 6

Attacks in Argument Dynamics

Chapter 4 identified precisely the problems we need to address in order to fully describe and generate dynamic arguments. This chapter considers one of those problems, namely how to characterise the general format of attacks and the possible contradictions in argument.

6.1 Types of Argument Claims

According to Definition 4.4, an argument A' attacks an argument A if and only if A' contradicts a claim supported by A and A is not preferred over A'. For the moment we shall assume that no preference criterion is defined, thus no argument is preferred over any other. We return to the topic of argument prioritisation later in Part III of this thesis.

To characterise the types of attack to an argument we then need to identify what are the claims supported by this argument and how these can be contradicted. In Chapter 4 we have referred to a *claim* as being the conclusion of a justification,[1] but here we take the view that claims are general statements (about sentences in the language) supported by arguments in general. If, for example, an argument A is a justification for φ in a theory Π, then based on A we can say that φ *is substantiated in* Π.

Whereas a justification can serve as a reason for accepting a sentence, other types of arguments—such as counter-justifications and refutations—can be used for rejecting a

[1] See Definition 4.3.

justification and consequently its conclusion. Notice that it only makes sense to talk about these in connection with some previously constructed justification, and not as individual entities. While counter-arguments are essentially justifications supporting a sentence that conflicts with some point of the original argument, refutations are used for blocking conclusions from being derived. That is, refutations are used for rejecting a premise (axiom) in a justification, either by removing it from the theory or by updating it so that the argument no longer follows. Thus refutations are logically valid but not sound, because they contain axioms not considered to be sound with respect to the theory in question. If an argument A is a refutation of a justification for φ in a theory Π, then based on A we can say that φ *is not substantiated.*

The following definition summarises these notions.

Definition 6.1 (Types of Claims) *Let A be an argument about φ in a theory Π. There are two cases to be considered:*

- *A is a justification $\Gamma \vdash_{\Pi} \varphi$*

  *Then A supports the claim that φ is substantiated in Π—i.e that φ is in the set of consequences of Π. We denote this by φ : **in**.*

- *A is a refutation[2] $\Gamma \not\vdash_{\Pi} \varphi$*

  *Then A supports the claim that φ is unsubstantiated in Π—i.e. that φ is not in the set of consequences of Π, <u>at least with respect to A</u>. We denote this by φ : **out**.*

$\square$

So claims are sentences annotated with labels **in** and **out**, which indicate whether the sentence is acceptable or not in the theory with respect to the argument in question. By adopting this notation the connection with truth maintenance systems draws even closer: sentences in a TMS are said to be *in* if they have at least one currently acceptable (valid) reason, and are said to be *out* otherwise (Doyle 1979). We shall be discussing points of contact between argumentation and TMS throughout this chapter before looking at this relationship more carefully in Section 6.4.

[2] $\Gamma \not\vdash_{\Pi} \varphi$ is a refutation of $\Gamma \vdash \varphi$ in Π if $\Gamma \vdash_{\Pi_p} \varphi$ is a justification of φ in some previous theory Π_p, and Π is obtained from Π_p by retracting some premise from $\Gamma \subseteq \Pi_p$.

Some important points need to be made about this, in relation to the discussion about truth and acceptability in Section 1.1.1. As argued by Doyle (1979, p. 238):

> The distinction between *in* and *out* is not that between *true* and *false*. The former classification refers to current possession of valid reasons for belief. *True* and *false*, on the other hand, classify statements according to truth value independent of any reasons for belief.

This distinction also holds for labels **in** and **out** in Definition 6.1. To say that a sentence φ is **out** of the set of consequences of a theory with respect to an argument (refutation) *is not equivalent* to saying that φ is not a consequence of the theory. The reason why is that we have taken a computational view where arguments may exist in a theory but may not yet have been found, thus labels give the status of sentences in relation to the arguments that were computed and presented so far. Although we have rejected a justification for φ, there is no guarantee of whether some alternative justification for it exists, in which case φ would in fact be a consequence of the theory. What we guarantee is that there is one less way of inferring the sentence within the theory, but that does not mean that its set of consequences is smaller.

6.1.1 Claim Dependencies in an Argument

Definition 6.1 gives the sorts of statements that can be made about an argument main conclusion, or *main claim*. As arguments are structured objects composed of sub-arguments, it should also be possible to make statements—or *indirect claims*—about the sub-conclusions underpinning the main claim, and to say things such as *a sentence is substantiated because it is based on other sentences which are themselves substantiated.* To capture these dependencies, claims supported by an argument are represented in a directed graph obtained from the corresponding argument tree. Appendix B gives the basic notation used in this thesis for expressing argument trees and directed graphs.

The following example illustrates this notion.

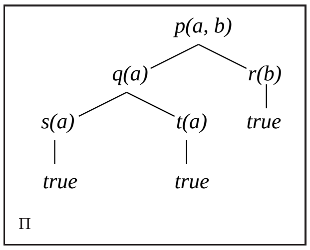
Example 6.1 *Let* Π *be the theory below in a Horn clause resolution-based language.*

$$
\begin{array}{rcl}
p(X,Y) & \leftarrow & q(X) \wedge\ r(Y) \\
q(X) & \leftarrow & s(X) \wedge\ t(X) \\
r(b) & \leftarrow & true \\
s(a) & \leftarrow & true \\
t(a) & \leftarrow & true
\end{array}
$$

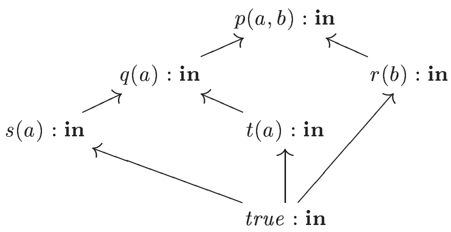
Then the argument A below is a justification for $p(a,b)$ in Π.

$$\{p(X,Y) \leftarrow q(X)\ \wedge\ r(Y),\ q(X) \leftarrow s(X)\ \wedge\ t(X),\ s(a) \leftarrow true,\ t(a) \leftarrow true,\ r(b) \leftarrow true\} \vdash_{\Pi} p(a,b)$$

Notice that arguments can also be represented as rooted trees: each premise in the argument defines a sub-tree with root corresponding to the conclusion of the axiom, and children corresponding to the sub-arguments allowing this conclusion to be derived. This alternative representation for A is given below:



The dependencies between claims supported by A are organised in the following structure.

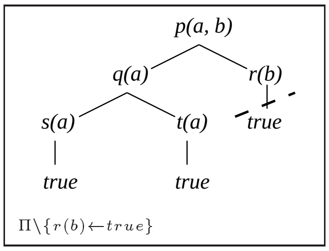


From this dependency structure we can say for instance that $p(a,b)$ is supported because both $q(a)$ and $r(b)$ are **in**. *The term true is always* **in**.

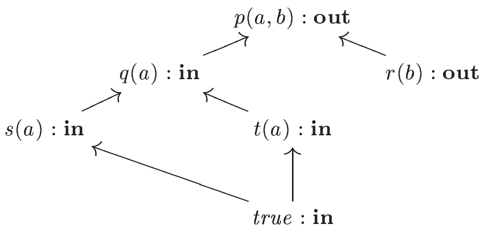
For the case of refutations, this sort of claim structure can be obtained by considering the dependencies in the refuted justification, then removing the rejected premise and finally propagating the labels appropriately. Consider for instance argument A above. There are many ways to refute A, one is by rejecting the premise $r(b) \leftarrow true$ thus retracting it from the theory so that A cannot be accepted as a justification for $p(a,b)$ with respect to $\Pi \setminus \{r(b) \leftarrow true\}$. This refutation is represented below.

$\{p(X,Y) \leftarrow q(X) \wedge r(Y),\ q(X) \leftarrow s(X) \wedge t(X),\ s(a) \leftarrow true,\ t(a) \leftarrow true,\ r(b) \leftarrow true\} \not\vdash_{\Pi \setminus \{r(b) \leftarrow true\}} p(a,b)$

A refutation can be depicted as a tree as follows, where we indicate the rejected premise by pruning the sub-tree defined by it.



Among the things that can be said about this, $p(a,b)$ is now argued to be unsubstantiated in $\Pi \setminus \{r(b) \leftarrow true\}$ because $r(b)$ is **out** *in $\Pi \setminus \{r(b) \leftarrow true\}$. The dependencies between claims are now represented as follows.*



□

The structure of claims supported by an argument is essentially a directed graph in which a node is labelled **in** only if all its supporting nodes are labelled **in**. This is again very similar to the sorts of dependency networks kept by truth maintenance systems, the only difference is that the dependency graph is obtained from a valid justification that has been (at some point) successfully generated via the provability relation $\vdash$, and hence well-founded either on valid assumptions or on the premise *true*. In **TMS** the graph is obtained from adding and deleting rules (so-called *justifications*) that are not necessarily related nor chained.

Remember, though, that in the case of refutations premises may have been either retracted or updated, and each of these possibilities must be carefully considered. But before defining the structure of claims formally, a note on notation: the symbols used for representing argument trees and directed graphs—e.g. the hooked arrow $\hookrightarrow$ to denote supporting edges in a graph—are presented in detail in Appendix B.

Definition 6.2 (Argument Claims) *Let A be an argument in Π. The claim structure supported by A is the directed graph $\mathcal{G}_A$ (with associated labelling function) inductively defined from the argument tree A as follows:*

Base case:

$\boxed{A = true}$

- $\mathcal{V}(\mathcal{G}_A) = \{true\}$ *and* $\mathcal{E}(\mathcal{G}_A) = \{\}$
- $label_{\mathcal{G}_A}(true) = \mathbf{in}$

$\boxed{A = tree(\varphi, assumption, \{\})}$

- $\mathcal{V}(\mathcal{G}_A) = \{\varphi\}$ *and* $\mathcal{E}(\mathcal{G}_A) = \{\}$
- $label_{\mathcal{G}_A}(\varphi) = \mathbf{in}$

Inductive case:

$\boxed{A = tree(\varphi_P,\ P,\ \{A_{\varphi_{P_1}}, A_{\varphi_{P_2}}, ..., A_{\varphi_{P_N}}\})}$

Let $\mathcal{G}_{A_1}, ..., \mathcal{G}_{A_N}$ be the claim structures supported by sub-arguments $A_{\varphi_{P_1}}, ..., A_{\varphi_{P_N}}$, respectively, such that $root(A_{\varphi_{P_i}}) = \varphi_{P_i}$.

Before we define how to combine such structures in order to obtain $\mathcal{G}_A$, consider the following auxiliary sets and labelling function (which merge the labelling functions obtained in the inductive step):

- $\mathcal{V}' = \bigcup_{i=1}^{N} \mathcal{V}(\mathcal{G}_{A_i})$ *and* $\mathcal{E}' = \bigcup_{i=1}^{N} \mathcal{E}(\mathcal{G}_{A_i})$
- $label' : \mathcal{V}' \mapsto \{\mathbf{in}, \mathbf{out}\}$, *where* $label'(\varphi) = \begin{cases} \mathbf{in} & \exists \mathcal{G}_{A_i}.label_{\mathcal{G}_{A_i}}(\varphi) = \mathbf{in} \\ \mathbf{out} & otherwise \end{cases}$

Moreover, let $\bigwedge$ be an operator for combining and propagating labels across supporting nodes in $\mathcal{G}_A$.

To define $\mathcal{G}_A$ we need to consider the possibilities for P, namely:

1. *P is an axiom in the theory;*
2. *P has been removed from the theory;*
3. *P has been replaced by some axiom P' in the theory.*

1. $P \in \Pi$

    - $\mathcal{V}(\mathcal{G}_A) = \mathcal{V}' \cup \{\varphi_P\} \quad and \quad \mathcal{E}(\mathcal{G}_A) = \mathcal{E}' \cup \bigcup_{i=1}^{N} \{\varphi_{P_i} \hookrightarrow \varphi_P\}$
    - $label_{\mathcal{G}_A}(\varphi) = \begin{cases} label'(\varphi) & \varphi \neq \varphi_P \\ \bigwedge\varphi & \varphi = \varphi_P \end{cases}$

2. $P \notin \Pi$ *has been removed in* Π

    Then the arguments supporting φ_P *are no longer relevant:*

    - $\mathcal{V}(\mathcal{G}_A) = \{\varphi\} \quad and \quad \mathcal{E}(\mathcal{G}_A) = \{\}$
    - $label_{\mathcal{G}_A}(\varphi) = \mathbf{out}$

3. $P \notin \Pi$ *has been replaced by* $P' \in \Pi$

    Let $\varphi_{P'}$ *and* $\varphi_{P'_1}, ..., \varphi_{P'_M}$ *denote the conclusion and preconditions of* P'*.*

    If $\varphi_{P'} \neq \varphi_P$*, this reduces to case 2 (as* P' *no longer derives* φ_P*).*

    Otherwise, if $\varphi_{P'} = \varphi_P$*, then:*

    - $\mathcal{V}(\mathcal{G}_A) = \mathcal{V}' \cup \{\varphi_P\} \cup \{\varphi_{P'_1}, ..., \varphi_{P'_M}\} \quad and \quad \mathcal{E}(\mathcal{G}_A) = \mathcal{E}' \cup \bigcup_{i=1}^{M} \{\varphi_{P'_i} \hookrightarrow \varphi_P\}$
    - $label_{\mathcal{G}_A}(\varphi) = \begin{cases} label'(\varphi) & \varphi \neq \varphi_P \text{ and } \varphi \in \mathcal{V}' \\ \mathbf{out} & \varphi \neq \varphi_P \text{ and } \varphi \notin \mathcal{V}' \\ \bigwedge\varphi & \varphi = \varphi_P \end{cases}$

$\square$

Instances of cases *1* and *2* are given in Example 6.1, whereas case *3* is illustrated later in Example 6.4. The observation below follows from this definition:

Observation 6.1 *If A is a justification for φ then $label_{\mathcal{G}_A}(\varphi) = \mathbf{in}$; otherwise, if A is a refutation of φ then $label_{\mathcal{G}_A}(\varphi) = \mathbf{out}$.* $\square$

6.2 The General Format of Attacks

The problem of how to generate an attack to a given argument can now be reduced to that of generating an argument that supports a contradictory claim. The basic intuition is simple: if a sentence is argued to be **in**, then in the next step of the argument we want to claim that it is **out**—and vice versa. In one direction, we can refute the argument that justifies this sentence; in the other, we can produce an alternative justification for it.

$$\varphi : \mathbf{in} \quad \overset{remove\_argument}{\rightsquigarrow} \quad \varphi : \mathbf{out}$$

$$\varphi : \mathbf{out} \quad \overset{add\_argument}{\rightsquigarrow} \quad \varphi : \mathbf{in}$$

Such types of attack are independent from the choice of logical system because they rely on supporting and blocking conclusions only. Nonetheless, it should be possible to account for any notion of conflict defined in the underlying language (e.g. through negation), meaning for instance that we could attack a justification not only by invalidating its premises but also by justifying an opposing view.

A question arises at this point, of how these relate to the attacks above. In other words, if $\overline{\varphi}$ denotes a sentence that conflicts[3] with φ then we want to determine whether the following types of attack are also legitimate:

1. If φ is argued to be **in**, then in the next step of the argument can we argue that φ is **out** by arguing that $\overline{\varphi}$ is **in**?

2. If φ is argued to be **out**, then in the next step of the argument can we claim that φ is **in** by arguing that $\overline{\varphi}$ is **out**?

The problem, though, is that the equivalence between $\varphi : \mathbf{in}$ and $\overline{\varphi} : \mathbf{out}$ does not generally hold. As discussed in Section 6.1, argumentation is concerned not with the truth of propositions but rather with *justifying* whether a proposition can be accepted as true on the basis of the reasons that can be constructed for it. From this perspective, a sentence can only be refuted if it has been previously justified. Arguing that a conflicting sentence is **out** does not mean that the sentence is not a consequence of the theory, and may not give enough reasons for accepting the sentence itself as substantiate (unless this is explicitly stated, e.g. by a burden shift premise).

[3] The only property assumed for the notion of conflict is that it is symmetric, so if $\overline{\varphi}$ conflicts with φ then φ conflicts with $\overline{\varphi}$.

Hence it is not necessarily the case that $\overline{\varphi}:\mathbf{out}$ implies $\varphi:\mathbf{in}$. On the other hand, however, it seems reasonable to contradict the claim that a sentence is **in** by justifying a conflicting sentence, as this gives enough reasons for not accepting the original sentence as substantiated; $\varphi:\mathbf{out}$ then follows as a consequence of $\overline{\varphi}:\mathbf{in}$. In this way, only the first type of attack above is also considered to be legitimate:

$$\varphi:\mathbf{in} \quad \overset{add\_argument}{\rightsquigarrow} \quad \overline{\varphi}:\mathbf{in}$$

Notice that in truth maintenance systems the situation is similar, as states *in* and *out*:

> [...] are not symmetric, for while reasons can be constructed to make P *in*, no reason can make P *out*. (At most, it can make $\neg P$ *in* as well.) (Doyle 1979, p. 234)

The following example illustrates the intuition behind this.

Example 6.2 *Let the following be sentences in a language for expressing the possible colours of an object x:*

$$\{colour(x, red), colour(x, yellow), colour(x, green)\}$$

such that conflict in this language is defined by $\overline{colour(X, C)} = colour(X, C')$*, where* $C' \neq C$*.*

Assume that $colour(x, red)$ *is currently* **in**. *According to the discussion above, possible attacks consist of either refuting* $colour(x, red)$ *or justifying* $\overline{colour(x, red)}$*, where:*

$$\overline{colour(x, red)} = colour(x, green) \quad or \quad \overline{colour(x, red)} = colour(x, yellow).$$

If the advanced attack has the form:

$$colour(x, red):\mathbf{in} \rightsquigarrow colour(x, green):\mathbf{in}$$

then $colour(x, red)$ *becomes* **out** *as* $colour(x, green)$ *is now* **in**. *At this point arguing that some conflicting sentence—e.g.* $colour(x, yellow)$*—is* **out** *may not change the current* **out** *status of* $colour(x, red)$*:*

$$colour(x, red):\mathbf{out} \not\rightsquigarrow colour(x, yellow):\mathbf{out}.$$

This sort of attack does not have the quality of refuting the sentence $colour(x, yellow)$ as a justification for it has not yet been advanced. On the other hand $colour(x, green)$ has been justified so the following attack is legitimate:

$$colour(x, green) : \mathbf{in} \rightsquigarrow colour(x, green) : \mathbf{out},$$

and it would consequently reinstantiate the **in** *status of $colour(x, red)$.* □

This sort of attack can be useful to introduce new sentences other than supporting sentences that are also relevant to the argumentation process. In this way, a sentence is now said to be **in** not only if all its supporting sentences are argued to be **in**, but also if no (known) conflicting sentence is **in** as well.

We now formalise this intuition, classifying the general purpose of revision operations for generating attacks in a dynamic argument in terms of of the general format of attacks discussed above. In Section 7.3.4 these are used as the starting point for defining a collection of more detailed revisions.

Definition 6.3 (General Types of Revision) *Let A be an argument in Π, and A' be an argument in a revised theory Π_ϕ such that it attacks A. To describe the types of attack-based revision ϕ yielding the derivation of A' (see Definition 4.6), we shall consider the possibilities for contradiction.*

On one hand, if A supports $\varphi : \mathbf{in}$ then A' has to support $\varphi : \mathbf{out}$, either because it directly rejects φ or because it supports $\overline{\varphi} : \mathbf{in}$.

(a) $\varphi : \mathbf{in} \overset{remove\_argument^{\Pi,A}}{\rightsquigarrow} \varphi : \mathbf{out}$

Here A' is necessarily a refutation of A, in which we reject the premise used for inferring φ. The purpose of ϕ is to refute φ by blocking the derivation of A, withdrawing this argument as being a valid, well-grounded justification for φ.

(b) $\varphi : \mathbf{in} \overset{add\_argument^{\Pi,A}}{\rightsquigarrow} \overline{\varphi} : \mathbf{in}$

Here A' is necessarily a counter-argument for φ, i.e. a justification for $\overline{\varphi}$ where $\overline{\varphi}$ and φ are conflicting sentences in the language. The purpose of ϕ is to allow A' to be derived, where ϕ may be trivial if A' can be inferred from Π.

On the other hand, for A' to contradict $\varphi : \mathbf{out}$, it must support $\varphi : \mathbf{in}$.

(c) $\varphi : \mathbf{out} \overset{add\_argument^{\Pi,A}}{\rightsquigarrow} \varphi : \mathbf{in}$

Here A' must be a justification for φ. As in case (b) above, the purpose of ϕ is to justify φ by allowing A' to be derived, and ϕ may be trivial if A' can already be inferred from Π. □

A couple more notes on terminology. An attack that contradicts the main claim of an argument is known as a *direct attack*, whereas an attack that contradicts an indirect claim of an argument is said to be an *indirect attack*. Moreover, the possible types of contradiction in Definition 6.3 are in accordance with the three general types of conflict (or attack) identified in the literature, namely *rebuttals*, *undercutting attacks* and *assumption attacks* (Prakken and Vreeswijk 1999):

1. *Undercutting attacks* reject not a sentence itself but the premise supporting its inference.

   > Undercutting attacks correspond to case *(a)* above: if a sentence is proved to be **in**, argue that it is **out** by refuting (undercutting) the justification given for it.

2. *Rebuttals* are symmetric types of attack in which arguments have conflicting conclusions.

   > Rebuttals are captured by case *(b)* above: if a sentence is proved to be **in**, rebut it by proving that a conflicting sentence is also **in**.

3. *Assumption attacks* prove the contrary of what was assumed without being proved.

   > Assumption attacks can be captured by case *(c)*, in the particular case of non-provability assumptions: if a sentence is assumed to be **out**, prove that it is in fact **in** (prove what was argued to be not provable). More generally, if assumptions are considered to be special sentences that can extend the initial language, then assumption attacks can be captured by case *(b)*: if an assumption is argued to be **in**, prove that its contrary is **in** (where the notion of the contrary of an assumption is similar to that of conflict, but asymmetric).

Notice that some care may be needed in handling conflicting sentences appropriately. The following example illustrates what problems might arise.

Example 6.3 *In Example 6.2, the claim structure supported after the attack:*

$$colour(x, red) : \mathbf{in} \rightsquigarrow colour(x, green) : \mathbf{in}$$

is depicted by the following directed graph, where the dotted edge represents a conflicting link rather than a supporting link (i.e. $colour(x, green)$ *is in conflict with* $colour(x, red)$*):*[4]

$$\begin{array}{c} colour(x, red) : \mathbf{out} \\ \uparrow \\ colour(x, green) : \mathbf{in} \end{array}$$

Both claims represent potential points of attack that can allow $colour(x, red)$ *to be reinstated. According to Definition 6.3, the possibilities for attack in the next step are:*

$$\begin{array}{rcl} colour(x, red) : \mathbf{out} & \rightsquigarrow & colour(x, red) : \mathbf{in} \\ colour(x, green) : \mathbf{in} & \rightsquigarrow & colour(x, green) : \mathbf{out} \\ colour(x, green) : \mathbf{in} & \rightsquigarrow & \overline{colour(x, green)} : \mathbf{in} \end{array}$$

Nevertheless, because $colour(x, green)$ *is itself a conflicting sentence (rather than a supporting node), not all sentences* $\overline{colour(x, green)}$ *in the third type of attack are guaranteed to change the status of the sentence* $colour(x, red)$ *above in a coherent way. Consider for instance the following attack, where* $\overline{colour(x, green)} = colour(x, yellow)$*:*

$$colour(x, green) : \mathbf{in} \rightsquigarrow colour(x, yellow) : \mathbf{in}$$

The structure of dependencies is now represented as:

$$\begin{array}{c} colour(x, red) : \mathbf{in} \\ \uparrow \\ colour(x, green) : \mathbf{out} \\ \uparrow \\ colour(x, yellow) : \mathbf{in} \end{array}$$

[4] Refer to Appendix B for detailed notation.

which essentially says that "x is red because it is not green, and it is not green because it is yellow", and that is clearly inconsistent. The reason why this type of attack is problematic is because sentences that conflict with $colour(x, green)$ *may also conflict with* $colour(x, red)$. *The solution is to restrict the types of rebuttals that can be generated for conflicting nodes to those that created the conflict itself; in this case the only choice for* $\overline{colour(x, green)}$ *that can effectively alter the status of* $colour(x, red)$ *is* $colour(x, red)$ *itself:*

$$colour(x, green) : \mathbf{in} \leadsto colour(x, red) : \mathbf{in}.$$

$\square$

The next example also illustrates some of the concepts presented so far:

Example 6.4 *Consider again argument* A *for* $p(a, b)$ *in Example 6.1. One way to refute* A *is for instance by arguing that* $q(a)$ *should not be substantiated:*

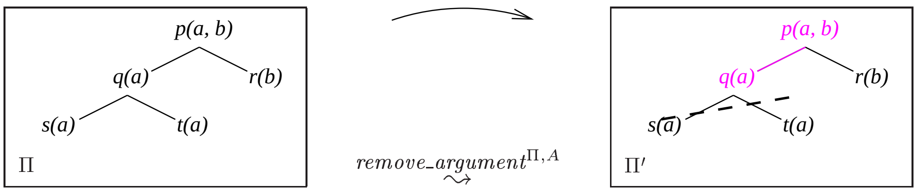
$$q : \mathbf{in} \overset{remove\_argument^{\Pi,A}}{\leadsto} q : \mathbf{out}$$

This attack corresponds to case (a) in Definition 6.3 instantiated to the sentence $q(a)$. *Note that this is an indirect attack to* A *because it contradicts an indirect claim.*

To present such an attack Π *needs to be revised into* Π' *so as to reject the premise used for deriving* $q(a)$. *The following revision operation does that by elaborating on the preconditions for applying the rule:*

$$\boxed{\begin{array}{l} p(X,Y) \leftarrow q(X) \wedge r(Y) \\ q(X) \leftarrow s(X) \wedge t(X) \\ r(b) \leftarrow true \\ s(a) \leftarrow true \\ t(a) \leftarrow true \end{array}} \quad \overset{remove\_argument^{\Pi,A}}{\leadsto} \quad \boxed{\begin{array}{l} p(X,Y) \leftarrow q(X) \wedge r(Y) \\ q(X) \leftarrow s(X) \wedge t(X) \wedge r(X) \\ r(b) \leftarrow true \\ s(a) \leftarrow true \\ t(a) \leftarrow true \end{array}}$$

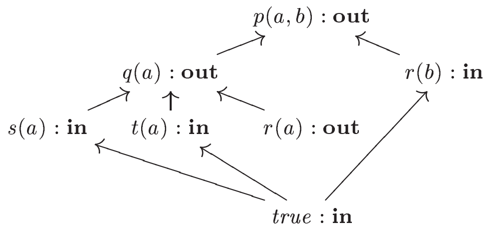
The sentence $q(a)$ *is refuted because* A *is no longer a sound argument with respect to* Π'. *This fragment of a dynamic argument is pictured below along the same lines as Figure 2.1.*



Notice that this diagram only represents the original argument being refuted because this is what the argument move is about. However, we want also to be able to capture the consequences of this revision, such as the addition of a new precondition, and this is given by the corresponding claim structure.

As discussed in Definition 6.2, the structure of claims supported by a refutation is dependent on the sub-argument defined by the rejected axiom; in this case, $q(X) \leftarrow s(X) \wedge\ t(X)$*. The reason why sentence* $q(a)$ *became unsupported after the update is because it now depends on a new precondition, namely* $r(a)$*, that is claimed to be* **out** *because it has not (yet) been shown to be supported.*

Every sentence that was dependent on $q(a)$ *also becomes unsupported after the refutation, though other claims such as* $s(a)$: **in** *still hold and are still relevant to the argument. The structure below represents these dependencies between sentences after the attack has been advanced, according to case 3 in Definition 6.2.*



This structure also gives claims that can be attacked in the next step. For instance, one could alter the status of $q(a)$*—and hence of* $p(a,b)$*—by justifying* $r(a)$*.* □

So the notion of *contradictory claims* provides a higher-level description of argument dynamics than that based on structural revision. Claims convey the intention of rejecting existing arguments and also of advancing new ones, without specifying premises to be retracted from or added to the theory. This is in line with the discussion in Section 4.1, where dynamic argumentation was described as an abstract process of manipulation of arguments as primitive entities. In this sense, we have now taken a first step towards capturing that abstract view in terms of a more pragmatic approach based on the revision of sets of premises.

Determining which claims are supported after an attack has been advanced is particularly important in the context of a dynamic argument, where we need to keep track of issues such as:

- which claims can be contradicted during the course of argument;
- which claims are relevant at each point in the process; and
- how these relate to the main sentence under dispute.

Next we describe a way to propagate the effects of an attack to the claims supported by the original argument, which is used in this thesis for generating and automating argument dynamics.

6.3 Possible Attacks in a Dynamic Argument

In a dynamic argument about a sentence φ, the purpose of each advanced argument is to alter the status of φ from *substantiated* (**in**, or *acceptable*) to *unsubstantiated* (**out**, or *unacceptable*), and vice-versa. As justifications and refutations are presented, dependencies between φ and other sentences are made explicit, and we should be able to look at these in order to select a claim to be contradicted so that it will change the current acceptability status of φ.

The moves $\langle A_0, \phi_1, A_1, ..., \phi_i, A_i \rangle$ advanced up to step $i \geq 0$ define a dependency structure of annotated sentences that represents not a precise record of the argumentation but rather the claims that are supported and relevant after argument A_i has been advanced. Essentially, this structure is a directed graph obtained from the corresponding claim structures $\mathcal{G}_{A_1}, ..., \mathcal{G}_{A_i}$ (see Definition 6.2) by combining them appropriately. In the same way, a node is labelled **in** only if all supporting nodes are labelled **in** (and no conflicting nodes are labelled **in**), and the claims to be contradicted are those that can effectively alter the status of the node containing φ.

Based on the definitions of a dynamic argument (4.7), argument claims (6.2) and general types of attack (6.3), we can now describe how to incrementally construct this dependency graph as the course of argument develops.

Definition 6.4 (Dependency Graph) *Let $\langle A_0, \phi_1, A_1, ..., \phi_i, A_i\rangle$ be the state of a dynamic argument $\delta(\varphi, \Pi)$ at step $i \geq 0$. The dependency graph of claims supported at this point is a directed graph $\mathcal{D}_i$ with labelling function, which can be defined as follows:*

Base case $(i = 0)$

By definition A_0 is a justification for φ, therefore $\mathcal{D}_0$ is equivalent to $\mathcal{G}_{A_0}$.

Inductive case $(i > 0)$

By definition A_i attacks A_{i-1}, thus A_i contradicts a claim supported by A_{i-1} in the context of $\langle A_0, \phi_1, A_1, ..., \phi_{i-1}, A_{i-1}\rangle$—i.e. a claim in $\mathcal{D}_{i-1}$. Let A_i be an attack to a sentence φ, and $\mathcal{G}_{A_i}$ the claim structure supported by it.

Moreover, let $\bigwedge$ be an operator for combining and propagating labels across supporting and conflicting nodes in $\mathcal{D}_i$. $\mathcal{D}_i$ is constructed from $\mathcal{D}_{i-1}$ and $\mathcal{G}_{A_i}$ as follows:

- $\mathcal{V}(\mathcal{D}_i) = \mathcal{V}(\mathcal{D}_{i-1}) \ \cup \ \mathcal{V}(\mathcal{G}_{A_i})$
- $\mathcal{E}(\mathcal{D}_i) = \Big(\mathcal{E}(\mathcal{D}_{i-1}) \setminus \mathcal{E}_{\mathcal{V}(\mathcal{G}_{A_i})}(\mathcal{D}_{i-1})\Big) \ \cup \ \mathcal{E}'$, *where*
  - $\mathcal{E}_{\mathcal{V}}(\mathcal{D}) \subseteq \mathcal{E}(\mathcal{D})$ *denotes the set of edges in a graph $\mathcal{D}$ that terminate at some node in the set $\mathcal{V}$; and*
  - $\mathcal{E}'$ *denotes the links between sentences in the argument structure, maybe with an additional conflicting edge in the case of rebuttals:*[5]

$$\mathcal{E}' = \begin{cases} \mathcal{E}(\mathcal{G}_{A_i}) \cup \{\overline{\varphi} \hookrightarrow \varphi\} & A_i \textit{ was given by } \varphi : \mathbf{in} \rightsquigarrow \overline{\varphi} : \mathbf{in} \\ \mathcal{E}(\mathcal{G}_{A_i}) & \textit{otherwise} \end{cases}$$

The labelling function $label_{\mathcal{D}_i}$ is defined below, where $\mathcal{V}' \subseteq \mathcal{V}(\mathcal{D}_i)$ denotes the set of nodes reachable in $\mathcal{D}_i$ from some node in $\mathcal{V}(\mathcal{G}_{A_i})$:

$$- \ label_{\mathcal{D}_i}(\psi) = \begin{cases} label_{\mathcal{G}_{A_i}}(\psi) & \psi \in \mathcal{V}(\mathcal{G}_{A_i}) \\ label_{\mathcal{D}_{i-1}}(\psi) & \psi \notin \mathcal{V}(\mathcal{G}_{A_i}) \textit{ and } \psi \notin \mathcal{V}' \\ \bigwedge\psi & \psi \notin \mathcal{V}(\mathcal{G}_{A_i}) \textit{ and } \psi \in \mathcal{V}' \end{cases}$$

□

[5] Remember from Appendix B that dotted arrows $\hookrightarrow$ represent conflicting edges in a graph whereas supporting links are depicted by $\hookrightarrow$.

The basic idea behind this definition is to remove any edges from the original dependency graph which are used to support sentences from the new argument, actually replacing these by the relations given in this argument.

Note that by construction the following observation holds:

Observation 6.2 *The only way to introduce a conflicting node in the dependency graph is by explicitly justifying a conflicting sentence according to some notion of conflict—case (b) of Definition 6.3.* □

> *In fact, the argument structures themselves only include supporting links (see Definition 6.2). This type of attack allows for an explicit introduction of conflict, which is not captured by the underlying argument generated mechanism, but which can be used both for rebutting sentences in the language and for contradicting assumptions.*

This is again another point of connection with truth maintenance systems, where sentences P and $\neg P$ are unrelated unless this is explicitly stated, and a node is expressly marked as a *contradiction*.

This definition tells us how the status of other sentences are affected by an attack, introducing sentences that become relevant in the light of the new argument and dismissing others that are no longer at issue. This dependency structure gives the possible attacks for the next step of argument, namely any claim such that altering its label will affect the status of the main sentence.

- if a sentence in **in**, then all its supporting sentences (which are **in**) and its conflicting sentences (which are **out**) are potential points of attacks;
- if a sentence is **out**, then potential points of attacks include conflicting sentences which are **in** or supporting sentences which are **out**.

The possible attacks to sentence φ at step i in a dynamic argument are then given by the transitive closure in $\mathcal{D}_i$ of these potential points of attack with respect to the current status of φ. This idea is equivalent to that of *supporting-nodes* defined by Doyle (1979), who refers to the corresponding transitive closure as the *ancestors* of a node.

6.4 Argumentation and Truth Maintenance Systems

Considering the many similarities pointed out in the previous sections, this is a good time to discuss the relation between these two approaches in more detail. Before deepening the discussion, let us briefly summarise the basic concepts behind truth maintenance systems (Doyle 1979; de Kleer 1986; Forbus and de Kleer 1993).

There are essentially two sorts of structures in a TMS: nodes representing propositions, and justifications associated with these nodes. Each justification consists of two lists of nodes—an *IN*-list and an *OUT*-list—such that a justification is said to be valid only if every node in the *IN*-list is *in* and every node in the *OUT*-list is *out*. Assumptions in particular are nodes whose supporting justification has an empty *IN*-list (so they cannot be justified) and a non-empty *OUT*-list (but they can be contradicted). There are also two types of mechanisms involved: a *truth maintenance procedure* for making revisions in the support status of nodes given that justifications may be added and retracted; and the *dependency-directed backtracking* for identifying which assumptions need to be changed in order to restore consistency in case of contradiction.

According to Doyle (1979, p. 236) the purpose of a TMS is that it:

> [...] records and maintains arguments for potential program beliefs, so as to distinguish, at all times, the current set of program beliefs.

And given that Doyle also proposes a way to "organize a problem solving program's use of the TMS into the form of dialectical argumentation"[6], the question of how exactly the two approaches relate becomes more and more persistent.

The difference turns out to be more a shift in emphasis than it is technical. Whereas truth maintenance systems are concerned with "how to make changes in computational models" (Doyle 1979, p. 231), models of argumentation as studied in AI—especially models of argument dynamics—are more concerned with the issue of *what changes to make*. From an argumentation perspective there is not much interest in maintaining or reestablishing consistency, but rather in exploring contradictions and introducing

[6] See Section 6 in (Doyle 1979).

conflicts and attacks deliberately.

In this way, while the *dependency-directed backtracking* mechanism is more about restoring consistency (and hence not as germane to the process of argumentation), the sort of truth-maintenance procedure on the other hand seems to have a significant role to play in argumentation models. Originally this is supposed to give the acceptability status of sentences in the current set of beliefs, but it could also be interpreted as a mechanism for keeping track of weaknesses and points of attack given the justifications considered so far.

We have found that by forcing notation and terminology to be similar the differences and relations between the argumentation and **TMS** became more apparent. For instance, we are now able to ask a more specific question, namely:

> If we keep adding and retracting justifications to a **TMS** according to the justifications and refutations advanced during an argumentation process, will the **TMS** network be equivalent to the dependency graph that is constructed incrementally during the course of the argument?

The answer to this question is *sometimes yes, but generally no.* The fundamental difference is that a **TMS** keeps a *set* of justifications associated with each node, each representing a different reason for it, whereas the dependency graph in Definition 6.4 only maintains the links associated to one argument, namely the argument that was last advanced (remember that every edge supporting a sentence is dismissed unless this sentence is not part of the new argument). So if an invalid justification becomes valid again there is no need to explicitly add this justification again, as the **TMS** automatically updates the status of the supported sentence to *in*. Argumentation mechanisms on the other hand must generate a new well-founded justification and the entire new argument needs to be explicitly advanced again.

It is true, though, that one could *bolt on* a **TMS** to our argument revision component to produce a more *sophisticated* system that can keep track of the consequences that follow from every argument advanced so far, even if these have not been explicitly stated before. However, it seems to us that the emphasis in argumentation is more on

```
add_just_rms(If, Then) :-
     add a new justification If → Then to the database
     and propagate the effects

del_just_rms(If, Then) :-
     remove a justification If → Then to the database
     and propagate the effects
```

Figure 6.1: Basic interfacing predicates as defined by Shoham (1994).

searching for and advancing appropriate arguments during the process. In any case, it is also possible to *force* a truth maintenance system to keep only one relevant justification associated to each node by deleting every previous justification when a new argument is advanced.

