UNIVERSITY OF CALGARY

Malicious Argumentation in Open Multi-Agent Systems

by

Andrew Kuipers

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

December, 2010

Abstract

Argumentation provides a powerful means to achieve complex, non-demonstrative reasoning within a multi-agent system. However, the advantages gained by this form of automated reasoning are not without consequence. When arguments are constructed out of formulae in an underlying formal logical language, decisions such as argument evaluation involve deciding the consistency or logical entailment of the component formulae of arguments. Given that these decisions are generally intractable, and further that an agent's decisions need to be resource bounded, malicious exploitation is possible. By strategically expanding the syntactic content of its arguments, a malicious agent may exploit the resource bounds of its opponent's decision procedures, effectively manipulating their outcome towards its own ends. This type of exploitation presents an important vulnerability in the security of open multi-agent systems employing argumentation as a means of communicative interaction. It is therefore necessary to investigate and develop strategies for detecting and defending against this type of malicious exploitation. However, such defense strategies will necessarily be insufficient; it would be impossible to construct a perfect defense against malicious resource exhaustion strategies in open multi-agent argumentation systems. Nonetheless, research into efficient means of minimizing the risk of such exploitation is warranted, as is continued investigation into further methods of malicious agent strategies in practical argumentation systems.

Acknowledgements

First and foremost, I would like to thank my supervisor, Jörg Denzinger, for the opportunity to work on this project, as well as his continued support and feedback during my investigation into this rather esoteric topic. I would also be terribly remiss to not thank my mother, Chyrelanne Kuipers, amidst the top of my acknowledgements; your passion for education has always been an insipration, and your kind words of encouragement and wisdom have helped me more than you'll know. To the other graduate students, both in my lab and otherwise, I thank you as well for a most pleasurable, and often quite entertaining, experience; an exhaustive list would simply be too long, but you know who you are! To both the faculty and staff in the department, through the classes I've taken and the administrative details you've taken care of, you've helped me immeasurably, and I thank you for that. Finally, I would like to thank my thesis committee for taking the time to scrutinize the research I've presented herein.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reasoning is a broad and pervasive topic throughout academia. In the sciences, both evidence-based reasoning and theoretical methods of reasoning form the cornerstone of scientific progress; in economics, statistical reasoning and game theoretic reasoning allow these complex systems to be understood and manipulated; in law, the judicial process is understood and applied through a process of reasoning. These are but a few examples, yet it should take no more than a moment's reflection to recognize myriad others, not only in academia but in business, politics, religion and the activities of our everyday lives. Reasoning is among the most powerful and essential cognitive instruments available to us; through it, understanding, explanation, investigation and interaction may be achieved, both formally and informally.

Formal applications of reasoning have traditionally focused heavily on demonstrative methods; that is to say, proof. Absolute and irrefutable truth is the consequence of proof, and as such, proof is highly desirable. In the metaphysical constructs of mathematics and formal logics, proof is both available and necessary, as these systems are rooted in absolutes. Those systems residing outside metaphysics, the informal and natural systems, do not so readily admit such absolutism. Our interpretation of these systems, and the formalisms we construct to reason over and understand these systems, must therefore respect this lack of absolutism. The phenomenal necessitates an insufficiency of interpretation.

Despite our inability to employ demonstrative proof within this context, reasoning is nonetheless possible, as evidenced by a vast history of non-demonstrative reasoning encompassing millennia of scientific inquiry and achievement. In the absence of proof and the absolutism of truth, scientific progress has relied on a method of reasoning commonly held to be in the realm of disputes that are decidedly non-scientific: argumentation. The study of argumentation has a rich and expansive history in the philosophical traditions, although it has often been viewed narrowly as the application of mere rhetoric and sophistry. However, argumentation forms the cornerstone of non-demonstrative reasoning; at its core, the study of argumentation is the investigation into the interaction between unprovables and the nature of justification.

As with argumentation, the study of artificial intelligence has been rife with misunderstanding, both in its popularization and occasionally even by its practitioners. In essence, artificial intelligence is concerned with applying computational resources towards exceedingly difficult problems in order to produce reasonable solutions within a reasonable amount of time. As such, the field is more than familiar with insufficiency and the lack of absolutes; achieving a “perfect” solution is generally infeasible for these problems, and often impossible. Whether the task is to dynamically route international air traffic, automatically test a complex software suite or convincingly interact with human beings, the information upon which the system must make its decisions is generally imperfect, incomplete or inconsistent. With the addition of realistic time constraints, perfection is simply unreasonable, regardless of the computing power employed.

Formal methods of reasoning have therefore naturally found application in the field of artificial intelligence. Traditionally, the focus has been on demonstrative forms of reasoning; constructing a proof of a particular logical or mathematical theorem is no trivial matter, and the automation of this task is of immense value to anyone requiring such a proof. Given this capacity for reasoning over a knowledge base, automated theorem provers have since been adapted as a means of manipulating knowledge in agent based systems. In such systems, agents often gather knowledge from their environment with the aim of using this knowledge to support decision making procedures. However, the strict demonstrative nature of these proof-based reasoning mechanisms is often insufficient given the incomplete, imperfect and often inconsistent knowledge an agent gathers from its environment.

To handle this insufficiency, attention has now been turned to automating non-demonstrative forms of reasoning. Defeasible and non-monotonic logics were developed to formalize exception-based reasoning, yet the most fruitful of these investigations into automating non-demonstrative reasoning is based on the most pervasive of its natural forms: argumentation. By drawing inspiration from how humans deal with incomplete, imperfect and inconsistent knowledge, a powerful means of automating non-demonstrative reasoning was born. The core of argumentation is also the principle distinction between it and other forms of automated reasoning, non-demonstrative or otherwise; rather than constructing a straight line of proof from evidence (or premises) to conclusions, argumentation focuses on the dialectic interaction between reasons for and against particular conclusions. By modeling the process of justification as a dialogue between one side for and another against a particular claim, automated argumentation provides not only a powerful means of automating non-demonstrative reasoning, but also one which can be understood naturally and efficiently by humans interacting with these systems.

With the rapid expansion and adoption of the Internet in the past few decades, the field of agent-based artificial intelligence has become increasingly interested in employing agents as a means of automating interactions between systems in this vast network. Rather than being simple pieces of software executing mundane tasks according to rigid protocols, such as e-mail or web browsers, the goal is to have these agents performing complex tasks based on sophisticated reasoning mechanisms, such as automated commercial negotiations or information search and retrieval. Given the nature of the information dealt with by agents in this on-line environment, automated argumentation is being investigated as a means of communicative interactions between agents engaged in activities both across the Internet and in other "open" multi-agent systems.

However, the additional value gained by agents employing automated argumentation in open multi-agent systems is not without consequences. While the Internet brought with it the value of a rich means of high-speed, long-distance communication, so too did it bring the potential for malicious abuse of this communication medium. Digital virii and other forms of malicious software are now able to propagate with increasing efficiency, and attempts to protect against these threats has resulted in a protracted conflict between the two sides. For each new defense against virii and malware, new attacks are developed for which this defense is inadequate. As automated argumentation is adopted as a means of interaction in open multi-agent systems such as the Internet, a similar situation will arise between agents employing forms of malicious argumentation and the defenses developed to counteract them.

The purpose of this thesis is to introduce the concept of malicious argumentation in the context of open multi-agent argumentation systems. When arguments are constructed out of formulae in an underlying formal logical language, many decisions in the argumentation system are based on deciding deduction or consistency in this logical language. For interesting logics, these decisions are generally intractable; often they cannot be decided within a reasonable time limit. Given that actions in multi-agent systems need to be resource bounded, the decisions of the argumentation system are also restricted by resource limitations. However, many of the decisions performed in an argumentation system are based on external data: the arguments presented by an agent's opponent. Therefore, a malicious agent may construct its arguments in such a way as to exhaust the resource bounds of its opponent's decision procedures, thereby altering the outcome of these decisions. This may allow a malicious agent to manipulate the outcome of the process of justification towards its own ends, which presents an important vulnerability in the security of open multi-agent argumentation systems.

The goal of this thesis is to provide an overview of the problem presented by malicious argumentation in open multi-agent systems, both by describing the concept from a high-level perspective as well as providing a practical demonstration of a malicious resource exhaustion strategy. Further, it is to provide a convincing account of the necessity of implementing a defense strategy to counter-act malicious argumentation, and conversely the necessary inadequacy of any such defense strategy. This thesis is organized into four primary chapters: Chapter 2 provides an introduction to the field of automated argumentation, Chapter 3 provides background information on automated theorem proving, Chapter 4 investigates the concept of malicious argumentation and provides a concrete example of a resource exhaustion strategy, and finally Chapter 5 provides a high-level discussion of defense strategies against malicious argumentation.

Chapter 2

Argumentation

This chapter is intended to provide the reader with some background knowledge in the field of automated argumentation. While not an exhaustive examination of the subject, this should provide sufficient knowledge for understanding the topic of this thesis. The chapter is organized as follows: in Section 2.1, the classifications of demonstrative and non-demonstrative reasoning are described, providing motivation for the basic goals and mechanisms of argumentation. In Section 2.2, abstract argumentation is introduced, which provides the general model for a system of arguments and relations between them. Further, this section examines the principles of argument evaluation, whereby the justification status of an argument is determined in relation to the other arguments in the abstract argumentation framework. Section 2.3 then looks at arguments as structures built out of formulae in an underlying formal logical language, rather than simple abstract entities. Included in this section is a definition of attack relation semantics, which provides a mechanism by which attack relations between structured arguments may be decided. Section 2.4 then examines argumentation as a means of interaction between agents in a multi-agent system, including the use of dialogue games to control the interaction, the relation between dialogue game rules and the semantics of argument evaluation described earlier, and the particular challenges presented by argumentation in open multi-agent systems. Finally, Section 2.5 describes the necessity of resource limitations in practical implementations of argumentation systems, which will form a focal point central to the research into malicious argumentation presented in this thesis.

2.1 Proof and Argument

Argumentation is a method of reasoning. Similar to proof, arguments are a way of connecting evidence to conclusions through the use of specific rules. However, an argument for a particular conclusion does not constitute a proof of that conclusion. A proof is formed through the valid application of inference rules upon true premises (axioms) in order to demonstrate the truth of a conclusion. If the proof is both valid and sound in this way, then the truth of the conclusion cannot be disputed. Proof is therefore classified as a *demonstrative* method of reasoning, as a sound and valid proof indisputably demonstrates the truth of its conclusion. Furthermore, proof is classified as *monotonic* with respect to set inclusion on the knowledge base, as the addition of new knowledge cannot retract previous conclusions that have been successfully proven.

In contrast, argumentation is a *non-demonstrative* method of reasoning. While arguments connect evidence to conclusions through the application of valid inference rules, neither the conclusions nor the evidence used are indisputable. Rather, the evidence is considered to be assumptions, not axioms, used to support the conclusion. In this way, a conclusion is *warranted* rather than proven through a process of argumentation. Given the possibility of disputation over the conclusion of a given argument, argumentation is itself classified as *non-monotonic* with respect to set inclusion on the knowledge base, as the addition of new knowledge may retract conclusions which were previously warranted.

Given this classification, it may seem that proof is a superior form of reasoning to argumentation, as it may draw indisputable conclusions from evidence. The conditions under which proof may operate, however, are far more restrictive than those under which argumentation is possible. When knowledge is incomplete, imperfect or inconsistent, an indisputable proof of a conclusion is impossible, yet arguments for the conclusion may nonetheless be constructed. Under such conditions, it may be that arguments can be constructed both for and against a particular conclusion; in this way, the conclusion may be disputed. Rather than being an undesirable side-effect, however, dispute is central to argumentation.

While proof is able to indisputably determine the truth of a conclusion, argumentation cannot, as the conclusion of arguments may always be disputed by subsequent arguments. Instead, the goal of argumentation is to determine the warrant, or *justification status*, of a conclusion through a process of disputation [Lou98]. Where a proof of the absolute truth of a given conclusion with respect to an incomplete, inconsistent or imperfect knowledge base is impossible, a process of argumentation may determine that a given conclusion is more justified than its logical complement. As may be evident from the term itself, then, an argument's justification status is generally not a simple boolean property, but rather a member of an ordered set describing levels of justification. There are, however, many different interpretations on how the justification status of a claim is determined, inasmuch as there are many different systems of argumentation in general.

Regardless of the semantics of justification used by a particular argumentation system, all argumentation systems employ a common process to determine the justification status of a claim: the *dialectic proof procedure*. This procedure is framed as an interaction between two parties, a proponent and an opponent, wherein the proponent puts forward arguments in support of the claim, and the opponent in turn puts forward arguments against the claim. As the interaction proceeds, the proponent's arguments may be either in direct support of the claim, or they may be arguments against the arguments put forward by the opponent, in order to defend the claim. Similarly, the opponent may either focus its arguments against the claim itself, or against the defense arguments of the proponent, in order to defend its attack against the claim. The termination conditions of the interaction, as well as the justification status of the claim computed by the interaction and the legal moves available to each side during the interaction, are all determined by the particular argumentation system in use.

2.2 Abstract Argumentation

Despite the particular details of an argumentation system, the justification status of a claim is determined as a property of the relations between arguments put forth by the proponent and opponent of the claim, rather than as a property of the content of these arguments. Towards this end, the justification status of a claim is computed in terms of an *argumentation framework*, which provides a layer of abstraction between the content of arguments and the semantics of justification. In an argumentation framework, arguments are abstract entities whose role is determined by their relation to other arguments. The seminal argumentation framework proposed by Phan Minh Dung [Dun95] is built around the irreflexive binary *attacks* relation between arguments.

Definition. An *argumentation framework* is a pair $\mathcal{AF} = \langle \mathcal{AR}, attacks \rangle$, where:

1. $\mathcal{AR}$ is the set of all arguments
2. $attacks \subseteq \mathcal{AR} \times \mathcal{AR}$ is a binary relation between arguments

The argumentation framework then describes a directed attack graph between abstract argument entities, where $attacks(A, B)$ is read as "argument A attacks argument B" (for $A, B \in \mathcal{AR}$). Similarly, a set of arguments $S \subseteq \mathcal{AR}$ attacks an argument $B \in \mathcal{AR}$ if $\exists A \in S$ such that $attacks(A, B)$.

Example 1. Let $\mathcal{AF}_{ex} = \langle \mathcal{AR}_{ex}, attacks_{ex} \rangle$ be an argumentation framework, where:

$\mathcal{AR}_{ex} = \{\ a,\ b,\ c,\ d,\ e\ \}$

$attacks_{ex} = \{\ (c,\ a),\ (c,\ b),\ (e,\ a),\ (e,\ b),\ (b,\ e),\ (b,\ c),\ (d,\ c)\ \}$

A visual representation of the attack relation graph can be seen in Figure 2.1.

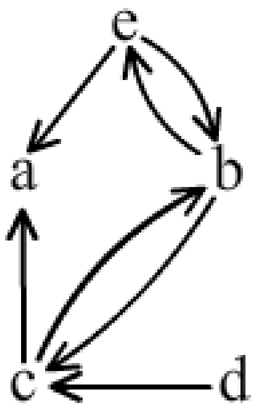

Figure 2.1: Argumentation Framework Example

2.2.1 Argumentation Semantics

As mentioned in Section 2.1, the goal of argumentation is to determine the justification status of a claim with respect to a given knowledge base. The justification status of an argument is determined through a process of *argument evaluation*, which is based on a particular argumentation semantics. The *argumentation semantics* defines a formal method, either procedurally or declaratively, by which an argument's justification status may be evaluated with respect to an abstract argumentation framework. The justification status of an argument may be a boolean property (justified or not), but is more often a complex valuation, such as in [PSJ98], to allow for a more fine-grained comparison between arguments.

There are two principal styles of defining argumentation semantics: either *extension-based* semantics, or *labelling-based* semantics [BG09]. An extension-based semantics for an argumentation framework $\langle \mathcal{AR}, attacks \rangle$ defines a set of *extensions*, where each extension defines a particular subset of $\mathcal{AR}$ based on the *attacks* relations between arguments in $\mathcal{AR}$. Arguments are then tested for membership in these extensions, and a hierarchical organization of a semantics' extensions may be used to define the complex valuation domain for an argument's justification status. Conversely, a labelling-based semantics defines a set of *labellings*, where each labelling defines a set of labels $\mathbb{L}$ and a function $L : \mathcal{AR} \rightarrow \mathbb{L}$ which assigns labels to arguments. The justification status of an argument is then derived from the labels assigned to the argument by the various labellings of the semantics. Of the two styles of argumentation semantics, the extension-based method is by far the predominant one.

2.2.2 Preferred, Grounded and Complete Extensions

In his seminal work on argumentation frameworks [Dun95], Phan Minh Dung defined a number of concepts which form the basis for many extension based argumentation semantics. These concepts are then used to define some of the fundamental extension based argumentation semantics: the *preferred extension*, which defines the credulous argumentation semantics, and the *grounded extension*, which defines the skeptical argumentation semantics. Further, Dung defines the *complete extension*, which he uses to provide a "link" between the preferred and grounded extensions.

The following concepts are defined in terms of an argumentation framework $\mathcal{AF} = \langle \mathcal{AR}, attacks \rangle$.

Definition. A set of arguments $S \subseteq \mathcal{AR}$ is *conflict-free* iff there does not exist $A, B \in S$ such that $attacks(A, B)$.

Intuitively, then, a conflict-free set of arguments is simply a set of arguments which do not attack each other.

Example 2. Considering the argumentation framework $\mathcal{AF}_{ex}$ from Example 1, we can then identify the following conflict-free sets:

$\{ a, b \}, \{ a, d \}, \{ b, d \}, \{ a, b, d \}, \{ c, e\}$

Note that sets containing single arguments ($\{\ a\ \}$, $\{\ b\ \}$, ...) are also conflict-free, although trivially so.

Definition. An argument $A \in \mathcal{AR}$ is *acceptable with respect to* a set of arguments $S \subseteq \mathcal{AR}$ iff for all $B \in \mathcal{AR}$ such that $attacks(B, A)$, there exists $C \in S$ such that $attacks(C, B)$

An argument is then acceptable with respect to a set of arguments just in case the argument is defended by that set; that is to say, if any argument attacking A is in turn attacked by an argument in S, then A is acceptable with respect to S.

Example 3. Given the argumentation framework $\mathcal{AF}_{ex}$ from Example 1, for a subset of $\mathcal{AR}_{ex}$ such as $S = \{\ b,\ d\ \}$, we can see that a is acceptable w.r.t S, as all arguments which attack a (the arguments c and e) are attacked by arguments in S. Further, b and d are both acceptable w.r.t S, as all arguments attacking b and d are attacked by arguments in S.

Definition. A set of arguments $S \subseteq \mathcal{AR}$ is *admissible* iff S is conflict-free, and $\forall A \in S$, A is acceptable with respect to S

An admissible set of arguments can then be understood as a set of arguments which do not attack each other, and also defend the set against attacks from arguments outside the set.

Example 4. For the argumentation framework $\mathcal{AF}_{ex}$ of Example 1, we can identify the following admissible sets:

$\{\ b\ \}$, $\{\ d\ \}$, $\{\ e\ \}$, $\{\ a,\ b\ \}$, $\{\ a,\ d\ \}$, $\{\ a,\ b,\ d\ \}$

Definition. The *characteristic function* of an argumentation framework $\mathcal{AF}$ is a function $F_{\mathcal{AF}} : 2^{\mathcal{AR}} \rightarrow 2^{\mathcal{AR}}$, defined as:

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

Using the above concepts, we can now define the preferred extension, which is used to define the credulous argumentation semantics, and the grounded extension used to define the skeptical argumentation semantics.

Definition. A *preferred extension* of an argumentation framework $\mathcal{AF}$ is an admissible set $S \subseteq \mathcal{AR}$ which is maximal with respect to set inclusion

Example 5. Using the argumentation framework defined in Example 1, by examining the admissible sets identified in Example 4, it should be readily apparent that the set $\{\ a,\ b,\ d\ \}$ is the only preferred extension of $\mathcal{AF}_{ex}$.

Definition. The *grounded extension* of an argumentation framework $\mathcal{AF}$ is the least fixed point of the characteristic function $F_{\mathcal{AF}}$, denoted $GE_{\mathcal{AF}}$

Example 6. Once again using the argumentation framework $\mathcal{AF}_{ex}$ of Example 1, the grounded extension $GE_{\mathcal{AF}_{ex}}$ can be computed through the following procedure:

$F_{\mathcal{AF}_{ex}}(\ \emptyset\ ) = \{\ d\ \}$

$F_{\mathcal{AF}_{ex}}(\{\ d\ \}) = \{\ d\ \}$

and so $GE_{\mathcal{AF}_{ex}} = \{\ d\ \}$

As can be seen from Examples 5 and 6, the grounded extension is a far more restrictive notion of argument acceptability than the preferred extension. Intuitively, this is because the grounded extension needs to be "grounded" in a set of arguments which are acceptable with respect to the empty set, and therefore do not need to be defended by other arguments, whereas the preferred extension is based on the idea of finding the largest set of arguments which provide collective defense for one another. The grounded and preferred extensions can then be seen as intuitively relating to the concepts of skepticism and credulity respectively, from which their respective semantics glean their names.

Definition. An admissible set of arguments $S \subseteq \mathcal{AR}$ is a *complete extension* iff for every argument $A \in \mathcal{AR}$ which is acceptable w.r.t S, $A \in S$. Defined in terms of the characteristic function $F_{\mathcal{AF}}$, S is a complete extension iff $F_{\mathcal{AF}}(S) = S$.

