

REASONING ABOUT COMPLEX AGENT KNOWLEDGE

ONTOLOGIES, UNCERTAINTY, RULES AND BEYOND

YUZHANG FENG

B.Sc.(Hons). NUS

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2010

Acknowledgement

I would like to take this opportunity to express my sincere gratitude to those who assisted me, in one way or another, with my Ph.D.

First and foremost, I would like to thank my Honor's Year Project and Ph.D. advisor Dr. Dong Jin Song and co-advisor Dr. Daqing Zhang for their never-ending enthusiasm, guidance, support, encouragement and insight throughout the course of my postgraduate study. Their diligent reading and insightful and constructive criticism of early drafts and many other works made this thesis possible.

I am grateful to Prof. Tan Chew Lim and Prof. Rudy Setiono for the critical comments on this thesis. I am also thankful to the external reviewer and numerous anonymous referees who have reviewed this thesis and previous publications that are parts of this thesis and their valuable comments have contributed to the clarification of many ideas presented in this thesis.

This thesis was in part funded by the projects "Formal Design Methods and DAML" and "Advanced Ontological Rules Language and Tools" supported by the Defence Science and Technology Agency of Singapore, "Rigorous Design Methods and Tools for Intelligent Autonomous Multi-Agent Systems" supported by Ministry of Education of Singapore, and "Systematic Design Methods and Tools for Developing Location Aware, Mobile and Pervasive Computing Systems" supported by the Media Development Authority of Singapore. My gratitude also goes to National University of Singapore for the generous financial support, in forms of scholarship and conference travel allowance.

I also wish to thank my seniors, Dr. Sun Jun, Dr. Chen Chunqing, and my cousin Dr. Li Yuan-Fang for their friendship, collaboration and generous sharing of research experience. I am also lucky to have my former and current lab mates from the formal methods group for their friendship and funny chit chat which helped me go through the long and sometimes rough way of Ph.D. study.

I wish to thank sincerely and deeply my parents who have raised me, taught me and supported me all these years and who always have faith in me.

I owe thanks to my beloved wife Hao Na. I would not have completed my thesis without your ceaseless love, encouragement and patience. Lastly I would like to dedicate this thesis to my lovely newborn daughter Yuanxin.

Contents

1	Introduction	1
1.1	Motivations and Goals	1
1.2	Thesis Outline	5
1.2.1	Chapter 2 - Background Overview	5
1.2.2	Chapter 3 - Checking Ontology-based Agent Knowledge	6
1.2.3	Chapter 4 - Checking Agent Knowledge With Uncertainty	7
1.2.4	Chapter 5 - Checking Rule-based Agent Knowledge	8
1.2.5	Chapter 6 - Checking Higher-order Agent Knowledge	8
1.2.6	Chapter 7 - Conclusion	9
1.3	Publications	9
2	Background Overview	11
2.1	Semantic Web	11
2.1.1	Semantic Web Languages	12
2.1.2	Semantic Web Reasoners	15
2.2	Prototype Verification System	17
2.3	Constraint Logic Programming	19

3	Checking Ontology-based Agent Knowledge	23
3.1	PVS Semantics for OWL DL	25
3.1.1	Basic Concepts	25
3.1.2	Class Descriptions	28
3.1.3	Axioms	36
3.1.4	Assertions	41
3.1.5	SWRL Rules	42
3.1.6	Proof Support for PVS	43
3.2	Reasoning about Ontologies in PVS	44
3.2.1	Standard SW Reasoning	44
3.2.2	Checking SWRL & Beyond	48
3.3	Chapter Summary	56
4	Checking Agent Knowledge With Uncertainty	59
4.1	OWL Abstract Syntax	61
4.2	OWL Semantics	63
4.3	Belief-augmented Frames	68
4.3.1	Belief Augmented Systems	69
4.3.2	Predefined Beliefs	71
4.3.3	Belief Augmented Frames Logic	71
4.4	Belief-augmented OWL (BOWL)	72
4.4.1	BAF	73
4.4.2	BOWL Syntax Extension	73
4.4.3	BOWL Semantic Extension	74
4.5	Reasoning about BOWL	79

4.5.1	Class Membership	79
4.5.2	Property Membership	83
4.5.3	Simple Implementation in $\text{CLP}(\mathcal{R})$	85
4.6	Case Study	86
4.6.1	The Sensor Ontology	86
4.6.2	Computing Confidence Values of Sensors	89
4.7	Chapter Summary	91
5	Checking Rule-based Agent Knowledge	93
5.1	Semantic Web Rules Language	95
5.2	Analyzing Agent Rule Bases	96
5.2.1	Inconsistency	97
5.2.2	Redundancy	107
5.2.3	Circularity	109
5.3	Prototype Implementation	111
5.4	Case Study	114
5.5	Chapter Summary	119
6	Checking Higher-order Agent Knowledge	123
6.1	Epistemic Logic	126
6.1.1	Semantics	131
6.1.2	A Classical Example	133
6.1.3	Reasoning about Epistemic Logics - The Model Checking Approach	134
6.2	Reasoning Framework	137
6.2.1	Proof Systems	140

6.2.2	Theorem Sets	150
6.2.3	Reasoning Systems	151
6.2.4	Reasoning Rule Sets	153
6.2.5	Framework Workflow	154
6.3	Examples	158
6.3.1	Formalizing the System	158
6.3.2	Constructing and Proving Reasoning Rules	160
6.3.3	Proof Strategies	161
6.3.4	Generalizing the Example	165
6.4	Chapter Summary	170
7	Conclusion	173
7.1	Main Contribution of the Thesis	173
7.2	Future Work Directions	177
7.2.1	Reasoning about Semantic Web Services	177
7.2.2	Combining Knowledge Uncertainty and Rules	180
7.2.3	Higher Automation for PVS Verification	180

Summary

Agent-based technology is one of the most vibrant and important areas of research and development that have emerged in information technology in recent years. An intelligent agent is an autonomous entity which observes and acts upon an environment and directs its activity towards achieving goals.

The distinguishing characteristics of intelligent agents are that they are autonomous, responsive, proactive and social. The key features of intelligent agents that has made them so is that intelligent agents have their knowledge of the world and themselves and that they have the capability to make deductions. Hence it is our belief that knowledge representation and reasoning is one of the most important research areas in agent-based technologies.

In the current stage, we have identified four challenges related to the field of agent knowledge representation and reasoning. (1) The *interoperability* and *heterogeneity* problem is how agents with different domains of discourse, employing different problem solving paradigms, and with different assumptions about their world and each other, can be made to interact in an effective and scalable manner. (2) As agents have a necessarily partial perspective of their world, and because their problem domain is open, complex and distributed, they require sophisticated mechanisms for reasoning with *uncertain*, *incomplete* and *contradictory* information. (3) *Rules* are natural means to specify reactive and possibly proactive behavior. It is a challenge for agents to perform reasoning on and with such rules. (4) The knowledge of an intelligent agent typically deals with what agents consider possible given their current information. This includes knowledge about facts as well as higher-order information about information that other agents have. It is a challenging task to enable systematic design of such intelligent agents as the reasoning process of interacting agents can be extremely complex.

This thesis presents our contribution to the solutions to the challenges. More specifically we employ a formal modeling approach to verifying ontology-based agent knowledge. We also extend the current state-of-the-art ontology language with the ability to model certainty factors about facts and proposed the corresponding reasoning algorithms. We define a set of notion for the quality of agent rule base and provide an automated checking mechanism. Lastly we present a formal hierarchical framework for specifying and reasoning about higher-order agent knowledge.

Key words: Knowledge, reasoning, Semantic Web, ontology, epistemic logic

List of Tables

3.1	The Model of Scheduling Tasks	49
4.1	OWL class expressions & their interpretations	66
4.2	OWL axioms & their interpretations	67
4.3	OWL assertions & their interpretations	67
4.4	BOWL class expressions & their interpretations	76

List of Figures

2.1	Semantic Web Stack	13
4.1	OWL class expressions	61
4.2	OWL class axioms	62
4.3	OWL assertions	63
4.4	BOWL assertions	74
5.1	Prototype Screen Shot	114
6.1	Three Wise Men problem in DEMO	136
6.2	Framework Architecture	138
6.3	Logic Hierarchy	138
6.4	Axiomatization K	142
6.5	Axiomatization S5	144
6.6	Axiomatization S5C	145
6.7	Axiomatization PA	147
6.8	Axiomatization PAC	149
6.9	Framework Workflow	156
6.10	Simplified proof tree for the three wise men problem	162
6.11	Proof Fragment for K elimination	163

6.12	K introduction in natural deduction	164
6.13	Simplified proof tree for the base case	169
6.14	Simplified proof tree for the inductive case	169
7.1	The main components of the OWL-S ontology.	178

Chapter 1

Introduction

1.1 Motivations and Goals

Agent-based technology is one of the most vibrant and important areas of research and development to have emerged in information technology in recent years. In the field of artificial intelligence, an intelligent agent [117] is an autonomous entity which observes and acts upon an environment and directs its activity towards achieving goals. Intelligent agents are a relatively new paradigm for developing software applications. Currently, agents are the focus of intense interest on the part of many sub-fields of computer science and artificial intelligence. Agents are being used in an increasingly wide variety of applications, ranging from comparatively small systems such as email filters to large, open, complex, mission critical systems such as air traffic

control.

Intelligent agent represents a new way of analyzing, designing and implementing complex software system. For agent-based technologies, the objectives are to create systems situated in dynamic and open environments, able to adapt to these environments and capable of incorporating autonomous and self-interested components. Agent-based systems provides concrete advantages such as: improving operational robustness with intelligent failure recovery, reducing sourcing costs by computing the most beneficial acquisition policies in online market and improving efficiency of manufacturing processes in dynamic environments.

There are some distinguishing characteristics of intelligent agents [66].

- Autonomous: Agents should be able to perform the majority of their problem solving tasks without the direct intervention of humans or other agents, and they should have a degree of control over their own actions and their own internal state.
- Responsive: Agents should perceive their environment (which may be the physical world, a user, a collection of agents, the Internet, etc.) and respond in a timely fashion to changes that occur in it.
- Proactive: Agents should not simply act in response to their environment, they should be able to exhibit opportunistic, goal-directed behavior and take the

1.1. Motivations and Goals

initiative where appropriate.

- Social: Agents should be able to interact, when they deem appropriate, with other artificial agents and humans in order to complete their own problem solving and to help others with their activities.

One of the key features of intelligent agents that has made them autonomous, responsive, proactive and social is that intelligent agents have their knowledge and perception of the world and themselves. Many of the problems machines are expected to solve will require extensive knowledge about the world. Among the things that AI needs to represent are: objects, properties, categories and relations between objects; situations, events, states and time; causes and effects; and knowledge about knowledge.

Humans are intelligent creatures not only because they possess vast amount of knowledge, but also because humans have the ability to reason about their knowledge. One classical example for deductive reasoning is that from the facts that “*all humans are mortal*” and that “*socrates is a human*”, one can conclude that “*socrates is mortal*”. In order for agents to be intelligent, it is also important for agents to be able to represent large quantity of knowledge in an effective way and to have an efficient way of inferring new knowledge from existing knowledge.

We have identified a number of challenges related to knowledge representation and

reasoning of intelligent agents at the current stage of agent-based research.

- **Interoperability and Heterogeneity:** Agent-based research is only just beginning to grapple with problems associated with the inevitable heterogeneity of its problem solving components. The basic problem is how agents with different domains of discourse, employing different knowledge representation schemes, different problem solving paradigms, and with different assumptions about their world and each other, can be made to interact in an effective and scalable manner.
- **Uncertainty, Vagueness and Incompleteness:** As agents have a necessarily partial perspective of their world, and because their problem domain is open, complex and distributed, they require sophisticated mechanisms for reasoning with uncertain, incomplete and contradictory information if they are to exhibit the desired degree of flexibility and robustness.
- **Rules-based Agent Knowledge and Reasoning:** Agents are situated in an environment and exhibit reactive and possibly proactive behavior. Rules are natural means to specify these forms of agent behavior. It is a challenge for agents to perform reasoning on and with such rules.
- **Multi-agent Knowledge Representation and Reasoning:** The area of multi-agent systems is traditionally concerned with formal representation of the

1.2. Thesis Outline

mental state of autonomous agents in a distributed setting. The knowledge of an intelligent agent typically deals with what agents consider possible given their current information. This includes knowledge about facts as well as higher-order information about information that other agents have. It is a challenging task to enable systematic design of such intelligent agents as the reasoning process of interacting agents can be extremely complex.

In this thesis it is our goal to address the above challenges by focusing on providing various reasoning support for knowledge-based multi-agent systems.

1.2 Thesis Outline

1.2.1 Chapter 2 - Background Overview

Chapter 2 is devoted to an introduction of the languages, notions and tools that are used in this thesis.

One of the knowledge representation formalisms used in this thesis is the Semantic Web languages. Hence we first introduce the Semantic Web technology in general and introducing a family of ontology languages, focusing on the current W3C recommendation for ontology language, Web Ontology Language (OWL). We present the

syntax and semantics of the main language constructs, followed by a brief discussion on their tool support.

Two specification and verification frameworks are used in this thesis for the purpose of knowledge reasoning, namely the Prototype Verification System [87] and the constraint logic programming technique [61]. Therefore we will give a brief overview of the two frameworks in Chapter 2. We briefly describe the PVS modelling language and how formal proofs can be constructed and checked in the PVS theorem prover. We also introduce the language features of Constraint Logic Programming and its operational model. We choose to use Chapter 2 to provide a general introduction of the formalisms and tools and we explain details to later chapters where they are used.

1.2.2 Chapter 3 - Checking Ontology-based Agent Knowledge

In Chapter 3, we demonstrate the ability of the PVS specification language and theorem prover in expressing ontology-based agent knowledge and checking ontology-related properties. Specifically, we define the semantics of ontology language OWL in the PVS language. By automatically transforming OWL and ontologies into PVS specification theories, core ontology reasoning services, namely concept subsumption, satisfiability and instantiation checking can be performed in the powerful PVS the-

1.2. Thesis Outline

orem prover. Further more we can also check for properties beyond the modelling power of the ontology language using our approach.

1.2.3 Chapter 4 - Checking Agent Knowledge With Uncertainty

As the Semantic Web is an extremely complex, globally distributed and constantly evolving medium, it is often the case that information on different sites is incomplete or inconsistent with respect to each other. Hence, intelligent agents need to cope with knowledge with uncertainty.

The Belief-augmented Frames (BAF) [105] is an extension to the Minsky frame knowledge representation system [81]. Its unique feature is that it adds a belief and a disbelief value to each frame in a frame system.

Chapter 4 is devoted to proposing to integrate BAF with OWL DL to form a new ontology language BOWL (Belief-augmented OWL) that can easily express beliefs. We systematically extend the syntax and semantics of OWL. We also define reasoning algorithms for the proposed language.

1.2.4 Chapter 5 - Checking Rule-based Agent Knowledge

Chapter 5 targets the reasoning support for rule-based intelligent agents. Information in the Semantic Web is semantically marked up so that not only human-to-human communication is possible, intelligent agents can also interpret and process the data. Ontology languages like Web Ontology Language (OWL) [79] provide the basic vocabularies for representing complex agent knowledge. In addition, Semantic Web Rules Language (SWRL) [56] provides a convenient mechanism for specifying Horn-style rules.

In Chapter 5, we define a set of notions for the correctness of rule base of an agent's knowledge. We demonstrate how to use the combination of the state-of-the-art Semantic Web reasoners and the constraint logic programming technique to help designer of such rule-based intelligent agent systems detect anomalies in the rule base.

1.2.5 Chapter 6 - Checking Higher-order Agent Knowledge

In Chapter 6, we presented a formal hierarchical framework for specifying and reasoning about higher-order agent knowledge, i.e. knowledge about knowledge. We encoded a hierarchy of epistemic logics K , $S5$, $S5C$, PAC and $PAL-C$ in the PVS specification language. We show that the PVS theorem prover can be used as a powerful reasoner for the logics, especially for systems with an arbitrary number of

1.3. Publications

intelligent agents.

1.2.6 Chapter 7 - Conclusion

Chapter 7 concludes the thesis, summarizes the main contributions and discusses future work directions.

1.3 Publications

Most of the work presented in this thesis has been published or accepted in international conferences proceedings.

The work on PVS semantics for OWL and SWRL and on using PVS to reason about OWL and SWRL (Chapter 3) is one of the first attempts in the literature to provide reasoning facilities for SWRL and beyond OWL. It has been published in the First International Colloquium on Theoretical Aspects of Computing (ICTAC 2004, Guiyang, China) [28].

The work on using BAF to incorporate uncertainty in ontology based agent knowledge (Chapter 4) has been published in the Twelfth IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007, Auckland, New Zealand) [38]. An extended journal version which includes in addition the reasoning algorithms

and CLP implementation has been submitted to Innovations in Systems and Software Engineering, A NASA Journal. [30].

The work on checking various types of SWRL rule anomalies by using a combination of a DL reasoner and CLP (Chapter 5) has been accepted for publication by the 4th IEEE International Conference on Secure Software Integration and Reliability Improvement (SSIRI 2010, Singapore) [39]. Furthermore, an extended journal version has been submitted for review to Transactions on Autonomous and Adaptive Systems [40].

The work on the integrated tools environment was presented at The Twelfth Asia-Pacific Software Engineering Conference (APSEC 2005, Taipei) [29].

The work on the formal hierarchical framework for specifying and reasoning about higher-order agent knowledge in PVS (Chapter 6) has been published in the Ninth International Conference on Formal Engineering Methods (ICFEM 2007, Boca Raton, USA) [26]. An extended version including the support for systems of arbitrary number of agents has been submitted for review to Formal Aspects of Computing [27].

I have also made contributions to other published work [41, 32, 31].

Chapter 2

Background Overview

2.1 Semantic Web

Proposed by Tim Berners-Lee et al., the Semantic Web (SW) [18] is a vision of next generation of the Web. Unlike conventional web as we have now, the Semantic Web is a platform for inter-machine data and information exchange, filtering, integration, etc., across organizational boundaries without human supervision. It extends the current web and reaches its full potential by making it truly ubiquitous and ready for the machines.

Semantic Web ontologies give precise and unambiguous meaning to Web resources, enabling software agents to understand them. An ontology is a specification of a

conceptualization [49]. It is a description of the concepts and relationships for a particular application domain. Ontologies can be used by software agents to precisely categorize and deduce knowledge.

2.1.1 Semantic Web Languages

Ontology languages are the building blocks of the Semantic Web. The development of ontology languages takes a layered approach. Depicted in Figure 2.1, the Semantic Web languages are constructed on top of mature languages and standards such as the XML [118], Unicode and Uniform Resource Identifier (URI) [16]. In the rest of this section, we briefly present some important languages in the Semantic Web.

Built on top of XML, the Resource Description Framework (RDF) [77] is a model of metadata defining a mechanism for describing resources without assumptions about a particular application domain. RDF describes web resources in a simple triplet format: $\langle \textit{subject predicate object} \rangle$, where *subject* is the resource of interest, *predicate* is one the properties of this resource and *object* states the value of this property. Besides this basic structure, a set of basic vocabularies are defined to describe RDF ontologies. This set includes vocabularies for defining and referencing RDF resources, declaring containers such as bags, lists, and collections. It also has a formal semantics that defines the interpretation of the vocabularies, the entailment between RDF graphs, etc. RDF Schema [22] provides facilities to describe RDF data. RDF Schema allows

2.1. Semantic Web

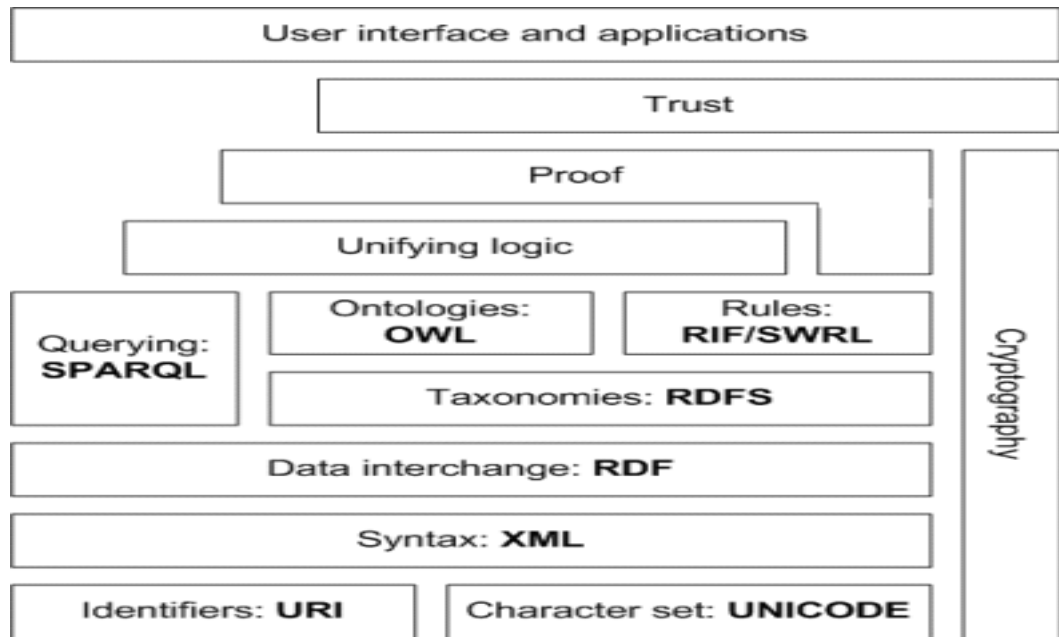


Figure 2.1: Semantic Web Stack

structured and semi-structured data to be mixed together, which makes them hard for machines to process.

The syntactic ambiguity and relatively limited expressiveness of RDF Schema is partially overcome by the DARPA Agent Markup Language (DAML) [112], which is built on top of RDF Schema and based on description logics. DAML pooled effort with the Ontology Inference Layer project [19] to produce the ontology language DAML+OIL. It provides a richer set of language primitives to describe classes and properties than RDF Schema and allows only structured data.

In 2004, a new ontology language based on DAML+OIL, the Web Ontology Language (OWL) [79] became the W3C Recommendation. It consists of three sublanguages:

OWL Lite, DL & Full, with increasing expressiveness. These languages are designed for user groups with different requirements. OWL Lite & DL are decidable but Full is generally not. The undecidability of OWL Full comes from relaxing certain constraints from OWL DL. For example, OWL Full does not enforce the mutual exclusiveness between classes, properties, data values and individuals. DAML+OIL is most comparable to OWL DL, which is a notational variance of description logic *SHOIN*(\mathcal{D}) [57].

Although the design of OWL has taken into consideration of the different expressiveness needs of different user groups, it is still not expressive enough. Some very desirable properties cannot be expressed even in OWL Full. An important reason for this is that although the language provides a relatively rich set of language primitives for describing classes, it does not provide as many primitives for describing properties. For example, it does not support property composition. In the light of this weakness, Horrocks and Patel-Schneider proposed an extension to OWL, the OWL Rules Language (ORL) [55] which is syntactically and semantically coherent to OWL.

The major extensions of ORL are the inclusion of Horn clause rules and variable declarations. The rules are in the form of **antecedent** \rightarrow **consequent**, where both antecedent and consequent are conjunctions of atoms: class membership, property membership, individual (in)equalities or built-ins. Informally, a rule means that if the antecedent holds, then the consequent must also hold. ORL is now known as

2.1. Semantic Web

Semantic Web Rule Language (SWRL) [56], with some sets of built-ins for handling data type, such as numbers, boolean values, strings, date and time, etc.

2.1.2 Semantic Web Reasoners

Besides ontology languages, we also witness the growth of ontology reasoners in the recent years. Here we survey some well known reasoners.

Closed world machine (CWM) [17] is a general-purpose data processor for the Semantic Web. Implemented in Python and command-line based, it is a forward chaining reasoner for RDF.

Triple [98] is an RDF query, inference and transformation language. It does not have a built-in semantics for RDF Schema, allowing semantics of languages to be defined with rules on top of RDF. This feature of Triple facilitates data aggregation as user can perform RDF reasoning and transformation under different semantics.

Fast Classification of Terminologies (FaCT) [54] is a TBox reasoner that supports automated concept-level reasoning. It does not support ABox (assertion Box, instance-level) reasoning. FaCT implements a reasoner for the description logic *SHIQ* [58]. It is implemented in Common Lisp.

KAON2 is a Java reasoner for *SHIQ* extended with the DL-safe fragment of SWRL. It implements a resolution-based decision procedure for general TBoxes (subsumption,

satisfiability, classification) and ABoxes (retrieval, conjunctive query answering). It comes with its own, Java-based interface, and supports the DIG-API.

RACER (**R**enamed **A**Box and **C**oncept **E**xpression **R**easoner) [50] is a reasoner for the description logic $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D})^-$ [51]. It can be regarded as (a) a SW inference engine, (b) a description logic reasoning system capable of both TBox and ABox reasoning and (c) a prover for modal logic Km. In the SW domain, RACER's functionalities include creating, maintaining and deleting ontologies, concepts, roles and individuals; querying, retrieving and evaluating the knowledge base, etc. It supports RDF, DAML+OIL and OWL. The RACER system has been commercialized and it is now known as RacerPro¹.

Pellet [99] is a free open-source Java-based reasoner for \mathcal{SROIQ} with simple data types (i.e., for OWL 1.1). It implements a tableau-based decision procedure for general TBoxes and ABoxes. Pellet employs many of the optimizations for standard DL reasoning as other state-of-the-art DL reasoners. It directly supports entailment checks and optimized ABox querying through its interface. Pellet supports the OWL-API, the DIG-API, and Jena interface [59].

¹<http://www.racer-systems.com/>

2.2 Prototype Verification System

Prototype Verification System (PVS) [87] is an integrated environment for formal specification and formal verification. It builds on over 25 years experience at SRI in developing and using tools to support formal methods. The primary purpose of PVS is to provide formal support for conceptualization and debugging in the early stages of the life cycle of the design of a hardware or software system. It supports a wide range of activities involved in creating, analyzing, modifying, managing, and documenting theories and proofs. The distinguishing feature of PVS is its synergistic integration of a highly expressive specification language and powerful theorem-proving capabilities.

A PVS specification consists of a collection of theories. Each theory consists of a signature for the type names and constants introduced in the theory, and the axioms, definitions, and theorems associated with the signature. A theory can build on other theories. A theory can be parametric in certain specified types and values. It is possible to place constraints, called assumptions, on the parameters of a theory.