6.4.1 Experiments with Truth Maintenance

Effective testing of this relation between truth maintenance procedures and dependency graphs was also possible. The experiments consisted in feeding both mechanisms with the same justifications and comparing the results at each step of argument. On the argumentation side we have used our own Prolog implementation; on the TMS side we have used Shoham's implementation of a reason maintenance system[7] as described in (Shoham 1994). Figure 6.1 gives the basic interface predicates in this system.

This section illustrates one such experiment, namely the use of the TMS mechanism in the context of Examples 6.1 and 6.4. Note that in this case the outcome is identical to the one given by the dependency graph as no alternative reasons exist simultaneously for any of the sentences.

In Shoham's implementation, a justification is an expression of the form:

$$If \rightarrow Then$$

where *If* is a list of nodes that justify the sentence *Then*. Supporting nodes of the form $(N, +)$ are in the so-called *IN*-list, while nodes denoted by $(N, -)$ are said to be in the

[7] The Prolog code is available online at `http://yoda.cis.temple.edu:8080/books/shoham/`.

OUT-list. Moreover, the special node *premise* is always *in* (note that this is equivalent to the special term *true* in Definitions 6.2 and 6.4).

What we mean by justification in this thesis is more like a collection of justifications in the **TMS** sense. So in order to supply appropriate information to the **TMS** machinery, justifications then need to be broken into the smaller steps that correspond to the application of each axiom. For instance, the following five justifications can represent the justification A for $p(a, b)$ given in Example 6.1:

```
| ?- add_just_rms(((q(a),+), (r(b),+)), p(a,b)),
     add_just_rms(((s(a),+), (t(a),+)), q(a)),
     add_just_rms(((premise,+)), s(a)),
     add_just_rms(((premise,+)), t(a)),
     add_just_rms(((premise,+)), r(b)).

yes
```

After adding these justifications, the status of the sentences are represented by the network in Figure 6.2, where the label *in* indicates that the corresponding sentence is in the database. The predicate `printdb/0` gives the current state of the database, where *in* sentences are denoted by the predicate `rms/1`.

```
| ?- printdb.
Database listing :
The facts:
rms(premise).
rms(s(a)).
rms(t(a)).
rms(q(a)).
rms(r(b)).
rms(p(a,b)).

Justifiers:
justifier(q(a), +, j1).
justifier(r(b), +, j1).
justifier(s(a), +, j2).
justifier(t(a), +, j2).
justifier(premise, +, j3).
justifier(premise, +, j4).
justifier(premise, +, j5).

Justificands:
justificand(j1, p(a,b)).
justificand(j2, q(a)).
justificand(j3, s(a)).
justificand(j4, t(a)).
justificand(j5, r(b)).

yes
```

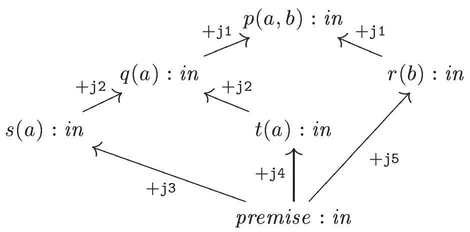


Figure 6.2: A **TMS** corresponding to argument A for $p(a, b)$.

Consider now the case of refutations, which are used for rejecting axioms in a justification, either by removing it from the theory or by updating it so that the argument no longer follows. The first case is also illustrated in Example 6.1, where a refutation for A was given on the basis of rejecting axiom $r(b) \leftarrow true$. In **TMS** style, such refutation could be obtained by deleting the corresponding justification as follows:

```
| ?- del_just_rms(((premise,+)), r(b)).

yes

| ?- printdb.
Database listing :
The facts:
rms(premise).
rms(s(a)).
rms(t(a)).
rms(q(a)).

Justifiers:
justifier(q(a), +, j1).
justifier(r(b), +, j1).
justifier(s(a), +, j2).
justifier(t(a), +, j2).
justifier(premise, +, j3).
justifier(premise, +, j4).

Justificands:  justificand(j1, p(a,b)).
justificand(j2, q(a)).
justificand(j3, s(a)).
justificand(j4, t(a)).

yes
```

Figure 6.3 gives the state of the database after justification $premise \rightarrow r(b)$ was deleted.

Notice though that in refutations axioms do not need to be rejected for good, but can be updated. Rather then refuting argument A by rejecting axiom $r(b) \leftarrow true$, Example 6.4

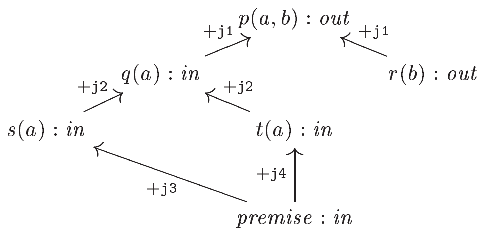


Figure 6.3: **TMS** from Figure 6.2 after *premise* $\rightarrow r(b)$ was deleted.

illustrates a type of refutation that elaborates on the preconditions for applying axiom $q(X) \leftarrow s(X) \wedge t(X)$. This can be captured in a **TMS** style as follows, considering the particular instance of this axiom supporting the sentence $p(a, b)$.

```
| ?- del_just_rms(((s(a),+), (t(a),+)), q(a)),
     add_just_rms(((s(a),+), (t(a),+), (r(a),+)), q(a)).

yes

| ?- printdb.
Database listing :
The facts:
rms(premise).
rms(s(a)).
rms(t(a)).
rms(r(b)).

Justifiers:
justifier(q(a), +, j1).
justifier(r(b), +, j1).
justifier(premise, +, j3).
justifier(premise, +, j4).
justifier(premise, +, j5).
justifier(s(a), +, j6).
justifier(t(a), +, j6).
justifier(r(a), +, j6).

Justificands:
justificand(j1, p(a,b)).
justificand(j3, s(a)).
justificand(j4, t(a)).
justificand(j5, r(b)).
justificand(j6, q(a)).

yes
```

Figure 6.4 gives the state of the database after justification $s(a), t(a) \rightarrow q(a)$ was elaborated into $s(a), t(a), r(a) \rightarrow q(a)$.

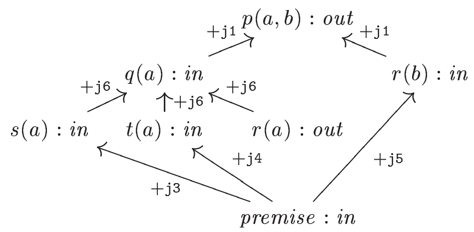


Figure 6.4: **TMS** from Figure 6.2 after $s(a), t(a) \rightarrow q(a)$ was updated.

The fact is that it is possible to use a **TMS** to keep a record of points of attack during argumentation. By taking the extra care of maintaining only one current justification for each node, and of grounding any variables in order to bind them appropriately, we can then get the desired results. In our current implementations of dynamic argumentation generators (see Chapter 8) it is possible to use Shoham's implementation as the dependency graph mechanism.

So this chapter described a high level account of attack-based revision relations in terms of potential contradictions, characterising the possible attacks during the course of a dynamic argument. Chapter 8 further illustrates these concepts in terms of the aflatoxin debate in Chapter 5. The next step is to propose a collection of operations that elaborate on the general types of revision *add_argument* and *remove_argument* in order to satisfy the sorts of attacks discussed in Section 6.2. Note, however, that up till now relations were described only at a fairly abstract level. But to define such a collection of more refined structural revisions we will need to commit to a particular underlying logical system.

Chapter 7

A Formal Classification of Argument Schemata

This chapter addresses another issue identified in Chapter 4, that of how to specify an appropriate set of revision operations for generating dynamic arguments, and the way we tackle this problem is by categorising argument revision schemata in terms of the types of attacks identified in the previous chapter. At this point we also commit to a specific underlying logical system.

7.1 Generating Dynamic Arguments

A dynamic argument as defined in Chapter 4 is a process of argument exchange which may involve structural changes to the underlying knowledge base. From the perspective of transformation of theories, the purpose of a dynamic argument is to produce a theory Π' from an initial theory Π which is more acceptable with respect to a sentence φ. At each step, the original theory may revised until no more attacks to φ—or counter attacks to defend φ—can be generated. Whether this process converges and all attacks are properly rejected depends on the types of predefined revision operations that are allowed.

Dynamic arguments can then be generated by a term rewriting system, expressed in a logic programming style in Figure 7.1.

The term κ represents the attack generation step, expressed here as a relation between

$$
\boxed{\begin{array}{lcl}
\delta(\varphi, \Pi, \Pi') & \leftarrow & \kappa(\Pi, \Pi'') \ \wedge \ \delta(\varphi, \Pi'', \Pi') \\
\delta(\varphi, \Pi, \Pi) & \leftarrow &
\end{array}}
$$

Figure 7.1: A system for generating dynamic arguments.

theories. In fact, $\kappa(\Pi, \Pi'')$ holds if and only if from Π'' we can derive an attack on an argument in Π. If ϕ is an attack-based revision that can be applied to Π, then $\kappa(\Pi, \Pi_\phi)$ holds by definition (see Definition 4.6).

In the rest of this chapter we propose a way to refine the relation κ for obtaining an organised collection of argument revision schemata based on the general characterisation of attacks given in Chapter 6. This collection provides a systematic way to define the set of revision operations that can be applied in a dynamic argument, also helping to identify useful properties that these operations could have.

7.2 A Logic Programming Framework

In Chapter 4 we proposed a generic formalisation of dynamic argumentation that was based on an arbitrary logical system, leaving a number of parameters undetermined. Nevertheless, one of our aims is to define a dynamic argumentation framework that is of practical use, and which can be applied in a systematic way. And whilst the concepts defined in Section 4.2 are abstract enough to capture the type of behaviour in which we are interested, they still leave too much to be specified for someone wishing to use them.

So to fully describe the concepts in Chapter 4 we shall adopt a specific underlying system, namely logic programming (based on the resolution method). The reasons behind this choice are manifold. First, logic programming theory has its roots in first-order predicate calculus. Because many people are familiar with first-order languages, there is no need to introduce and explain new symbols, connectives or semantics. Moreover, logic programming has proved suitable for a number of tasks in the knowledge representation realm. In our case, for instance, it is natural to think of logic programs as a way of expressing theories that represent models, contracts or beliefs. Finally, as an executable language, logic programming is also computationally attractive.

We often assume that theories correspond to *general logic programs* in a first-order language, as illustrated in Chapter 5. Though many of our results are based on *definite logic programs*, which are general programs restricted to Horn clauses (without the occurrence of negation as failure, and thus with the advantage of monotonicity), we also discuss whether and how these results extend to the more generic case. A brief account of logic programming is given in Appendix A.

Given an underlying logic programming framework, we can now elaborate on the formal definitions in Section 4.2. In particular, the notion of argument follows directly. Just as in Definition 4.3, an argument contains the clauses used in the derivation of a sentence and can be depicted by the corresponding support tree.

Attacks are reduced to contradictory claims (as presented in Definition 6.3), although what it means for two sentences to be in conflict still remains to be specified. In logical languages, conflict is often represented in terms of explicit negation and thus reduced to inconsistency. Rather than allowing an explicit account of (classical) negation within the logic, we treat conflict as a meta-level relation between predicates in the language. This approach is in line with a number of proposals in the literature (Bondarenko et al. 1997; Ambler 1996).

What is more, in the case of logic programming the types of argument claims seem to be naturally associated with the notion of *interpretation*. The interpretation $\iota(\Pi)$ of a (definite) logic program Π contains all ground atoms that can be deduced from Π; that is, all the ground sentences that are justified in this theory. Hence, stating that an argument in Π supports the claim φ : **in** corresponds to saying that at least one ground instance of φ is in $\iota(\Pi)$. Again, this correspondence does not hold as neatly for the case of refutations because being **out** does not necessarily mean not being **in**.[1]

But this is not necessarily bad news. In fact, in the case of monotonic systems we can associate the interpretation sets of an original theory and a revised theory by means of set inequality relations. Moreover, the notion of argument (and of argument claim) is important here because it helps focusing on certain elements of these sets, rather than calculating and enumerating them all. We discuss these properties in Chapter 9.

[1] See discussion in Section 6.1.

7.2.1 Considering Negation as Failure

At this point we should make some remarks about how arguments involving negation relate to the corresponding interpretation sets. The interpretation set of a *general logic program* under the closed world assumption consists of all the ground atoms that can be derived from this theory *plus* the negation of the ground atoms that cannot be inferred from it.

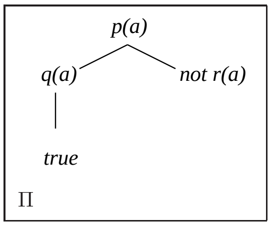
It is worth noting that argumentation has been used for capturing various alternative semantics for general logic programs, such as well-founded or stable semantics (Bondarenko et al. 1997). Our approach to negation here is slightly different. We are less interested in discussing what is the right semantics for negation in logic programs than in handling it as finite failure by using an extension of the original resolution mechanism. These two approaches to negation are distinct and have been characterised by Dix and Brewka (1997) as the *NML-approach* (focus on non-monotonic issues) and the *LP-approach* (focus on logic programs themselves), respectively. Of course there are non-monotonic aspects of our proposal, and these are discussed in Chapter 9.

In any case, if negation as failure is involved then the argument premises should contain not only clauses from the program, but also the negated ground consequences needed in the derivation. Such sentences are considered to be *assumptions* because they cannot be formally proved to be **in**, but only assumed to be **in** because some contradictory sentence is **out**.

Example 7.1 *Let* Π *be the following general logic program:*

$$\begin{array}{rcl} p(X) & \leftarrow & q(X) \wedge \ not\ r(X) \\ q(a) & \leftarrow & true \\ r(b) & \leftarrow & true \end{array}$$

Then argument A below is a derivation (based on the resolution mechanism extended with negation as finite failure) supporting $p(a)$*:*



and which can be represented by:

$$\{p(X) \leftarrow q(X) \wedge not\ r(X),\ q(a) \leftarrow true\} \cup \{not\ r(a)\} \vdash_{\Pi} p(a).$$

The structure below depicts the dependencies between claims in argument A.

$$\begin{tikzcd}
 & p(a) : \mathbf{in} & \\
q(a) : \mathbf{in} \arrow[ur] & & not\ r(a) : \mathbf{in} \arrow[ul] \\
true : \mathbf{in} \arrow[u] & & r(a) : \mathbf{out} \arrow[u, dotted]
\end{tikzcd}$$

□

7.3 A System of Argument Rewrites

Having introduced the notion of a general attack-based relation between theories, this section describes an organised collection of argument revision schemata for satisfying this relation. This is done by specifying a rewriting system for refining the relation κ in Figure 7.1 into argument schemata for theory revision based on the general characterisation of attacks.

As discussed in Chapter 5, we want to provide descriptions of argument schemata to which domain-specific knowledge can be applied. To enrich and give coherence to our proposal, we organise these schemata in a cascade of levels ranging from an overall classification in terms of interpretation to the manipulation of particular clauses in a theory, eventually getting to a domain-specific level. The suggested organisation provides a pragmatic way to define revision schemata for attack generation, but it turns out to be useful also in supporting explanation and retrospective analysis of a dynamic argument at different levels of abstraction.

As we go down this classification tree, we instantiate the necessary parameters for generating a valid attack. To guarantee that this is the case, to each rewrite we associate a set of relevant properties that can be verified during or after the instantiation. Properties associated with each rewrite persist through subsequent rewrites, thus accumulating a set of properties during the process.

Rewrite rules also have conditions which are used to instantiate and constrain their parameters. There are two types of conditions here: some are concerned with choosing an element from a set ($\in$-conditions); others, with instantiating the rewrites with these selected elements ($=$-conditions). Satisfying these generates an instance of an attack.

In what follows, rewrites are grouped into sections according to the different classification levels. A standard presentation pattern is adopted for each rewrite rule, consisting of an informal description together with the formal rewrite rule, and the related properties and conditions.

7.3.1 The General Attack Relation between Theories

This section gives the rule for refining the general relation between theories, thus allowing it to be rewritten as an attack generation step involving some unrestricted attack-based revision to the underlying theory. The idea is to constrain this operation as the attack relation is refined, and the property *attacks* can be used to ensure that the original argument is in fact attacked by (and not preferred over) the argument that is generated.

Argument Rewrite 1 *For a general attack relation between two theories* Π *and* Π' *to hold, we can identify an argument* A *in* Π *such that* ϕ *is an (unrestricted) attack-based revision operation to* Π *with respect to* A*, and hence in* Π' *we can derive an argument* A' *that attacks* A.[2]

$$\begin{array}{lll} \kappa(\Pi, \Pi') & \Rightarrow & \mathit{argument}(A, \Pi), \\ & & \Pi \overset{\phi^{\Pi,A}}{\rightsquigarrow} \Pi', \\ & & \mathit{argument}(A', \Pi') \end{array}$$

$$\begin{array}{ll} \textsf{Properties:} & \{\ \mathit{attacks}(A', A)\ \} \\ \textsf{Conditions:} & \mathit{true} \end{array}$$

7.3.2 The General Form of Theory Revision

Any revision operation is characterised by two sets containing the axioms to be removed from and added to the theory, respectively. Defining a meaningful revision operation

[2] See Definition 4.6.

is then reduced to selecting these sets appropriately. Notice that this rewrite is less about refining the revision operation per se than about specifying which tasks should be defined for such an operation to be performed.

Argument Rewrite 2 *An unconstrained attack-based revision operation is characterised by sets of axioms $\mathcal{R}$ and $\mathcal{A}$ that will be removed from and added to Π with respect to the argument A being attacked, such that the resulting set is a theory (that is, a consistent set of axioms).*

$$\Pi \overset{\phi^{\Pi,A}}{\leadsto} \Pi' \quad \Rightarrow \quad \begin{array}{l} \mathit{select}(A, \Pi, \mathcal{R}, \mathcal{A}), \\ \mathit{revise}(\Pi, \mathcal{R}, \mathcal{A}, \Pi') \end{array}$$

Properties:	$\{\ \mathit{consistent}(\Pi')\ \}$
Conditions:	true

7.3.3 Types of Argument Claims

Rewrites in this section allow arguments to be rewritten in terms of the general claims they support. Like the previous rule, they do not specify how to refine the revision operation itself, but are useful for harnessing the possible claims that can be supported by an argument. Given an argument A, these claims can be selected from the possible attack points in the corresponding structure $\mathcal{G}_A$ of argument claims.[3]

Argument Rewrite 3 *An argument A in a theory Π may support the claim that sentence X is substantiated.*

$$\mathit{argument}(A, \Pi) \quad \Rightarrow \quad \mathit{in}(X, A, \Pi)$$

Properties:	$\{\}$
Conditions:	$X : \mathbf{in} \in \mathcal{G}_A$

Argument Rewrite 4 *An argument A in a theory Π may support the claim that a sentence X is not substantiated.*

[3] During a dynamic argument, claims can be selected from the overall dependency graph $\mathcal{D}$, which by definition (see Definition 6.4) includes the claims supported by the last advanced argument. See Section 6.3 for more details.

$$argument(A, \Pi) \quad \Rightarrow \quad out(X, A, \Pi)$$

Properties: $\{\}$

Conditions: $X : \mathbf{out} \in \mathcal{G}_A$

7.3.4 From Contradictory Claims to General Types of Revision

This section gives rewrites for capturing the general purpose of revision operations in terms of the contradictions they generate. Based on Definition 6.3, these rewrites represent the first level of instantiation of revision operations in our classification. The type of property that is accumulated here can be used to ensure that the argument to be generated supports the intended contradiction, and also that it is valid in the context of moves advanced so far. (e.g. that it is consistent and has not been presented before under the same circumstances).

According to Definition 4.6, attack-based operations may depend on the theory and the argument (and consequently on a claim supported by it) to be attacked. These have been denoted so far as superscript symbols, but here we express them as extra parameters in the predicate for selecting the sets of axioms that characterise a revision.

Argument Rewrite 5 *A revision to* Π *can be defined by a predicate that selects the sets* $\mathcal{A}$ *and* $\mathcal{R}$ *based on an argument* A *in* Π*, with the purpose of rejecting this argument. If* A *substantiates a sentence* X*, the attack may consist in refuting* A *so that in the revised theory it no longer substantiates* X*.*

$$\begin{array}{rcl} in(X, A, \Pi) & \Rightarrow & in(X, A, \Pi) \\ select(A, \Pi, \mathcal{R}, \mathcal{A}) & \Rightarrow & remove\_argument(X, A, \Pi, \mathcal{R}, \mathcal{A}) \\ argument(A', \Pi') & \Rightarrow & out(X, A', \Pi') \end{array}$$

Properties: $\{\ supports(A', X : \mathbf{out}, \Pi')\ \}$

Conditions: $A' = A$

Argument Rewrite 6 *A revision to* Π *can be defined by a predicate that selects the sets* $\mathcal{A}$ *and* $\mathcal{R}$ *based on an argument* A *in* Π*, with the purpose of introducing a justification* A' *which attacks* A*. If* A *substantiates a sentence* X*, then* A' *may substantiate a conflicting* *sentence Y in the revised theory.*

$$\begin{array}{rcl} in(X,A,\Pi) & \Rightarrow & in(X,A,\Pi) \\ select(A,\Pi,\mathcal{R},\mathcal{A}) & \Rightarrow & add\_argument(Y,A,\Pi,\mathcal{R},\mathcal{A}) \\ argument(A',\Pi') & \Rightarrow & in(Y,A',\Pi') \end{array}$$

Properties: $\{\ supports(A',Y:\mathbf{in},\Pi')\ \}$

Conditions: $Y \in conflict(X)$

Argument Rewrite 7 *A revision on Π can be defined by a predicate that selects the sets $\mathcal{A}$ and $\mathcal{R}$ based on an argument A in Π, with the purpose of introducing a justification A' which attacks A. If A supports the claims that a sentence X is unsubstantiated, then A' may substantiate X in the revised theory.*

$$\begin{array}{rcl} out(X,A,\Pi) & \Rightarrow & out(X,A,\Pi) \\ select(A,\Pi,\mathcal{R},\mathcal{A}) & \Rightarrow & add\_argument(X,A,\Pi,\mathcal{R},\mathcal{A}) \\ argument(A',\Pi') & \Rightarrow & in(X,A',\Pi') \end{array}$$

Properties: $\{\ supports(A',X:\mathbf{in},\Pi')\ \}$

Conditions: *true*

7.3.5 From Dealing with Arguments to Dealing with Premises

The rules in this section relate the general types of revision for introducing or withdrawing an argument with fundamental types of operation—namely *trivial*, *elementary* and *updating* (see Definition 4.5 and Section 5.2). These are fundamental in the sense that they represent the minimum changes necessary for adding or removing an argument, and more complex operations can be defined by expanding the sets $\mathcal{R}$ and $\mathcal{A}$ in a way that the associated properties still hold.

Argument Rewrite 8 *A revision $(\mathcal{R},\mathcal{A})$ for introducing a justification for X (based on argument A in Π) may be a trivial operation.*

$$add\_argument(X,A,\Pi,\mathcal{R},\mathcal{A}) \quad \Rightarrow \quad trivial(\mathcal{R},\mathcal{A})$$

Properties: $\{\}$

Conditions: $\mathcal{R}=\emptyset$, $\mathcal{A}=\emptyset$

Argument Rewrite 9 *A revision* $(\mathcal{R}, \mathcal{A})$ *for introducing a justification for* X *(based on an argument* A *in* Π*) may be an elementary operation that justifies* X *by adding a premise* P *to the theory. So* $\mathcal{R}$ *is empty, and* $\mathcal{A}$ *is a singleton containing* P*.*

$$\textit{add\_argument}(X, A, \Pi, \mathcal{R}, \mathcal{A}) \quad \Rightarrow \quad \textit{elementary}(\textit{justify}(X), A, \Pi, P)$$

Properties: $\{\}$

Conditions: $\mathcal{R} = \emptyset,$
$\mathcal{A} = \{P\}$

Argument Rewrite 10 *A revision* $(\mathcal{R}, \mathcal{A})$ *for removing a justification* A *for* X *in* Π *may be an elementary operation that refutes* X *by removing a premise* P *from the theory. So* $\mathcal{R}$ *is a singleton containing* P*, and* $\mathcal{A}$ *is empty.*

$$\textit{remove\_argument}(X, A, \Pi, \mathcal{R}, \mathcal{A}) \quad \Rightarrow \quad \textit{elementary}(\textit{refute}(X), A, \Pi, P),$$

Properties: $\{\}$

Conditions: $\mathcal{R} = \{P\},$
$\mathcal{A} = \emptyset$

Argument Rewrite 11 *A revision* $(\mathcal{R}, \mathcal{A})$ *for introducing a justification for* X *(based on an argument* A *in* Π*) may be an updating operation that justifies* X *by removing a premise* P *from the theory and adding an updated axiom* P'*. So* $\mathcal{R}$ *and* $\mathcal{A}$ *are singletons containing* P *and* P'*, respectively.*

$$\textit{add\_argument}(X, A, \Pi, \mathcal{R}, \mathcal{A}) \quad \Rightarrow \quad \textit{updating}(\textit{justify}(X), A, \Pi, P, P')$$

Properties: $\{\}$

Conditions: $\mathcal{R} = \{P\},$
$\mathcal{A} = \{P'\}$

Argument Rewrite 12 *A revision* $(\mathcal{R}, \mathcal{A})$ *for removing a justification* A *for* X *in* Π *may be an updating operation that refutes* X *by removing a premise* P *from the theory and adding an updated axiom* P'*. So* $\mathcal{R}$ *and* $\mathcal{A}$ *are singletons containing* P *and* P'*, respectively.*

$$remove\_argument(X, A, \Pi, \mathcal{R}, \mathcal{A}) \quad \Rightarrow \quad updating(refute(X), A, \Pi, P, P')$$

$$\begin{array}{ll} \textsf{Properties:} & \{\} \\ \textsf{Conditions:} & \mathcal{R} = \{P\}, \\ & \mathcal{A} = \{P'\} \end{array}$$

7.3.6 Logic-Specific Rules for Specifying Premises

Rewrites in this section further refine sets $\mathcal{R}$ and $\mathcal{A}$ in elementary and updating revisions via predicates that specify the premises in these sets accordingly.

The predicate *fact* for example gives the sorts of facts that can be added to the theory by an elementary revision intended to justify a sentence X—namely any axiom of the form $H \leftarrow true$ such that X and H are unifiable, and H is an atom from $\mathcal{L}$.

These sorts of rewrites are logic-specific because they rely on the syntax and mechanisms of (general) logic programs to define the shape and structure of these premises. General program clauses are denoted here by $H \leftarrow \mathbf{B}$, where H is a positive literal and $\mathbf{B}$ is a conjunction of literals. Individual literals are denoted by the (possibly indexed) letter B. A substitution $\sigma \in subst$ that represents the most general unifier between two sentences is denoted by mgu.[4]

Some predicates in these rewrites might require interaction with a user to supply key components, for instance for introducing new literals or axioms and defining substitutions. There are no difficulties in selecting premises to be removed from the theory because this is a finite set which can be easily traversed, but determining exactly the components of a new premise is likely to depend on domain information. What we do at this point is to describe the general shape of new axioms, which can be further instantiated by domain-specific schemata.

The level of classification in this section corresponds to the schemata to which domain-specific knowledge was applied in Chapter 5. For comparison we refer to the corresponding informal schemata between parentheses.

[4] Please refer to Appendix A for the definition of syntax adopted in this section.

Argument Rewrite 13 (Informal Schema 1) *An elementary operation intended to justify X may be established by adding a fact $H \leftarrow true$ such that X and H are unifiable.*

$$elementary(justify(X), A, \Pi, P) \quad \Rightarrow \quad add(fact(P))$$

$$\text{Properties:} \qquad \{\ unify(X, H)\ \}$$

$$\text{Conditions:} \qquad \begin{array}{l} H \in \mathcal{L}, \\ P = H \leftarrow true \end{array}$$

Argument Rewrite 14 (Informal Schema 2) *An elementary operation intended to justify X may be established by adding a substantiated clause $H \leftarrow \mathbf{B}$ to the theory that allows X to be deduced.*

$$elementary(justify(X), A, \Pi, P) \quad \Rightarrow \quad add(substantiated\_rule(P))$$

$$\text{Properties:} \qquad \left\{ \begin{array}{c} unify(X, H), \\ satisfiable(\mathbf{B}\sigma, \Pi) \end{array} \right\}$$

$$\text{Conditions:} \qquad \begin{array}{l} H, \mathbf{B} \in \mathcal{L}, \\ P = H \leftarrow \mathbf{B}, \\ \sigma = mgu(X, H) \end{array}$$

Argument Rewrite 15 (Informal Schema 3) *An elementary operation intended to justify X may be established by adding a rule $H \leftarrow not\ B$ that gives X because B cannot be derived when H unifies with X.*

$$elementary(justify(X), A, \Pi, P) \quad \Rightarrow \quad add(burden\_shift\_rule(P))$$

$$\text{Properties:} \qquad \left\{ \begin{array}{c} unify(X, H), \\ \neg satisfiable(B\sigma, \Pi) \end{array} \right\}$$

$$\text{Conditions:} \qquad \begin{array}{l} H, B \in \mathcal{L}, \\ P = H \leftarrow not\ B, \\ \sigma = mgu(X, H) \end{array}$$

Argument Rewrite 16 (Informal Schema 4) *An elementary operation intended to refute X (by rejecting the argument A supporting it) may be established by removing the clause $H \leftarrow \mathbf{B}$ used in A to derive X because this is an invalid rule.*

$$elementary(refute(X), A, \Pi, P) \quad \Rightarrow \quad retract(invalid\_rule(P))$$

Properties: $\{\ unify(X, H)\ \}$

Conditions:
$H \leftarrow \mathbf{B} \in A,$
$P = H \leftarrow \mathbf{B},$
$\exists \sigma' \in subst.\ affirm(\mathbf{B}\sigma' \wedge not(H\sigma'))$

Argument Rewrite 17 (Informal Schema 5) *An elementary operation intended to refute X (by rejecting the argument A supporting it) may be established by removing the clause $H \leftarrow \mathbf{B}$ used in A to derive X because this is a weak rule.*

$$elementary(refute(X), A, \Pi, P) \quad \Rightarrow \quad retract(weak\_rule(P))$$

Properties: $\{\ unify(X, H)\ \}$

Conditions:
$H \leftarrow \mathbf{B} \in A,$
$P = H \leftarrow \mathbf{B},$
$\exists \sigma' \in subst.\ affirm(not(\mathbf{B}\sigma'))$

Argument Rewrite 18 (Informal Schema 6) *An elementary operation intended to refute X (by rejecting the argument A supporting it) may be established by removing the clause $H \leftarrow \mathbf{B}$ in A used to derive X because it expresses a mistaken correlation.*

$$elementary(refute(X), A, \Pi, P) \quad \Rightarrow \quad retract(misrelation(P))$$

Properties: $\{\ unify(X, H)\ \}$

Conditions:
$H \leftarrow \mathbf{B} \in A,$
$P = H \leftarrow \mathbf{B},$
$\exists \sigma', \sigma'' \in subst.$
$\quad affirm(\mathbf{B}\sigma' \wedge not(H\sigma') \wedge H\sigma'' \wedge not(\mathbf{B}\sigma''))$

Argument Rewrite 19 (Informal Schema 7) *An updating operation intended to justify X may be established by removing a clause from Π, and adding a variant obtained from this by dismissing some precondition that was blocking the derivation of X.*

$$updating(justify(X), A, \Pi, P, P') \quad \Rightarrow \quad \begin{array}{l} retract(irrelevance(P)), \\ add(irrelevance(P')) \end{array}$$

Properties: $\left\{ \begin{array}{c} unify(X, H), \\ satisfiable((B_1 \wedge ... \wedge B_{i-1} \wedge B_{i+1} \wedge ... \wedge B_m)\sigma, \Pi), \\ \neg satisfiable(B_i\sigma, \Pi) \end{array} \right\}$

Conditions:
$H \leftarrow B_1 \wedge ... \wedge B_m \in \Pi,$
$P = H \leftarrow B_1 \wedge ... \wedge B_m,$
$B_i \in \{B_1, ..., B_m\},$
$P' = H \leftarrow B_1 \wedge ... \wedge B_{i-1} \wedge B_{i+1} \wedge ... \wedge B_m,$
$\sigma = mgu(X, H)$

Argument Rewrite 20 (Informal Schema 12) *An updating operation intended to refute X (by rejecting the argument A supporting it) may be established by removing the clause used in A to derive X and adding an elaborated variant containing an extra premise which is not satisfiable, thus blocking the derivation of X.*

$$updating(refute(X), A, \Pi, P, P') \quad \Rightarrow \quad \begin{array}{l} retract(elaboration(P)), \\ add(elaboration(P')) \end{array}$$

Properties: $\left\{ \begin{array}{c} unify(X, H), \\ satisfiable((B_1 \wedge ... \wedge B_m)\sigma, \Pi) \\ \neg satisfiable(B\sigma, \Pi) \end{array} \right\}$

Conditions:
$H \leftarrow B_1 \wedge ... \wedge B_m \in A,$
$P = H \leftarrow B_1 \wedge ... \wedge B_m,$
$B \in \mathcal{L},$
$i \in \{0, ..., m\},$
$P' = H \leftarrow B_1 \wedge ... \wedge B_i \wedge B \wedge B_{i+1} \wedge ... \wedge B_m,$
$\sigma = mgu(X, H)$

Argument Rewrite 21 (Informal Schema 11) *An updating operation intended to justify X may be established by removing a clause from Π and adding a variant that allows X to be inferred, generalising the original rule so that the set of ground instances of the original rule is smaller than the set of ground instances of the variant rule.*

$$\begin{array}{lcl} \mathit{updating}(\mathit{justify}(X), A, \Pi, P, P') & \Rightarrow & \mathit{retract}(\mathit{generalisation}(P)), \\ & & \mathit{add}(\mathit{generalisation}(P')) \end{array}$$

$$\begin{array}{ll} \textsf{Properties:} & \left\{ \begin{array}{c} \mathit{unify}(X, H\sigma'), \\ \mathit{satisfiable}((\mathbf{B}\sigma')\sigma, \Pi), \\ \mathit{ground}(P, \Pi) \subset \mathit{ground}(P', \Pi) \end{array} \right\} \\ \textsf{Conditions:} & \begin{array}{l} H \leftarrow \mathbf{B} \in \Pi, \\ P = H \leftarrow \mathbf{B}, \\ \sigma' \in \mathit{inverse\_subst}, \\ P' = (H \leftarrow \mathbf{B})\sigma', \\ \sigma = \mathit{mgu}(X, H\sigma') \end{array} \end{array}$$

Argument Rewrite 22 (Informal Schema 10) *An updating operation intended to refute X (by rejecting the argument A for it) may be established by removing the clause used in A to derive X and adding a variant that blocks the derivation of X, specialising the original rule so that the set of ground instances of the original rule is greater than the set of ground instances of the variant rule. Derivation of X can fail for two reasons: either because X no longer unifies with the head of the new rule or, if it does, because the body is not satisfiable.*

$$\begin{array}{lcl} \mathit{updating}(\mathit{refute}(X), A, \Pi, P, P') & \Rightarrow & \mathit{retract}(\mathit{specialisation}(P)), \\ & & \mathit{add}(\mathit{specialisation}(P')) \end{array}$$

$$\begin{array}{ll} \textsf{Properties:} & \left\{ \begin{array}{c} \mathit{unify}(X, H), \\ \mathit{ground}(P', \Pi) \subset \mathit{ground}(P, \Pi), \\ \forall (H_g \leftarrow \mathbf{B}_g) \in \mathit{ground}(P\sigma, \Pi) \cap \mathit{ground}(P', \Pi). \\ \neg \mathit{satisfiable}(\mathbf{B}_g, \Pi) \end{array} \right\} \\ \textsf{Conditions:} & \begin{array}{l} H \leftarrow \mathbf{B} \in A, \\ P = H \leftarrow \mathbf{B}, \\ \sigma = \mathit{mgu}(X, H), \\ \sigma' \in \mathit{subst}, \\ P' = (H \leftarrow \mathbf{B})\sigma' \end{array} \end{array}$$

Argument Rewrite 23 (Informal Schema 8) *An updating operation intended to justify X may be established by removing a clause from Π and adding a variant that revises the original conclusion, so that X can now be inferred.*

$$
\begin{array}{rcl}
updating(justify(X), A, \Pi, P, P') & \Rightarrow & retract(misconclusion(P)), \\
 & & add(misconclusion(P'))
\end{array}
$$

$$
\begin{array}{ll}
\text{Properties:} & \left\{ \begin{array}{c} unify(X, H'), \\ satisfiable(\mathbf{B}\sigma, \Pi) \end{array} \right\} \\
\text{Conditions:} & \begin{array}{l} H \leftarrow \mathbf{B} \in \Pi, \\ P = H \leftarrow \mathbf{B}, \\ H' \in \mathcal{L}, \\ P' = H' \leftarrow \mathbf{B}, \\ \sigma = mgu(X, H') \end{array}
\end{array}
$$

Argument Rewrite 24 (Informal Schema 8) *An updating operation intended to refute X (by rejecting the argument A for it) may be established by the removing the clause used in A to derive X and adding a variant that revises the original conclusion, so that X no longer follows.*

$$
\begin{array}{rcl}
updating(refute(X), A, \Pi, P, P') & \Rightarrow & retract(misconclusion(P)), \\
 & & add(misconclusion(P'))
\end{array}
$$

$$
\begin{array}{ll}
\text{Properties:} & \left\{ \begin{array}{c} unify(X, H), \\ \neg unify(X, H') \end{array} \right\} \\
\text{Conditions:} & \begin{array}{l} H \leftarrow \mathbf{B} \in A, \\ P = H \leftarrow \mathbf{B}, \\ H' \in \mathcal{L}, \\ P' = H' \leftarrow \mathbf{B} \end{array}
\end{array}
$$

Argument Rewrite 25 (Informal Schema 9) *An updating operation intended to justify X may be established by removing a clause from Π and adding the reversed rule so that X can be inferred.*

$$
\begin{array}{rcl}
updating(justify(X), A, \Pi, P, P') & \Rightarrow & retract(reversion(P)) \\
 & & add(reversion(P'))
\end{array}
$$

$$
\begin{array}{ll}
\text{Properties:} & \left\{ \begin{array}{c} unify(X, B), \\ satisfiable(H\sigma, \Pi) \end{array} \right\} \\
\text{Conditions:} & \begin{array}{l} H \leftarrow B \in \Pi, \\ P = H \leftarrow B, \\ \sigma = mgu(X, B), \\ P' = B \leftarrow H \end{array}
\end{array}
$$

Argument Rewrite 26 (Informal Schema 9) *An updating operation intended to refute X (by rejecting the argument A for it) may be established by the removing the clause used in A to derive X and adding the reversed rule so that X no longer follows.*

$$\begin{array}{lll} updating(refute(X), A, \Pi, P, P') & \Rightarrow & retract(reversion(P)) \\ & & add(reversion(P')) \\ & & \\ & \text{Properties:} & \left\{ \begin{array}{c} unify(X,H), \\ \neg unify(X,B) \end{array} \right\} \\ & & \\ & \text{Conditions:} & \begin{array}{l} H \leftarrow B \in A, \\ P = H \leftarrow B, \\ P' = B \leftarrow H \end{array} \end{array}$$

7.3.7 Domain-Specific Level

Figure 7.2 depicts the organised collection of rewrites up to the logic-specific level, where predicates give the general shape of the clauses to be added and removed, thus expressing standard types of revisions in argument. Appendix C gives the possible schemata for argument revision obtained from this classification.