The grounded, preferred and complete extensions are then related by the following properties [Dun95]:

1. Every preferred extension is a complete extension, but not every complete extension is a preferred extension.

2. The grounded extension is the smallest complete extension with respect to set inclusion.

3. The set of complete extensions of an argumentation framework form a complete semilattice with respect to set inclusion.

2.3 Structured Arguments

Abstract argumentation provides a means to evaluate systems of arguments, yet it is notably abstracted away from the content of the arguments themselves. For practical purposes, it is useful to represent arguments as structures built out of sentences in a particular language, rather than as abstract argument entities. In this way, relations between arguments may be computed on the basis of the content of arguments, making use of relations between sentences in the language from which arguments are constructed. While sentences in natural languages could be used to form the content of structured arguments in automated argumentation, the meaning of and relations between natural language sentences cannot be easily computed. Instead, arguments used in automated argumentation are constructed out of sentences in formal logical languages, in which the meaning of sentences and relations between them may be precisely defined and computed.

Natural language argumentation makes use of well-defined argument structures, such as the *syllogism* and *enthymeme*. The structures used in automated argumentation are generally inspired by those used in natural language argumentation, albeit defined formally and often in terms of the underlying logical language. While the structural details of arguments may vary between different formal systems of automated argumentation, all share a common structural feature of distinguishing a *conclusion* of the argument, and the premises which *support* the conclusion. Many argumentation systems [PSJ98, APM00, AMP00, BH01, PWA03, BH05] make use of the following argument structure:

Definition. Let $\mathcal{L}$ be a formal logical language, where $\Sigma_{\mathcal{L}}$ is the set of all formulae which may be constructed out of $\mathcal{L}$.

An *argument* is a pair $\langle \Phi, \alpha \rangle$ where $\Phi \subseteq \Sigma_{\mathcal{L}}$ is a set of formulae supporting a conclusion formula $\alpha \in \Sigma_{\mathcal{L}}$, such that:

1. The support set Φ is consistent: $\Phi \nvdash \bot$
2. The conclusion α is a logical consequence of Φ: $\Phi \vdash \alpha$
3. Φ is a minimal set satisfying (1) and (2): there is no $\Phi' \subset \Phi$ such that $\Phi' \nvdash \bot \wedge \Phi' \vdash \alpha$

In an argumentation system, arguments are usually drawn from a knowledge base $\Delta \subseteq \Sigma_{\mathcal{L}}$, as it is often undesirable to allow arguments to be constructed out of any formulae expressible in the underlying language. For a given argument $\langle \Phi, \alpha \rangle$ and knowledge base Δ, it is generally the case that $\Phi \subseteq \Delta$, yet $\alpha \notin \Delta$; an argument's support should be based on existing knowledge, yet the conclusion is generally new knowledge derived from the existing knowledge.

Further, it is not necessary that the knowledge base Δ is consistent. In fact, it is generally assumed that $\Delta \vdash \bot$, as one of the primary reasons for employing argumentation techniques is to deal with inconsistent knowledge. Using the above definition of an argument, it is nonetheless necessary that the support set $\Phi \subseteq \Delta$ is consistent, even if Δ is not. If it were allowable that $\Phi \vdash \bot$, then an inconsistent support set Φ could be used as support for any conclusion, since if $\Phi \vdash \bot$, then for all $\alpha \in \Sigma_{\mathcal{L}}$ such that $\Phi \vdash \alpha$. In general, then, the process of argumentation is used to identify consistent sets of support in an inconsistent knowledge base in order to justify conclusions outside the knowledge base. Since opposing arguments may be drawn from the same inconsistent knowledge base, the justification status of an argument cannot be determined on the basis of the content of the argument alone, but rather must be based on the relations to other arguments drawn from the knowledge base, as discussed in Section 2.2.1.

2.3.1 Attack Relation Semantics

In abstract argumentation, where arguments are merely abstract entities without structure or content, it is sufficient to simply enumerate the relations between arguments. This is useful when investigating the semantics of justification, or other properties of the argumentation framework, where the details of argument content are inconsequential, as it is the network of relations between arguments that is important in this context. However, in a practical argumentation system, where arguments are constructed out of formulae in an underlying logical language, the relations between arguments are based on relations between their composite formulae. For a sufficiently expressive underlying logical language, such as First Order Logic, there are an infinite number of possible arguments, and so these relations must be computed dynamically during argument evaluation.

The primary relation of interest in all argumentation systems is the *attacks* relation. Following the work of John Pollock [Pol91], most argumentation systems make use of two different kinds of attack relations: the *rebuttal*, and the *undercut*. Informally, a rebuttal is a reason for denying the conclusion of an argument, whereas an undercut is a reason for denying the connection between an argument's support and its conclusion. In the context of the argument structure presented in Section 2.3, these relations may be generally expressed as follows:

Definition. An argument $\langle \Phi, \alpha \rangle$ is a *rebuttal* of an argument $\langle \Psi, \beta \rangle$ iff α and β conflict.

Definition. An argument $\langle \Phi, \alpha \rangle$ is an *undercut* of an argument $\langle \Psi, \beta \rangle$ iff α and Ψ conflict.

Both rebuttal and undercut are based on the concept of *conflict* between composite elements of the arguments involved. The specific means of expressing conflict in particular argumentation systems is dependant on the underlying logic employed by the system. However, in general, composite elements (logical sentences) of arguments conflict if the elements together derive a contradiction. For example, consider Besnard and Hunter's definitions of rebuttal and undercut for a system of argumentation based on classical propositional logic [BH01] :

Example 7. An argument $\langle \Psi, \beta \rangle$ is a *rebuttal* for an argument $\langle \Phi, \alpha \rangle$ iff $\beta \leftrightarrow \neg\alpha$ is a tautology.

Let Δ be the knowledge base $\{\ a \wedge c,\ \neg b \rightarrow d,\ a \rightarrow \neg(b \vee c),\ \neg d\ \}$

Let A_1 be the argument $\langle\ \{\ a \rightarrow \neg(b \vee c),\ a \wedge c\ \},\ a \rightarrow \neg b\ \rangle$

A_1 can be verified as a valid argument by testing that:

1. The support set is consistent:

   $\{\ a \rightarrow \neg(b \vee c),\ a \wedge c\ \} \nvdash \bot$

2. The conclusion is a logical consequence of the support:

   $\{\ a \rightarrow \neg(b \vee c),\ a \wedge c\ \} \vdash (a \rightarrow \neg b)$

3. The support is minimal w.r.t set inclusion:

   there is no $\Phi' \subset \{\ a \rightarrow \neg(b \vee c),\ a \wedge c\ \}$ such that $\Phi' \vdash a \rightarrow \neg b$

Now, let A_2 be the argument $\langle\ \{a \wedge c,\ \neg b \rightarrow d,\ \neg d\},\ a \wedge b\ \rangle$

As with A_1, A_2 can be tested for validity in accordance with conditions 1, 2 and 3 above.

Finally, a rebuttal between A_1 and A_2 can be determined by testing whether $((a \wedge b) \leftrightarrow \neg(a \rightarrow \neg b)) \vdash \top$. A simplified proof of this is as follows:

1. $((a \wedge b) \leftrightarrow \neg(a \rightarrow \neg b)) \vdash \top$
2. $((a \wedge b) \leftrightarrow \neg(\neg a \vee \neg b)) \vdash \top$ (re-write consequence as disjunction)
3. $((a \wedge b) \leftrightarrow (a \wedge b)) \vdash \top$ (by De Morgan's law)
4. $\square$

Therefore, A_2 is a rebuttal of A_1 (and conversely, A_1 is a rebuttal of A_2, as rebuttals are symmetric attack relations).

While this definition formulates conflict between the conclusions as testing for a tautology ($\beta \leftrightarrow \neg\alpha \vdash \top$), this is equivalent to deciding the contradiction $\beta \leftrightarrow \alpha \vdash \bot$. As will be discussed in Chapter 3, expressing such conditions as the derivation of a contradiction rather than a tautology is more efficient when using a refutation based theorem prover.

Example 8. An *undercut* for an argument $\langle \Phi, \alpha \rangle$ is an argument $\langle \Psi, \neg(\phi_1 \wedge ... \wedge \phi_n) \rangle$ where $\{\phi_1, ..., \phi_n\} \subseteq \Phi$.

Let $\Delta = \{\ a \rightarrow b,\ a \wedge c,\ d \rightarrow \neg a,\ d\ \}$

Then the argument $\langle \{\ a \wedge c,\ a \rightarrow b\ \},\ b \rangle$ is undercut by the argument $\langle \{\ d,\ d \rightarrow \neg a\ \},\ \neg(a \wedge c) \rangle$

This definition of the undercut attack relation illustrates a more direct means of identifying conflict, as it does not rely on deduction. It is only necessary that the conclusion of the attacking argument is the negation of a subset of elements from the support of the attacked argument.

In contrast, consider Amgoud *et al*'s definition of the undercut relation, from [AMP00]

Example 9. An argument $\langle \Phi, \alpha \rangle$ is undercut by an argument $\langle \Psi, \beta \rangle$ iff there exists $\phi \in \Phi$ such that $\phi \equiv \neg\beta$

Let $\Delta = \{\ a \to b,\ a \wedge c,\ d \to \neg a,\ d\ \}$

Then the argument $\langle \{\ a \wedge c,\ a \to b\ \},\ b \rangle$ is undercut by the argument $\langle \{\ d,\ d \to \neg a\ \},\ \neg a \vee \neg c \rangle$

Note that these arguments in Example 9 do not satisfy the conditions for Besnard and Hunter's undercut relation described in Example 8, as it is not the case that the conclusion of the attacking argument is the negation of a conjunction of elements of the support of the attacked argument. Amgoud's definition is therefore a more inclusive definition of undercut, and similar to Besnard and Hunter's definition of rebuttal in Example 7, this definition relies on deduction, as deciding that $\phi \equiv \neg\beta$ is equivalent to deciding $\phi \leftrightarrow \neg\beta \vdash \top$, or rather, that $\phi \leftrightarrow \beta \vdash \bot$.

2.3.2 Attack Relations, Argument Evaluation and Deduction

In order to determine the justification status of an argument, it is necessary to compute attack relations between arguments, as the argument's justification status is determined as a property of the attack relation graph in the argumentation framework. As seen in Section 2.3.1, computing an attack relation between two arguments may involve deduction, such as in the rebuttal relation described in Example 7, in which it is necessary to test whether $\beta \leftrightarrow \alpha \vdash \bot$ to determine whether the conclusions of the arguments conflict.

It may be that testing for a particular attack relation between two given arguments does not require deduction, as in the undercut relation described in Example 8. Deduction is nonetheless a necessary component procedure of the argument evaluation process for argumentation systems in which the set of all arguments cannot be pre-computed. Given an argument $\langle \Phi, \alpha \rangle$ and a knowledge base Δ, argument evaluation involves searching for arguments $\langle \Psi, \beta \rangle$ such that $\langle \Psi, \beta \rangle$ *attacks* $\langle \Phi, \alpha \rangle$. If the *attacks* relation being used is, for instance, the undercut relation described in Example 8, this requires searching for a set $\Psi \subseteq \Delta$ such that $\Psi \nvdash \bot$ and $\Psi \vdash \beta$ where $\beta = \neg(\phi_1 \wedge ... \wedge \phi_n)$ such that $\{\phi_1, ..., \phi_n\} \subseteq \Phi$. While in simple cases it may be that $\beta \in \Delta$, such that $\langle\{\ \beta\ \},\ \beta\rangle$ *undercuts* $\langle\Phi, \alpha\rangle$, in general it is necessary to search for a consistent subset of Δ that deductively entails β. Therefore, regardless of whether deduction is used to define an attack relation, searching for attacking arguments during argument evaluation nonetheless requires deduction.

2.4 Argumentation in Multi-Agent Systems

As discussed in Section 2.1, the goal of argumentation is to determine the justification status of a claim with respect to a incomplete, inconsistent or imperfect knowledge base. Early investigations into automated argumentation [Dun95, Lou98, PS99] focused on using argumentation as a form of logic programming, where for a given knowledge base, the justification status of a claim could be computed by the dialectic proof procedure using particular argumentation semantics. Given that the field of automated argumentation draws inspiration from natural language argumentation, and further that the justification procedure is modelled after a dialogue between two parties, argumentation has naturally been adapted as a means of communicative interation between agents in a multi-agent system.

While multi-agent argumentation systems make use of concepts developed in earlier argumentation systems, there are nonetheless significant differences between these branches of automated argumentation. Although agents in multi-agent argumentation systems may assume the roles of *pro* and *con* in a given dialogue, these roles are not identical to the roles of pro and con in the dialectic proof procedure, particularly due to the information available to these different entities. In a multi-agent system, argumentative agents generally possess individual knowledge bases, and so agents involved in an argumentative interaction do not have complete access to all information which may be used to construct arguments during the interaction. Further, these agents generally possess individual goals, which may warrant their strategic manipulation of the argumentative interaction. When argumentation is implemented as a form of logic programming, however, both parties have access to the same information, and both have the same goal of determining the justification status of a claim.

2.4.1 Dialogue Games

The interactions between agents in a multi-agent argumentation system are controlled by a *dialogue game*, which defines rules regarding which "moves" may be made by participants at each stage of the interaction. The game consists of a set of two or more participants (agents), a set of locutions defining structured utterances which can be made by the participants, and a public commitment store containing the propositions which the various participants have committed to. Further, the interaction between the participants is controlled by a system of rules, which may be categorized as follows [MP02]:

- **Commencement Rules:** rules defining the conditions under which a dialogue may begin.
- **Locutions:** rules describing which utterances a participant may make, and the structure of these utterances.
- **Combination Rules:** rules defining the conditions under which particular locutions may be permitted.
- **Commitments:** rules defining the conditions under which a participant expresses commitment to a proposition.

- **Termination Rules:** rules describing the conditions which cause the interaction to end.

While the commencement and termination rules control the dialogue itself, the locution, combination and commitment rules control the individual moves made by players within the dialogue. A *move* in the dialogue is often defined as a structure containing a locution rule describing the structure of the utterance, combination rules describing the conditions under which the move is allowed, and commitment rules describing how the participants commitment stores are updated. For example, consider the following definition of the *assert* locution from [AMP00], wherein participant P is addressing participant C:

Example 10. assert(p) - where p is a propositional formula

Rationality - the player uses its argumentation system to check if there is an acceptable argument for p

Dialogue - the other player can respond with:

1. accept(p)
2. assert($\neg p$)
3. challenge(p)

Update - $CS_i(P) = CS_{i-1}(P) \cup \{p\}$ and $CS_i(C) = CS_{i-1}(C)$

In this example, the locution rule states that the utterance is structured as the term "assert" followed by a propositional formula surrounded by parentheses. The combination rules for this locution include the *rationality* rule, which states the conditions under which this locution may be performed, and the *dialogue* rule, which describes locutions the other agent may make in response to this locution. Finally, the participants' commitments are modified through an *update* rule, which describes how the commitment stores of each

Table 2.1: Walton and Krabbe's Dialogue Type Classification

Dialogue Type	Initial Situation	Participant's Goal	Dialogue Goal
Persuasion	Conflicting opinions	Persuade other participant	Resolve conflict
Inquiry	Need for proof	Find and verify evidence	Prove or disprove hypothesis
Negotiation	Conflicting desire for resources	Maximize resources attained	Reasonable distribution
Information Seeking	A participant lacks information	Give or receive information	Information exchange
Deliberation	Situation requiring action	Co-ordinate goals or actions	Find best course of action

player is updated as a result of the locution be uttered (where $CS_i(P)$ refers to the set of propositions committed to by player P at time i).

In addition to the rules used in dialogue games, different dialogue game types can be identified by higher level concepts such as the initial situation, the goal of the participants, and the overall goal of the dialogue. Walton and Krabbe [WK95] have identified a set of basic dialogue types, which include: *persuasion* dialogues, in which a participant attempts to convince its counterpart to accept a proposition, *inquiry* dialogues, wherein participants collectively attempt to answer a question, *negotiation* dialogues, in which participants argue over the division of resources, *information seeking* dialogues, wherein a participant seeks the answer to a question from others, and *deliberation* dialogues, in which participants collectively determine a course of action to take for a particular situation. The details of these different types of dialogues are summarized in Table 2.1, adapted from [AMP00].

2.4.2 Semantics of Assertion and Acceptance

In multi-agent argumentation systems, the goal of an interaction is generally more complex than simply determining the justification status of a claim, as discussed in Section 2.4.1. However, the process of determining the justification status of a claim nonetheless plays a role in multi-agent argumentation. When an agent receives an argument from its counterpart in a dialogue game, the agent must determine whether or not it will accept the argument. To make this decision, the agent needs to determine the justification status of the argument with respect to its knowledge base, using a particular argumentation semantics. Further, when asserting an argument to its counterpart, an agent also needs to test the justification status of the argument, in order to decide whether the argument is worth transmitting to its counterpart. In both cases, the dialectic procedure described in Section 2.2.1 may be used to determine the justification status of the argument, although the argumentation semantics used in each case are not necessarily equivalent.

The particular semantics used by an agent is referred to as the agent's *attitude*, which is further divided into an *assertion attitude* describing the conditions under which an agent may assert an argument, and an *acceptance attitude* describing the conditions under which an agent will accept an argument. While there are common terms used to refer to agent attitudes, such as *skeptical* or *credulous* acceptance attitudes, the particular semantics related to these attitudes varies from system to system, depending on the argumentation framework used by the system. These attitudes are often related in their general concepts, however, insomuch as a skeptical agent will generally be more restrictive than a credulous agent in the arguments they accept.

Parsons *et al.* define the following assertion and acceptance attitudes [PWA03] for arbitrary agents G and H engaged in a dialogue, either of which may be in the role of *pro* or *con*. These attitudes are defined in terms of a preference-ordered argumentation framework, which will be described briefly as well.

Example 11. Agent attitudes in a preference-ordered argumentation framework

Let $\mathcal{AF} = \langle \mathcal{A}(\Sigma), Undercut, Pref \rangle$

where $\mathcal{A}(\Sigma)$ is the set of arguments that can be built from the set of formulae Σ and $Undercut \subseteq \mathcal{A}(\Sigma) \times \mathcal{A}(\Sigma)$ is the binary undercut relation between arguments and $Pref$ is a (partial or complete) pre-ordering on $\mathcal{A}(\Sigma) \times \mathcal{A}(\Sigma)$

- For $A_1, A_2 \in \mathcal{A}(\Sigma)$, if $A_1 \gg^{Pref} A_2$ then A_1 is *stronger* than A_2
- For $A_1, A_2 \in \mathcal{A}(\Sigma)$, if A_2 *undercuts* A_1, then A_1 *defends itself* against A_2 iff $A_1 \gg^{Pref} A_2$
- For $S \subseteq \mathcal{A}(\Sigma), A \in \mathcal{A}(\Sigma)$, S *defends* A iff for all $B \in \mathcal{A}(\Sigma)$ such that B *undercuts* A and A does not defend itself against B then there exists $C \in S$ such that C *undercuts* B and B does not defend itself against C
- Let $\mathcal{F}(S) = \{ A \in \mathcal{A}(\Sigma) \mid S \; defends \; A \}$, where $S \subseteq \mathcal{A}(\Sigma)$
- The set of *acceptable* arguments S_Σ is the least fixpoint of the function $\mathcal{F}$

Let $\mathcal{A}(G, H) = \mathcal{A}(\Sigma_G \cup CS(H))$

where Σ_G is the knowledge base of agent G

and $CS(H)$ is the public commitment store of agent H

Assertion attitudes

- If G is *confident*, then it can assert any proposition α for which there is an argument $\langle \Phi, \alpha \rangle \in \mathcal{A}(G, H)$
- If G is *careful* then it can assert any proposition α for which there is an argument $\langle \Phi, \alpha \rangle \in \mathcal{A}(G, H)$ and no stronger argument $\langle \Psi, \neg\alpha \rangle \in \mathcal{A}(G, H)$
- If G is *thoughtful* then it can assert any proposition α for which there is an acceptable argument $\langle \Phi, \alpha \rangle \in \mathcal{A}(G, H)$

Acceptance attitudes

- If G is *credulous* then it can accept any proposition α previously asserted by H if $\langle \Phi, \alpha \rangle \in \mathcal{A}(G, H)$
- If G is *cautious* then it can accept any proposition α previously asserted by H for which there is an argument $\langle \Phi, \alpha \rangle \in \mathcal{A}(G, H)$ and there is no stronger argument $\langle \Psi, \neg\alpha \rangle \in \mathcal{A}(G, H)$
- If G is *skeptical* then it can accept any proposition α previously asserted by H for which there is an acceptable argument $\langle \Phi, \alpha \rangle \in \mathcal{A}(G, H)$

In comparison to the extensions described in Section 2.2.2, the assertion and acceptance attitudes also use extension-based semantics which are defined in terms of relations in the argumentation framework, rather than the content of arguments themselves. However, while the agent attitudes described above make use of terms such as *credulous* and *skeptical*, these are quite different from the credulous and skeptical semantics defined by Dung [Dun95]. For instance, the credulous argumentation semantics of Dung makes use of the preferred extension, which may intuitively be described as a maximal set providing collective defense for its elements. In contrast, the semantics used by the credulous agent attitude described above does not take into account attack relations between arguments, but rather only requires that an argument can be constructed for the proposition, resulting in a far more inclusive definition of credulous semantics. This is generally indicitive of the state of the field of automated argumentation, as commonly used terms are often re-defined to suit an author's particular purposes.