The PVS specification language is based on simply typed higher-order logic. Within a theory, types can be defined starting from base types (Booleans, numbers, etc.) using the function, record, and tuple type constructions. The terms of the language can be constructed using function application, lambda abstraction, and record and tuple construction. The PVS type system also features dependent function, record,

and tuple type constructions. There is also a facility for defining a certain class of abstract datatype (namely well-founded trees) theories automatically.

PVS has a powerful interactive theorem prover [87]. The basic deductive steps in PVS are large compared with many other systems; there are atomic commands for induction, quantifier reasoning, automatic condition rewriting, simplification, etc. The primary emphasis in the PVS proof checker is on supporting the construction of readable proofs. User-defined proof strategies can be used to enhance the automation in the proof checker. On the whole, PVS provides more automation than a low-level proof checker such as LCF [47] and HOL [46], and more control than a highly automatic theorem prover such as Otter [78].

The prover maintains a proof tree. The users' goal is to construct a complete proof tree, in which all leaves (proof goals) are recognized as true. The proof goals in PVS are represented as sequents which consist of a list of formulae called the antecedents and a list of formulae called the consequents. The formal interpretation of a sequent is that the conjunction of the antecedents implies the disjunction of the consequents. Either or both of the antecedents and consequents may be empty. An empty antecedent is equivalent to the sequent being true, and an empty consequent is equivalent to the sequent being false. So if both are empty, the sequent is false. Every proof in PVS starts with a empty antecedent and a single consequent.

The PVS Prelude Library is a collection of basic theories about logic, functions,

2.3. Constraint Logic Programming

predicates, sets, numbers, and other datatypes. The theories in the prelude library are visible in all PVS contexts, unlike those from other libraries that have to explicitly imported. Broadly speaking, the prelude can be divided into the logic, functions, relations, induction, sets, numbers, sequences, sum types, quotient types, and mu-calculus.

2.3 Constraint Logic Programming

Constraint Logic Programming (CLP) [61] began as a natural combination of two declarative paradigms: constraint solving and logic programming. This combination helps make CLP programs both expressive and flexible, and in some cases, more efficient than other kinds of programs. CLP has been successfully applied to model programs and transition systems for the purpose of verification [63], showing that their approach performs better than the well-known state-of-the-art systems with higher efficiency.

The CLP scheme defines a class of languages based upon the paradigm of rule-based constraint programming, where $\text{CLP}(\mathcal{R})$ [62] is an instance of this class with the special support of real numbers.

A CLP atom is of the form $p(t_1, \dots, t_n)$ where p is a predicate symbol distinct from $=$, $<$, and \leq , and t_1, \dots, t_n are terms which can be predicates, variables or constants.

A variable starts with a upper-case letter whereas a constant starts with a lower-case letter.

A CLP rule is of the form

$$A_0 \text{ :- } \alpha_1, \dots, \alpha_k.$$

where each α_i , is either a primitive constraint (such as an arithmetic comparison) or an atom. The atom A_0 is called the head of the rule while the remaining atoms and primitive constraints are known collectively as the body of the rule. In case there are no atoms in the body, we may call the rule a fact or a unit rule.

A CLP program is defined as a finite set of rules. Rules in CLP have much the same format as those in PROLOG except that primitive constraints may appear with atoms in the body. The same applies to a CLP goal which is of the form

$$? \text{ - } \alpha_1, \dots, \alpha_k.$$

where each α_i , is either a primitive constraint or an atom.

Furthermore, each primitive constraint in a goal is classified as being either solved or delayed. A sub-collection of the atoms and constraints in a goal is sometimes called a subgoal of the goal.

The operational model of CLP can be explained as follows. let P denote a CLP program. Let G denote a goal with a subsequence of atoms denoted by A_1, \dots, A_n ,

2.3. Constraint Logic Programming

a subsequence of solved constraints denoted by $\alpha_1, \dots, \alpha_m$, and a subsequence of delayed constraints denoted by β_1, \dots, β_k . We say that there is a derivation step from G to another goal G' if one of the following holds:

- G' denotes a goal with a subsequence of atoms denoted by A_1, \dots, A_n , a subsequence of solved constraints denoted by $\alpha_1, \dots, \alpha_m, \beta_i$, and a subsequence of delayed constraints denoted by $\beta_1, \dots, \beta_{i-1}, \beta_{i+1}, \dots, \beta_k$. Furthermore the conjunction of the solved constraints in G' is solvable.
- P contains a rule R which can be renamed so that it contains only new variables and takes the form: the head atom is B , the subsequence of body atoms are denoted by B_1, \dots, B_s , and the subsequence of constraints in the body are $\gamma_1, \dots, \gamma_t$. In G' , the subsequence of atoms are $A_1, \dots, A_{j-l}, B_1, \dots, B_s, A_{j+l}, \dots, A_n$, the subsequence of solved constraints are $\alpha_1, \dots, \alpha_m$, the subsequence of delayed constraints are $\beta_1, \dots, \beta_k, A = B, \gamma_1, \dots, \gamma_t$. Furthermore, the conjunction of the solved constraints in G' is solvable.

Roughly speaking, in each derivation step either an atom gets expanded by applying a rule (by using techniques similar to unification and SLD-resolution) or a delayed constraint gets solved.

In an initial goal, all the constraints are delayed. A derivation sequence is a possibly infinite sequence of goals, starting with an initial goal, wherein there is a derivation

step to each goal from the preceding goal. A sequence is successful if it is finite and its last goal contains only solved constraints. A sequence is conditionally successful if it is finite and its last goal contains only solved and delayed constraints. Finally, a finitely failed sequence is finite, neither successful nor conditionally successful, and such that no derivation step is possible from its last goal.

Chapter 3

Checking Ontology-based Agent Knowledge

As we have identified in Chapter 1, one of the challenges for the realization of intelligent agents is the problem of interoperability and heterogeneity. More specifically, the basic problem is how agents with different domains of discourse, employing different knowledge representation schemes, different problem solving paradigms, and with different assumptions about their world and each other, can be made to interact in an effective and scalable manner.

The Semantic Web [18] technology is one of the most promising solutions to the problem. In the past decade, there has been increasing interest in the use of the Se-

semantic Web for semantically marking up information and services for intelligent agent to interpret, creating a platform for inter-machine data and information exchange, filtering, and integration across domain boundaries without human supervision.

Based on Description Logic [7], ontology languages such as DAML+OIL and (part of) OWL were originally designed to be decidable [112, 65], in order for intelligent software agents to *automatically* process data on the Semantic Web. However, the trade-off is the limited expressiveness, which forbids some very complex, but desirable properties to be specified. For this reason, OWL Rules Language (ORL) [55], which is later extended to Semantic Web Rule Language [56], has been proposed.

However such extensions are beyond the power of current state-of-the-art Semantic Web reasoners such as FaCT++ [108] and RACER [50]. This means that it is not possible to use such reasoners to reason about knowledge represented in the extended ontology language. Furthermore some very desirable, domain-specific properties, such as those with multiple quantification, still cannot be expressed even in the extended ontology language.

The lack of the above-mentioned knowledge reasoning capabilities prevents the realization of reliable intelligent agent-based systems and hence calls for a complementary reasoning support. In this chapter we demonstrate how PVS [87] can be used to reason about complex properties in ontology-based agent knowledge.

3.1. PVS Semantics for OWL DL

We begin by presenting the encoding of the OWL DL language semantic in the PVS specification language in Section 3.1. In Section 3.2 we demonstrate the different reasoning tasks that PVS can perform. Section 3.3 summarizes the main contributions of this chapter.

3.1 PVS Semantics for OWL DL

In order to use PVS to verify ontologies with SWRL axioms, it is necessary to define the PVS semantics for OWL DL & SWRL. In this section, we present the PVS semantic encoding of the OWL DL and SWRL language primitives. We start with basic concepts and then describe the encoding for class descriptions, axioms and assertions.

3.1.1 Basic Concepts

Everything in the Semantic Web is a resource. We model it by defining an abstract datatype in PVS.

```
Resource[Class,Individual,Dtype,Datatype,O_Property,D_Property: TYPE]: DATATYPE
BEGIN
  cls(c: Class): cls?
  idv(i: Individual): idv?
  dtv(d: Dtype): dtv?
  dtv(d: Datatype): dtv?
  oppt(p: O_Property): oppt?
  dppt(p: D_Property): dppt?
END Resource
```

It simply states the forms a resource can take. For example, `cls(c: Class): cls?` states that a resource can be of type (`cls?`) (for classes) by using the constructor `cls` and an argument `c` of type `Class` which is passed as one of the type parameters. Similarly a resource can be an individual, a datatype, a datatype, an object property and a datatype property.

To facilitate reasoning about literals, we define datatype also in a PVS abstract datatype as follows.

```
Datatype: DATATYPE
BEGIN
  natDV(n: nat): natDV?
  intDV(i: int): intDV?
  realDV(r: real): realDV?
  boolDV(b: bool): boolDV?
  stringDV(s: string): stringDV?
END Datatype
```

A datatype can be a natural number datatype, an integer datatype, a real number datatype, a boolean datatype, or a string datatype. This abstract datatype can be extended to support other semantic web datatypes.

3.1. PVS Semantics for OWL DL

Then in the main semantics definition theory we define `Class`, `O_Property`, `D_Property`, `Dtype` and `Individual` as uninterpreted nonempty types. By the PVS language semantics, uninterpreted types are mutually disjoint. Then we import the two abstract datatypes. Each class in OWL has a number of individuals associated with it, the instances of this class. Similarly each datatype has a set of datavalues associated with it. So we define two functions `instances` and `datavalues` that map a class to a set of individuals, and a datatype to a set of datavalues respectively. A property can be either an object properties or a datatype properties. Object properties have individuals as their ranges whereas datatype properties have datavalues as their ranges. Also every property has a set of tuples associated with it: the property instances. So we define two functions `sub_val_O` and `sub_val_D` that map an object property and a datatype property to their property instances respectively.

```
Class,O_Property,D_Property,Dtype,Individual: TYPE+

IMPORTING Datavalue
IMPORTING Resource[Class,Individual,Dtype,Datavalue,O_Property,D_Property]

instances: [(cls?) -> set[(idv?)]]
datavalues: [(dtt?) -> set[(dtv?)]]
sub_val_O: [(oppt?) -> set[[idv?), (idv?)]]]
sub_val_D: [(dppt?) -> set[[idv?), (dtv?)]]]
```

3.1.2 Class Descriptions

Classes in OWL are first-class citizens. A class description describes an OWL class, either by a class name or by specifying the class extension of an unnamed anonymous class. In this sub-section, we description how different class descriptions in OWL can be modelled in PVS. OWL distinguishes four types of class descriptions:

1. a class identifier (a URI reference)
2. an exhaustive enumeration of individuals that together form the instances of a class
3. a property restriction
4. a set operation on one or more class descriptions

The first type is special in the sense that it describes a class through a class name (syntactically represented as a URI reference). Two special basic classes are pre-defined in OWL. The class *Thing* contains all individuals and the class *Nothing* contains no individual. So we define the following classes and their associated axioms.

```

Thing: (cls?)
Thing_ax: AXIOM FORALL (i: (idv?): member(i, instances(Thing))

Nothing: (cls?)
Nothing_ax: AXIOM FORALL (i: (idv?): NOT member(i, instances(Nothing))

```

3.1. PVS Semantics for OWL DL

A class description of the "enumeration" kind is defined with the `owl:oneOf` property. The value of this built-in OWL property must be a list of individuals that are the instances of the class. This enables a class to be described by exhaustively enumerating its instances. So we define `oneOf` as a function from a list of individuals to a class. The semantics of the class description is defined as by the axiom `oneOf_ax`.

```
oneOf : [list[(idv?)] -> (cls?)]
oneOf_ax : AXIOM FORALL (li:list[(idv?)]),(c:(cls?)):
    (c = oneOf(li) IMPLIES FORALL (i:(idv?):
        (member(i,instances(c)) IFF member(i,li)))
```

A property restriction is a special kind of class description. It describes an anonymous class, namely a class of all individuals that satisfy the restriction. OWL distinguishes two kinds of property restrictions: value constraints and cardinality constraints.

A value constraint puts constraints on the range of the property when applied to this particular class description. There are three types of value constraint, namely *owl:allValuesFrom*, *owl:someValuesFrom* and *owl:hasValue*.

The property *owl:allValuesFrom* attempts to establish a maximal set of individuals as a class. It defines a class `c1` of all individuals `i1` for which it holds that if the pair `(i1,i2)` is in the property `p` then `i2` is an instance of class `c2`. So we model it as a function from a property `p` and a class `c2` to a class `c1` and specify its meaning as an axiom as follows. We also have a variant for the case where `p` is a datatype property and `c2` a datatype.

```

allValuesFrom_0 : [(oppt?), (cls?) -> (cls?)]
allValuesFrom_0_ax : AXIOM
  FORALL (c1, c2: (cls?)),(p: (oppt?):
    (c1 = allValuesFrom_0(p, c2) IMPLIES
      FORALL (i1: (idv?): (member(i1, instances(c1)) IFF
        FORALL (i2: (idv?):
          (member((i1,i2),sub_val_0(p)) IMPLIES
            member(i2,instances(c2))))))

allValuesFrom_D: [(dppt?), (dtt?) -> (cls?)]
allValuesFrom_D_ax: AXIOM
  FORALL (c1: (cls?)),(p: (dppt?)),(c2: (dtt?):
    (c1 = allValuesFrom_D(p,c2) IMPLIES
      FORALL (i1:(idv?): (member(i1,instances(c1)) IFF
        FORALL (i2: (dtt?):
          (member((i1,i2),sub_val_D(p)) IMPLIES
            member(i2,datavalues(c2))))))

```

The value constraint *owl:someValuesFrom* is a built-in OWL property that links a restriction class to a class description or a data range. A restriction containing an *owl:someValuesFrom* constraint describes a class of all individuals for which at least one value of the property concerned is an instance of the class description or a data value in the data range. In other words, it attempts to establish a maximal set of individuals as a class. It defines a class *c1* of all individuals *i1* for which it holds that there exists an individual *i2* such that (*i1*,*i2*) is in the property *p* and that *i2* is an instance of class *c2*. So we model it as a function from a property *p* and a class *c2* to a class *c1* and specify its meaning as an axiom as follows. We also have a variant for the case where *p* is a datatype property and *c2* a datatype.

3.1. PVS Semantics for OWL DL

```
someValuesFrom_0 : [(oppt?), (cls?) -> (cls?)]
someValuesFrom_0_ax : AXIOM
  FORALL (c1, c2: (cls?)), (p: (oppt?)):
    (c1 = someValuesFrom_0(p, c2) IMPLIES
      FORALL (i1: (idv?): (member(i1, instances(c1)) IFF
        EXISTS (i2: (idv?):
          (member((i1,i2), sub_val_0(p)) AND member(i2, instances(c2))))))

someValuesFrom_D: [(dppt?), (dtt?) -> (cls?)]
someValuesFrom_D_ax: AXIOM
  FORALL (c1: (cls?)), (p: (dppt?)), (c2: (dtt?):
    (c1 = allValuesFrom_D(p, c2) IMPLIES
      FORALL (i1: (idv?): (member(i1, instances(c1)) IFF
        EXISTS (i2: (dtt?):
          (member((i1,i2), sub_val_D(p)) AND member(i2, datavalues(c2))))))
```

The value constraint *owl:hasValue* is a built-in OWL property that links a restriction class to a value, which can be either an individual or a data value. A restriction containing a *owl:hasValue* constraint describes a class of all individuals for which the property concerned has at least one value semantically equal to the given value. In other words, it attempts to establish a maximal set of individuals as a class. It defines a class *c* of all individuals *i1* for which it holds that there exists an individual *i2* such that (*i1*, *i2*) is in the property *p*. So we model it as a function from a property *p* and an individual *i2* to a class *c* and specify its meaning as an axiom as follows. We also have a variant for the case where *p* is a datatype property and *i* a data value.

```

hasValue_O : [(oppt?), (idv?) -> (cls?)]
hasValue_O_ax : AXIOM
  FORALL (i2: (idv?)), (c: (cls?)), (p: (oppt?)):
    (c = hasValue_O(p, i2) IMPLIES
      FORALL (i1: (idv?): (member(i1, instances(c)) IFF
        member((i1,i2), sub_val_O(p))))

hasValue_D: [(dppt?), (dtv?) -> (cls?)]
hasValue_D_ax: AXIOM
  FORALL (i2: (idv?)), (c: (cls?)), (p: (dppt?)):
    (c = hasValue_D(p, i2) IMPLIES
      FORALL (i1: (idv?): (member(i1, instances(c1)) IFF
        member((i1,i2), sub_val_D(p))))

```

A cardinality constraint puts constraints on the number of values a property can take, in the context of this particular class description.

The cardinality constraint *owl:maxCardinality* is a built-in OWL property that links a restriction class to a data value belonging to the value space of the XML Schema datatype *nonNegativeInteger*. A restriction containing an *owl:maxCardinality* constraint describes a class of all individuals that have at most N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint. We have the following two variants of the semantics encoding, one for object properties and one for datatype properties.

3.1. PVS Semantics for OWL DL

```
maxCardinality_0: [(oppt?), nat -> (cls?)]
maxCardinality_0_ax: AXIOM
  FORALL (c: (cls?)), (op: (oppt?)), (n: nat):
    (c = maxCardinality_0(op, n) IMPLIES
      FORALL (i: (idv?): member(i, instances(c)) IFF
        EXISTS (s: finite_set[(idv?):
          card(s) = n AND subset?(image(sub_val_0(op), singleton(i)), s))

maxCardinality_D: [(dppt?), nat -> (cls?)]
maxCardinality_D_ax: AXIOM
  FORALL (c: (cls?)), (dp: (dppt?)), (n: nat):
    (c = maxCardinality_D(dp, n) IMPLIES
      FORALL (i: (idv?): member(i, instances(c)) IFF
        EXISTS (s: finite_set[(dtv?):
          card(s) = n AND subset?(image(sub_val_D(dp), singleton(i)), s))
```

The cardinality constraint *owl:minCardinality* is a built-in OWL property that links a restriction class to a data value belonging to the value space of the XML Schema datatype *nonNegativeInteger*. A restriction containing an *owl:minCardinality* constraint describes a class of all individuals that have at least N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint. We have the following two variants of the semantics encoding, one for object properties and one for datatype properties.

```

minCardinality_O: [(oppt?), nat -> (cls?)]
minCardinality_O_ax: AXIOM
  FORALL (c: (cls?)), (op: (oppt?)), (n: nat):
    (c = minCardinality_O(op, n) IMPLIES
      FORALL (i: (idv?): member(i, instances(c)) IFF
        EXISTS (s: finite_set[(idv?):
          card(s) = n AND subset?(s, image(sub_val_O(op), singleton(i))))

minCardinality_D: [(dppt?), nat -> (cls?)]
minCardinality_D_ax: AXIOM
  FORALL (c: (cls?)), (dp: (dppt?)), (n: nat):
    (c = minCardinality_D(dp, n) IMPLIES
      FORALL (i: (idv?): member(i, instances(c)) IFF
        EXISTS (s: finite_set[(dtv?):
          card(s) = n AND subset?(s, image(sub_val_D(dp), singleton(i))))

```

The cardinality constraint *owl:cardinality* is a built-in OWL property that links a restriction class to a data value belonging to the range of the XML Schema datatype *nonNegativeInteger*. A restriction containing an *owl:cardinality* constraint describes a class of all individuals that have exactly N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint. We have the following two variants of the semantics encoding, one for object properties and one for datatype properties.

3.1. PVS Semantics for OWL DL

```
cardinality_0: [(oppt?), nat -> (cls?)]
cardinality_0_ax: AXIOM
  FORALL (c: (cls?)), (op: (oppt?)), (n: nat):
    (c = cardinality_0(op, n) IMPLIES
      FORALL (i: (idv?): member(i, instances(c)) IFF
        EXISTS (s: finite_set[(idv?):
          card(s) = n AND s = image(sub_val_0(op), singleton(i)))

cardinality_D: [(dppt?), nat -> (cls?)]
cardinality_D_ax: AXIOM
  FORALL (c: (cls?)), (dp: (dppt?)), (n: nat):
    (c = cardinality_D(dp, n) IMPLIES
      FORALL (i: (idv?): member(i, instances(c)) IFF
        EXISTS (s: finite_set[(dtv?):
          card(s) = n AND s = image(sub_val_D(dp), singleton(i)))
```

An OWL class can also be described by using set operations on one or more class descriptions. There are three types of class description related to set operations, namely *owl:intersectionOf*, *owl:unionOf* and *owl:complementOf*.

The *owl:intersectionOf* property links a class to a list of class descriptions. An *owl:intersectionOf* statement describes a class for which the class extension contains precisely those individuals that are members of the class extension of all class descriptions in the list. Thus we define it as a function from a list of classes to a class.

```
intersectionOf : [list[(cls?)] -> (cls?)]
intersectionOf_ax : AXIOM FORALL (lc: list[(cls?)]), (c: (cls?):
  (c = intersectionOf(lc) IMPLIES FORALL (i: (idv?):
    (member(i, instances(c)) IFF FORALL (sc: (cls?):
      (member(sc, lc) IMPLIES member(i, instances(sc)))))
```

The *owl:unionOf* property links a class to a list of class descriptions. An *owl:unionOf* statement describes an anonymous class for which the class extension contains those individuals that occur in at least one of the class extensions of the class descriptions in the list. Thus we define it as a function from a list of classes to a class.

```

unionOf : [list[(cls?)] -> (cls?)]
unionOf_ax : AXIOM FORALL (lc:list[(cls?)]),(c:(cls?):
    (c = unionOf(lc) IMPLIES FORALL (i:(idv?):
        (member(i,instances(c)) IFF EXISTS (sc:(cls?):
            (member(sc,lc) AND member(i,instances(sc))))))

```

An *owl:complementOf* property links a class to precisely one class description. An *owl:complementOf* statement describes a class for which the class extension contains exactly those individuals that do not belong to the class extension of the class description that is the object of the statement. Thus we define it as a function from a classes to another class.

```

complementOf : [(cls?) -> (cls?)]
complementOf_ax : AXIOM FORALL (c1,c2:(cls?):
    (c2 = complementOf(c1) IMPLIES FORALL (i:(idv?):
        (member(i,instances(c1)) IFF NOT member(i,instances(c2))))

```

3.1.3 Axioms

Class descriptions form the building blocks for defining classes through class axioms. The simplest form of a class axiom is a class description of type 1, It just states the existence of a class, using *owl:Class* with a class identifier.