Notice that in practice not all the conditions in the rewrites can be satisfied in a straightforward way, especially if they involve the selection of elements from infinite or unspecified sets. For instance, deciding exactly which literals or substitutions instantiate certain schemata is likely to be dependent on the domain, as illustrated in Chapter 5. The next level in the classification should then be composed of domain-specific schemata, from which we can construct libraries of possible revisions for generating dynamic arguments automatically.

A worked example in the next chapter illustrates one way in which this classification can be used to define possible revision operations in a particular domain.

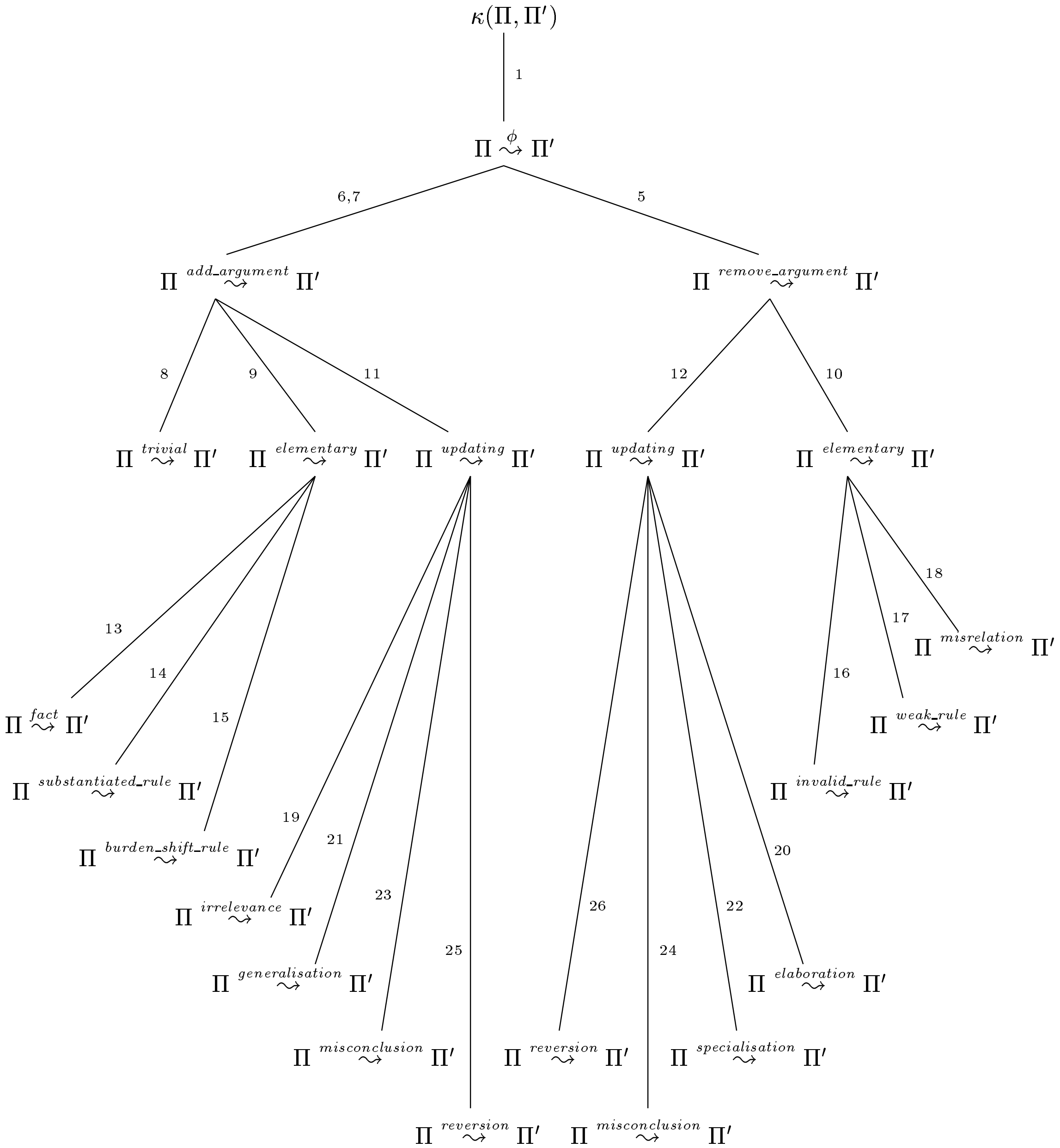


Figure 7.2: Organisation of argument revision schemata obtained via our rewriting system. Schemata 2, 3 and 4 are not depicted in the diagram because they have no immediate effect on refining a revision operation, but are still useful for harnessing the possible revisions that are allowed.

Chapter 8

Worked Example: Defining Domain-Specific Schemata

The system of rewrites in Section 7.3 not only allows harnessing of argument rewriting,[1] but also provides a technique for systematically generating attacks in dynamic argumentation systems like the one in Figure 7.1. This chapter describes how examples from the *aflatoxin debate* in Chapter 5 could be modelled in this dynamic argument framework.

8.1 Two Dynamic Argumentation Systems

Figure 7.1 gives the general form of a system for dynamic argumentation that explores possible attacks to a sentence and converges when no more attacks can be generated. Such a system constitutes the essence of the implementations we developed, two of which we demonstrate in this chapter. We have implemented these systems in Prolog as described in Figure 8.1, which gives the top-level clauses corresponding to the specification in Figure 7.1, here with an extra argument for recording the sequence of moves.

In summary, the predicate `dynamic_arg/4` generates dynamic arguments about a particular sentence given an initial theory, thus producing a revised theory that is more acceptable with respect to the sentence only if all attacks to it have been dismissed. Here theories are represented as lists of axiom; new axioms are added at the end of the

[1] See Appendix C and Figure 7.2.

```
%--------------------------------------------------------
% dynamic_arg(X, TInit, T, D) :-
%   D is a dynamic argument about a theory TInit with
%   respect to a sentence X, that converges to theory T

dynamic_arg(X, TInit, T, D) :-
    initialise(X, TInit, DInit),
    dynamic_arg(X, TInit, T, DInit, D).

dynamic_arg(X, TNow, T, DSofar, D) :-
    gen_attack(X, TNow, TNext, DSofar, NewDSofar),
    dynamic_arg(X, TNext, T, NewDSofar, D).
dynamic_arg(X, T, T, D, D).
```

Figure 8.1: Prolog specification of a generic dynamic argumentation system.

list whereas updated premises just replace the original ones. But like in sets, there are no duplicate entries of the same axiom.

The extra parameter `D` is a structured term comprising both the sequence of arguments and revisions $\langle A_0, \phi_1, A_1, ..., \phi_i, A_i \rangle$ advanced so far, as well as the current dependency graph $\mathcal{D}_i$. In Prolog terms, `D` (or `DSofar`) is represented as follows:

```
d([Ai, Ri,..., A1, R1, A0], Di).
```

The first parameter has the sequence of arguments in reversed order in an accumulator style, as it used to accumulate information on the way down through the recursion. The predicate `initialise/3` instantiates this term to:

```
d([A0], D0)
```

by generating an initial justification A_0 for X, and initialising the dependency tree $\mathcal{D}_0$ with the corresponding structure of argument claims. Predicate `dynamic_arg/5` then recursively explores the possible attacks via `gen_attack/5` until no more attacks can be generated, and so the final instantiation of `D` occurs.

The crucial question then is how to define the predicate `gen_attack/5` appropriately. In what follows we briefly describe two ways for doing that.

8.1.1 Generating Attacks Interactively

One possibility is for `gen_attack/5` to explore the attack relation by going down the classification level in Figure 7.2 and querying for appropriate information as it reaches choice points, namely:

- which rewrite rule to apply at each level; and
- how to instantiate the conditions in the rewrite.

In the latter case, interaction happens exactly at stages where an element must be selected from a set—that is, when $\in$-conditions need to be satisfied.

Once all the necessary information has been supplied, the system performs the corresponding revision, generates the new attacking argument and checks the relevant properties that were accumulated down the schemata classification. Because in this way it is always possible to come up with a new attack, the process only terminates once the user decides not to attack the last advanced argument.

This system is highly flexible and interactive, and is mostly intended to illustrate the concepts introduced in the previous chapters. Its use is demonstrated in Section 8.2.

8.1.2 Generating Attacks Automatically

Another possibility is to allow the systematic search of possible sequences of argument exchange, in which case `gen_attack/5` constructs attacks automatically from a predefined catalogue Φ of argument revision schemata[2] rather than by interactively going down the classification tree of possible revisions. Libraries of revision schemata are composed of *flattened* revisions, as described in Appendix C. These represent the general format of attacks, with the properties accumulated down the corresponding path in the classification and maybe some domain-specific information incorporated appropriately.

At each step the claims constituting the possible points of attack can be calculated from the current dependency graph, and the system selects one of these such that it matches

[2] See Definition 4.7.

some revision schema in Φ (i.e such that there is a schema in Φ that can be used to attack the claim). An argument is then generated, and the corresponding properties of the applied schema can guarantee that it supports the intended attack in the context of the arguments advanced so far.[3] The dynamic argument terminates once no more attacks can be constructed from the schemata in Φ.

We illustrate the use of this system in Section 8.3.

8.2 The Aflatoxin Debate Revisited

To reconstruct the examples given in Chapter 5 we use the interactive argumentation system from Section 8.1.1. For clarity of presentation we cast the output of this system into an *easier-to-read* format, representing argument trees and other structured terms graphically and using different font types to reproduce the interaction between the system and the user: for instance, sans serif and *italics* are used to denote requests for information by the system and *information supplied by the user*, respectively.

The following is an argument process, as generated by the system, about the FDA policy that restricts aflatoxin levels to 20*ppb*.

The initial theory *TInit* is represented by the following general logic program:

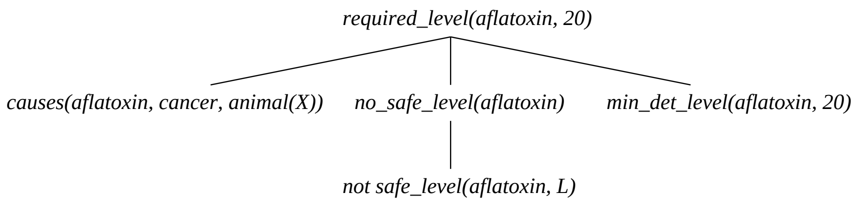
$$\begin{array}{rcl} min\_det\_level(aflatoxin, 20) & \leftarrow & true \\ causes(aflatoxin, cancer, animal(X)) & \leftarrow & true \\ required\_level(Ag, L) & \leftarrow & causes(Ag, cancer, X) \wedge \\ & & no\_safe\_level(Ag) \wedge \\ & & min\_det\_level(Ag, L) \\ no\_safe\_level(Ag) & \leftarrow & not\ safe\_level(Ag, L) \end{array}$$

According to Definition 4.7, the first argument to be advanced is a justification[4] supporting the main claim $required\_level(aflatoxin, 20)$.

Argument A_0 is a justification for $required\_level(aflatoxin, 20)$.

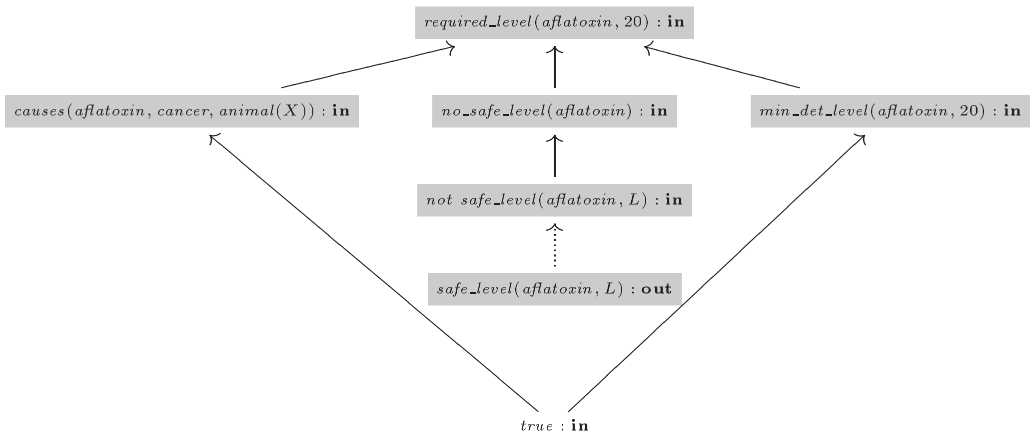
[3] Things that can be checked here include whether the argument is consistent and if it has not been advanced before (so as to avoid circularity). Appendix D describes precisely what it means for an argument to support a claim in the context of a dynamic argument, considering that this argument may be based on a revision.

[4] For clarity of presentation, from now on we omit the term *true* from the representation of argument trees (but not from the dependency graphs).



The dependencies between claims at this initial stage are represented below, with highlighted nodes corresponding to possible attack points.

$\mathcal{D}_0$:



Do you want to attack this argument? (yes/no) *yes*

Revision ϕ_1 is determined interactively as follows.

Enter rewrite choice from the following:

Rewrite 3: select an **in** claim to be attacked

Rewrite 4: select an **out** claim to be attacked

Rewrite 4

Enter **out** claim to be attacked

$\mathit{safe\_level}(\mathit{aflatoxin}, L) : \mathbf{out}$

Enter rewrite choice from the following:

Rewrite 7: add an argument supporting the sentence

Rewrite 7

Enter rewrite choice from the following:

Rewrite 8: perform a trivial revision for justifying the sentence

Rewrite 9: perform an elementary revision for justifying the sentence

Rewrite 11: perform an updating revision for justifying the sentence

Rewrite 9

Enter rewrite choice from the following:

Rewrite 13: justify the sentence by adding a new fact
Rewrite 14: justify the sentence by adding a new substantiated rule
Rewrite 15: justify the sentence by adding a new burden shift rule

Rewrite 13

Enter fact for justifying the sentence

$safe\_level(aflatoxin, s)$

In this way,

$$\phi_1 : add(fact(safe\_level(aflatoxin, s) \leftarrow true))$$

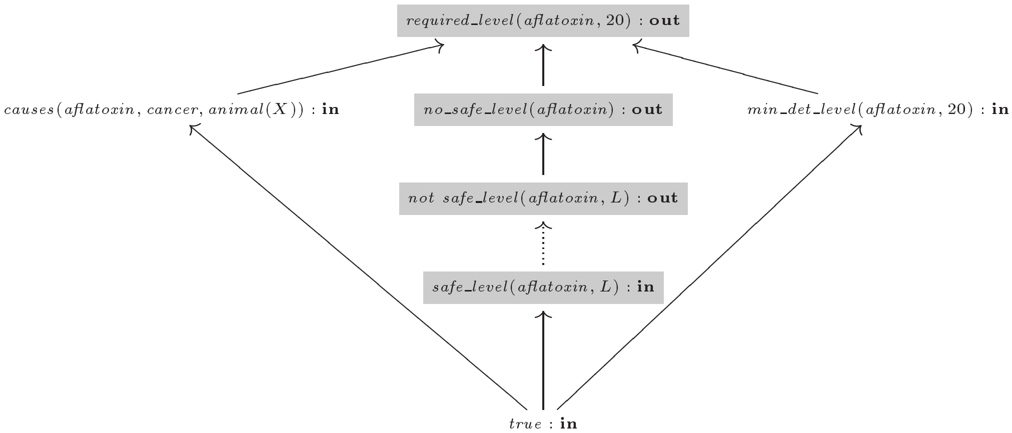
is an attack-based revision that can be used to construct an argument for justifying that a safe exposure level s does exist for aflatoxins, which is far greater than 20ppb. Moreover, the properties accumulated during the instantiation can ensure that the generated argument in fact supports that $safe\_level(aflatoxin, L) : \mathbf{in}$.

Argument A_1 is a justification for $safe\_level(aflatoxin, s)$.

safe_level(aflatoxin, s)

The dependencies between claims at this stage are represented below, again with highlighted nodes corresponding to possible attack points. Remember that the attack points are only those nodes that contribute to the current status of the main sentence.

$\mathcal{D}_1$:



Do you want to attack this argument? (yes/no) *yes*

Revision ϕ_2 can be determined interactively as follows.

Enter rewrite choice from the following:

Rewrite 3: select an **in** claim to be attacked

Rewrite 4: select an **out** claim to be attacked

Rewrite 4

Enter **out** claim to be attacked

$required\_level(aflatoxin, 20)$: **out**

Enter rewrite choice from the following:

Rewrite 7: add an argument supporting the sentence

Rewrite 7

Enter rewrite choice from the following:

Rewrite 8: perform a trivial revision for justifying the sentence

Rewrite 9: perform an elementary revision for justifying the sentence

Rewrite 11: perform an updating revision for justifying the sentence

Rewrite 11

Enter rewrite choice from the following:

Rewrite 19: justify the sentence by dismissing an irrelevant precondition from an existing axiom

Rewrite 21: justify the sentence by generalising an exiting axiom

Rewrite 23: justify the sentence by changing the conclusion of an exiting axiom

Rewrite 25: justify the sentence by reversing an exiting axiom

Rewrite 19

Enter axiom to be updated via the irrelevance schema

$$required\_level(Ag, L) \leftarrow causes(Ag, cancer, X) \wedge$$
$$no\_safe\_level(Ag) \wedge$$
$$min\_det\_level(Ag, L)$$

Enter precondition to be removed

$$no\_safe\_level(Ag)$$

In this way,

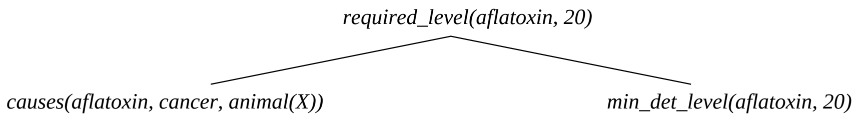
$$\phi_2 : retract(irrelevance\begin{pmatrix} required\_level(Ag, L) \leftarrow \\ \quad causes(Ag, cancer, X) \wedge \\ \quad no\_safe\_level(Ag) \wedge \\ \quad min\_det\_level(Ag, L) \end{pmatrix})$$

and

$$add(irrelevance\begin{pmatrix} required\_level(Ag, L) \leftarrow \\ \quad causes(Ag, cancer, X) \wedge \\ \quad min\_det\_level(Ag, L) \end{pmatrix})$$

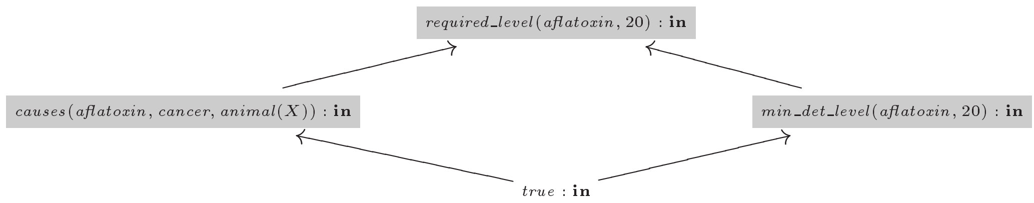
is an attack-based revision allows argument 5.4 to be derived, reinstating the claim that the maximum required level for aflatoxins should be set to 20ppb (see **Informal Schema 7**).

Argument A_2 is a justification for $required\_level(aflatoxin, 20)$.



The dependencies between claims at this stage are represented below, again with highlighted nodes corresponding to possible attack points.

$\mathcal{D}_2$:



Do you want to attack this argument? (yes/no) *yes*

Revision ϕ_3 can be determined interactively as follows.

Enter rewrite choice from the following:

Rewrite 3: select an **in** claim to be attacked

Rewrite 4: select an **out** claim to be attacked

Rewrite 3

Enter **in** claim to be attacked

$required\_level(aflatoxin, 20)$: **in**

Enter rewrite choice from the following:

Rewrite 5: remove the argument supporting the sentence

Rewrite 6: add an argument supporting a conflicting sentence

Rewrite 5

Enter rewrite choice from the following:

Rewrite 10: perform an elementary revision for refuting the sentence
Rewrite 12: perform an updating revision for refuting the sentence

Rewrite 12

Enter rewrite choice from the following:

Rewrite 20: refute the sentence by elaborating the axiom supporting it
Rewrite 22: refute the sentence by specialising the axiom supporting it
Rewrite 24: refute the sentence by changing the conclusion of the axiom supporting it
Rewrite 26: refute the sentence by reversing the axiom supporting it

Rewrite 22

Enter substitution that specialises the axiom
$required\_level(Ag, L) \leftarrow causes(Ag, cancer, X) \ \wedge min\_det\_level(Ag, L)$

$X = human$

In this way,

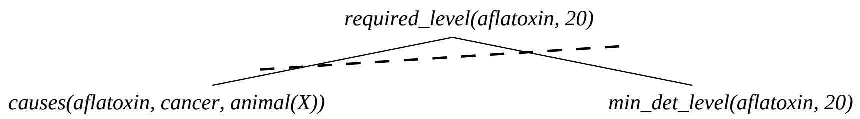
$$\phi_2 : retract(specialisation \begin{pmatrix} required\_level(Ag, L) \leftarrow \\ \quad causes(Ag, cancer, X) \wedge \\ \quad min\_det\_level(Ag, L) \end{pmatrix})$$

and

$$add(specialisation \begin{pmatrix} required\_level(Ag, L) \leftarrow \\ \quad causes(Ag, cancer, human) \wedge \\ \quad min\_det\_level(Ag, L) \end{pmatrix})$$

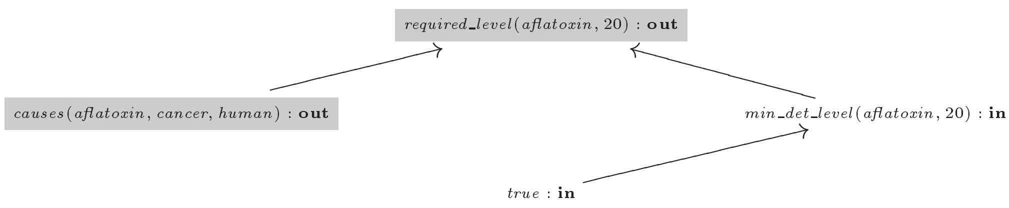
is an attack-based revision that refutes argument A_2 (see **Informal Schema 10**).

Argument A_3 is a refutation of $required\_level(aflatoxin, 20)$.



The dependencies between claims at this stage are represented below, again with highlighted nodes corresponding to possible attack points.

$\mathcal{D}_3$:



Do you want to attack this argument? (yes/no) *yes*

Revision ϕ_4 is determined interactively as follows.

Enter rewrite choice from the following:

Rewrite 3: select an **in** claim to be attacked

Rewrite 4: select an **out** claim to be attacked

Rewrite 4

Enter **out** claim to be attacked

$causes(aflatoxin, cancer, human)$: **out**

Enter rewrite choice from the following:

Rewrite 7: add an argument supporting the sentence

Rewrite 7

Enter rewrite choice from the following:

Rewrite 8: perform a trivial revision for justifying the sentence

Rewrite 9: perform an elementary revision for justifying the sentence

Rewrite 11: perform an updating revision for justifying the sentence

Rewrite 9

Enter rewrite choice from the following:

Rewrite 13: justify the sentence by adding a new fact

Rewrite 14: justify the sentence by adding a new substantiated rule

Rewrite 15: justify the sentence by adding a new burden shift rule

Rewrite 14

Enter head and body of a substantiated rule for justifying the sentence

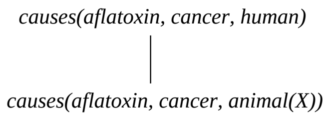
$causes(Ag, P, human)$
$causes(Ag, P, animal(X))$

In this way,

$$\phi_4 : add(substantiated\_rule\left(\begin{array}{r} causes(Ag, P, human) \leftarrow \\ causes(Ag, P, animal(X)) \end{array}\right))$$

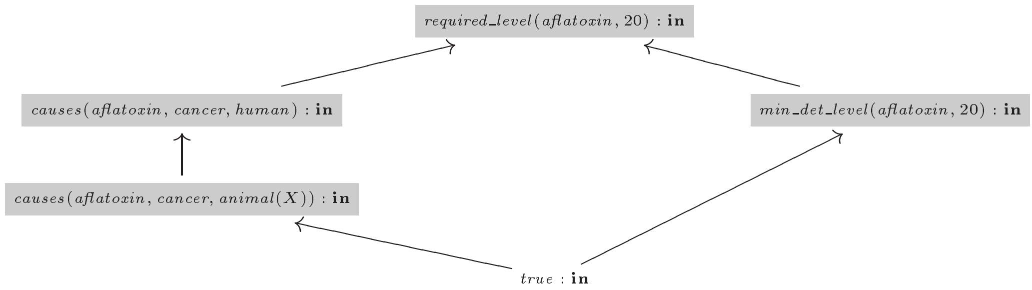
is an attack-based revision that can be used to construct an argument for justifying that aflatoxins cause cancer in humans (see **Informal Schema 2**).

Argument A_4 is a justification for $causes(aflatoxin, cancer, human)$.



The dependencies between claims at this initial stage are represented below, with highlighted nodes corresponding to possible attack points.

$\mathcal{D}_4$:



Do you want to attack this argument? (yes/no) *yes*

Revision ϕ_5 can be determined interactively as follows.

Enter rewrite choice from the following:

Rewrite 3: select an **in** claim to be attacked

Rewrite 4: select an **out** claim to be attacked

Rewrite 3

Enter **in** claim to be attacked

$causes(aflatoxin, cancer, human)$: **in**

Enter rewrite choice from the following:

Rewrite 5: remove the argument supporting the sentence

Rewrite 6: add an argument supporting a conflicting sentence

Rewrite 5

Enter rewrite choice from the following:

Rewrite 10: perform an elementary revision for refuting the sentence

Rewrite 12: perform an updating revision for refuting the sentence

Rewrite 12

Enter rewrite choice from the following:

Rewrite 20: refute the sentence by elaborating the axiom supporting it

Rewrite 22: refute the sentence by specialising the axiom supporting it
Rewrite 24: refute the sentence by changing the conclusion of the axiom supporting it
Rewrite 26: refute the sentence by reversing the axiom supporting it

Rewrite 20

Enter extra literal to be introduced in the axiom
$causes(Ag, P, human) \leftarrow causes(Ag, P, animal(X))$

$similar\_physiology(human, X)$

Enter position in the axiom body in which to introduce the literal (0-1)

1

In this way,

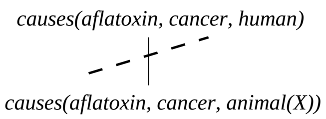
$$\phi_4 : retract(elaboration \begin{pmatrix} causes(Ag, P, human) \leftarrow \\ \quad causes(Ag, P, animal(X)) \end{pmatrix})$$

and

$$add(elaboration \begin{pmatrix} causes(Ag, P, human) \leftarrow \\ \quad causes(Ag, P, animal(X)) \wedge \\ \quad similar\_physiology(human, X) \end{pmatrix})$$

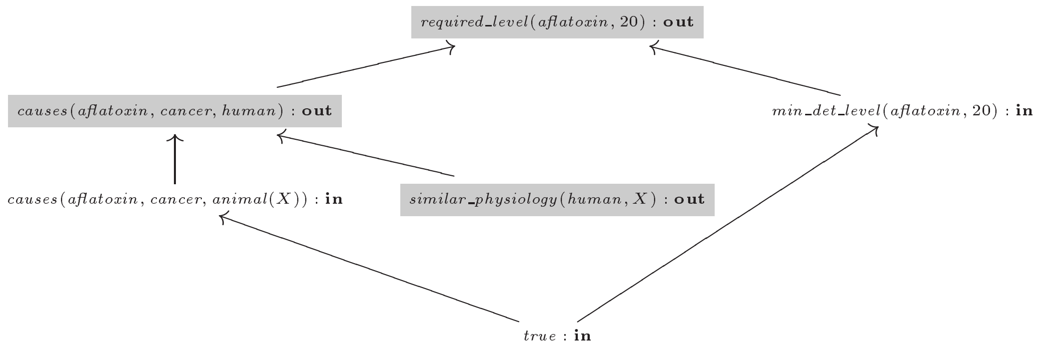
is an attack-based revision that refutes argument A_4 (see **Informal Schema 12**).

Argument A_5 is a refutation of $causes(\mathit{aflatoxin}, cancer, human)$.



The dependencies between claims at this initial stage are represented below, with highlighted nodes corresponding to possible attack points.

$\mathcal{D}_5$:



Do you want to attack this argument? (yes/no) *no*

With the argument terminating at this stage, the revised theory below is said to be unacceptable with respect to the sentence $required\_level(aflatoxin, 20)$:

$$
\begin{array}{rcl}
min\_det\_level(aflatoxin, 20) & \leftarrow & true \\
causes(aflatoxin, cancer, animal(X)) & \leftarrow & true \\
required\_level(Ag, L) & \leftarrow & causes(Ag, cancer, human) \wedge \\
 & & min\_det\_level(Ag, L) \\
no\_safe\_level(Ag) & \leftarrow & not\ safe\_level(Ag, L) \\
safe\_level(aflatoxin, s) & \leftarrow & true \\
causes(Ag, P, human) & \leftarrow & causes(Ag, P, animal(X)) \wedge \\
 & & similar\_physiology(human, X)
\end{array}
$$

This example illustrates the sort of arguments we can automate. Although the system applied here is highly interactive and relies on a great amount of information to be provided by a user, it can be quite useful in analysing specific arguments and exploring the roles of certain types of revision in a domain. There is scope for making use of the revisions defined during this interactive process in order to automatically explore other possible courses of argument.

8.3 Searching for Alternative Arguments

Given that a catalogue Φ of possible attack-based revision schemata has been specified, the system in Section 8.1.2 can then be used to generate dynamic arguments in an automated form. The question, then, is how to specify Φ.

8.3.1 A Catalogue of Argument Schemata for the Aflatoxin Example

One way to define such a catalogue for the aflatoxin example is to consider each schema in Φ to be the *flattened* equivalent of an operation determined during the interactive argumentation. For instance, the following representation of ϕ_1 could be included in Φ:

Domain-Specific Revision Schema ϕ_1: $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 9 \Rightarrow 13$

$$
\begin{array}{c}
out(safe\_level(aflatoxin, L), A, \Pi), \\
add(fact(P)),\ \ revise(\Pi, \{\}, \{P\}, \Pi'), \\
in(safe\_level(aflatoxin, L), A', \Pi')
\end{array}
$$

$$\text{Properties:} \quad \left\{ \begin{array}{c} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', safe\_level(aflatoxin, L) : \mathbf{in}, \Pi'), \\ unify(safe\_level(aflatoxin, L), safe\_level(aflatoxin, s)) \end{array} \right\}$$

$$\text{Conditions:} \quad \begin{array}{l} safe\_level(aflatoxin, s) \in \mathcal{L}, \\ P = safe\_level(aflatoxin, s) \leftarrow true \end{array}$$

This schema is obtained directly from the rewrites used in the interactive system, but it could as well be defined manually by a designer of an argumentation system. It is important to note that we do not require all the properties to be verified, so designers might choose to disregard properties which they feel are redundant or not relevant. Here, for instance, properties like *unify* could be safely dismissed as it is valid independently of the actual revision being performed and the new attack being generated. Also, because we are not considering priorities between arguments, *attacks* holds by definition as the arguments must support contradictory claims (**out** and **in**, respectively). Furthermore, since our choice of formal language does not include classical negation there are no risks of logical inconsistency, so as a designer we can choose not to verify consistency in the revised set of axioms. One crucial property to be tested, though, is that of *supports*, because it guarantees that an attacking argument can in fact be generated and advanced.

Hence in this case the following is an equivalent description of ϕ_1 above.

Domain-Specific Revision Schema ϕ_1: $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 9 \Rightarrow 13$

$$\begin{array}{c} out(safe\_level(aflatoxin, L), A, \Pi), \\ add(fact(P)), \ revise(\Pi, \{\}, \{P\}, \Pi'), \\ in(safe\_level(aflatoxin, L), A', \Pi') \end{array}$$

$$\text{Properties:} \quad \{ \ supports(A', safe\_level(aflatoxin, L) : \mathbf{in}, \Pi') \ \}$$

$$\text{Conditions:} \quad \begin{array}{l} safe\_level(aflatoxin, s) \in \mathcal{L}, \\ P = safe\_level(aflatoxin, s) \leftarrow true \end{array}$$

Another point to be noted here is that conditions for applying the corresponding logic-specific rewrites from Section 7.3.6 still remain. As remarked in the previous section, interaction may happen only in cases where an element must be selected from a set ($\in$-conditions). Although in domain-specific schemata such elements have been determined, conditions are still needed in order instantiate them appropriately throughout the schema.

Similarly the following is a representation of revision ϕ_2.

Domain-Specific Revision Schema ϕ_2: $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 11 \Rightarrow 19$

$$\begin{array}{c} out(required\_level(aflatoxin, 20), A, \Pi), \\ retract(irrelevance(P)),\ \ add(irrelevance(P')),\ \ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ in(required\_level(aflatoxin, 20), A', \Pi') \end{array}$$

$$\text{Properties:} \quad \left\{ \begin{array}{c} supports(A', required\_level(aflatoxin, 20) : \mathbf{in}, \Pi'), \\ unify(required\_level(aflatoxin, 20), required\_level(Ag, L)), \\ satisfiable(B_1 \wedge ... \wedge B_{i-1} \wedge B_{i+1} \wedge ... \wedge B_m, \Pi) \\ \neg satisfiable(no\_safe\_level(aflatoxin), \Pi) \end{array} \right\}$$

$$\text{Conditions:} \quad \begin{array}{l} P = required\_level(Ag, L) \leftarrow B_1 \wedge ... \wedge B_m \in \Pi, \\ B_i = no\_safe\_level(Ag) \in \{B_1, ..., B_m\}, \\ P' = H \leftarrow B_1 \wedge ... \wedge B_{i-1} \wedge B_{i+1} \wedge ... \wedge B_m \end{array}$$

Note that a schema that is obtained from the interactive system is specific to the attack performed in that system, and in this example these are grounded to the case of the required level of *aflatoxin* being 20*ppb*. However, because the attack is based on a more generic statement $required\_level(Ag, L)$, and because the property

$$unify(required\_level(aflatoxin, 20), required\_level(Ag, L)),$$

holds, then schema ϕ_2 can be generalised so as to attack any sentence of the form $required\_level(Ag, L)$:

Domain-Specific Revision Schema ϕ_2: $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 11 \Rightarrow 19$

$$\begin{array}{c} out(required\_level(Ag, L), A, \Pi), \\ retract(irrelevance(P)),\ \ add(irrelevance(P')),\ \ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ in(required\_level(Ag, L), A', \Pi') \end{array}$$

$$\text{Properties:} \quad \left\{ \begin{array}{c} supports(A', required\_level(Ag, L) : \mathbf{in}, \Pi'), \\ satisfiable(B_1 \wedge ... \wedge B_{i-1} \wedge B_{i+1} \wedge ... \wedge B_m, \Pi) \\ \neg satisfiable(no\_safe\_level(Ag), \Pi) \end{array} \right\}$$

$$\text{Conditions:} \quad \begin{array}{l} P = required\_level(Ag, L) \leftarrow B_1 \wedge ... \wedge B_m \in \Pi, \\ B_i = no\_safe\_level(Ag) \in \{B_1, ..., B_m\}, \\ P' = H \leftarrow B_1 \wedge ... \wedge B_{i-1} \wedge B_{i+1} \wedge ... \wedge B_m \end{array}$$

Other operations are described analogously.

Domain-Specific Revision Schema ϕ_3: $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 5 \Rightarrow 12 \Rightarrow 22$

$$\begin{array}{c} in(required\_level(Ag, L), A, \Pi), \\ retract(specialisation(P)),\ add(specialisation(P')),\ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ out(required\_level(Ag, L), A, \Pi') \end{array}$$

Properties: $\left\{ \begin{array}{c} supports(A, required\_level(Ag, L) : \mathbf{out}, \Pi') \\ \neg satisfiable(causes(Ag, cancer, X)\sigma', \Pi) \end{array} \right\}$

Conditions: $P = required\_level(Ag, L) \leftarrow \mathbf{B} \in A,$
$causes(Ag, cancer, X) \in \mathbf{B},$
$\sigma' = [X = human],$
$P' = (H \leftarrow \mathbf{B})\sigma'$

Domain-Specific Revision Schema ϕ_4: $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 9 \Rightarrow 14$

$$\begin{array}{c} out(causes(Ag, P, human), A, \Pi), \\ add(substantiated\_rule(P)),\ revise(\Pi, \{\}, \{P\}, \Pi'), \\ in(causes(Ag, P, human), A', \Pi') \end{array}$$

Properties: $\left\{ \begin{array}{c} supports(A', causes(Ag, P, human) : \mathbf{in}, \Pi') \\ satisfiable(causes(Ag, P, animal(X)), \Pi) \end{array} \right\}$

Conditions: $causes(Ag, P, human), causes(Ag, P, animal(X)) \in \mathcal{L},$
$P = causes(Ag, P, human) \leftarrow causes(Ag, P, animal(X))$

Domain-Specific Revision Schema ϕ_5: $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 5 \Rightarrow 12 \Rightarrow 20$

$$\begin{array}{c} in(causes(Ag, P, human), A, \Pi), \\ retract(elaboration(P)),\ add(elaboration(P')),\ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ out(causes(Ag, P, human), A, \Pi') \end{array}$$

Properties: $\left\{ \begin{array}{c} supports(A, causes(Ag, P, human) : \mathbf{out}, \Pi') \\ satisfiable(B_1 \wedge ... \wedge B_m, \Pi), \\ \neg satisfiable(similar\_physiology(human, X), \Pi) \end{array} \right\}$

Conditions: $P = causes(Ag, P, human) \leftarrow \mathbf{B} \in A,$
$causes(Ag, P, animal(X)) \in \mathbf{B},$
$B = similar\_physiology(human, X) \in \mathcal{L},$
$P' = causes(Ag, P, human) \leftarrow \mathbf{B} \wedge B$

In this way, Φ can be defined as the following set:

$$\Phi = \{\phi_1, \phi_2, \phi_3, \phi_4, \phi_5\}.$$

8.3.2 Exploring the Search Space of Arguments

The system in Section 8.1.2 can now be used to explore the search space of arguments. Moreover, given the selection of possible revision schemata, we expect the system to be able to re-generate the dynamic argumentation that was constructed interactively in Section 8.2:

$$\langle A_0, \phi_1, A_1, \phi_2, A_2, \phi_3, A_3, \phi_4, A_4, \phi_5, A_5 \rangle.$$

This argument in particular does not succeed in defending the FDA policy for restricting aflatoxin levels to 20*ppb*. It would be interesting, however, to see whether other courses of argument—if they exist—yield the same conclusion.