2.4.3 Argumentation in Open Multi-Agent Systems

An important distinction in multi-agent systems is the difference between closed and open multi-agent systems. In a closed multi-agent system, agents are designed and controlled by a single entity[1], and the agents run on trusted resources under the control of that entity. In contrast, an open multi-agent systems allows for agents that are designed and controlled by different entities, and the agents may run on untrusted and failure prone resources. Where in a closed multi-agent system agent interactions are ultimately governed by the design of the agents, in an open multi-agent system the agents' interactions may be governed only through protocol, as the agents may be designed by many different entities. Open multi-agent systems therefore allow for the possibility of malicious agents that pursue their own goals by subverting the intention of the system towards their own ends. Malicious agents are distinct from competitive agents in that, while competitive agents pursue their own goals, they nonetheless "play fair" with other agents, whereas malicious agents will attempt to exploit and subvert the system in order to achieve their goals.

Argumentation has been proposed as a means of interaction between agents in open multi-agent systems, for applications such as automated negotiations in E-Commerce systems [BN02a, BN02b] and as a general means of negotiating agent rights in open multi-agent systems [Alo04]. However, while security measures have been studied to counteract malicious agents in general open multi-agent systems [SC00], little attention has been paid to the unique challenges of malicious behaviour in open multi-agent argumentation systems. Recently, Rahwan and Larson [RL08] investigated the possibility of malicious agents strategically withholding information in order to manipulate the outcome of argumentative interactions. Further, the research of P.E. Dunne [Dun03] investigates the strategic manipulation of dialogue games, whereby a malicious agent may attempt to delay a particular conclusion until the resource bounds of the dialogue game have been exhausted, and therefore manipulate the outcome of the interaction. In this thesis, how-

[1]This is not to say that agents in a closed MAS are necessarily centrally controlled, but rather that they are instantiated and terminated by a single entity

ever, the focus is on malicious agent strategies designed to exploit the resource bounds of another agent in order to manipulate the outcome of the argumentation process.

2.5 Resource Bounded Argumentation

Agents interacting within an environment must make decisions as to which actions to perform in the situations they find themselves, and these decisions are based on a process of reasoning. The reasoning process, however, is non-trivial, and may be intractably complex due to the logics, knowledge and computations used to make these decisions. Nonetheless, agents need to make decisions in a timely fashion, as the situation may be dynamically changing during the reasoning process, and the results of a lengthy decision-making computation may be inapplicable upon completion. It is therefore necessary for practical purposes that agents' decision procedures are resource bounded [BIP88].

When argumentation is used in practical systems, then resource bounds must be imposed. Particularly when argumentation is used in a multi-agent system, either as a decision support mechanism [KM03] or as a means of communicative interaction with other agents [APM00, PSJ98], the amount of time consumed by the argumentation process must be limited in order for agents to act in a timely fashion. The specific resource limit may either be self-imposed by the agent, or dictated by protocol, as is typical when agents are engaged in communication.

The highest level at which resource bounds must be placed on the argumentation process is at the level of the dialogue game. Given the possibility of infinitely long, or even extremely long finite lines of argumentation, it is necessary to impose a limit on the resources consumed during the dialogue as a whole [Lou98]. In terms of the categorization of dialogue rules given in Section 2.4.1, such a resource limit would be formalized as a *termination rule* for the dialogue, although resource exhaustion isn't the only condition under which dialogue termination could occur[2].

The next level at which resource limitations must be considered concerns the locution decision procedure used by agents to determine which "move" to play during their turn of the dialogue. As discussed in Section 2.4.2, this procedure involves an agent testing the acceptability of their counterpart's previous move, and subsequently searching for an argument to respond with which is admissible in accordance to their assertion semantics. Depending on the particular argumentation semantics used for these purposes, an agent's locution decision procedure may involve lengthy or even intractable search processes. However, if this procedure is not bound by resource constraints, an agent could surpass the resource limit of the dialogue while deciding its next move, either unintentionally or strategically, preventing its counterpart from engaging in the dialogue. In the interest of fairness, then, it is necessary to evenly distribute resources between all parties engaged in the dialogue [Lou98], and therefore it is necessary to limit the resources consumed by agents' locution decision procedures during each turn of the dialogue game.

Finally, at the lowest level, it may be necessary for agents to impose resource limits on the attack relation decision procedure. If an agent is using a sufficiently expressive underlying logical language to express the content of arguments, such as first-order predicate logic [BH05], deciding deduction may be an intractable procedure. As discussed in Section 2.3.1, the attack relation is often based on deciding deductive relations between component formulae of arguments. Given that most argumentation semantics involve computing attack relations between numerous arguments, it is necessary to distribute the resources allocated to the argument evaluation procedure amongst the individual attack relation computations. While an agent may or may not use a fairness criteria to evenly distribute these resources, however, any one attack relation decision is maximally

[2]Rather, it is preferable for a dialogue to terminate "naturally" due to a particular state of the dialogue being reached rather than "un-naturally" due to resource exhaustion

bounded by the resource limit of the argument evaluation procedure, which is in turn bounded by the resource limit of locution decision during an agent's turn in the dialogue game.

Chapter 3

Automated Theorem Proving

This chapter is intended to provide the reader with enough background information on automated theorem proving to understand its use in the context of this thesis. The chapter is organized as follows: in Section 3.1, a general description of logics is provided, which includes a description of the logical language by which formulae are constructed, the truth domain onto which logical formulae are evaluated, and the semantics of this evaluation. In Section 3.2, the logical calculus is described, which provides the means of syntactic evaluation and manipulation of logical formulae. Section 3.3 then describes the relations of soundness and completeness between the semantic evaluation defined by the interpretations and the syntactic evaluation defined by the calculus. Finally, in Section 3.4, the search control is described, which provides the means by which the proof of theorems may be automated.

3.1 Logics

Informally, a logic is a means to relate formal syntax and semantics. Syntax deals with representation; the collection of symbols used, and the ways they can be combined together. In logics, the syntax is *formal*, in that the representational constructs must adhere to specific rules, and do not permit the breaking of these rules for artistic reasons or otherwise, as might be permitted in natural languages. Conversely, semantics deals with the meaning of these syntactic constructs; again, in logics, semantics is formalized. Whereas meaning in natural languages is often open to individual interpretation or ambiguity, the semantics of logics are based on strict formal rules. The meaning of an expression in a logic is determined by its relation to the truth-domain of the logic via interpretation. Interpretation, as it relates to logics, is once again a formalized notion, rather than the notion of interpretation used in natural languages. For each syntactic symbol or structure of a logic, an interpretation assigns either a member of the truth domain, the object domain, or a relation between these domains. While a logical expression can be evaluated in relation to a particular intended interpretation, it is often the case that we are interested in evaluating an expression in relation to many or all possible interpretations. For instance, it is often desirable to determine whether an expression is tautological or contradictory; that is to say, whether an expression is evaluated as true or false, respectively, under all possible interpretations. Given this informal description of logics, we now turn towards a formal definition.

Definition. A *logic* is represented as a triple ($\mathcal{L}$, $\mathcal{W}$, $\mathcal{I}$), where:

- $\mathcal{L}$ is the *language* of the logic, which defines the syntax of the logic, expressed as the set of all well-formed formulae.
- $\mathcal{W}$ is the *truth-domain* of the logic, a set of truth-values onto which formulae are evaluated.
- $\mathcal{I}$ is the set of all *interpretations* of the logic, defining the logic's semantics.

3.1.1 The Logical Language $\mathcal{L}$

The language of a logic defines the syntactic form of valid expressions in that logic. Similar to natural written languages, logical languages consist of expressions constructed out of sequences of symbols in accordance with particular rules. The set of all symbols from which these sequences are constructed is also referred to as an *alphabet*, which is divided into several classes denoting the role of each symbol. For convenience, symbols used in logics are often borrowed from the Roman and Greek alphabets, although it should be noted that these logics are in no way bound to those particular symbols; rather, it is important only that the different classes of symbols are unique in order to avoid ambiguity.

Definition. An *alphabet* is a set of symbols used by a language, divided into distinct classes of symbols. In general, logical languages can be described by the following ten classes of symbols:

1. *constants* (C) - symbols which reference a particular object in the domain, often denoted by lowercase letters at the beginning of the roman alphabet, ie: *a, b, c*
2. *variables* (V) - symbols which may reference any object in the domain, often denoted by lowercase letters at the end of the roman alphabet, ie: *x, y, z*
3. *functions* (F) - symbols referencing a mapping between n domain objects onto a single domain object, often denoted by lowercase roman letters such as *f, g, h*
4. *function variables* (FV) - symbols referencing any mapping between n domain objects onto a single domain object
5. *predicates* (P) - symbols referencing a mapping between n domain objects onto a single truth value, often denoted by uppercase roman letters such as *P, Q, R*
6. *predicate variables* (PV) - symbols referencing any mapping between n domain objects onto a single truth value
7. *interpreted predicates* (PI) - symbols referencing a mapping between n domain objects onto a single truth value, for which the interpretation is constant
8. *junctors* (J) - symbols referencing a mapping between n (where n is generally 1 or 2) truth values onto a single truth value, denoted by symbols such as $\vee,\ \wedge,\ \neg,\ \rightarrow,\ \leftrightarrow$
9. *quantifiers* (Q) - symbols referencing a construct which specifies a quantity of the object domain over which an open variable in a particular formula is assigned,

mapping onto a truth value; quantifiers are denoted by symbols such as $\exists$, $\forall$

10. *punctuation* - symbols used to denote groupings of other symbols, or to provide clarification and readability, such as (,) and ,

Example 12. *Propositional logic* is a logic which contains only predicate variables, referred to as *propositions*, and the basic set of logical junctors. Additionally, it may be convenient to defined a few interpreted predicates for a propositional logic, such as *true* and *false*. Since propositional logic does not make use of an *object domain*, there are no object domain variables, constants, functions, function variables, predicates or quantifiers.

The alphabet of propositional logic can be described by the following sets of symbols:

- Predicate variables $PV = \{\ p,\ q,\ r,\ ...\ \}$
- Interpreted predicates $PI = \{\ true,\ false\ \}$
- Junctors $J = \{\ \vee,\ \wedge,\ \neg,\ \ \rightarrow,\ \ \leftrightarrow\}$
- $C = V = F = FV = P = Q = \emptyset$

Example 13. *First-Order Predicate Logic* includes object domain variables and constants, as well as functions, predicates, the basic set of logical junctors, and the existential and universal quantifiers. While more robust than propositional logic, first-order predicate logic still does not permit the use of function or predicate variables.

The alphabet of first-order predicate logic can by the following sets of symbols:

- Constants $C = \{\ a,\ b,\ c,\ ...\ \}$
- Variables $V = \{\ x,\ y,\ z,\ ...\ \}$
- Functions $F = \{\ f,\ g,\ h,\ ...\ \}$

- Predicates $P = \{\ P,\ Q,\ R,\ ...\ \}$
- Junctors $J = \{\ \vee,\ \wedge,\ \neg,\ \ \rightarrow,\ \ \leftrightarrow\}$
- Quantifiers $Q = \{\ \exists,\ \forall\ \}$
- $PV = PI = FV = \emptyset$

Sequences of symbols from the alphabet of a logic are referred to as *expressions* in that language. However, not all expressions are valid; the set of all expressions in a language is the set of all possible sequences of the symbols of that language's alphabet. Those expressions which are valid in a given language are referred to as the *formulae* of that language.

Definition. The *language* $\mathcal{L}$ of a logic is the set of all formulae of that logic, where these formulae consist of a sequence of symbols from an *alphabet* composed in accordance with specific *formation rules.*

The formulae of a logical language $\mathcal{L}$ are constructed in accordance with certain formation rules. These formation rules are used to define two groups of syntactic constructs: the *terms*, which are to be evaluated to elements of the object-domain, and *well-formed formulae*, which will evaluate to elements of the truth-domain. The set $\mathcal{L}$ consists only of well-formed formulae, where the terms of a language are constituent parts of the well-formed formulae.

Definition. A *term* is defined recursively as follows:

- Any variable symbol $v \in V$ is a term
- Any constant symbol $c \in C$ is a term
- If $f \in F$ is an n-ary function symbol and $t_1,\ ...,\ t_n$ are terms, then $f(t_1,\ ...,\ t_n)$ is a term

- If $g \in FV$ is an n-ary function variable symbol and $t_1,\ ...,\ t_n$ are terms, then $g(t_1,\ ...,\ t_n)$ is a term

Definition. A *well-formed formula* (wff) is defined recursively as follows:

- If $R \in P \cup PI$ is an n-ary predicate symbol and $t_1,\ ...,\ t_n$ are terms, then $R(t_1,\ ...,\ t_n)$ is a wff (these particular wffs are also known as *atoms*)
- If $R \in PV$ is an n-ary predicate variable symbol and $t_1,\ ...,\ t_n$ are terms, then $R(t_1,\ ...,\ t_n)$ is a wff
- If $\star \in J$ is an n-ary junctor and $\alpha_1,\ ...,\ \alpha_n$ are wffs, then $\star(\alpha_1,\ ...,\ \alpha_n)$ is a wff [1]
- If α is a wff, $x \in V \cup FV \cup PV$ is a variable, and $\Box \in Q$ is a quantifier, then $\Box\ x \bullet \alpha$ is a wff

Example 14. In propositional logic, the sets of interpreted predicates PI and predicate variables PV contain only 0-ary symbols. Using the alphabet described in Example 12, consider the following expressions:

1. $p \vee q \wedge (r \rightarrow true)$
2. $\rightarrow (\wedge(q, \neg(r)), \vee(q, p))$
3. $p(r) \rightarrow (q \vee r)$
4. $r \leftrightarrow \neg q(p \vee r)$

Expressions 1 and 2 are valid well-formed formulae, where as 3 and 4 are not. Note that propositional logic does not contain any terms, as it does not use an object domain, but rather has only symbols which evaluate onto the truth domain.

[1] note that unary junctors are often written in prefix notation as $\star\alpha_1$, and binary junctors often are written in infix notation as $\alpha_1 \star \alpha_2$

3.1.2 Normal Forms

When implementing a theorem prover, it is often necessary to transform formulae from their general form as described by the logical language into a specific *normal form*, in order to facilitate the particular calculus being used. In resolution-based theorem proving for first-order predicate logic, formulae are generally converted to clausal normal form, in which clauses are disjunctions of literals, and sets of clauses represent the conjunction of their elements.

Definition. A *literal* is either an atom or the negation of an atom, where the former is known also as a *positive literal* and the latter as a *negative literal.*

Definition. A *clause* is a formula of the form:

$\forall\, x_1, ..., x_n\ (L_1\ \vee\ ...\ \vee\ L_m)$

where $L_1,\ ...\ , L_m$ are literals (positive or negative) and $x_1,\ ...\ , x_n$ are all variables occuring in these literals.

It is often convenient to collect together positive and negative literals in the clausal representation, and represent the clause as a conditional:

$\forall\, x_1, ..., x_n\ (A_1 \vee\ ...\ \vee A_i \vee\ \neg B_1 \vee\ ...\ \vee \neg B_j)$

$\forall\, x_1, ..., x_n\ (A_1 \vee\ ...\ \vee A_i \leftarrow B_1 \wedge\ ...\ \wedge B_j)$

3.1.3 The Truth Domain $\mathcal{W}$

The *truth domain* $\mathcal{W}$ is the set of values onto which formulae in a logic are evaluated. Commonly, logics make use of the boolean truth domain $\{\ true,\ \ false\ \}$, although in general they are not restricted to just this domain. Many-valued logics may introduce additional truth values to the boolean domain, such as *unknown*, *undecidable*, etc., or they may make use of infinite truth domains, such as the interval of real numbers [0,1] used in many probabilistic logics. Further, some many-valued logics express the truth-domain as complex structures such as trees, graphs or lattices, in order to express complex relations between the various truth values.

3.1.4 The Interpretations $\mathcal{I}$

An interpretation is a method of assigning meaning to symbols in a logical language, thereby creating the semantics for a logic. Symbols are interpretted by assigning to them either a member of an object-domain, a member of the truth domain, or a mapping between these domains.

Definition. Formally, an *interpretation* I consists of:

1. A non-empty set $\mathcal{D}$, known as the *object domain* of the interpretation
2. An assignment of an element of $\mathcal{D}$ to each constant in C
3. An assignment of an element of $\mathcal{D}$ to each variable in V
4. A mapping $\mathcal{D}^n \rightarrow \mathcal{D}$ for each n-ary function symbol in F
5. A mapping $\mathcal{D}^n \rightarrow \mathcal{D}$ for each n-ary function variable symbol in FV
6. A mapping $\mathcal{D}^n \rightarrow \mathcal{W}$ for each n-ary predicate symbol in P
7. A mapping $\mathcal{D}^n \rightarrow \mathcal{W}$ for each n-ary predicate variable symbol in PV
8. A mapping $\mathcal{D}^n \rightarrow \mathcal{W}$ for each n-ary interpreted predicate symbol in PI
9. A mapping $\mathcal{W}^n \rightarrow \mathcal{W}$ for each n-ary junctor in J
10. A combination rule for truth values in $\mathcal{W}$ for each quantifier $\Box$ in Q, such that $I(\Box\ x \bullet \alpha)$ is determined by combining the truth values of all formulae generated by substituting the variable x in α with an arbitrary element of $\mathcal{D}$, or the function variable x in α with an arbitrary function over $\mathcal{D}$ of the correct arity, or the predicate variable x in α with an arbitrary predicate with the correct arity

$\mathcal{I} = \{I_1, I_2, ...\}$ is then a set of interpretations, such that:

- $I_i(R) = I_j(R)$ for all interpretations I_i, I_j and all interpreted predicate symbols $R \in PI$
- $I_i(\star) = I_j(\star)$ for all interpretations I_i, I_j and all junctor symbols $\star \in J$
- $I_i(\Box) = I_j(\Box)$ for all interpretations I_i, I_j and all quantifier symbols $\Box \in Q$

Example 15. Using the alphabet defined for propositional logic in Example 12, consider the formula $\neg p \vee q$. Using the boolean truth-domain $\mathcal{W}_{bool} = \{true, false\}$, both predicate variable symbols p and q may be interpreted as $true$ or $false$.

The junctor $\neg$ is unary, interpreted as:

- $I(\neg(true)) = false$
- $I(\neg(false)) = true$

The junctor $\vee$ is binary, interpreted as:

- $I(\vee(false, false)) = false$
- $I(\vee(false, true)) = true$
- $I(\vee(true, false)) = true$
- $I(\vee(true, true)) = true$

The set of all possible interpretations for $\neg p \vee q$ is then : $\mathcal{I} = \{$

- $I_0 : I(p) = false, I(q) = false :$ $$\begin{aligned} I(\neg p \vee q) &= I(\vee(I(\neg(false)), false)) \\ &= I(\vee(true, false)) \\ &= true \end{aligned}$$

- $I_1 : I(p) = false, I(q) = true :$ $$\begin{aligned} I(\neg p \vee q) &= I(\vee(I(\neg(false)), true)) \\ &= I(\vee(true, true)) \\ &= true \end{aligned}$$
- $I_2 : I(p) = true, I(q) = false :$ $$\begin{aligned} I(\neg p \vee q) &= I(\vee(I(\neg(true)), false)) \\ &= I(\vee(false, false)) \\ &= false \end{aligned}$$
- $I_3 : I(p) = true, I(q) = true :$ $$\begin{aligned} I(\neg p \vee q) &= I(\vee(I(\neg(true)), true)) \\ &= I(\vee(false, true)) \\ &= true \end{aligned}$$

}

3.1.5 Models and Satisfiability

Interpretations provide meaning for formulae in a logic. Once meaning has been established, it is necessary to then examine what kind of questions about formulae in a logic we would want to answer. For a given formula in a particular logic, we are primarily interested in questions about how the formula relates to elements of the truth domain under the set of all interpretations $\mathcal{I}$. For a boolean logic, such as propositional logic, we are then interested in determining under which interpretations the formula evaluates to true, and conversely under which interpretations it evaluates to false.

Definition. If a formula $\alpha \in \mathcal{L}$ is true under a given interpretation $I \in \mathcal{I}$, then I is a *model* for α, expressed as $\models_I \alpha$.

When necessary, a superscript will be used with the model symbol to denote a particular logic under which an interpretation is a model for a particular formula, such that $\models_I^{\mathcal{S}} \alpha$ is to be interpreted as α *is true under interpretation* I *for a logic* $\mathcal{S} = (\mathcal{L}_{\mathcal{S}}, \mathcal{W}_{\mathcal{S}}, \mathcal{I}_{\mathcal{S}})$.

Definition. If there exists an $I \in \mathcal{I}$ such that $\models_I \alpha$, then α is *satisfiable*.

Definition. Conversely, if there does not exist an $I \in \mathcal{I}$ such that $\models_I \alpha$, then α is *unsatisfiable*; a formula which is unsatisfiable is also referred to as a *contradiction*, as it evaluates to false under every interpretation.

Definition. If every interpretation $I \in \mathcal{I}$ is a model for α, then α is a *tautology*, expressed as $\models \alpha$.

In addition to determining whether a particular formula is satisfiable, tautological, contradictory, etc., it is often desirable to determine consequence relations under interpretations. That is to say, whether a formula is true under all interpretations which are models for a specific set of formulae. Particularly, we are interested in determining this consequence relation with respect to a distinguished set of formulae for the logic, known as *axioms*.

Definition. Given a set of formulae $\Gamma \subseteq \mathcal{L}$ and a formula $\alpha \in \mathcal{L}$, if for all interpretations $I \in \mathcal{I}$ such that $\models_I \Gamma$ it is the case that $\models_I \alpha$, then α is a *theorem* of Γ, denoted $\Gamma \models \alpha$.