The *rdfs:subClassOf* construct is defined as part of RDF Schema. Its meaning in

3.1. PVS Semantics for OWL DL

OWL is exactly the same: if the class description $C1$ is defined as a subclass of class description $C2$, then the set of individuals in the class extension of $C1$ should be a subset of the set of individuals in the class extension of $C2$. The property *rdfs:subClassOf* is defined as a boolean function from two classes. Here we make use of the some predicate pre-defined in the PVS prelude.

```
subClassOf?(c1,c2:(cls?): bool = subset?(instances(c1),instances(c2))
```

A class axiom may contain (multiple) *owl:equivalentClass* statements. *owl:equivalentClass* is a built-in property that links a class description to another class description. The meaning of such a class axiom is that the two class descriptions involved have the same class extension (i.e., both class extensions contain exactly the same set of individuals). We similarly model *owl:equivalentClass* as a boolean function as shown below.

```
equivalentClass?(c1,c2:(cls?): bool =  
  FORALL (i:(idv?): (member(i,instances(c1)) IFF member(i,instances(c2)))
```

A class axiom may also contain (multiple) *owl:disjointWith* statements. *owl:disjointWith* is a built-in OWL property with a class description as domain and range. Each *owl:disjointWith* statement asserts that the class extensions of the two class descriptions involved have no individuals in common. We similarly model *owl:disjointWith* as a boolean function as shown below.

```
disjointWith?(c1,c2:(cls?): bool =  
  FORALL (i:(idv?): (member(i,instances(c1)) IFF NOT member(i,instances(c2)))
```

A *rdfs:subPropertyOf* axiom defines that the property is a subproperty of some other property. Formally this means that if $P1$ is a subproperty of $P2$, then the property extension of $P1$ (a set of pairs) should be a subset of the property extension of $P2$ (also a set of pairs). Therefore it is modeled as a boolean function of two properties. We use the overloading mechanism in PVS to capture the different cases of object and datatype properties.

```
subPropertyOf?(p1,p2:(oppt?): bool = subset?(sub_val_0(p1),sub_val_0(p2))
subPropertyOf?(p1,p2:(dppt?): bool = subset?(sub_val_D(p1),sub_val_D(p2))
```

The *owl:equivalentProperty* construct can be used to state that two properties have the same property extension. Syntactically, *owl:equivalentProperty* is a built-in OWL property with *rdfs:Property* as both its domain and range. We similarly model it as boolean functions.

```
equivalentProperty?(p1,p2:(oppt?): bool = FORALL (i1,i2:(idv?):
  (member((i1,i2),sub_val_0(p1)) IFF member((i1,i2),sub_val_0(p2)))
equivalentProperty?(p1,p2:(dppt?): bool = FORALL (i1:(idv?),i2:(dtv?):
  (member((i1,i2),sub_val_P(p1)) IFF member((i1,i2),sub_val_P(p2)))
```

Syntactically, *owl:inverseOf* is a built-in OWL property with *owl:ObjectProperty* as its domain and range. An axiom of the form $P1 \text{ owl:inverseOf } P2$ asserts that for every pair (x,y) in the property extension of $P1$, there is a pair (y,x) in the property extension of $P2$, and vice versa. We similarly model it as a boolean function in PVS.

```
inverseOf?(p1,p2:(oppt?): bool = FORALL (i1,i2:(idv?):
  (member((i1,i2),sub_val_0(p1)) IFF member((i2,i1),sub_val_0(p2)))
```

For a property one can define (multiple) *rdfs:domain* axioms. Syntactically, *rdfs:domain*

3.1. PVS Semantics for OWL DL

is a built-in property that links a property to a class description. An *rdfs:domain* axiom asserts that the subjects of such property statements must belong to the class extension of the indicated class description. Thus we define

```
domain?(p:(oppt?),c:(cls?): bool = FORALL (i1:(idv?),i2:(idv?):  
  (member((i1,i2),sub_val_O(p)) IMPLIES member(i1,instances(c)))  
domain?(p:(dppt?),c:(cls?): bool = FORALL (i1:(idv?),i2:(dtv?):  
  (member((i1,i2),sub_val_D(p)) IMPLIES member(i1,instances(c)))
```

For a property one can define (multiple) *rdfs:range* axioms. Syntactically, *rdfs:range* is a built-in property that links a property to either a class description or a data range. An *rdfs:range* axiom asserts that the values of this property must belong to the class extension of the class description or to the data values in the specified data range. The semantic definitions for *rdfs:range* is a little more complicated for datatype properties. We have to consider different cases for different datatypes of ontology languages.

```

range?(p:(oppt?),c:(cls?): bool = FORALL (i1,i2:(idv?):
  (member((i1,i2),sub_val_0(p)) IMPLIES member(i2,instances(c)))
dnat, dint, dreal, dbool, dstring: (dtt?)
range?(p:(dppt?),dt:(dtt?): bool =
  IF dt = dnat THEN FORALL (i1:(idv?),i2:Datavalue):
    (member((i1,dtv(i2)),sub_val_D(p)) IMPLIES (natDV?)(i2))
  ELSIF dt = dint THEN FORALL (i1:(idv?),i2:Datavalue):
    (member((i1,dtv(i2)),sub_val_D(p)) IMPLIES (intDV?)(i2))
  ELSIF dt = dreal THEN FORALL (i1:(idv?),i2:Datavalue):
    (member((i1,dtv(i2)),sub_val_D(p)) IMPLIES (realDV?)(i2))
  ELSIF dt = dbool THEN FORALL (i1:(idv?),i2:Datavalue):
    (member((i1,dtv(i2)),sub_val_D(p)) IMPLIES (boolDV?)(i2))
  ELSIF dt = dstring THEN FORALL (i1:(idv?),i2:Datavalue):
    (member((i1,dtv(i2)),sub_val_D(p)) IMPLIES (stringDV?)(i2))
  ELSE FALSE ENDIF

```

A functional property is a property that can have only one (unique) value y for each instance x , i.e. there cannot be two distinct values $y1$ and $y2$ such that the pairs $(x,y1)$ and $(x,y2)$ are both instances of this property. Thus the OWL semantic for *owl:FunctionalProperty* is defined as follows.

```

functionalProperty?(p:(oppt?): bool = FORALL (r1,r2,r3:(idv?):
  (member((r1,r2),sub_val_0(p)) AND member((r1,r3),sub_val_0(p)))
  IMPLIES r2 = r3
functionalProperty?(p:(dppt?): bool = FORALL (r1:(idv?), (r2,r3:(dtv?):
  (member((r1,r2),sub_val_D(p)) AND member((r1,r3),sub_val_D(p)))
  IMPLIES r2 = r3

```

If a property is declared to be inverse-functional, then the object of a property statement uniquely determines the subject (some individual). More formally, if we state that P is an *owl:InverseFunctionalProperty*, then this asserts that a value y can only be the value of P for a single instance x , i.e. there cannot be two distinct instances

3.1. PVS Semantics for OWL DL

$x1$ and $x2$ such that both pairs $(x1,y)$ and $(x2,y)$ are instances of P . Thus the OWL semantic for *owl:InverseFunctionalProperty* is defined as follows.

```
inverseFunctionalProperty?(p:(oppt?): bool = FORALL (r1,r2,r3:(idv?):
  (member((r1,r3),sub_val_0(p)) AND member((r2,r3),sub_val_0(p)))
  IMPLIES r1 = r2
inverseFunctionalProperty?(p:(dppt?): bool = FORALL (r1:(idv?), (r2,r3:(dtv?):
  (member((r1,r3),sub_val_D(p)) AND member((r2,r3),sub_val_D(p)))
  IMPLIES r1 = r2
```

When one defines a property P to be a transitive property, this means that if a pair (x,y) is an instance of P , and the pair (y,z) is also instance of P , then we can infer the the pair (x,z) is also an instance of P . The OWL semantic for *owl:TransitiveProperty* is defined as follows.

```
transitiveProperty?(p:(oppt?): bool = FORALL (r1,r2,r3:(idv?):
  (member((r1,r3),sub_val_0(p)) AND member((r2,r3),sub_val_0(p)))
  IMPLIES member((r1,r3),sub_val_0(p))
```

A symmetric property is a property for which holds that if the pair (x,y) is an instance of P , then the pair (y,x) is also an instance of P . The OWL semantic for *owl:SymmetricProperty* is defined as follows.

```
symmetricProperty?(p:(oppt?): bool = FORALL (r1,r2:(idv?):
  (member((r1,r2),sub_val_0(p)) IFF member((r2,r1),sub_val_0(p)))
```

3.1.4 Assertions

The built-in OWL property *owl:sameAs* links an individual to an individual. Such an *owl:sameAs* statement indicates that two URI references actually refer to the

same thing. The built-in OWL *owl:differentFrom* property links an individual to an individual. An *owl:differentFrom* statement indicates that two URI references refer to different individuals. We model the semantics as follows.

```
sameAs?(r1,r2:(idv?): bool = (r1 = r2)
differentFrom?(r1,r2:(idv?): bool = NOT (r1 = r2)
```

3.1.5 SWRL Rules

In SWRL [56] a rule consists of an antecedent and a consequent, each of which consists of a (possibly empty) set of atoms. Atoms can be of the form $C(x)$, $P(x, y)$, $sameAs(x, y)$, $differentFrom(x, y)$ or a built-in atom where C is an OWL class description, P is an OWL property, and x , y are either variables, OWL individuals or OWL data values. Informally, an atom $C(x)$ holds if x is an instance of the class description C , an atom $P(x, y)$ holds if x is related to y by property P , an atom $sameAs(x, y)$ holds if x is interpreted as the same object as y , and an atom $differentFrom(x, y)$ holds if x and y are interpreted as different objects. A rule may be read as meaning that if the antecedent holds (is “true”), then the consequent must also hold.

A SWRL rule will be modeled as a PVS rewrite rule, e.g., a universally quantified predicate of the form

$$a_1 \wedge a_2 \wedge \dots \wedge a_m \Rightarrow c_1 \wedge c_2 \wedge \dots \wedge c_n$$

3.1. PVS Semantics for OWL DL

where a_i and c_j are one of the four forms of atoms.

3.1.6 Proof Support for PVS

To make the proving process of PVS more automated, a set of rewrite rules and theorems is also defined. They aim to hide certain amount of underlying model from the verification and reasoning and to achieve abstraction and automation. Usually these rules relate several classes and properties by defining the effect of using them in a particular way. Before using the theorems, we first need to prove that they are valid based on our semantic encoding.

One simple example is the `subClassOf_transitive` theorem. It states that if a class `c1` is a sub-class of a class `c2` and `c2` is a sub-class of a class `c3`, then `c1` is a sub-class of `c3`.

<pre>subClassOf_transitive: THEOREM FORALL (c1,c2,c3:(cls?)): subClassOf?(c1,c2) AND subClassOf?(c2,c3) IMPLIES subClassOf?(c1,c3)</pre>
--

This theorem can be easily proved by asserting the definition of `subClassOf?` and issuing the (`grind`) command.

The following theorem, *member_subClassOf* states that an instance of a particular class is also an instance of all the super classes of this class.

<pre>member_subClassOf: THEOREM FORALL (i:(idv?)),(c1,c2:(cls?)): member(i, instances(c1)) AND subClassOf?(c1,c2) IMPLIES member(i, instances(c2))</pre>
--

The development of the reasoning rules is an incremental process. We start with 20 reasoning rules which we feel are useful in performing the initial Semantic Web reasoning tasks. In the process of reasoning, we constantly include, prove and use more reasoning rules in the rule set. This is useful because the proof of such rules can help verify the correctness of our semantic encoding. Another reason for developing the reasoning rules is that most of the reasoning rules are re-used more than once. Currently there are altogether more than 200 rules stored in the PVS theory.

3.2 Reasoning about Ontologies in PVS

In this section, we demonstrate how PVS can be used to check ontology-related properties and to reason beyond the modeling power of OWL & ORL. It is presented in two parts. Firstly, standard SW reasoning are performed. In the second part, we show how PVS can reason ORL and more complex properties that even ORL cannot express.

3.2.1 Standard SW Reasoning

Standard SW reasoning includes three categories, namely inconsistency checking, subsumption reasoning and instantiation reasoning. The following subsections illustrate each category with an example.

3.2. Reasoning about Ontologies in PVS

Inconsistency Checking

Ensuring the consistency of ontologies is an important task in various stages of ontology development, as inconsistent ontologies may lead agents to reason erroneously and make wrong conclusions.

To be precise, knowledge base consistency amounts to verifying whether every concept in the knowledge base admits at least one individual [84].

The following is an example of inconsistency checking in the animal ontology. After transforming the ontology into a PVS specification, we identified the following closely related classes, properties and their axioms.

```
Animal, Vegetarian, Cow, MadCow, Food, Meat, Vegetable: (cls?)
eats: (oppt?)
Vegetarian_subClassOf_ax_1: AXIOM subClassOf?(Vegetarian, Animal)
Vegetarian_allValuesFrom_ax_1: AXIOM Vegetarian = allValuesFrom(eats, Vegetable)
Cow_subClassOf_ax_1: AXIOM subClassOf?(Cow, Vegetarian)
MadCow_subClassOf_ax_1: AXIOM subClassOf?(MadCow, Cow)
MadCow_subClassOf_ax_2: AXIOM subClassOf?(MadCow, someValuesFrom(eats, Meat))
Meat_subClassOf_ax_1: AXIOM subClassOf?(Meat, Food)
Vegetable_subClassOf_ax_1: AXIOM subClassOf?(Vegetable, Food)
Vegetable_disjointWith_ax_1: AXIOM disjointWith?(Vegetable, Meat)
```

We suspect that there is an inconsistency in the class of *MadCow*. To prove that, we assert the following theorem, which means that the class of *MadCow* does not admit any individual.

```
MadCow_inconsistent: THEOREM
  (EXISTS (i:INDIVIDUAL):member(i, instances(MadCow))) IMPLIES FALSE
```

After applying `(lemma)` to supply PVS with known facts (axioms), applying `(skolem!)` to remove quantifiers and instructing PVS to understand the subclass relationship between *MadCow* and *Vegetarian*, we need to prove

`member(i!1,instances(Vegetarian))`, that *i!1* is a member of *Vegetarian*, which can be proved by the theorem *member_subClassOf* introduced in Section 3.1.6.

By expanding the definition of *Vegetarian* and exploiting the fact that *MadCow* is a subclass of an anonymous class that *eats Meat*, we can finish up the proof using a `(grind)`, which is a catch-all strategy that is frequently used to automatically complete a proof branch or to apply all the obvious simplifications.

Subsumption Reasoning

The task of subsumption reasoning is to infer that an OWL class is a sub-class of another. This could be accomplished in PVS fairly automatically. One of the simplest ways is by using the fact that `subClassOf?` is a transitive property, which can be easily proved by PVS.

There are other ways of proving subsumption relationships. One of them is by inter-class relationships such as `intersectionOf` and `UnionOf`. For example, we have the following transformed ontology fragment and we want to prove that the class *TallMan* is a subclass of *Person* using theorem *TallMan_subClassOf_Person*.

3.2. Reasoning about Ontologies in PVS

```
TallMan_intersection_ax: AXIOM
  TallMan=intersectionOf((:TallThing,Man:))
Person_union_ax: AXIOM
  Person=unionOf((:Man,Woman:))
TallMan_subClassOf_Person: THEOREM
  subClassOf?(TallMan,Person)
```

The main steps of this proof are to prove separately `subClassOf?(TallMan,Man)` and `subClassOf?(Man,Person)`. Then the simple subsumption reasoning can finish proving the theorem. The above two goals can be proved by the application of two user defined theorems relating *intersectionOf* and *unionOf* to *subClassOf*, respectively.

Instantiation Reasoning

Instantiation reasoning asserts that one resource is or is not an instance of a class. Some SW reasoning tools such as FaCT are designed to only support concept-level reasoning. Hence reasoning at the instance-level cannot be performed by these tools. We demonstrate through an example that PVS supports instance-level reasoning.

In the example ontology, we defined an individual called *Santa*, who can move by both walking and flying, by the following axioms.

```
Santa_moves_walk_ax: AXIOM moves(Santa,walk)
Santa_moves_fly_ax: AXIOM moves(Santa,fly)
```

We want to prove that *Santa* is not an instance of the class *Person*. By stating the facts that all instances of the *Person* class can move only by walk, that the individual

Santa can fly, and that *walk* and *fly* are disjoint, we can finish the proof with a (`grind`) command.

3.2.2 Checking SWRL & Beyond

The above examples demonstrate PVS’s power of performing consistency, subsumption and instantiation reasoning about OWL ontologies with certain degree of automation. Now we shall illustrate that PVS can reason about ORL and more complex properties that even ORL cannot capture.

SWRL Reasoning

We illustrate our idea with an example ontology about scheduling agents for different tasks, which is represented in n3 [15] syntax below in Table 3.1. Informally, there is a set of tasks and a set of agents. Any task can be assigned to any agent. There is also a set of discrete time points and a set of data. A time point may precede another. Each task starts and ends at particular time points and may possibly use a piece of data. A task could relate to another task. Some tasks may overlap with some other task(s).

We first check the consistency the the ontology by using the methodology described in the previous section. We omit the details and the result shows the ontology is

3.2. Reasoning about Ontologies in PVS

Table 3.1: The Model of Scheduling Tasks

:Agent a owl:Class. :Task a owl:Class. :TimePoint a owl:Class. :Data a owl:Class. :relatesTo a owl:TransitiveProperty; rdfs:domain Task; rdfs:range Task; :assignedTo a owl:ObjectProperty; rdfs:domain Task; rdfs:range Agent. :starts a owl:ObjectProperty; rdfs:domain Task; rdfs:range TimePoint. :ends a owl:ObjectProperty; rdfs:domain Task; rdfs:range TimePoint. :precedes a owl:TransitiveProperty; rdfs:domain TimePoint; rdfs:range TimePoint. :overlaps a owl:ObjectProperty; rdfs:domain Task; rdfs:range Task. :uses a owl:ObjectProperty; rdfs:domain Task; rdfs:range Data.	:a1 a Agent. :a2 a Agent. :t1 a Task; :starts :tp1; :ends :tp3; :assignedTo :a1. :t2 a Task; :starts :tp2; :ends :tp4; :uses :d2; :assignedTo :a2. :t3 a Task; :starts :tp4; :ends :tp5; :relatesTo :t1; :uses :d1; :assignedTo :a2.	:tp1 a TimePoint; :precedes :tp2. :tp2 a TimePoint; :precedes :tp3. :tp3 a TimePoint; :precedes :tp4. :tp4 a TimePoint; :precedes :tp5. :tp5 a TimePoint. :d1 a Data. :d2 a Data.
--	--	---

consistent.

On top of the OWL and RDF ontology, there are four SWRL rules. The first one states that an agent cannot be assigned to two overlapping tasks.

$$\begin{aligned}
& Task(?t1) \wedge Task(?t2) \wedge Agent(?a1) \wedge Agent(?a2) \wedge \\
& assignedTo(?t1, ?a1) \wedge assignedTo(?t2, ?a2) \wedge overlaps(?t1, ?t2) \\
& \rightarrow \\
& differentFrom(?a1, ?a2)
\end{aligned}$$

The transformed PVS theorem is given below.

```

rule_1: AXIOM FORALL(t1,t2,a1,a2 : (idv?)):
  member(t1,instances(Task)) AND
  member(t2,instances(Task)) AND
  member(a1,instances(Agent)) AND
  member(a2,instances(Agent)) AND
  member((t1,a1),sub_val_0(assignedTo)) AND
  member((t2,a2),sub_val_0(assignedTo)) AND
  member((t1,t2),sub_val_0(overlaps))
  IMPLIES
  differentFrom?(a1,a2)

```

The second rule requires that related tasks must be assigned to the same agent.

$$\begin{aligned}
 & Task(?t1) \wedge Task(?t2) \wedge Agent(?a1) \wedge Agent(?a2) \wedge \\
 & assignedTo(?t1, ?a1) \wedge assignedTo(?t2, ?a2) \wedge relatesTo(?t1, ?t2) \\
 & \rightarrow \\
 & sameAs?(?a1, ?a2)
 \end{aligned}$$

The transformed PVS theorem is given below.

```

rule_2: AXIOM FORALL(t1,t2,a1,a2 : (idv?)):
  member(t1,instances(Task)) AND
  member(t2,instances(Task)) AND
  member(a1,instances(Agent)) AND
  member(a2,instances(Agent)) AND
  member((t1,a1),sub_val_0(assignedTo)) AND
  member((t2,a2),sub_val_0(assignedTo)) AND
  member((t1,t2),sub_val_0(relatedTo))
  IMPLIES
  sameAs?(a1,a2)

```

The third rule requires that any two overlapping tasks cannot use the same piece of data.

3.2. Reasoning about Ontologies in PVS

$$\begin{aligned} & Task(?t1) \wedge Task(?t2) \wedge Data(?d1) \wedge Data(?d2) \wedge \\ & uses(?t1, ?d1) \wedge uses(?t2, ?d2) \wedge overlaps(?t1, ?t2) \\ & \rightarrow \\ & differentFrom?(?d1, ?d2) \end{aligned}$$

The transformed PVS theorem is given below.

```
rule_3: AXIOM FORALL(t1,t2,d1,d2 : (idv?)):  
  member(t1,instances(Task)) AND  
  member(t2,instances(Task)) AND  
  member(d1,instances(Data)) AND  
  member(d2,instances(Data)) AND  
  member((t1,d1),sub_val_0(uses)) AND  
  member((t2,d2),sub_val_0(uses)) AND  
  member((t1,t2),sub_val_0(overlaps))  
  IMPLIES  
  differentFrom?(d1,d2)
```

The last rule defines when two tasks are overlapping - when one task that starts earlier ends after the other task starts.

$$\begin{aligned} & Task(?t1) \wedge Task(?t2) \wedge \\ & TimePoint(?tp1) \wedge TimePoint(?tp2) \wedge TimePoint(?tp3) \wedge TimePoint(?tp4) \wedge \\ & starts(?t1, ?tp1) \wedge ends(?t1, ?tp2) \wedge starts(?t2, ?tp3) \wedge ends(?t2, ?tp4) \wedge \\ & precedes(?tp1, ?tp3) \wedge precedes(?tp3, ?tp2) \\ & \rightarrow \\ & overlaps(?t1, ?t2) \end{aligned}$$

The transformed PVS theorem is given below.

```
rule_4: AXIOM FORALL(t1,t2,tp1,tp2,tp3,tp4 : (idv?)):  
  member(t1,instances(Task)) AND  
  member(t2,instances(Task)) AND  
  member(tp1,instances(TimePoint)) AND  
  member(tp2,instances(TimePoint)) AND  
  member(tp3,instances(TimePoint)) AND  
  member(tp4,instances(TimePoint)) AND  
  member((t1,tp1),sub_val_0(starts)) AND  
  member((t1,tp2),sub_val_0(ends)) AND  
  member((t2,tp3),sub_val_0(starts)) AND  
  member((t2,tp4),sub_val_0(starts)) AND  
  member((tp1,tp3),sub_val_0(precedes)) AND  
  member((tp3,tp2),sub_val_0(precedes))  
  IMPLIES  
  overlaps?(t1,t2)
```

To prove that the SWRL rule is not consistent with the ontology, we simply prove the following PVS theorem: `violateConstraint: theorem FALSE`.

A proof strategy is intended to capture patterns of inference steps. A defined proof rule is a strategy that is applied in a single atomic step so that only the final effect of the strategy is visible and the intermediate steps are hidden from the user. We define a number of proof strategies, such as `(installTimePoint)`, `(installData)`, `(installAgent)`, etc., each of which introduces all the axioms one by one of a particular class. The following strategy introduces to PVS all facts related to all the time points.

3.2. Reasoning about Ontologies in PVS

```
(defstep installTimePoint ()
  (then
    (lemma "tp1_instanceOf_ax")
    (lemma "tp1_precedes_ax")
    (lemma "tp2_instanceOf_ax")
    (lemma "tp2_precedes_ax")
    (lemma "tp3_instanceOf_ax")
    (lemma "tp3_precedes_ax")
    (lemma "tp4_instanceOf_ax")
    (lemma "tp4_precedes_ax")
    (lemma "tp5_instanceOf_ax")
  )
  "Installing all axioms of TimePoint"
  "Installing all axioms of TimePoint"
)
```

Then we also define a strategy which finds and installs the transitive closure of the property `precedes`, i.e., the relative temporal order of all pairs of time points, as follows. This is needed for determining instances of the `overlaps` property later.

```
(defstep installAllPrecedes ()
  (then
    (lemma "precedes_transitive_ax")
    (rewrite "transitiveProperty?")
    (try (forward-chain -1) (installAllPrecedes) (delete -1))
  )
  "Finding and installing all precedes property instances"
  "Finding and installing all precedes property instances"
)
```

Basically this strategy repeatedly forward-chains the `precedes_transitive_ax` axiom until there is no more effect. Similarly, we find all instances of the property `relatesTo` by using the strategy `installAllRelatesTo` (not shown here).

Now we apply the rules. First, we apply the fourth rule to discover all instances of the property `overlaps` by using the strategy `installAllOverlaps` below.

```
(defstep installAllOverlaps ()
  (then
    (lemma "rule_4")
    (try (forward-chain -1) (installAllOverlaps) (delete -1))
  )
  "Finding and installing all overlaps property instances"
  "Finding and installing all overlaps property instances"
)
```

Then we can apply the other three rules one by one by using strategies similarly.

We apply the `(grind)` command, which proves the theorem. It means that the rules are inconsistent with the ontology and the instances. A closer look at the ontology discovers that tasks *t1* and *t2* are related and yet overlapping.

Reasoning Beyond SWRL

Although SWRL greatly increases the expressiveness of the family of ontology languages, there are still many complex, domain-specific properties which cannot be expressed even in SWRL. In this section, we illustrate how PVS can be used to express and prove such properties.

One of the restrictions of SWRL is that it allows only universal quantification. This prevent many desirable properties from being specified. For example, in our example ontology we want to specify a “liveness” property that *all agents that are not assigned*

3.2. Reasoning about Ontologies in PVS

a task which starts before a given time point will be assigned at least one task which starts after that time point. This property involves nested universal and existential quantification and clearly cannot be expressed even in SWRL. However it can be easily specified using our framework in PVS as follows.

```
liveness_ax: theorem
FORALL (a:(idv?):
  member(a,instances(Agent)) IMPLIES
    NOT EXISTS (t1,tp1:(idv?):
      (member(t1,instances(Task)) AND
       member(tp1,instances(TimePoint)) AND
       member((t1,a),sub_val_0(assignedTo)) AND
       member((t1,tp1),sub_val_0(starts)) AND
       member((tp1,tp),sub_val_0(precedes))) IMPLIES
        EXISTS (t2,tp2:(idv?):
          (member(t2,instances(Task)) AND
           member(tp2,instances(TimePoint)) AND
           member((t2,a),sub_val_0(assignedTo)) AND
           member((t2,tp2),sub_val_0(starts)) AND
           member((tp,tp2),sub_val_0(precedes))))
```

To prove that this theorem is not satisfiable, we prove the negated theorem. Using a similar approach with proof strategies as described in the previous section (asserting the known axioms, finding property closures and applying rules), we can prove the negated theorem with different values for the given time point `tp` to a large degree of automation in the PVS theorem prover.

Generally speaking, the formulation of the PVS theorems is done through the interactions between domain experts, ontology developers and PVS users. The domain

experts state desired properties and requirements while ontology developers and the PVS users decide which of the properties can be part of the ontology and which are too complex and can only be stated as PVS theorems.

3.3 Chapter Summary

The contribution of this chapter is three-fold.

1. We have defined the PVS semantics for the ontology languages OWL DL and SWRL, which is the foundation for the later work on checking Web ontologies using the PVS theorem prover.
2. We have also developed a substantial set of reasoning rules on top of the semantics encoding. The set of reasoning rules can not only verify the correctness of our encoding, but also help improve the degree of automation for the later reasoning process.
3. We have demonstrated how the PVS theorem prover can be used to perform not only standard Semantic Web reasoning tasks such as inconsistency checking, subsumption reasoning and instantiation reasoning, but also, more importantly, advanced SWRL rule checking and even checking complex properties which cannot even be expressed in SWRL.

3.3. Chapter Summary

Our work on the extended support for checking SWRL-enabled OWL ontologies is, to our knowledge, the first reasoning support for SWRL in the literature.

Chapter 4

Checking Agent Knowledge With Uncertainty

In the previous chapter, we have seen how PVS theorem prover can be effective for the purpose of advanced knowledge reasoning for intelligent agents. As agents have a necessarily partial perspective of their world, and because their problem domain is open, complex and distributed, they require sophisticated mechanisms for reasoning with uncertain, incomplete and contradictory information if they are to exhibit the desired degree of flexibility and robustness.

The cornerstone language in the Semantic Web is Web Ontology Language (OWL) [11], which provides core language constructs to semantically mark up resources. OWL

is based on description logic [7], a subset of first-order predicate logic. This dictates that any formula can be inferred from an inconsistent knowledge base (aggregate information). This is neither practical nor desirable as no useful reasoning services are available even in the presence of very slight inconsistency. Hence, a mechanism of representing confidence and ignorance is very desirable.

The Belief-augmented Frames (BAF) [105] is an extension to the Minsky frame knowledge representation system [81]. Its unique feature is that it adds a belief and a disbelief value to each frame in a frame system. In BAF, belief and disbelief values are independent from each other, allowing for greater flexibility in modeling arguments for and against a fact. Consequently, ignorance and confidence can be incorporated based on these values.

In this chapter, we propose to integrate BAF with OWL DL to form a new ontology language BOWL (Belief-augmented OWL) that can easily express beliefs. We start by discussing the OWL abstract syntax and model-theoretic semantics in Section 4.1 and Section 4.2. We give an overview of BAF in Section 4.3. In Section 4.4 we present how OWL can be extended with BAF in a syntactically and semantically coherent manner. Section 4.5 discusses how we can reason about BOWL ontologies and how confidence factors can be computed. In Section 4.6, we present an example in the sensor fusion domain to demonstrate the reasoning process in BOWL. Section 4.7 summarizes the main contributions of this chapter and compares some related work.