The system takes advantage of the fact that complicated choice points (such as selecting an element from an unspecified or infinite set) have already been explored by the interactive system and resolved in the schemata in Φ. The search space of possible arguments can be exhaustively explored by traversing well defined sets: at each step i of the process the system selects one possible claim to be attacked (from $\mathcal{D}_i$) and one matching argument schema (from Φ) that gives an attack to this claim. For the same initial theory *TInit* from Section 8.2, the query:

```
| ?- findall(D, dynamic_arg(required_level(aflatoxin, 20), TInit, T, D), AllD).
```

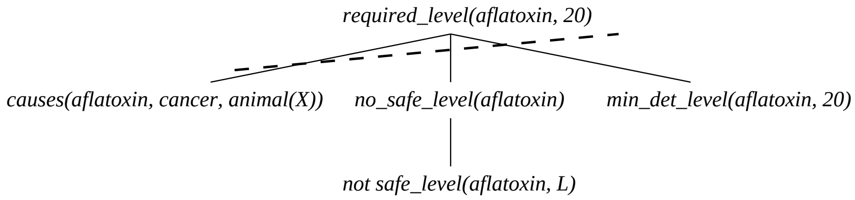
gives three possible dynamic arguments based on Φ:

$$\langle A_0, \phi_1, A_1, \phi_2, A_2, \phi_3, A_3, \phi_4, A_4, \phi_5, A_5 \rangle$$

$$\langle A_0, \phi_3, A_3', \phi_4, A_4, \phi_5, A_5 \rangle$$

$$\langle A_0, \phi_3, A_3', \phi_4, A_4, \phi_1, A_1, \phi_2, A_2, \phi_5, A_5 \rangle$$

where A_3' is the following refutation of A_0.



The resulting theories are different for each case,[5] and $required\_level(aflatoxin, 20)$ is not established in any of them.

It seems also that generic trivial revisions[6] should always be included in the library of possible revisions, so inherent conflicts and alternative justifications for a sentence can be explored automatically. In this example, however, no argument can be generated based on trivial revisions, as no conflicts are explicitly defined and no two alternative arguments for any relevant sentence coexist. Every possible attack consists in either blocking a derivation or introducing a new justification.

An issue arises here. We have shown that domain-specific schemata can be obtained from arguments that are constructed interactively, but these may be over-specified. Take for instance schema ϕ_1. Rather than committing to a particular safe level s, we could leave this as an open parameter to be automatically instantiated during the argumentation. This means we need to refine our framework for dynamic argumentation in order to incorporate special mechanisms that provide the necessary information for instantiation. This is quite an important point, as automating this process is not only crucial for understanding dynamics in argument, but it is also useful for (autonomous) agents that want to apply this technique to particular problems.

Because in these cases we may know less about specific revisions, we need to know more about the consequences of applying certain types of revision. The next chapter includes an investigation of desirable properties that libraries of revisions can have, and how these can affect the automatic generation of arguments.

---

[5] Considering that theories are implemented as lists, $\Pi_{\phi_1\phi_2\phi_3\phi_4\phi_5}$ is distinct from $\Pi_{\phi_3\phi_4\phi_1\phi_2\phi_5}$. Both are composed of the same premises, but in a different order (one is a permutation of the other).

[6] See Section C.1 in Appendix C.

Chapter 9

Roles and Properties of our Approach

So far we have presented the formal basis of our approach to argumentation, showing that it is practicable to model and to automate argument dynamics by specifying a catalogue of schemata for generating attacks. We have also presented a classification that allows different types of attacks to be explored in a systematic way, and which together with the possibility for automatic testing and search, allows us to understand more about dynamics in argument.

This chapter now considers some of the roles and properties of this formalisation, and possible uses of our classification both in analysing generated arguments as well as in generating new ones. The discussion in the next sections is guided by the following questions:

- what sorts of properties can we give to our formalisation?
- to what extent can examples from existing frameworks can be captured?
- how well can existing approaches deal with the types of dynamic argument explored here?
- what are the benefits and limitations of our approach?

9.1 Non-monotonic Aspects of Dynamic Argumentation

As discussed in Chapter 3, research in argumentation in the context of non-monotonic reasoning is about characterising precisely the class of acceptable arguments from a *fixed* knowledge base, so that "the role of argumentation is to justify the use of certain defeasible rules deriving a conclusion in preference to the use of other defeasible rules deriving conflicting conclusions" (Kowalski and Toni 1996). Section 9.1.1 investigates how our model relates to these argumentation frameworks if we fix the set of possible attack-based revisions to trivial revisions only, and whether representative examples can then be captured.

Other types of revision, however, specify from a procedural perspective how to challenge information and introduce new arguments. This brings other non-monotonic issues into play that are related to the actual transformation of theories via attacks. We discuss these in Section 9.1.2.

9.1.1 Determining Acceptability in Fixed Theories

Work in argument-based semantics concentrates primarily on defining in a declarative way (for instance by a fixpoint operator, or in terms of multiple extensions) when arguments and sentences are justified given certain relations of conflict and defeat. Sometimes proof theories are also developed, which are concerned with establishing—often in a dialectical style—the status of individual arguments according to the underlying status characterisation.

Recall from Chapter 3 that there are in general three classes of arguments, namely *justified*, *defensible* and *overruled*. Exactly how these are defined varies between the different types of argument-based semantics proposed so far, but the general intuition is often the same: justified arguments are those acceptable from a sceptical perspective, whereas defensible arguments are those acceptable for a credulous reasoner; overruled arguments are defeated by a justified one, and hence not acceptable.

The model proposed here, however, is more a constructive theory of how argument processes are generated than a way of characterising sets of acceptable arguments according to their relation to all other arguments. Nonetheless the two approaches are expected to be related, mainly for the following reasons. First, there is an element of acceptability also in our formalisation—according to Definition 4.7 a sentence is acceptable if it is possible to generate a dynamic argument about it in which all attacks generated from a catalogue Φ of argument schemata are appropriately dismissed. Second, if Φ is fixed to contain only trivial revisions then it can generate and explore every possible attack from a fixed knowledge base by means of the underlying provability relation. Thereby dynamic argumentation can be seen as a proof-theoretical mechanism for determining whether an argument is *defensible*; i.e. acceptable from a credulous perspective.

To make comparison easier, our model could be described in terms of the architecture for argumentation frameworks proposed by Dung (1995) and discussed in Chapter 3.

The Argument Generation Unit (AGU) generates arguments and specifies the attack relationships between them.

Here the **AGU** is composed of the underlying provability relation $\vdash$, and the library of possible attack-based revisions Φ restricted to trivial operations:[1]

$$\Phi = \{\mathit{trivial}(X : \mathbf{in} \rightsquigarrow \overline{X} : \mathbf{in}),\ \mathit{trivial}(X : \mathbf{out} \rightsquigarrow X : \mathbf{in})\}.$$

Note that by definition, if A and A' are both arguments in a theory Π, and A' attacks A, then A' can be generated via a trivial revision.

The Argument Processing Unit (APU) corresponds to the proof theory for determining whether a sentence or an argument is acceptable.

Here the **APU** corresponds to the dynamic argumentation mechanism which instantiates possible schemata in Φ and verifies the corresponding properties. A sentence φ is said to be acceptable if $\langle A_0, \phi_1, A_1, ..., \phi_N, A_N\rangle$ is a dynamic argument with respect to the (fixed) underlying theory Π such that all attacks to φ have been dismissed (i.e. $\varphi : \mathbf{in} \in \mathcal{D}_N$) .

The following example illustrates this notion.

[1] Section C.1 gives the general description of trivial operations, represented here by expressions parameterised by the relevant type of attack.

Example 9.1 *Let Π be the following theory in a Horn clause resolution-based system:*[2]

$$\begin{array}{rcl} \textit{pacifist}(X) & \leftarrow & \textit{quaker}(X) \\ \textit{no\_pacifist}(X) & \leftarrow & \textit{republican}(X) \\ \textit{quaker}(\textit{nixon}) & \leftarrow & \textit{true} \\ \textit{republican}(\textit{nixon}) & \leftarrow & \textit{true} \end{array}$$

such that pacifist(X) and no_pacifist(X) are conflicting sentences in the language. Also, let A_p and A_{np} be the arguments supporting pacifist(nixon) and no_pacifist(nixon), respectively.

In this case, the query:

```
| ?- findall(D, dynamic_arg(pacifist(nixon), TInit, T, D), AllD).
```

gives only one possible course of argument (with T = TInit):

$$\left\langle \begin{array}{ll} A_p, & \textit{trivial}(\textit{pacifist}(\textit{nixon}) : \mathbf{in} \rightsquigarrow \textit{no\_pacifist}(\textit{nixon}) : \mathbf{in}), \quad A_{np}, \\ & \textit{trivial}(\textit{no\_pacifist}(\textit{nixon}) : \mathbf{in} \rightsquigarrow \textit{pacifist}(\textit{nixon}) : \mathbf{in}), \quad A_p \end{array} \right\rangle.$$

Note that A_p was allowed to be advanced again as it had not yet been used to attack A_{np}. This captures the behaviour characteristic of a credulous reasoner: if arguments A and B attack each other with equal strength, and B is used to attack A during argumentation, then A can be used to attack, and consequently dismiss, B.[3]

Analogously, the query:

```
| ?- findall(D, dynamic_arg(no_pacifist(nixon), TInit, T, D), AllD).
```

gives also one possible argument, in which A_{np} is also established as defensible. □

Very often such **APU**s are defined in a dialectical style, as argument games between a proponent and an opponent:

[2] This example is drawn from (Prakken and Vreeswijk 1999), a comprehensive study on the relation between non-monotonic reasoning and argumentation. Their general discussion, though, abstracts from the internal structure of arguments, assuming both arguments and attacks to be primitive concepts. Thus in order to experiment with their examples we have reconstructed them in a logic programming, resolution-based style.

[3] Appendix D gives the restrictions for advancing an attack by means of the property *supports*.

> The proponent starts with an argument to be tested, and each of the following move consists of an argument that attacks the last move of the other party with a certain minimum force. The initial argument provably has a certain status if the proponent has a winning strategy, i.e., if he can make the opponent run out of moves whatever moves the opponent makes. The exact rules of the game depend on the semantics it is meant to capture. (Prakken and Vreeswijk 1999, p. 82)

In fact, Jakobovits (2000) has identified some of the issues that give the fine tuning for the game rules so that it captures the intended semantics. These include:[4]

- Can players repeat arguments?
- Must the player react immediately?
- May players contradict themselves?
- Can players use arguments which have already been attacked by the opponent?
- Can a player use arguments which have already been used by the opponent?

The following for instance is a specification of a proof-theoretical dispute that captures sceptical reasoning, in which only justified arguments (rather than defensible ones) are considered to be acceptable:[5]

Definition 9.1 (Proof-theoretical Dispute) *A (proof-theoretical) dispute on an argument A is a non-empty sequence of moves of the form $move_i = (Player_i, A_i)$ with $A_0 = A$ such that:*

- *$Player_i =$ PROPONENT if and only if i is even; otherwise $Player_i =$ OPPONENT.*
- *If $Player_i = Player_j =$ PROPONENT, $i \neq j$, then $A_i \neq A_j$.*
- *If $Player_i =$ PROPONENT, $i \geq 0$, then A_i strictly attacks A_{i-1}. (That is, A_i attacks A_{i-1} but A_{i-1} does not attack A_i.)*
- *If $Player_i =$ OPPONENT, then A_i attacks A_{i-1}.*

[4] Later in Chapter 12 we consider some issues on how this view relates to the sorts of protocols, languages and game theory in multi-agent negotiation.

[5] Adapted from (Prakken and Vreeswijk 1999, p. 82).

The different burdens of proof for the **PROPONENT** *and the* **OPPONENT** *guarantee that if the* **PROPONENT** *wins the dispute, then A is justified.* □

Clearly dynamic arguments can also be seen as an argument game—both represent processes of argument exchange, the main difference being that attacks in our formalism are generated from a library of argument schemata. Remember that in a dynamic argument each step is intended to alternately change the acceptability status of the sentence under dispute, either from **out** to **in** or from **in** to **out**; in this way, the first can be seen as moves advanced by a **PROPONENT**, and the latter by an **OPPONENT**. The **PROPONENT** is also the first player to move by advancing a justification for the sentence. Finally, two catalogues of argument schemata could be considered, one to be used by the **PROPONENT** and another by the **OPPONENT**, but for the type of credulous reasoning illustrated in Example 9.1 these can be assumed to be identical.

Because our mechanism is essentially credulous, in order to capture the sort of sceptical reasoning in Definition 9.1 we need to account for some of the conditions that determine the exact rules of that game. It turns out that the sorts of of features addressed in (Jakobovits 2000) can be easily incorporated into our original mechanism by means of the properties that are tested in connection with each attack-based schema. For instance, a sentence is said to be acceptable from a sceptical perspective (justified) if we can generate a dynamic argument $\langle A_0, \phi_1, A_1, ..., \phi_N, A_N\rangle$ such that:

- $\phi_i \in \Phi_{\textbf{PROPONENT}}$ if i is even, where $\Phi_{\textbf{PROPONENT}}$ is obtained from Φ above as follows: first, refine the property *supports* so that it disallows any repetition of arguments whatsoever; then, introduce the extra property $\neg attacks(A', A)$ to each schema in $\Phi_{\textbf{PROPONENT}}$, thus forcing the attack to be strict;
- $\phi_i \in \Phi_{\textbf{OPPONENT}}$ if i is odd, where $\Phi_{\textbf{OPPONENT}}$ is equivalent to Φ above.

Example 9.2 *Consider again the theory in Example 9.1, together with the mechanism above for generating sceptical dynamic arguments. Now the query:*

```
| ?- findall(D, dynamic_arg(pacifist(nixon), TInit, T, D), AllD).
```

gives only the argumentation below, and therefore the argument for pacifist(nixon) is not sceptically acceptable.

$$\langle \ A_p, \quad trivial(pacifist(nixon) : \mathbf{in} \rightsquigarrow no\_pacifist(nixon) : \mathbf{in}), \quad A_{np} \ \rangle.$$

□

This connection with proof theories is not surprising, as our formalisation takes an essentially procedural view on argumentation. Existing proof-theoretical models of argumentation can be expressed as dynamic argumentation mechanisms by restricting the types of revision to trivial ones and by adapting the corresponding properties so that it gives the same behaviour. Nevertheless "it turns out that all semantics have some problems", and yet much work remains to be done in providing correctness and completeness results for the proof-theories proposed so far (Prakken and Vreeswijk 1999).

Some other issues have been raised by Jakobovits (2000) which are less concerned with argument-based semantics than with real disputes, and therefore closer to our interests. These are:

- How should attack between arguments be defined?
- Is the set of possible arguments known before the dialogue takes place, or is it generated dynamically?

These questions have been addressed extensively throughout this thesis, but we now focus a bit more on the latter, especially on the consequences of actually changing the set of possible arguments dynamically.

9.1.2 Non-monotonicity in Argument-based Theory Revision

According to Prakken (2000, p. 2) the difference between proof-theoretical disputes in the context of argument-based semantics and *real* disputes is that:

> [...] while in proof-theoretical disputes all arguments are constructed from a *given* body of information, in disputes between real agents this body of information is

usually constructed dynamically, during the dispute, since the participants can at any time supply new or withdraw old information.

Non-monotonicity in this case is not only about some arguments being preferred over others, but rather about the actual addition and retraction of information. Assuming that the underlying provability relation is monotonic,[6] this section looks at how certain types of attack can affect the interpretation set of the corresponding theory.

The reason why such a characterisation is important is because argument dynamics can also be viewed as a process of theory manipulation intended to generate more acceptable theories. Central to this view is the notion of interpretation.[7] When designing argument systems (and libraries of revision schemata), or analysing an argument produced by such systems, it should be possible to describe how attack-based transformations affect the corresponding interpretation set.

One way of expressing such relations is by considering the characteristics of certain types of schema in order to make predictions about the behaviour of the interpretation set. This gives a high level description of key relations between transformation steps without saying exactly how the arguments are going to be (or were) derived. A neat correspondence would for instance say that adding an argument causes the interpretation set to expand, while removing an argument constrains it. Unfortunately this is not always the case, as adding an argument sometimes means blocking others, and vice-versa.

The question now is whether the attack relation $\kappa(\Pi, \Pi')$ can be expressed via set inequality relations between interpretation sets $\iota(\Pi)$ and $\iota(\Pi')$ (assuming the underlying logic to be monotonic). In what follows the classification in Figure 7.2 is used for guiding this analysis by considering the possibilities for an unconstrained attack-based revision

[6] Although the extended resolution method for treating negation as failure in general logic programs is clearly non-monotonic, it is possible to consider these from an abductive perspective that consists in adding non-provability assumptions as facts to the theory and treating these monotonically. For more details, refer to section on the Abstract Argumentation Framework in Chapter 3, and later in Section 9.5.

[7] At this point the relation with the fields of transformation (Pettorossi and Proietti 1998) and synthesis (Deville and Lau 1994) of logic programs becomes more apparent. Transformation of logic programs is concerned with preserving the semantic value of a specification as we derive correct and efficient programs from it, so at each transformation step the interpretation set must remain the same. On the other hand, some recent approaches to structural synthesis have considered inequalities between sets of consequences as the basis for refinement of specifications (Robertson 1999b).

$\Pi \overset{\phi}{\rightsquigarrow} \Pi'$. Between parentheses we refer to the sections in Appendix C that define the corresponding schemata.

Adding an Argument

Trivial Revision (Section C.1)

$$\Pi \overset{trivial}{\rightsquigarrow} \Pi' \;\rightarrow\; \iota(\Pi) = \iota(\Pi')$$

By definition, trivial revisions involve no changes to the theory, so $\Pi = \Pi'$.

Elementary Revisions (Section C.2)

$$\Pi \overset{elementary}{\rightsquigarrow} \Pi' \;\rightarrow\; \iota(\Pi) \subseteq \iota(\Pi')$$

This follows from the monotonicity of the system and the fact that elementary revisions for *adding an argument* consist in adding axioms only—i.e. $\Pi \subseteq \Pi'$. In fact:

Adding a Fact (Section C.2.1)

$$\Pi \overset{fact}{\rightsquigarrow} \Pi' \;\rightarrow\; \Pi \subseteq \Pi'$$

Adding a Substantiated Rule (Section C.2.2)

$$\Pi \overset{substantiated\_rule}{\rightsquigarrow} \Pi' \;\rightarrow\; \Pi \subseteq \Pi'$$

Adding a Burden Shift Rule (Section C.2.3)

$$\Pi \overset{burden\_shift\_rule}{\rightsquigarrow} \Pi' \;\rightarrow\; \Pi \subseteq \Pi'$$

Remember that negation as failure can be represented by extra non-provability assumptions in the language.

Updating Revisions (Section C.3)

$$\Pi \overset{updating}{\rightsquigarrow} \Pi' \;\not\rightarrow\; \iota(\Pi) \subseteq \iota(\Pi')$$

In this case it is harder to predict how the interpretation set behaves in general, because $\Pi \not\subseteq \Pi'$. However, looking at the properties associated to each logic-specific schema in this category can provide more information about the changes.

Removing Irrelevance in a Rule (Section C.3.1)

$$\Pi \overset{irrelevance}{\rightsquigarrow} \Pi' \;\rightarrow\; \iota(\Pi) \subseteq \iota(\Pi')$$

This follows from the properties associated to the irrelevance schema: the axiom from Π that is updated in Π' is such that all conclusions derived from it are still derived, and others are now allowed, namely those dependent on the satisfiability of the removed literal.

Generalising a Rule (Section C.3.2)

$$\Pi \overset{generalisation}{\rightsquigarrow} \Pi' \;\rightarrow\; \iota(\Pi) \subseteq \iota(\Pi')$$

Again this follows from the properties associated to the schema: the axiom in Π' that is updated from Π is obtained via the application of an inverse substitution (from terms to variables), so everything that was derived before can still be inferred.

Revising the Consequent of a Rule (Section C.3.3)

$$\Pi \overset{misconclusion}{\rightsquigarrow} \Pi' \;\not\rightarrow\; \iota(\Pi) \subseteq \iota(\Pi')$$

Revising the consequent of a rule may introduce new elements into $\iota(\Pi')$ but may also block others from being derived. In this case, no generic relation between the interpretations sets can be identified.

Reversing a Rule (Section C.3.4)

$$\Pi \overset{reversion}{\rightsquigarrow} \Pi' \;\not\rightarrow\; \iota(\Pi) \subseteq \iota(\Pi')$$

As above, no set inequality relation between the two interpretation sets can be said to hold in the general case.

Removing an Argument

Elementary Revisions (Section C.4)

$$\Pi \overset{elementary}{\rightsquigarrow} \Pi' \;\rightarrow\; \iota(\Pi) \supseteq \iota(\Pi')$$

This follows from the monotonicity of the system and the fact that elementary revisions for *removing an argument* consist in retracting axioms only—i.e. $\Pi \supseteq \Pi'$. In fact:

Retracting an Invalid Rule (Section C.4.1)

$$\Pi \overset{invalid\_rule}{\rightsquigarrow} \Pi' \;\rightarrow\; \Pi \supseteq \Pi'$$

Retracting a Weak Rule (Section C.4.2)

$$\Pi \overset{weak\_rule}{\rightsquigarrow} \Pi' \;\rightarrow\; \Pi \supseteq \Pi'$$

Retracting a Misrelation (Section C.4.3)

$$\Pi \overset{misrelation}{\rightsquigarrow} \Pi' \;\rightarrow\; \Pi \supseteq \Pi'$$

Updating Revisions (Section C.5)

$$\Pi \overset{updating}{\rightsquigarrow} \Pi' \;\not\rightarrow\; \iota(\Pi) \supseteq \iota(\Pi')$$

Again it is harder to predict how the interpretation set behaves in general, because $\Pi \not\supseteq \Pi'$. Analogously as in the updating cases above, the properties associated to each logic-specific schema in this category can give more information about the changes.

Elaborating Preconditions in a Rule (Section C.5.1)

$$\Pi \overset{elaboration}{\rightsquigarrow} \Pi' \;\rightarrow\; \iota(\Pi) \supseteq \iota(\Pi')$$

The properties associated to the elaboration schema guarantee that some conclusions that were allowed in Π will be blocked in Π', namely those dependent on the satisfiability of the new literal which is required to be unsatisfiable in the theory.

Specialising a Rule (Section C.5.2)

$$\Pi \overset{specialisation}{\rightsquigarrow} \Pi' \;\rightarrow\; \iota(\Pi) \supseteq \iota(\Pi')$$

Again this follows from the properties associated to the specialisation schema: the axiom in Π' that is updated from Π is obtained via the application of a substitution (from variables to terms), thus some of its original conclusions may no longer be inferred.

Revising the Consequent of a Rule (Section C.5.3)

$$\Pi \overset{misconclusion}{\rightsquigarrow} \Pi' \;\not\rightarrow\; \iota(\Pi) \supseteq \iota(\Pi')$$

Revising the consequent of a rule may block some elements from $\iota(\Pi)$ but may also introduce new ones, so no generic relation between the interpretations sets can be identified.

Reversing a Rule (Section C.5.4)

$$\Pi \overset{reversion}{\rightsquigarrow} \Pi' \;\not\rightarrow\; \iota(\Pi) \supseteq \iota(\Pi')$$

For the same reasons, no relations between the two interpretation sets can be guaranteed to hold in the general case.

Designers of argument systems may choose types of schema that conform to certain characteristics so as to predict an overall behaviour of the transformation process. For instance if every schema ϕ in a catalogue Φ is such that $\iota(\Pi) \subseteq \iota(\Pi_\phi)$, then the interpretation set of a theory is guaranteed to either expand or at least remain unchanged throughout any dynamic argument. On the other hand, if Φ also contains certain operations such that $\iota(\Pi) \supseteq \iota(\Pi_\phi)$, then nothing can be said about the global development of the argument, as transitivity cannot be applied in this case.

Describing possible revisions in terms of interpretation sets can provide yet more information for influencing and guiding the design of domain specific schemata from the classification in Figure 7.2. A question arises at this point, and is considered later in Section 9.3, of whether this classification is complete is some sense. Also related to this, Section 9.4 discusses the role of this classification in retrospective analysis and explanation of arguments.

9.2 Termination

At this stage termination can be reduced to the existence of finite relevant arguments in the theory. A revision can only be applied once to generate the same attack, and assuming that the number of possible revisions in the catalogue is finite, the question is whether an infinite number of attacks satisfying the requirements of a certain schema can be generated. Problems can arise for infinite chains of argumentation, but these are a problem for dialectical models of argumentation frameworks as well, as they may be captured by fixpoint approaches but not by exhaustively considering every argument in the theory (Prakken and Vreeswijk 1999).

9.3 Is Our Classification Complete?

Chapter 7 proposed a way to parameterise the attack generation step by constraining the types of revision operation in a cascade of levels which eventually gets to be domain specific. We do not claim that this is the only way to characterise the possible revisions to argument, nor that the entire collection in Figure 7.2 is complete. We do argue, though, that this classification is complete *up to a level*, namely the level of instantiation described in Section 7.3.5.

In fact, the level of instantiation described in Section 7.3.4 and depicted in Figure 9.1 below is complete by definition, because it is based on the characterisation of attacks given in Chapter 6. Any attack consists in either introducing a (not necessarily new) argument, or retracting an existing one.

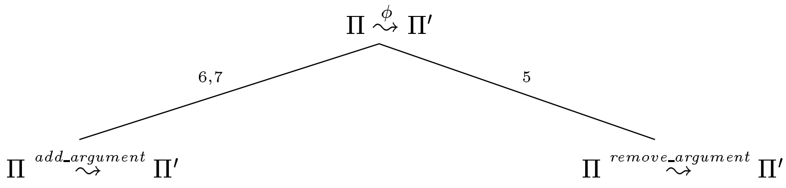


Figure 9.1: General types of revision.

Section 7.3.5 gives the general structure of the revisions for adding and removing an argument, as depicted in Figure 9.2. Because only fundamental types of revision are considered, completeness can only be established with respect to these.

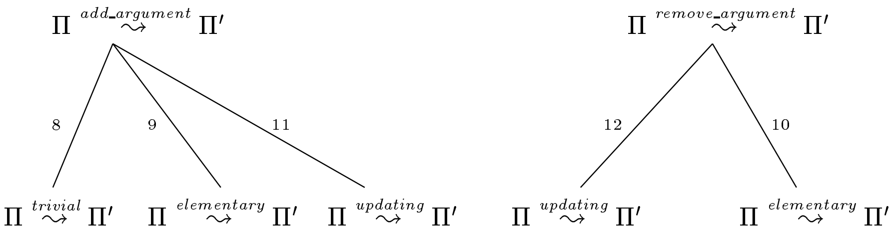


Figure 9.2: From dealing with arguments to dealing with premises.

Arguing that this level is complete is equivalent to saying that the following are the only possible combinations for adding or removing an argument:

- adding an argument via a trivial revision;
- adding an argument via an elementary revision (adding a new premise);
- adding an argument via an updating revision (updating an existing premise);
- removing an argument via an elementary revision (removing a premise);
- removing an argument via an updating revision (updating an existing premise);

which in its turn is equivalent to saying the remaining combinations below cannot be used to describe an attack:

- removing an argument via a trivial revision;
- adding an argument via an elementary revision (removing a premise);
- removing an argument via an elementary revision (adding a new premise).

Let us then consider each possibility as follows:

Removing an argument via a trivial revision.

This is clearly not possible, as removing an argument means refuting it, and that must involve some revision to the theory.

Removing an argument via an elementary revision (adding a premise).

The most straightforward example in this case involves the extended resolution mechanism for negation as failure, in which adding a premise can block conclusions based on certain non-provability assumptions.

For instance, the fact $q(a) \leftarrow \mathit{true}$ can be added to the theory below is order to refute the argument for $p(a)$ that is based on the non-provability of $q(a)$.

$$\begin{array}{rcl} p(X) & \leftarrow & \mathit{not}\ q(X) \\ q(b) & \leftarrow & \mathit{true} \end{array}$$

Instead, this could be interpreted as *adding an argument* for $q(a)$ rather than as *removing the argument* for $p(a)$, and hence could be obtained from rewrite 9.

Adding an argument via an elementary revision (removing a premise).

Analogously, this case can also be reduced to that of removing an argument by removing a premise, and thus obtained from rewrite 10.

Hence this level of instantiation is complete for fundamental types of operations. From the level of logic-specific schemata[8] downwards completeness results can no longer be guaranteed, because it is always possible to give more or less detailed descriptions of the structural revisions that are allowed.

For instance elementary revisions for adding a premise are quite unspecific, the main restriction concerning the head of the clause to be added which has to unify with the

[8] See Section 7.3.6.

sentence in question. Some updating revisions on the other hand are more specific as they depend on an existing axiom and on a well-defined way of transforming this axiom (e.g. specialisation or generalisation). Also, our experience in modelling arguments has shown that the types of schema for *reversing a rule* or *revising its consequent* are not as frequent as other updating schemata. We have even considered not including these in the collection given in Chapter 7, but finally decided on keeping them to illustrate the possibility of introducing and preserving diverse forms of revision.

9.4 Communicating Dynamic Arguments

This hierarchical classification not only promotes a methodical design of argument schemata in which domain-specific instances may be gradually devised, but also supports analysis, explanation and presentation of produced arguments. Given that predefined schemata may be recognised by the corresponding path in the hierarchy, two sorts of information may be combined in communicating each step in the argument: the various levels of instantiation for the revision operator, and the possible relations between interpretation sets.

We now revisit parts of the example in Chapter 8 to exemplify alternative modes of argument communication. In particular, we consider the dynamic argument below:

$$\langle A_0, \phi_1, A_1, \phi_2, A_2, \phi_3, A_3, \phi_4, A_4, \phi_5, A_5 \rangle.$$

A plain form of presentation which consists in laying out the whole argument in all its details, with argument trees and instantiated argument schemata, may be denoted as follows:

$$A_0 \overset{\phi_1}{\leadsto} A_1 \overset{\phi_2}{\leadsto} A_2 \overset{\phi_3}{\leadsto} A_3 \overset{\phi_4}{\leadsto} A_4 \overset{\phi_5}{\leadsto} A_5.$$

Sometimes, though, a higher level presentation may be more appropriate, and the following sections illustrate how the information associated to each schema may be employed for that purpose.

9.4.1 Different Levels of Instantiation

This section explores alternative presentations of an argument based on the various levels of instantiation given by the hierarchy in Figure 7.2. Also, assume that at every level the description is parameterised by the type of attack it promotes, so it is possible to say which claim is supported at each step.

Expressing General Types of Revision

At this level of description—given in Section 7.3.4—the argument process, which starts with a justification for $required\_level(aflatoxin, 20)$, unfolds as follows:

$$A_0 \overset{add\_argument}{\rightsquigarrow} A_1 \overset{add\_argument}{\rightsquigarrow} A_2 \overset{remove\_argument}{\rightsquigarrow} A_3 \overset{add\_argument}{\rightsquigarrow} A_4 \overset{remove_a rgument}{\rightsquigarrow} A_5$$

The first move
: consists in *adding an argument* for $safe\_level(aflatoxin, s)$.

The second move
: consists in *adding an argument* for $required\_level(aflatoxin, 20)$.

The third move
: consists in *removing the argument* for $required\_level(aflatoxin, 20)$.

The fourth move
: consists in *adding an argument* for $causes(aflatoxin, cancer, human)$.

The final move
: consists in *removing the argument* for $causes(aflatoxin, cancer, human)$.

Expressing Fundamental Types of Revision

At this level of description—given in Section 7.3.5—the argument process, which starts with a justification for $required\_level(aflatoxin, 20)$, unfolds as follows:

$$A_0 \overset{elementary}{\rightsquigarrow} A_1 \overset{updating}{\rightsquigarrow} A_2 \overset{updating}{\rightsquigarrow} A_3 \overset{elementary}{\rightsquigarrow} A_4 \overset{updating}{\rightsquigarrow} A_5$$

The first move
: performs an *elementary revision* for justifying $safe\_level(aflatoxin, s)$.

The second move
: performs an *updating revision* for justifying $required\_level(aflatoxin, 20)$.

The third move

performs an *updating revision* for refuting $required\_level(aflatoxin, 20)$.

The fourth move

performs an *elementary revision* for justifying $causes(aflatoxin, cancer, human)$.

The final move

performs an *updating revision* for refuting $causes(aflatoxin, cancer, human)$.

Expressing Logic-Specific Types of Revisions

At this level of description—given in Section 7.3.6—the argument process, which starts with a justification for $required\_level(aflatoxin, 20)$, unfolds as follows:

$$A_0 \overset{fact}{\rightsquigarrow} A_1 \overset{irrelevance}{\rightsquigarrow} A_2 \overset{specialisation}{\rightsquigarrow} A_3 \overset{substantiated\_rule}{\rightsquigarrow} A_4 \overset{elaboration}{\rightsquigarrow} A_5$$

The first move

consists in *adding a fact* to justify $safe\_level(aflatoxin, s)$.

The second move

consists in *removing irrelevance in a rule* to justify $required\_level(aflatoxin, 20)$.

The third move

consists in *specialising a rule* to refute $required\_level(aflatoxin, 20)$

The fourth move

consists in *adding a substantiated rule* to justify $causes(aflatoxin, cancer, human)$.

The final move

consists in *elaborating a rule* to refute $causes(aflatoxin, cancer, human)$.

Finally each step in the argument may be communicated in its integral form as originally illustrated in Chapter 8.

9.4.2 Relations between Theories

Dynamic arguments may also be presented at a yet higher level of description for expressing set relations between theories, without knowing directly how each consecutive theory interacts nor how and what arguments triggered the transformation. According to the properties discussed in Section 9.1.2,[9] the process of theory transformation based on the aflatoxin example gives the following relations between each transformation step:

$$\iota(\Pi) \subseteq \iota(\Pi_{\phi_1}) \subseteq \iota(\Pi_{\phi_1\phi_2}) \supseteq \iota(\Pi_{\phi_1\phi_2\phi_3}) \subseteq \iota(\Pi_{\phi_1\phi_2\phi_3\phi_4}) \supseteq \iota(\Pi_{\phi_1\phi_2\phi_3\phi_4\phi_5})$$

Notice that nothing can be guaranteed about the relation between the initial and final theory in this case, because different kinds of transformation (expanding and constraining) have been involved.

9.5 The Abstract Argumentation Framework: Limitations

While Section 9.1.1 elaborated on how dynamic argumentation relates to formalisms for defeasible argumentation, this section looks at types of arguments that cannot be entirely captured by these. In particular, it takes the Abstract Argumentation Framework as a representative formalism and applies it to the example of argument from the safety engineering domain in Chapter 2, identifying questions which the existing framework do not answer but which are needed to represent a larger class of dynamic arguments. This brings in some of the issues to be addressed in the next part of this thesis in connection to the automation of such examples.

The reason why the Abstract Argumentation Framework is used here is because it is flexible and generic, subsuming other approaches to defeasible argumentation (see Chapter 3). Also, it incorporates some elements of revision, such as treating assumptions as extra facts in the theory (and which can be attacked by proving their contrary), and extending axioms to include other non-provability assumptions. But although in (Kowalski and Toni 1996) it is claimed that the Abstract Argumentation Framework "seems to correspond well with informal argumentation", there are some informal and useful arguments that cannot be represented within it.

[9] Assume that this example is now modelled as a definite logic program where negated atoms are treated as positive assumptions extending the language, and which are considered to be true but can be attacked by their *contrary* as defined by an asymmetric conflict relation. This guarantees the monotonicity of the underlying language, and hence the use of properties discussed in Section 9.1.2. Notice that this only affects revision ϕ_1, which now defines an attack of the form $\mathit{cannot\_be\_shown}(\mathit{safe\_level}(\mathit{aflatoxin}, L)) : \mathbf{in} \rightsquigarrow \mathit{safe\_level}(\mathit{aflatoxin}, L) : \mathbf{in}$ rather than the original $\mathit{safe\_level}(\mathit{aflatoxin}, L) : \mathbf{out} \rightsquigarrow \mathit{safe\_level}(\mathit{aflatoxin}, L) : \mathbf{in}$.

One such example is the safety argument about the pressure tank control system in Section 2.1.1. Argumentation was used as a way of revising the system model in order to increase its acceptability with respect to known faults, based on a technique known as *fault tree analysis*. Fault tree analysis is essentially a method that "starts with an event directly related to an identified hazard, the 'top event', and works backwards to determine its cause" (Storey 1996, p.43). Taking the top event to be the rupture of the tank, the safety argument in Section 2.1.1 followed by considering the possible causes (or *minimal cut sets* of events) leading to this event and adapting the model where necessary so as to make the system acceptably tolerant to these. For instance, one possible cause leading to the rupture of the tank is the primary failure of the relay *k2* (see Figure 2.2), and the corresponding course of dynamic argument could be represented as follows:

$$
\begin{array}{lll}
\langle & A_0: & \textit{The system is operational at all times} \\[6pt]
 & \phi_1: & \textsf{introduce primary failure of } \textit{k2} \\[6pt]
 & A_1 & \textit{If relay k2 fails to open when it should, the} \\
 & & \textit{system is no longer operational at all times} \\[6pt]
 & \phi_2: & \textsf{add a redundant relay to the model} \\[6pt]
 & A_2 & \textit{If we add an extra relay in parallel,} \\
 & & \textit{then the system is still operational} \qquad\qquad \rangle
\end{array}
$$

To represent this argument in terms of an Abstract Argumentation Framework we first select a supporting deductive system, for instance the Horn clause resolution-based system of (Kowalski and Toni 1996). Let $\mathcal{A}$ be the set of assumptions of the form $cannot\_be\_shown(\varphi)$, where φ is a sentence in the underlying language. Assuming that we know how to extend the rules appropriately, the following is a (simplified) model Π for the pressure tank system in Figure 2.2.

$$
\begin{array}{rcl}
operational\_tank(T) & \leftarrow & on\_motor(T) \ \wedge \ not\_full(T) \\
operational\_tank(T) & \leftarrow & off\_motor(T) \ \wedge \ pressurised(T) \\
not\_operational\_tank(T) & \leftarrow & on\_motor(T) \ \wedge \ pressurised(T) \\
on\_motor(T) & \leftarrow & closed(relay(k2), T) \\
off\_motor(T) & \leftarrow & open(relay(k2), T) \\
closed(relay(K), T) & \leftarrow & energised(relay(K), T) \ \wedge \\
 & & cannot\_be\_shown(open(relay(K), T) \\
open(relay(K), T) & \leftarrow & deenergised(relay(K), T) \ \wedge \\
 & & cannot\_be\_shown(closed(relay(K), T)
\end{array}
$$

Briefly, the tank is operational at a time point T if the motor is pumping water into it when it is not full, or when the tank is pressurised but the motor is off. Otherwise, the tank is not operational if the motor is still on when the tank is pressurised. Notice that $operational\_tank(T)$ and $not\_operational\_tank(T)$ are conflicting sentences in the language.