3.2 Logical Calculus

As stated in the previous section, the goal when using a logic is generally to evaluate the relation between a formula or set of formulae and the truth domain under all interpretations in $\mathcal{I}$. While it may be feasible to perform this evaluation under all the interpretations for formulae in a simple logic, for complex logics permitting an infinite or even large finite number of possible interpretations, such an evaluation is likely far too computationally complex. The use of interpretations for explaining logical evaluation is useful from a theoretical perspective, although for practical purposes it is insufficient as a means of efficiently evaluating formulae. There is, however, another approach by which formulae may be evaluated without the need for iterating over every possible interpretation in the logic. By successively applying truth-preserving *inference rules*, formulae may be re-written into a form for which the evaluation under all interpretations in $\mathcal{I}$ is known, thereby effectively evaluating the formulae without the need for iterating over every interpretation from $\mathcal{I}$. A system of inference rules, in combination with a distinguished set of formulae known as *axioms*, is referred to as a *logical calculus*.

Definition. For a given logic $(\mathcal{L}, \mathcal{W}, \mathcal{I})$, a *calculus* is a structure $(\mathcal{A}, \mathcal{R})$, where:

- $\mathcal{A} \subseteq \mathcal{L}$ is a distinguished set of formulae, known as the *axioms*
- $\mathcal{R}$ is a set of truth-preserving *inference rules*

3.2.1 Inference Rules

Rather than simply working with individual formulae, theorem proving deals with collections of formulae, such as the axioms, the supporting formulae and the set of formulae representing the theorem to be proved. In order to perform a proof, it is necessary that these various collections of formulae be structured in a particular *knowledge representation.* This representation may be as simple as a set containing all formulae in those collections, or it may have a more complex structure such as a set of sets, a tree, or otherwise. Given a particular knowledge representation, *inference rules* may be defined to perform syntactic manipulation of the knowledge encoded in that representation.

Definition. An *inference rule* is a transformation on a knowledge representation, taking the form: $\dfrac{premise}{conclusion}$

An inference rule is applied to a knowledge representation by first matching the premise of the rule against components of the knowledge representation, and then replacing those components matched by the premise with the conclusion of the rule to create a new knowledge representation. An inference rule is correct just in case it preserves the truth-value evaluation of the knowledge representation across the transformation. That is to say, if the initial knowledge representation is contradictory, then the knowledge representation resulting from the application of the inference rule must also be contradictory.

The purpose of inference rules is therefore to manipulate the syntax of the represented knowledge while preserving the semantics. The goal of theorem proving is to demonstrate that the theorem being proved is a semantic consequence of supporting formulae. This can be accomplished by identifying syntactic properties of the knowledge representation for which certain semantic properties are known. However, these syntactic properties may not be immediately identifiable, in which case it is necessary to apply inference rules until either the syntactic property in question can be easily identified, or it is impossible to apply further inference rules to the knowledge representation.

Definition. A *deduction* from Δ to Φ, represented as $\Delta \vdash \Phi$, is a means of establishing that Φ is a syntactic consequence of Δ.

Let $\Phi \subseteq \mathcal{L}$ and $\Delta \subseteq \mathcal{L}$ be sets of formulae.

Let Γ_0 be the initial knowledge representation of Φ and Δ, constructed in accordance with a particular calculus $\mathcal{C} = (\mathcal{A}, \mathcal{R})$, and $\Psi \in \mathcal{L}$ be a distinguished syntactic element representing the goal.

Let $\alpha \vdash_r \beta$ represent the application of the single inference rule $r \in \mathcal{R}$ to the knowledge representation α, the result of which is a new knowledge representation β.

If there exists a finite sequence of inference rules $r_1, ..., r_k$, where $r_i \in \mathcal{R}$ for $1 \leq i \leq k$, such that $\Gamma_0 \vdash_{r_1} \Gamma_1 \vdash_{r_2} ... \vdash_{r_k} \Gamma_k$ and $\Psi \in \Gamma_k$, then $\Delta \vdash \Phi$. Otherwise, if no such sequence exists, then $\Delta \nvdash \Phi$.

Where necessary, deduction using a particular calculus $\mathcal{C} = (\mathcal{A}, \mathcal{R})$ will be denoted $\vdash^{\mathcal{C}}$.

3.2.2 Resolution

The *resolution* inference rule is used in refutation-based theorem proving, whereby the theorem to be proven is negated and conjoined with the support, and the resolution inference rule is applied until either the empty clause is syntactically derived, or no further applications of the inference rule are possible. The goal is then to show that the initial knowledge representation (the support set conjoined with the negated theorem) is semantically contradictory, given that there exists a syntactic derivation to the empty set, which is easily identifiable and known to be evaluated as false under all interpretations.

Formally, the resolution rule for first-order predicate logic can be written as:

$$\frac{\alpha_1 \vee ... \vee \alpha_i \vee ... \vee \alpha_n \ , \ \beta_1 \vee ... \vee \neg\beta_j \vee ... \vee \beta_m}{\theta(\alpha_1 \vee ... \vee \alpha_{i-1} \vee \beta_1 \vee ... \vee \beta_{j-1} \vee \beta_{j+1} \vee ... \vee \beta_m \vee \alpha_{i+1} \vee ... \vee \alpha_n)}$$

where $\theta = mgu(\alpha_i, \beta_j)$

Essentially, the resolution inference rule is used to identify pairs of complementary literals α_i and $\neg\beta_j$ which are unifiable (unification will be discussed further in Section 3.2.3). If two formulae $\alpha_1 \vee ... \vee \alpha_n$ and $\beta_1 \vee ... \vee \beta_m$ are identified which contain such complementary literals, then the formulae can be re-written by removing the complementary literals α_i and $\neg\beta_j$ and combining what's left of the formulae by disjunction, and applying the unifier of α_i and β_j.

Example 16. Let $\Delta = \{\forall x \bullet (\neg P(x) \vee Q(x)), P(a)\}$ be a set of supporting formulae and $\Phi = \{\exists y \bullet Q(y)\}$ be the theorem to be proved, both in first-order predicate logic.

The initial knowledge representation is then:

$\Gamma_0 = \Delta \cup \neg\Phi = \{(\neg P(x) \vee Q(x)), P(a), \neg Q(y)\}$.

The distinguished syntactic element representing the goal is the empty set, $\Psi = \Box$.

By selecting the first two formulae of Γ_0 and applying the resolution rule to the complementary literals $\neg P(x)$ and $P(a)$ using the unifier $\{x \approx a\}$, this results in the formula $Q(a)$.

$\Gamma_1 = \{(\neg P(x) \vee Q(x)), P(a), \neg Q(y), Q(a)\}$.

The resolution rule can further be applied to the formulae $Q(a)$ and $\neg Q(y)$ using the unifier $\{y \approx a\}$, which results in the empty set $\Box$.

$\Gamma_2 = \{(\neg P(x) \vee Q(x)), P(a), \neg Q(y), Q(a), \Box\}$.

Since $\Psi \in \Gamma_2$, by refutation, $\{\exists y \bullet Q(y)\}$ has therefore been shown to be a consequence of the set $\{\forall x \bullet (\neg P(x) \vee Q(x)), P(a)\}$.

3.2.3 Unification

When a logic uses variables, those variables may be substituted by simple or complex terms, atoms or formulae, depending on the domain over which the variables vary. Inference rules often require a comparrison between components of the premises or conclusion, such as in Example 16 above, where the literals α_i and β_j need to be matched. Given formulae in a first-order predicate logic, such as $P(a)$ and $\neg P(x) \vee Q(x)$ in which a is a constant, x is a variable and P and Q are predicates, it is necessary to match $P(a)$ and $P(x)$ in order to execute the inference rule and achieve the result $Q(a)$. To accomplish this, a *substitution* needs to be applied to $\neg P(x) \vee Q(x)$ in which all occurances of the variable x are replaced with the constant a. The application of this substitution to the conditional would result in $\neg P(a) \vee Q(a)$, which can then be syntactically matched against the formula $P(a)$ in order to perform the resolution inference rule and derive the conclusion $Q(a)$. The process by which a substitution is found in order to make two or more terms or formulae equivalent is known as *unification*.

Definition. A *substitution* is a finite set $\theta = \{\ v_1 \approx t_1\ , ...,\ v_n \approx t_n\ \}$, where each v_i is a variable, each t_i is a term in which v_i does not occur, and $v_1, ..., v_n$ are each distinct. The application of a substitution θ to a formula α results in a formula α' in which all occurances of the variable v_i have been replaced with the term t_i, for all $1 \leq i \leq n$.

Example 17. Let $\alpha = P(x, y, f(x, a))$ and $\theta = \{\ x \approx g(y),\ y \approx b\ \}$.

Then $\theta\alpha = P(g(y), b, f(g(y), a))$.

Definition. Let $\theta = \{\ v_1 \approx t_1\ , ..., \ v_n \approx t_n\ \}$ and $\sigma = \{\ u_1 \approx s_1\ , ..., \ u_m \approx s_m\ \}$ be substitutions. The *composition* $\theta\sigma$ of θ and σ is the substitution obtained from:

$\{\ v_1 \approx \sigma\ t_1\ , ..., \ v_n \approx \sigma\ t_n,\ u_1 \approx s_1\ , ..., \ u_m \approx s_m\ \}$

by removing any $v_i \approx \sigma\ t_i$ for which $v_i = \sigma\ t_i$ and removing any $u_j \approx s_j$ for which $u_j \in \{v_1, ..., v_n\}$.

Definition. Let $S = \{\ s_1\ , ..., \ s_n\ \}$ be a finite set of expressions (terms, formulae, etc.) and θ be a substitution. θ is a *unifier* for S if θS is a singleton; that is to say, if $\theta s_1 = \theta s_2 = ... = \theta s_n$. A unifier θ for S is a *most general unifier* (mgu) for S if for all unifiers σ of S, there exists a substitution γ such that $\sigma = \theta\gamma$.

Example 18. $S = \{\ P(f(x), z),\ P(y, a)\ \}$ is unifiable by $\sigma = \{\ y \approx f(a),\ x \approx a,\ z \approx a\ \}$. However, $\theta = \{\ y \approx f(x),\ z \approx a\ \}$ is a most general unifier for S, as $\sigma = \theta\{x \approx a\}$, and there exists a suitable γ for other unifiers of S as well.

Unification therefore provides a useful tool by which equivalence of expressions can be expressed in inference rules for logics which make use of variable symbols, such as first-order logics.

3.3 Soundness and Completeness

It is not enough to simply have a system of inference rules with which to perform deductions on formulae in a logic. It is also necessary that the system of inference rules is correct with respect to the logic. That is to say, if a formula can be derived through the application of inference rules, then it also must be true under all interpretations. This is known as the *soundness* property of a logical calculus. Conversely, it is also desirable that there exist derivations for those formulae which are true under all interpretations, a property which is known as *completeness*. Minimally, a logical calculus must be sound, and to be robust it is also useful for it to be complete. Formally, these properties can be defined as follows.

Definition. A logical calculus $\mathcal{C} = (\mathcal{A},\ \mathcal{R})$ is *sound* for a logic $\mathcal{S} = (\mathcal{L},\ \mathcal{W},\ \mathcal{I})$ if and only if for all sets of formulae $\Gamma \subseteq \mathcal{L}$ and for all formulae $\alpha \in \mathcal{L}$, if $\Gamma \vdash^{C} \alpha$ then $\Gamma \models^{S} \alpha$.

Definition. A logical calculus $\mathcal{C} = (\mathcal{A},\ \mathcal{R})$ is *complete* for a logic $\mathcal{S} = (\mathcal{L},\ \mathcal{W},\ \mathcal{I})$ if and only if for all sets of formulae $\Gamma \subseteq \mathcal{L}$ and for all formulae $\alpha \in \mathcal{L}$, if $\Gamma \models^{S} \alpha$ then $\Gamma \vdash^{C} \alpha$.

Example 19. A calculus using just the resolution inference rule described in Section 3.2.2 is sound for first-order predicate, yet not complete. In order to be complete, such a calculus would also need the addition of the *factorization* inference rule, defined as:

$$\frac{\alpha_1 \vee ... \vee \alpha_i \vee ... \vee \alpha_j \vee ... \vee \alpha_n}{\theta(\alpha_1 \vee ... \vee \alpha_i \vee ... \vee \alpha_{j-1} \vee \alpha_{j+1} \vee ... \vee \alpha_n)}$$

where $\theta = mgu(\alpha_i, \alpha_j)$.

Simply put, if a formula contains a disjunction of literals which can be unified, then remove one of them and apply the unifier to the formula.

3.4 Search Control

Automated theorem provers operate by performing search on formulae in a particular logic, in accordance with the inference rules and axioms specified by a calculus for that logic. Essentially, a theorem prover is searching for a sequence of inference rules which syntactically connect the input formulae to the axioms, the direction of which is determined by the specific calculus being used. Each step of the search is defined by a particular knowledge representation, and the search proceeds by transitioning between knowledge representations until the goal can be identified in the current representation. A *transition* between knowledge representations is defined as the selection of an inference rule to perform, and also the component(s) of the current knowledge representation to perform this inference rule on. This choice is not a trivial one, as at any given knowledge representation, there may be many formulae to which inference rules can be applied, and further, there may be many different inference rules to choose from. The mechanism which makes this decision is known as the *search control*, which, in conjunction with a logic and a calculus, forms the final component of an automated theorem prover.

Given that at each step of the search there are likely many possible formulae and applicable inference rules to choose from, the goal of the search control is to choose the transition which will result in a knowledge representation that is closest to one containing the goal of the search. The distance between any particular knowledge representation and one containing the goal of search can be measured as the number of transitions necessary to reach the goal representation from that particular knowledge representation. Optimally, the search control should choose transitions that result in the shortest path between the initial representation and the goal, in order to generate the shortest proof of the theorem, but also to minimize the amount of computational resources needed to perform the proof. Design of a search control for an automated theorem prover must then balance these two factors of optimization. If a particular search control performs lengthy and complex computations in order to determine the optimal transition to perform at each step of the search, it may indeed result in the shortest possible proof, yet it will likely be sub-optimal with regards to computational resource consumption.

This balance is generally struck by implementing search controls for automated theorem provers as heuristic approximation methods based on the syntactic features of formulae. Given that, for a particular calculus, the number of inference rules is fixed, whereas the number of formulae to which it is applicable to a given knowledge representation is quite variable, it is often the formulae selection component of search control which is the most difficult. In order to keep the complexity of the search control minimal, formulae are selected by measuring them according to a utility function which evaluates formulae according to syntactic features, such as length, occurances of particular symbols, depth of terms, etc.. By evaluating utility as a property of syntactic features, the computational complexity of performing this evaluation can be kept in roughly linear time with respect to the number of symbols occuring in a formula.

Example 20. The *weighted sum* [Fuc96] utility function which counts the occurances of various classes of symbols and assigns weights to each in their summation.

Let λ be a term or wff, V be the set of variable symbols and F the set of function symbols; the weighted sum $w(\lambda)$ is defined as:

$$w(\lambda) = \begin{cases} 1, & \text{if } \lambda \in V \cup C \\ 2 + \sum_{i=1}^{n} w(t_i), & \text{if } \lambda \equiv f(t_1, ..., t_n) \text{ where } f \in F \cup FV \\ 1 + \sum_{i=1}^{n} w(t_i), & \text{if } \lambda \equiv R(t_1, ..., t_n) \text{ where } R \in P \cup PV \cup PI \\ \sum_{i=1}^{n} w(\alpha_i), & \text{if } \lambda \equiv \star(\alpha_1, ..., \alpha_n) \text{ where } \star \in J \\ w(\alpha), & \text{if } \lambda \equiv \Box x_1, ..., x_n \bullet \alpha \text{ where } \Box \in Q \end{cases}$$

By selecting terms according to a minimization criteria using w, this utility function biases towards selecting terms with more variables rather than function symbols, and formulae that are connected by junctors rather than predicates, given the difference in weights assigned to each case.

The principle behind such heuristic utility functions is to imbue the search control with human knowledge of selection preferences while still maintaining a low computational complexity. For the utility function w presented in Example 20 above, the human knowledge being imparted is that smaller facts with fewer functions than variables are more general, and therefore more useful during deduction. The intent is that selecting facts with minimal utility values according to w will result in transitions that bring the knowledge representation closer to one containing the goal. However, heuristics such as these cannot be proven as always selecting the best transition to perform, and the applicability of a particular utility metric can only be established through experimental result. While it may be that, in general, selecting smaller formulae in accordance with w results in shorter proofs than selecting larger terms, there may also exists problems for which this selection criteria is quite inefficient, and in these cases selecting larger terms results in shorter proofs.

Given that different search controls may evaluate the utility of possible transitions from any particular knowledge representation differently, differences in search controls will lead to different paths being taken through the search space. Assuming the underlying calculus is sound, if a particular search control reaches a representation containing the goal, we can be assured that the theorem is in fact a semantic consequence. However, even given a sound and complete calculus, it may be that certain search controls are unable to find syntactic derivations of theorems which are semantically provable, or even provable using another search control. This can occur if the search control violates the principle of *fairness*. Simply put, fairness dictates that every valid transition from a particular search state must at least have the possibility of being chosen. If certain transitions are outright denied the possibility of being chosen, it may be that, while the utility value generated by the heuristic metric being used makes this trasition appear to be a bad choice, it is actually a necessary transition on the path to the goal state.

Finally, regardless of the search control being used, unless the theorem being proved is a member of the axioms to begin with, there exists a minimal sequence of transitions necessary to prove the theorem. This minimal path between the initial representation and the goal is the optimal solution against which search controls may be compared to determine efficiency, and regardless of the search control being used, none of them can generate a proof of the theorem in fewer transitions. Necessarily, this minimal proof is bound to the particular calculus being used, and for a different calculus there may be a proof with a smaller number of transitions, but nonetheless for any calculus there exists a minimal proof for a given theorem.

Chapter 4

Malicious Argumentation

This chapter addresses the principal contribution of this thesis. In Section 4.1 the general pitfalls of resource bounded argumentation are discussed, focusing in particular on the potential exploitation of these bounds by malicious agents in an open multi-agent system. Section 4.2 then analyzes the decision procedures involved in argument validity, acceptability evaluation and the dialogue game for possible exploitation based on resource bounds exhaustion. In order to maintain the scope of this thesis, a malicious agent strategy is described only for one of these decision procedures: the attack relation decision procedure computed during argument evaluation. Towards this end, Section 4.3 discusses methods by which a malicious agent may alter the content of an argument in order to exhaust the resources allocated to its opponent's attack relation decision procedure in order to manipulate the outcome of this decision procedure. Finally, in Section 4.4, an example of this malicious agent strategy is given for a hypothetical open multi-agent system that uses argumentation to manage contracted designs.

4.1 Consequences of Resource Bounded Argumentation

Agents often need to interact in order to achieve their goals. It may be that they require resources controlled by another agent, that they require the cooperation of another agent to complete a task, or that they require or wish to affect another agent's internal state. Argumentation provides a means for this interaction to occur, often offering advantages over simple symbolic communication and more traditional approaches to these problems. Rather than simply transmitting solutions, argumentative agents include supporting reasons behind these solutions, which allows these agents to engage in a complex reasoning process in order to find consensus despite conflicts in and between their knowledge bases.

The advantages gained through the use of argumentation, however, are not without consequence. As discussed in Section 2.5, practical implementations of argumentation must impose resource bounds at several levels of the process, due to the intractability of procedures used by the argumentation process. These resource bounds result in *non-monotonicity in computation*, such that a different result could have emerged from the process if it were to have run longer. This consequence has been recognized for over a decade by the field of automated argumentation [Lou98], although few researchers have paid much attention to the implications of computational nonmonotonicity in argumentation.

Due to the resource bounds imposed on various procedures involved in the argumentation process, it may be that an argument which has been evaluated by an agent as acceptable would be found unacceptable by that agent if greater resources were afforded to the process. If that agent were to be engaged in, for instance, a deliberation dialogue, this may have the consequence of the agent agreeing to perform actions it would otherwise have not agreed to perform, if given greater resources to perform argument evaluation with. While this is an unavoidable consequence of resource bounded argumentation, when implemented in a closed multi-agent system, fairness criteria can be used to ensure that agents have relatively equitable resource bounds [Lou98]. Such an approach does not avoid the consequences of resource bounded argumentation, but rather attempts to minimize it by "levelling the playing field", so that each agent has a roughly equal chance of making incorrect decisions due to resource bounds exhaustion.

However, fairness in resource consumption can only be achieved if the agents involved are willing to play fair. In a closed multi-agent system, achieving such fairness is a relatively trivial matter, as the agents involved are designed and controlled by a single entity, and so resource consumption can be controlled locally. In an open multi-agent system, where agents may be designed and controlled by many different entities and executed on distributed and uncontrolled resources, achieving fairness in resource consumption is likely impossible. In such systems, resource consumption can only be controlled by protocol rather than design, and protocol can only control those aspects of an agent that are externally verifiable, such as time. While imposing temporal limits on agent interactions can ensure that these interactions occur in a timely fashion, this cannot ensure equitable resource consumption, as agents may have great disparity in the computational resources available to them.

In open multi-agent argumentation systems, then, some agents may have an advantage over others through access to greater computational resources. By affording greater resources to, for instance, their argument evaluation procedures, these agents may be able to make better decisions than agents with fewer resources. While this is not a particularly desirable consequence, it does not undermine the fundamental autonomy of the agents involved in the system, and so may be considered an acceptable limitation of open multi-agent argumentation systems.

When dealing with open multi-agent systems, however, the possibility of malicious self-interested agents must be accounted for. While agents may need to cooperate, whereby cooperating agents work together in pursuit of a mutually beneficial goal that may not be achievable seperately, it is generally the case in open multi-agent systems that agents are primarily interested in achieving their individual goals. In pursuit of their goals, agents may attempt to strategically "bend the rules" to gain an advantage over other agents in the system. In contrast to closed multi-agent systems, open multi-agent systems in particular must consider the possibility of such malicious behaviour, as the agents within the system are generally implemented by different entities, and so a high-level altruistic notion of "fair play" cannot be instilled in the agents by design.