4.1 OWL Abstract Syntax

In OWL, abstract concepts are *classes*, related by binary relations called *properties* and are populated by concrete *individuals*. In a compact form, OWL can be presented in the Description Logics (DL) syntax [57].

Classes in OWL are first-class citizens. Existing classes can be used in *class expressions* to define new complex ones using class constructors as shown in Fig. 4.1 where \mathcal{C} represents (possible complex) class expressions; C is a class name; P stands for a property; n is a natural number and a_i 's are individuals. In OWL, datatype properties and object properties are distinguished. Without loss of generality and for brevity reasons, we omit the discussion related to datatypes. They can be treated in a similar manner.

$\mathcal{C} ::= C$	[Class name]
\top	[Top class]
\perp	[Bottom class]
$\mathcal{C} \sqcup \mathcal{C}$	[Class union]
$\mathcal{C} \sqcap \mathcal{C}$	[Class intersection]
$\neg \mathcal{C}$	[Class negation]
$\forall P.\mathcal{C}$	[Universal quantification]
$\exists P.\mathcal{C}$	[Existential quantification]
$P: o$	[Value restriction]
$\geq n P$	[At least number restriction]
$\leq n P$	[At most number restriction]
$\{a_1, \dots, a_n\}$	[Enumeration]

Figure 4.1: OWL class expressions

OWL also defines *class axioms* that inter-relate classes. These class axioms, shown in Fig. 4.2 below, include class subsumption, equivalence, disjointness, etc. Axioms are also defined for describing properties. These include property subsumption, equivalence, domain, range, etc.

$\mathcal{AX} ::= C \sqsubseteq C$	[Class subsumption]
$C = C$	[Class equivalence]
$C \sqcap C = \perp$	[Class disjointness]
$P \sqsubseteq P$	[Property subsumption]
$P = P$	[Property equivalence]
$\geq 1 P \sqsubseteq C$	[Property domain]
$\top \sqsubseteq \forall P.C$	[Property range]
$\top \sqsubseteq \leq 1 P$	[Functional property]
$P = (\neg P)$	[Inverse property]

Figure 4.2: OWL class axioms

Assertions in OWL are used to model individuals. An assertion can model the fact that an individual is an instance of a class, or an individual is related to another individual by a property. In addition, assertions can be used to state the (in)equality of two individuals. Fig. 4.3 shows the assertions available in OWL. Note that the symbol \mathcal{AS} represents assertions.

4.2. OWL Semantics

$\mathcal{AS} ::= a \in \mathcal{C}$	[Class membership]
$\langle a, b \rangle \in P$	[Property membership]
$a = b$	[Individual equality]
$a \neq b$	[Individual inequality]

Figure 4.3: OWL assertions

4.2 OWL Semantics

OWL has direct model-theoretic semantics [88] which starts with the notion of a vocabulary.

Definition 4.2.1 (OWL Vocabulary) *An OWL vocabulary V consists of a set of literals V_L and the following seven sets of URI references.*

- V_C : the set of the class names containing owl:Thing and owl:Nothing
- V_D : the set of the datatype names
- V_{AP} : the set of the annotation property names
- V_{IP} : the set of the individual-valued property names
- V_{DP} : the set of the data-valued property names
- V_I : the set of the individual names
- V_O : the set of the ontology names

In any vocabulary, V_C and V_D are disjoint and V_{DP} , V_{IP} , V_{AP} , and V_{OP} are pairwise disjoint.

Definition 4.2.2 (OWL Datatype) *As in RDF, a datatype d is characterized by a lexical space, $L(d)$, which is a set of Unicode strings; a value space, $V(d)$; and a total mapping $L2V(d)$ from the lexical space to the value space.*

Definition 4.2.3 (Datatype Map) *A datatype map D is a partial mapping from URI references to datatypes that maps `xsd:string` and `xsd:integer` to the appropriate XML Schema datatypes.*

Definition 4.2.4 (Abstract OWL Interpretation) *An abstract OWL interpretation with respect to a datatype map D with vocabulary V_L , V_C , V_D , V_I , V_{DP} , V_{IP} , V_{AP} , and V_O is a tuple of the form: $I = \langle R, EC, ER, L, S, LV \rangle$ where (with \mathcal{P} being the power set operator)*

- R : the resources of I , is a non-empty set
- LV : the literal values of I which is a subset of R
- EC : a mapping such that
 - $EC : V_C \rightarrow \mathcal{P}(O)$ where O is a non-empty subset of R and disjoint from LV

4.2. OWL Semantics

- $EC : V_D \rightarrow \mathcal{P}(LV)$
- $EC(\text{owl:Thing}) = O$
- $EC(\text{owl:Nothing}) = \{\}$
- $EC(\text{rdfs:Literal}) = LV$
- ER : a mapping such that
 - $ER : V_{DP} \rightarrow \mathcal{P}(O \times LV)$
 - $ER : V_{IP} \rightarrow \mathcal{P}(O \times O)$
 - $ER : V_{AP} \cup \text{rdf:type} \rightarrow \mathcal{P}(R \times R)$
- $L : TL \rightarrow LV$ where TL is the set of typed literals in V_L
- S : a mapping such that
 - $S : V_C \cup V_D \cup V_{DP} \cup V_{IP} \cup V_{AP} \cup V_O \cup \{\text{owl:Ontology}, \text{owl:DeprecatedClass}, \text{owl:DeprecatedProperty}\} \rightarrow R$
 - $S : V_I \rightarrow O$

In other words, if we denote the *domain of interpretation* as $\Delta^{\mathcal{I}}$, then the interpretation function $\cdot^{\mathcal{I}}$ maps an individual name into a member of the domain $\Delta^{\mathcal{I}}$; a class name into a set of elements in the domain and a property name (note that we are talking about object properties) into a set of pairs of domain elements ($\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$).

The class expressions, axioms, assertions and their interpretation can be summarized in Table 4.1, Table 4.2 and Table 4.3 below.

OWL class expression	Interpretation
C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
\top	$\top^{\mathcal{I}} = \Delta$
\perp	$\perp^{\mathcal{I}} = \emptyset$
$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$\forall P.C$	$\{x \mid \forall y. \langle x, y \rangle \in P^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
$\exists P.C$	$\{x \mid \exists y. \langle x, y \rangle \in P^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
$P: o$	$\{x \mid \langle x, o^{\mathcal{I}} \rangle \in P^{\mathcal{I}}\}$
$\geq n P$	$\{x \mid \#\{y \mid \langle x, y \rangle \in P^{\mathcal{I}}\} \geq n\}$
$\leq n P$	$\{x \mid \#\{y \mid \langle x, y \rangle \in P^{\mathcal{I}}\} \leq n\}$
$\{a_1, \dots, a_n\}$	$\{a_1^{\mathcal{I}}, \dots, a_n^{\mathcal{I}}\}$

Table 4.1: OWL class expressions & their interpretations

Definition 4.2.5 (Ontology Satisfaction) *An abstract OWL interpretation I with respect to a datatype map D with vocabulary consisting of V_L , V_C , V_D , V_I , V_{DP} , V_{IP} , V_{AP} , and V_O satisfies an OWL ontology O if and only if*

4.2. OWL Semantics

Table 4.2: OWL axioms & their interpretations

OWL axiom	Interpretation
$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
$C_1 = C_2$	$C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$
$C_1 \sqcap C_2 = \perp$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \emptyset$

Table 4.3: OWL assertions & their interpretations

OWL assertion	Interpretation
$a \in \mathcal{C}$	$a^{\mathcal{I}} \in \mathcal{C}^{\mathcal{I}}$
$\langle a, b \rangle \in P$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in P^{\mathcal{I}}$
$a = b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
$a \neq b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$

- each URI reference in O used as a class ID (datatype ID, individual ID, data-valued property ID, individual-valued property ID, annotation property ID, ontology ID) belongs to V_C (V_D , V_I , V_{DP} , V_{IP} , V_{AP} , V_O , respectively);
- each literal in O belongs to V_L ;
- I satisfies each axiom and each fact in O , except for Ontology Annotations; and
- I satisfies each ontology mentioned in an owl:imports annotation directive of O .

Definition 4.2.6 (Ontology Consistency) *A collection of abstract OWL ontologies and axioms and facts is consistent with respect to datatype map D if and only if there is some interpretation I with respect to D such that I satisfies each ontology and axiom and fact in the collection.*

4.3 Belief-augmented Frames

In belief models the possibility of an event occurring is modeled as a range of values rather than as a single point probability. This range allows us to express ignorance, which standard statistical measures do not accommodate. Statistical measures only provide limited ability to express ignorance. For example, a service agent, who predicted with a certainty of $x\%$ that a particular service is available at a certain point of time, may be reluctant to predict with a certainty of $(100 - x)\%$ that the service is not available. This apparent contradiction is a reflection of classical statistics' inability to cater to ignorance. Various models of beliefs have been proposed, including the seminal Dempster-Shafer theory [24, 97]. Smets generalized the model in [101] to form the Transferable Belief Model. Picard [91] proposed the Probabilistic Argumentation System, which combines propositional logic with probability measures to perform reasoning.

4.3. Belief-augmented Frames

4.3.1 Belief Augmented Systems

In classical AI a frame represents an object in the world, and slots within the frame indicate the possible relations that this object can have with other objects. A value (or set of values) in a slot indicates the other objects that are related to this object through the relation represented by the slot. The existence of a slot-value pair indicates a relation.

- In BAFs each slot-value pair is augmented by a pair of belief/disbelief masses. We define this pair as **BAF**, with **BAF_range** as the value range from 0 to 1, *i.e.*, $[0, 1]$. Hence, a BAF b is a pair $\langle a_t, a_f \rangle$, where a_t and a_f are from **BAF_range**.
- Furthermore, we define two functions ϕ^T and ϕ^F to project out the belief/disbelief values of a BAF value pair¹.

$$\begin{aligned} \phi^T, \phi^F : \text{BAF} &\rightarrow \text{BAF_range} \\ \forall x : \text{BAF} \bullet \phi^T(x) = x.1 \wedge \phi^F(x) = x.2 \end{aligned}$$

Note that ϕ^T and ϕ^F of a particular BAF may not necessarily sum to 1. This frees us from the classical statistical assumption that $\phi^T(\text{rel}) = 1 - \phi^F(\text{rel})$ and it allows us to model ignorance. It is also possible that $\phi^T(\text{rel}) + \phi^F(\text{rel}) > 1$. More information on this can be found in [105] where ignorance is discussed in detail. Both ϕ^T and ϕ^F may be derived from various independent sources, or

¹The symbol \bullet can be interpreted as ‘such that’.

may be computed by using a system of logic called "BAF-Logic" which will be presented in the next subsection. Effectively this allows us to model the belief in a problem as a set of arguments for the belief, and a set of arguments against it.

- While ϕ^T and ϕ^F represent the degree of belief for and against a claim, the overall truth is given by the Degree of Inclination DI , a function from BAF , pairs of BAF values, to DI_range , an interval from -1 to 1 $([-1, 1])$.

$$DI : BAF \rightarrow DI_range$$

$$\forall x : BAF \bullet DI(x) = \phi^T(x) - \phi^F(x)$$

$DI(rel)$ measures the overall degree of truth of the relationship rel , with -1 representing falsehood, 1 representing truth, and values in between representing various degrees of truth and falsehood. As an example, we could take -0.25 to mean "possibly false", -0.5 to mean "probably false", etc.

- The Utility Function U is defined as follows:

$$U : BAF \rightarrow BAF_range$$

$$\forall x : BAF \bullet U(x) = \frac{1 + DI(x)}{2}$$

U maps the Degree of Inclination to a $[0, 1]$ range. If we normalize the Utility Functions for all relations so that they sum to 1, we can use these normalized values as statistical measures representing the probability of a relation being true.

4.3. Belief-augmented Frames

It can be easily verified that for any BAF value b , $U(b) = 1 - U(\neg^B b)$ and if $U(b) \geq n$, then $U(\neg^B b) \leq 1 - n$, for $n \in [0, 1]$. Note that the BAF *not* operator \neg^B will be introduced in the next subsection.

4.3.2 Predefined Beliefs

We define three predefined BAFs $\phi_{\text{one}} = \langle 1, 0 \rangle$ and $\phi_{\text{zero}} = \langle 0, 1 \rangle$. They are convenient shorthand for frequently used BAF values.

4.3.3 Belief Augmented Frames Logic

Belief Augmented Frame Logic (BAF-Logic) is a system designed to reason about the ϕ^T and ϕ^F values in the frame.

- We define the BAF *conjunction* \cap^B as a function from two BAFs to a BAF. Hence, given two BAFs P and Q , their conjunction $P \cap^B Q$ is defined as follows.

$$\begin{aligned} & _ \cap^B _ : \text{BAF} \times \text{BAF} \rightarrow \text{BAF} \\ & \forall P, Q : \text{BAF} \bullet \\ & \quad \phi^T(P \cap^B Q) = \min(\phi^T(P), \phi^T(Q)) \wedge \\ & \quad \phi^F(P \cap^B Q) = \max(\phi^F(P), \phi^F(Q)) \end{aligned}$$

This definition is based on the intuitive idea that the strength of $P \cap^B Q$ being true rests on the strength of the weakest proposition P or Q . Likewise, if either P or Q were false, then $P \cap^B Q$ would be false, and we can base our degree of

belief in $P \cap^B Q$ being false on the strongest proposition that either P or Q is false.

- Similarly we define the BAF *disjunction* \cup^B between P and Q as:

$$\begin{aligned} & _ \cup^B _ : \text{BAF} \times \text{BAF} \rightarrow \text{BAF} \\ & \forall P, Q : \text{BAF} \bullet \\ & \quad \phi^T(P \cup^B Q) = \max(\phi^T(P), \phi^T(Q)) \wedge \\ & \quad \phi^F(P \cup^B Q) = \min(\phi^F(P), \phi^F(Q)) \end{aligned}$$

- Finally, we define the BAF *not* operator \neg^B as:

$$\begin{aligned} & \neg^B _ : \text{BAF} \rightarrow \text{BAF} \\ & \forall P : \text{BAF} \bullet \\ & \quad \phi^T(\neg^B P) = \phi^F(P) \wedge \phi^F(\neg^B P) = \phi^T(P) \end{aligned}$$

This means that the degree that we believe that our data support $\neg^B P$ is equal to the degree that they refute P . Likewise the degree that our data refute $\neg^B P$ is equal to the degree that they support P .

4.4 Belief-augmented OWL (BOWL)

The Belief-augmented OWL incorporates belief/disbelief values defined in BAF into OWL to enable the representation & reasoning of incomplete, subjective and sometimes conflicting resources on the Semantic Web. In this section, we introduce BOWL and present its semantics by defining interpretations of the various language constructs of BOWL.

4.4. Belief-augmented OWL (BOWL)

4.4.1 BAF

Each fact in BOWL is augmented with a BAF, a pair consisting of a belief and a disbelief measure of the type `BAF_range`. Hence, a BAF value is of the form $\langle b_t, b_f \rangle$, where b_t and b_f are the belief/disbelief values, respectively. So we add the following to the OWL abstract syntax.

$$\text{BAF} ::= \langle \text{BAF_range}, \text{BAF_range} \rangle$$

`BAF_range` can be viewed as a data type derived from `float` defined in XML Schema [119].

In the following, we will use angle brackets “ $\langle \rangle$ ” to denote the association of an OWL language construct and its BAF value.

4.4.2 BOWL Syntax Extension

As in OWL, BOWL axioms are about classes and properties and BOWL assertions are facts about ground knowledge entities such as individuals and data values. BOWL augments OWL assertions with BAF values, as follows. Both class and property membership assertions are treated alike. For any OWL assertion, its BOWL extension is summarized in Figure. 4.4.

The second kind of facts asserts the relationship between individuals. BOWL attaches ϕ_{one} to each of these assertions hence the (in)equality between individuals will be treated in the same way as in OWL.

$\mathcal{AS} ::= \langle a \in \mathcal{C}, \text{BAF} \rangle$	[Class membership]
$\langle \langle a, b \rangle \in P, \text{BAF} \rangle$	[Property membership]
$\langle a = b, \phi_{\text{one}} \rangle$	[Individual equality]
$\langle a \neq b, \phi_{\text{one}} \rangle$	[Individual inequality]

Figure 4.4: BOWL assertions

4.4.3 BOWL Semantic Extension

We construct the semantics of BOWL by extending the model-theoretic semantics of OWL [88]. Firstly, we assume that the **datatype map** \mathcal{D} (as in OWL) is extended to include a mapping from the (abbreviated) URI `owl:BAF` to the datatype `BAF`, as defined in Section 4.3. For brevity reasons and without loss of generality, we leave out the discussion related to data types, such as datatype properties, etc. They can be treated similarly as object properties.

The BOWL Interpretation

A BAF extended interpretation \mathcal{I}_b is a pair $\mathcal{I}_b = (\Delta^{\mathcal{I}_b}, \cdot^{\mathcal{I}_b})$, where $\Delta^{\mathcal{I}_b}$ is, as in the OWL case, the *domain of interpretation* and $\cdot^{\mathcal{I}_b}$ is the interpretation function. In OWL, the interpretation function maps an individual name into a member of the domain; a class name into a set of elements in the domain and a property name into a set of pairs of domain elements $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The BOWL interpretation still maps an individual name into a member of the domain $\Delta^{\mathcal{I}_b}$. However, it maps a class (*resp.* a

4.4. Belief-augmented OWL (BOWL)

property) into a function from members of $\Delta^{\mathcal{I}_b}$ (*resp.* pairs of members of $\Delta^{\mathcal{I}_b}$) into a BAF value.

$$\begin{aligned} \mathcal{C}^{\mathcal{I}_b} &: \Delta^{\mathcal{I}_b} \rightarrow \mathbf{BAF} \\ \mathcal{P}^{\mathcal{I}_b} &: \Delta^{\mathcal{I}_b} \times \Delta^{\mathcal{I}_b} \rightarrow \mathbf{BAF} \end{aligned}$$

As defined above, each of the BOWL classes and properties is a function returning a BAF value. Intuitively, the BAF interpretation returns this value as the belief/disbelief value of an individual (*resp.* a pair of individuals) being a member of the class (*resp.* property).

The class expressions in BOWL are interpreted as functions returning a BAF value as shown in Table 4.4. With a little abuse of notation, we use lambda expressions to represent the functions.

The duality of the concept relationships in the BOWL interpretation are worth discussing. In the following, the symbol \cong denotes concept equivalence. We have in BOWL $\neg \top \cong \perp$, $C \sqcap \top \cong C$, $C \sqcup \top \cong \top$, $C \sqcap \perp \cong \perp$, $C \sqcup \perp \cong C$, $\neg \neg C \cong C$, $\neg (C \sqcup D) \cong (\neg C) \sqcap (\neg D)$, $\neg (C \sqcap D) \cong (\neg C) \sqcup (\neg D)$, $C_1 \sqcap (C_2 \sqcup C_3) \cong (C_1 \sqcap C_2) \sqcup (C_1 \sqcap C_3)$, $C_1 \sqcup (C_2 \sqcap C_3) \cong (C_1 \sqcup C_2) \sqcap (C_1 \sqcup C_3)$. For concepts involving roles, we have $\neg (\forall P.C) \cong \exists P.(\neg C)$, $\forall P.\top \cong \top$, $\exists P.\perp \cong \perp$ and $(\forall P.C) \sqcap (\forall P.D) \cong \forall P.(C \sqcap D)$. The proof of these equivalence relationships are obvious to see and omitted.

Class Expression	Interpretation
\mathcal{C}	$C^{\mathcal{I}_b} \in \Delta^{\mathcal{I}_b} \rightarrow \text{BAF}$
\top	$\lambda a. \phi_{\text{one}}$
\perp	$\lambda a. \phi_{\text{zero}}$
$\mathcal{C}_1 \sqcup \mathcal{C}_2$	$\lambda a. \mathcal{C}_1^{\mathcal{I}_b}(a) \cup^{\mathcal{B}} \mathcal{C}_2^{\mathcal{I}_b}(a)$
$\mathcal{C}_1 \sqcap \mathcal{C}_2$	$\lambda a. \mathcal{C}_1^{\mathcal{I}_b}(a) \cap^{\mathcal{B}} \mathcal{C}_2^{\mathcal{I}_b}(a)$
$\neg \mathcal{C}$	$\lambda a. \neg^{\mathcal{B}} \mathcal{C}^{\mathcal{I}_b}(a)$
$\forall P. \mathcal{C}$	$\lambda a. \bigcap_{b \in \Delta^{\mathcal{I}_b}}^{\mathcal{B}} (\neg^{\mathcal{B}} P^{\mathcal{I}_b}(a, b) \cup^{\mathcal{B}} \mathcal{C}^{\mathcal{I}_b}(b))$
$\exists P. \mathcal{C}$	$\lambda a. \bigcup_{b \in \Delta^{\mathcal{I}_b}}^{\mathcal{B}} (P^{\mathcal{I}_b}(a, b) \cap^{\mathcal{B}} \mathcal{C}^{\mathcal{I}_b}(b))$
$P: o$	$\lambda a. P^{\mathcal{I}_b}(a, o^{\mathcal{I}_b})$
$\geq n P$	$\lambda a. \bigcup^{\mathcal{B}} (\bigcap_{i=1}^n P^{\mathcal{I}_b}(a, b_i)), \text{ for all } \{b_1, b_2, \dots, b_n\} \subseteq \Delta^{\mathcal{I}_b}$
$\leq n P$	$\lambda a. \neg^{\mathcal{B}} (\bigcup^{\mathcal{B}} (\bigcap_{i=1}^{n+1} P^{\mathcal{I}_b}(a, b_i))), \text{ for all } \{b_1, \dots, b_{n+1}\} \subseteq \Delta^{\mathcal{I}_b}$
$\{a_1, \dots, a_n\}$	$\lambda a. \text{ IF } a = a_i^{\mathcal{I}_b} \text{ THEN } \phi_{\text{one}} \text{ ELSE } \phi_{\text{zero}}$

Table 4.4: BOWL class expressions & their interpretations

Interpreting BOWL Axioms

Because BOWL interprets classes and properties in a different way, the conditions for BOWL axioms to be satisfied by an BOWL interpretation have to be changed too.

Definition 4.4.1 (Class Subsumption) *A given abstract BOWL interpretation \mathcal{I}_b satisfies a class subsumption axiom $C \sqsubseteq D$ if and only if $\text{DI}(\bigcap_{a \in \Delta^{\mathcal{I}_b}}^{\mathcal{B}} (\neg^{\mathcal{B}} C^{\mathcal{I}_b}(a) \cup^{\mathcal{B}} D^{\mathcal{I}_b}(a)))$.*

4.4. Belief-augmented OWL (BOWL)

$$D^{\mathcal{I}_b}(a))) > 0.$$

The above interpretation is made by viewing $C \sqsubseteq D$ as a first-order logic formula $\forall c. c \in C \rightarrow c \in D \equiv \forall c. (\neg c \in C) \vee c \in D$. Intuitively, the universal quantifier is translated to the distributed BAF conjunction, the logical not is translated to a BAF negation and the logical or is translated to a BAF or. The class subsumption axiom is true if and only if the degree of inclination of the corresponding BAF is positive.

Definition 4.4.2 (Class Equivalence) *A given abstract BOWL interpretation \mathcal{I}_b satisfies a class subsumption axiom $C \equiv D$ if and only if $\text{DI}(\bigcap_{a \in \Delta^{\mathcal{I}_b}}^{\mathcal{B}} ((C^{\mathcal{I}_b}(a) \cap^{\mathcal{B}} D^{\mathcal{I}_b}(a)) \cup^{\mathcal{B}} (\neg^{\mathcal{B}} C^{\mathcal{I}_b}(a) \cap^{\mathcal{B}} D^{\mathcal{I}_b}(a)))) > 0$.*

Similarly we define BOWL axiom satisfaction conditions for all OWL axioms. For the sake of space, we do not elaborate them all.

Interpreting BOWL Assertions

As shown above, the BOWL interpretation is of the form $\langle \alpha, b \rangle$, where α is an assertion in OWL and b is a value in BAF. For assertion $\langle \alpha, b \rangle$, its interpretation is just the BAF value b .

Knowledge Base, Satisfiability and Entailment

As for the OWL case, the BOWL knowledge base consists of a finite number of BOWL axioms and assertions. We denote the knowledge base by Σ , the TBox by Σ_T and the ABox by Σ_A .

A BOWL interpretation \mathcal{I}_b satisfies a knowledge base Σ iff it satisfies all of its elements (axioms and/or assertions). Then the interpretation is a *model* of the knowledge base.

The satisfiability of an axiom/assertion of the form $\langle \alpha, b \rangle$ by a BOWL interpretation is denoted by $\mathcal{I}_b \models \langle \alpha, b \rangle$.

Since BAF is a pair of values in the range of $[0, 1]$, a single value needs to be derived from this pair of values to determine the satisfiability of the axiom by the interpretation.

The Utility function U defined in BAF (Section 4.3) is a normalized version of the Degree of Inclination, which is the difference between the belief and disbelief values. Given a BAF value, its utility gives the overall truth value, ideally suited for the above purpose.

Therefore, $\mathcal{I}_b \models \langle \alpha, b \rangle$ iff $U(\alpha^{\mathcal{I}_b}) \geq U(b)$, where α can be an axiom, $C_1 \sqsubseteq C_2$, $C_1 = C_2$ or $C_1 \sqcap C_2 = \perp$; or an assertion $\langle a \rangle \in C$ or $\langle a_1, a_2 \rangle \in P$, etc. The interpretation of α is, as given in the previous two subsections, a BAF value. Hence, the utility function U gives a single value for comparison. If the utility of the interpretation of

4.5. Reasoning about BOWL

α is greater than or equal to that of b , then we conclude that the axiom is satisfied by the interpretation.

A knowledge base Σ entails an axiom $\langle \alpha, b \rangle$, denoted by $\Sigma \models \langle \alpha, b \rangle$, iff it is satisfied by each of the models (interpretation) of Σ . Similarly, a knowledge base Σ entails an assertion $\langle \alpha, b \rangle$, denoted by $\Sigma \models \langle \alpha, b \rangle$, iff all its models satisfies $\langle \alpha, b \rangle$.

A BOWL interpretation \mathcal{I}_b satisfies an assertion $\langle a \in C, b \rangle$ iff $\mathsf{U}(C^{\mathcal{I}_b}(a^{\mathcal{I}_b})) \geq \mathsf{U}(b)$.