In particular, assume that at a given time t the relay $k2$ is de-energised and the tank is pressurised:

$$\begin{array}{rcl} pressurised(t) & \leftarrow & true \\ deenergised(relay(k2), t) & \leftarrow & true \end{array}$$

Also, as described in Section 2.1 it is possible for the contacts of relay $k2$ to fail to open when the coil is de-energised, causing the rupture of the tank. This fault may be represented by following axiom:

$$closed(relay(K), T) \quad \leftarrow \quad deenergised(relay(k2), T)$$

So let Π denote the set of clauses above, and let:

$$\Delta = \{cannot\_be\_shown(closed(relay(k2), t)\}$$

An argument supporting that the tank is operational at time t can be obtained if the assumption Δ is added to Π:

$$\Pi \cup \Delta \vdash operational\_tank(t). \tag{9.1}$$

Besides, the following argument for $not\_operational\_tank(t)$ can also be derived:

$$\Pi \vdash not\_operational\_tank(t). \tag{9.2}$$

Because the underlying system is monotonic, the addition of clauses only allows more possible conclusions to be derived. In this way 9.1 and 9.2 are two conflicting arguments, but 9.1 cannot defend itself against 9.2. On the basis of acceptable arguments (or admissible assumptions), this framework discriminates between faulty and non-faulty behaviours and allows only the inference of $not\_operational\_tank(t)$.

This is an important point because it shows that the Abstract Argumentation Framework can formalise part of the safety argument about the pressure tank system. In this particular case where a fault is present, it does not allow a safe conclusion to be erroneously derived.

Yet this is a matter of safety, so it is essential that we can adapt the model to exhibit only safe behaviours. The revisions allowed in this framework, though, are only about adding extra non-provability assumptions. This makes it possible to defend *operational_tank* from attacks by first *assuming* that things cannot go wrong, and then prioritising these arguments over the arguments for *not_operational_tank*. But these solutions do not represent any enhancement of system safety because they do not change the structure of the system. Arguments such as *"if an extra relay is added in parallel this attack will no longer be relevant enough to be a concern"*—which are common when modelling systems in safety-critical domains—cannot be expressed by means of extra non-provability assumptions.[10]

This is another important point: the Abstract Argumentation Framework does say a lot about what an argument is within a collection of logic programming clauses, but it does not prescribe strategies for revising these clauses. Yet this is the most essential task in dynamic argumentation. Although the Abstract Argumentation Framework is effective in expressing the defeasibility in argument, it does not account for many of the features which are responsible for argument dynamics, as such as how attacks can map onto changes to the argument. In terms of argument schemata, for instance, one could define a revision schema that adds redundancy to the system by elaborating on the axioms that depend on the behaviour of *k2*.

In summary, we took a novel view of specifications as arguments (e.g. the description of the pressure tank model as an argument) and observed that in safety domains fault trees are used to criticise specifications. The Abstract Argumentation Framework alone does not enable us to replicate automatically the reasoning that is done based on fault tree analysis. What we found though is that it is in fact representationally adequate but not enough distinctions were made to actually represent that reasoning.

In the next part we give a more detailed architecture for adversarial argument that allows for external sources of information (such as the fault tree model) to generate instances

[10] It could be argued that one could just augment the set of clauses in some way, but the Abstract Argumentation Framework itself does not account for any methodology supporting such a task.

of attacks from a catalogue of possible revision schemata. This architecture includes the mechanisms necessary to give dynamics to arguments, accounting for important aspects such as:

- where attacks come from;
- how these lead to counter-attacks;
- how the source of these attacks maps onto changes to the argument; and
- how to determine the comparative strength of criticism.

Part III

Instantiating Applications

Chapter 10

A General Architecture for Dynamic Argumentation Systems

Having explored the mechanisms for argument dynamics in Part II, the aim of this part is to define the different notions involved in dynamic argumentation separately, thus providing a clearer picture of how arguments can be generated and evaluated, and also allowing for a larger class of arguments to be formally represented by considering possible external sources of criticism and attack.

The architecture presented here was first proposed in (Carbogim et al. 1999) as an extension of the Abstract Argumentation Framework in (Kowalski and Toni 1996; Bondarenko et al. 1997) and of the Argumentation Framework in (Dung 1995), in the sense that these formalisms were used as a starting point in developing a more detailed framework for implementing adversarial argument. This chapter gives a general definition of the three types of components forming the basis of this proposal, namely a *theory*, a *criticism theory* and a *control module*.

The rest of this part suggests ways of instantiating this architecture so as to obtain, in a systematic way, relevant domain-specific applications of dynamic argumentation systems. At this point we return to the scenarios introduced in Chapter 2 to describe possible argumentation systems to solve those problems. We first illustrate the use of this architecture in a safety-engineering domain (Chapter 11), before describing an application in the context of multi-agent negotiation (Chapter 12).

10.1 The Theory

As discussed in Chapter 4, argument dynamics is defined around the notion of a theory, so it only makes sense to keep it at the centre of the architecture. Theories represent the *objects* about which we argue.

Definition 10.1 (Theory) *Let* $(\mathcal{L}, \vdash)$ *be a logical system, and* $\mathcal{F}_{\mathcal{L}}$ *be the set of axioms in* $\mathcal{L}$*. A theory* Π *is any consistent subset of* $\mathcal{F}_{\mathcal{L}}$*.*[1] □

Notice that this is where arguments and attacks are generated (via possible argument-based schemata), and so far the types of schemata used in Part II depended only on the theory itself. But remember from the discussion in Section 8.3.2 that domain-specific schemata may sometimes be over-specified, so it is interesting to leave certain open parameters to be automatically instantiated during the argumentation, allowing for points of attack to be provided by an external source. This, though, requires an additional theory.

10.2 The Criticism Theory

The intuition behind a criticism theory is to provide potential points of attack to arguments within the architecture itself. These can guide the generation of attacks in the theory Π, but are defined separately from it by means of an additional theory Π_{crit}.

Definition 10.2 (Criticism Theory) *Let* $(\mathcal{L}_{crit}, \vdash_{crit})$ *be a logical system, and* $\mathcal{F}_{\mathcal{L}_{crit}}$ *the set of axioms in* $\mathcal{L}_{crit}$*. A criticism theory* Π_{crit} *is any consistent subset of* $\mathcal{F}_{\mathcal{L}_{crit}}$*.* □

The notion of argument can be defined analogously to Definition 4.3 by means of $\vdash_{crit}$, but no notion of attack is specified within the criticism theory itself. Arguments from Π_{crit} are mapped onto the theory in order to instantiate certain attacks in Π.

Note that the criticism theory is not a mere partition of the theory itself, because it may involve different inference mechanisms. In fact, the provability relations from theory and criticism theory might be different (e.g. $\vdash$ can be deductive and $\vdash_{crit}$ abductive,

[1] This definition corresponds to Definition 4.2 in Chapter 4.

as illustrated in the next chapter), and this is one reason for defining them separately. This dissociation, however, is not always clear because the theories might clash when the underlying languages are equivalent, as in the case of the Abstract Argumentation Framework which is abductive at the meta-level although it uses a deductive monotonic logic at the object-level.

The process for interpreting arguments from Π_{crit} to Π is based on the relation:

$$map \subseteq \mathcal{L} \times \mathcal{L}_{crit}$$

which essentially associates consequences from both theories,[2] identifying which sentences in the criticism theory correspond to which sentences in the theory. This process is described below and illustrated in Figure 10.1:

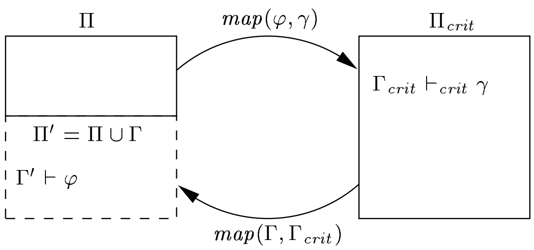


Figure 10.1: Argument level: generating arguments based on a criticism theory.

1. Let φ be a sentence in $\mathcal{L}$.
2. If $map(\varphi, \gamma)$ holds, then γ is a point of contact between the additional theory and the main one.
3. Let $\Gamma_{crit} \vdash_{crit} \gamma$ be an argument for γ in Π_{crit}.
4. Let Γ be the corresponding set of sentences in $\mathcal{L}$ obtained from Γ_{crit} according to the mapping above; i.e. $map(\Gamma, \Gamma_{crit})$.
5. Then in the extended theory $\Pi' = \Pi \cup \Gamma$ it should be possible to derive an argument $\Gamma' \vdash \varphi$ for φ that is based on the corresponding argument for γ in Π_{crit}.

[2] The relation $map(\varphi, \gamma)$ is said to hold between two formulae φ, γ if $(\varphi, \gamma) \in map$. Analogously, $map(\Gamma, \Gamma')$ holds between two sets of formulae if for every $\varphi \in \Gamma$ there exists $\gamma \in \Gamma'$ such that $map(\varphi, \gamma)$, and vice-versa.

Notice that it is hard if not impossible to place any general constraints on the mapping relation so that the process above is always guaranteed to give a corresponding argument in the main theory. There is more in sharing inferences than just translating expressions between logical systems (Corrêa da Silva et al. 1999)—for instance, the inference mechanisms of each systems need to be compatible. This is a difficult assumption to make, but in our case this process is quite disciplined and regulated.

The reason why a criticism theory can provide points of attack to the theory is because conditions in an attack-based revision schemata may involve the generation of arguments from this theory as a way of instantiating certain elements in a schema. While Figure 10.1 gives the general intuition behind mapping arguments from one theory to another, exactly which axioms are added or altered in the theory are defined within a schema by using the sorts of methods discussed in the previous part. In this way, the mapping relation above can be viewed as a special type of revision schema that depends on external theories to be instantiated. It becomes more a mapping of concepts than of inferences, and whether the intended argument can then be generated is ensured by the properties associated to each schema as discussed in Chapter 7.

Finally, the last type of component in our architecture accounts for the notion of priorities and preferences between arguments.

10.3 The Control Module

In human argument it is often the case that extra information is applied to control the generation of arguments, for instance when preferences are used for deciding between conflicting arguments (Prakken and Sartor 1997; Simari and Loui 1992; Brewka 1996; Amgoud and Cayrol 1998). Remember that our definition of attack[3] already incorporates the notion of preference between arguments (and which could be checked by associated properties during the argumentation), although throughout Part II it was assumed that every argument had equal strength.

The role of the control module is to define comparative and prioritisation measures for arguments, and also to specify criteria based on these measures for choosing stronger

[3] See Definition 4.4 in Chapter 4.

arguments, or for adjudicating between conflicting arguments. Of course criteria for deciding whether arguments are preferred may not always exist, but if they do they are likely to be domain-specific (Carbogim et al. 2000b; Konolige 1988; Prakken and Sartor 1997).

There are many ways one can capture the notion of comparative measure between arguments, such as defining a strict partial ordering on the set of defeasible rules (Prakken and Sartor 1997), or adopting the specificity principle (Simari and Loui 1992). But the control module also allows other types of prioritisation based on the individual strength of arguments, which could be done in different ways, e.g. by prioritising consequences in the theory or by prioritising arguments from the criticism theory. Notice that the study of preferences is a subject in itself, and the focus on this thesis has been more on the structure of arguments. In any case, Section 11.4 briefly discusses the possibilities for prioritisation in this architecture.

So as to allow adaptation separately from the rest of the framework, we characterise the control module as a meta-level component which treats theory and/or criticism theory as sources of information and propagates priority measures through them accordingly. This architecture—sketched in Figure 10.2—is similar to the type of layered meta-interpreter described in (Yalcinalp and Sterling 1991; Sterling and Shapiro 1994), and is useful for separating the parts dealing with priorities from those dealing with the structure of the arguments.

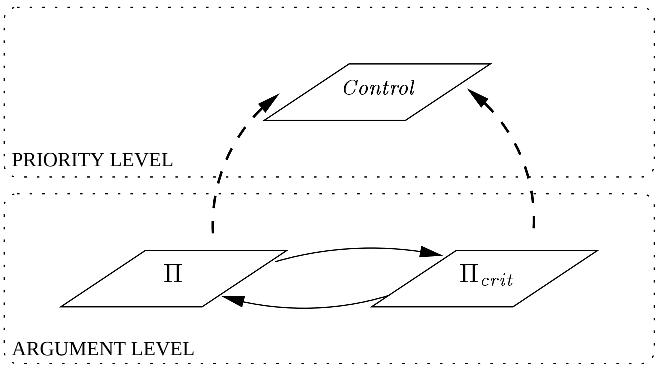


Figure 10.2: Architecture overview: interactions between the control module and the theories in the argument level are of a different nature than those between theory and criticism theory, and thus are represented by dashed arrows rather than by the solid arrows depicted in Figure 10.1.

Definition 10.3 (Control Module) *Given a logical system* $(\mathcal{L}', \vdash')$ *the control module defines a priority measure on* $\mathcal{L}'$ *and a mechanism for propagating these measures on top of the argument generation mechanism* $\vdash'$. $\square$

This three-component architecture extends the formalism presented in Part II in order to account for external instantiation mechanisms and for attacks based on priorities and preferences. Defining these separately allows for different strategies to be used in each of them, which can be useful for understanding even more of the dynamics in argument. Together with the library of possible argument schemata, this architecture can derive and compare the arguments and attacks that will be needed for generating dynamic arguments[4] by the sorts of mechanisms explored in the previous part (and outlined in Figure 7.1). As before, attacks allow changes to be made to the theory, but now these can be based on preference criteria for comparing arguments, priority measures for qualifying attacks, and on reasoned criticism arguments with the source of these criticisms explicitly defined.

This is a quite generic and unrestricted description so as to allow many possibilities for instantiation. This thesis in particular explores such possibilities in two different domains, namely safety-engineering in Chapter 11, and agent negotiation in Chapter 12.

[4] As discussed in Section 9.1, this is similar to the **Argument Generation Unit** in (Dung 1995).

Chapter 11

Worked Example: Instantiating the Architecture

In Section 3.4 we briefly discussed the importance of safety arguments in safety-critical domain. Now this chapter looks at how the architecture proposed in Chapter 10 can be instantiated to describe certain relations in examples taken from the safety-engineering community, and whether dynamic arguments can support the design and development of models, being used as part of safety cases for supporting that the design of the proposed system is acceptably safe (Krause et al. 1997; Gurr 1997).

More specifically, we consider the safety argument in Sections 2.1.1 and 9.5 in which fault tree analysis was used as a source of possible arguments against the safety of a pressure tank control system, guiding the revision of a system model in order to increase its acceptability with respect to known faults (Vesely et al. 1981). The following sections describe exactly how this example can be modelled in our dynamic argumentation framework, derived from the existing implementation of the system. This is done in two steps:

- first, we define the three components of the architecture as described in Chapter 10, namely a theory, a criticism theory, and a control module;
- second, we instantiate the mechanism for generating dynamic arguments discussed in Part II by defining a suitable library of domain-specific revision schemata.

The implementation described in Chapter 8 is then used to automatically generate

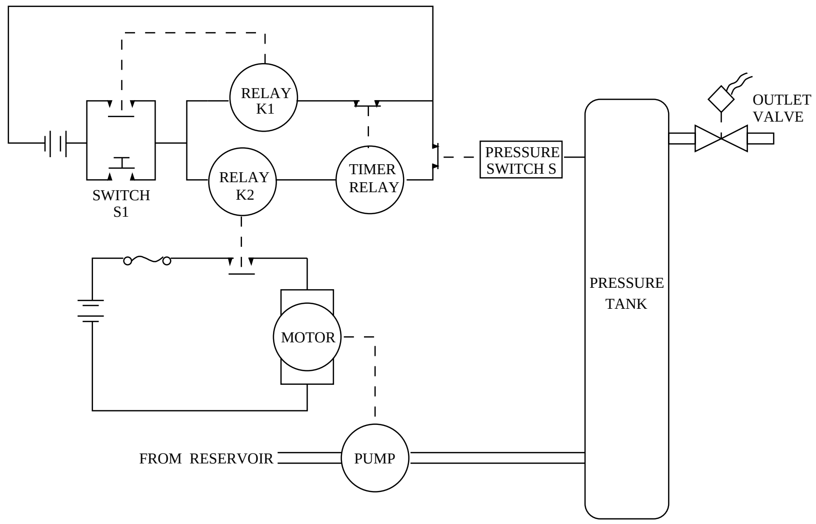


Figure 11.1: A pressure tank system (see Figure 2.2).

dynamic arguments about the pressure tank system being operational, and produces acceptable theories—in this case, models of the system—if all attacks based on the defined revisions and on the fault theory are successfully dismissed.[1]

11.1 Instantiating the Architecture in a Safety Domain

We start by defining the components in the architecture. For the sake of clarity here we just refer to fragments of the instantiation. The complete architecture definition in Prolog as used in the implementation for this example is described in Appendix E.

11.1.1 The Theory: The Pressure Tank Model

In this case, the theory—i.e. the object about which we argue—is the model of the pressure tank system in Figure 2.2 (reproduced here in Figure 11.1). The following is one way to express this model in terms of a Horn clause resolution-based system.

[1] See Definition 4.7 of a dynamic argument.

$$\begin{array}{rcl}
operational_tank(T) & \leftarrow & on_motor(T) \ \wedge \ not_full(T) \\
operational_tank(T) & \leftarrow & off_motor(T) \ \wedge \ pressurised(T) \\
not_operational_tank(T) & \leftarrow & on_motor(T) \ \wedge \ pressurised(T) \\
on_motor(T) & \leftarrow & closed(relay(k2), T) \\
off_motor(T) & \leftarrow & open(relay(k2), T) \\
closed(relay(K), T) & \leftarrow & energised(relay(K), T) \\
open(relay(K), T) & \leftarrow & deenergised(relay(K), T)
\end{array}$$

As in Section 9.5, here we adopt a simplified version of the pressure tank model, where *energised*/*deenergised* and *pressurised*/*not_full* are observable predicates in the sense that they are given as facts in the theory. For instance, assume that at time 60 the tank is observed to be pressurised, and relay $k2$ deenergised:[2]

$$\begin{array}{rcl}
pressurised(60) & \leftarrow & true \\
deenergised(relay(k2), 60) & \leftarrow & true
\end{array}$$

In our implementation, we use the expression $main(T)$ as a way to identify the main theory T in the architecture. If $(\mathcal{L}, \vdash)$ is the underlying logical system, then expressions $theory(T, \Pi)$ and $provability(T, P)$ are used to define the set of axioms Π corresponding to the initial theory in $\mathcal{L}$, and a predicate P for generating arguments based on the provability relation $\vdash$. In this example, this is instantiated in Prolog as follows:

```
main(ptmodel).
provability(ptmodel, solve).
theory(ptmodel, TInit).
```

where:

- `TInit` is the list of axioms above defining the functioning of the system, corresponding to the axioms in Π, and
- `solve` is a meta-interpreter that gives an argument for a sentence from a list of axioms according to resolution-based proof rules.

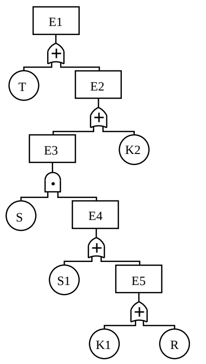
Remember that in our Prolog implementation we represent sets of axioms as lists. For ease of reference in our discussion we associate a number with each axiom. `TInit` is then represented as follows:

[2] The specification in Appendix E is more complex because it models the behaviour of relays and the pressurisation of the tank in terms of the behaviour of the other components and the given pressurisation time.

$$
\begin{array}{rrcl}
1 & operational\_tank(T) & \leftarrow & on\_motor(T) \ \wedge \ not\_full(T) \\
2 & operational\_tank(T) & \leftarrow & off\_motor(T) \ \wedge \ pressurised(T) \\
3 & not\_operational\_tank(T) & \leftarrow & on\_motor(T) \ \wedge \ pressurised(T) \\
4 & on\_motor(T) & \leftarrow & closed(relay(k2), T) \\
5 & off\_motor(T) & \leftarrow & open(relay(k2), T) \\
6 & closed(relay(K), T) & \leftarrow & energised(relay(K), T) \\
7 & open(relay(K), T) & \leftarrow & deenergised(relay(K), T) \\
8 & pressurised(60) & \leftarrow & true \\
9 & deenergised(relay(k2), 60) & \leftarrow & true
\end{array}
$$

11.1.2 The Criticism Theory: The Fault Tree Model

A candidate theory for a criticism theory is the fault tree model associated with the system. As discussed in Section 2.1.1 a fault tree is a model of the faults that can lead to an unsafe event in a system, and which is defined separately from the system model itself. Fault trees are basically composed of *and-* and *or-gates*, and therefore can be easily expressed in a declarative way in terms of Horn clauses. Figure 11.2 gives the basic fault tree for the pressure tank system in Figure 11.1 as described in (Vesely et al. 1981).



E1	top event (tank rupture after the start of pumping)
E2, E3, E4, E5	intermediate fault events
R	primary failure of timer relay
S	primary failure of pressure switch
S1	primary failure of switch S1
K1	primary failure of relay K1
K2	primary failure of relay K2
T	primary failure of pressure tank

Figure 11.2: Basic fault tree for the pressure tank example: circles denote basic events (faults) that require no further development, whereas boxes denote intermediate events in which a fault occurs because of one or more antecedent causes acting through logic gates. *Or-gates* and *and-gates* are represented by $+$ and $\cdot$, respectively.

The following is a fragment of this fault tree model represented here in our declarative style. The top event—denoted here by *tank_rupture*—occurs if the pump continuously operates for more than 60 seconds, which may happen if relay *k2* contacts fail to open after this time interval.

$$
\begin{array}{llcl}
1 & \mathit{tank\_rupture} & \leftarrow & \mathit{continuous\_pump\_operation} \\
2 & \mathit{continuous\_pump\_operation} & \leftarrow & \mathit{primary\_failure}(\mathit{relay}(k2))
\end{array}
$$

Let the underlying criticism language $\mathcal{L}_{crit}$ be a Horn clause based language, and $\vdash_{crit}$ be an abductive provability relation. We define a set of abducibles that correspond to the basic events in the fault tree, and arguments for *tank_rupture* are based on assumptions abductively selected from this set. For instance:

$$\{\mathit{primary\_failure}(\mathit{relay}(k2))\} \vdash_{crit} \mathit{tank\_rupture}$$

Note that identifying minimal cut sets in fault trees—i.e. the combination of failures leading to system fault—is equivalent to applying abduction with minimality constraints to the corresponding declarative model. Hence $\vdash_{crit}$ differs from the deductive inference used to determine consequences within the theory.

In our implementation, we use the expression $\mathit{crit}(T_c)$ as a way to identify a criticism theory T_c in the architecture. Analogously as in the case above, we use expressions $\mathit{theory}(T_c, \Pi_{crit})$ and $\mathit{provability}(T_c, P_c)$ to define the set of axioms Π_{crit} corresponding to this criticism theory in $\mathcal{L}_{crit}$, and a predicate P_c for generating arguments based on the provability relation $\vdash_{crit}$. The sort of fault tree based reasoning above can then be characterised in Prolog as follows:

```
crit(ftree).
provability(ftree, solve_abd).
theory(ftree, TCrit).
```

where:

- `TCrit` is a list of axioms defining the fault tree model, corresponding to the axioms in Π_{crit}, and
- `solve_abd` is an abductive meta-interpreter for these axioms.

Figure 11.3 illustrates the argumentation process for generating attacks based on a criticism theory for this particular safety argument. Π and Π_{crit} are fragments of the

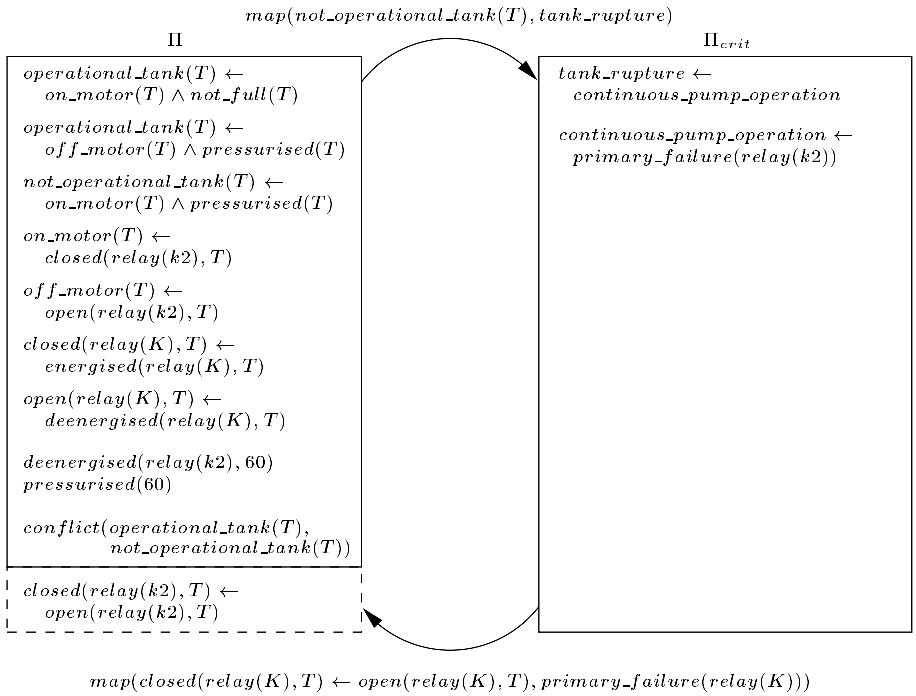


Figure 11.3: Generating attacks to the pressure tank model based on the fault theory.

pressure tank system model and the associated fault tree model, respectively. From the argument for *tank_rupture* in Π_{crit} which is based on the primary failure of relay *k2*, we add to the theory the axiom $closed(relay(k2), T) \leftarrow open(relay(k2), T)$ for representing this type of failure—namely, *k2* is closed when it should be open. Using this premise we can derive an argument for $not\_operational\_tank(60)$ which attacks the argument for $operational\_tank(60)$ in Π.

To illustrate why we cannot *flatten* all of this into a single theory, as we would have to do if we followed the approach described in the Abstract Argumentation Framework,[3] consider the example in Figure 11.4. If we interpret mappings m_1 and m_2 in (a) as implications, then we merge the theories as in (b). But then we do not have a means of driving the non-monotonic revisions to the argument, since we do not know that the fault structure is tested differently from the rest (hence we do not know where to apply abduction).

Recall that mappings between the theory and a criticism theory can be specified as

[3] See Sections 3.1.4 and 9.5.

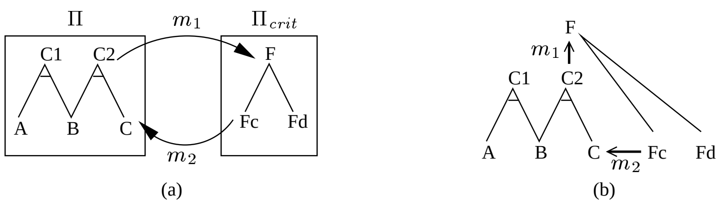


Figure 11.4: Our proposal (a) and the *flattened* equivalent (b).

argument schemata. The mapping above, for instance, corresponds to an operation for adding a substantiated clause based on arguments generated from the fault theory. The fault theory gives exactly which components can be validly instantiated in the domain-specific schema below so that, if they fail, they should lead to a system fault. Notice that this schema is obtained from the general schema for adding substantiated clauses (see Section C.2.2) in a similar way as described in Section 8.3.

Domain-specific Schema PRIMARY FAILURE OF ACTIVE COMPONENTS: $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 6 \Rightarrow 9 \Rightarrow 14$

$$\begin{array}{c} in(operational\_tank(T), A, \Pi), \\ add(substantiated\_rule(P)),\ revise(\Pi, \{\}, \{P\} \cup \mathcal{A}', \Pi'), \\ in(not\_operational\_tank(T), A', \Pi') \end{array}$$

Properties: $\left\{ \begin{array}{c} supports(A', not\_operational\_tank(T) : \mathbf{in},\ \Pi'), \\ satisfiable(on\_motor(T)\ \wedge\ pressurised(T),\ \Pi \cup \mathcal{A}') \end{array} \right\}$

Conditions:
$$\begin{array}{l} operational\_tank(T) : \mathbf{in} \in \mathcal{G}_A, \\ not\_operational\_tank(T),\ on\_motor(T)\ \wedge\ pressurised(T) \in \mathcal{L}, \\ P = not\_operational\_tank(T) \leftarrow on\_motor(T)\ \wedge\ pressurised(T), \\ gen\_argument(ftree, tank\_rupture, A_{crit}), \\ \mathcal{A}' = \{closed(C,T) \leftarrow open(C,T) \mid \\ \qquad\qquad\qquad primary\_failure(C) \in A_{crit}, type(C) \neq tank\} \end{array}$$

such that arguments are generated via the corresponding meta-interpreter *solve_abd*:

```
gen_argument(ftree, X, A) :-
    theory(ftree, TCrit),
    solve_abd(X, A, TCrit).
```

An interesting point to make about this schema is that it is not *fundamental* (i.e. *trivial*, *elementary* or *updating*) like most schemata discussed so far. Apart from the clause for

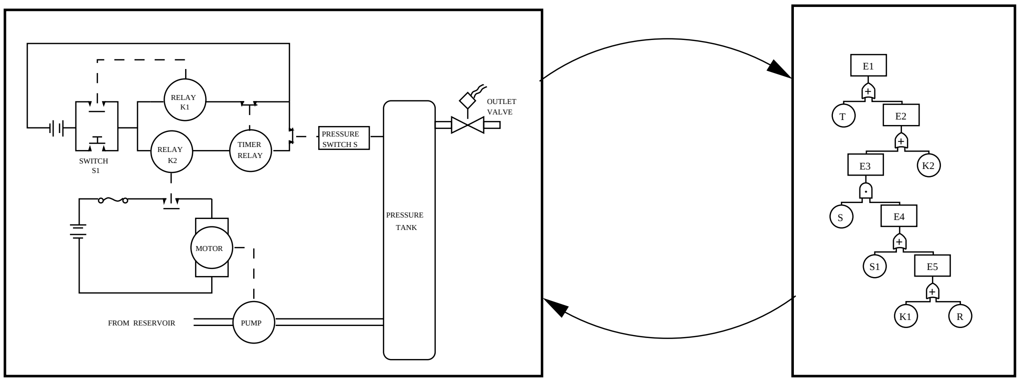


Figure 11.5: Generating attacks to models based on fault theories.

deriving $not\_operational\_tank(T)$—which is the clause intended to be substantiated—we need to add the axioms in $\mathcal{A}'$ so as to substantiate it. In fact, remember from Section 7.3.5 that more complex operations can be defined by expanding the sets $\mathcal{R}$ and $\mathcal{A}$ in a way that the associated properties still hold.[4] The only adaptation is that any properties involving the original theory Π should consider also the *extra* axioms—e.g. in the schema above rather than checking that the body of the main clause is satisfiable in Π we need to check that it is satisfiable in $\Pi \cup \mathcal{A}'$, so that P can in fact give the intended argument in $\Pi' = \Pi \cup \mathcal{A}' \cup \{P\}$.[5]

The last two conditions in the schema give exactly what extra axioms should be added based on the criticism argument for *tank_rupture*. The reason why we disregard the possible primary failure of the tank itself is because, according to standard techniques for fault tree evaluation, the tank is a passive component (Vesely et al. 1981, p. VIII-12) rather than an active component such as a relay or a switch.

[4] Recall from Definition 4.5 that a structural revision operation is characterised by a pair $(\mathcal{R}, \mathcal{A})$ of sets of axioms.

[5] Note that the axiom P in schema PRIMARY FAILURE OF ACTIVE COMPONENT is already in the theory. Although theories are implemented as lists they are supposed to behave like sets, so the addition of a new element which is equivalent to an existing one does not create a duplicate (equivalent) entry in the list.

11.1.3 The Control Module

There are various types of results that can be obtained from fault tree evaluation techniques, including determination of minimal cut sets, numerical probabilities associated with these sets, and quantitative and qualitative rankings of contribution to system failures (e.g. according to the size of each minimal cut set). That means that there are also various ways of prioritising the attacks which are based on these subsets of assumptions from the fault tree model.

One way for instance is by assigning probabilities to the basic events and propagating these through the fault tree model according to the laws of probability theory. We can estimate the probability of the top event being derived from an argument and the criteria for deciding whether this argument defeats a safe argument from the theory is based on the analysis of this probability. For instance, this can be compared to a threshold importance value, under which attacks based on the argument can be disregarded. That means that not every combination of events leading to the top event needs to trigger a revision in the model so as to generate an attacking argument; i.e. an attack is relevant enough to be a concern if the probability of the minimal cut set on which it is based is not acceptable for safety standards. This closely resembles the method of analysis for fault tolerance used in practice, as described in (Vesely et al. 1981).

As discussed in Chapter 10, we characterise this sort of prioritisation by layered meta-interpreters so as to propagate priority measures on top of the generation of arguments. In our representation, the expression $\mathit{filter}(P_1, P_2)$ denotes that the meta-interpreter P_2 treats the definition of meta-interpreter P_1 for argument generation as a source of information. In this example, the expression below:

```
filter(solve_abd, solve_filter).
```

states that *solve_filter* considers the probabilities assigned to basic events and propagates these appropriately as arguments are constructed by *solve_abd*, thus filtering the arguments that are strong or relevant enough and hence allowed to be advanced as attacks. The predicate *solve_filter* (rather than *solve_abd*) is used to generate prioritised arguments from the fault tree model:

```
gen_argument(ftree, X, A) :-
    theory(ftree, TCrit),
    solve_filter(solve_abd(X, A, TCrit)).
```

This is one way of prioritising arguments in the control module; others are discussed in Section 11.4.

11.2 Generating Dynamic Arguments

With the architecture components defined in this way, we can then use the mechanisms discussed in Part II to generate dynamic arguments in this domain. Note, though, that with only one type of schema—namely PRIMARY FAILURE OF ACTIVE COMPONENTS—arguments can just introduce faults to the model. But as discussed in Section 9.5, it is important to allow adaptation of the model. One might say that the base model of the system—i.e. the initial model—satisfies all points of attack given by the corresponding fault tree, although only the attacks that are strong enough (according to the prioritisation definition) can in fact be advanced. In these cases, we should try to dismiss these attacks by making appropriate changes to the structure of the model.

However, the current specification does not give any means for that. Even if we consider trivial schemata, there are no alternative arguments for *operational_tank*, and in any case these would not be enough to raise the confidence that the system is acceptably safe. To reinstate a particular conclusion after it has been attacked we need to perform some action to change the theory such that this attacking argument can no longer be derived. This is illustrated in Section 2.1.1, where a parallel relay was introduced to improve system safety. One way to undermine the argument based on the failure of *k2*, and thus to considerably improve system safety, is by adding some redundancy to the system (i.e. another relay in parallel to *k2*).

What it means for a new relay to be added in parallel to an existing relay is that the new relay must have the same behaviour as the original one. Moreover, if some conclusion depended on the original relay being open, the same conclusion depends on the new relay being open (only one relay being open is sufficient to derive it). In terms of the model, we can duplicate the clauses defining the behaviour of the original relay in order to define the behaviour of the new relay (whether and when its contacts are open or closed), as well as those clauses in which the preconditions involve the original relay being open. Once we add a redundant relay, if some conclusion depended on the original relay being *closed*, it now depends on *both* relays being closed, and this is the clause in the model that needs to be elaborated in order to block certain undesired conclusions. This could be captured by the complex (i.e. non fundamental) domain-specific schema below for adding a redundant relay, obtained from the general schema for elaborating a rule (see Section C.5.1) in a similar way as described in Section 8.3.[6]

Domain-specific Schema REDUNDANCY OF RELAY: $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 5 \Rightarrow 12 \Rightarrow 20$

$$\frac{in(X, A, \Pi),}{retract(elaboration(P)),\ add(elaboration(P')),\ revise(\Pi, \{P\}, \{P'\} \cup \mathcal{A}'', \Pi'),\\ out(X, A, \Pi')}$$

Properties: $\left\{ \begin{array}{c} supports(A, X : \mathbf{out}, \Pi'), \\ unify(X, H), \\ \neg satisfiable(B\sigma, \Pi \cup \mathcal{A}') \end{array} \right\}$

Conditions:
$$\begin{array}{l} X : \mathbf{in} \in \mathcal{G}_A, \\ closed(relay(R), T) \leftarrow open(relay(R), T) \in A, \\ P = H \leftarrow B_1 \wedge ... \wedge B_m \in A, \\ B_i = closed(relay(R), T), \\ new\_component\_id(R_1), \\ B = closed(relay(R_1), T) \in \mathcal{L}, \\ P' = H \leftarrow B_1 \wedge ... \wedge B_i \wedge B \wedge B_{i+1} \wedge ... \wedge B_m, \\ \sigma = mgu(X, H), \\ \mathcal{A}' = \{P_{1[R/R_1]} \mid P_1 \in \Pi, P_1 \neq P\} \end{array}$$

Thus, just because we are adopting an architecture that allows instantiation from external sources it is not strictly necessary for *all* schemata to be instantiated in that way. Schemata like the one above suggest general ways for adapting models according to known faults that have been introduced to the model deliberately. They can then be applied to other arguments, thus producing alternatives for design based on the initial model. These can vary, for instance, according to different measures of prioritisation (i.e. which combination of events can be *safely* dismissed) and also to the ordering in which arguments have been generated (e.g. adding a redundant component might block other attacks from being supported based on the fault tree). Once a dynamic

[6] As defined in Appendix A, $F_{[T_1/T_2]}$ denotes the formula obtained from a formula F by replacing every occurrence of the term T_1 by the term T_2.

argumentation process is over—i.e. once every minimal cut set has been dismissed—we may have produced an alternative, more elaborated model. Once again, one may want to consider the fault tree model for the new structured system and rerun the process.

11.3 A Dynamic Argument in the Safety Domain

Now that the architecture and a catalogue Φ of attack-based revision schemata have been specified, the system in Section 8.1.2 can be used to generate dynamic arguments in an automated form.