In open multi-agent argumentation systems in which agents are resource bounded, then, resource disparity may be strategically exploited by malicious agents towards their own ends. By utilizing superior resources and sufficient knowledge of its opponent, a malicious agent may simulate its opponent's decision procedures and construct an argument specifically designed to exhaust its opponent's resource bounds. In this way, an argument which would be evaluated as unacceptable by the agent's opponent, due to the opponent discovering a conflict between its knowledge base and the argument and therefore an attack against the argument, may be rendered acceptable by introducing superfluous complexity into the argument such that the opponent cannot deduce the conflict within its resource limitations. By manipulating the outcome of argument evaluation, a malicious agent may in turn affect the outcome of the dialogue game itself. If the agents are involved in deliberation regarding a course of action, this could result in the malicious agent controlling its opponent's actions; if involved in negotiation of resource distribution, the malicious agent could affect an unequal distribution of resources; if involved in persuasion, the malicious agent could cause its opponent to adopt particular knowledge it would otherwise not accept. Rather than resource exhaustion occuring happenstantially, as can occur in a closed multi-agent argumentation system, malicious agents in an open multi-agent argumentation system may exploit resource disparity to force resource exhaustion for their own ends. This clearly undermines the security of these systems, as such techniques would allow malicious agents to gain an undue amount of influence over other agents in the system, compromising the overall system goal in favour of the malicious agent's individual goals.

4.2 Decision Procedures Susceptible to Malicious Argumentation

Malicious argumentation is made possible by agents needing to perform intractable, resource bounded decision procedures on complex data constructed (at least partially) by other agents. Given knowledge of the procedure and resource bounds, the other agent can structure the data in such a way that resource exhaustion is gauranteed. For a given resource bounded decision procedure returning a boolean value, the case of resource exhaustion represents a third possibility; neither the conditions for a positive or negative result have been reached, yet nonetheless the resources allocated to the procedure have been exhausted, and a result must be returned. It is therefore necessary for the procedure to map the case of resource exhaustion onto either a positive or negative result. A malicious agent may then exploit such a procedure by employing a resource exhaustion strategy if it desires the result that occurs in the case of resource exhaustion rather than the result that would occur if the procedure is able to terminate "naturally".

The use of complex formal logics for representing knowledge in argumentation systems, and in particular, for representing *exchanged* knowledge, opens the door for malicious argumentation strategies based on resource bounds exhaustion. Decisions requiring intractible procedures such as deduction to be performed on complex knowledge received from an external source must be resource bounded, and are therefore susceptible to malicious exploitation based on resource bounds exhaustion. However, such malicious argumentation techniques are not limited to just those decision procedures that operate on complex logical formulae; any resource bounded intractable decision procedure involving external data may be susceptible to malicious argumentation. We herein examine procedures that may potentially be exploited by malicious agents, although this examination is by no means exhaustive. Further, we shall focus on the use of a resolution based theorem prover in the decision procedures which require deduction to be performed. However, these techniques are not limited to just those argumentation systems using resolution based provers; any theorem prover is susceptible to resource exhaustion techniques, although the specific instantiation of these techniques will likely need to be tailored to exploit the particular inference rules and search control used by that prover.

4.2.1 Argument Validity

Upon receiving an argument from another agent, before performing argument evaluation, the argument should be tested for validity. In a closed multi-agent argumentation system, this may not be necessary, as agents are designed by a single entity and therefore can be designed to only transmit valid arguments. In open multi-agent argumentation systems, however, the possibility that agents may employ invalid arguments requisites an initial test of argument validity. The conditions of argument validity, as described in Section 2.3, are defined for an argument $\langle \Phi, \alpha \rangle$ as follows:

1. The support set Φ is consistent: $\Phi \nvdash \bot$
2. The conclusion α is a logical consequence of Φ: $\Phi \vdash \alpha$
3. Φ is a minimal set satisfying (1) and (2): there is no $\Phi' \subset \Phi$ such that $\Phi' \nvdash \bot$ and $\Phi' \vdash \alpha$

It should be immediatly apparent that the argument validity decision procedure is intractable; each of the three conditions comprising the procedure involve deduction, which is a known intractable procedure for all logics of interest. Given that determining argument validity is a component procedure of an agent's locution decision procedure, as it must be performed on an incoming argument in order for the agent to decide which locution to respond with in the dialogue, the argument validity decision procedure must also be resource bounded. It is therefore possible that a malicious agent may target this procedure for strategic resource exhaustion, with the aim of passing off an invalid argument as valid.

As each of these conditions involves deciding deduction between sets of formulae, it is necessary to employ an automated theorem prover in their decision procedures. As it is necessary to impose resource limits on these decision procedures, the outcome of performing deduction will be one of three different possibilities: *true*, *false*, or *resource exhaustion*. However, the decision procedures themselves need to evaluate to a purely boolean value; it is therefore necessary for the procedure to handle the case of resource exhaustion in its deduction component by mapping it onto a boolean value.

4.2.1.1 Consistency of Support

Consider a decision procedure for the first condition, using a resolution-based theorem prover as described in Section 3.2.2 in a straightforward manner to test a set of formulae for consistency. That is to say, rather than operating the prover in a refutation-based manner to decide a consequence relation, the prover can be used to test for consistency by simply giving it a set of formulae from which it attempts to derive a contradiction. The simplest result is the case in which the set of formulae is inconsistent, and the prover is able to derive a contradiction within the resource limits, in which case the decision procedure can return the answer *false*. If resources are exhausted before a contradiction can be derived, it may be tempting to design the decision procedure to also answer *false*, in order to account for cases where the set of formulae is inconsistent yet the contradiction is outside the resource bounds. However, in the case where the set of formulae is consistent, it is necessary for the prover to exhaust all possible applications of inference rules. If all possible inference rule applications are exhausted before the prover's resource bounds, the decision procedure can obviously answer *true*, yet in practical applications, the resource limits of the prover will likely be exhausted well before all possible inference rule applications are exhausted. In order to avoid false positive identifications of inconsistent sets of formulae, for practical purposes the case of resource exhaustion should also be mapped to *true*. However, while this enables the argumentation system to handle larger valid arguments, it also opens the possibility of malicious agents presenting large invalid arguments that, due to resource exhaustion, are evaluated as being valid.

4.2.1.2 Logical Entailment of Conclusion

The second condition of argument validity, that the argument's conclusion must be a logical consequence of the support, is slightly different than the first, with respect to the case of resource exhaustion. Again, consider a decision procedure for this condition which decides the consequence relation between the support and conclusion using a resolution based theorem prover. As described in Section 3.2.2, the conclusion is negated, added to the support, and the resolution inference rule is applied until either a contradiction is derived or no further applications of the inference rule are possible. In the case where a contradiction is derived, the conclusion is a consequence of the support, and otherwise it is not. Unlike the first condition, in which applicable inference rule exhaustion is needed to establish validity, in the second condition inference rule exhaustion is needed to determine that the argument is invalid. In general, it is likely that an invalid argument will exhaust the resource limitations imposed on the decision procedure before all applicable inference rules are exhausted. However, mapping the case of resource exhaustion onto the result of *false* for the decision procedure will cause false negative results for large valid arguments, as it may be the case that a contradiction is present in the knowledge representation yet resources are exhausted before it can be derived. In order for the decision procedure to correctly handle valid arguments, the case of resource exhaustion should be mapped to *true*, yet this also allows the possibility of malicious agents constructing large invalid arguments which cause resource exhaustion before inference rule exhaustion, and are thereby deemed valid by the decision procedure.

4.2.1.3 Minimality of Support

The third condition of argument validity states that the support set should be minimal with respect to set inclusion while still fulfilling the previous two conditions. For practical purposes, this decision procedure is quite computationally expensive. In order to test this condition, it is necessary to iterate over elements of the powerset of the support set, and further, to test both of the above conditions on each of these elements. In the worst-case scenario, where there does not exist a subset of the support set for which conditions 1 and 2 hold, it is necessary to iterate over all elements of the powerset of the support set, and so for a support set of size n the upper complexity bounds of this decision procedure is 2^n times greater than the complexity of verifying the previous two conditions on the support set alone. Further, this worst-case scenario occurs in the case that the argument is valid with respect to this condition. The best-case complexity scenario occurs when this condition is violated, such that the first subset that is tested meets conditions 1 and 2, and so the complexity of this decision procedure is minimally bounded by the comlexity of the decision procedures for conditions 1 and 2. With knowledge of the order in which these subsets are tested, it may be possible for a malicious agent to strategically manipulate the outcome of this resource bounded decision procedure in order to get another agent to evaluate an argument with a non-minimal support set as valid.

Each of the component decision procedures of the argument validity decision procedure, when implemented in a practical argumentation system wherein resource limitations must be imposed on intractable procedures, may be susceptible to malicious argumentation based on exhausting resources in order to manipulate the outcome of decision procedures. While these decision procedures can be designed to map cases of resource exhaustion as identifying invalid arguments, this approach has the consequence of identifying large valid arguments as invalid. In trivial scenarios, it may be possible to keep all valid arguments within a reasonable size so as to avoid such cases. However, in practical applications employing large knowledge bases and involving complex reasoning, it is not unreasonable to assume that large arguments for which the resources allocated to the argument validity decision procedure are insufficient might naturally occur in such systems. Therefore, restricting an argumentation system to only allow those arguments which can be validated within the allocated resources cannot be seen as an acceptable solution to the problems presented by malicious argumentation.

On an aside, the minimality condition itself is rather questionable. Consider the sets of propositional logic formulae $\Phi = \{ A \wedge B \}$ and $\Psi = \{ A, B \}$. Both $\Phi \vdash A$ and $\Psi \vdash A$, however Φ satisfies the minimality condition whereas Ψ does not, as there exists a subset $\{ A \} \subset \Psi$ such that $\{ A \} \vdash A$. Given that Φ and Ψ are logically equivalent, it is clear then that the outcome of this condition can be manipulated by altering the syntactic form of an argument's support set while retaining its semantic content. By framing minimality of the support set as property based on set operations, this condition is unable to capture the notion of truly minimal support; that is, the least amount of information necessary to support the argument's conclusion.

4.2.2 Argument Evaluation

As discussed in Section 2.2.1, evaluating the justification status of an argument involves computing properties of the argument in relation to the argumentation framework, the specific properties being defined by the particular argumentation semantics being used. In order to perform this evaluation, the attack relations between arguments which comprise the argumentation framework must then be computed. In order to correctly evaluate the justification status of an argument, it is therefore necessary for an agent to generate all possible arguments from its knowledge base, and further to determine if there exist attack relations between these arguments.

While generating the set of all possible arguments from an agents knowledge base, as well as computing attack relations between these arguments, is quite an expensive and generally intractable procedure, it is possible for an agent to perform this computation prior to entering an argumentative dialogue. Even if an agent were to perform this computation dynamically during the dialogue, in which case the procedure would need to be resource bounded, nonetheless the process of generating arguments from an agent's knowledge base and computing attack relations between them is not susceptible to manipulation by a malicious agent, given that these computations are based entirely on the agent's internal data. However, evaluating the justification status of an argument presented by another agent is susceptible to malicious manipulation, as the argument being evaluated originates from a source external to the agent.

In order for an agent to evaluate an argument originating externally, the agent must determine the attack relations between that argument and the arguments in the set of all arguments $\mathcal{AR}$ constructable from the agent's knowledge base, regardless of the particular argumentation semantics being used. Determining these attack relations requires the agent to compute conflict between component formulae of the argument being evaluated and the arguments in $\mathcal{AR}$. As discussed in Section 2.3.1, deciding conflict generally involves performing deduction in the underlying logical language, which is generally an intractable and therefore resource bounded procedure when implemented in a practical argumentation system. Consider then a situation wherein an agent is presented with an argument α, which the agent must evaluate in order to determine whether to accept the argument or not. Assume the agent is using an arbitrary argumentation semantics to perform this evaluation with, for which there exists an argument $\beta \in \mathcal{AR}$ which attacks the argument α such that the outcome of the evaluation hinges on deciding the attack relation from β onto α; that is to say, the argument α is unacceptable under the particular semantics just in case the agent is able to determine that β attacks α. The agent must then determine that there is a conflict between component formulae of α and β within the resource bounds imposed on the attack relation decision procedure in order to determine that the argument α is unacceptable. Given that we are assuming β attacks α, such a conflict does exist, yet nonetheless it is necessary for the agent to employ an automated reasoning component to derive the conflict between the component formulae of α and β. In the case where the resources allocated to the attack relation decision procedure are exhausted before this conflict can be derived, the agent must map this outcome onto a result of either true or false for the decision procedure.

As with the argument validity decision procedure discussed above in Section 4.2.1, mapping the case of resource exhaustion to a result of either true or false for the attack relation decision procedure will have different consequences for the system. Consider an argumentation system which uses a resolution based theorem prover as described in Section 3.2.2 to determine conflict, where conflict is defined as a contradiction between the component formulae of the arguments involved. The first option to consider is the case of mapping resource exhaustion onto a result of *true* for the attack relation decision procedure; that is, if resources are exhausted when testing for an attack relation between two arguments, the procedure will report that an attack relation does exist between the arguments. As with the first validity condition discussed in Section 4.2.1, the resolution based prover is used to directly determine whether a contradiction is present, rather than being used in the standard refutation-based method of deciding a consequence relation. In the case when the arguments being tested do not attack one another, the prover must then exhaust all possible inference rule applications in order to determine that a contradiction is not present. For large enough arguments, it is likely that the resources allocated to the decision procedure will be exhausted before all possible inference rule applications are exhausted; mapping the case of resource exhaustion onto the result of true for the attack relation decision procedure will then likely result in many false-positive identifications of attack relations between arguments. Not only will this have the effect of the agent evaluating many acceptable arguments as unacceptable, it has the further consequence of the agent responding with counter-attacks that do not in fact attack the argument they purport to.

The attack relation decision procedure should therefore be implemented to map the case of resource exhaustion onto a result of false; that is, upon the resource limit being reached, the procedure should terminate with the result that there does not exist an attack relation between the arguments being tested. As with the decision procedures examined in Section 4.2.1, however, this opens the decision procedure up to the possibility of exploitation, whereby a malicious agent may construct an argument designed to exhaust the resource bounds of the evaluating agent's attack relation decision procedure before an attack relation onto the argument can be found. In this way, a malicious agent may manipulate the outcome of its opponent's argument evaluation procedure in order to render an unacceptable argument as acceptable, given the resource limitations imposed on the opponent's argument evaluation procedure.

4.2.3 The Dialogue Game

The final decision procedures susceptible to malicious argumentation that shall be examined herein relate to the decisions made at the level of the dialogue game. Of the five classes of dialogue game rules described in Section 2.4.1, two are particularly vulnerable to malicious argumentation: the class of commitment rules, and the termination rules. While it may be found that the classes of commencement, locution and combination rules are not only vulnerable but also advantageous to exploit through malicious argumentation, we shall herein consider only the exploitation of the commitment and termination rules.

4.2.3.1 Commitment Rules

The commitment rules of a dialogue game describe the conditions under which an agent may commit to a particular proposition. The propositions an agent has committed to are added to a public commitment store for that agent, which represents a subset of the agent's knowledge base that the agent has made publicly available through the arguments it has presented during the dialogue. While the agent's knowledge base may be inconsistent, it is necessary that the agent's commitment store is consistent, in order to maintain consistency throughout the agent's line of argumentation during the dialogue. Commitment rules then describe the means of updating the commitment store and maintaining consistency as agents present arguments, as well as allowing for deletions from the commitment store when an agent chooses to backtrack and retract previously asserted arguments. While the commitment store is "public", in a distributed multi-agent argumentation system, it is often necessary for each agent to maintain the set of commitments asserted by its opponent(s) in the argumentative dialogue. This requires agents to verify the consistency of new knowledge added to a commitment store when an opponent presents an argument, which in turn requires the agent to employ an automated theorem prover to test whether the contents of the commitment store entail a contradiction or not.

As with the argument validity and acceptability evaluation decision procedures, we shall consider an agent making use of a resource bounded resolution based theorem prover to determine the consistency of a commitment store. Again, it is necessary to examine the consequences of mapping the case of resource exhaustion onto either a positive or negative result for the commitment store consistency decision procedure. Given that verifying the consistency of a commitment store involves attempting to derive a contradiction from a set of formulae, the resolution based prover may be used in a straightforward manner to test for a contradiction, rather than in a refutation-based manner to establish a consequence relation. In the case that the commitment store being tested is in fact consistent, it is necessary for the prover to exhaust all possible inference rule applications; given a sizeable collection of commitments, it is likely that the resources allocated to this decision procedure will be exhausted before all possible inference rule applications. If the case of resource exhaustion is mapped onto a negative result for the consistency test, it is likely then that many consistent commitment stores will be reported as inconsistent due to resource exhaustion. To avoid incorrectly identifying inconsistency in a consistent commitment store, the case of resource exhaustion should therefore be mapped onto a positive result for the commitment store consistency decision procedure. As with the argument validity and acceptability evaluation decision procedures, however, this allows for the possibility of malicious manipulation of this decision procedure's outcome; a malicious agent may construct an argument designed to exhaust its opponent's commitment store consistency decision procedure before the opponent can determine that the argument contradicts previous arguments put forward by the malicious agent. A malicious agent may then successfully employ inconsistent lines of argumentation during the dialogue, through which it may manipulate the outcome of the dialogue game as a whole.

4.2.3.2 Termination Rules

The final decision procedure we shall consider as potentially susceptible to manipulation by a malicious agent relates to the termination rules of a dialogue game. Rather than considering how a malicious agent may manipulate the outcome of the termination condition decision procedures directly through strategic manipulation of the content of arguments, as the decision procedures described above have been analyzed, we shall instead consider how an agent may affect the outcome of the dialogue game as a whole by stratigically manipulating the "flow" of arguments during the dialogue. Argumentative dialogue games between agents are instantiated in order to affect a joint decision between agents; as outlined in Table 2.1, an argumentative dialogue may be used to persuade an agent of a particular proposition, to negotiate the division of resources, to decide on a course of action, and so forth. The termination rules of a dialogue game define not only the point at which the dialogue ends, but also the outcome of the dialogue; the "winner" and "loser" of the interaction, if such terms are applicable, or the course of action to be taken, the effects on an agents knowledge, the division of resources, or whatever the purpose of the dialogue may be. Given that such a dialogue game is instantiated to perform a decision, the dialogue itself must be resource bounded, so that actions based on the outcome of the dialogue game may occur in a timely fashion. These resource bounds are generally implemented as a termination rule for the dialogue game, and as with the decision procedures performed by agents during their turns, the case of resource exhaustion must be mapped onto a particular result for the dialogue game.

Unlike the decision procedures analyed so far, the outcome of the dialogue game in the case of resource exhaustion is not as clear, but rather must be based on the particulars of the system and dialogue game type being used. However, given that agents involved in the dialogue game must be aware of the termination rules of the game, and how the case of resource exhaustion is handled, strategic manipulation of the dialogue game is nonetheless possible. Whereas the decision procedures used in argument validity and acceptability evaluation could be exploited through resource exhaustion by adding formulae to the content of arguments, the decision procedure based on the dialogue may be exploited by adding arguments to the content of the dialogue. For instance, if a malicious agent finds itself in a winning position which could be overturned by its opponent introducing a particular argument, the malicious agent may employ delay tactics to draw its opponent's attention away from that particular line of argumentation until the resources allocated to the dialogue are exhausted. If a malicious agent finds itself unable to win a dialogue game, it may construct superfluous arguments designed to delay its opponent from finding a winning argument, and thus draw a stalemate from the interaction rather than a loss. While these techniques are likely more difficult to implement than the resource exhaustion strategies described above which target the automated theorem proving component of an argumentative agent, it is nonetheless possible for a malicious agent to exploit the rules of a dialogue game in order to manipulate its outcome by introducing superfluous arguments designed to exhaust the resources allocated by the termination conditions of the dialogue game.

4.3 Strategically Manipulating Arguments

An agent employs a malicious argumentation strategy when it desires to manipulate the outcome of one of its opponent's decision procedures for a given argument or line of argumentation. A malicious resource exhaustion strategy can be performed by an agent when the decision procedure being targeted returns the desired result in the case of resource exhaustion. To accomplish this, the agent modifies the content of the argument or line of argumentation by adding superfluous information designed to exhaust the decision procedure's resource bounds, while still retaining the original content of the argument. This is a notably distinct case from lying, as when an agent lies, it misrepresents information, such as reporting that it believes propositions it does not, or plans to perform actions it cannot or has no intention to. In a malicious resource exhaustion strategy, the agent does not violate the felicity conditions of the locution being performed; rather, the agent manipulates how its opponent interprets the information by modifying the content so that a different result is achieved than what would have occured if the opponent had sufficient time to completely analyze the information.

We shall herein focus on techniques for exploiting a particular decision procedure: the attack relation decision procedure used by the argument evaluation decision procedure.

By manipulating the outcome of the attack relation decision procedure, a malicious agent may render a particular argument acceptable to its opponent which would be found unacceptable if the opponent had sufficient resources to thoroughly examine the argument. Given that exploiting the attack relation decision procedure involves exhausting the resource bounds of an automated theorem prover as it searches for a contradiction in a set of formulae, the techniques described here could also be used to exploit other decision procedures which make use of an automated theorem prover for similar purposes, such as the argument validity decision procedures discussed in Section 4.2.1 or the commitment store rules of the dialogue game discussed in Section 4.2.3. Nonetheless, in order to maintain the scope of this thesis, the discussion of strategically manipulating arguments for resource bounds exhaustion based exploitation shall be limited to just the attack relation decision procedure. Towards this end, two techniques will be described in the following subsections: implication chaining, by which arbitrarily large consequence relations can be constructed, and tautology injection, which makes use of arbitrarily large tautologies to exhaust a theorem provers resources. These techniques will then be put to use in an example of a malicious argumentation scenario in Section 4.4.