The same applies to the case of property assertions of the form $\langle \langle a_1, a_2 \rangle \in P, b \rangle$, *i.e.*, $\mathsf{U}(P^{\mathcal{I}_b}(a_1^{\mathcal{I}_b}, a_2^{\mathcal{I}_b})) \geq \mathsf{U}(b)$.

4.5 Reasoning about BOWL

As we have discussed earlier, each fact in OWL is augmented with a BAF, a pair consisting of a belief value and a disbelief value. In this section, we present some algorithms for BOWL class membership and property membership entailment. More specifically, given an ontology and a BOWL class or property membership assertion, the algorithm determines if the ontology entails the BOWL assertion.

4.5.1 Class Membership

We explain the algorithm for class membership. Algorithm 1 is simply the outer algorithm which calls Algorithm 2 to compute the BAF values of a class membership

assertion in an ontology.

<p>Data: Ontology \mathcal{O}, and BOWL assertion $\langle a \in \mathcal{C}, \beta \rangle$ where a is an individual, \mathcal{C} is a class description and β is a BAF</p> <p>Result: Returns true if $\mathcal{O} \models \langle a \in \mathcal{C}, \beta \rangle$ and false otherwise compute the belief values, γ, of $a \in \mathcal{C}$ in \mathcal{O} by Algorithm 2; return $\gamma \geq^{\mathcal{B}} \beta$;</p>

Algorithm 1: BOWL class membership assertion entailment

Data: Ontology \mathcal{O} and OWL assertion $a \in C$ where a is an individual and C is a class description

Result: Computes the BAF values of $a \in C$ in \mathcal{O} , or $\text{compute}(\mathcal{O}, a, C)$

```

if  $\langle a \in C, \beta \rangle$  is given in  $\mathcal{O}$  then
    | return  $\beta$ ;
else
    if  $C$  is a class name then
         $\beta_1 \leftarrow \phi_{\text{zero}}$ ;
        for  $D_i \sqsubseteq C$  do
            |  $\beta_1 \leftarrow \beta_1 \cup^{\mathcal{B}} \text{compute}(\mathcal{O}, a, D_i)$ ;
        end
         $\beta_2 \leftarrow \phi_{\text{zero}}$ ;
        for  $D_i = C$  or  $C = D_i$  do
            |  $\beta_2 \leftarrow \beta_2 \cup^{\mathcal{B}} \text{compute}(\mathcal{O}, a, D_i)$ ;
        end
         $\beta_3 \leftarrow \phi_{\text{zero}}$ ;
        for  $\geq 1 P_i \sqsubseteq C$  do
            |  $\gamma \leftarrow \phi_{\text{zero}}$ ;
            for  $(a, a_j) \in P$  do
                |  $\gamma \leftarrow \gamma \cup^{\mathcal{B}} \text{compute}(\mathcal{O}, a, a_j, P_i)$ ;
            end
             $\beta_3 \leftarrow \beta_3 \cup^{\mathcal{B}} \gamma$ ;
        end
         $\beta_4 \leftarrow \phi_{\text{zero}}$ ;
        for  $\top \sqsubseteq \forall P.C$  do
            |  $\gamma \leftarrow \phi_{\text{zero}}$ ;
            for  $(a_j, a) \in P$  do
                |  $\gamma \leftarrow \gamma \cup^{\mathcal{B}} \text{compute}(\mathcal{O}, a, a_j, P_i)$ ;
            end
             $\beta_4 \leftarrow \beta_4 \cup^{\mathcal{B}} \gamma$ ;
        end
        return  $\beta_1 \cup^{\mathcal{B}} \beta_2 \cup^{\mathcal{B}} \beta_3 \cup^{\mathcal{B}} \beta_4$ ;
    endif
    else if  $C$  is  $\top$  then
        | return  $\phi_{\text{one}}$ ;
    endif
    else if  $C$  is  $\perp$  then
        | return  $\phi_{\text{zero}}$ ;
    endif
    else if  $C$  is  $C_1 \sqcup C_2$  then
        | return  $\text{compute}(\mathcal{O}, a, C_1) \cup^{\mathcal{B}} \text{compute}(\mathcal{O}, a, C_2)$ ;
    endif
    else
        | omitted...
    endif
endif
    
```

Algorithm 2: Computing class membership assertion belief values

4.5. Reasoning about BOWL

Algorithm 2 works as follows. It first checks if the requested BAF values are already given in the ontology. If so, it halts and returns it directly. Otherwise it computes the BAF values according to the types of the class description. If it is the top or bottom class, it simply returns ϕ_{one} and ϕ_{zero} respectively. If it is a class name, it computes the BAF values by four axioms, namely class subsumption, class equivalence, property domain and property range, doing a BAF disjunction of the four results which are obtained by recursively calling Algorithm 2 and 4. If the class description is a class union, we return the BAF union of the BAF values for the individual to be the instance of the two classes. The other cases are closely related to the semantics of class descriptions introduced in Section 4.4.3 and are left out for brevity.

4.5.2 Property Membership

The reasoning algorithm for property membership is simpler than that for class membership, because neither OWL nor BOWL allows the notion of property description; we can describe a property only by referencing its name. Thus the reasoning algorithms are described in Algorithm 3 and 4, where Algorithm 3 is the outer algorithm for property membership entailment and Algorithm 4 is for computing the belief values of a given property membership assertion. In Algorithm 4, we only consider sub property, equivalent property, inverse property and transitive property relationships

and take the BAF disjunction when computing the belief values.

Data: Ontology \mathcal{O} , and BOWL assertion $\langle (a_1, a_2) \in P, \beta \rangle$ where a_1 and a_2 are two individual, P is a property name and β is a BAF
Result: Returns true if $\mathcal{O} \models \langle (a_1, a_2) \in P, \beta \rangle$ and false otherwise
compute the belief values, γ , of $(a_1, a_2) \in P$ in \mathcal{O} by Algorithm 4;
return $\gamma \geq^B \beta$;

Algorithm 3: BOWL property membership assertion entailment

Data: Ontology \mathcal{O} and OWL assertion $(a_1, a_2) \in P$ where a_1, a_2 are individuals and P is a property name
Result: Computes the BAF values of $(a_1, a_2) \in P$ in \mathcal{O} , or $\text{compute}(\mathcal{O}, a_1, a_2, P)$

```

if  $\langle (a_1, a_2) \in P, \beta \rangle$  is given in  $\mathcal{O}$  then
  | return  $\beta$ ;
else
  |  $\beta_1 \leftarrow \phi_{\text{zero}}$ ;
  | for  $Q_i \sqsubseteq P$  do
  |   |  $\beta_1 \leftarrow \beta_1 \cup^B \text{compute}(\mathcal{O}, a_1, a_2, Q_i)$ ;
  | end
  |  $\beta_2 \leftarrow \phi_{\text{zero}}$ ;
  | for  $Q_i = P$  or  $P = Q_i$  do
  |   |  $\beta_2 \leftarrow \beta_2 \cup^B \text{compute}(\mathcal{O}, a_1, a_2, Q_i)$ ;
  | end
  |  $\beta_3 \leftarrow \phi_{\text{zero}}$ ;
  | for  $P = (\neg Q_i)$  or  $(\neg Q_i) = P$  do
  |   |  $\beta_3 \leftarrow \beta_3 \cup^B \text{compute}(\mathcal{O}, a_1, a_2, Q_i)$ ;
  | end
  |  $\beta_4 \leftarrow \phi_{\text{zero}}$ ;
  | if  $\text{Tr}(P)$  then
  |   | for  $(a_1, b_i), (b_i, a_2) \in P$  do
  |     |  $\gamma = \text{compute}(\mathcal{O}, a_1, b_i, Q_i) \cap^B \text{compute}(\mathcal{O}, b_i, a_2, Q_i)$ 
  |     |  $\beta_4 \leftarrow \beta_4 \cup^B \gamma$ ;
  |   | end
  | else
  |   endif
  | return  $\beta_1 \cup^B \beta_2 \cup^B \beta_3 \cup^B \beta_4$ ;
endif

```

Algorithm 4: Computing property membership assertion belief values

4.5.3 Simple Implementation in $\text{CLP}(\mathcal{R})$

Following the reasoning algorithms above, we have implemented a simple reasoner for BOWL in $\text{CLP}(\mathcal{R})$.

For class membership computations, we define a CLP predicate `computeClass(I,C,A,B)` with I being an individual, C being a class (which can be both a class name or a complicated class description), A being the belief value and B being the disbelief value for the particular class membership assertion. For property membership computations, we similarly define a CLP predicate `computeClass(I1,P,I2,A,B)` with $I1$ and $I2$ being individuals, P being a property name, A being the belief value and B being the disbelief value for the particular property membership assertion. Then for the entailment of class membership assertions, we define a CLP predicate `entails(instance(I,C),A,B)` with I being an individual, C being a class, and A and B being the belief and disbelief values respectively. Similarly for the entailment of property membership assertions, we define a predicate `entails(sub_val(I1,P,I2),A,B)` with $I1$ and $I2$ being individuals, P being a property name, A being the belief value and B being the disbelief value for the particular property membership assertion. Then the algorithms can be easily converted to $\text{CLP}(\mathcal{R})$ programmes. A partial code listing for Algorithm 2 can be found in Appendix. I. Because our CLP program is highly recursive, we apply the coinductive tabling [64] techniques to prevent infinite loops and reduce unnecessary invocations of the rules.

4.6 Case Study

In this section, we present an example in the sensor fusion domain to demonstrate the derivation of belief values using BAF-Logics. Sensor fusion [23] technologies aim at fusing information from different sensors (possibly of different types) to detect, recognize, identify or track a target. Sensor fusion has important applications in the defense domain where accurate sensor decisions minimize casualty and improve strike efficiency.

Decision fusion [23] is a branch of sensor fusion technology where sensors are combined in various configurations (parallel, serial, etc.) and their decisions are given confidence factors by the decision fusion processor, which calculates the final decision after a number of iterations.

We believe that decision fusion can be a new application domain for the Semantic Web as sensors may reside at different geographical sites and communications between sensors and the decision fusion processor can be expressed in terms of ontologies to maximize portability and inter-operability of the sensor networks.

4.6.1 The Sensor Ontology

We developed a BOWL ontology which defines taxonomies of sensors, environmental conditions, targets, etc. It also defines an object-property *isAffectedBy*, capturing the

4.6. Case Study

fact that sensors are affected by environmental conditions. Furthermore we define a class *CurrentCondition* as a sub class of *Environment* and a BAF value is attached to every environment condition instance to capture how certain we are about the presence of the condition in the working environment of the sensors. Ontology fragments are presented in the DL syntax. Note that whenever the BAF value is omitted from the quadruple, it is assumed to be ϕ_{one} , which is $\langle 1, 0 \rangle$.

$$\begin{aligned} \textit{Sensor} &\sqsubseteq \top \\ \textit{ActiveSensor} &\sqsubseteq \textit{Sensor} \\ \textit{PassiveSensor} &\sqsubseteq \textit{Sensor} \\ \textit{EMFrequencySensor} &\sqsubseteq \textit{Sensor} \\ \textit{OtherSensor} &\sqsubseteq \textit{Sensor} \\ \geq 1 \textit{ is_affected_by} &\sqsubseteq \textit{Sensor} \\ \top &\sqsubseteq \forall \textit{ is_affected_by}. \textit{CurrentCondition} \\ \textit{Environment} &\sqsubseteq \top \\ \textit{CurrentCondition} &\sqsubseteq \textit{Environment} \\ \textit{LightCondition} &\sqsubseteq \textit{Environment} \\ \textit{TerrainCondition} &\sqsubseteq \textit{Environment} \\ \textit{WeatherCondition} &\sqsubseteq \textit{Environment} \\ \textit{RainCondition} &\sqsubseteq \textit{WeatherCondition} \\ \textit{SmokeCondition} &\sqsubseteq \textit{WeatherCondition} \\ \textit{WindCondition} &\sqsubseteq \textit{WeatherCondition} \end{aligned}$$

Sensors and sensor networks identify targets. Thus we define a class called *Target* and an object-property called *identifies* as follows. The object property *identifies* has as domain the union of the classes *Sensor* and *SensorNetwork* and has as range *Target*.

$$\begin{aligned} \textit{Target} &\sqsubseteq \top \\ \geq 1 \textit{ identifies} &\sqsubseteq (\textit{Sensor} \sqcup \textit{SensorNetwork}) \\ (\top &\sqsubseteq \forall \textit{ identifies}. \textit{Target}) \end{aligned}$$

Certain kinds of sensors are more affected by certain environmental conditions than

others. To capture this information in BOWL, we could have defined some sub properties of *isAffectedBy*, such as *isSlightlyAffectedBy* and *isSeverelyAffectedBy*. Unfortunately, such properties are by no means clear or meaningful to software agents, decision fusion processors in this case. Linguistic hedges like “very” and “quite” are impossible to represent in bi-valued logic systems such as classic description logics, but can be easily captured in belief systems such as BAF-Logic. In our example, different sensors are affected by various environmental conditions differently. We unify several properties into a single property *isAffectedBy*. Then its fuzzy set [120] is $\{completely, severely, moderately, slightly, not\}$. Now, we can assign a BAF supporting value to each element in the fuzzy set to express the extent to which a certain sensor is affected by an environmental condition. Following the conventions used in fuzzy sets, the *isAffectedBy* set may be modeled as $\{1/completely, 0.75/severely, 0.5/moderately, 0.25/slightly, 0/not\}$.

Here we assume complete knowledge (no ignorance) of the sensors and compute the refuting masses by subtracting the supporting masses from 1. Then we get the BAF values $\langle 1, 0 \rangle$, $\langle 0.75, 0.25 \rangle$, $\langle 0.50, 0.50 \rangle$, $\langle 0.25, 0.75 \rangle$ and $\langle 0, 1 \rangle$ respectively.

The following shows how BAFs are added to ground facts (instances). We know the existence of *Speed7Wind1*, a wind instance and a chemical sensor *ChmSensor1*. *ChmSensor1* is affected by *Speed7Wind1* with supporting and refuting measures of 0.75, 0.25 respectively.

4.6. Case Study

$$\begin{aligned} &\langle \text{Speed7 Wind1} \in \text{WindCondition}, \langle 1.0, 0.0 \rangle \rangle \\ &\langle \text{Speed7 Wind1} \in \text{CurrentCondition}, \langle 0.8, 0.1 \rangle \rangle \\ &\langle \text{ChmSensor1} \in \text{ChemicalSensor}, \langle 1.0, 0.0 \rangle \rangle \\ &\langle (\text{ChmSensor1}, \text{Speed7 Wind1}) \in \text{isAffectedBy}, \langle 0.75, 0.25 \rangle \rangle \end{aligned}$$

4.6.2 Computing Confidence Values of Sensors

As stated in [23], initial sensor confidence values are important for the overall system performance. In our examples, the initial sensor confidence values are an effect of the conjunction of several environment conditions. We define a class *TrustedSensor* to denote the belief that the decision fusion processor puts in a particular sensor.

$$\text{TrustedSensor} \equiv \text{Sensor} \sqcap \forall \text{isAffectedBy}. \text{CurrentCondition}$$

This axiom states that the degree of a sensor instance is trusted depends on how it is affected by current conditions. As defined in the semantics, if a certain sensor is affected by more than one environmental condition, the conjunction of all these conditions is taken into account. For example, night vision devices are severely affected by both rain conditions and smoke conditions. We first define the class *NightVisionDevice* as follows. All subsumption axioms have BAF value of ϕ_{one} .

$$\begin{aligned} \text{EMFrequencySensor} &\sqsubseteq \text{Sensor} \\ \text{ElectroOpticalSensor} &\sqsubseteq \text{EMFrequencySensor} \\ \text{NightVisionDevice} &\sqsubseteq \text{ElectroOpticalSensor} \end{aligned}$$

Suppose that a night-vision device nvd_1 is currently deployed in an area where both rain $rain_1$ and smoke $smoke_1$ are present. The sensor fusion processor has certain

belief value for each condition to be *current*. Since night vision devices are severely affected by these conditions, we associate a BAF value $\langle 0.75, 0.25 \rangle$ to each of the property instances for *isAffectedBy* as follows.

$$\begin{aligned} &\langle nvd_1 \in \text{NightVisionDevice}, \langle 1.0, 0.0 \rangle \rangle \\ &\langle rain_1 \in \text{RainCondition}, \langle 1.0, 0.0 \rangle \rangle \\ &\langle smoke_1 \in \text{SmokeCondition}, \langle 1.0, 0.0 \rangle \rangle \\ &\langle rain_1 \in \text{CurrentCondition}, \langle 0.7, 0.3 \rangle \rangle \\ &\langle smoke_1 \in \text{CurrentCondition}, \langle 0.5, 0.5 \rangle \rangle \\ &\langle (nvd_1, rain_1) \in \text{isAffectedBy}, \langle 0.75, 0.25 \rangle \rangle \\ &\langle (nvd_1, smoke_1) \in \text{isAffectedBy}, \langle 0.75, 0.25 \rangle \rangle \end{aligned}$$

Our goal is to see whether the knowledge entails the assertion that nvd_1 is a trusted sensor with belief value $\langle 0.7, 0.3 \rangle$. So we formulate a CLP goal

`entails(instance(nvd1, trustedSensor), 0.7, 0.3)`

The programme terminates and returns a “No”. This means that the BAF values of the assertion calculated from the BOWL ontology gives a smaller utility value than $\langle 0.7, 0.3 \rangle$ does. If we are interested in finding the precise BAF values of the specific assertion we can fire the CLP goal

`computeClass(nvd1, trustedSensor, A, B)`

Then the programme returns $A = 0.5$ and $B = 0.5$. Further analysis shows $U(\langle 0.5, 0.5 \rangle) = 0.5 < 0.7 = U(\langle 0.7, 0.3 \rangle)$. Therefore, $U(\alpha) = 0.7 > U(\langle 0.5, 0.5 \rangle) = 0.5$. Hence, sensor nvd_1 cannot be inferred to be a trusted sensor with that high confidence.

4.7 Chapter Summary

In this chapter, we propose BOWL, Belief-augmented OWL, as an ontology language enriched with belief information. As an extension of OWL DL, BOWL can be used to associate belief and disbelief factors directly with web resources, enabling software agents to perform more flexible and accurate reasoning. We define the abstract syntax of BOWL and augment the model-theoretic semantics of OWL to incorporate belief values. We also define the reasoning tasks and algorithms for BOWL and present a prototype implementation using the constraint logic programming technique.

Expert systems are used to assist decision making in individual narrow application domains, such as the medical domain. Historically, uncertainty has been an research subject in expert systems. A comprehensive survey can be found at [85]. Probabilistic measures were used in expert systems to deal with uncertainty. For example, the Pathfinder project [52] uses subjective probability theory, belief networks, for decision support system for hematopathology diagnoses. It uses influence diagrams to track dependencies among observed features. Possibility theory was also incorporated in some expert systems for dealing with uncertain information. The Cadiag-2 [1] system was developed to diagnose rheumatic, hepatic and pancreatic diseases. It is based on fuzzy theory and fuzzy set [120] and uses fuzzy inference to propagate and track belief. In general, expert systems are very limited in scope and specifically designed for particular application domains. On the contrary, the Semantic Web is envisioned to

be an open environment, encompassing more complex and more unreliable resources and data.

There have been many proposals [70, 102, 107] on probabilistic/fuzzy extensions to description logics such as \mathcal{ALC} and CLASSIC, which are less expressive than the description logic ($\mathcal{SHOIN}(D)$) on which is OWL based. Ding and Peng proposed a Bayesian network-based probabilistic extension to OWL [25]. The main focus of their works is the modeling of a priori and conditional probabilities of OWL classes and the reasoning tasks are concept satisfiability, overlapping and subsumption, which is different from ours. Nottelmann and Fuhr proposed a probabilistic Datalog-based extension to DAML+OIL [86]. Their approach is less general than ours in the sense that a fact with both true and false evidence present is considered inconsistent, whereas in our approach evidence for and against a fact are allowable and ignorance is computed based on these values. Straccia [102] proposed a fuzzy extension for the description logic \mathcal{ALC} , which is less expressive than $\mathcal{SHOIN}(D)$. Based on this work, Straccia proposed a fuzzy extension for OWL [103]. Again, based on fuzzy set, each assertion is associated with a single value representing its fuzziness. Therefore, our approach is more flexibility as software agents may very possibly receive both supporting and refuting values for a certain assertion. BOWL enables agents to make use of these values at their own discretion. The other approaches, however, hide this step of processing and is hence more rigid and less transparent.

Chapter 5

Checking Rule-based Agent

Knowledge

In computer science, rule-based systems are used as a way to store and manipulate knowledge to interpret information in a useful way. They are often used in artificial intelligence applications and research because they are modular, easy to understand, executable, expressive and declarative.

Rule-based agents have played an important role in other areas of AI, as evinced by rule-based agent architectures such as SOAR [71] and Sim-Agent [100]. Such architectures allow a great degree of abstraction in specifying the behaviour of agents. Rule-based programming extensions are also increasingly being offered as add-ons to

existing, lower-level, agent toolkits, e.g. JADE [13] and FIPA-OS [93].

The consistency and correctness of their knowledge bases are vital to the proper functioning of intelligent agents. For Semantic Web agent in particular, ontology reasoning tools such as RACER [50] and FaCT++ [108] have been developed for this purpose. With the inclusion of the rule language into the family of ontology languages, it is important to verify not only that an ontology is consistent *with respect to* a set of rules but also that the set of rules is consistent by itself. Most current ontology reasoning tools have mainly focused on ontology *without* rules.

In Chapter 3, we have demonstrated how PVS, a generic theorem prover, can be used to verify SWRL rules and beyond in the ontology-based agent knowledge representation. This ensures that an ontology is consistent *with respect to* a set of rules. In this chapter we go one step further by proposing an ontology and rule verification mechanism for discovering rule anomalies such as inconsistency, redundancy and circularity among rules by combining the Constraint Logic Programming (CLP) framework and the state-of-the-art Semantic Web reasoning technique. We place our focus on OWL DL from which SWRL is extended.

This chapter is organized as follows. Section 5.1 introduces the Semantic Web Rule Language (SWRL) with a focus on its semantics. We discuss how rules formulated on top of ontologies can be analyzed for anomalies in Section 5.2. We describe our prototype implementation in Section 5.3 and apply our approach to a context-aware

5.1. Semantic Web Rules Language

system in Section 5.4. Section 5.5 concludes the contribution of the chapter and discusses some related work.

5.1 Semantic Web Rules Language

Extended from OWL DL, Semantic Web Rules Language (SWRL) [56] is syntactically and semantically coherent to OWL. The major extensions of SWRL with respect to OWL includes Horn-style rules and (universally quantified) variable declaration. The rules are in the form of **antecedent** \rightarrow **consequent**, where both antecedent and consequent are conjunctions of atoms. Atoms can be of the form $C(x)$, $P(x, y)$, $sameAs(x, y)$, $differentFrom(x, y)$ or a pre-defined built-ins where C is an OWL description, P is an OWL property, and x and y are either variables, OWL individuals or OWL data values. Informally, a rule means that if the antecedent holds, the consequent must also hold.

A simple example rule shown below states that if action task $?b$ is a sub task of action task $?a$, then $?b$'s duration should be inside of $?a$'s duration.

$$\begin{aligned} & ActionTask(?a) \wedge ActionTask(?b) \wedge \\ & differentFrom(?a, ?b) \wedge hasAsASubAction(?a, ?b) \wedge \\ & startTime(?a, ?ast) \wedge endTime(?a, ?aet) \wedge \\ & startTime(?b, ?bst) \wedge endTime(?b, ?bte) \\ & \rightarrow lessThan(?ast, ?bst) \wedge lessThan(?bte, ?aet) \end{aligned}$$

Formally the semantics of SWRL is an extension of that of OWL.

Definition 5.1.1 (Binding) *Given an abstract OWL interpretation I , a binding $B(I)$ is an abstract OWL interpretation that extends I such that S maps individual variables to elements of O and L maps data variables to elements of LV respectively.*

A rule atom is satisfied by a binding under some conditions described in [56].

Definition 5.1.2 (Rule Satisfaction) *A binding $B(I)$ satisfies an antecedent A if and only if A is empty or $B(I)$ satisfies every atom in A . A binding $B(I)$ satisfies a consequent C if and only if C is not empty and $B(I)$ satisfies every atom in C . A rule is satisfied by an interpretation I if and only if for every binding $B(I)$ such that $B(I)$ satisfies the antecedent, $B(I)$ also satisfies the consequent.*

The semantic conditions relating to facts, axioms and ontologies are unchanged, with rules being treated as axioms.

5.2 Analyzing Agent Rule Bases

We are able to detect three types of anomalies related to a set of rules, namely inconsistency, redundancy and circularity, as identified in [94]. In this section we give formal definitions for each type of anomalies and describe how a combination of description logic reasoners and CLP can be used to detect the anomalies.

5.2. Analyzing Agent Rule Bases

To discover rule anomalies, each SWRL rule is translated into a CLP atom of the form `rule(name,head,body)` where `name` is the name of the rule, `head` and `body` are two lists of atom which are translated from the rule head and rule body respectively. We translate rule variables into CLP variables and rule constants into CLP constants accordingly. Keeping the rule variables as CLP variables makes it easier for unification which is important when we want to establish subsumption relationships between two lists of predicates. It should also be noted that we require the ontology on which the rules are based to be checked for consistency *before* the rules can be analyzed for various types of anomalies.

5.2.1 Inconsistency

Rule inconsistency means that the SWRL rules are not correct. In other words, conflicting results can be inferred from the rules. To differentiate rule inconsistency from ontology inconsistency, we do not consider SWRL rules as OWL axioms.

Definition 5.2.1 (Rule Inconsistency) *A set of rules R defined on top of an ontology O containing a collection of axioms and facts is inconsistent with respect to a datatype map D if and only if there is some interpretation I such that I satisfies all axioms and facts, but for all such I there always exists a rule r in R such that there exists some binding B such that $B(I)$ satisfies the antecedent of r , but does not*

satisfy the consequent of r .

We define three types of inconsistency, namely conflicting antecedent/consequent, contradicting consequents and chained inconsistency.

Conflicting Antecedent/Consequent

Inconsistency by conflicting antecedent occurs when a rule has two or more antecedent atoms which cannot be satisfied at the same time.