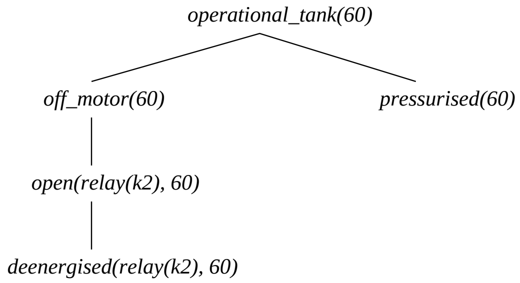
Let `TInit` be the theory in Section 11.1.1, and Φ be:

$$\{\textsc{primary failure of active components}, \textsc{redundancy of relay}\}.$$

Assume also that the threshold importance value is set e.g. to 0.1, meaning that an attack based on the fault tree model (criticism theory) can only be advanced if the corresponding minimal cut set contributes in more than 10% to the probability of the top event being derived.[7] Below we present a dynamic argument process about the tank being operational as generated by our implementation, in the same format as the aflatoxin argument in Section 8.2.

According to Definition 4.7, the first argument to be advanced is a justification supporting the main claim that the pressure tank system is operational, for instance, at time 60.

Argument A_0 is a justification for $\mathit{operational\_tank}(60)$.



[7] The relative quantitative importance of minimal cut sets is obtained by taking the ratio of the minimal cut set probability to the total system probability (Vesely et al. 1981).

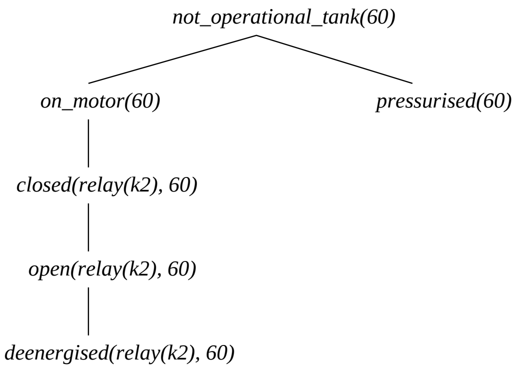
Revision ϕ_1 is obtained from schema PRIMARY FAILURE OF ACTIVE COMPONENTS, and from the argument for *tank_rupture* in the fault tree model which is based on $\{primary\_failure(relay(k2))\}$. As described by Vesely et al. (1981), the relative quantitative importance of this minimal cut set is 86%, and hence above the stipulated threshold.

In this way,

$$\phi_1 : add(substantiated\_rule \begin{pmatrix} not\_operational\_tank(T) \leftarrow \\ \quad on\_motor(T) \ \wedge \ pressurised(T), \\ \\ closed(relay(k2), T) \leftarrow \\ \quad open(relay(k2), T) \end{pmatrix})$$

is an attack-based revision that can be used to construct an argument for justifying that the system is not operational at time 60. Moreover, the properties accumulated during the instantiation can ensure that the generated argument in fact supports that $not\_operational\_tank(60)$: **in**.

Argument A_1 is a justification for $not\_operational\_tank(60)$.



Revision ϕ_2 adapts the model via schema REDUNDANCY OF RELAY, adding a new relay $k2'$ in parallel to $k2$ in order to refute the claim that the motor is on at time 60.

In this way,

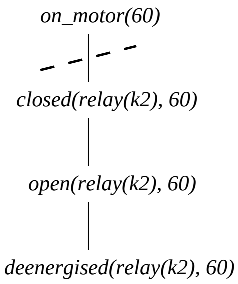
$$\phi_4 : retract(elaboration \begin{pmatrix} on\_motor(T) \leftarrow \\ \quad closed(relay(k2), T) \end{pmatrix})$$

and

$$add(elaboration \begin{pmatrix} on\_motor(T) \leftarrow \\ \quad closed(relay(k2), T) \ \wedge closed(relay(k2'), T), \\ \\ deenergised(relay(k2'), 60) \leftarrow true, \\ \\ off\_motor(T) \leftarrow open(relay(k2'), T) \end{pmatrix})$$

is an attack-based revision that that refutes argument A_1.

Argument A_2 is a refutation of $on\_motor(60)$.



At this point no other attack can be generated such that the claim $operational\_tank(60)$ becomes unsubstantiated. The revised theory below is said to be acceptable with respect to the faults in the fault tree model.

1	$operational\_tank(T)$	$\leftarrow$	$on\_motor(T) \wedge not\_full(T)$
2	$operational\_tank(T)$	$\leftarrow$	$off\_motor(T) \wedge pressurised(T)$
3	$not\_operational\_tank(T)$	$\leftarrow$	$on\_motor(T) \wedge pressurised(T)$
4	$on\_motor(T)$	$\leftarrow$	$closed(relay(k2), T) \wedge closed(relay(k2'), T)$
5	$off\_motor(T)$	$\leftarrow$	$open(relay(k2), T)$
6	$closed(relay(K), T)$	$\leftarrow$	$energised(relay(K), T)$
7	$open(relay(K), T)$	$\leftarrow$	$deenergised(relay(K), T)$
8	$pressurised(60)$	$\leftarrow$	$true$
9	$deenergised(relay(k2), 60)$	$\leftarrow$	$true$
10	$closed(relay(k2), T)$	$\leftarrow$	$open(relay(k2), T)$
11	$deenergised(relay(k2'), 60)$	$\leftarrow$	$true$
12	$off\_motor(T)$	$\leftarrow$	$open(relay(k2'), T)$

In a nutshell, there are two advantages in defining dynamic argument systems based on this architecture: one is because we allow external sources of criticism to be represented; the second is to allow modular representation of priorities. This separation is interesting because it allows, for instance, different inferences to be used and different strategies of prioritisation to be tested. Now that we have seen an example of argument prioritisation, the next section discusses some of the uses of priorities in the generation and selection of arguments within our architecture.

11.4 Argument Prioritisation in the Architecture

As mentioned in Chapter 4 and illustrated in Section 11.1.3, the use of defeat and prioritisation criteria to represent that certain arguments may be preferred over others is an important element in the generation and development of argumentation processes.

The issue of preferences in argumentation has been extensively studied in the literature, and various frameworks for dealing explicitly with priorities and with how preference relations can be integrated into argumentation systems have been proposed (Amgoud and Cayrol 1998; Prakken and Sartor 1997; Brewka 1996; Vreeswijk 1993; Grosof 1997).

In general, prioritisation of arguments involve the aggregation of preference criteria given some precedence ordering. Very often it is assumed that a partial—i.e. transitive—ordering between arguments (or between axioms in the knowledge base) exist, based on which the notion of defeat is defined and conflicts are resolved. Examples of prioritisation criteria are the specificity principle, reliability of sources, or yet temporal precedence of arguments or axioms.

This section does not present a general account of priorities in argumentation, nor it proposes a specific representation for it (which is likely to be domain-specific, as discussed earlier in this thesis). Priorities and preferences are not a main part of this thesis, but it is interesting to note that our architecture also allows for prioritisation of individual arguments as a way of measuring the *quality* of these arguments, thus blocking some from being advanced and reducing the space of possible dynamic arguments. Most existing systems only consider preferences as a way of comparing two (conflicting) arguments.

Hence, given that up to this point we have presented details of a dynamic argument generator, various worked examples and an architecture for dynamic argument systems, we now briefly discuss suitable prioritisation representations linked to our applications, looking at some of the possibilities for priority handling within the architecture proposed in Chapter 10. For instance, the example above described a way to prioritise arguments generated from a fault theory in a safety-critical domain. This is one type of prioritisation which involves the filtering of arguments according to some *relevance* criterion, and which is discussed in Section 11.4.1.

Another possibility for prioritisation in our architecture, involving the primary comparison of arguments, is discussed in Section 11.4.2. Finally, Section 14.2.2 addresses some related issues in connection with the selection of arguments to be advanced.

11.4.1 Priority Criteria for Generating Arguments

This sort of prioritisation concerns the generation of individual arguments, and hence can be applied both to arguments generated in the main theory as well as to arguments generated in a criticism theory. Essentially, it is about the *quality* of the arguments. As described in Section 11.1.3, given a provability relation for argument generation—either in the theory or in a criticism theory—we can define a layered meta-interpreter that uses the definition of the first to propagate certain priority measures as arguments are generated, *filtering* those arguments that satisfy some threshold condition given some precedence ordering. In the example in Section 11.1.3, this was related to the quantitative contribution of minimal cut sets to system failure.

	Π_{crit}	Π_{crit} + filter
Π	any generated argument is relevant	prioritise criticism
Π + filter	prioritise consequences in the theory	prioritise criticism and prioritise consequences in the theory

Figure 11.6: Prioritisation in the generation of individual arguments.

Table 11.6 summarises the possibilities for prioritisation of argument generation in our architecture. The fault tree example for instance fits in the top-right box. Notice that this has nothing to do with checking whether the property *supports* holds or not. In that case, arguments may not be advanced because they have already been considered in the process. Here arguments may be blocked because they are not *relevant enough*, or *good enough*, in the domain.

11.4.2 Preference Relations for Comparing Arguments

Another possibility is to use priorities to block arguments from being advanced not because they are not relevant enough *per se*, but because they are not strong enough to defeat some conflicting argument. This sort of prioritisation does not occur in the context of generating individual arguments, but in the context of attacks and conflicts during the argumentation process. Hence, it only occurs within the main theory, when an attack of the form **in** $\rightsquigarrow$ **in** is advanced. If no prioritisation of this sort is defined, then no argument is preferred over any other (remember that this is the basic assumption we adopted throughout Part II of this thesis).

Comparative measures between conflicting arguments in the main theory can also be defined in the same layered style, in which a meta-interpreter is used on top of the argument generator to propagate some preference measure. But in this case, rather than comparing the conclusive force of one argument to some threshold value, the preference mechanism compares the relative force of two conflicting arguments. A meta-interpreter for argument comparison again uses the definition of the provability relation in the main theory to propagate certain preference measures according to the argument structure, taking as input any two arguments which can then be compared according to some precedence ordering.

Notice that we may have different criteria for argument filtering (e.g. we may only consider to be relevant those arguments involving less than five inference steps) and argument comparison (e.g. we may decide between two conflicting arguments, both consisting of less than five inference steps, by analysing their conclusive force based on an explicit partial order on the axioms in the underlying theory).

Chapter 12

Relating Argument Dynamics to a Multi-Agent Problem

Another potential area of application of dynamic arguments is that of negotiation between autonomous agents,[1] in which agents must come to a mutually acceptable agreement about some matter (Parsons and Jennings 1997; Parsons et al. 1998; Sierra et al. 1997b). In fact, Jennings et al. (1998) have characterised three general topics in research in negotiation, namely *negotiation protocols*, *agreement objects* and *agents' strategies*. The first focuses on defining the *rules of the game*, such as the types of participants, the possible negotiation states and valid actions of each participant in each state. The second is about specifying the range of negotiable issues—e.g. price, delivery date, quality—over which agreement is to be reached. Finally, the last is concerned with the agents' decision making strategies, and is often shaped by the first two.

The difference in focus between negotiation protocols and agreement objects is similar to the sort of distinction between protocol- and object-based argumentation discussed in Sections 3.3.3 and 3.3.4. Remember that we can emphasise different aspects of the process depending on what we want to formalise. On the one hand, emphasis is on communication between agents, and on defining protocols for exchanging messages containing proposals and counter-proposals, and for deciding which conclusion is acceptable to every agent involved. On the other hand, though, emphasis is on the structure of the agreement rather than on communication and exchange of messages. This is about negotiating complex terms and conditions of a proposed deal/agreement, and adjusting

[1] See discussion in Sections 2.1.2 and 3.3.

the terms of such agreements based on reasoned arguments by the agents involved.

Because the focus on argument dynamics is on the development of an object, we found that the particular problem of forming contracts between negotiating agents conforms to a style of reasoning similar to that of generating dynamic argument. This chapter describes a way to instantiate a system from the general architecture in Chapter 10 for generating arguments in this domain.

12.1 Contract-based Negotiation

Work on contracts is not new. Sierra et al. (1997a) proposed a model of negotiation based on contracts that are represented as collections of issues (variables) whose values need to be set. Through negotiation, an agent proposes values within its acceptable range until an assignment of values suiting every participant is obtained.

Although contracts are essentially collections of negotiable issues, some approaches focus less on the process of assigning acceptable values to negotiables than on structuring these in terms of logical rules. In the logic-programming community, for instance, Daskalopulu and Sergot (1997) have investigated the use of logic-based (automated) tools supporting the analysis and representation of legal contracts in large-scale, long-term engineering trading agreements. These are substantially more complex than *sales of goods* contracts. Reeves et al. (1999, 2000) also propose a way for representing contracts as courteous logic programs (Grosof 1997). The idea is to have a declarative description of the specification of a contract, and then generate final, executable contracts via automatic configuration for different types of auctions.

In our model of contract-based negotiation[2] we also consider contracts to be sets of logical rules which can, via dynamic argumentation processes, be adjusted based on reasoned arguments by the agents involved in the agreement. By *adjusting* we mean not only changing the values associated with negotiable issues, but also the structure of the corresponding rules and hence the relations between negotiables.

[2] The model we present in this chapter was initially proposed in (Carbogim and Robertson 1999).

12.1.1 Contract-based Negotiation as Dynamic Argumentation

The worked example in this chapter is based on the model for contract-based negotiation described in Section 2.1.2. Remember that contracts are objects which regulate agreements between two autonomous agents—a *consumer* and a *producer*—about the supply of a particular product (or service). The purpose of contract-based negotiation is to adjust the terms of this agreement so that it is acceptable for both parties involved. We assume that only two parties are involved in the negotiation, although an agent may also be involved in other different processes simultaneously, even playing distinct roles in distinct processes (e.g. being a producer in some contexts and a consumer in others).

The process of contract-based negotiation starts when one of the parties proposes a binding contract to regulate some agreement between producer and consumer. We can assume that the producer makes this first proposal (e.g. upon previous request from the consumer). This contract is now the *object of negotiation* between the parties and is represented as a set of formulae stating the conditions for accomplishing the agreement.

The consumer receives the contract from the producer and analyses it. If it agrees with the clauses, then the process of negotiation is over. But if the consumer has reasons to believe that this particular contract will not be successfully completed, it sends the contract back to the producer with the appropriate criticisms. The producer then tries to adapt some of the clauses in that particular contract in order to make it more acceptable, sending it back again to the consumer for further analysis. The process of adjusting the contract continues until there are no more criticisms (i.e. it is acceptable for both producer and consumer) or until one of the parties withdraws.

12.1.2 A Simple Language for Contracts

This section describes the basic scenario we use to develop our ideas on how negotiation relates to dynamic argumentation, and a simple specification language used to represent contracts in this scenario. In particular, we consider two types of agents:

Producers. The term $\mathit{producer}(X, P)$ denotes that agent X wants to sell product P.

Consumers. The term $\mathit{consumer}(Y, P)$ denotes that agent Y wants to acquire P.

Agents

producer X
consumer Y

Agreement

agent X to supply product P to agent Y

Contract

explicitly state the conditions for the agents to commit to this agreement

Figure 12.1: Basic scenario for contract negotiation between producer and consumer.

If a producer X has agreed to supply product P to a consumer Y, then a contract-based negotiation process is carried out by X and Y in order to adjust the clauses of this agreement so that it is acceptable to both parties. This scenario is summarised in Figure 12.1.

Being the producer, X proposes an initial contract to Y stating the conditions for success of the arrangement between them. These conditions might include the appropriate delivery of the product by X within the stipulated time, and the appropriate payment for it by the consumer Y. The form of a generic contract clause is given below, assuming an underlying Horn clause resolution-based system.

A contract between a producer X and a consumer Y for the supply of a product P is successfully completed if all the agreed terms $T_1, ..., T_m$ are fulfilled. Each term T_i—such as for instance quantity, delivery or payment— may depend on X, Y and P.

$$\begin{array}{lll} \mathit{contract\_completion}(X,Y,P) & \leftarrow & \mathit{producer}(X,P) \wedge \\ & & \mathit{consumer}(Y,P) \wedge \\ & & \mathit{fulfill}(T_1) \wedge ... \wedge \mathit{fulfill}(T_m) \end{array}$$

For instance, the following is a possible instantiation of this general clause with two contractual terms, namely *delivery* and *payment*.

A contract between a producer X and a consumer Y for the supply of a product P is successfully completed if the agreed terms of delivery of P are fulfilled by X, and the agreed terms of payment for P are fulfilled by Y.

$$\begin{array}{rcl} contract\_completion(X, Y, P) & \leftarrow & producer(X, P) \wedge \\ & & consumer(Y, P) \wedge \\ & & fulfill(delivery(X, Y, P)) \wedge \\ & & fulfill(payment(Y, X, P)) \end{array}$$

The question now is how to determine whether a particular term has been fulfilled or not. Terms in a contract usually specify values of negotiables and determine actions to be taken. Fulfilment then depends on whether the result of implementing these terms conforms to what is set by the contract-holders. For instance, we can say that payment terms are fulfilled if the consumer pays for the amount specified in the contract.

A term T in a contract is fulfilled if the result of its implementation conforms to the corresponding value V set in the contract.

$$fulfill(T) \quad \leftarrow \quad set(T, V) \wedge outcome(T, V)$$

Because it is impossible to predict the outcome of implementing certain conditions at the time of contract definition, we assume by default that whatever conditions specified in the contract will be implemented by the responsible agent accordingly.

$$outcome(T, V) \quad \leftarrow \quad set(T, V)$$

As we will see in the following example, this is useful because it gives points of attack and criticism within the contract. Contradictions arise when an agent argues that some specified contractual clause should be implemented differently, hence yielding a different outcome than the one initially indicated. In this way, it is possible to derive contradictory claims based on distinct outcomes V and V' for the same contractual term T.

12.1.3 An Example of Contract Formation

If we adopt this representation, as well as a model of time based on integer time points (for instance representing days), the following set of formulae specifies a contract between two agents a and b.

$$
\begin{array}{rrcl}
1 & \mathit{contract\_completion}(X,Y,P) & \leftarrow & \mathit{producer}(X,P) \wedge \\
 & & & \mathit{consumer}(Y,P) \wedge \\
 & & & \mathit{fulfill}(\mathit{delivery}(X,Y,P)) \wedge \\
 & & & \mathit{fulfill}(\mathit{payment}(Y,X,P)) \\
2 & \mathit{fulfill}(T) & \leftarrow & \mathit{set}(T,V) \wedge \\
 & & & \mathit{outcome}(T,V) \\
3 & \mathit{set}(\mathit{delivery}(X,Y,P),D) & \leftarrow & \mathit{production\_time}(X,P,D) \\
4 & \mathit{set}(\mathit{payment}(Y,X,P),(V,\mathit{std})) & \leftarrow & \mathit{price}(P,X,V) \\
5 & \mathit{outcome}(T,V) & \leftarrow & \mathit{set}(T,V) \\
6 & \mathit{producer}(a,p) & \leftarrow & \mathit{true} \\
7 & \mathit{consumer}(b,p) & \leftarrow & \mathit{true} \\
8 & \mathit{production\_time}(a,p,14) & \leftarrow & \mathit{true} \\
9 & \mathit{price}(p,a,10) & \leftarrow & \mathit{true}
\end{array}
$$

For a contract to be acceptable to an agent, we mean that the main conclusion for success—in this case $\mathit{contract\_completion}(a,b,p)$—is substantiated, and that the agent has no reason to attack it. We can say that the contract above is acceptable to agent a because it is consistent with its internal theory (since a proposed it), and because from this set of axioms we can derive a justification for $\mathit{contract\_completion}(a,b,p)$. In this case, the contract is successfully completed if delivery terms are fulfilled—i.e. product is delivered within two weeks— and also payment terms are fulfilled—i.e. the consumer pays the stipulated price for the product, say 10.

As noted before, some of the possibilities for contradiction in this model have to do with the value of the negotiables—in this case, there are three of them: the time for delivery D of the product P by the producer X to the consumer Y; and the amount V to be payed by the consumer Y, as well as the form of payment (initially set for std, i.e. standard 30-day payment). Hence expressions $\mathit{outcome}(\mathit{delivery}(X,Y,P),D)$ and $\mathit{outcome}(\mathit{delivery}(X,Y,P),D')$ are contradictory if D and D' are instantiated to different values.

At this point the producer a sends this contract to the consumer b, who investigates within its internal theory whether some conflicting arguments can be derived. Note that the consumer is not trying to block the conclusion in an strictly opponent fashion—b too wants to establish the agreement. But because the contract was proposed by the producer, we can just assume that it is acceptable to a but not necessarily to the consumer b. For instance, b might want the product to be delivered at a different date, one week earlier than what was initially proposed by the producer. The consumer then adds the following clause to the contract.

$$\begin{array}{l} outcome(delivery(a,b,p),D) \leftarrow \\ \qquad production\_time(a,p,D_1) \wedge \\ \qquad D \ is \ D_1 - 7 \end{array}$$

The clause above can be seen as a criticism to the initially proposed contract, as the consumer can now derive a conflicting argument for $outcome(delivery(a,b,p),14)$, namely $outcome(delivery(a,b,p),7)$. Remember from Chapter 6 that this sort of attack causes the main claim $contract\_completion(a,b,p)$ to become unsubstantiated.

The consumer b sends this version of the contract back to a, who tries to reconcile the criticism with the original clauses by attempting to re-substantiate the conclusion $contract\_completion$. A new version of the contract features changes based on b criticism. For example, a can accept b's request for an earlier delivery by updating the conclusion in the clause just introduced by b so that it is now used for specifying the new agreed delivery time.

$$\begin{array}{l} set(delivery(a,b,p),D) \leftarrow \\ \qquad production\_time(a,p,D_1) \wedge \\ \qquad D \ is \ D_1 - 7 \end{array}$$

The clause above specifically addresses the delivery of product p by a to b, as opposed to the more generic formula proposed initially. This may still be valid in general, but here we implement a sort of prioritisation based on recency allowing only the most recent clause among the possible (unifying) definitions of $set(T,V)$ to be used in a derivation.

But nothing comes without a price, so a may introduce other changes to compensate for this concession of delivering a product one week earlier than usual. For instance, a may increase by 10% the amount to be charged for p.

$$\begin{array}{l} set(payment(b,a,p),(V,std)) \leftarrow \\ \qquad price(a,p,V_1) \wedge \\ \qquad V \ is \ V_1 + 0.1 \times P \end{array}$$

After these changes a justification for $contract\_completion$ can again be derived, with delivery set to an earlier date but at a higher cost. The producer sends this new version of the contract back again to b, who can either agree with it or provide some more reasoned criticism. Suppose that b still does not find this deal acceptable and asks for a further discount of 15% on the value of the product. This is done by adding the following clause to the contract, which suggests a smaller cost to be charged for p than the one stipulated by a.

$$\begin{array}{l} \mathit{outcome}(\mathit{payment}(b,a,p),(V,\mathit{std})) \leftarrow \\ \qquad \mathit{price}(a,p,V_1) \wedge \\ \qquad V_2 \ \mathit{is}\ V_1 + 0.1 \times V_1 \wedge \\ \qquad V \ \mathit{is}\ V_2 - 0.15 \times V_2 \end{array}$$

Again there is a contradiction, and so *contract_completion* is once more unsubstantiated. This time a can accept b's request for a discount, but not without constraining the payment form from *standard 30-day* to *immediate*.

$$\begin{array}{l} \mathit{set}(\mathit{payment}(b,a,p),(V,\mathit{imm})) \leftarrow \\ \qquad \mathit{price}(a,p,V_1) \wedge \\ \qquad V_2 \ \mathit{is}\ V_1 + 0.1 \times V_1 \wedge \\ \qquad V \ \mathit{is}\ V_2 - 0.15 \times V_2 \end{array}$$

The producer sends this contract again for b's scrutiny. If b cannot find any more reasons for not accomplishing the agreement successfully—e.g. b has no target requirements with respect to payment form, and all other requirements with respect to delivery and price have been met—then b agrees with the current proposal. The final binding contract that sets the terms for the supply of p by a to b is then described below. Note that only the more recent clauses defining each contractual term are kept, in accordance with the sort of prioritisation mentioned above.

$$\begin{array}{llcl}
1 & \mathit{contract_completion}(X,Y,P) & \leftarrow & \mathit{producer}(X,P) \wedge \\
 & & & \mathit{consumer}(Y,P) \wedge \\
 & & & \mathit{fulfill}(\mathit{delivery}(X,Y,P)) \wedge \\
 & & & \mathit{fulfill}(\mathit{payment}(Y,X,P)) \\
 & & & \\
2 & \mathit{fulfill}(T) & \leftarrow & \mathit{set}(T,V) \wedge \\
 & & & \mathit{outcome}(T,V) \\
 & & & \\
5 & \mathit{outcome}(T,V) & \leftarrow & \mathit{set}(T,V) \\
 & & & \\
6 & \mathit{producer}(a,p) & \leftarrow & \mathit{true} \\
7 & \mathit{consumer}(b,p) & \leftarrow & \mathit{true} \\
8 & \mathit{production_time}(a,p,14) & \leftarrow & \mathit{true} \\
9 & \mathit{price}(p,a,10) & \leftarrow & \mathit{true} \\
 & & & \\
10 & \mathit{set}(\mathit{delivery}(a,b,p),D) & \leftarrow & \mathit{production_time}(a,p,D_1) \wedge \\
 & & & D \ \mathit{is}\ D_1 - 7 \\
 & & & \\
11 & \mathit{set}(\mathit{payment}(b,a,p),(V,\mathit{imm})) & \leftarrow & \mathit{price}(a,p,V_1) \wedge \\
 & & & V_2 \ \mathit{is}\ V_1 + 0.1 \times V_1 \wedge \\
 & & & V \ \mathit{is}\ V_2 - 0.15 \times V_2
\end{array}$$

This example illustrates the type of process in which we are interested in connection with the problem of contract-based negotiation in multi-agent domains. The arguments exchanged between the agents during this process are concerned primarily with what is acceptable to each of them, and with how to adapt the contract so that these are taken into consideration.

Next we discuss how to instantiate architecture in order to capture this process as a dynamic argument generated by the mechanism described in Chapter 8.

12.2 Instantiating the Architecture in an Agent Scenario

This section describes how the example above can be modelled in our dynamic argumentation framework, derived from the existing implementation of the system. Here we define each component of the architecture as described in Chapter 10, as well as a suitable library of domain-specific revision schemata. The implementation described in Chapter 8 is then used to automatically generate dynamic arguments about the contract being successfully completed, producing a mutually acceptable contract when all attacks generated by the agents via the defined revisions have been appropriately dismissed.

12.2.1 The Theory: The Contract between Producer and Consumer

Let a, b be two autonomous agents—producer and consumer, respectively—negotiating the terms of a contract regarding the supply of a particular product p. This contract is represented by a theory Π in the Horn clause resolution-based language described in Section 12.1.2. This theory contains, in particular, a top-level goal specifying the terms for the successful completion of the contract.

In this example, this is instantiated in Prolog style as follows:

```
main(contract).
provability(contract, solve).
theory(contract, TInit).
```

where `TInit` is the initial list of axioms given in Section 12.1.3, and `solve` is a meta-interpreter for deriving arguments by means of resolution-based proof rules.

The adjustment of Π is then guided by a dynamic argument about the successful completion of the contract—i.e. about $contract\_completion(a, b, p)$. Attacks are generated by a and b; or, in this architecture, by the criticism theories.

12.2.2 The Criticism Theories: Producer and Consumer

One interesting aspect of this example is that it illustrates the uses of two criticism theories in the architecture. Let Π_a and Π_b represent the internal theories of agents a and b, respectively. Within each agent's theory we assume that there is a module that accounts for contract manipulation and negotiation which is based on some provability relation. In particular, let $\Pi_{a_c} \subseteq \Pi_a$, $\Pi_{b_c} \subseteq \Pi_b$ be such subsets of agents a and b's internal theories. For the sake of clarity, we assume that the languages underlying Π, Π_{a_c} and Π_{b_c} are equivalent, meaning that agents have agreed on a set of terms and definitions to be used in contracts. However, we make no further assumptions about agents' theories, in particular about the way beliefs are represented or revised. In fact, the rest of an agent's theory does not even need to be logical, as illustrated in Figure 12.2.

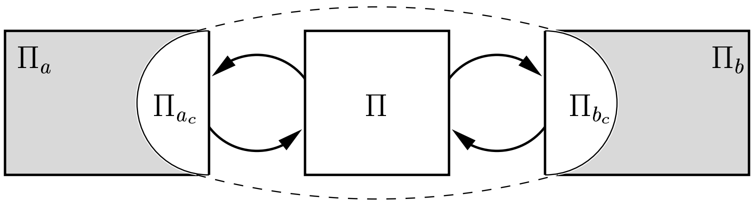


Figure 12.2: Contract-based negotiation in the architecture.

In this process, the role of the criticism theory is played by the sub-theories or modules Π_{a_c} and Π_{b_c}. The question now is how to define these modules.

Devising a formalism for representing autonomous, negotiating agents is outside the scope of this thesis. However, in order to experiment with our ideas and further investigate the relation between negotiation and dynamic argumentation, we have defined a simple representation language for the agents' contract manipulation module so that we can capture the sort of argument given in the previous example. This domain-specific language is essentially used to describe the range of acceptability for contractual terms for the agents, and what possible adjustments and concessions could be made in each case. Below we describe the terms used for specifying this for individual agents of each type—namely, consumers and producers.

Consumers Remember that from a consumer's perspective, the criticism theory allows the generation of arguments for attacking the successful completion of the contract. These attacks, as discussed in Section 12.1.3, are essentially rebuttal attacks of the form:

$$outcome(T, V) : \mathbf{in} \rightsquigarrow outcome(T, V') : \mathbf{in}$$

for different values V and V' such that V' is *better* for b (e.g. cheaper). Arguments in the criticism theory will then give possible acceptable values for contractual terms. The role of consumers is then to make such proposals throughout the negotiation process.

In particular, the expression $acceptable\_value(T, V, NV)$ is used to obtain a new value NV for a contractual term T which is acceptable—or *more* acceptable—to the consumer (as opposed to the current value V). The following clauses for instance define the contract manipulation module for agent b in the example from Section 12.1.3.

1 $acceptable\_value(delivery(X, b, p), V, 7) \leftarrow$
$\quad Diff \;\; is \;\; V - 7 > 0 \wedge$
$\quad reconcile(delivery(X, b, p), V - Diff)$

2 $acceptable\_value(payment(b, X, p), (V, \_), (V - 15\%, \_)) \leftarrow$
$\quad V > 10 \wedge$
$\quad reconcile(payment(b, X, p), (V - 15\%, \_))$

The expression $reconcile(T, V)$ suggests that V should be incorporated to the corresponding contractual clause for T, thus giving a new outcome for T which contradicts the one previously derived.

Similar to the fault tree case, if we consider these expressions to be assumptions, then we can use abduction to select which adjustments are needed in order to reconcile the criticism with the original clauses in the contract. For instance, if we take $\mathcal{L}_{crit}$ to be a Horn clause based language, and $\vdash_{crit}$ to be an abductive provability relation, then the following is an argument supporting the proposal of an earlier delivery for p:

$$\{reconcile(delivery(a, b, p), 14 - 7)\} \vdash_{crit} acceptable(delivery(a, b, p), 14, 7)$$

Note that the description of b's contract manipulation module is characterised within the architecture as follows:

```
crit(b).
provability(b, solve_abd).
theory(b, TCrit_b).
```

where `TCrit_b` is the list of axioms above and `solve_abd` is an abductive meta-interpreter for these axioms.

Recall that mappings between a theory and a criticism theory can be specified as argument schemata—in this case, revisions are about adding a substantiated clause supporting the consumer's proposal of a conflicting outcome.

Domain-specific Schema PROPOSAL OF CONTRADICTORY OUTCOME BY CONSUMER: $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 6 \Rightarrow 9 \Rightarrow 14$

$$\frac{\begin{array}{c} in(outcome(T,V), A, \Pi), \\ add(substantiated\_rule(P)),\ \ revise(\Pi, \{\}, \{P\}, \Pi'), \\ in(outcome(T,V'), A', \Pi') \end{array}}{}$$

$$\begin{array}{ll} \textsf{Properties:} & \left\{ \begin{array}{c} supports(A', outcome(T, V') : \textbf{in},\ \Pi'), \\ satisfiable(\mathbf{B},\ \Pi) \end{array} \right\} \\ \\ \textsf{Conditions:} & \begin{array}{l} outcome(T,V) : \textbf{in} \in \mathcal{G}_A, \\ H, \mathbf{B} \in \mathcal{L}, \\ P = H \leftarrow \mathbf{B}, \\ gen\_argument(b, acceptable\_value(T,V,V'), A_b), \\ reconcile(T,E) \in A_b, \\ set(T, Var) \leftarrow \mathbf{B}' \in A, \\ P' = outcome(T, Var) \leftarrow \mathbf{B}', \\ adjust(P', E, P) \end{array} \end{array}$$

The term $adjust(P', E, P)$ denotes that P is obtained by adjusting the variables in axiom P' according to expression E. Also, remember that b's arguments are generated via the corresponding meta-interpreter `solve_abd`:

```
gen_argument(b, X, A_b) :-
    theory(b, TCrit_b),
    solve_abd(X, A_b, TCrit_b).
```

Producers From the point of view of the producer, the contract manipulation module allows new contract versions to be generated, in which arguments can be derived so as to reestablish the substantiated status of *contract_completion*. These attacks, as discussed in the example 12.1.3, have the general form below:

$$\mathit{contract\_completion} : \mathbf{out} \rightsquigarrow \mathit{contract\_completion} : \mathbf{in}.$$

Producers need to decide whether it is possible or not to reconcile the consumer's proposal (of outcome V' for term T) with the original theory, and if so which other changes should also be carried out. In particular, we use expression $\mathit{is\_acceptable}(T, V)$ not only to verify whether a proposal for a contractual term is acceptable for the producer, but also to (abductively) determine if other adjustments and conditions need to hold for V to become acceptable. The following clauses for instance define the contract manipulation module for agent a in the example from Section 12.1.3.

$$\begin{array}{ll} 1 & \mathit{is\_acceptable}(\mathit{delivery}(a, Y, p), D) \leftarrow \\ & \qquad \mathit{production\_time}(a, p, D) \end{array}$$

$$\begin{array}{ll} 2 & \mathit{is\_acceptable}(\mathit{delivery}(a, Y, p), D) \leftarrow \\ & \qquad \mathit{production\_time}(a, p, D1) \wedge \\ & \qquad D < D1 \wedge \\ & \qquad \mathit{price}(a, p, V) \wedge \\ & \qquad \mathit{reconcile}(\mathit{payment}(Y, a, p), (V + 10\%, \_)) \end{array}$$

$$\begin{array}{ll} 3 & \mathit{is\_acceptable}(\mathit{payment}(Y, a, p), (V, \mathit{std})) \leftarrow \\ & \qquad \mathit{price}(a, p, V) \end{array}$$

$$\begin{array}{ll} 4 & \mathit{is\_acceptable}(\mathit{payment}(Y, a, p), (V, \mathit{std})) \leftarrow \\ & \qquad \mathit{price}(a, p, V1) \wedge \\ & \qquad V < V1 \wedge \\ & \qquad \mathit{reconcile}(\mathit{payment}(Y, a, p), (\_, \mathit{imm})) \end{array}$$

Note that the logical system underlying this criticism theory is equivalent to the one adopted by the consumer b: same language $\mathcal{L}_{crit}$, same abductive provability relation $\vdash_{crit}$, and same set of abducibles—namely, the set of expressions of the form $\mathit{reconcile}(T, E)$. The description of a's contract manipulation module is then characterised within the architecture as follows:

```
crit(a).
provability(a, solve_abd).
theory(a, TCrit_a).
```

where `TCrit_a` is the list of axioms above and `solve_abd` is the abductive meta-interpreter also used by b.

Thus to defend from the sorts of attacks put forward by a consumer, producers need to adapt the clause supporting the contradictory proposal so that it is used to specify the new agreed value for the contractual term in question. In this case changes are about updating this clause by revising its conclusion. Abduction can indicate whether other revisions need to be made—e.g. whether substantiated clauses need to be added so as to reconcile other criticisms with the original axioms.

Domain-specific Schema COUNTER-PROPOSAL FOR CONTRACT COMPLETION BY PRODUCER: $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 11 \Rightarrow 23$

$$\frac{out(contract\_completion(X,Y,Pr),A,\Pi),}{\begin{array}{c} retract(misconclusion(P)),\ add(misconclusion(P')),\ revise(\Pi,\{P\},\{P'\}\cup\mathcal{A}',\Pi'), \\ in(contract\_completion(X,Y,Pr),A',\Pi')\end{array}}$$

$$\text{Properties:} \quad \left\{ \begin{array}{c} supports(A', contract\_completion(X,Y,Pr):\mathbf{in},\Pi'), \\ satisfiable(\mathbf{B},\Pi\cup\mathcal{A}') \end{array} \right\}$$

$$\begin{array}{ll} \text{Conditions:} & contract\_completion(X,Y,Pr):\mathbf{out}\in\mathcal{G}_A, \\ & contract\_completion(X,Y,Pr)\leftarrow\mathbf{B}\in\Pi, \\ & outcome(T,V'):\mathbf{in},\, outcome(T,V):\mathbf{out}\in\mathcal{G}_A, \\ & gen\_argument(a, is\_acceptable(T,V'),A_a), \\ & P = outcome(T,E)\leftarrow\mathbf{B}'\in A, \\ & H' = set(T,Var)\in\mathcal{L}, \\ & P' = H'\leftarrow\mathbf{B}', \\ & \mathcal{A}' = \{P_i \mid reconcile(T_i,E_i)\in A_a, \\ & \qquad\qquad P_i' = set(T_i,Var_i)\leftarrow\mathbf{B}_i\in_R\Pi\cup\{P'\}, \\ & \qquad\qquad adjust(P_i',E_i,P_i)\} \end{array}$$

Note that $\in_R$ corresponds to the usual set-membership $\in$ operator restricted to a recency ordering R in the set (list) of axioms. That is, $X \in_R S$ selects the most recent element in S that unifies with X.

12.2.3 The Control Module

With respect to preferences, priority measures could be defined from two perspectives. On one hand, agents can have preferences based on utility functions, values (Fox and Parsons 1998) or explicitly represented by means of a special meta-predicate (Sierra et al. 1997b). These are specified within the agents theories and can be used to prioritise criticisms.

On the other hand, we can also have criteria for preferring one clause over another in a contract (e.g. the one introduced most recently). In any case these notions of prioritisation conform to the architecture in Section 10.3, but in this example we only implement the latter. Because we are adopting a fairly simple specification language for the agents' contract manipulation module, we assume in this case that arguments generated in the criticism theories have equal weight.

Similarly to `solve_filter` in the fault tree example,[3] this sort of recency-based prioritisation concerns the generation of individual arguments. Assuming an ordering of recency among axioms, a layered meta-interpreter can then prioritise consequences in the theory, blocking any derivation which is based on an *earlier* definition of a clause. In terms of our architecture, this is represented as follows:

```
filter(solve, solve_recency).
```

states that *solve_recency* considers the underlying recency ordering for filtering arguments constructed by *solve*, allowing only those based on the most recent definitions of a predicate to be advanced. The predicate *solve_recency* (rather than *solve*) is used to generate prioritised arguments from the contract:

```
gen_argument(contract, X, A) :-
    theory(contract, T),
    solve_recency(solve(X, A, T)).
```

Thus this example populates the bottom-left box for argument prioritisation in Figure 11.6.