4.3.1 A Resource Bounded Undercut Relation

For the purpose of the argument manipulation strategies described in Section 4.3.2 and Section 4.3.3, as well as the malicious argumentation example scenario discussed in Section 4.4, we shall focus on resource bounded exploitation of a particular attack relation. The attack relation that will be used in these sections is the *undercut* attack relation, based on those described in Section 2.3.1. Further, the attack relation needs to be described as a resource limited decision procedure. Rather than strictly limiting time or computational cycles, for simplicity's sake we shall instead limit the number of inference steps performed by the automated reasoning component used to decide syntactic consequence in the argumentation system's attack relation decision procedure, similar to the method used in [Lou98]. Towards this end, the symbol $\vdash_k$ will be used to denote deduction limited to k inference steps; that is, if the theorem prover being used can determine that $\phi \vdash \psi$ in k or less inference steps using a given search control, then $\phi \vdash_k \psi$, otherwise $\phi \nvdash_k \psi$.

The resource limited undercut attack relation used herein can then be described formally as:

Definition. An argument $\langle \Phi, \alpha \rangle$ is an *undercut*$_k$ of an argument $\langle \Psi, \beta \rangle$ iff $\Psi \cup \{ \alpha \} \vdash_k \bot$.

4.3.2 Implication Chaining

The goal of the argument manipulation techniques described herein is to modify the contents of a given argument such that the interpretation of the symbols present in the original argument remains the same yet the resources required to decide a certain property of the argument may be arbitrarily increased. It is necessary that these techniques are scalable, as a particular malicious argument needs to be tailored to the specific resource bounds of the opponent's decision procedure. The first of such techniques we shall examine makes use of chains of material consequence relations to arbitrarily increase the resources needed to detect a contradiction when using a resolution based automated theorem prover.

The material consequence relation $P \rightarrow Q$, where P and Q are well-formed formulae in a logical language, is interpreted for a boolean logic as "if P then Q"; that is, if P is true, then Q must also be true, yet if P is false, Q may be either true or false. The "P" side of the relation is referred to as the *antecedent*, and the "Q" side as the *consequent* of the material consequence relation. Any given well-formed formula of the language can be modified without changing the interpretation of the symbols involved in the original formula by constructing a new formula in which the original formula is the consequent of a material implication with a true antecedent.

Example 21. Consider the initial formula Q in boolean propositional logic; that is, an assertion that Q is true.

A new formula can then be constructed by re-writing Q to be the consequent of a material consequence relation with a true antecedent; e.g: $P \wedge (P \rightarrow Q)$.

The interpretation of this new formula is different from the original, specifically because the new symbol P has been added, which must also be interpreted. However, the symbol Q nonetheless has the same interpretation in the new formula $P \wedge (P \rightarrow Q)$ as it did in the original formula Q.

The purpose of such a manipulation, as stated above, is to increase the resources needed to determine particular properties of the formulae when using an automated theorem prover. The property we are interested in here is a contradiction, by way of using a resolution based automated theorem prover. The technique of re-writing a formula to be the consequent of a true antecedent can be used to increase resource consumption by a resolution based prover, as can be seen in the following example.

Example 22. Consider a set of formulae $\{ Q, \ \neg Q \}$, which is quite obviously contradictory. Through the application of a single resolution inference rule, this contradiction can be found:

$$\frac{Q, \quad \neg Q}{\square}$$

Now consider a modification of the formula Q in this set, as described above, to be $P \wedge (P \rightarrow Q)$. Given that we're dealing with a resolution based prover, this new formula needs to be converted into clauses as follows: $\{ P, \ \neg P \vee Q \}$.

The new set of formulae to be tested for contradiction is then: $\{ P, \ \neg P \vee Q, \ \neg Q \}$

In order to establish a contradiction now, two inference rules need to be applied to the set of formulae:

1. Resolution: $\dfrac{P,\ \ \neg P \vee Q}{Q}$, Result: $\{\ P,\ \neg P \vee Q,\ \neg Q,\ Q\ \}$
2. Resolution: $\dfrac{Q,\ \ \neg Q}{\Box}$, Result: $\{\ P,\ \neg P \vee Q,\ \neg Q,\ Q,\ \Box\ \}$

This technique can be arbitrarily scaled to require a specific number of resolution inference rules to be applied before the contradiction can be found. For instance, to force the theorem prover to use n resolution inference rules to establish a contradiction, the set of formulae can be re-written as:

$$\{\ P_1,\ P_1 \rightarrow P_2,\ ...,\ P_{n-1} \rightarrow Q,\ \neg Q\ \}$$

In order to use this technique to exploit the resource bounds of the attack relation decision procedure, a malicious agent needs to know the resource bound imposed on this procedure by its opponent, as well as the component of its argument which contradicts an element of the opponent's knowledge, which will form the basis for the attack relation.

Example 23. Consider an argument $\langle \Phi, \alpha \rangle = \langle \{\ P,\ P \rightarrow Q\ \}, Q \rangle$ of agent a_1, which is undercut only by the argument $\langle \Psi, \beta \rangle = \langle \{\ \neg P\ \}, \neg P \rangle$ of agent a_2.

Agent a_1 will play the role of the malicious agent in this scenario, and a_2 will be the opponent. If presented with the argument $\langle \Phi, \alpha \rangle$, agent a_2 would respond with the attacking argument $\langle \Psi, \beta \rangle$. We shall assume that a_2's attack relation decision procedure (in this case, the undercut relation decision procedure described in Section 4.3.1 above) is limited to k inference steps, where k is sufficient to decide that $\langle \Psi, \beta \rangle$ $undercut_k$ $\langle \Phi, \alpha \rangle$.

If the malicious agent a_1 desires that a_2 does not attack the argument $\langle \Phi, \alpha \rangle$, it can employ an implication chaining technique to exhaust a_2's resources before a_2 can discover the contradiction in $\Phi \cup \beta$. Agent a_1 can then re-write $\langle \Phi, \alpha \rangle$ as follows:

$$\langle \Phi', \alpha \rangle = \langle \{\ P_1,\ P_1 \rightarrow P_2,\ ...,\ P_k \rightarrow P,\ P \rightarrow Q\ \}, Q \rangle$$

It will then take a_2 a total of $k+1$ inference steps to determine the contradiction in $\Phi' \cup \beta$. Given that a_2 has only k inference steps allocated to the undercut relation decision procedure, it will have exhausted its resources before it can determine that $\langle \Psi, \beta \rangle$ $undercut_k$ $\langle \Phi', \alpha \rangle$. Assuming that a_2 is mapping the case of resource exhaustion as discussed in Section 4.2.2, agent a_2 will then accept the argument $\langle \Phi', \alpha \rangle$, as it cannot find an attack against the argument.

Note that it isn't necessary for a malicious agent to know the exact value of k used by its opponent. It would be enough for the agent to approximate a value for k that it can reasonably assume is far past the actual value of k. If the agent's opponent is using commercially available software, this may be as simple as using default or maximal values for that software. Further, it may be possible for a malicious agent to perform this approximation through a learning process. A number of trivial arguments could be used to test different values for k until a reasonable approximation can be formed, which can then be used in a malicious argumentation strategy to manipulate the opponent's decision procedure(s) for an important argument.

4.3.3 Tautology Injection

The next argument manipulation strategy we shall examine makes use of tautologies to exhaust the resources of an opponent's attack relation decision procedure. A tautology is a formula in a boolean logic which is interpreted as true under all interpretations. Simple tautologies are formulae such as $P \vee \neg P$, which states that P is either true or false; under a boolean logic, it is impossible for a wff to be neither true nor false, and so this formula evaluates to true under all possible assignments of truth-values to P. This example tautology is quite simple, however it can be easily modified to introduce superfluous complexity; consider that the propositional variable P in the formula can be replaced by any well-formed formula of the logic and the resulting formula will still be tautological. Further, there are many tautologies of much greater complexity than $P \vee \neg P$, such as the formula $((P \to Q) \wedge (R \to S)) \to ((P \vee R) \to (Q \vee S))$. By composing such tautologies with other formulae, as well as conjoining them with other tautologies, one can create increasingly complex formulae which nonetheless will always be evaluated as true under all interpretations.

Similar to the method of implication chaining described above in Section 4.3.2, tautology injection can be used to increase the resources needed to establish a particular contradiction by making the formula which contradicts the opponents knowledge the consequent of a material consequence relation with a tautology as the antecedent. The opponent's automated theorem proving component must then establish the truth of the antecedent (the tautology) before it can establish the truth of the consequent.

Example 24. The formula Q in propositional logic can be re-written as $\mathcal{T} \rightarrow Q$, where $\mathcal{T}$ is any tautology. For instance:

- $(P \vee \neg P) \rightarrow Q$
- $(P \leftrightarrow (P \wedge P)) \rightarrow Q$
- $((R \leftrightarrow S) \leftrightarrow (\neg R \leftrightarrow \neg S)) \rightarrow Q$

Each of these formulae is logically equivalent to Q, however when combined with the formula $\neg Q$, a theorem prover will require greater resources to establish the contradiction than it would to determine $\{Q,\ \neg Q\} \vdash \bot$.

Example 25. Consider the formula $(P \vee \neg P) \rightarrow Q$, which can be written in clausal form as $\{\ \neg P \vee Q,\ P \vee Q\ \}$.

Using a resolution based theorem prover, it will take three inference steps to determine that $\{\ \neg P \vee Q,\ P \vee Q,\ \neg Q\ \} \vdash \bot$:

1. Resolution: $\dfrac{\neg P \vee Q,\quad P \vee Q}{Q \vee Q}$, Result: $\{\ \neg P \vee Q,\ P \vee Q,\ \neg Q,\ Q \vee Q\ \}$
2. Factorization: $\dfrac{Q \vee Q}{Q}$, Result: $\{\ \neg P \vee Q,\ P \vee Q,\ \neg Q,\ Q \vee Q,\ Q\ \}$

3. Resolution: $\dfrac{\neg Q,\ \ Q}{\square}$, Result: $\{\ \neg P,\ P \vee Q,\ \neg Q,\ Q \vee Q,\ Q,\ \square\ \}$

Given that $\{\ Q,\ \neg Q\ \} \vdash \bot$ can be established in a single inference step, the injection of a simple tautology is able to increase the resource consumption of the theorem prover. Tautologies of arbitrary complexity can be constructed to exhaust specific resource bounds using a number of simple methods, such as conjoining tautologies in the antecedent, or replacing symbols in the tautologies with more complex formulae.

Example 26. The tautology used in Example 25 can be scaled by conjoining multiple instances of the tautology, albeit with different symbols. For instance:

$$((P_1 \vee \neg P_1) \wedge \ldots \wedge (P_n \vee \neg P_n)) \rightarrow Q$$

The general clausal form for the above formula is rather unwieldy, and so we shall examine the case of determining a contradiction between this formula for $n = 2$ and the formula Q, in order to compare to the case of $n = 1$ given in Example 25. For $((P_1 \vee \neg P_1) \wedge (P_2 \vee \neg P_2)) \rightarrow Q$ then, the clausal form is:

$$\Delta = \{\ Q \vee \neg P_1 \vee \neg P_2,\ Q \vee \neg P_1 \vee P_2,\ Q \vee P_1 \vee \neg P_2,\ Q \vee P_1 \vee P_2\ \}$$

Let $\Delta_0 = \Delta \cup \{\ \neg Q\ \}$. In order to establish that $\Delta_0 \vdash \bot$ using a resolution based theorem prover, the following inference steps are performed:

1. Resolution: $\dfrac{Q \vee P_1 \vee P_2,\ \ Q \vee \neg P_1 \vee P_2}{Q \vee Q \vee P_2 \vee P_2}$, Result: $\Delta_1 = \Delta_0 \cup \{\ Q \vee Q \vee P_2 \vee P_2\ \}$
2. Factorization: $\dfrac{Q \vee Q \vee P_2 \vee P_2}{Q \vee P_2 \vee P_2}$, Result: $\Delta_2 = \Delta_1 \cup \{\ Q \vee P_2 \vee P_2\ \}$
3. Factorization: $\dfrac{Q \vee P_2 \vee P_2}{Q \vee P_2}$, Result: $\Delta_3 = \Delta_2 \cup \{\ Q \vee P_2\ \}$
4. Resolution: $\dfrac{Q \vee P_2,\ \ Q \vee P_1 \vee \neg P_2}{Q \vee Q \vee P_1}$, Result: $\Delta_4 = \Delta_3 \cup \{\ Q \vee Q \vee P_1\ \}$
5. Factorization: $\dfrac{Q \vee Q \vee P_1}{Q \vee P_1}$, Result: $\Delta_5 = \Delta_4 \cup \{\ Q \vee P_1\ \}$

6. Resolution: $\dfrac{Q \vee P_1, \quad Q \vee \neg P_1 \vee \neg P_2}{Q \vee Q \vee \neg P_2}$, Result: $\Delta_6 = \Delta_5 \cup \{\ Q \vee Q \vee \neg P_2\ \}$
7. Factorization: $\dfrac{Q \vee Q \vee \neg P_2}{Q \vee \neg P_2}$, Result: $\Delta_7 = \Delta_6 \cup \{\ Q \vee \neg P_2\ \}$
8. Resolution: $\dfrac{Q \vee \neg P_2, \quad Q \vee P_2}{Q \vee Q}$, Result: $\Delta_8 = \Delta_7 \cup \{\ Q \vee Q\ \}$
9. Factorization: $\dfrac{Q \vee Q}{Q}$, Result: $\Delta_9 = \Delta_8 \cup \{\ Q\ \}$
10. Resolution: $\dfrac{Q, \quad \neg Q}{\Box}$, Result: $\Delta_{10} = \Delta_9 \cup \{\ \Box\ \}$

The choice of n for a particular resource bound k is not as clear as with the method of implication chaining described in Section 4.3.2, given the growth pattern of inference rules needed to determine a contradiction using n conjoined tautologies of the form $P_i \vee \neg P_i$ in the antecedent of the constructed consequence relation. Nonetheless, for any given resource bound k, it is possible to find a value for n such that the opponent's theorem prover cannot determine the existence of a contradiction within k inference steps.

4.4 Example of Malicious Argumentation

We shall now examine the use of malicious argumentation in a more complete example scenario. After a description of the system, a "normal" interaction scenario is considered first, wherein an agent presents an unacceptable argument to its opponent, and the opponent is able to discover and respond with an attacking argument. Modifications to the initial argument's support are then considered, using the methods of implication chaining and tautology injection described in Section 4.3 to exhaust the opponent's resource bounds before the attacking argument can be discovered. In this way, the initially unacceptable argument is rendered acceptable to the opponent, while still retaining the initial semantic content of the argument.

4.4.1 Example System

In this example, a company uses a task postings board to list requests for certain components to be designed and constructed by outside contractors. The items listed on this board contain the functional requirements of the components to be designed, written as formal specifications in first-order logic, and possibly other requirements, also in first-order logic. A manager agent a_m is then responsible for testing the acceptability of design arguments proposed by a contractor agent a_c, responding with an attacking argument if the design is found to be unacceptable.

4.4.1.1 Example Semantics

The specifications, as well as the design proposed by contractor agents, make use of a shared catalogue language Σ_{cat} based on a common ontology of symbols and concepts Γ_{cat}, which describes myriad parts, properties and functionalities available to the company and their contractors. Along with the design specification $\Theta_{spec} \subset \Sigma_{cat}$, the company also publishes a set of conditions $\Theta_{int} \subset \Sigma_{cat}$, which describes internal corporate policies or other knowledge apart from the design specifications that may be used to attack design proposal arguments. Designs proposed by contractors then must have a conclusion that satisfies Θ_{spec}, and be acceptable with relation to $\Theta_{spec} \cup \Theta_{int}$.

The following predicates are used by a_m to describe design specifications and other internal conditions, and also by a_c in the design proposal.

$UseC(X)$	: design uses component X
$HasP(X,Y)$	: component X has property Y
$ProvF(X,Y)$	: component X provides function Y
$Conn(X,Y)$	: component X connected to component Y

4.4.1.2 Testing Acceptability

Rather than describing a full dialogue between the agents, we shall focus here on the initial assert locution in which a_c proposes a design argument $\langle \Phi, \alpha \rangle$, and a_m tests the acceptability of this argument, responding with an attacking argument if one is found. Due to resource constraints, a_m will impose an inference count limit k on the process of determining argument acceptance, as described in Section 2.5; we do not, however, consider any resource constraints placed on a_c, as we are concerned here only with the process of resource bound acceptance used by a_m which may be exploited by malicious agents.

In order to determine the acceptability of $\langle \Phi, \alpha \rangle$ with respect to its internal knowledge Θ_{int} and the design specifications Θ_{spec}, agent a_m will attempt to construct an argument $\langle \Psi, \beta \rangle$ from the formulae in $\Theta_{spec} \cup \Theta_{int}$ such that $\langle \Psi, \beta \rangle$ *attacks* $\langle \Phi, \alpha \rangle$. The resource bounded undercut relation described in Section 4.3.1, limited to k inference steps, will be used for the *attacks* relation in this example. Agent a_m will then need to test the resource bounded attack relation decision procedure $\langle \Psi, \beta \rangle$ $undercut_k$ $\langle \Phi, \alpha \rangle$. To accomplish this, a_m will need to employ its automated reasoner to determine whether there exists $\langle \Psi, \beta \rangle$ in $\mathcal{A}(\Theta_{spec} \cup \Theta_{int})$ such that $\Phi \cup \{\beta\} \vdash_k \bot$. While argument evaluation as described in Section 2.2.1 is more complicated than simply testing a single attack relation, for the sake of brevity we focus on a single attack relation decision, which will nonetheless be a component of argument evaluation. In the case that an argument $\langle \Psi, \beta \rangle$ cannot be found such that $\langle \Psi, \beta \rangle$ $undercut_k$ $\langle \Phi, \alpha \rangle$, then $\langle \Phi, \alpha \rangle$ will be considered acceptable.

4.4.1.3 Automated Reasoning Component

In this example, we make use of first-order predicate logic for the underlying logic of the argumentation system, similar to the approaches found in [PSJ98, BH05, Alo04]. For experimental purposes, we make use of the Prover9 automated theorem prover [McC].

While Prover9 allows the use of several variants of the resolution calculus, to keep our example simple, we configure it to use only the binary resolution inference rule (and factorization). Further, we have modified the Prover9 system to impose a limit on the number of inference steps performed during the search for a proof, which we use as the resource limit of the deduction component.

4.4.2 "Normal" Interaction Scenario

In this example, the manager agent a_m has posted a request for a design of a component which can produce fixed amplitude waveforms of 20mA at a frequency of 30Hz, represented by *amp20mA* and *genFreq30Hz* respectively. This is represented in the formal semantics defined above as :

$$\exists x : ProvF(x, genFreq30Hz) \wedge\ HasP(x, amp20mA). \tag{C_1}$$

4.4.2.1 The Design Proposal

An external contractor agent a_c analyzes this request and uses its knowledge of various electric components to construct a design which incorporates the WG3000 wave generator. However, its knowledge of this component specifies that the WG3000 must be connected to an adequate power supply in order to function, which is represented as:

$$HasP(wg3000, amp20mA). \tag{C_2}$$

$$\begin{gathered}\exists x : (UseC(wg3000) \wedge Conn(x, wg3000) \wedge ProvF(x, power20mA)) \\ \rightarrow ProvF(x, genFreq30Hz).\end{gathered} \tag{C_3}$$

Agent a_c also has knowledge of a specific power supply which can be used for this purpose, the PSU423, which provides the necessary 20mA output required for use with the WG3000.

$$UseC(psu423) \rightarrow ProvF(psu423, power20mA). \tag{C_4}$$

Along with the specifications of the components described in $C_2, ..., C_4$, agent a_c must include design specifications on their use and interconnections, expressed as:

$$UseC(wg3000) \land UseC(psu423) \land Conn(psu423, wg3000). \qquad (C_5)$$

The design is expressed by agent a_c as the argument $\langle \Phi, \alpha \rangle = \langle \{C_2, ..., C_5\}, C_1 \rangle$[1]. Upon receipt of the proposal from a_c, agent a_m can verify the argument's validity as described in Section 2.3, by testing that $C_2 \land ... \land C_5 \vdash C_1$ using its automated reasoner, as well as the conditions that $C_2 \land ... \land C_5 \nvdash \bot$ and $\neg \exists S' \subset \{C_2, ..., C_5\} : S' \nvdash \bot \land S' \vdash C_1$.