Definition 5.2.2 (Conflicting Antecedent) *A rule $A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow C_1 \wedge C_2 \wedge \dots \wedge C_m$ is said to have a conflicting antecedent if and only if $\neg (A_{i_1} \wedge \dots \wedge A_{i_p})$ for some $1 \leq i_1, \dots, i_p \leq n$.*

In the above definition, $\neg (A_{i_1} \wedge \dots \wedge A_{i_p})$ is interpreted as “with the inclusion of assertions A_{i_1} through A_{i_p} (variables replaced by fresh constants), the original ontology becomes inconsistent”. Similar interpretation is employed in the following definitions in this section.

In terms of the original model-theoretic (MT) definition, we have

Definition 5.2.3 (Conflicting Antecedent(MT)) *A rule r defined on top of an ontology O containing a collection of axioms and facts is said to have a conflicting*

5.2. Analyzing Agent Rule Bases

antecedent if and only if for all interpretations I that satisfies the axioms and facts of O , there exists some binding B such that $B(I)$ does not satisfy the antecedent of r .

We check rules with conflicting antecedent as follows. First we check the consistency of the given ontology. Then for each rule, we assert the antecedent atoms into the original ontology in FaCT++ as facts and collect the built-ins. For each atom assertion, if we encounter a individual ID or a data value, we assert it as it is. If we encounter a SWRL rule variable, we replace it with a fresh individual name not existing in the ontology and assert the new atom. In such a way we assert the antecedent atoms with all the variables skolemized into constants. Then we check the consistency of the resultant ontology and solve the collected constraints using the CLP constraint solving capability. The SWRL rule has conflicting antecedent if and only if the resultant ontology is inconsistent or the collected constraints are unsatisfiable. In the end we remove all asserted atoms from the ontology to preserve its correctness.

Inconsistency by conflicting consequent occurs when a rule has two or more consequent atoms which cannot be satisfied at the same time.

Definition 5.2.4 (Conflicting Consequent) *A rule $A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow C_1 \wedge C_2 \wedge \dots \wedge C_m$ is said to have a conflicting consequent if and only if $\neg (C_{j_1} \wedge \dots \wedge C_{j_q})$ for some $1 \leq j_1, \dots, j_q \leq m$.*

Similarly we have the model-theoretic definition.

Definition 5.2.5 (Conflicting Consequent(MT)) *A rule r defined on top of an ontology O containing a collection of axioms and facts is said to have a conflicting consequent if and only if for all interpretations I that satisfies the axioms and facts of O , there exists some binding B such that $B(I)$ does not satisfy the consequent of r .*

The steps for checking conflicting consequent are similar to those for checking conflicting antecedent, only that we assert the consequent atoms instead of the antecedent atoms.

Contradicting Consequents

Inconsistency by contradicting consequents means that a rule, whose antecedent subsumes that of another rule, has a consequent atom contradicting with that of the other rule. First we define conjunction subsumption.

Definition 5.2.6 (Conjunction Subsumption) *Consider two conjunctions of rule atoms*

$$l_1 : A_1 \wedge A_2 \wedge \dots \wedge A_n$$

$$l_2 : B_1 \wedge B_2 \wedge \dots \wedge B_m$$

where each A_i and B_j are predicates over some constants and variables. Therefore we do not consider built-in atoms at the moment. l_1 is subsumed by l_2 (or l_2 subsumes

5.2. Analyzing Agent Rule Bases

l_1) under the substitutions σ_1 and σ_2 , denoted $l_1 \subseteq_{(\sigma_1, \sigma_2)} l_2$, if and only if $\{A_i\sigma_1 \mid 1 \leq i \leq n\} \subseteq \{B_j\sigma_2 \mid 1 \leq j \leq m\}$.

The substitution we discuss in this paper is restricted to replacing a variable with another variable or replacing a variable to a constant, but not replacing a constant with a variable.

Definition 5.2.7 (Antecedent/Consequent Subsumption) *Consider two antecedents (consequents) of rule atoms*

$$l_1 : AP_1 \wedge \dots \wedge AP_{n_1} \wedge AC_1 \wedge \dots \wedge AC_{m_1}$$

$$l_2 : BP_1 \wedge \dots \wedge BP_{n_2} \wedge BC_1 \wedge \dots \wedge BC_{m_2}$$

where AP_1 through AP_{n_1} and BP_1 through BP_{n_2} are SWRL non-built-in atoms and AC_1 through AC_{m_1} and BC_1 through BC_{m_2} are SWRL built-ins. Then we have l_1 is subsumed by l_2 (or l_2 subsumes l_1) under the substitutions σ_1 and σ_2 , denoted $l_1 \subseteq_{(\sigma_1, \sigma_2)} l_2$, if and only if (1) $AP_1 \wedge \dots \wedge AP_{n_1} \subseteq_{(\sigma_1, \sigma_2)} BP_1 \wedge \dots \wedge BP_{n_2}$, and (2) $(BC_1 \wedge \dots \wedge BC_{m_2})\sigma_2 \Rightarrow (AC_1 \wedge \dots \wedge AC_{m_1})\sigma_2$.

In this definition, the symbol \Rightarrow means logical implication. Intuitively speaking, we treat the subsumption relationship of antecedents (consequents) *with* built-ins as follows. We group the non-built-in atoms of each antecedent (consequent) together and check if these two conjunctions of non-built-in atoms have a subsumption relation

under two substitutions. If so, we apply the two substitutions to the two remaining groups of built-ins and check if an implication relation can be established between these two groups of built-ins.

In terms of model-theoretic semantics, we have the following generalized definition for the subsumption.

Definition 5.2.8 (Antecedent/Consequent Subsumption (MT)) *Given an abstract interpretation I , a conjunction of rule atoms l_1 is subsumed by a conjunction of rule atoms l_2 if and only if there exists a binding B such that if $B(I)$ satisfies l_2 , then it also satisfies l_1 .*

The above definition of subsumption is general in a semantic sense and is computationally expensive. In implementation, we implement the following CLP rules for checking term subsumption in terms of a substitution. By terms we mean single CLP predicates. $T1$ and $T2$ are predicates, possibly with constants and variables. `subsume(T1,T2)` succeeds if $T2$ is more general than $T1$. The operator `=..` converts a CLP atom $p(t_1, \dots, t_n)$ into a list of length $n + 1$ with the p being the head of the list and a list containing t_1 through t_n as the tail of the list. We omit details of low level implementation for predicates such as `skolemization`. Basically the program assigns constants to the variables in $T1$ and matches $T1$ to $T2$.

5.2. Analyzing Agent Rule Bases

```
subsume(T1,T2) :-  
    T1 =.. [Type|L11], T2 =.. [Type|L21],  
    L11 = [H1|L1], L21 = [H2|L2],  
    assert(store(L1,L2)), retract(store(M1,M2)),  
    skolemization(M1),  
    find_substitution(M1,M2,L1,L2).
```

Then we can easily construct CLP rules for subsumption between two lists of terms. At this point the unification mechanism of CLP plays an important part in ensuring that the substitution is carried forward from one term subsumption to the others, thus helping us deal with the problem of maintaining the same substitution with ease.

```
list_subsume(_, []). list_subsume(L2, [X1|L1]) :-  
    list_subsume2(L2,X1), list_subsume(L2,L1).  
list_subsume2([Y|_],X) :- subsume(Y,X), list_subsume2([_|R],X) :-  
list_subsume2(R,X).
```

We also implement CLP code to check if a set of built-ins imply another set of built-ins. More specifically, to check if $X_1, \dots, X_n \Rightarrow Y_1, \dots, Y_m$ we simply query the goal $X_1, \dots, X_n, \text{not } (Y_1, \dots, Y_m)$. The implication holds if and only if the goal fails. We implement this using a CLP predicate `builtin_imply/2`. Here the constraint solving capability is utilized for checking implication relations between sets of built-ins in an easy and natural way. At the moment, most comparison and mathematical built-ins can be handled with ease, except that for some built-ins, we are restricted by the limitation of $\text{CLP}(\mathcal{R})$. For example, we require that the constraints involving *swrlb:multiply*, *swrlb:divide*, *swrlb:integerDivide*, *swrlb:mod*, *swrlb:pow*, *swrlb:abs*, *swrlb:sin*, *swrlb:cos*, *swrlb:tan* to be linear for the constraint solver in CLP to work.

Then we can formally define the notion of inconsistency by contradicting consequents.

Definition 5.2.9 (Contradicting Consequents) *Two rules*

$$A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow C_1 \wedge C_2 \wedge \dots \wedge C_m$$

$$B_1 \wedge B_2 \wedge \dots \wedge B_p \rightarrow D_1 \wedge D_2 \wedge \dots \wedge D_q$$

are said to have contradicting consequents if and only if there exists substitutions σ_1 and σ_2 , $A_1 \wedge A_2 \wedge \dots \wedge A_n \subseteq_{(\sigma_1, \sigma_2)} B_1 \wedge B_2 \wedge \dots \wedge B_p$ and $\neg ((C_1 \wedge C_2 \wedge \dots \wedge C_m)\sigma_1 \wedge (D_1 \wedge D_2 \wedge \dots \wedge D_q)\sigma_2)$.

In terms of model-theoretic definition, we have

Definition 5.2.10 (Contradicting Consequents(MT)) *Two rules r_1 and r_2 defined on top of an ontology O containing a collection of axioms and facts are said to have contradicting consequents if and only if (1) the antecedent of r_1 is subsumed by that of r_2 (2) for all interpretations I that satisfy all axioms and facts, (i) I satisfies r_1 and r_2 separately and (ii) there exists some binding B such that $B(I)$ satisfies the antecedents of r_1 and r_2 , but does not satisfy the consequent of r_1 and r_2 .*

The steps for checking inconsistency by contradicting consequents are as follows. As the first step, we eliminate all possible rule inconsistency by conflicting consequent. We then run the CLP code in the following code box to get the pairs of rules with

5.2. Analyzing Agent Rule Bases

subsuming antecedents. Then for each pair, we skolemize the variables in the two sets of consequent atoms and assert them into the ontology. Following that we check for ontology inconsistency by using the FaCT++ reasoner. The rule has contradicting consequents if the new ontology is inconsistent. At last we remove the asserted atoms after each round.

```
subsumingBody(R1,R2) :-  
    rule(R1,Head1,Body1), rule(R2,Head2,Body2),  
    not R1 = R2, list_subsume(Body2,Body1).
```

Chained Inconsistency

Unlike inconsistency by conflicting antecedent or contradicting consequents, which involves one or two rules, chained inconsistency involves an arbitrary number of rules and is the most expensive to find. Informally speaking, chained inconsistency occurs when a set of rules can be connected to form a chain by using \rightarrow and the consequent of the last rule consists of an atom which is in conflict with an antecedent element of the first rule.

Definition 5.2.11 (Chained Inconsistency) *A set of p rules*

$$\begin{aligned} A_1^1 \wedge A_2^1 \wedge \dots \wedge A_{n_1}^1 &\rightarrow C_1^1 \wedge C_2^1 \wedge \dots \wedge C_{m_1}^1 \\ &\dots \\ A_1^i \wedge A_2^i \wedge \dots \wedge A_{n_i}^i &\rightarrow C_1^i \wedge C_2^i \wedge \dots \wedge C_{m_i}^i \end{aligned}$$

...

$$A_1^p \wedge A_1^p \wedge \dots \wedge A_{n_p}^p \rightarrow C_1^p \wedge C_2^p \wedge \dots \wedge C_{m_p}^p$$

is said to have chained inconsistency if and only if there exists p substitution σ_1 through σ_p such that for all q where $2 \leq q \leq p$, $A_1^q \wedge \dots \wedge A_{n_q}^q \subseteq_{(\sigma_q, \sigma_{q-1})} C_1^{q-1} \wedge \dots \wedge C_{m_{q-1}}^{q-1}$ and $\neg ((A_1^1 \wedge A_2^1 \wedge \dots \wedge A_{n_1}^1)\sigma_1 \wedge (C_1^p \wedge C_2^p \wedge \dots \wedge C_{m_p}^p)\sigma_p)$. Then the chain is said to have length $p - 1$.

We detect inconsistency by chained inconsistency by the following steps. We allow the users to provide the chain length as a parameter, say n . Then we start from length 1, that is, we start detecting chained inconsistency by two rules. We invoke the following CLP rules with **Length** being 1. This will give us a set of rule pairs satisfying the chained condition. Then we do the skolemized assertion of the antecedent atoms of the first rule and the consequent atoms of the second rule and check for ontological inconsistency in FaCT++. After the checking we remove all asserted atoms. A chained inconsistency of length 1 is found if and only if an inconsistency is found by FaCT++. Then we repeat the process for lengths up to n .

```

chained(R1,Length) :-
    rule(R1,Head1,Body1), rule(R2,Head2,Body2),
    not R1 = R2, list_subsume(Body2,Head1),
    chained(Body1,R2,Length-1).
chained(_,_,0). chained(Body1,R2,Length) :-
    rule(R2,Head2,Body2), rule(R3,Head3,Body3),
    not R2 = R3, list_subsume(Body3,Head2),
    chained(Body1,R3,Length-1).

```

5.2.2 Redundancy

Unlike inconsistency, redundancy does not mean that the rule or the rule set is incorrect, but rather that the rule set can be more succinct. However, according to our experience with the case studies, it is often the case that redundancy is a result of defining some rule incorrectly. Hence it is useful to identify rule redundancy as an indication of rule inconsistency. We define three types of redundancy, namely subsumption of rules, implication of superclasses and antecedent redundancy.

Subsumption of Rules

A rule R is redundant if another rule R' subsumes it. A rule R' subsumes another rule R if the antecedent of R' is more general than that of R and the consequent of R is more general than that of R' .

Definition 5.2.12 (Rule Subsumption) *Consider two rules*

$$R_1 : A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow C_1 \wedge C_2 \wedge \dots \wedge C_m$$

$$R_2 : B_1 \wedge B_2 \wedge \dots \wedge B_p \rightarrow D_1 \wedge D_2 \wedge \dots \wedge D_q$$

R_2 subsumes R_1 if and only if for some substitutions σ_1 and σ_2 , $B_1 \wedge \dots \wedge B_p \subseteq_{(\sigma_2, \sigma_1)}$

$A_1 \wedge \dots \wedge A_n$ and $C_1 \wedge \dots \wedge C_m \subseteq_{(\sigma_1, \sigma_2)} D_1 \wedge \dots \wedge D_q$.

The model-theoretic definition of rule subsumption is as follows.

Definition 5.2.13 (Rule Subsumption(MT)) *Given an abstract interpretation I , a rule r_1 is subsumed by a rule r_2 if and only if there exists a binding B such that (1) if $B(I)$ satisfies the antecedent of r_1 , then it also satisfies the antecedent of r_2 and (2) if $B(I)$ satisfies the consequent of r_2 , then it also satisfies the consequent of r_1 .*

This rule redundancy can be checked completely by CLP programs without the need for description logic reasoner. The CLP rules for detecting subsumption between rule are as follows. We reuse some rules defined earlier, such as `builtin_imply` for checking constraint implication.

```
rule_subsumption :-
    rule(R1,Head1,Body1), rule(R2,Head2,Body2),
    not R1 = R2,
    list_subsume(Head1,Head2),
    builtin_imply(Head1,Head2),
    list_subsume(Body2,Body1),
    builtin_imply(Body2,Body1).
```

Antecedent Redundancy

Antecedent redundancy means that some antecedent atoms can be inferred from the rest of the antecedent atoms. In other words, the redundant atom can be removed.

Definition 5.2.14 (Antecedent Redundancy) *A rule $A_1 \wedge \dots \wedge A_n \rightarrow C_1 \wedge \dots \wedge C_m$ is said to have antecedent redundancy if and only if $A_{j_1} \wedge \dots \wedge A_{j_k} \Rightarrow A_i$ where $j_1, \dots, j_k, i \in \{1, \dots, n\}$ and $i \notin \{j_1, \dots, j_k\}$.*

5.2. Analyzing Agent Rule Bases

More formally, the notion of antecedent redundancy is defined as follows.

Definition 5.2.15 (Antecedent Redundancy(MT)) *A rule $A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow C_1 \wedge C_2 \wedge \dots \wedge C_m$ is said to have antecedent redundancy if and only if for all interpretations I , every binding $B(I)$ that satisfies a strict subset of the antecedent satisfies an antecedent atom outside the subset.*

We detect antecedent redundancy by using only FaCT++ as follows. For each rule, we skolemize all variables as we do for the previous types of anomalies. Then for each atom, we assert the rest of the atoms into the ontology and see if the atom can be inferred from the new ontology. The atom is redundant and hence can be removed from the rule if it can be inferred from the new ontology.

5.2.3 Circularity

A rule set is said to have a circularity anomaly when a rule has circular dependencies between its antecedent and consequent or when two rules have mutual dependency between their antecedents and consequents. While circularity will not cause any problems in some cases, it sometimes leads to non-termination of rule invocation. So we believe that it is useful to indicate rule circularity in a rule set and let the designers decide if the circularity is intended.

Definition 5.2.16 (*Single-rule Circularity*) A rule $A_1 \wedge \dots \wedge A_n \rightarrow C_1 \wedge \dots \wedge C_m$ has single-rule circularity if and only if $C_j \Rightarrow A_i$, for some i and j where $1 \leq i \leq n, 1 \leq j \leq m$.

More formally, the notion of single-rule circularity is defined as follows.

Definition 5.2.17 (**Single-rule Circularity**) A rule $A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow C_1 \wedge C_2 \wedge \dots \wedge C_m$ is said to have single-rule circularity if and only if for all interpretations I , every binding $B(I)$ that satisfies an consequent atom satisfies an antecedent atom.

The algorithm for detecting single-rule circularity is similar to that for detecting antecedent and consequent redundancy. For this case, we check, by using FaCT++, for each antecedent atom to see if it can be inferred from the ontology after we skolemize the rule and assert the consequent atoms.

Definition 5.2.18 (*Double-rule Circularity*) Consider two rules

$$R_1 : A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow C_1 \wedge C_2 \wedge \dots \wedge C_m$$

$$R_2 : B_1 \wedge B_2 \wedge \dots \wedge B_p \rightarrow D_1 \wedge D_2 \wedge \dots \wedge D_q$$

R_1 and R_2 have double-rule circularity if and only if there exists substitutions σ_1 and σ_2 such that $C_i \subseteq_{(\sigma_1, \sigma_2)} B_1 \wedge B_2 \wedge \dots \wedge B_p$ and $D_j \subseteq_{(\sigma_2, \sigma_1)} A_1 \wedge A_2 \wedge \dots \wedge A_n$ where $1 \leq i \leq m, 1 \leq j \leq q$.

5.3. Prototype Implementation

Double-rule circularity can be checked by using only CLP. The CLP rules for detecting double-rule circularity anomalies are as follows.

```
double_circularity :-  
    rule(R1,Head1,Body1), rule(R2,Head2,Body2),  
    not R1 = R2, member(H1,Head1),  
    member(H2,Head2), list_subsume([H1],Body2),  
    builtin_imply([H1],Body2),  
    list_subsume([H2],Body1),  
    builtin_imply([H2],Body1).
```

5.3 Prototype Implementation

This work is a continuation from our previous work [34, 33, 35, 28, 29], in which we applied a spectrum of formal methods and tools such as Z, Alloy, and PVS to complement the state-of-the-art ontology reasoning tools in checking some ontology-based domain models to verify some advanced properties beyond the expressiveness of the ontology languages. In [29], we presented a tools environment to realize our approach. The tools environment allows a domain engineer to construct or load a given ontology, to invoke a standard reasoner to check the consistency of the ontology, to issue queries to the ontology, to translate the ontology into various formalisms and languages, to invoke external tools to verify complex domain properties. The tool is implemented in Java and uses a number of external packages such as Jena [59], Protege [69] and OWL-API [12].

We extend the tool environment to implement the SWRL rule verification mechanism

described in this paper. A screen shot of the tool with some sample rules is shown in Fig. 5.1. The tabs in the main window accommodate the original ontology, the translated specification in various formalisms. The SWRL Rules tab is divided into two blocks. The left block is used for displaying the SWRL rules found in the ontology which is loaded in the ontology tab.

The rules are displayed in a table. For easier readability, we have employed a human readable syntax for the rules. This syntax is modified from what is described in [56]. In this syntax, a rule is broken down to three parts, namely the rule name, the antecedent atoms and the consequent atoms. Both antecedent atoms and consequent atom are conjunctions of atoms in the form $a_1 \wedge a_2 \wedge \dots \wedge a_n$. The conjunction sign is written as the character “ \wedge ”. Variables are indicated using the standard convention of prefixing them with a question mark (e.g. ?x). All ontology entities such as classes, properties, individuals and rules use local names, which means that name spaces are not displayed.

There are three buttons on the left block above the table for creating a new rule, editing a selected rule and deleting a selected rule respectively. For rule creation and modification, we implement a predictive editing mechanism in which the users can construct the rules by having the tool automatically generating part of the rule (for example the matching parentheses) and by choosing from a list of referenced classes, properties and individuals. Every time a rule is changed, we perform checking on the

5.3. Prototype Implementation

syntax and rule safety by which we mean all variables that appear in the consequent atoms appear in the antecedent and all class names, property names and individual names that appear in the antecedent and consequent are defined in the ontology.

The block on the right hand side of the SWRL rule tab contains all the buttons for performing various types rule verification, with those for inconsistency, redundancy and circularity grouped together respectively. If a button, say Conflicting Antecedent, gets pushed, it will go through all the rules currently in the rule table on the left and find the first rule which has the corresponding anomaly. The result of the checking is given by highlighting the rules and the atoms with anomalies. The ontology engineer can choose to edit the rule, ignore the anomaly and proceed with checking, or abort the checking operation. The Check All button is used when the ontology engineer wants to check for all types of anomalies in one go.

Although we primarily use the DL reasoner FaCT++ and the constraint logic programming technique, we hide these two internal tools away from the domain engineers by interfacing with the DIG sockets and local temporary files. So the rule checking is completely automated and the integration of the formalisms is seamless.

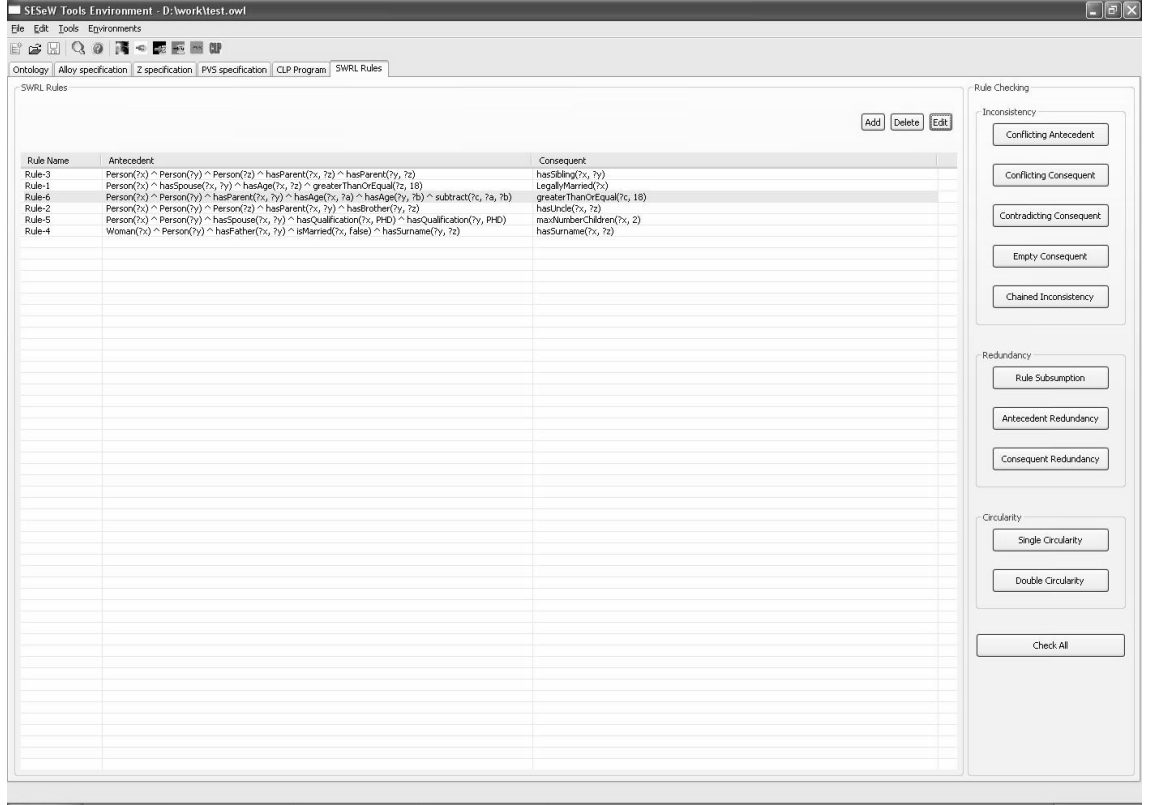


Figure 5.1: Prototype Screen Shot

5.4 Case Study

In this section, we present preliminary experimental results of applying our approach to an actual ontology-based context-aware system. The objective of these experiments is to conduct a quantitative feasibility study for checking ontology rule anomalies.

Our case study is an extended version of CONON [115] which was proposed to model general context in a pervasive computing environment. It provides an upper context

5.4. Case Study

ontology that captures general concepts as well as extensibility for adding domain-specific ontology in a hierarchical manner. A typical and successful use of ontologies for context representation, CONON is part of a Semantic Space infrastructure [114] which was deployed by networking a diversity of sensors and context wrappers for sensing and markup of various contexts including location, schedule, temperature, noise, light, door status, device status and computer application status.

The original CONON was encoded in pure OWL DL. The user-defined context reasoning rules are small in number (10 rules) and are specified using horn-like rules, but not exactly in SWRL. We extend CONON ontology with more classes and properties. We also enrich the set of user-defined rules and specify them in legal SWRL. The extended ontology contains 223 OWL classes, 1136 RDF triples and 23 rules. The experiments are conducted on a WindowsXP system with a Pentium 4, 3.0GHz processor and 1GB DDR2 RAM.