It is important to note that contracts are defined and altered in terms of agents' internal theories and target requirements. In this sense, they are similar to **KQML** messages, where the use of performatives is described in terms of the agents' cognitive states (Labrou and Finin 1994). However, differently from **KQML**, contracts are structures—or objects—which are manipulated by agents and used to test whether certain properties are satisfied or not.

By instantiating the architecture in this way, the sorts of dynamic argument mechanisms

[3] See Section 11.1.3.

discussed in Part II can then be applied in order to obtain, for instance, the final contract in the example above.

12.3 A Dynamic Argument for Contract Formation

A couple of remarks in comparison to the previous adaptation of the architecture in Chapter 11 should be made at this point. First, in this case we have all the schemata being instantiated by external sources of criticism. No schema is dependent on the theory only.

Second, each agent has access to its library of schemata only. In fact, Φ could be represented as follows:

$$\Phi = \Phi_{\text{PRODUCER}} \cup \Phi_{\text{CONSUMER}}, \text{ where}$$

$$\begin{aligned}\Phi_{\text{PRODUCER}} &= \{\textsc{counter-proposal for contract completion}\}\\ \Phi_{\text{CONSUMER}} &= \{\textsc{proposal of contradictory outcome}\}\end{aligned}$$

Recall that this is similar to the sort of discussion about disputes and argument games in Section 9.1.1 between opponents and proponents.

Now that the architecture and a catalogue Φ of attack-based revision schemata have been specified, the system in Section 8.1.2 can be used to generate the dynamic argument in Section 12.1.3 in an automated form.

So let `TInit` be the initial theory as described in Section 12.2.1, and Φ as defined above. Below we present a dynamic argumentation process about $\mathit{contract\_completion}(a, b, p)$ as generated by our implementation. According to Definition 4.7, the first argument to be advanced is a justification supporting the main claim that the contract will be successfully completed.

Argument A_0 is a justification for $\mathit{contract\_completion}(a, b, p)$ advanced by a, based on $\mathit{outcome}(\mathit{delivery}(a, b, p), 14)$ and $\mathit{outcome}(\mathit{payment}(b, a, p), (10, \mathit{std}))$.

Revision ϕ_1 is obtained from schema PROPOSAL OF CONTRADICTORY OUTCOME, and from the argument for $\mathit{acceptable\_value}(\mathit{delivery}(a, b, p), 14, 7)$ in b's criticism module which is based on $\mathit{reconcile}(\mathit{delivery}(a, b, p), 14 - 7)$.

Argument A_1 is a justification for $outcome(delivery(a,b,p),7)$ advanced by b.

Revision ϕ_2 adapts the contract through schema COUNTER-PROPOSAL FOR CONTRACT COMPLETION based on the a's argument for $is\_acceptable(delivery(a,b,p),7)$ generated from $reconcile(payment(b,a,p),(10+10\%,\_))$.

Argument A_2 is a justification for $contract\_completion(a,b,p)$ advanced by a, based on $outcome(delivery(a,b,p),7)$ and $outcome(payment(b,a,p),(11,std))$.

Revision ϕ_3 is obtained from schema PROPOSAL OF CONTRADICTORY OUTCOME, and from the argument for $acceptable\_value(payment(b,a,p),(11,\_),(9.35,\_))$ in b's criticism module which is based on $reconcile(payment(b,a,p),(11-15\%,\_))$.

Argument A_3 is a justification for $outcome(payment(b,a,p),(9.35,std))$ advanced by b.

Revision ϕ_4 adapts the contract through schema COUNTER-PROPOSAL FOR CONTRACT COMPLETION based on a's argument for $is\_acceptable(payment(b,a,p),(9.35,imm))$ generated from $reconcile(payment(b,a,p),(\_,imm))$.

Argument A_4 is a justification for $contract\_completion(a,b,p)$ advanced by a, based on $outcome(delivery(a,b,p),7)$ and $outcome(payment(b,a,p),(9.35,imm))$.

12.4 Issues Raised by this Example

The type of application proposed in this chapter relates a style of reasoning that is common in multi-agent scenarios to the sort of argument dynamics that we have explored in this thesis. What we have done is to reduce a problem of contract negotiation to the generation of a dynamic argument in which criticism theories are components of the agents.

Note that in the case of the fault tree obtaining the criticism theory was actually a straightforward task, as fault trees are standard practice in the safety domain and thus widely available. In the case of agent-based applications, however, we felt the need of devising a simple specification language for individual agents in order to illustrate our ideas. This language, though, is quite specific to this application and more thought should be given on how it relates to existing proposals for agent formalisms and architectures. As a matter of fact, our implementation is essentially sequential, and no form of agent communication or interaction is prescribed.

This opens interesting possibilities for research, in which for instance different agents can adopt different proof strategies and act in parallel in order to explore possible attacks in different ways.

Part IV

Conclusions and Discussion

Chapter 13

Contributions

The overall goal of the research described in this thesis is to explore the role of formal argumentation systems in the area of knowledge engineering. As stated in Chapter 1, our work has been guided by two leading, general questions, namely:

- How can knowledge engineers benefit from argumentation-based approaches to knowledge representation and reasoning?
- How can we improve the methodology for building systems for supporting such tasks?

Regarding the first question, there is no doubt that the general paradigm of argument-based reasoning has proved applicable to a variety of tasks, especially those involving inconsistent and incomplete information. Chapter 3 gave a detailed analysis of the sorts of problems that can be tackled by this means, and throughout the thesis we have presented other applications and examples of uses of argumentation.

This links to the second question above. Again, as discussed in Chapter 1, we were motivated by the need to take more complex arguments into account in a systematic way. The way we did that was first by identifying a particular type of argumentation process which could allow for different types or argument to be represented, and then by constructing an abstract formal framework for capturing those processes and allowing for domain-specific applications to be instantiated from this framework.

We took a pragmatic approach to formal argumentation and to the generation of dynamic arguments, which was essentially based on catalogues of (domain-specific) schemata for generating arguments and attacks. The development of our framework was steered by the questions stated in Chapter 2, which are reproduced below before we summarise the main technical contributions of this thesis.

- Which concepts are involved in argument dynamics, and which of these would be interesting to formalise? Can these be defined in a general way or are they (or some of them) domain-specific?
- How to represent and generate an argument? What types of arguments are important to be represented?
- How do arguments relate to each other and what types of relationships can be defined between arguments?
- Where do attacks come from?
- What mechanisms are used to prioritise arguments, and how can contextual (domain) information be incorporated into such mechanisms?
- When do dynamic arguments terminate?

Exploring these questions produced the following main contributions:

A Problem-oriented Classification of Argument-based Research.

Chapter 3 characterised the types of problems in knowledge engineering that can be addressed by argumentation. These problems range from non-monotonic and defeasible reasoning to decision making under uncertainty, and from negotiation to design.

A Formalisation of Dynamic Argumentation.

First of all, in Chapter 2 we have characterised exactly what we mean by argument dynamics, and how these compare to other approaches to formal argumentation. Dynamic arguments are based on the generation of processes of argument exchange where the knowledge base from which arguments are derived is dynamic, i.e. it can be changed during the process itself. Arguments are essentially proofs given via an underlying provability relation from this knowledge base. The concept of *dynamic argumentation* is novel in itself, although related to what is sometimes referred to as procedural models of argumentation (see Section 3.1). In this sense, we are concerned with showing that this concept—*dynamic argument*—is useful and usable.

A novel formalisation of dynamic argumentation was given in Chapter 4, based on attack-based revisions used for revising a knowledge base so as to generate a particular attack. In connection with this formalisation, we have taken a novel view of theories (i.e. knowledge bases) as arguments, and dynamic argumentation as a process for theory transformation guided by attacks.

A Precise Characterisation of Attacks.

Chapter 6 presented a precise, well-founded characterisation of attacks, and of possible contradictions in arguments.

An Analysis of the Relation between Argumentation and TMS.

In our representation we were able to effectively compare the functions of truth maintenance systems and argumentation. We also have shown that it is possible to use a truth maintenance system mechanism for implementing and maintaining the structure of claims which can be attacked during a dynamic argument (Section 6.4).

Implementation of a Mechanism for Generating Dynamic Arguments.

We have defined a general mechanism for argument generation from the perspective of transformation of theories. Every step of argument is represented by a general attack-relation between the original theory and a revised theory (Section 7.1), and the theory may be revised until no more attacks can be generated.

Two dynamic argumentation systems have been implemented from this mechanism by considering different possibilities for attack generation based on the classification of argument schemata presented in Chapter 7:

- one system generates attacks interactively by querying for appropriate information each time it reaches selection points in this classification (Section 8.1.1)
- the other generates attacks automatically from a catalogue of argument schemata previously obtained from the classification (Section 8.1.2).

A Method of Specifying Argument Schemata.

We have devised a formal classification of argument schemata (Chapter 7), which is essentially an abstract top-down approach to capture argument structure, and obtaining argument schemata for generating attacks. This classification, which is based on an underlying logic programming based theory, was defined in terms of the general types of attack (Chapter 6), and inspired by standard argumentative structures from studies in the fields of informal logic and argumentation theory (Chapter 5).

A crucial element in this approach is the notion of properties associated to each rewrite. Properties accumulate as we go down in the hierarchy of rewrites defining possible argument schemata, and they give a large flexibility to our framework and to designers of catalogues of (domain-specific schemata). Chapter 8 described a way to define a catalogue of domain-specific schemata from this classification. Section 8.3 in particular discussed the use of properties in great detail. More domain-specific schemata were defined later in the applications in Part III.

We also have shown that this classification is complete up to a certain point. Other uses of our proposed methodology are also supported, such as communication and retrospective analysis of arguments (Chapter 9).

A General Architecture for Dynamic Argumentation Systems

We have devised an architecture that elaborates on the mechanisms for dynamic argument generation so as to allow for external instantiation of revision schemata, and for attacks based on priorities and preferences (Chapter 10).

Instantiation of Applications.

> Because we cannot formally prove the correctness of our model, applications are necessary to judge the relevance of the theory. We have tackled two distinct problems by adapting the architecture to domain-specific scenarios. This is done by maintaining the overall, generic mechanism for generating dynamic arguments, but allowing for domain-specific adaptation of the components of the architecture, and of the catalogue of revision schemata (Chapters 11 and 12).

In summary, we have given details of dynamic argumentation generators and of an architecture of dynamic argumentation systems. Also, we have presented an analysis of various examples and of different problems with similar grounds in argumentation, based on the same architecture. Linked to some of these examples we have presented a restricted analysis of prioritisation.

Chapter 14

What Next to Do?

Throughout this thesis we have touched upon many related issues of interest, unsolved problems and possible avenues for future work. In this chapter we look carefully at these issues, elaborating upon the limitations of our approach as well as the limitations of this document, and expanding some of the topics which we believe deserve—or require—further exploration.

14.1 The Fine Print

Some of the shortcomings of this thesis are intrinsic to our research given the scope of our problem and the applicability of our formalism.

The most obvious limitation is that innumerable forms and types of arguments that cannot be captured by our model. This is fine, though, because we do not aim at formalising argumentation. Our perspective of the problem is that argumentation can be used to model particular styles of reasoning—and not that formal styles of reasoning can be used to model argumentation. A similar distinction is made by Reed (1997).

In fact, the sorts of problems that can be captured as dynamic arguments are those that can be idealised as operations and transformations over sets of axioms which can be guided by arguments. This is a very abstract problem description, and similarly the solution we provided was as abstract as possible. Essentially, we assume that theories can be expressed as sets of axioms, and that suitable predefined catalogues of revision schemata are available from start.

By taking this view we have described a way of automating the generation of dynamic arguments. Remember from Chapter 2 that one of the reasons why we believe it is important to formalise and automate argumentation processes is because argument-based methodologies should be supported by (semi-) automated tools, which can both guide knowledge engineers in developing knowledge bases that derive the intended consequences, and also support designers of argument systems in investigating properties and effects of certain attacks and revisions in a domain. Also, automated argumentation systems can be used by artificial agents that want to employ this solution to solve particular problems.

However, such a level of abstraction has made it very hard to demonstrate general formal properties of the framework and to make stronger claims about the types of argument processes that are generated. It is difficult to prove for instance whether arguments will terminate just by looking at a set of possible (unconstrained) revision operations. Other questions may arise such as how much do we have to know in advance in order to define a suitable catalogue.

We do not claim that realistically these mechanisms can be used in their target domains of application as they are. Although the classification provides a systematic way to build argument schemata, knowledge of formal methods and domain-specific engineering work are still needed to be put in the task (for instance, on deciding the terms and expressions to be used in each particular domain). This is unlikely to be a trivial task, and support tools still need to be provided.

Related to this we have presented no definition of what constitutes a good dynamic argument. Would it be possible to find an appropriate metric so as to evaluate generated arguments automatically? One possibility for evaluation is to have human users to analyse the plausibility of the arguments. But automatic evaluation using some defined metric could allow for the analysis and comparison of strategies and schemata for argument generation in terms of the *quality* of the dynamic argumentation processes, and of the final, resulting theories.

Note that the design of our generic framework as well as the classification of schemata have been informed by many ideas which were transferred from the roots of argumentation theory. And while many concepts underlying our classification of schemata were built assuming a logic programming representation, the core concepts of dynamic argumentation—those defined in Chapter 4—are logic-independent and should easily adapt to different logics. This brings up another question, though—namely of how easy would it be to transfer these logic-specific concepts across other choices of logical representation.

In summary, our approach to argumentation is different from the conventional static approaches in the literature. By taking this view we have broadened the scope of application of argumentation in knowledge engineering contexts, but we have also made it harder to recognise suitable problems in which to apply this technique. So, how easy is it really to decide whether some problem can be characterised as dynamic argumentation, and what would be a suitable catalogue of revisions in a target domain? To which degree does our formalisation, including the classification, prescribe how to tackle a particular problem?

Maybe some of these questions could be further elucidated if other limitations of this thesis had been addressed—in this case, limitations which stem from the time-limit of our research project, such as:

- A complete analysis of the use of priorities and preferences in argument generation was not the focus of this thesis, although these can play an important role in the generation of arguments. We have only examined this linked to the applications of the architecture, but not in a deep way.
- Linked to this, an analysis of selection strategies might shed more light on the actual generation of arguments, in particular to the selection of arguments and how this could affect the process, generating more efficient arguments.

So next we describe a research wish list, which we believe would provide clearer answers regarding the usefulness and usability of dynamic argumentation.

14.2 A Wish List

Our work on arguments and dynamics opens up a number of issues and areas for future exploration, some of these are discussed below.

14.2.1 Analysis of Priorities and Preferences

Section 11.4 has investigated two ways in which our architecture allows for prioritisation of arguments: one involves the direct comparison of arguments in the theory; the other is about prioritising individual arguments both in the theory and in the criticism theory according to some measure of quality.

This is as far as our analysis has gone, apart from providing some examples in connection with the domains of application given in Part III. A deeper analysis of such prioritisation techniques, especially in the context of Figure 11.6, is fundamental for a deeper understanding of dynamic argumentation.

14.2.2 Strategies for Selecting Arguments

This follows as a consequence of the work in prioritisation, and can also shed light on aspects of efficient argument generation in connection with procedural and heuristic layers of argument systems discussed in Chapter 3.

A trivial strategy for selecting the next argument to be advanced is simply to advance the first argument that is generated. This is particularly satisfactory if we are able to explore the whole search space of possible dynamic arguments, as described in Section 8.3. However, given that priority measures and precedence orderings may exist, one might use this information to decide upon the *best* possibility, in what is essentially a *generate-and-test* approach.

Determining which argument to advance is a actually a different task from that of selecting which claim to attack (see Chapter 6). In our approach, the latter is equivalent to selecting which instance of attack-based schema to apply in the next step. But this task too can be guided by some sort of prioritisation, e.g. an explicit partial ordering on the schemata in the catalogue Φ.

In any case, rather than adopting a *generate-all-and-select-best* strategy, it would be interesting to analyse whether we can combine the tasks of argument generation with that of selection so that at each step in the argument process one possible argument is given.

14.2.3 Automated Evaluation of Dynamic Arguments

While prioritisation is concerned with the quality of arguments within the theory, evaluation of dynamic arguments would probably take other criteria into consideration, maybe related to the quality of the final theory and to other concepts in the heuristic layer, such as efficiency and persuasiveness.

Coming up with some metric for this is a difficult task, and would probably have to be informed by analysis and experiments with human users.

14.2.4 Formal Analysis of the Framework

As we have argued before, proving generic formal results about our framework—e.g. termination—is a very difficult task. This may be made easier if we assume certain properties about the sorts of revisions allowed, maybe even in connection with templates and libraries of domain-specific revisions.

There is much scope for research in this area. Other areas of study may provide clues and results that could be applicable to our approach, such as research in term rewriting systems (given that our formalism can be defined as such).

14.2.5 Adopting Different Underlying Logics

In many cases, general logic programs may not be the best choice of representation for a theory. Although we do not commit to a particular logic until later in the thesis, much of what makes it applicable to domains is dependent on this language.

This opens an interesting possibility for research, namely whether we can identify a precise notion of attack and describe a similar classification of schemata based on a different representation language, and to what extent the logic-specific elements in this thesis could be reused.

14.2.6 Editors and Tools Supporting the Design of Argument Systems

This is perhaps one of the main areas for improvement in our work. Providing tools such as schemata editors, with support for the adaptation of properties and for realistic use of argumentation mechanisms in domains of applications. Support for generic as well as domain-specific argument schemata could also improve usability of this technique.

14.2.7 Testing Properties

Maybe in relation to the editors and tools mentioned above, it should be possible to make a better use of the properties in argument schemata. One possibility, for instance, is to allow users/designers to introduce extra properties, disregard others, or yet define new ones, also testing the consequences of these choices in relation to the outcome of argumentation processes.

14.2.8 Applications to Domains

This involves much more than just adapting the architecture so as to generate dynamic arguments in particular domains. Research in this area demands identification of suitable target domains, and a serious analysis of requirements of users/designers in these domains.

For instance, we have just briefly touched issues like communication of dynamic arguments, but these are likely to involve different aspects for different domains.

14.2.9 Application in Real Multi-Agent Scenarios

Realistic multi-agent applications are characterised by aspects and features such as communication languages, interaction and parallel processing. It would be interesting to examine precisely how our mechanisms could fit within such scenarios, and also how dynamic argumentation relates to existing languages for negotiation.

Appendix A

Basic Syntax: Logic Programming

This appendix gives the basic syntax of logic programming theory used in this thesis. For a complete account of logic programming theory, see (Lloyd 1987; Apt 1995).

Syntax. The syntax of logic programs is based on the usual concepts of *terms*, *atoms* and *well-formed formulae* from first order languages. A *literal* is an atom (positive literal) or the negation of an atom (negative literal).

A *program clause*, or *definite clause*, is a clause of the form:

$$H \leftarrow B_1 \wedge ... \wedge B_n \tag{A.1}$$

where $H, B_1, ..., B_n$ are positive literals. H and $B_1 \wedge ... \wedge B_n$ are called the *head* and *body* of the clause, respectively. A *goal clause* is a clause of the form:

$$\leftarrow B_1 \wedge ... \wedge B_n \tag{A.2}$$

A *Horn clause* is either a program clause or a goal clause.

General clauses are essentially program clauses which allow negative literals to occur in the body of the clause. A general clause has the form:

$$H \leftarrow B_1 \wedge ... \wedge B_m \wedge not\ B_{m+1} \wedge ... \wedge not\ B_n \tag{A.3}$$

where *not* stands for negation as failure, and each B_i is a positive literal.

A *definite logic program* is a finite set of definite clauses. A *general logic program* is a finite set of general clauses. General logic programs are sometimes called *normal logic programs*.

The body of a clause can be denoted by a single bold letter $\mathbf{B}$ representing a conjunction of literals. Individual literals are denoted by the (possibly indexed) letter B.

Substitution and Unification. Unification gives means to compute values in logic programs. Variables V_i can be associated with terms T_i via substitutions of the form $\sigma = [V_i/T_i]$. Unification is concerned with finding a substitution which can be applied to two expressions and make them syntactically identical.

The *most general unifier* is the simplest substitution that unify two expressions. A substitution σ that represents the most general unifier between two expressions sentences is denoted by mgu.

Also, we use the expression $F_{[T_1/T_2]}$ to denote the formula obtained from a formula F by replacing every occurrence of a term T_1 by term T_2.

Appendix B

Basic Notation: Trees and Graphs

This appendix gives the basic notation used in this thesis for representing trees and directed graphs. Trees are mainly used to represent arguments, whereas more generic directed graphs are used to express dependencies between claims in an argument.

B.1 Directed Graphs

A graph $\mathcal{G}$ is a pair $(\mathcal{V}, \mathcal{E})$ of vertices (or nodes) and edges (or links), respectively. The set of vertices of a graph $\mathcal{G}$ can be referred to as $\mathcal{V}(\mathcal{G})$, and the set of edges as $\mathcal{E}(\mathcal{G})$.

A graph is said to be directed if the edges have an orientation. An edge $\psi \hookrightarrow \varphi$ is said to initiate at node ψ and terminate at node φ.

For the type of application in this thesis, it is useful to differentiate between two types of edges, namely those initiating at a supporting node, and those initiating at a conflicting node:

- if ψ supports φ then $\psi \hookrightarrow \varphi$ is said to be a supporting link;
- if ψ conflicts with φ then $\psi \hookrightarrow \varphi$ is said to be a conflicting link.

Alternatively, edges can be represented diagrammatically as follows, where dotted lines denote a conflicting link.

$$\begin{tikzcd} \varphi \\ \psi \arrow[u] \end{tikzcd} \qquad\qquad \begin{tikzcd} \varphi \\ \psi \arrow[u, dotted] \end{tikzcd}$$

Moreover, we are interested in directed graphs with labelling functions for expressing the support status of each node. The labelling function associated with a graph $\mathcal{G}$ is denoted by:

$$label_{\mathcal{G}} : \mathcal{V}(\mathcal{G}) \mapsto \{\mathbf{in}, \mathbf{out}\}$$

The status $label_{\mathcal{G}}(\psi)$ associated with each node in $\mathcal{G}$ may be determined either by an external factor (e.g. given by some other labelling function), or by means of an operator $\bigwedge$ that derives this value from the status of other nodes in the graph.

Essentially, $\bigwedge \psi$ gives the status **in** only if all supporting nodes of ψ in $\mathcal{G}$ are **in**, and all conflicting nodes of ψ in $\mathcal{G}$ are **out**; otherwise, $\bigwedge \psi$ derives **out**. This operator can be applied given that other nodes have their labels already defined, thus forming the base step of the definition.

B.2 Argument Trees

A tree is essentially an acyclic, connected graph. In particular, here we use rooted trees for representing arguments derived from a provability relation, such that lower nodes support the conclusion above. In this representation, nodes in an argument tree have at most one parent.

Each premise P of the form $H \leftarrow B_1 \wedge \ ... \ \wedge B_N$ in an argument defines a tree with root H and subtrees $A_{B_1}, ..., A_{B_N}$ corresponding to the arguments supporting sentences $B_1, ..., B_N$, respectively. Such trees are denoted here by the expression:

$$tree(H,\ P,\ \{A_{B_1}, ..., A_{B_N}\}).$$

Appendix C

Harnessing Argument Rewriting

This appendix contains the possible schemata for argument revision as represented in Figure 7.2 and produced by the rewriting system of Section 7.3. Altogether they indicate the general format of attack, with properties accumulated down the classification and conditions that give the structure of the premises to be added and retracted (including $\in$-conditions from the previous rewrites that can be used select appropriate instances of an attack).

C.1 Trivial Revisions

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 6 \Rightarrow 8$

$$\begin{gathered} in(X, A, \Pi), \\ trivial(\{\}, \{\}),\ \ revise(\Pi, \{\}, \{\}, \Pi'), \\ in(Y, A', \Pi') \end{gathered}$$

$$\text{Properties:} \quad \left\{ \begin{gathered} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', Y : \mathbf{in}, \Pi') \end{gathered} \right\}$$

$$\begin{array}{ll} \text{Conditions:} & X : \mathbf{in} \in \mathcal{G}_A, \\ & Y \in conflict(X) \end{array}$$

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 8$

$$\begin{gathered} out(X, A, \Pi), \\ trivial(\{\}, \{\}),\ \ revise(\Pi, \{\}, \{\}, \Pi'), \\ in(X, A', \Pi') \end{gathered}$$

$$\text{Properties:} \quad \left\{ \begin{gathered} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', X : \mathbf{in}, \Pi') \end{gathered} \right\}$$

$$\text{Conditions:} \quad X : \mathbf{out} \in \mathcal{G}_A$$

C.2 Elementary Revisions for Adding an Argument

C.2.1 Adding a Fact

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 6 \Rightarrow 9 \Rightarrow 13$

$$\frac{in(X, A, \Pi),}{\begin{array}{c} add(fact(P)),\ \ revise(\Pi, \{\}, \{P\}, \Pi'), \\ in(Y, A', \Pi') \end{array}}$$

$$\text{Properties:} \quad \left\{ \begin{array}{c} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', Y : \mathbf{in}, \Pi'), \\ unify(Y, H) \end{array} \right\}$$

$$\text{Conditions:} \quad \begin{array}{l} X : \mathbf{in} \in \mathcal{G}_A, \\ Y \in conflict(X), \\ H \in \mathcal{L}, \\ P = H \leftarrow true \end{array}$$

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 9 \Rightarrow 13$

$$\frac{out(X, A, \Pi),}{\begin{array}{c} add(fact(P)),\ \ revise(\Pi, \{\}, \{P\}, \Pi'), \\ in(X, A', \Pi') \end{array}}$$

$$\text{Properties:} \quad \left\{ \begin{array}{c} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', X : \mathbf{in}, \Pi'), \\ unify(X, H) \end{array} \right\}$$

$$\text{Conditions:} \quad \begin{array}{l} X : \mathbf{out} \in \mathcal{G}_A, \\ H \in \mathcal{L}, \\ P = H \leftarrow true \end{array}$$

C.2.2 Adding a Substantiated Rule

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 6 \Rightarrow 9 \Rightarrow 14$

$$\begin{array}{c} in(X, A, \Pi), \\ add(substantiated\_rule(P)),\ revise(\Pi, \{\}, \{P\}, \Pi'), \\ in(Y, A', \Pi') \end{array}$$

$$\text{Properties:} \quad \left\{ \begin{array}{c} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', Y : \mathbf{in}, \Pi'), \\ unify(Y, H), \\ satisfiable(\mathbf{B}\sigma, \Pi) \end{array} \right\}$$

$$\text{Conditions:} \quad \begin{array}{l} X : \mathbf{in} \in \mathcal{G}_A, \\ Y \in conflict(X), \\ H, \mathbf{B} \in \mathcal{L}, \\ P = H \leftarrow \mathbf{B}, \\ \sigma = mgu(Y, H) \end{array}$$

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 9 \Rightarrow 14$

$$\begin{array}{c} out(X, A, \Pi), \\ add(substantiated\_rule(P)),\ revise(\Pi, \{\}, \{P\}, \Pi'), \\ in(X, A', \Pi') \end{array}$$

$$\text{Properties:} \quad \left\{ \begin{array}{c} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', X : \mathbf{in}, \Pi'), \\ unify(X, H), \\ satisfiable(\mathbf{B}\sigma, \Pi) \end{array} \right\}$$

$$\text{Conditions:} \quad \begin{array}{l} X : \mathbf{out} \in \mathcal{G}_A, \\ H, \mathbf{B} \in \mathcal{L}, \\ P = H \leftarrow \mathbf{B}, \\ \sigma = mgu(X, H) \end{array}$$

C.2.3 Adding a Burden Shift Rule

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 6 \Rightarrow 9 \Rightarrow 15$

$$\begin{gathered} in(X, A, \Pi), \\ add(burden\_shift\_rule(P)),\ revise(\Pi, \{\}, \{P\}, \Pi'), \\ in(Y, A', \Pi') \end{gathered}$$

Properties: $\left\{ \begin{array}{c} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', Y : \mathbf{in}, \Pi'), \\ unify(Y, H), \\ \neg satisfiable(B\sigma, \Pi) \end{array} \right\}$

Conditions: $\begin{array}{l} X : \mathbf{in} \in \mathcal{G}_A, \\ Y \in conflict(X), \\ H, B \in \mathcal{L}, \\ P = H \leftarrow not\ B, \\ \sigma = mgu(Y, H) \end{array}$

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 9 \Rightarrow 15$

$$\begin{gathered} out(X, A, \Pi), \\ add(burden\_shift\_rule(P)),\ revise(\Pi, \{\}, \{P\}, \Pi'), \\ in(X, A', \Pi') \end{gathered}$$

Properties: $\left\{ \begin{array}{c} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', X : \mathbf{in}, \Pi'), \\ unify(X, H), \\ \neg satisfiable(B\sigma, \Pi) \end{array} \right\}$

Conditions: $\begin{array}{l} X : \mathbf{out} \in \mathcal{G}_A, \\ H, B \in \mathcal{L}, \\ P = H \leftarrow not\ B, \\ \sigma = mgu(X, H) \end{array}$

C.3 Updating Revisions for Adding an Argument

C.3.1 Removing Irrelevance in a Rule

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 6 \Rightarrow 11 \Rightarrow 19$

$$\begin{gathered} in(X, A, \Pi), \\ retract(irrelevance(P)),\ \ add(irrelevance(P')),\ \ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ in(Y, A', \Pi') \end{gathered}$$

Properties:
$$\left\{ \begin{gathered} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', Y : \mathbf{in}, \Pi'), \\ unify(Y, H), \\ satisfiable((B_1 \wedge ... \wedge B_{i-1} \wedge B_{i+1} \wedge ... \wedge B_m)\sigma, \Pi), \\ \neg satisfiable(B_i\sigma, \Pi) \end{gathered} \right\}$$

Conditions:
$X : \mathbf{in} \in \mathcal{G}_A$,
$Y \in conflict(X)$,
$H \leftarrow B_1 \wedge ... \wedge B_m \in \Pi$,
$P = H \leftarrow B_1 \wedge ... \wedge B_m$,
$B_i \in \{B_1, ..., B_m\}$,
$P' = H \leftarrow B_1 \wedge ... \wedge B_{i-1} \wedge B_{i+1} \wedge ... \wedge B_m$,
$\sigma = mgu(Y, H)$

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 11 \Rightarrow 19$

$$\begin{gathered} out(X, A, \Pi), \\ retract(irrelevance(P)),\ \ add(irrelevance(P')),\ \ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ in(X, A', \Pi') \end{gathered}$$

Properties:
$$\left\{ \begin{gathered} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', X : \mathbf{in}, \Pi'), \\ unify(X, H), \\ satisfiable((B_1 \wedge ... \wedge B_{i-1} \wedge B_{i+1} \wedge ... \wedge B_m)\sigma, \Pi), \\ \neg satisfiable(B_i\sigma, \Pi) \end{gathered} \right\}$$

Conditions:
$X : \mathbf{out} \in \mathcal{G}_A$,
$H \leftarrow B_1 \wedge ... \wedge B_m \in \Pi$,
$P = H \leftarrow B_1 \wedge ... \wedge B_m$,
$B_i \in \{B_1, ..., B_m\}$,
$P' = H \leftarrow B_1 \wedge ... \wedge B_{i-1} \wedge B_{i+1} \wedge ... \wedge B_m$,
$\sigma = mgu(X, H)$

C.3.2 Generalising a Rule

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 6 \Rightarrow 11 \Rightarrow 21$

$$\begin{gathered} in(X, A, \Pi), \\ retract(generalisation(P)),\ \ add(generalisation(P')),\ \ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ in(Y, A', \Pi') \end{gathered}$$

$$\text{Properties:} \quad \left\{ \begin{gathered} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', Y : \mathbf{in}, \Pi'), \\ unify(Y, H\sigma'), \\ satisfiable((\mathbf{B}\sigma')\sigma, \Pi), \\ ground(P, \Pi) \subset ground(P', \Pi) \end{gathered} \right\}$$

$$\text{Conditions:} \quad \begin{aligned} & X : \mathbf{in} \in \mathcal{G}_A, \\ & Y \in conflict(X), \\ & H \leftarrow \mathbf{B} \in \Pi, \\ & P = H \leftarrow \mathbf{B}, \\ & \sigma' \in inverse\_subst, \\ & P' = (H \leftarrow \mathbf{B})\sigma', \\ & \sigma = mgu(Y, H\sigma') \end{aligned}$$

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 11 \Rightarrow 21$

$$\begin{gathered} out(X, A, \Pi), \\ retract(generalisation(P)),\ \ add(generalisation(P')),\ \ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ in(X, A', \Pi') \end{gathered}$$

$$\text{Properties:} \quad \left\{ \begin{gathered} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', X : \mathbf{in}, \Pi'), \\ unify(X, H\sigma'), \\ satisfiable((\mathbf{B}\sigma')\sigma, \Pi), \\ ground(P, \Pi) \subset ground(P', \Pi) \end{gathered} \right\}$$

$$\text{Conditions:} \quad \begin{aligned} & X : \mathbf{out} \in \mathcal{G}_A, \\ & H \leftarrow \mathbf{B} \in \Pi, \\ & P = H \leftarrow \mathbf{B}, \\ & \sigma' \in inverse\_subst, \\ & P' = (H \leftarrow \mathbf{B})\sigma', \\ & \sigma = mgu(X, H\sigma') \end{aligned}$$

Notice that in these schemata the property $ground(P, \Pi) \subset ground(P\sigma', \Pi)$ holds by construction because σ' is an inverse substitution.

C.3.3 Revising the Consequent of a Rule

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 6 \Rightarrow 11 \Rightarrow 23$

$$\begin{gathered} \mathit{in}(X, A, \Pi), \\ \mathit{retract}(\mathit{misconclusion}(P)),\ \mathit{add}(\mathit{misconclusion}(P')),\ \mathit{revise}(\Pi, \{P\}, \{P'\}, \Pi'), \\ \mathit{in}(Y, A', \Pi') \end{gathered}$$

Properties: $\left\{ \begin{array}{c} \mathit{attacks}(A', A), \\ \mathit{consistent}(\Pi'), \\ \mathit{supports}(A', Y : \mathbf{in}, \Pi'), \\ \mathit{unify}(Y, H'), \\ \mathit{satisfiable}(\mathbf{B}\sigma, \Pi) \end{array} \right\}$

Conditions: $\begin{array}{l} X : \mathbf{in} \in \mathcal{G}_A, \\ Y \in \mathit{conflict}(X), \\ H \leftarrow \mathbf{B} \in \Pi, \\ P = H \leftarrow \mathbf{B}, \\ H' \in \mathcal{L}, \\ P' = H' \leftarrow \mathbf{B}, \\ \sigma = \mathit{mgu}(Y, H') \end{array}$

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 11 \Rightarrow 23$

$$\begin{gathered} \mathit{out}(X, A, \Pi), \\ \mathit{retract}(\mathit{misconclusion}(P)),\ \mathit{add}(\mathit{misconclusion}(P')),\ \mathit{revise}(\Pi, \{P\}, \{P'\}, \Pi'), \\ \mathit{in}(X, A', \Pi') \end{gathered}$$

Properties: $\left\{ \begin{array}{c} \mathit{attacks}(A', A), \\ \mathit{consistent}(\Pi'), \\ \mathit{supports}(A', X : \mathbf{in}, \Pi'), \\ \mathit{unify}(X, H'), \\ \mathit{satisfiable}(\mathbf{B}\sigma, \Pi) \end{array} \right\}$

Conditions: $\begin{array}{l} X : \mathbf{out} \in \mathcal{G}_A, \\ H \leftarrow \mathbf{B} \in \Pi, \\ P = H \leftarrow \mathbf{B}, \\ H' \in \mathcal{L}, \\ P' = H' \leftarrow \mathbf{B}, \\ \sigma = \mathit{mgu}(X, H') \end{array}$

C.3.4 Reversing a Rule

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 6 \Rightarrow 11 \Rightarrow 25$

$$\begin{gathered} in(X, A, \Pi), \\ retract(reversion(P)),\ \ add(reversion(P')),\ \ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ in(Y, A', \Pi') \end{gathered}$$

Properties: $\left\{ \begin{array}{c} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', Y : \mathbf{in}, \Pi'), \\ unify(Y, B), \\ satisfiable(H\sigma, \Pi) \end{array} \right\}$

Conditions: $\begin{array}{l} X : \mathbf{in} \in \mathcal{G}_A, \\ Y \in conflict(X), \\ H \leftarrow B \in \Pi, \\ P = H \leftarrow B, \\ \sigma = mgu(Y, B), \\ P' = B \leftarrow H \end{array}$

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 4 \Rightarrow 7 \Rightarrow 11 \Rightarrow 25$

$$\begin{gathered} out(X, A, \Pi), \\ retract(reversion(P)),\ \ add(reversion(P')),\ \ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ in(X, A', \Pi') \end{gathered}$$

Properties: $\left\{ \begin{array}{c} attacks(A', A), \\ consistent(\Pi'), \\ supports(A', X : \mathbf{in}, \Pi'), \\ unify(X, B), \\ satisfiable(H\sigma, \Pi) \end{array} \right\}$

Conditions: $\begin{array}{l} X : \mathbf{out} \in \mathcal{G}_A, \\ H \leftarrow B \in \Pi, \\ P = H \leftarrow B, \\ \sigma = mgu(X, B), \\ P' = B \leftarrow H \end{array}$

C.4 Elementary Revisions for Removing an Argument

Remember that the property *attacks* holds if an argument in the revised theory attacks an argument in the original theory. Hence in this context $attacks(A, A)$ does not stand for self-defeating arguments, but rather it denotes that argument A in Π' is a refutation of argument A in Π.