4.4.2.2 The Attacking Argument

However, let us assume that a_m has a further condition in Θ_{int} which specifies that the component must be compliant with the Restriction of Hazardous Substances (RoHS) directive, which agent a_c has not taken into account in its design:

$$\forall x : UseC(x) \leftrightarrow HasP(x, rohsCompliant). \qquad (C_6)$$

Included in this directive is a restriction on the use of various toxic substances, including the use of lead:

$$\begin{aligned} &\forall x : HasP(x, rohsCompliant) \leftrightarrow \\ &\neg(HasP(x, containsLead) \lor HasP(x, containsCadmium) \\ &\lor HasP(x, containsHexavalentChromium)). \end{aligned} \qquad (C_7)$$

Further, a_m has the knowledge that the PSU423 power supply unit contains lead:

$$HasP(psu423, containsLead). \qquad (C_8)$$

The clauses C_6, C_7 and C_8 can then be used as support for the conclusion:

$$\neg UseC(psu423). \qquad (C_9)$$

[1]Formulae are entered into the theorem prover in the order they appear here

This can be formed into the argument $\langle \Psi, \beta \rangle = \langle \{C_6, C_7, C_8\}, C_9 \rangle$. Through the use of its automated reasoner, a_m can determine that $\Phi \cup \{\beta\} \vdash \bot$ due to the conflicting clauses $UseC(psu423)$ and $\neg UseC(psu423)$, and so the argument $\langle \Psi, \beta \rangle$ attacks the argument $\langle \Phi, \alpha \rangle$. To test this example, the Prover9 automated theorem prover [McC] was configured to use the standard simple weighted term selection search control. In this configuration, it took the prover a single inference step to prove that $\Phi \cup \{\beta\} \vdash \bot$, and so the prover could determine that $\Phi \cup \{\beta\} \vdash_k \bot$, and therefore that $\langle \Psi, \beta \rangle$ *undercut*$_k$ $\langle \Phi, \alpha \rangle$, for any $k \geq 1$. Agent a_m could then respond to a_c with the attacking argument $\langle \Psi, \beta \rangle$. The Prover9 code for this example can be found in Appendix A.1, including statistics on the theorem prover's execution during the search for this proof.

4.4.3 Malicious Argumentation Scenario

Knowing the attack to the argument $\langle \Phi, \alpha \rangle$ described above, a_c could make use of a resource exhaustion strategy to overwhelm agent a_m's deductive reasoning capacity before a_m can determine that its argument $\langle \Psi, \beta \rangle$ attacks $\langle \Phi, \alpha \rangle$. This would be of advantage to a_c if, for example, it has a surplus of PSU423 units and needs to get rid of them. Using further knowledge of the automated reasoner used by a_m, such as the resource bounds and the search control, a_c can modify the argument $\langle \Phi, \alpha \rangle$ so that a_m will not find the contradiction within the given inference limit, in which case a_m will accept the argument due to the mapping of the case of resource bounds exhaustion as described above in Section 4.2.2.

4.4.3.1 Implication Chaining

The malicious argumentation technique of implication chaining described in Section 4.3.2 can be used by agent a_c to construct a new argument $\langle \Phi', \alpha \rangle$. This can be accomplished by modifying the clause C_5 to make the contradicting term $UseC(psu423)$ the consequent of an implication chain with a true antecedent so that it cannot be resolved with the term

$\neg UseC(psu423)$ before the inference limit k is reached. An example of this would be to create a sequence of fake parts, whose use implies the use of the next fake part in the sequence, and finally the part in question. The clause C_5 can then be rewritten as:

$$\begin{aligned}&UseC(wg3000) \wedge UseC(fake_1)\\ &\wedge\, UseC(fake_1) \rightarrow UseC(fake_2)\\ &\wedge ... \wedge UseC(fake_{n-1}) \rightarrow UseC(fake_n)\\ &\wedge\, UseC(fake_n) \rightarrow UseC(psu423)\\ &\wedge\, Conn(psu423, wg3000).\end{aligned} \tag{$C_5^{n'}$}$$

Note that this modification of the argument's support still does not violate the minimality criteria of arguments presented in Section 2.3, as the removal of any one of these added clauses would violate the condition that $\Phi \vdash \alpha$.

To test the attack relation between the new argument $\langle \Phi', \alpha \rangle$ and the attacking argument $\langle \Psi, \beta \rangle$, agent a_m's resource bounded automated reasoner is used to determine:

$$\{C_2, ..., C_4, C_5^{n'}\} \cup \{\neg UseC(psu423)\} \vdash_k \perp \tag{P_1}$$

If the resource limit k is exhausted before P_1 can be proven by resolving the terms $UseC(psu423)$ and $\neg UseC(psu423)$ to a contradiction, then it cannot be established that $\langle \Psi, \beta \rangle attacks_k \langle \Phi', \alpha \rangle)$. To demonstrate the use of this method of resource exhaustion, Prover9 is used to test P_1 with incremental values of k. Values of n for the clause $C_5^{n'}$ were then found which exceed k inferences before P_1 is proven. The results of this can be found in Table 4.1, and the Prover9 code for this example can be found in Appendix A.2, along with more statistics from the execution of the prover on this code.

4.4.3.2 Tautology Injection

The tautology injection technique described in Section 4.3.3 can also be used to exhaust the resources allocated to agent a_m's attack relation decision procedure. A new argument $\langle \Phi'', \alpha \rangle$ can be constructed by agent a_c, in which the term $UseC(psu433)$ in modified to

inference limit (k)	chain length (n)
100	49
200	99
300	149
400	199
500	249

Table 4.1: Results for Implication Chains

be the consequent of a consequence relation with a large tautology as the antecedent. By conjoining sufficient repetitions of a single tautology, agent a_m will be unable to determine that $\langle \Psi, \beta \rangle$ $undercut_k$ $\langle \Phi'', \alpha \rangle$. To accomplish this, the clause C_5 in the example can then be rewritten as:

$$\begin{aligned}
&UseC(wg3000)\wedge \\
&((((a_1 \rightarrow b_1) \wedge (b_1 \rightarrow c_1)) \rightarrow (a_1 \rightarrow c_1)) \\
&\wedge ... \wedge \\
&(((a_n \rightarrow b_n) \wedge (b_n \rightarrow c_n)) \rightarrow (a_n \rightarrow c_n))) \\
&\rightarrow UseC(psu423) \wedge Conn(psu423, wg3000).
\end{aligned} \tag{$C_5^{n''}$}$$

As with the method of implication chaining described above, the Prover9 automated theorem prover is used to test the attack relation between the new argument $\langle \Phi'', \alpha \rangle$ and agent a_m's attacking argument $\langle \Psi, \beta \rangle$ by evaluating the following condition:

$$\{C_2, ..., C_4, C_5^{n''}\} \cup \{\neg UseC(psu423)\} \vdash_k \bot \tag{P_2}$$

The results of evaluating P_2 for different values of k are shown in Table 4.2, where a value of n has been found such that the resources k allocated to the procedure are exhausted before P_2 can be proven. The Prover9 code for this example can be found in Appendix A.3, along with statistics for the execution of the prover on this code.

inference limit (k)	repetitions (n)
100	3
200	3
300	3
400	4
500	4

Table 4.2: Results for Tautology Injection

Chapter 5

Defense Strategies

After having examined malicious strategies agents may employ in an open multi-agent argumentation system, we now turn to considerations of how such strategies may be defended against. In Section 5.1, modifications are described to incorporate defense strategies into the general argumentative agent model. Section 5.2 then investigates an initial defense strategy using pattern matching to detect instances of malicious argumentation strategies, particularily those described in Section 4.3. Other potential defense strategies are then briefly described in Section 5.3, followed by a discussion of the general principles of defense against malicious argumentation in Section 5.4.

5.1 Modifying the Agent Model

In this section, modifications to the argumentation model described in Chapter 2 are considered, with the intention of incorporating defense strategies against malicious argumentation. Malicious argumentation strategies targetting different decision procedures in the argumentation model, as described in Section 4.2, will require defense strategies to be implemented in different components of an argumentative agent. In order to maintain the scope of this thesis, given that the investigation into malicious strategies in Chapter 4 focused primarily on the argument evaluation decision procedure, so too will our discussion of defense strategies.

In general, a defense strategy for a particular decision procedure can be implemented as a sort of "wrapper" for the procedure. That is to say, the defense strategy may perform pre-computations on the input to the procedure before the decision procedure itself is executed, and further there may be post-computations on the output of the decision procedure. It may also be necessary to modify the decision procedure itself so that the output is richer; that is, rather than a simply boolean result, the procedure could be modified to return additional information regarding its execution, with the intention of providing the defense strategy's post-computation with greater information regarding the execution of the procedure. While the wrapper model may not always be an applicable method of implementing a defense strategy against malicious argumentation, in general it is a good way to conceptualize the role of defense in the general argumentative agent model.

If the purpose of a defense wrapper for an agent's decision procedure is to detect instances of malicious argumentation strategies targetting that decision procedure, it must also be considered how the defense strategy will respond to a positive detection of a malicious strategy. In the case that a malicious strategy is not detected by the defense wrapper, the decision procedure it defends may be executed normally. When a malicious strategy is identified by the defense wrapper, the output of the procedure will necessarily need to be modified. However, this may not be as simple as inverting the output of the decision procedure. For instance, consider a defense strategy wrapping the attack relation decision procedure used during the argument evaluation procedure. In the case when a malicious strategy is identified by the defense wrapper, yet the attack relation decision procedure nonetheless does not identify an attack relation between the given arguments, it still cannot be decided that an attack between the arguments does in fact exist. To respond with a counter-attack on the basis that the argument being tested has been identified as implementing a malicious strategy may break the felicity condition of the counter-attack performative. It may be that if sufficient resources were available to complete the decision procedure, an attack relation would still not be found. It may be that the malicious strategy was intended to exhaust the resources of a different attack relation decision, and not the particular one being tested. Nonetheless, if the defense strategy identifies a malicious strategy being employed in a particular argument, this result cannot be ignored.

Incorporating defense strategies into the argumentation model cannot therefore be as simple as implementing defense wrappers designed to detect instances of malicious argumentation strategies. These wrappers may be used to detect malicious argumentation, yet the model must also be modified to incorporate means of reacting to the detection of malicious argumentation. Towards this end, the dialogue game may be modified by adding additional performatives relating to detection of malicious argumentation strategies. Furthermore, the termination conditions of the dialogue may be conditional on the outcome of the defense wrappers; an agent may simply terminate the interaction in the case that it detects its opponent employing a malicious argumentation strategy. The agent's logic, that is, the logic of justification, may also be modified to incorporate identification of malicious arguments. These topics will be discussed further in Section 5.3, albeit in no great depth, as a proper investigation of these topics requisites a research project of greater scope than is afforded by this thesis. The primary focus of this chapter will be to investigate a defense strategy for identifying a particular malicious argumentation strategy targeting the attack relation decision procedure; considerations of how the outcome of this defense wrapper will be incorporated into the larger agent model, i.e: the dialogue game, logic, etc., must be left for future research into this topic.

5.2 Detecting Patterns of Malicious Arguments

The malicious argumentation strategies described in Section 4.3 are based on the principle of expanding the syntactic form of an argument while retaining the semantic content, such that the resources allocated to the deduction system used in an agent's decision procedures will be exhausted before a particular result can be derived. This is accomplished by modifying the argument using relatively simple syntactic patterns, such as long chains of simple implications (Section 4.3.2) or conjuctions of simple tautologies to construct large, scalable tautologies (Section 4.3.3). These strategies are designed to target the attack relation decision procedure used by the argument evaluation decision procedure, in order to “hide” particular attack relations by exhausting the resources of the deduction system before a conflict between arguments can be decided. A defense strategy could then be implemented by constructing a wrapper around the attack relation decision procedure designed to detect these patterned syntactic constructions.

5.2.1 Pattern Matching Implication Chains

The implication chaining malicious argumentation strategy described in Section 4.3.2 makes use of a simple, scalable syntactic modification of an argument’s contents to exhaust the resource bounds of an opponent’s attack relation decision procedure. As the modification is constructed in a purely mechanical manner, it creates an easily identifiable pattern in the syntactic form of the argument, which can be detected through the use of a fairly simple pattern matching algorithm. The attack relation decision procedure can then be modified by implementing a defense wrapper around the procedure in order to identify instances of implication chain patterns before the attack relation decision procedure is executed.

There is a danger, however, in simply matching chains of implications and identifying them as instances of a malicious argumentation strategy. For instance, a “chain” consisting of a single implication will likely occur often in entirely legitimate arguments; even larger chains of implications can be used in arguments that are constructed with no malicious intent. The pattern matching must therefore be performed in such a way as to minimize the possibility of such false-positive identifications of malicious argumentation strategies. For implication chains then, there is a particular type that can be identified as an instance of malicious argumentation with a high degree of certainty: those implication chains of such a length that the resources allocated to the deduction system would be exhausted before their final consequent can be derived. While it may be possible that such implication chains occur "naturally" within the system, it is far more likely that they are instances of a malicious argumentation strategy.

A pattern to detect implication chains of a particular length can then be constructed as follows:

$\lambda_1,\ \lambda_1 \to \lambda_2,\ \ldots,\ \lambda_{n-1} \to \lambda_n$

To match this pattern against a given argument $\langle \Phi, \alpha \rangle$, each λ_i $(1 \leq i \leq n)$ needs to be substituted with an appropriate formula from the support set $\Phi \subseteq \mathcal{L}$. The chain length n for the pattern can then be set to a minimal value such that the resources allocated to the prover will be exhausted before the final consequent matched by λ_n can be derived. While it may be that implication chains employed by a malicious resource exhaustion strategy are much longer than what is matched by this pattern, nonetheless the initial segment of the chain will be matched by this pattern, regardless of how much longer the actual chain is.

Example 27. Let $\langle \Phi, \alpha \rangle = \langle\ \{P_1, P_1 \to P_2\ , \ldots,\ P_{m-1} \to P_m, P_m \to Q\},\ Q\ \rangle$

Assume a resource bound of k inference steps, and that $m = 2k$.

Let $n = k$ be the length of the implication chain pattern to be matched.

The following substitutions can then be identified:

$\lambda_1 \approx P_1,\ \lambda_2 \approx P_2,\ \ldots,\ \lambda_n \approx P_{m/2}$

Even though the implication chain in $\langle \Phi, \alpha \rangle$ continues from $P_{(m/2)+1}$ to P_m, an implication chain pattern of length n has been found in the argument that will at the very least exhaust the resources available, regardless of how much longer it is.

5.2.2 Counter-Measure Against this Defense Strategy

As shown above, the simple implication chaining malicious argumentation strategy described in Section 4.3.2 can be detected through the use of a fairly simple pattern matching defense strategy. However, using the concept of implication chains as the basic scaling pattern, more complex malicious argumentation strategies can be developed for which the defense strategy will be inadequate. The pattern matching technique described above is based on the idea that the consequent of one link in the chain will be the antecedent of the next link. Knowing this, a malicious strategy based on implication chaining could be developed that will retain the basic principle of constructing arbitrarily long chains of implications, yet does not follow this consequent/antecedent matching pattern.

Example 28. $P_0,\ P_0 \to P_1,\ (P_0 \wedge P_1) \to P_2\ , ...,\ (P_{n-2} \wedge P_{n-1}) \to P_n$

This example of an implication chaining strategy achieves the same effect as the basic implication chaining strategy from Section 4.3.2, yet will not be detected by the pattern matching defense strategy described above in Section 5.2.1, as the consequent of each implication is not syntactically equivalent to the antecedent of the next implication in the chain.

As with the basic implication chaining strategy, this new strategy constructs implication chains through a highly patterned, mechanical construction. It would therefore be a simple matter to construct a pattern matching defense strategy to detect this malicious argumentation strategy as well. However, it would also be a simple matter to construct another variant of the basic implication chaining strategy for which a pattern matching defense strategy has not been constructed yet. For every patterned malicious argumentation strategy, a pattern matching defense strategy can be constructed, and conversely, for every defense based on pattern matching, new malicious patterns can be constructed that are not detected by the defense strategy.

5.2.3 General Limitations of Pattern Matching

The basic idea behind the type of malicious argumentation strategies discussed in this thesis is to construct a syntactic modification that both preserves the original semantic content of an argument, and can be arbitrarily scaled to exhaust any resource bound imposed on the underlying deduction system. This scaling is accomplished by repeating a particular syntactic pattern, connected either through implication as seen in the implication chaining strategy (Section 4.3.2), a conjuction as used in the tautology injection strategy (Section 4.3.3), or other means not yet explored. Defense strategies can then be implemented to detect these patterns, and identify when the occurance of such a pattern is likely being used to exhaust the deduction system's resource bounds. However, as shown in Section 5.2.2 above, it can often be a relatively simple task to construct new variants of the basic scaling pattern that are not detected by the defense strategies. A defense strategy can implement pattern matching techniques to address all instances of known malicious argumentation patterns, and even anticipate new patterns that have not yet been encountered, but it cannot address every possible syntactic manipulation that could be used to exhaust the deduction system's resource bounds.

Consider the tautology injection strategy from Section 4.3.3; using a repeated conjunction of a single simple tautology, a large syntactic construct can be built to exhaust a particular resource bound. While a pattern matching defense could be developed to identify the particular tautology used in the example, it is well known that there are an infinite number of possible tautologies, of greater and more elaborate complexity than the one used herein. It would be impossible to account for every single tautology in the pattern matching algorithm. Further, a simple modification to the malicious strategy would be to use different tautologies in conjunction with one another, rather than simply repeating the same tautology. This would further increase the difficulty in detecting this type of malicious argumentation strategy. Regardless of the number of patterns a defense strategy accounts for, there will always be an infinite number of patterns it cannot detect.

Further, pattern matching itself is not a trivial operation. While the algorithms are, in general, less expensive than those used by the deduction system, they nonetheless require resources to execute. The more patterns that a particular defense strategy accounts for, the greater the resources required to complete its analysis of a given argument. It will likely be necessary to impose resource bounds on any defense strategy implemented in an argumentation system, and as such, it may be that malicious argumentation strategies are developed specifically to exhaust the resources allocated to an agent's defense mechanisms. Furthermore, it may be that, through happenstance, legitimate arguments are identified as containing malicious argumentation patterns by the defense mechanisms. As the number of patterns tested by a defense strategy grows, so too does the chance of false-positive identifications of malicious argumentation. Therefore, while pattern matching can be used as a means of identifying instances of particular malicious argumentation strategies, it is by no means a perfect defense strategy against malicious argumentation.

5.3 Other Potential Defense Strategies

While the pattern matching defense strategy discussed in Section 5.2 above is an obvious means of implementing a defense strategy, it is certainly not the only means of defending against malicious argumentation. In this section, other potential defense strategies will be addressed, albeit to no great depth. This includes considerations of how the outcome of malicious argumentation detection mechanisms (the *defense wrappers* described in Section 5.1) can be incorporated into the larger agent model, as well as how the logic of justification and the protocol governing the dialogue game might be modified to integrate a defense strategy into the agent model. This discussion of defense strategies should be seen as merely illustrative rather than exhaustive; there are myriad means by which an agent may defend against malicious arguments, and a thorough investigation of these strategies is well beyond the scope of this thesis. As with the previous section, this discussion should be understood as a motivation for future research into this expansive topic, rather than good and proper research in and of itself.

5.3.1 Mapping the Outcome of Defense Wrappers

As discussed in Section 5.1, decision procedures susceptible to malicious argumentation can be modified to incorporate a *defense wrapper* designed to detect instances of malicious argumentation. This may be implemented as a pre-processing pattern matching algorithm as described in Section 5.2, or as a post-processing algorithm making use of information resulting from the execution of the decision procedure. Regardless of the specific implementation, however, the outcome of the defense wrapper needs to be incorporated into the larger agent model.

Rather than considering every decision procedure in the argumentative agent model, we shall focus this discussion on the attack relation decision procedure in the argument evaluation process, as has been done numerous times throughout this thesis. Consider then an implementation of the pattern matching defense wrapper; in the case that the defense wrapper does not identify a potential malicious argumentation pattern in the input to the attack relation decision procedure, the decision procedure itself can execute normally. However, when the pattern matching wrapper identifies an instance of a malicious argumentation pattern, the correct outcome of the procedure is not entirely clear. As mentioned above, even a positive identification of a malicious pattern does not imply that the malicious pattern was designed to hide the specific attack relation being currently tested. The decision procedure cannot then return a positive result that an attack relation does in fact exist between the input arguments being tested. To do so would possibly violate the felicity conditions of the counter-attack performative; that is, the agent might respond with a counter-attack argument that is not in fact a counter-attack.

However, given that the defense wrapper identified the input as containing a malicious pattern, it would be remiss for the system to simply ignore this result. Rather than strictly modifying the boolean output of the wrapped decision procedure (in this case, positive / negative identification of an attack relation), a more complex response can be considered. The first we shall consider would be to simply reject the argument. That is not to say that the argument is rejected because it can be attacked; when an attack is found, the agent responds with a counter-attack, as is the way of the dialectic process of justification. Rather, a rejection would likely constitute the removal of that particular argument from the dialectic process entirely, or in the most extreme case, a termination of the dialogue itself. The specific nature of such a rejection would have to be decided at the level of the dialectic protocol, but regardless of the implementation, this is a fairly extreme response to the detection of a pattern that only holds the possibility of being employed in a malicious resource exhaustion strategy.

Rejection of an argument is not the only means of incorporating the results of a defense wrapper. Another strategy might be to "flatten" the results of the pattern matching algorithm. That is to say, similar to the way that inference rules in the deductive system match patterns of logical formulae to manipulate the syntactic representation of the semantic content in a logic, the pattern matching defense wrapper can be seen as a pre-processing syntactic manipulation designed to identify and reduce very specific syntactic patterns. If the pattern matching defense wrapper described in Section 5.2.1 above were to identify a long chain of inferences eventually ending in the consequent Q, the argument fed as input to the attack relation decision procedure can be syntactically modified by simply replacing the chain with the consequent Q. As the pattern matching defense wrapper is designed to identify specific implication chain patterns of a particular minimum length, this pseudo inference rule is nowhere near complete (albeit, hopefully, sound). Nonetheless, implementation of very specific "inference rules" in the defense wrapper designed to detect known patterns of malicious argumentation may be a significant reduction in computational resources required, in comparrison to only using the general inference rules of the deduction system itself, therefore allowing the system to efficiently detect counter-attacks for particular arguments.