Our program reports four anomaly cases. The first anomaly detected is of the type antecedent redundancy resulted by the following two rules. It takes our program 1.8 seconds to detect this anomaly.

$$\begin{aligned}
 & \text{User}(\text{?}u) \wedge \text{Workstation}(\text{?}ws) \wedge \text{hasWorkstation}(\text{?}u, \text{?}ws) \wedge \\
 & \text{hasScheduledSkypeMeeting}(\text{?}u, \text{Now}) \wedge \\
 & \text{hasApplication}(\text{?}ws, \text{Skype}) \wedge \text{hasStatus}(\text{Skype}, \text{On}) \wedge \\
 & \text{Microphone}(\text{?}mp) \wedge \text{hasDevice}(\text{?}ws, \text{?}mp) \wedge \\
 & \text{hasStatus}(\text{?}mp, \text{On}) \\
 & \quad \rightarrow \text{hasSituation}(\text{?}u, \text{AtSkypeConversation})
 \end{aligned}$$

$$\begin{aligned}
 & \text{User}(\text{?}u) \wedge \text{Workstation}(\text{?}ws) \wedge \text{hasWorkstation}(\text{?}u, \text{?}ws) \wedge \\
 & \text{hasScheduledSkypeMeeting}(\text{?}u, \text{Now}) \wedge \\
 & \text{hasApplication}(\text{?}ws, \text{Skype}) \\
 & \quad \rightarrow \text{hasStatus}(\text{Skype}, \text{On})
 \end{aligned}$$

The first rule is a situation inference rule which says that if a user has a scheduled Skype meeting at the current time and both the Skype application and microphone on his workstation are on, then he or she has the situation *AtSkypeConversation*. The second rule is used to automatically trigger the user's Skype application if he or she has a scheduled Skype meeting at the current time. After a closer investigation of the rules, we find that in the first rule the status of Skype is redundant and can be inferred from the rest of the antecedent atoms as a result of the second rule. Hence $\text{hasApplication}(\text{?}ws, \text{Skype})$ and $\text{hasStatus}(\text{Skype}, \text{On})$ should be removed.

The second anomaly detected is of the type subsumption of rules resulted by the following two rules. It takes our program 2.3 seconds to detect this anomaly.

5.4. Case Study

$$\begin{aligned} & \text{User}(?u) \wedge \text{LivingRoom}(?lr) \wedge \text{TV}(?tv) \wedge \\ & \text{isLocatedIn}(?u, ?lr) \wedge \text{isLocatedIn}(?tv, ?lr) \wedge \\ & \text{hasStatus}(?tv, \text{On}) \\ & \rightarrow \text{hasSituation}(?u, \text{Entertaining}) \end{aligned}$$
$$\begin{aligned} & \text{User}(?u) \wedge \text{LivingRoom}(?lr) \wedge \text{TV}(?tv) \wedge \text{GamePlayer}(?gp) \wedge \\ & \text{isLocatedIn}(?u, ?lr) \wedge \text{isLocatedIn}(?tv, ?lr) \wedge \\ & \text{isLocatedIn}(?gp, ?lr) \wedge \text{hasStatus}(?tv, \text{On}) \\ & \text{hasStatus}(?gp, \text{On}) \\ & \rightarrow \text{hasSituation}(?u, \text{Entertaining}) \end{aligned}$$

These two rules are used to capture the different cases of a user having a situation of *Entertaining*, i.e., when he or she is watching TV or playing video game. The first rule says that if a user is in the living room where the TV is on, then he/she is entertaining. The second rule says that if a user is in the living room where both the TV and the game player are on, then he/she is entertaining. The first rule has less conditions (more general antecedent), but a conclusion as strong as that of the second rule. According to the definition of rule subsumption, the first rule subsumes the second rule. Therefore the second rule is redundant and should be removed without loss of information. Actually after studying the system requirements, both rules are needed and two situations should be distinguished. The conclusions of the two rules need to be strengthened to *WatchingTV* and *PlayingGame* respectively.

The third kind of anomaly detected is of the type chained inconsistency resulted from the following three rules. It takes our program 7.6 seconds to detect this anomaly.

```

AmlyopiaPatient(?u) ∧ LightService(?lightService) ∧
Ward(?w) ∧ isLocatedIn(?u, ?w) ∧ isLocatedIn(?ls, ?w) ∧
hasStatus(?ls, On)
    → hasCondition(?u, Awake)

User(?u) ∧ condition(?u, AWAKE)
    → DayTime(Now)

LightService(?ls) ∧ DayTime(Now)
    → status(?ls, Off)

```

The first rule is used to infer if a patient is awake. It says that if a user has vision defects and the light of his or her ward is on, then the user is awake. The second rule says that if a user is awake, then it means now is day time. The third rule says that in the day time, the light should be switched off. Each of the three rules seems to make some sense by itself at a glance, but will cause an inconsistency when put together. More specifically we have chain conditions from the first rule to the third rule. It gives a contradiction that light on implies light off. After analyzing the rules and the system requirements, we found the second rule is wrong and needs to be removed.

The last anomaly detected is of the type circularity resulted from the following two rules. It takes our program 2.5 seconds to detect this anomaly.

```

User(?u) ∧ Room(?r) ∧ LightService(?ls) ∧
isLocatedIn(?u, ?r) ∧ isLocatedIn(?ls, ?r) ∧
isSleepingTime(?u, Now) ∧ hasStatus(?ls, Off)
    → hasSituation(?u, Sleeping)

User(?u) ∧ Room(?r) ∧ LightService(?ls) ∧
isLocatedIn(?u, ?r) ∧ isLocatedIn(?ls, ?r) ∧
hasSituation(?u, Sleeping)
    → hasStatus(?ls, Off)

```

5.5. Chapter Summary

The first rule is a typical rule used to infer a high level situation, *Sleeping*, from low level context information, light status and time. Serving a different purpose, the second rule is used to control the light service of a room according to the occupant's situation. Our tool detects a double-rule circularity. Indeed, assuming no original or asserted knowledge about the situation or light status, a query to either of these two goals will cause the rules to fire one after another repeatedly. After analyzing the system, we find the knowledge about light status can be asserted by light service controller and light sensor wrapper into the knowledge base. In that case the non-termination will not happen.

5.5 Chapter Summary

The Semantic Web provides standard, formal and semantic-rich languages to represent an intelligent agent's knowledge, greatly enhancing the interoperability of heterogeneous agents. While the inclusion of rule language into the Semantic Web has added more expressiveness, it also introduces new challenges for ontology reasoning at the same time. As new facts can be inferred from rules, it is important to provide standard ontology reasoning support in the presence of rules. Furthermore as rules can also be redundant and/or inconsistent by themselves, it is important to provide facilities to check rule anomalies too. In this chapter we propose to use a combina-

tion of standard DL reasoning and the constraint logic programming techniques to discover anomalies in a SWRL agent rule base.

Some popular Semantic Web tools have recently included support for reasoning about rules. The widely used ontology editor Protégé [69] has a plugin for SWRL Jess [42] tab. It is a (partial) SWRL reasoner implemented on top of a rule-based system. It suffers from the limitations that it does not support class restrictions and that it does not handle the inconsistency between rule-inferred knowledge and the OWL ontology, which are all supported in our tool. Pellet [99] recently releases preliminary support for DL-safe SWRL reasoning, a subset of SWRL rules. But it does not support anonymous classes, datatype properties or built-in functions, which are easily dealt with in our approach.

Researchers have also made considerable amount of effort to combine the strengths of both the Semantic Web and logic programming. One branch is on extending the Semantic Web with logic programming rules for more expressiveness [83]. Their focus is on adding more modelling power to ontology languages while preserving decidability. In the other branch the possibility of using logic programming techniques for reasoning about ontologies is studied [17, 95, 48, 42, 36]. Compared to these works, which focus on only ontology reasoning, our approach has much stronger support for rules. We support not only a much larger set of SWRL built-ins, thanks to the strong support from $\text{CLP}(\mathcal{R})$ for concrete domains, but also more importantly

5.5. Chapter Summary

checking rule anomalies which none of the above tools supports. The only previous work on discovering anomalies among SWRL rules is [10]. But they focus on the basic features of SWRL and OWL. SWRL built-ins are omitted. Only a subset of OWL DL containing subclass relations and property transitivity, complement and disjointness is considered.

Chapter 6

Checking Higher-order Agent Knowledge

In the previous chapters we have considered providing various reasoning support for agent knowledge in the form of facts (Chapter 3), facts with uncertainties (Chapter 4 and rules(Chapter 3 and 5). In this chapter we take the knowledge of agent to a higher level; we are interested in reasoning about the knowledge of an intelligent agent about the knowledge of other agents.

The area of multi-agent systems is traditionally concerned with formal representation of the mental state of autonomous agents in a distributed setting. For this purpose, many modal logics have been developed and investigated. Among them epistemic

logic, the logic of knowledge, is one of the most studied and has grown to find diverse applications such as artificial intelligence in computer science and game theory in economics as a means of reasoning about the knowledge and/or belief of agents. Typical applications include protocol verification in computer securities [37, 80].

Epistemic logic typically deals with what agents consider possible given their current information. This includes knowledge about facts as well as higher-order information about information that other agents have.

The knowledge of an agent is more complex than a simple collection of static data; it evolves with time, typically as a result of agent communication. Public announcement is one of the simplest form of communication action. Public announcement logic [92] extends normal epistemic logics with modal operator for public announcement. These logics can be perceived as a basis not only for specification languages of a particular spectrum of multi-agent systems, but also for mechanized machine-aided reasoning.

Researchers have proposed several approaches towards reasoning about agent knowledge represented in modal logics, including tableau-based provers and SAT-based algorithms with representative systems such as FaCT [54] and $K_{SAT}C$ [45]. Recently some state-of-the-art model checkers [43, 90, 111] have been developed for automated verification of epistemic properties. However such approaches suffer from some major drawbacks. Firstly, the system to be verified has to be fully specified even if the property only concerns with a fragment of the system. Secondly, as the sizes of the

states and relation are exponential to the number of proposition of the system, the model checkers suffer from what is known as the state explosion problem. The task of representing and verifying against all possible computations of a system may not be problematic for small examples, but may become infeasible for realistic multi-agent systems. Lastly and perhaps most importantly, these model checkers typically deal with systems with fixed number of epistemic states. But we are often faced with systems with an arbitrary number of epistemic states as the number of agents is neither fixed nor known in advance. Consequently the properties are often beyond the expressiveness of epistemic logic and hence cannot be verified by model checkers.

In this chapter we explore a complementary approach. In the specification language of a well established interactive theorem prover, we build a reasoning framework which consists of (1) logic-level proof systems for deriving logic theorems, (2) theorem sets for storing the logic-level theorems, (3) object-level reasoning systems for application modelling and verification, and (4) reasoning rule sets for the object-level reasoning system. With this separation of concerns between the logic meta-level and application object-level, we are able to not only derive all valid formulae of a logic but also specify multi-agent applications and perform verification under one umbrella.

Other than obtaining a sound and complete reasoning system, many other advantages arise from using this translation approach. Firstly we can exploit the well supported theorem prover for the purpose of doing proofs in the multi-agent logic. Secondly as

we are able to quantify over functions, we obtain the generality and power of higher-order logic. Thirdly, theories in PVS can be easily extended and reused. This means that we can extend our framework to support other epistemic logics with minimal effort. At the same time, system developers can easily select the suitable reasoning system to specify and verify the system being developed. Lastly we can utilize the power of proof strategies in PVS for proof automation.

This chapter is organized as follows. In Section 6.1, we provide a brief overview of some well accepted epistemic logics, detailing their language syntax and semantics and some model checkers for epistemic logics. We describe our reasoning framework in detail in Section 6.2, which also includes a discussion on how the system is typically used. An example will be used to explain the proof process and how we use proof strategies to enhance automation in Section 6.3. Section 6.4 summarizes the contribution of the chapter and compares some related work.

6.1 Epistemic Logic

In computer science, it is often useful to reason about modes of truth. Modal logic, or (less commonly) intensional logic is the branch of logic that deals with sentences that are qualified by modalities such as can, could, might, may, must, possibly, and necessarily, and others. A formal modal logic represents modalities using modal

6.1. Epistemic Logic

sentential operators. The basic set of modal operators are usually given to be \Box and \Diamond . In alethic modal logic (i.e. the logic of necessity and possibility) \Box represents necessity and \Diamond represents possibility.

When applied to knowledge representation and reasoning about multi-agent systems, the specific type of modal logics is called epistemic logic. For example each of many interacting agents may have different knowledge about the environment. Furthermore, each agent may have different knowledge about the knowledge of other agents. The formula $\Box\varphi$ is read as: it is known that φ .

In the context of epistemic logic, one can view worlds that are possible for an agent in a world as epistemic alternatives, that are compatible with the agent's information in that world.

Epistemic logic K is the weakest epistemic logic that does not have any 'optional' formula schemes. It is based on a set of atomic propositions and a set of agents. It just contains propositional logic and all instances of formula scheme K . Here the operator K has exactly the same properties as \Box .

Definition 6.1.1 *Let \mathcal{P} be the set of atomic propositions, and \mathcal{A} a set of agents. The language of the logic, \mathcal{L}_K is defined by the following grammar.*

$$\phi ::= \top \mid \perp \mid p \mid \neg \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid \phi \leftrightarrow \phi \mid K_a \phi$$

where $p \in \mathcal{P}$ and $a \in \mathcal{A}$. $K_a \phi$ means that 'agent a knows that ϕ holds'.

Also known as $S5$, $KT45$ is probably one of the most well known epistemic logics.

Having the same language as logic K , $KT45$ adds three axioms:

- Truth: The agent knows only true things.
- Positive Introspection: If an agent knows something, he knows that he knows it.
- Negative Introspection: If the agent does not know something, he knows that he does not know it.

When reasoning about the knowledge of a group of agents, it becomes useful to reason not just about knowledge of an individual agent, but also about the knowledge of the group. Epistemic logic $KT45^n$ which is also known as $S5C$ extends $KT45$ by providing support for shared knowledge and common knowledge among a set of agents.

Definition 6.1.2 *The language of the logic, \mathcal{L}_{KT45^n} is defined by the following grammar.*

$$\phi ::= \top \mid \perp \mid p \mid \neg \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid \phi \leftrightarrow \phi \mid K_a \phi \mid E_G \phi \mid C_G \phi$$

where $E_G \phi$ (shared knowledge) means that every agent in the group G knows ϕ and $C_G \phi$ (common knowledge) means that every agent in G knows about ϕ and every

6.1. Epistemic Logic

agent knows that every agent knows ϕ , etc. It captures a higher state of knowledge and can be thought of as an infinite conjunction $E_G\phi \wedge E_GE_G\phi \wedge E_GE_GE_G\phi \wedge \dots$

First studied by Lewis [72], the notion common knowledge has received much attention in the area of economics and computer science after Aumann's seminal result [6]. The inclusion of common knowledge for a group of agents adds much more complexity to the task of reasoning about multi-agent systems. As a result, many previous reasoning systems of epistemic logic have left out the notion of common knowledge.

The knowledge of an agent is more complex than a collection of static data; it evolves typically as a result of agent communication. Dynamic epistemic logics analyze changes in both basic and higher-order information. A public announcement in public announcement logic (PAL) [92] is an epistemic update where all agents commonly know that they learn that a certain formula holds. Public announcement logic with common knowledge (PAL-C) extends PAL with support for common knowledge.

Definition 6.1.3 *The language of the logic, \mathcal{L}_{PAL} is defined by the following grammar.*

$$\phi ::= \top \mid \perp \mid p \mid \neg \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid \phi \leftrightarrow \phi \mid K_a\phi \mid [\phi]\phi$$

The language of the logic, $\mathcal{L}_{\text{PAL-C}}$ is defined by the following grammar.

$$\phi ::= \top \mid \perp \mid p \mid \neg \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid \phi \leftrightarrow \phi \mid K_a\phi \mid E_G\phi \mid C_G\phi \mid [\phi]\phi$$

where $[\varphi]\psi$ means that ' ψ holds after every announcement of φ '.

There have been various discussions of the equivalence and translations between $S5$ and PAL [8, 44]. Every formula in the language of public announcement logic without common knowledge is equivalent to a formula in the language of epistemic logic.

Theorem 6.1.1 *For any arbitrary atomic proposition p and PAL formulae φ , ψ and χ , the following hold.*

$$\begin{array}{lll}
 [\varphi]p & \leftrightarrow & (\varphi \rightarrow p) \\
 [\varphi](\psi \wedge \chi) & \leftrightarrow & ([\varphi]\psi \wedge [\varphi]\chi) \\
 [\varphi](\psi \rightarrow \chi) & \leftrightarrow & ([\varphi]\psi \rightarrow [\varphi]\chi) \\
 [\varphi]\neg \psi & \leftrightarrow & (\varphi \rightarrow \neg [\varphi]\psi) \\
 [\varphi]K_a\psi & \leftrightarrow & (\varphi \rightarrow K_a[\varphi]\psi) \\
 [\varphi][\psi]\chi & \leftrightarrow & [\varphi \wedge [\varphi]\psi]\chi
 \end{array}$$

These results conveniently provide us with a rewrite system that allows us to eliminate announcement from the logical language. In other words PAL is a syntactical extension to $S5$ and is equivalent to $S5$. However when common knowledge is added, an equivalence cannot be formulated, thus creating complexity for reasoning with common knowledge in dynamic epistemic logic.

6.1. Epistemic Logic

6.1.1 Semantics

Typically the semantics of various epistemic logics are given using the idea of *possible worlds* and *Kripke structures*.

Definition 6.1.4 *Given a set of atomic propositions \mathcal{P} and a set of agents \mathcal{A} , a Kripke model is a structure $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{V} \rangle$, where*

- \mathcal{S} is a set of states or possible worlds. It is sometimes also called the domain $\mathcal{D}(\mathcal{M})$ of \mathcal{M} .
- $\mathcal{R} : \mathcal{A} \rightarrow \mathcal{S} \times \mathcal{S}$ is a function, which maps from each agent $a \in \mathcal{A}$ to its possibility relation. Intuitively, $(s, t) \in \mathcal{R}(a)$ if agent a cannot differentiate between s and t .
- $\mathcal{V} : \mathcal{P} \rightarrow 2^{\mathcal{S}}$ is an evaluation function that for every $p \in \mathcal{P}$ yields the set of states in which p is true.

Epistemic formulae are interpreted on epistemic states (\mathcal{M}, s) consisting of a Kripke model $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{V} \rangle$ and a state $s \in \mathcal{S}$.

Definition 6.1.5 *Given a model $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{V} \rangle$ we have that a formula φ is true in (\mathcal{M}, s) , written as $\mathcal{M}, s \models \varphi$, as follows:*

$\mathcal{M}, s \models p$	<i>iff</i>	$s \in \mathcal{V}(p)$
$\mathcal{M}, s \models (\varphi \wedge \psi)$	<i>iff</i>	$\mathcal{M}, s \models \varphi$ and $\mathcal{M}, s \models \psi$
$\mathcal{M}, s \models \neg \varphi$	<i>iff</i>	not $\mathcal{M}, s \models \varphi$
$\mathcal{M}, s \models K_a \varphi$	<i>iff</i>	for all t such that $(s, t) \in \mathcal{R}(a)$, $\mathcal{M}, t \models \varphi$
$\mathcal{M}, s \models E_G \varphi$	<i>iff</i>	for all a such that $a \in G$, $\mathcal{M}, t \models K_a \varphi$
$\mathcal{M}, s \models C_G \varphi$	<i>iff</i>	for all t such that $(s, t) \in T^*$, $\mathcal{M}, t \models \varphi$,
$\mathcal{M}, s \models [\varphi] \psi$	<i>iff</i>	$\mathcal{M}, s \models \varphi$ implies $\mathcal{M} _\varphi, s \models \psi$

where T^* is the reflexive transitive closure of $\bigcup_{a \in G} \mathcal{R}(a)$ and the model $\mathcal{M}|_\varphi = \langle \mathcal{S}', \mathcal{R}', \mathcal{V}' \rangle$ is defined by restricting \mathcal{M} to those worlds where φ holds. So $\mathcal{S}' = \llbracket \varphi \rrbracket$, $\mathcal{R}' = \mathcal{R}(a) \cap \llbracket \varphi \rrbracket^2$ and $\mathcal{V}'(p) = \mathcal{V}(p) \cap \llbracket \varphi \rrbracket$, where $\llbracket \varphi \rrbracket = \{s \in \mathcal{S} \mid \mathcal{M}, s \models \varphi\}$.

When $\mathcal{M}, s \models \varphi$ for all $s \in \mathcal{D}(\mathcal{M})$, we write $\mathcal{M} \models \varphi$. If $\mathcal{M} \models \varphi$ for all Kripke models \mathcal{M} , we say that φ is valid. If for formula φ there is a state (\mathcal{M}, s) such that $\mathcal{M}, s \models \varphi$, we say that φ is satisfied in (\mathcal{M}, s) .

Kripke semantics makes our epistemic logic *intensional*, in the sense that we give up the property of extensionality, which dictates that the truth of a formula is completely determined by the truth of its sub-formulae.

6.1.2 A Classical Example

Now we present a classical example of epistemic logics, the Three Wise Men problem [67], which captures the knowledge and the reasoning process of a typical agent in a multi-agent environment. We take the following problem specification as in [37].

*There are three wise men. It is common knowledge that there are three red hats and two white hats. The king puts a hat on each of them so that they cannot see their own hat, and asks each one **in turn** if they know the colour of the their hats. Suppose the first man says he does not know; then the second says he does not know either. It follows that the third man must be able to tell his hat is red.*

We can formalize the problem as follows, very similar to the formalization in [60], only adding public announcement features. Let p_i be the proposition meaning that the wise man i has a red hat; so $\neg p_i$ means that he has a white hat. Let Γ be the set of formulae

$$\begin{aligned} &\{C(p_1 \vee p_2 \vee p_3), \\ &C(p_1 \rightarrow K_2 p_1), C(\neg p_1 \rightarrow K_2 \neg p_1), C(p_1 \rightarrow K_3 p_1), C(\neg p_1 \rightarrow K_3 \neg p_1), \\ &C(p_2 \rightarrow K_1 p_2), C(\neg p_2 \rightarrow K_1 \neg p_2), C(p_2 \rightarrow K_3 p_2), C(\neg p_2 \rightarrow K_3 \neg p_2), \\ &C(p_3 \rightarrow K_1 p_3), C(\neg p_3 \rightarrow K_1 \neg p_3), C(p_3 \rightarrow K_2 p_3), C(\neg p_3 \rightarrow K_2 \neg p_3)\} \end{aligned}$$

We want to prove

$$\Gamma \vdash [\neg (K_1 p_1 \vee K_1 \neg p_1)][\neg (K_2 p_2 \vee K_2 \neg p_2)]K_3 p_3$$

6.1.3 Reasoning about Epistemic Logics - The Model Checking Approach

As the semantics of epistemic logic are given in Kripke structures, model checking is a natural method of verifying epistemic properties. There have been some recent results along this line of research [14, 109, 116, 90, 43, 111]. In this section, we briefly survey the three model checkers and discuss their strengths and weaknesses.

MCK (Model Checking Knowledge) [43] is a prototype model checker for temporal and knowledge specifications. It deals with the logic of knowledge and both linear and branching time using BDD based algorithms. The overall setup assumes a number of agents acting in an environment, by temporal development. This is modeled by an interpreted system where agents perform actions according to a protocol. Actions and the environment may be only partially observable at each instant in time.

MCMAS (Model Checking Multi-Agent Systems) [90] employs ordered binary decision diagrams comparable to the approach used in MCK for verification of system description and protocol properties. It allows input in terms of interpreted systems. In MCMAS the global state is represented by the tuple of the local states of the agents.

DEMO (Dynamic Epistemic MOdelling) [111] is an explicit state model checker based on a dynamic epistemic logic. It allows modeling epistemic updates, graphical display

6.1. Epistemic Logic

of Kripke structures involved (i.e., epistemic or state models, and action models that represent epistemic actions), formula evaluation in epistemic states, etc. Epistemic models are minimized under bi-simulation. As an example we present a specification of the Three Wise Men problem in DEMO as shown in Fig. 6.1.

We use three agent `a`, `b`, and `c` for the three wise men. The proposition `p0` is true if and only if agent `a` is wearing a red hat. The proposition `p1` is true if and only if agent `a` is wearing a white hat. Similarly propositions `q0`, `q1`, `r0` and `r1` are for agent `b` and `c`. The states are specified in `val` and the accessibility relation is explicitly specified in `acc`. Finally the property to verify is specified as `check`.

Though the model checking technique is advantageous over theorem proving for its automation, it has some drawbacks. For one, the system to be verified has to be fully specified even if the property only concerns with a fragment of the system. It is even worse for the case of DEMO in which all states and accessibility relations have to be manually specified. The sizes of the states and relation are exponential to the number of proposition of the system. For another, while model checking technique provides a fully automated mechanism for verifying properties of a system, it suffers from what is known as the state explosion problem. The task of representing and verifying against all possible computations of a system may not be problematic for small examples, but may become unfeasible for realistic multi-agent systems. The last and most important drawback is that these model checkers only deal with finite

```

module twm where import DEMO

  a_announce = public(K a (Neg a_knows_a))
  b_announce = public(K b (Neg b_knows_b))

  a_knows_a = Disj[K a p0, K a p1]
  a_knows_b = Disj[Disj[(Neg q0), K a q0], Disj[(Neg q1), K a q1]]
  a_knows_c = Disj[Disj[(Neg r0), K a r0], Disj[(Neg r1), K a r1]]
  b_knows_b = Disj[K b q0, K b q1]
  b_knows_a = Disj[Disj[(Neg p0), K b p0], Disj[(Neg p1), K b p1]]
  b_knows_c = Disj[Disj[(Neg r0), K b r0], Disj[(Neg r1), K b r1]]
  c_knows_c = Disj[K c r0, K c r1]
  c_knows_a = Disj[Disj[(Neg p0), K c p0], Disj[(Neg p1), K c p1]]
  c_knows_b = Disj[Disj[(Neg q0), K c q0], Disj[(Neg q1), K c q1]]
  check = isTrue ( upd ( upd twm a_announce ) b_announce ) (c_knows_c)

twm :: EpistM
twm = (Pmod [0..6] val acc [0]) where
    val = [(0,[P 0,Q 0,R 1]), (1,[P 0,Q 1,R 0]), (2,[P 0,Q 1,R 1]),
           (3,[P 1,Q 0,R 0]), (4,[P 1,Q 0,R 1]), (5,[P 1,Q 1,R 0]),
           (6,[P 1,Q 1,R 1])]
    acc = [(a,0,0),(a,0,4),(a,1,1),(a,1,5),(a,2,2),(a,2,6),(a,3,3),(a,4,0),
           (a,4,4),(a,5,1),(a,5,5),(a,6,2),(a,6,6),(b,0,0),(b,0,2),(b,1,1),
           (b,2,0),(b,2,2),(b,3,3),(b,3,5),(b,4,4),(b,4,6),(b,5,3),(b,5,5),
           (b,6,4),(b,6,6),(c,0,0),(c,1,1),(c,1,2),(c,2,1),(c,2,2),(c,3,3),
           (c,3,4),(c,4,3),(c,4,4),(c,5,5),(c,5,6),(c,6,5),(c,6,6)]

p0, p1, q0, q1, r0, r1 :: Form
p0 = Prop (P 0); p1 = Prop (P 1)
q0 = Prop (Q 0); q1 = Prop (Q 1)
r0 = Prop (R 0); r1 = Prop (R 1)
    
```

Figure 6.1: Three Wise Men problem in DEMO

6.2. Reasoning Framework

state systems, but in many cases the state space is infinite due to arbitrary number of agents involved.