C.4.1 Retracting an Invalid Rule

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 5 \Rightarrow 10 \Rightarrow 16$

$$\begin{gathered} in(X, A, \Pi), \\ retract(invalid\_rule(P)),\ \ revise(\Pi, \{P\}, \{\}, \Pi'), \\ out(X, A, \Pi') \end{gathered}$$

$$\text{Properties:} \quad \left\{ \begin{gathered} attacks(A, A), \\ consistent(\Pi'), \\ supports(A, X : \mathbf{out}, \Pi'), \\ unify(X, H) \end{gathered} \right\}$$

$$\begin{array}{ll} \text{Conditions:} & X : \mathbf{in} \in \mathcal{G}_A, \\ & H \leftarrow \mathbf{B} \in A, \\ & P = H \leftarrow \mathbf{B}, \\ & \exists \sigma' \in subst.\ affirm(\mathbf{B}\sigma' \wedge not(H\sigma')) \end{array}$$

C.4.2 Retracting a Weak Rule

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 5 \Rightarrow 10 \Rightarrow 17$

$$\begin{gathered} in(X, A, \Pi), \\ retract(weak\_rule(P)),\ \ revise(\Pi, \{P\}, \{\}, \Pi'), \\ out(X, A, \Pi') \end{gathered}$$

$$\text{Properties:} \quad \left\{ \begin{gathered} attacks(A, A), \\ consistent(\Pi'), \\ supports(A, X : \mathbf{out}, \Pi'), \\ unify(X, H) \end{gathered} \right\}$$

$$\begin{array}{ll} \text{Conditions:} & X : \mathbf{in} \in \mathcal{G}_A, \\ & H \leftarrow \mathbf{B} \in A, \\ & P = H \leftarrow \mathbf{B}, \\ & \exists \sigma' \in subst.\ affirm(not(\mathbf{B}\sigma')) \end{array}$$

C.4.3 Retracting a Misrelation

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 5 \Rightarrow 10 \Rightarrow 18$

$$\begin{gathered} in(X, A, \Pi), \\ retract(misrelation(P)),\ \ revise(\Pi, \{P\}, \{\}, \Pi'), \\ out(X, A, \Pi') \end{gathered}$$

Properties: $\left\{ \begin{gathered} attacks(A, A), \\ consistent(\Pi'), \\ supports(A, X : \mathbf{out}, \Pi'), \\ unify(X, H) \end{gathered} \right\}$

Conditions: $X : \mathbf{in} \in \mathcal{G}_A$,
$H \leftarrow \mathbf{B} \in A$,
$P = H \leftarrow \mathbf{B}$,
$\exists \sigma', \sigma'' \in subst. affirm(\mathbf{B}\sigma' \wedge not(H\sigma') \wedge H\sigma'' \wedge not(\mathbf{B}\sigma''))$

C.5 Updating Revisions for Removing an Argument

Again, note that the property $attacks(A, A)$ denotes that argument A in a revised theory Π' is a refutation of argument A in the original theory Π.

C.5.1 Elaborating Preconditions in a Rule

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 5 \Rightarrow 12 \Rightarrow 20$

$$\begin{gathered} in(X, A, \Pi), \\ retract(elaboration(P)),\ \ add(elaboration(P')),\ \ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ out(X, A, \Pi') \end{gathered}$$

$$\text{Properties:} \quad \left\{ \begin{array}{c} attacks(A, A), \\ consistent(\Pi'), \\ supports(A, X : \mathbf{out}, \Pi'), \\ unify(X, H), \\ satisfiable((B_1 \wedge ... \wedge B_m)\sigma, \Pi) \\ \neg satisfiable(B\sigma, \Pi) \end{array} \right\}$$

$$\text{Conditions:} \quad \begin{array}{l} X : \mathbf{in} \in \mathcal{G}_A, \\ H \leftarrow B_1 \wedge ... \wedge B_m \in A, \\ P = H \leftarrow B_1 \wedge ... \wedge B_m, \\ B \in \mathcal{L}, \\ i \in \{0, ..., m\}, \\ P' = H \leftarrow B_1 \wedge ... \wedge B_i \wedge B \wedge B_{i+1} \wedge ... \wedge B_m, \\ \sigma = mgu(X, H) \end{array}$$

C.5.2 Specialising a Rule

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 5 \Rightarrow 12 \Rightarrow 22$

$$\begin{gathered} in(X, A, \Pi), \\ retract(specialisation(P)),\ \ add(specialisation(P')),\ \ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ out(X, A, \Pi') \end{gathered}$$

$$\text{Properties:} \quad \left\{ \begin{array}{c} attacks(A, A), \\ consistent(\Pi'), \\ supports(A, X : \mathbf{out}, \Pi'), \\ unify(X, H), \\ ground(P', \Pi) \subset ground(P, \Pi), \\ \forall (H_g \leftarrow \mathbf{B}_g) \in ground(P\sigma, \Pi) \cap ground(P', \Pi). \\ \neg satisfiable(\mathbf{B}_g, \Pi) \end{array} \right\}$$

$$\text{Conditions:} \quad \begin{array}{l} X : \mathbf{in} \in \mathcal{G}_A, \\ H \leftarrow \mathbf{B} \in A, \\ P = H \leftarrow \mathbf{B}, \\ \sigma = mgu(X, H), \\ \sigma' \in subst, \\ P' = (H \leftarrow \mathbf{B})\sigma' \end{array}$$

Here the property $ground(P\sigma', \Pi) \subset ground(P, \Pi)$ also holds by construction because σ' is a substitution.

C.5.3 Revising the Consequent of a Rule

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 5 \Rightarrow 12 \Rightarrow 24$

$$\begin{gathered} in(X, A, \Pi), \\ retract(misconclusion(P)),\ add(misconclusion(P')),\ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ out(X, A, \Pi') \end{gathered}$$

$$\text{Properties:} \quad \left\{ \begin{gathered} attacks(A, A), \\ consistent(\Pi'), \\ supports(A, X : \mathbf{out}, \Pi'), \\ unify(X, H), \\ \neg unify(X, H') \end{gathered} \right\}$$

$$\text{Conditions:} \quad \begin{aligned} & X : \mathbf{in} \in \mathcal{G}_A, \\ & H \leftarrow \mathbf{B} \in A, \\ & P = H \leftarrow \mathbf{B}, \\ & H' \in \mathcal{L}, \\ & P' = H' \leftarrow \mathbf{B} \end{aligned}$$

C.5.4 Reversing a Rule

Applying Argument Rewrites $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 5 \Rightarrow 12 \Rightarrow 26$

$$\begin{gathered} in(X, A, \Pi), \\ retract(reversion(P)),\ add(reversion(P')),\ revise(\Pi, \{P\}, \{P'\}, \Pi'), \\ out(X, A, \Pi') \end{gathered}$$

$$\text{Properties:} \quad \left\{ \begin{gathered} attacks(A, A), \\ consistent(\Pi'), \\ supports(A, X : \mathbf{out}, \Pi'), \\ unify(X, H), \\ \neg unify(X, B) \end{gathered} \right\}$$

$$\text{Conditions:} \quad \begin{aligned} & X : \mathbf{in} \in \mathcal{G}_A, \\ & H \leftarrow B \in A, \\ & P = H \leftarrow B, \\ & P' = B \leftarrow H \end{aligned}$$

Appendix D

Checking the Property *supports*

The property *supports* ensures whether an argument can be advanced or not in order to support the intended claim in the context of moves advanced so far. Its main purpose is to avoid circularity and ineffective repetition of arguments.

Intuitively, an argument is not allowed if it has been advanced before to attack the same claim via the same attack-based revision operation. Moreover, if the revision is non-trivial, then the argument must account for some premise that has either been retracted or added by the corresponding operation.

So, let

$$\langle A_0, \phi_1, A_1, ..., \phi_{i-1}, A_{i-1} \rangle$$

represent the argument process so far, and A_i be an argument in the revised theory Π_{ϕ_i} that is generated via the operation ϕ_i in order to support a claim C. Then:

$$\mathit{holds}(\mathit{supports}(A_i, C, \Pi_{\phi_i}),\ \phi_i,\ \langle A_0, \phi_1, A_1, ..., \phi_{i-1}, A_{i-1} \rangle)$$

ensures that argument A_i is a valid move in the process.

The predicate `holds/3` is used to check the various properties associated with an attack. Below is the specification currently used in our system for verifying the property *supports*. Note that as in the case of any other property, designers of argument systems may choose to relax or strengthen this specification. Predicate `argtree_member/2` verifies if a premise is used the argument; i.e. if it defines some sub-tree in the corresponding argument tree.

The predicate `nextto/3` is a list operation defined in SICStus that checks whether two elements appear side-by-side in a list.

```
%-----------------------------------------------------------------
% Checking property:  supports

%--- trivial revision
holds(supports(A, X:in, Theory), RevisionOp, ArgSofar) :-
    trivial(RevisionOp),
    attack_type(RevisionOp, X:out ==> X:in),
    \+ member(A, ArgSofar).

holds(supports(A, X:in, Theory), RevisionOp, ArgSofar) :-
    trivial(RevisionOp),
    attack_type(RevisionOp, X1:in ==> X:in),
    \+ nextto(A, RevisionOp, ArgSofar).

%--- non-trivial revision, where Rem is nonempty
holds(supports(A, C, Theory), RevisionOp, ArgSofar) :-
    nontrivial(RevisionOp, r(Rem, Add)),
    member(Axiom, Rem),
    argtree_member(Axiom, A),
    \+ nextto(A, RevisionOp, ArgSofar).

%--- non-trivial revision, where Rem is empty
holds(supports(A, C, Theory), RevisionOp, ArgSofar) :-
    nontrivial(RevisionOp, r(Rem, Add)),
    \+ member(Axiom, Rem),
    member(Axiom, Add),
    argtree_member(Axiom, A),
    \+ nextto(A, RevisionOp, ArgSofar).
```

Figure D.1: Checking the property *supports*.

Appendix E

Architecture: the Pressure Tank Example

This Appendix gives the Prolog file containing the architecture definition for the pressure tank example, as described in Chapter 11.

```
%%% File:
%%%    architecture.pl
%%% Author:
%%%    Daniela Carbogim
%%% Purpose:
%%%    Specify each component of the architecture to be
%%%    used by the argument generator

%%%-----------------------------------------------------------
%%%-                         Theory                          -
%%%-----------------------------------------------------------

main(ptmodel).
provability(ptmodel, solve).
theory(ptmodel,
  [axiom(1,  operational_tank(tank(pt), T),
               [on(motor(m), T), not_full(tank(pt), T)]),
   axiom(2,  operational_tank(tank(pt), T),
               [off(motor(m), T), full(tank(pt), T)]),
   axiom(3,  not_operational_tank(tank(pt), T),
               [on(motor(m), T), full(tank(pt), T)]),
   axiom(4,  on(motor(m), T),
               [closed(relay(k2), T)]),
   axiom(5,  off(motor(m), T),
               [open(relay(k2), T)]),
   axiom(6,  closed(relay(k2), T),
               [closed(relay(k1), T), closed(switch(ps), T)]),
   axiom(7,  open(relay(k2), T),
               [open(relay(k1), T)]),
   axiom(8,  open(relay(k2), T),
               [open(switch(ps), T)]),
   axiom(9,  closed(relay(k1), T),
               [closed(relay(timer), T),
                closed(switch(s1), Tp), precedes(T, Tp)]),
```

```
   axiom(10, open(relay(k1), T),
              [open(relay(timer), T)]),
   axiom(11, closed(switch(ps), T),
              [not_full(tank(pt), T)]),
   axiom(12, open(switch(ps), T),
              [full(tank(pt), T)]),
   axiom(13, closed(switch(s1), T),
              [initial_time(T)]),
   axiom(14, open(switch(s1), T),
              [initial_time(Ti), greater(T, Ti)]),
   axiom(15, closed(relay(timer), T),
              [timing(relay(timer), TC, T),
               pressurisation_time(TP), greater(TP, TC)]),
   axiom(16, open(relay(timer), T),
              [timing(relay(timer), TC, T),
               pressurisation_time(TP), geq(TC, TP)]),
   axiom(17, timing(relay(timer), 0, T),
              [initial_time(T)]),
   axiom(18, timing(relay(timer), 0, T),
              [initial_time(Ti), greater(T, Ti), open(switch(ps), T)]),
   axiom(19, timing(relay(timer), TC, T),
              [previous(T, Tp),
               timing(relay(timer), TCp, Tp), increment(TCp,1,TC)]),
   axiom(20, full(tank(pt), T),
              [pressurisation_time(TP), mod(T, TP, 0)]),
   axiom(21, not_full(tank(pt), T),
              [pressurisation_time(TP), mod(T, TP, X), greater(X, 0)]),
   axiom(22, previous(T, Tp),
              [initial_time(Ti), greater(T, Ti), increment(T,-1,Tp)]),
   axiom(23, precedes(T, Tp),
              [previous(T, Tp)] ),
   axiom(24, precedes(T, Tp),
              [previous(T, Tp1), precedes(Tp1, Tp)] ),
   axiom(25, initial_time(0), true),
   axiom(26, pressurisation_time(60), true)]).


%%%--- Conflicting predicates

conflict(operational_tank(P, T), not_operational_tank(P, T)).


%%%--- Provability relation for the theory

solve([], [], _Theory).
solve([X|R], [ArgX|ArgR], Theory) :-
    solve(X, ArgX, Theory),
    solve(R, ArgR, Theory).
solve(true, true, _Theory).
solve(X, arg(X, Id, ArgB), Theory) :-
    member_list(axiom(Id, X, B), Theory),
    solve(B, ArgB, Theory).
solve(X, arg(X, pmtv, true), _Theory) :-
    primitive_pred(X), X.

primitive_pred(greater(_,_)).
primitive_pred(geq(_,_)).
primitive_pred(increment(_,_,_)).
primitive_pred(mod(_,_,_)).
```

```
%%%----------------------------------------------------------
%%%-                    Criticism Theory                    -
%%%----------------------------------------------------------

crit(ftree).
provability(ftree, solve_abd).
theory(ftree,
 [axiom(e1A, tank_rupture, [primary_failure(tank(pt))]),
  axiom(e1B, tank_rupture, [continuous_pump_operation]),
  axiom(e2A, continuous_pump_operation, [primary_failure(relay(k2))]),
  axiom(e2B, continuous_pump_operation, [emf_applied_on(relay(k2))]),
  axiom(e3,  emf_applied_on(relay(k2)),
                [primary_failure(switch(ps)), emf_applied_on(switch(ps))]),
  axiom(e4A, emf_applied_on(switch(ps)), [primary_failure(switch(s1))]),
  axiom(e4B, emf_applied_on(switch(ps)), [emf_applied_on(relay(k1))]),
  axiom(e5A, emf_applied_on(relay(k1)), [primary_failure(relay(k1))]),
  axiom(e5B, emf_applied_on(relay(k1)), [primary_failure(relay(timer))])]).


%%%--- Provability relation for the criticism theory
:- dynamic solve_abd/3.

solve_abd([], [], _FTree).
solve_abd([X|R], Arg, FTree) :-
    solve_abd(X, ArgX, FTree),
    solve_abd(R, ArgR, FTree),
    append(ArgX, ArgR, Arg).
solve_abd(true, [], _FTree).
solve_abd(X, [X], _FTree) :-
    abducible(X).
solve_abd(X, A, FTree) :-
    member_list(axiom(_Id, X, B), FTree),
    solve_abd(B, A, FTree).

abducible(primary_failure(_)).
```

```
%%%-----------------------------------------------------------
%%%-                     Control Module                       -
%%%-----------------------------------------------------------

filter(solve_abd, solve_filter).

%%%--- Measure values for criticism theory(ftree, solve_abd)

m_ftree(abducible(primary_failure(tank(pt))),      5.0e-06).
m_ftree(abducible(primary_failure(relay(k2))),     3.0e-05).
m_ftree(abducible(primary_failure(switch(ps))),    1.0e-04).
m_ftree(abducible(primary_failure(switch(s1))),    3.0e-05).
m_ftree(abducible(primary_failure(relay(k1))),     3.0e-05).
m_ftree(abducible(primary_failure(relay(timer))), 1.0e-04).

filter_threshold(M) :-
    M > 0.1.


%%%--- Propagation mechanism solve_filter for solve_abd

solve_filter(solve_abd(X, A, FTree)) :-
    measure_arg(solve_abd(X, A, FTree), M1),
    measure_sent(X, M2, FTree),
    combine_measure(M1, M2, M),
    filter_threshold(M).

measure_sent(X, M, FTree) :-
    findall(MX, measure_arg(solve_abd(X, _A, FTree), MX), Ms),
    combine_measure_sent(Ms, M).

measure_arg((A1, A2), M) :-
    measure_arg(A1, M1),
    measure_arg(A2, M2),
    combine_measure_arg(M1, M2, M).
measure_arg(true, 1).
measure_arg(abducible(X), M) :-
    m_ftree(abducible(X), M).
measure_arg(solve_abd(X, A, FTree), M) :-
    clause(solve_abd(X, A, FTree), B),
    measure_arg(B, M).
measure_arg(A, 1) :-
    \+ A = true,
    \+ A =.. [solve_abd|_],
    \+ A =.. [abducible|_],
    \+ A = (_A1, _A2),
    A.

combine_measure_arg(M1, M2, M) :-
   M is M1*M2.
combine_measure_sent(Ms, M) :-
   sum(Ms, M).
combine_measure(M1, M2, M) :-
   M is M1/M2.

%---------------------------------------------------------------
% EOF EOF EOF EOF EOF EOF EOF EOF EOF EOF EOF EOF EOF EOF EOF EOF
%---------------------------------------------------------------
```

Bibliography

Ambler, S. (1996). A categorical approach to the semantics of argumentation. *Mathematical Structures in Computer Science*, 6(2):167–188.

Amgoud, L. and Cayrol, C. (1998). On the acceptability of arguments in preference-based argumentation frameworks. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI98)*, pages 1–7.

Anrig, B., Bissig, R., Haenni, R., Kohlas, J., and Lehmann, N. (1999). Probabilistig argumentation systems: introduction to assumption-based modeling with ABEL. Technical report 99-1, Institute of Informatics of the University of Fribourg. It is possible to download ABEL from the project homepage at `http://www2-iiuf.unifr.ch/tcs/ABEL/`.

Antoniou, G. (1998). A tutorial on default reasoning. *The Knowledge Engineering Review*, 13(3):225–246.

Apt, K. R. (1995?). *From logic programming to Prolog*. ?

Bench-Capon, T., Freeman, J., Hohmann, H., and Prakken, H. (2000). Computational models, argumentation theories and legal practice. In *Handbook on Argument and Computation, Bonskeid Symposium on Argument and Computation*. (in preparation).

Bondarenko, A., Dung, P. M., Kowalski, R. A., and Toni, F. (1997). An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93(1–2):63–101.

Brewka, G. (1994). A reconstruction of Rescher's theory of formal disputation based on default logic. In Cohn, A. G., editor, *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI94)*, pages 366–370.

Brewka, G. (1996). Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research*, 4:19–36.

Buckingham Shum, S. and Hammond, N. (1994). Argumentation-based design rationale: what use at what cost? *International Journal of Computer Studies*, 40(4):603–652.

Carbogim, D., Krabbe, E. C. W., Norman, T., and Walton, D. (2000a). Argumentation in multi-agent systems. In *Handbook on Argument and Computation, Bonskeid Symposium on Argument and Computation*. (in preparation).

Carbogim, D. and Robertson, D. (1999). Contract-based negotiation via argumentation. In *Workshop on Multi-Agent Systems in Logic Programming (MAS99) at the International Conference on Logic Programming (ICLP99)*. Also available at the workshop site `http://www.cs.sfu.ca/conf/MAS99/`.

Carbogim, D. and Wassermann, R. (2000). Full acceptance via argumentation. In *Proceedings of the Discussion Track of The International Joint Conference SBIA/IBERAMIA2000 (15th Brazilian Symposium on Artificial Intelligence and 7th Ibero-American Conference on Artificial Intelligence)*, Atibaia, Brazil.

Carbogim, D. V., Robertson, D., and Lee, J. (2000b). Argument-based applications to knowledge engineering. *The Knowledge Engineering Review*, 15(2):119–149.

Carbogim, D. V., Robertson, D. S., and Lee, J. R. (1999). Extending the abstract argumentation framework to describe argument dynamics. DAI Research Paper 940, Artificial Intelligence, Division of Informatics, University of Edinburgh.

Chesñevar, C. I., Maguitman, A. G., and Loui, R. P. (1999). Logical models of argument. *ACM Computing Surveys*, submitted. Available at `http://www.cs.wustl.edu/~loui/survey.ps`.

Conklin, J. and Begeman, M. L. (1988). gIBIS: a hypertext tool for exploratory policy discussion. *ACM Transactions on Information Systems*, 6(4):303–331.

Corrêa da Silva, F. S., Vasconcelos, W. W., Agustí, J., Robertson, D., and Melo, A. C. V. (1999). Why ontologies are not enough for knowledge sharing. In *Proceedings of the 12th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA-AIE99)*, Cairo, Egypt.

Crosswhite, J., Fox, J., Reed, C., Scaltsas, D., and Stumpf, S. (2000). Computational models of rhetorical argument. In *Handbook on Argument and Computation, Bonskeid Symposium on Argument and Computation.* (in preparation).

Das, S., Fox, J., and Krause, P. (1996). A unified framework for hypothetical and practical reasoning (1): theoretical foundations. In Gabbay, D. M. and Ohlbach, H. J., editors, *Proceedings of the International Conference on Formal and Applied Practical Reasoning (FAPR96)*, pages 58–72.

Daskalopulu, A. and Sergot, M. (1997). Representation of legal contracts. *AI and Society*, 11(1–2):6–17.

de Kleer, J. (1986). An assumption based truth maintenance system. *Artificial Intelligence*, 28:127–162.

Deville, Y. and Lau, K.-K. (1994). Logic program synthesis. *Journal of Logic Programming*, 19/20:321–350.

Dimopoulos, Y., Nebel, B., and Toni, F. (1999). Preferred arguments are harder to compute than stable extensions. In Dean, T., editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI99)*, pages 36–41.

Dix, J. and Brewka, G. (1997). Knowledge representation with logic programs. In Dix, J., Pereira, L. M., and Przymusinski, T. C., editors, *Logic Programming and Knowledge Representation, Third International Workshop (LPKR97)*, number 1471 in Lecture Notes in Computer Science, pages 1–51, Port Jefferson, USA. Springer-Verlag. To appear in Handbook of Philosophical Logic, 2nd edition, Volume 6, Chapter 6, Oxford University Press, 2001.

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*, 12(3):231–272.

Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358.

Elvang-Goransson, M., Krause, P., and Fox, J. (1993). Acceptability of arguments as logical uncertainty. In Clarke, M., Kruse, R., and Moral, S., editors, *Proceedings of European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU93)*, volume 747 of *Lecture Notes in Computer Science*, pages 85–90. Springer-Verlag.

Ferguson, G. and Allen, J. F. (1994). Arguing about plans: plan representation and reasoning for mixed-initiative planning. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS94)*, pages 43–48, Chicago, USA.

Finin, T., Labrou, Y., and Mayfield, J. (1997). KQML as an agent communication language. In Bradshaw, J. M., editor, *Software Agents*, pages 291–316. AAAI Press/MIT Press.

Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., and Nuseibeh, B. (1994). Inconsistency handling in multi-perspective specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578.

Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., and Goedicke, M. (1992). Viewpoints: a framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1):31–58.

Fogelin, R. J. and Sinnott-Armstrong, W. (1997). *Understanding arguments, an introduction to informal logic.* Harcourt Brace College Publishers, fifth edition.

Forbus, K. D. and de Kleer, J. (1993). *Building problem solvers.* The MIT Press.

Fox, J. (1994). An example of the use of formal argument in assessing cancer risk: the cases for and against FDA policy on aflatoxins. Technical report, Imperial Cancer Research Fund, London.

Fox, J. and Das, S. (1996). A unified framework for hypothetical and practical reasoning (2): lessons from medical applications. In Gabbay, D. M. and Ohlbach, H. J., editors, *Proceedings of the International Conference on Formal and Applied Practical Reasoning (FAPR96)*, pages 73–92.

Fox, J. and Das, S. (2000). *Safe and Sound: Artificial Intelligence in Hazardous Applications.* Jointly published by the AAAI and MIT Press.

Fox, J. and Krause, P. (1992). Qualitative frameworks for decision support: lessons from medicine. *The Knowledge Engineering Review*, 7(1):19–33.

Fox, J., Krause, P., and Ambler, S. (1992). Arguments, contradictions and practical reasoning. In Neumann, B., editor, *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI92)*, pages 623–627.

Fox, J. and Parsons, S. (1998). Arguing about beliefs and actions. In Hunter, A. and Parsons, S., editors, *Applications of Uncertainty Formalisms*, volume 1455 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Freeman, K. (1993). *Toward formalizing dialectical argumentation*. PhD thesis, Department of Computer and Information Science of the University of Oregon, USA.

Freeman, K. and Farley, A. M. (1992). Argumentation in weak theory domains. In Ryan, K. and Sutcliffe, R., editors, *Proceedings of the 5th Irish Conference on Artificial Intelligence and Cognitive Science*. Springer-Verlag.

Freeman, K. and Farley, A. M. (1996). A model of argumentation and its application to legal reasoning. *Artificial Intelligence and Law*, 4(3–4):163–197.

Gabbay, D. M. (1996). *Labelled Deductive Systems: principles and applications. Volume 1: introduction*, volume 33 of *Oxford Logic Guides*. Oxford University Press.

Gabbay, D. M. (1999). What's on my mind... column. *Journal of Logic and Computation*, 9(1):3–6.

Gabbay, D. M. (2000). *Dynamics of Practical Reasoning.* (draft).

Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4):365–385.

Gerlofs, J.-M., Gilbert, M., Grasso, F., Groarke, L., and Gurr, C. (2000). The persuasion machine: An exercise in argumentation and computational linguistics. In *Handbook on Argument and Computation, Bonskeid Symposium on Argument and Computation.* (in preparation).

Gilbert, M. (1995). The delimitation of argument. *Inquiry*, 15(1):63–75.

Girle, R., Hitchcock, D., McBurney, P., and Verheij, B. (2000). Practical reasoning: an argument and computation perspective. In *Handbook on Argument and Computation, Bonskeid Symposium on Argument and Computation.* (in preparation).

Gordon, T. F. and Karacapilidis (1997). The Zeno argumentation framework. In Borges, D. L. and Kaestner, C. A. A., editors, *Proceedings of the International Conference on Artificial Intelligence and Law (ICAIL-97)*, pages 10–18, University of Melbourne, Australia. More information on the Zeno System is available at the project homepage at `http://ais.gmd.de/MS/zeno/zenoSystem.html`.

Grasso, F. (1998). Exciting avocados and dull pears: combining behavioural and argumentative theory for producing effective advice. In *Proceedings of 20th Annual Meeting of the Cognitive Science Society (COG-SCI98)*, pages 436–441, Madison, USA.

Grosof, B. (1997). Prioritized conflict handling for logic programs. In Maluszynski, J., editor, *Proceedings of the International Symposium on Logic Programming (ILPS97)*, pages 197–211, Port Jefferson, USA.

Gurr, C. A. (1997). Knowledge engineering in the communication of information for safety critical systems. *Knowledge Engineering Review*, 12(3):249–270.

Haenni, R. (1998). Modeling uncertainty with propositional assumption-based systems. In Hunter, A. and Parsons, S., editors, *Applications of Uncertainty Formalisms*, volume 1455 of *Lecture Notes in Artificial Intelligence*, pages 446–470. Springer Verlag.

Haggith, M. (1996). *A meta-level framework for representing and reasoning about disagreement*. PhD thesis, University of Edinburgh, Department of Artificial Intelligence.

Jackson, M. (1994). Problems, methods and specialisation. *Software Engineering Journal*, 9(6):249–255.

Jakobovits, H. (2000). *On the theory of argumentation frameworks*. PhD thesis, Vrije Universiteit Brussel.

Jennings, N. R., Faratin, P., Johnson, M. J., O'Brien, P., and Wiegand, M. E. (1996). Agent-based business process management. *International Journal of Cooperative Information Systems*, 5(2–3):105–130.

Jennings, N. R., Parsons, S., Noriega, P., and Sierra, C. (1998). On argumentation-based negotiation. In *Proceedings of the International Workshop on Multi-Agents System (IWMAS)*, pages 1–7, Boston, USA.

Kakas, A. C. and Toni, F. (1999). Computing argumentation in logic programming. *Journal of Logic and Computation*, 9(4):515–562.

Konolige, K. (1988). Defeasible argumentation in reasoning about events. In Ras, Z. W. and Saitta, L., editors, *Proceedings of the Third International Symposium on Methodologies for Intelligent Systems (ISMIS 1988)*, pages 380–390, Turin, Italy.

Kowalski, R. and Toni, F. (1994). Argument and reconciliation. In *Proceedings of the Workshop of Applications of Logic Programming to Legal Reasoning at the International Symposium on Fifth Generation Computer Systems (ICOT94)*, Tokyo.

Kowalski, R. and Toni, F. (1996). Abstract argumentation. *Artificial Intelligence and Law*, 4(3–4):275–296. Special Issue on Logical Models of Argumentation, H. Prakken and G. Sartor, editors.

Krause, P., Ambler, S., Elvang-Goransson, M., and Fox, J. (1995). A logic of argumentation for reasoning under uncertainty. *Computational Intelligence*, 11(1):113–131.

Krause, P. and Clark, D. (1993). *Representing uncertain knowledge: an artificial intelligence approach*. Intellect Books, Oxford.

Krause, P., Hesketh, J., and Robertson, D. (1997). Reliable and accountable system design. *Knowledge Engineering Review*, 12(3):289–305.

Labrou, Y. and Finin, T. (1994). A semantics approach for KQML – a general purpose communication language for software agents. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM94)*, pages 447–455, Gaithersburg, USA.

Labrou, Y., Finin, T., and Peng, Y. (1999). Agent communication languages: the current landscape. *IEEE Intelligent Systems*, 14(2):45–52.

Lin, F. and Shoham, Y. (1989). Argument systems: a uniform basis for nonmonotonic reasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR89)*, pages 245–255. Morgan Kaufmann Publishers Inc.

Lloyd, J. W. (1987). *Foundations of logic programming (2ed).* Springer Verlag.

Loui, R. P. (1987). Defeat among arguments: a system of defeasible inference. *Computational Intelligence*, 2:100–106.

Loui, R. P. (1998). Process and policy: resource-bounded non-demonstrative reasoning. *Computational Intelligence*, 14(1):1–38.

MacKenzie, D. (1996). A worm in the bud? Computers, systems, and the safety-case problem. Prepared for a Dibner Institute Symposium on The Spread of the Systems Approach, convened by Thomas P. Hughes. Cambridge, Mass, 3-5 May, 1996.

McBurney, P. and Parsons, S. (1999). Truth or consequence: using argumentation to reason about risk. In *BPS Symposium on Practical Reasoning*, London, UK.

McBurney, P. and Parsons, S. (2000). Risk agoras: Using dialectical argumentation to debate risk. *Risk Management Journal*, (forthcoming).

Móra, I. A., Alferes, J. J., and Schroeder, M. (1998). Argumentation and cooperation for distributed extended logic programs. In *Proceedings of the 7th Workshop on Non-Monotonic Reasoning (NMR98)*, Trento, Italy.

Moran, T. P. and Carroll, J. M., editors (1996). *Design rationale: concepts, techniques, and use.* Computer, Cognition, and Work. Lawrence Erlbaum Associates.

Ng, B. H.-K., Wong, K.-F., and Low, B.-T. (1998). Resolving conflicting arguments under uncertainties. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI98).*

Norman, T. and Reed, C., editors (2000). *Handbook on Argument and Computation, Bonskeid Symposium on Argument and Computation.* (in preparation).

Nute, D. (1988). Defeasible reasoning and decision support systems. *Decision Support Systems*, 4:97–110.

Nute, D. (1994). Defeasible logic. In Gabbay, D. M., Hogger, C. J., and Robinson, J. A., editors, *Nonmonotonic Reasoning and Uncertain Reasoning*, volume 3 of *Handbook of Logic for Artificial Intelligence and Logic Programming*, pages 353–395. Oxford University Press.

Nwana, H. and Ndumu, D. (1999). A perspective on software agents research. *The Knowledge Engineering Review*, 14(2):125–142.

Parsons, S. and Fox, J. (1997). Argumentation and decision making. In *Proceedings of the IEE Coloquium on Decision Making and Problem Solving.*

Parsons, S. and Jennings, N. R. (1997). Negotiation through argumentation – a preliminary report. In *Proceedings of 2nd International Conference on Multi-Agents System*, pages 267–274, Kyoto, Japan.

Parsons, S., Sierra, C., and Jennings, N. R. (1998). Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292.

Perelman, C. and Olbrects-Tyteca, L. (1969). *The New Rethoric - a treatise on argumentation*. University of Notre Dame Press, Notre Dame/London.

Pettorossi, A. and Proietti, M. (1998). Transformation of logic programs. In Gabbay, D. M., Hogger, C. J., and Robinson, J. A., editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, Logic Programming, pages 697–787. Oxford Science Publications.

Pollock, J. (1987). Defeasible reasoning. *Cognitive Science*, 11(4):481–518.

Prakken, H. (1995). From logic to dialectics in legal argument. In *Proceedings of the Fifth International Conference on Artificial Intelligence and Law*, pages 165–174, Washington DC, USA. ACM Press.

Prakken, H. (1997a). Dialectical proof theory for defeasible argumentation with defeasible priorities (preliminary report). In *Proceedings of the 4th ModelAge Workshop on Formal Models of Agents.*

Prakken, H. (1997b). *Logical tools for modelling legal argument: a study of defeasible reasoning in law.* Kluwer Academic Publishers.

Prakken, H. (2000). Relating protocols for dynamic dispute with logics for defeasible argumentation. *Synthese*, forthcoming. Special issue on New Perspectives in Dialogical Logic, S. Rahman and H. Rückert, editors.

Prakken, H. and Sartor, G. (1996). A dialectical model of assessing conflicting arguments in legal reasoning. *Artificial Intelligence and Law*, 4(3–4):331–368.

Prakken, H. and Sartor, G. (1997). Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-classical Logics*, 7(1):25–75. Special issue on Handling Inconsistency in Knowledge Systems.

Prakken, H. and Vreeswijk, G. (1999). Logics for defeasible argumentation. In Gabbay, D., editor, *Handbook of Philosophical Logic (to appear).* Kluwer Academic Publishers, second edition.

Rao, A. S. and Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture. In Allen, J., Fikes, R., and Sandewall, E., editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR91)*, pages 473–484. Morgan Kaufmann.

Rao, A. S. and Georgeff, M. P. (1995). Formal models and decision procedures for multi-agent systems. Technical report 61, Australian Artificial Intelligence Institute, Melbourne, Australia.

Reed, C. (1997). Representing and applying knowledge for argumentation in a social context. *AI & Society*, 11(3–4):138–154.

Reeves, D. M., Grosof, B. N., Wellman, M. P., and Chan, H. Y. (1999). Towards a declarative language for negotiating executable contracts. In Finin, T. and Grosof, B., editors, *Proceedings of the AAAI99 Workshop on Artificial Intelligence in Electronic Commerce (AIEC99)*, Orlando, USA. Paper also available as IBM Research Report RC 21476.

Reeves, D. M., Grosof, B. N., Wellman, M. P., and Chan, H. Y. (2000). Generating auction configurations from declarative contract descriptions. In *Proceedings of the AAAI2000 Workshop on Knowledge-based Electronic Markets (KBEM-00).*

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13:81–132.

Rittel, H. and Webber, M. (1973). Dilemmas in a general theory of planning. *Policy Science*, 4:155–169.

Robertson, D. (1995). Some thoughts on logics of arguments. An unpublished note on the paper 'An example of the use of formal argument in assessing cancer risk' by John Fox.

Robertson, D. (1999a). Can formal argumentation raise our confidence in safe design? In *Towards System Safety: Proceedings of the Seventh Safety-Critical Systems Symposium*, Huntingdon, UK.

Robertson, D. (1999b). *Pragmatics in the synthesis of logic programs.* PhD thesis, Universitat Autònoma de Barcelona.

Robertson, D. and Agustí, J. (1999). *Software blueprints: lightweight uses of logic in conceptual modelling.* Addison-Wesley (ACM Press).

Rodricks, J. V. (1992). *Calculated risks: the toxicity and human health risks of chemicals in our environment.* Cambridge University Press.

Schroeder, M. (1999a). An efficient argumentation framework for negotiating autonomous agents. In *Proceedings of the Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW99).*

Schroeder, M. (1999b). Using VRML to visualise argumentation, agents and arguing agents: a preliminary report. In *Proceedings of Educational Applications of VRML (VRML99).*

Shoham, Y. (1994). *Artificial Intelligence Techniques in Prolog.* Morgan Kaufmann Publishers.

Sierra, C., Faratin, P., and Jennings, N. R. (1997a). A service-oriented negotiation model between autonomous agents. In *Proceedings of 8nd European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, Ronneby, Sweden.

Sierra, C., Jennings, N. R., Noriega, P., and Parsons, S. (1997b). A framework for argumentation-based negotiation. In *Proceedings of 4th International Workshop on Agent Theories, Architectures and Languages (ATAL'97)*, pages 167–182, Rhode Island, USA.

Sigman, S. and Liu, X. F. (1999). An intelligent argumentation methodology for capturing and analysing design rationale from multiple persperctives. In *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering (SEKE99)*, pages 227–231.

Simari, G. R. and Loui, R. P. (1992). A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53(2–3):125–157.

Sterling, L. and Shapiro, E. (1994). *The art of Prolog.* MIT Press, 2nd edition.

Storey, N. (1996). *Safety-critical computer systems.* Addison-Wesley.

Sycara, K. (1990). Persuasive argumentation in negotiation. *Theory and Decision*, 28(3):203–242.

Toulmin, S. (1958). *The uses of argument.* Cambridge University Press, Cambridge.

van Eemeren, F. H., Grootendorst, R., and Henkemans, F. S. (1996). *Fundamentals of argumentation theory.* Lawrence Erlbaum Associates.

van Eemeren, F. H., Grootendorst, R., and Kruiger, T. (1987). *Handbook of argumentation theory: a critical survey of classical backgrounds and modern studies.* Studies of Argumentation in Pragmatics and Discourse Analysis. Foris, Dordrecht.

Verheij, B. (1996). *Rules, reasons, arguments: formal studies of argumentation and defeat.* PhD thesis, Universiteit Maastricht.

Vesely, W. E., Goldberg, F. F., Roberts, N. H., and Haasl, D. F. (1981). *Fault tree handbook (NUREG-0492).* U.S. Nuclear Regulatory Commission.

Vreeswijk, G. (1993). *Studies in defeasible argumentation.* PhD thesis, Vrije Universiteit, Amsterdam.

Vreeswijk, G. (1997). Abstract argumentation systems. *Artificial Intelligence*, 90(1–2):225–279.

Walton, D. N. (1996). *Argument Schemes for Presumptive Reasoning.* Lawrence Erlbaum Associates.

Walton, D. N. and Krabbe, E. C. W. (1995). *Commitment in dialogue: basic concepts of interpersonal reasoning.* State University of New York Press.

Warnick, B. and Kline, S. L. (1992). The New Rethoric's argument schemes: a rethorical view of practical reasoning. *Argumentation and Advocacy*, 29(Summer 1992):1–15.

Wooldridge, M. and Parsons, S. (2000). Languages for negotiation. In Horn, W., editor, *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI2000)*, Berlin, Germany. John Wiley.

Yalcinalp, L. U. and Sterling, L. (1991). Uncertainty reasoning in Prolog with layered meta-interpreters. In *Proceedings of the 7th IEEE Conf. in AI Applications.*

Index