5.3.2 Modifying the Logic of Justification

As described in Section 2.2.1, the logic of justification is based on the notion of an *extension*; a particular subset of the arguments presented in the dialogue game (or, more abstractly, in the argumentation framework) that satisfies a particular criteria. Those arguments belonging to an extension defined by the acceptability semantics used by the system are justified, whereas those that do not belong to the extension are not justified. However, the notions of justification and acceptability can be modified to incorporate the computational resource bounds imposed on a practical implementation of an argumentation system. For instance, an extension can be segregated into two classes of arguments: those for which the process of justification could be completed within the given computational resource bounds, and those for which the resources allocated to the process were exhausted before completion. These could be identified as the classes *complete acceptance* and *resource exhausted acceptance*, respectively.

The segregation of these two classes follows quite simply from the outcome of the decision procedures. It is only necessary for the decision procedure to report whether it was able to complete its decision within the imposed resource bounds. For instance, when performing argument evaluation, if any of the attack relation decisions cannot be completed within resource bounds, the outcome of evaluation is placed into the *resource exhausted* class. The outcome may still be acceptable or unacceptable, but nonetheless the result is "weakened" due to being a member of the resource exhausted class.

Conversely, if the entire process was able to complete within the given resource bounds, this can be seen as a “stronger” result of the argument evaluation procedure, with the arguments then being members of the *complete acceptance* class.

At this point, the terms “stronger” and “weaker” have been used to relate to the difference between membership in these classes. However, these are intuitive notions; to properly implement such a defense strategy, it would be necessary to formally define the notions of “stronger” and “weaker” with respect to the argumentation model as a whole. This could entail a modification of the attack relation conditions, in that weaker acceptable arguments may not attack stronger ones. Further, this may require modifications at the level of the dialectic protocol; weak-accept, strong-accept, weak-counter-attack, strong-counter-attack, etc.. Given that agents engage in an argumentative dialogue for some higher purpose, such as deliberation, negotiation or pursuasion, it must then be considered how weak and strong notions of justification affect this higher level decision procedure. This modification which, at face value, seems relatively simple, would in fact require heavy modifications to the argumentation model as a whole. Nonetheless, it may prove to be a valuable defense strategy against malicious resource exhaustion strategies.

5.3.3 Modifying the Dialectic Protocol

As already mentioned in the preceding sections, the incorporation of a defense strategy into the larger argumentation model may require modifications to dialectic protocol. New performatives may be introduced to the protocol, such as the argument rejection performative discussed in Section 5.3.1, or the weak/strong acceptance and counter-attack performatives discussed in Section 5.3.2 above. Implementing such new performatives is not as simple as introducing them at the level of the dialectic protocol; it must also be considered how their condition’s are satisfied by the lower levels of the argumentation model, and what effects their introduction will have on the larger decision procedure for which the dialectic procedure is employed, such as deliberation, negotiation, persuasion or otherwise.

In addition to the various performatives already briefly discussed, there are others that warrant consideration. For instance, in the case that resources are exhausted while computing a particular decision procedure, instead of rejecting an argument or demoting its status as described above, an agent may request a simplified version of the argument. This request could be formalized as an additional performative in the dialectic protocol. However, the addition of a new performative into the dialectic protocol may introduce new complications which require deeper consideration. For instance, it is necessary to consider how an agent may determine whether the new "simplified" argument it recieves in response to this request is actually semantically similar to the original argument for which resources were exhausted. Given that the agent did not have sufficient resources to complete its analysis of the original argument, it has no real basis for comparison with the new argument. Further, it must be considered how such a performative opens up the potential for abuse by malicious agents; an agent may repeatedly request simplifications on arguments for which it has sufficient resources to perform decisions on, with the aim of strategically consuming the resources allocated to the dialogue as a whole.

Given that the rules of the dialogue game govern the interactions between agents, any additions to these rules will invariably introduce new complexities into the agent interactions. While these complexities may allow for certain desirable behaviours that would be impossible given the constraints of a simpler dialectic protocol, they may also introduce undesirable consequences which may not be immediately perceptible. Particularly in open multi-agent systems, agents may not "play by the rules"; performatives often have conditions dependent on an agent's internal state, which cannot be externally verified. Therefore, while modifications to the dialectic protocol may provide an important tool for incorporating defense strategies against malicious argumentation, such modifications may also introduce the possibility of new malicious strategies, and so must be performed conservatively and with attention paid to the potential for abuse.

5.4 General Principles of Defense

As seen in the discussion of a mechanism for detecting instances of malicious argumentation by pattern matching in Section 5.2, as well as the general strategies for incorporating defense mechanisms into the larger argumentative agent model discussed in Section 5.3, none of these methods provides a perfect means to defend against malicious argumentation strategies. Whether there are too many different malicious argumentation strategies to account for, that the defense mechanisms themselves must be resource bounded, or that the implementation of the defense strategy would open the possibility of new malicious argumentation strategies, all defenses against malicious argumentation are necessarily imperfect. All defense strategies require a trade-off; rejecting arguments for which argument evaluation can't complete within the given resource bounds provides a higher degree of security, yet restricts the scope of legitimate arguments that can be employed within the system; matching a greater number of malicious argumentation patterns in turn requires a greater amount of resources to be allocated to pattern matching, restricting the resources available to other decision procedures involved in argumentation. Any defense strategy devoloped to counteract malicious argumentation may in turn be undermined by further developments in malicious argumentation strategies.

Nonetheless, it is important for defense strategies against malicious argumentation to be investigated and developed. If no defenses against malicious argumentation are considered, then even the simplest resource exhaustion strategy can be employed to great effect. The situation of defending against malicious argumentation can be seen as analogous to that of detecting and defending against virii and malware [Ayc08]; it would be impossible to construct a "perfect" defense that could detect any virus or piece of malicious software. However, it is still necessary to detect virii and malware that are well known, so that the simplest virus can't wreak havok on our computing equipment. Further, virus and malware defense strategies are subject to the same practical resource bounds as defenses against malicious argumentation; a computer can't spend all of its resources examining software for malicious code, as the computing resources need to also be used by the actual software a user wishes to run. Therefore, both in argumentation defense strategies and virus and malware detection, a balance must be struck between the robustness of the defense strategy and the resources allocated to the actual tasks the system is designed to accomplish.

The general principles of defense against malicious argumentation can then be summarized as follows:

1. *No Perfect Defense* - all defenses against malicious argumentation are necessarily imperfect, in that they cannot account for every possible malicious argumentation strategy

2. *Balanced Defense* - all defense strategies must make trade-offs, either between robustness and security, between coverage of attacks and resource consumption, or otherwise

3. *Raise the Bar* - at the very least, a defense strategy should prevent the simplest malicious argumentation strategies from being effective, so that a malicious agent requires a certain degree of expertise to surpass the defenses

By recognizing the essential inadequacies of defense strategies, the developer of an argumentation system will never be under the illusion that their system is impenetrable. Understanding the balance that needs to be struck, and the dimensions upon which a defense strategy needs to be balanced, is crucial for implementing an effective defense strategy that does not undermine the purpose of the system being developed. And finally, it is imperative that a defense strategy be implemented in such a way as to at least make it difficult for an agent to successfully employ a malicious argumentation strategy. By addressing the simple malicious argumentation strategies, such as those described in this thesis, greater expert knowledge is required by a malicious agent, therefore reducing the number of agents able to successfully employ a malicious argumentation strategy. In this way, the risk of using argumentation in open-multi agent systems can be reduced; it cannot be eliminated entirely, but it must be reduced to an acceptable level before such systems can be considered for popular use in commercial applications or otherwise.

Chapter 6

Conclusion

The goal of this thesis was to introduce the potential for malicious actions performed by agents involved in argumentation in an open multi-agent system, as well as to provide an outline of the difficulties inherit in defending against such malicious argumentation. To accomplish this, a general account of malicious resource exhaustion strategies targeting the intractability of decisions in the underlying formal logical language was given, as well as a practical example of how such a resource exhaustion strategy could be instantiated to manipulate the outcome of the argument evaluation procedure. Further, the topic of defense was addressed from a high-level perspective, analyzing a few strategies that could be used to counter-act certain malicious argumentation strategies, and concluding with a general analysis of defense and the necessary insufficiency of any particular defense strategy.

In summary, although argumentation is a powerful means of communicative interaction in open multi-agent systems, the advantages gained through the use of argumentation are not without consequence. By employing an underlying formal logical language to convey the content of arguments, agents gain an expressive power that cannot be attained through simple symbolic methods of communication. However, the advantages gained through the expressive power of a logical language are balanced against the intractability of certain procedures on this language, such as deciding consistency and logical entailment. Due to the necessity of limiting the resources available to agents during their respective turns in a dialectic interaction, resource exhaustion is an unavoidable possibility. In a closed argumentation system, the effects of resource exhaustion can be minimized through the use of fairness criteria and a greater control over the actions of agents in the system. However, in open multi-agent argumentation systems, fairness of resource consumption cannot be guaranteed by protocol, and the system as a whole has little control over the actions of individual agents. As agents in an open-multi agent system generally execute on uncontrolled client software, not only may agents be self-interested, but they may attempt to perform malicious actions to gain an advantage over other agents in the system. Defense strategies may be implemented to counter-act malicious resource exhaustion strategies, such as matching syntactic patterns or altering the rules of the dialogue game controlling the agents' interactions. However, such defense strategies are necessarily imperfect; while certain patterns may be detected, it would be impossible to exhaustively detect every pattern that could be used by a malicious agent; while the rules of the dialogue game may be modified to facilitate defense strategies, these modifications may also open up new potential malicious exploitations of the system. Nonetheless, it is necessary to implement some means of defense against malicious argumentation, or these systems will be susceptible to the simplest exploits. The goal of defense is therefore to increase the effort required of a malicious agent, so as to minimize the potential for exploitation while still retaining the core functionality of the argumentation system.

6.1 Future Work

There are myriad directions in which the concepts presented in this thesis could be explored. While a preliminary examination of malicious agent strategies in open multi-agent argumentation systems has been given, it is by no means an exhaustive investigation of this topic. The discussion of malicious strategies presented in this thesis has primarily been focused on the argument evaluation procedure, yet even for this decision procedure, only two methods of implementing a malicious strategy were given; an investigation into further implementations of resource exhaustion strategies targeting the argument evaluation procedure would likely yield many new techniques. Apart from argument evaluation, the argument validity and dialogue game decision procedures were also briefly evaluated for their susceptibility to malicious argumentation; a deeper investigation into particular malicious argumentation techniques targeting these decision procedures would also be warranted.

The discussion of strategies to defend against malicious argumentation presented in this thesis is both preliminary and high-level; the primary goal of this thesis was to establish the potential for malicious activity in open multi-agent argumentation systems. A deeper investigation into defense strategies is certainly an important direction for future research into this broad topic. Both in the development of techniques to counter-act particular forms of malicious argumentation, as well as research into general strategic considerations of defense against malicious argumentation, there is a vast potential for investigation into this aspect of practical argumentation.

For both the development of new malicious argumentation strategies and investigation into defense techniques to counter-act malicious argumentation, the development of proper implementations would be invaluable. Currently, the field of argumentation has little in the way of implemented systems, however this is quickly changing. The current state of theoretical research into automated argumentation has reached a point where the development of applications is warranted, and the community is certainly increasing its focus on this direction of research. By working in conjunction with existing projects in development, or even developing separate software in order to perform experiments on malicious argumentation and defense strategies, new insights into the strategic considerations of practical argumentation systems will likely arise that would not otherwise.

Conversely, there are also theoretical aspects of malicious argumentation that would be valuable to investigate. While this topic is primarily concerned with a practical aspect of implemented argumentation systems, it would nonetheless be useful to introduce a higher degree of formalism to the concepts discussed within this thesis. This may include a formal characterization of vulnerability due to the particular features of an argumentation system, or a more precise definition of a malicious resource exhaustion strategy by introducing concepts from formal proof theories. Further, it may be valuable to construct proofs of some of the central topics of this thesis, such as the necessary inadequacy of defense strategies, that are now only dealt with through natural language arguments.

It should therefore be apparent that the topic of malicious argumentation strategies in open multi-agent systems holds great potential for future research projects. Investigation into either the development and understanding of malicious strategies and defense strategies, both from a practical and a theoretical perspective, would likely produce some interesting and valuable results. Given that the argumentation community is just now emerging from its theoretical infancy into a mature field warranting the development of practical applications, continued research into these strategic manipulations and the possibility of malicious exploitation is not only justified, but necessary.

Appendix A

Prover9 Example Code

A.1 Unmodified Example Code

```
set(raw).

set(binary_resolution).

set(print_gen).

set(prolog_style_variables).


formulas(sos).

    %%% Support {C2,...,C5} from argument <{C2, ..., C5}, C1> %%%


    % Clause C2 %

    HasP(wg3000, amp20mA).


    % Clause C3 %

    exists X (UseC(wg3000) & Conn(X, wg3000) & ProvF(X, power20mA))

        -> ProvF(wg3000, genFreq30Hz).


    % Clause C4 %

    UseC(psu423) -> ProvF(psu423, power20mA).


    % Clause C5 %

    UseC(wg3000).
    UseC(psu423).

    Conn(psu423, wg3000).


    %%% Conclusion C9 from argument <{C6,C7,C8}, C9> %%%


    -UseC(psu423).


end_of_list.
```

A.1.1 Statistics

```
Given=0. Generated=7. Kept=7. proofs=1.

Usable=0. Sos=0. Demods=0. Limbo=6, Disabled=7. Hints=0.

Kept_by_rule=0, Deleted_by_rule=0.

Forward_subsumed=0. Back_subsumed=0.

Sos_limit_deleted=0. Sos_displaced=0. Sos_removed=0.

New_demodulators=0 (0 lex), Back_demodulated=0. Back_unit_deleted=0.

Demod_attempts=0. Demod_rewrites=0.

Res_instance_prunes=0. Para_instance_prunes=0. Basic_paramod_prunes=0.

Nonunit_fsub_feature_tests=0. Nonunit_bsub_feature_tests=0.

Megabytes=0.02.

User_CPU=0.01, System_CPU=0.03, Wall_clock=0.
```

A.2 Implication Chaining Example Code

```
set(raw).

set(binary_resolution).

set(print_gen).

set(prolog_style_variables).


formulas(sos).

   %%% Support {C2,...,C5'n} from argument <{C2, ..., C5'n}, C1> %%%


   % Clause C2 %

   HasP(wg3000, amp20mA).


   % Clause C3 %

   exists X (UseC(wg3000) & Conn(X, wg3000) & ProvF(X, power20mA))

      -> ProvF(wg3000, genFreq30Hz).


   % Clause C4 %

   UseC(psu423) -> ProvF(psu423, power20mA).


   % Clause C5'n (where n = 5) %

   UseC(wg3000).

   UseC(fake1).

   UseC(fake1) -> UseC(fake2).

   UseC(fake2) -> UseC(fake3).

   UseC(fake3) -> UseC(fake4).
```

```
    UseC(fake4) -> UseC(fake5).

    UseC(fake5) -> UseC(psu423).

    Conn(psu423, wg3000).



    %%% Conclusion C9 from argument <{C6,C7,C8}, C9> %%%

    -UseC(psu423).


end_of_list.
```

A.2.1 Statistics

```
Given=14. Generated=21. Kept=21. proofs=1.

Usable=9. Sos=4. Demods=0. Limbo=0, Disabled=19. Hints=0.

Kept_by_rule=0, Deleted_by_rule=0.

Forward_subsumed=0. Back_subsumed=7.

Sos_limit_deleted=0. Sos_displaced=0. Sos_removed=0.

New_demodulators=0 (0 lex), Back_demodulated=0. Back_unit_deleted=0.

Demod_attempts=0. Demod_rewrites=0.

Res_instance_prunes=0. Para_instance_prunes=0. Basic_paramod_prunes=0.

Nonunit_fsub_feature_tests=0. Nonunit_bsub_feature_tests=18.

Megabytes=0.03.

User_CPU=0.01, System_CPU=0.03, Wall_clock=0.
```

A.3 Tautology Injection Example Code

```
set(raw).

set(binary_resolution).

set(print_gen).

set(prolog_style_variables).


formulas(sos).

    %%% Support {C2,...,C5''n} from argument <{C2, ..., C5''n}, C1> %%%


    % Clause C2 %

    HasP(wg3000, amp20mA).


    % Clause C3 %

    exists X (UseC(wg3000) & Conn(X, wg3000) & ProvF(X, power20mA)) ->

        ProvF(wg3000, genFreq30Hz).


    % Clause C4 %

    UseC(psu423) -> ProvF(psu423, power20mA).


    % Clause C5''n (where n = 5) %

    UseC(wg3000).

    (   (((a1 -> b1) & (b1 -> c1)) -> (a1 -> c1)) &

        (((a2 -> b2) & (b2 -> c2)) -> (a2 -> c2)) &

        (((a3 -> b3) & (b3 -> c3)) -> (a3 -> c3)) &

        (((a4 -> b4) & (b4 -> c4)) -> (a4 -> c4)) &
        (((a5 -> b5) & (b5 -> c5)) -> (a5 -> c5))
    ) -> UseC(psu423).
    Conn(psu423, wg3000).



    %%% Conclusion C9 from argument <{C6,C7,C8}, C9> %%%
    -UseC(psu423).

end_of_list.
```

A.3.1 Statistics

```
Given=5893. Generated=127926. Kept=23554. proofs=1.
Usable=22. Sos=2. Demods=0. Limbo=0, Disabled=24559. Hints=0.
Kept_by_rule=0, Deleted_by_rule=0.
Forward_subsumed=104372. Back_subsumed=23529.
Sos_limit_deleted=0. Sos_displaced=0. Sos_removed=0.
New_demodulators=0 (0 lex), Back_demodulated=0. Back_unit_deleted=0.
Demod_attempts=0. Demod_rewrites=0.
Res_instance_prunes=0. Para_instance_prunes=0. Basic_paramod_prunes=0.
Nonunit_fsub_feature_tests=58916. Nonunit_bsub_feature_tests=47063.
Megabytes=10.16.
User_CPU=8.08, System_CPU=13.62, Wall_clock=108.
```

Bibliography

[Alo04] Eduardo Alonso. Rights and argumentation in open multi-agent systems. *Artificial Intelligence Review*, 21(1):3–24, 2004.

[AMP00] Leila Amgoud, N Maudet, and Simon Parsons. Modelling dialogues using argumentation. In *Proceedings of the Fourth International Conference on Multi-Agent Systems*, pages 31–38, 2000.

[APM00] Leila Amgoud, Simon Parsons, and N Maudet. Arguments, dialogue, and negotiation. *aa*, pp:338–342, 2000.

[Ayc08] J. Aycock. *Spyware and Adware*, volume 50 of *Advances in Information Security*. Springer-Verlag New York Inc, 2008.

[BG09] Pietro Baroni and Massimiliano Giacomin. *Semantics of Abstract Argument Systems*, pages 25–44. Springer Publishing Company, Inc., 2009.

[BH01] Philippe Besnard and Anthony Hunter. A logic-based theory of deductive arguments. *Proceedings Of The National Conference On Artificial Intelligence*, 128:203–235, 2001.

[BH05] Philippe Besnard and Anthony Hunter. Practical first-order argumentation. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 20, page 590. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[BIP88] Michael E. Bratman, David J. Israel, and Martha E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(3):349–355, September 1988.

[BN02a] L. Brito and J. Neves. Argument exchange in heterogeneous electronic commerce environments. *Proceedings of the first international joint*, pages 410–417, 2002.

[BN02b] L. Brito and J. Neves. Properties and Complexity in Feasible Logic-Based Argumentation for Electronic Commerce. *Lecture notes in computer science*, pages 90–100, 2002.

[Dun95] Phan Minh Dung. On the Acceptability of Arguments and its Fundamental Role in Non-Monotonic Reasoning, Logic Programming and N-Person Games. *Artificial Intelligence*, 77:321–357, 1995.

[Dun03] P.E. Dunne. Prevarication in dispute protocols. In *Proceedings of the 9th international conference on Artificial intelligence and law*, pages 12–21. ACM, 2003.

[Fuc96] M. Fuchs. Powerful Search Heuristics Based on Weighted Symbols, Level and Features. *Proceedings of the 9th Florida Artificial Intelligence Research Symposium*, 1996.

[KM03] Antonis C Kakas and P. Moraitis. Argumentation based decision making for autonomous agents. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 883–890. ACM New York, NY, USA, 2003.

[Lou98] R.P. Loui. Process and policy: Resource-bounded nondemonstrative reasoning. *Computational Intelligence*, 14(1):1–38, 1998.

[McC] W. McCune. Prover9 manual. `http://www.cs.unm.edu/~mccune/mace4/manual/2009-11A/`, as seen on Oct. 8, 2010.

[MP02] Peter McBurney and Simon Parsons. Dialogue Games in Multi-Agent Systems. *Informal Logic*, 22(3):257–274, 2002.

[Pol91] John L Pollock. A Theory of Defeasible Reasoning. *International Journal*, 6:33–54, 1991.

[PS99] Henry Prakken and Giovanni Sartor. A system for defeasible argumentation, with defeasible priorities. *Practical Reasoning*, pages 510–524, 1999.

[PSJ98] Simon Parsons, Carles Sierra, and N Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261, 1998.

[PWA03] Simon Parsons, Michael Wooldridge, and Leila Amgoud. Properties and complexity of some formal inter-agent dialogues. *Journal of Logic and Computation*, 13(3):347, 2003.

[RL08] Iyad Rahwan and Kate Larson. Mechanism design for abstract argumentation. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 1031–1038. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[SC00] Katia Sycara and H. Chi Wong. Adding Security and Trust To Multiagent Systems. *Applied Artificial Intelligence*, 14(9):927–941, October 2000.

[WK95] D.N. Walton and E.C.W. Krabbe. *Commitment in dialogue: Basic concepts of interpersonal reasoning.* State Univ of New York Pr, 1995.