Hence in this work we take a different and complementary approach. We encode the epistemic logics in an expressive specification language and perform the reasoning in a well supported theorem prover in a user-guided fashion. However for simplicity, our current framework allows only public announcement and leaves out private announcement to future work.

6.2 Reasoning Framework

The system architecture of our reasoning framework is depicted in Fig. 6.2. Based on the encoding of the logic formulae, the framework primarily consists of four components, namely *Proof Systems*, *Theorem Sets*, *Reasoning Systems* and *Reasoning Rule Sets*. A solid arrow from a component B to a component A indicates that A imports B. A dotted arrow from a component A to a component B represents dataflow from A to B.

In addition, because of the relationship between the epistemic logics, we organize the encodings for each epistemic logic in a hierarchical fashion too, as shown in Fig. 6.3. So we have in effect established a two-dimensional hierarchy – *horizontal* hierarchy among different components for a particular logic and *vertical* hierarchy

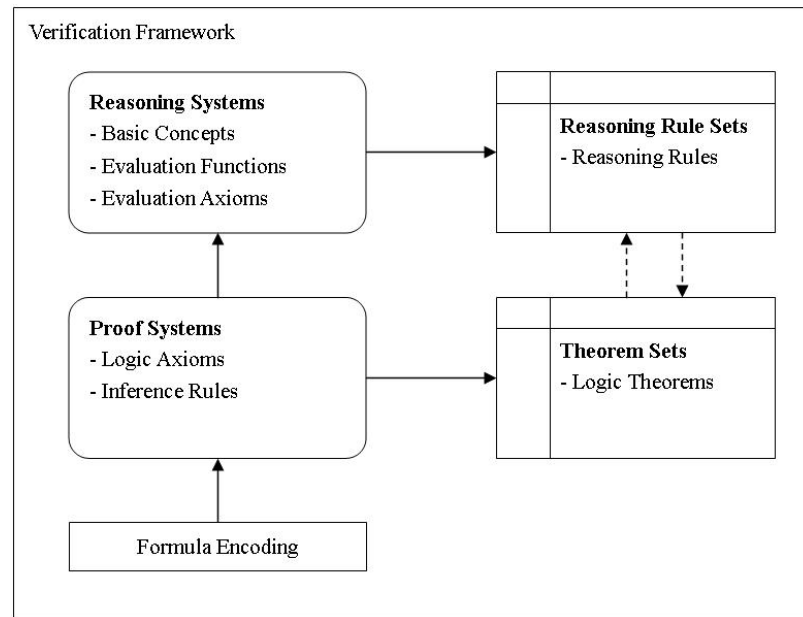


Figure 6.2: Framework Architecture

among different logics. As a result, system developers can easily select and reuse the desired system environment for specification and reasoning.

In this section, we explain the functionalities of each individual component and how they are used with each other as a system. Due to space limitations, the full PVS

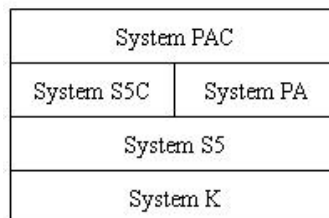


Figure 6.3: Logic Hierarchy

6.2. Reasoning Framework

specification is not completely shown, but can be found online¹.

The logic formulae are encoded using the PVS abstract datatype construct as shown below.

```
1.  palc_formula[AGENT: TYPE]: DATATYPE BEGIN
2.    ktop: ktop?
3.    kbottom: kbottom?
4.    base: base?
5.    knot(sub: palc_formula): knot?
6.    kand(left: palc_formula, right: palc_formula): kand?
7.    kor(left: palc_formula, right: palc_formula): kor?
8.    kif(left: palc_formula, right: palc_formula): kif?
9.    kiff(left: palc_formula, right: palc_formula): kiff?
10.   k(agent: AGENT, sub: palc_formula): k?
11.   e(agents: set[AGENT], sub: palc_formula): e?
12.   c(agents: set[AGENT], sub: palc_formula): c?
13.   pa(inner: palc_formula, outer: palc_formula): pa?
14. END palc_formula
```

The PVS abstract datatype mechanism is useful because it automatically generates theories containing axioms and definitions for a class of recursive datatypes. The datatype declaration simply specifies the ways a logic formula can be constructed. For example the 10th line

`k(agent: AGENT, sub: palc_formula): k?`

specifies that a formula can be constructed by using a *constructor* `k` and two arguments where the first is of type `AGENT` and the second is a `palc_formula`. `k?` is a

¹<http://www.comp.nus.edu.sg/~fengyz/PVSFramework>

recognizer for formulae constructed in this way. Hence it is later easy to assert if a formula is of a certain type. `agent` and `sub` are accessors for the arguments.

Type-checking the datatype specification automatically generates two separate PVS theory files named `palc_formula_adt.pvs` and `palc_formula_adt_reduce.pvs` which contain axioms and definitions over the logic formulae, which are very useful when proving theorems. An example is shown below. It defines what is meant by two `k?` formulae being equivalent.

```
palc_formula_k_extensionality: AXIOM FORALL (k?_var: (k?), k?_var2:(k?)):
    agent(k?_var) = agent(k?_var2) AND sub(k?_var) = sub(k?_var2)
    IMPLIES k?_var = k?_var2;
```

6.2.1 Proof Systems

A logic is a set of formulae. An axiomatization is a syntactic way to specify a logic: it gives a core set of formulae, called axioms, and inference rules, from which all other valid formulae in the logic can be derived. Now we formalize the definition of derivations and theorems.

Definition 6.2.1 *Let \mathbf{X} be an axiomatization of an arbitrary logic with axioms A_1, \dots, A_n and derivation rules R_1, \dots, R_m . Then a derivation for a formula φ within \mathbf{X} is a finite sequence $\varphi_1, \dots, \varphi_k$ of formulae such that $\varphi_k = \varphi$ and every φ_i in the sequence is an instance of one of the axioms A_1, \dots, A_n , or the result of the application of one of the rules R_1, \dots, R_m to some formulae φ_j where $j < i$. If there is a derivation for*

6.2. Reasoning Framework

φ in \mathbf{X} , we write $\mathbf{X} \vdash \varphi$, or if the system \mathbf{X} is clear from the context, we just write $\vdash \varphi$. We also say that φ is a theorem of \mathbf{X} .

We want to construct a framework that can be used to reason about an arbitrary model. On the one hand, we need to be sure that our system is complete; all valid formulae can be proved. On the other hand, we want the base model to be as concise as possible. Hence we encode the axiomatizations, obtaining completeness at minimal cost.

In our architecture, the *Proof Systems* component captures the axiomatizations of various epistemic logics. Because some axiomatizations extend some others, we utilize the reuse facilities of PVS by storing each sub-component in a separate theory and using the `IMPORTING` clause to capture the extensional relationship. In effect, we construct a hierarchy of proof systems following the relationships among various logics.

System **K**

The basic axiomatization **K** for the epistemic logic K is comprised of the axioms A_1 and A_2 , together with the derivation rules R_1 and R_2 as given below.

The encoding of the axiomatization for the logic K is shown below.

A_1	φ	φ is any propositional tautology
A_2	$(K_a\varphi \wedge K_a(\varphi \rightarrow \psi)) \rightarrow K_a\psi$	K -axiom
R_1	$\vdash \varphi, \vdash \varphi \rightarrow \psi \Rightarrow \vdash \psi$	Modus Ponens
R_2	$\vdash \varphi \Rightarrow \vdash K_a\varphi$	K -Necessitation

Figure 6.4: Axiomatization **K**

```

systemK[AGENT: TYPE] : THEORY BEGIN
  IMPORTING palc_formula_adt[AGENT]
  derives: [palc_formula -> bool]
  tautology: [palc_formula -> bool]
  pro_tauto: AXIOM FORALL (p: palc_formula): tautology(p) IMPLIES derives(p)
  k_axiom : AXIOM FORALL (p1,p2: palc_formula),(a: AGENT):
    derives(kif(kand(k(a,p1),k(a,kif(p1,p2))),k(a,p2)))
  modus_ponens : AXIOM FORALL (p1,p2: palc_formula):
    derives(p1) AND derives(kif(p1,p2)) IMPLIES derives(p2)
  k_necessitation: AXIOM FORALL (p: palc_formula),(a: AGENT):
    derives(p) IMPLIES derives(k(a,p))
END systemK

```

In this encoding, `derives` is defined as a function from a `palc_formula` to a boolean value. Formally $derives(\varphi)$ holds if and only if φ is a theorem in the system, that is $\vdash \varphi$. The two logic axioms and the two derivation rules are specified as PVS axioms.

With the axioms and derivation rule, the proof systems can be proved to be sound and complete. We adopt the soundness and completeness results of [110] and omit the proof here.

6.2. Reasoning Framework

As we have discussed in Section 6.1, the logic **PAL** is as expressive as the logic $S5$. Furthermore it has been shown that the computation complexity of **PAL** coincide with that of epistemic logic $S5$ [74]. But the logical language of public announcement is a convenient specification tool to express this particular sort of dynamics of multi-agent systems. In fact it has been shown that there are properties that can be expressed exponentially more succinctly in **PAL** than in $S5$. Hence in our framework, **PAL** is *not* encoded as $S5$. Being an essentially syntactical extension to $S5$, the proof system of **PAL** is encoded by extending that of $S5$ with additional axioms based on the equivalence. The purpose of doing so, like the purpose of having **PAL** with the existence of $S5$, is to provide the users with flexibility and convenience to specify the systems. Furthermore it establish a higher level of reasoning by providing axioms and theorems on the public announcement operator rather than simply the K operator.

System $S5$

As we have discussed in Section 6.1, the logic $S5$ extends the logic K by adding three axioms, namely Truth axiom, positive introspection and negative introspection. These three axioms are captured in the axiomatization of logic $S5$ as shown in Fig. 6.5.

The corresponding PVS encoding is shown below. As illustrated, we only need to import the theory `systemK` and add the three new axioms.

A_1	φ	φ is any propositional tautology
A_2	$(K_a\varphi \wedge K_a(\varphi \rightarrow \psi)) \rightarrow K_a\psi$	K -axiom
A_3	$K_a\varphi \rightarrow \varphi$	K -veridicality
A_4	$K_a\varphi \rightarrow K_aK_a\varphi$	positive introspection
A_5	$\neg K_a\varphi \rightarrow K_a\neg K_a\varphi$	negative introspection
R_1	$\vdash \varphi, \vdash \varphi \rightarrow \psi \Rightarrow \vdash \psi$	Modus Ponens
R_2	$\vdash \varphi \Rightarrow \vdash K_a\varphi$	K -Necessitation

Figure 6.5: Axiomatization **S5**

```

systemS5[AGENT : TYPE] : THEORY BEGIN
  IMPORTING systemK[AGENT]
  k_veridicality : AXIOM FORALL (p:palc_formula),(a:AGENT) :
    derives(kif(k(a,p),p))
  pos_intro : AXIOM FORALL (p:palc_formula),(a:AGENT) :
    derives(kif(k(a,p),k(a,k(a,p))))
  neg_intro : AXIOM FORALL (p:palc_formula),(a:AGENT) :
    derives(kif(knot(k(a,p)),k(a,knot(k(a,p)))))
END systemS5

```

System S5C

The logic $S5C$ extends the logic $S5$ by adding the notion of group knowledge. The axiomatization **S5C** of $S5C$ extends **S5** by adding four axioms and one derivation rule as shown in Fig. 6.6.

Axiom A_6 states that common knowledge is veridical. Axiom A_7 ensures all re-

6.2. Reasoning Framework

A_1	φ	φ is any propositional tautology
A_2	$(K_a\varphi \wedge K_a(\varphi \rightarrow \psi)) \rightarrow K_a\psi$	K -axiom
A_3	$K_a\varphi \rightarrow \varphi$	K -veridicality
A_4	$K_a\varphi \rightarrow K_aK_a\varphi$	positive introspection
A_5	$\neg K_a\varphi \rightarrow K_a\neg K_a\varphi$	negative introspection
A_6	$C_B\varphi \rightarrow \varphi$	C -veridicality
A_7	$C_B\varphi \rightarrow E_B C_B\varphi$	iteration
A_8	$(C_B\varphi \wedge C_B(\varphi \rightarrow \psi)) \rightarrow C_B\psi$	normality
A_9	$C_B(\varphi \rightarrow E_B\varphi) \rightarrow (\varphi \rightarrow C_B\varphi)$	induction
R_1	$\vdash \varphi, \vdash \varphi \rightarrow \psi \Rightarrow \vdash \psi$	Modus Ponens
R_2	$\vdash \varphi \Rightarrow \vdash K_a\varphi$	K -Necessitation
R_3	$\vdash \varphi \Rightarrow \vdash C_B\varphi$	C -Necessitation

Figure 6.6: Axiomatization **S5C**

strictions of the definition of common knowledge, that is for every $k \in \mathbb{N}$, we have $\vdash C_B \varphi \rightarrow E_B^k \varphi$. Axiom A_8 and derivation rule R_3 ensures that C is a normal modal operator. Finally A_9 explains how one can derive that φ is a common knowledge.

The corresponding PVS encoding is shown below.

```

systemS5C[AGENT : TYPE] : THEORY BEGIN
  IMPORTING systemS5[AGENT]
  c_veridicality : AXIOM FORALL (p:palc_formula),(g:set[AGENT]) :
    derives(kif(c(g,p),p))
  iteration : AXIOM FORALL (p:palc_formula),(g:set[AGENT]) :
    derives(kif(c(g,p),e(g,c(g,p))))
  normality : AXIOM FORALL (p1,p2:palc_formula),(g:set[AGENT]) :
    derives(kif(kand(c(g,p1),c(g,kif(p1,p2))),c(g,p2)))
  induction : AXIOM FORALL (p:palc_formula),(g:set[AGENT]) :
    derives(kif(c(g,kif(p,e(g,p))),kif(p,c(g,p))))
  c_necessitation : AXIOM FORALL (p:palc_formula),(g:set[AGENT]) :
    derives(p) IMPLIES derives(c(g,p))
END systemS5C

```

System PA

The logic **PAL** extends the logic $S5$ by incorporating the public announcement operator. As shown in Fig. 6.7, the axiomatization **PA** of **PAL** extends that of $S5$ by adding the announcement elimination axioms(A_6 through A_9), an announcement composition axiom(A_{10}) and a derivation rule for announcement necessitation(R_3). The announcement elimination axioms reduces a formula with public announcement operators to one without them. In other words, when no common knowledge is in-

6.2. Reasoning Framework

A_1	φ	φ is any propositional tautology
A_2	$(K_a\varphi \wedge K_a(\varphi \rightarrow \psi)) \rightarrow K_a\psi$	K -axiom
A_3	$K_a\varphi \rightarrow \varphi$	K -veridicality
A_4	$K_a\varphi \rightarrow K_aK_a\varphi$	positive introspection
A_5	$\neg K_a\varphi \rightarrow K_a\neg K_a\varphi$	negative introspection
A_6	$[\varphi]p \leftrightarrow (\varphi \rightarrow p)$	atomic permanence
A_7	$[\varphi]\neg\psi \leftrightarrow (\varphi \rightarrow \neg[\varphi]\psi)$	announcement and negation
A_8	$[\varphi](\psi \wedge \chi) \leftrightarrow [\varphi]\psi \wedge [\varphi]\chi$	announcement and conjunction
A_9	$[\varphi]K_a\psi \leftrightarrow (\varphi \rightarrow K_a[\varphi]\psi)$	announcement and knowledge
A_{10}	$[\varphi][\psi]\chi \leftrightarrow [\varphi \wedge [\psi]\chi]$	announcement composition
R_1	$\vdash \varphi, \vdash \varphi \rightarrow \psi \Rightarrow \vdash \psi$	Modus Ponens
R_2	$\vdash \varphi \Rightarrow \vdash K_a\varphi$	K -Necessitation
R_3	$\vdash \varphi \Rightarrow \vdash [\psi]\varphi$	Announcement Necessitation

Figure 6.7: Axiomatization **PA**

volved, the public announcement operator does not add expressiveness. However this is not true when common knowledge is involved. We shall discuss more in Section [6.2.1](#).

The corresponding PVS encoding is shown below.

```

systemPA[AGENT : TYPE] : THEORY BEGIN
  IMPORTING systemS5[AGENT]
  atomic_permanance : AXIOM FORALL (f:palc_formula),(p:(base?)) :
    derives(kiff(pa(f,p),kif(f,p)))
  announce_neg : AXIOM FORALL (f:palc_formula),(p:(knot?)) :
    derives(kiff(pa(f,p),kif(f,knot(pa(f,sub(p))))))
  announce_con : AXIOM FORALL (f:palc_formula),(p:(kand?)) :
    derives(kiff(pa(f,p),kand(pa(f,left(p)),pa(f,right(p)))))
  announce_k : AXIOM FORALL (f:palc_formula),(p:(k?)) :
    derives(kiff(pa(f,p),kif(f,k(agent(p)),pa(f,sub(p)))))
  announce_compo : AXIOM FORALL (f:palc_formula),(p:(pa?)) :
    derives(kiff(pa(f,p),pa(kand(f,pa(f,inner(p))),outer(p))))
  announce_necessitation : AXIOM FORALL (p1,p2:palc_formula) :
    derives(p2) IMPLIES derives(pa(p1,p2))
END systemPA

```

System PAC

The axiomatization **PAC** of the logic **PAL-C** extends both **S5C** (for common knowledge aspect) and **PA** (for announcement aspect). Furthermore it adds a derivation rule for announcement and common knowledge. The full axiomatization is shown in Fig. 6.8. We skip the discussions of axiomatization of the logic *S5*, *S5C* and **PAL**. To illustrate the advantage of using the PVS reuse mechanism, we consider the axiomatization **PAC** (not shown here) of the logic **PAL-C** and its corresponding encoding which is shown below. **PAC** has fourteen axioms and 5 derivation rules, but the corresponding encoding has only one PVS axiom for one of the derivation rules.

The corresponding PVS encoding of **PAC** is shown below. As we have discussed the

6.2. Reasoning Framework

A_1	φ	φ is any propositional tautology
A_2	$(K_a\varphi \wedge K_a(\varphi \rightarrow \psi)) \rightarrow K_a\psi$	K -axiom
A_3	$K_a\varphi \rightarrow \varphi$	K -veridicality
A_4	$K_a\varphi \rightarrow K_aK_a\varphi$	positive introspection
A_5	$\neg K_a\varphi \rightarrow K_a\neg K_a\varphi$	negative introspection
A_6	$C_B\varphi \rightarrow \varphi$	C -veridicality
A_7	$C_B\varphi \rightarrow E_B C_B\varphi$	iteration
A_8	$(C_B\varphi \wedge C_B(\varphi \rightarrow \psi)) \rightarrow C_B\psi$	normality
A_9	$C_B(\varphi \rightarrow E_B\varphi) \rightarrow (\varphi \rightarrow C_B\varphi)$	induction
A_{10}	$[\varphi]p \leftrightarrow (\varphi \rightarrow p)$	atomic permanence
A_{11}	$[\varphi]\neg\psi \leftrightarrow (\varphi \rightarrow \neg[\varphi]\psi)$	announcement and negation
A_{12}	$[\varphi](\psi \wedge \chi) \leftrightarrow [\varphi]\psi \wedge [\varphi]\chi$	announcement and conjunction
A_{13}	$[\varphi]K_a\psi \leftrightarrow (\varphi \rightarrow K_a[\varphi]\psi)$	announcement and knowledge
A_{14}	$[\varphi][\psi]\chi \leftrightarrow [\varphi \wedge \psi]\chi$	announcement composition
R_1	$\vdash \varphi, \vdash \varphi \rightarrow \psi \Rightarrow \vdash \psi$	Modus Ponens
R_2	$\vdash \varphi \Rightarrow \vdash K_a\varphi$	K -Necessitation
R_3	$\vdash \varphi \Rightarrow \vdash C_B\varphi$	C -Necessitation
R_4	$\vdash \varphi \Rightarrow \vdash [\psi]\varphi$	Announcement Necessitation
R_5	$\vdash \chi \rightarrow [\varphi]\psi, \vdash \chi \wedge \varphi \rightarrow E_B\chi$ $\Rightarrow \vdash \chi \rightarrow [\varphi]C_B\psi$	Announcement and Common Knowledge

Figure 6.8: Axiomatization **PAC**

encoding of **PAC** imports both **S5C** and **PA**.

```

systemPAC[AGENT : TYPE] : THEORY BEGIN
  IMPORTING systemS5C[AGENT]
  IMPORTING systemPA[AGENT]
  announce_c : AXIOM FORALL (p1,p2,p3:palc_formula),(g:set[AGENT]) :
    derives(kif(p1,pa(p2,p3))) AND derives(kif(kand(p1,p2),e(g,p3)))
    IMPLIES derives(kif(p1,pa(p2,c(g,p3))))
END systemPAC

```

6.2.2 Theorem Sets

This component contains a set of theorem sets, one for each of the proof systems. Each theorem set contains the theorems that have been proved in the corresponding proof system. More formally, for all formulae φ in the theorem set $derives(\varphi)$ evaluates to true. In other words, these theorems can be applied to any arbitrary model expressed in the logic. The importing relationships among the theorem sets are the same. Such structure makes the access to a particular logic with its proof system and theorems easier. It should be noted that, although the proof systems are complete, these theorem sets are by no means complete. It initially contains some basic and commonly used theorems such as the axioms in the axiomatization. These theorems can be used (and hence do not need to be proved again) with the derivation rules for proving new theorems which are then added back into the theorem set. Therefore the size of the theorem set grows with the use of the system. It should be emphasized that the proof systems being complete ensures that all valid formulae can be proved as theorems.

6.2. Reasoning Framework

6.2.3 Reasoning Systems

As compared with the proof systems which aim at proving general theorems at logic level, the reasoning systems build an environment for reasoning about concrete object-level models. This component evaluates a formula on a given model. We specify the reasoning system for logic K below.

```
reasonerK : THEORY BEGIN
  Agent : TYPE
  IMPORTING systemK[Agent]
  Knowledge : TYPE = palc_formula
  eval : [Knowledge -> bool]
  knot_ax : AXIOM FORALL (k:Knowledge):
    eval(knot(k)) IMPLIES NOT eval(k)
  kand_ax : AXIOM FORALL (k1,k2:Knowledge):
    eval(kand(k1,k2)) IMPLIES (eval(k1) AND eval(k2))
  kor_ax : AXIOM FORALL (k1,k2:Knowledge):
    eval(kor(k1,k2)) IMPLIES (eval(k1) OR eval(k2))
  kif_ax : AXIOM FORALL (k1,k2:Knowledge):
    eval(kif(k1,k2)) IMPLIES (eval(k1) IMPLIES eval(k2))
  kiff_ax : AXIOM FORALL (k1,k2:Knowledge):
    eval(kiff(k1,k2)) IMPLIES (eval(k1) IFF eval(k2))
END reasonerK
```

Agent is defined as an uninterpreted type and is passed down to the proof systems as a type parameter. *Knowledge* is defined as a type equivalent to logic formulae. Given a model every piece of knowledge should have a truth value at any time. Therefore we define a *evaluation function* `eval`. More formally, for a formula φ of a given model, $eval(\varphi)$ holds if and only if the formula φ evaluates to true in the model. We have

defined five logical connectives for knowledge corresponding to the logical negation, conjunction, disjunction, implication and equivalence. The advantage of doing this is that we can easily compose new knowledge from existing ones. As a result we need to define a set of evaluation axioms which map the logical connectives for knowledge to their logic counterparts straightforwardly.

We maintain the hierarchical structure of the whole verification framework by specifying a reasoning system for each of the logics. However the reasoning systems for different logics do not differ a lot because much of the difference is reflected in the underlying proof systems and theorem sets. We still specify the reasoning systems in separate theories for consistency. The reasoning systems of *S5* and *PAL* are the same as that of *K*. The reasoning system of *PAL-C* is the same as that of *S5C*. So we only describe the reasoning system for *S5C* below.

```

reasonerS5C: THEORY BEGIN
  IMPORTING reasonerS5, systemS5C[Agent]
  e_ax: AXIOM FORALL (g: set[Agent]),(k1: Knowledge):
    eval(e(g,k1)) IFF FORALL (a: Agent): member(a,g) IMPLIES eval(k(a,k1))
  c_ax: AXIOM FORALL (g: set[Agent]),(k1: Knowledge):
    eval(c(g,k1)) IFF eval(e(g,k1)) AND eval(c(g,e(g,k1)))
END reasonerK

```

We define evaluation axioms for modal connectives for shared knowledge and common knowledge. As we have discussed earlier, *C* is in fact an infinite conjunction of *E*. As PVS only allows finite conjunctions we model the evaluation function for the common knowledge connective using a recursive definition. During the reasoning process we

6.2. Reasoning Framework

can choose the extent to which we expand the axiom.

6.2.4 Reasoning Rule Sets

Definitions and evaluation functions alone are not sufficient to prove the properties efficiently. The last component of the framework is the *Reasoning Rule Sets*. They are encoded based on the reasoning systems. In other words, they are applied when reasoning about actual models. Therefore their aims are not to make the system complete, but to achieve a higher degree of automation by abstracting certain amount of underlying model from the reasoning process. The reasoning rule sets are constructed hierarchically, similar to the other three components. Each reasoning rule set of a logic initially contains the (non-inherited) axioms from the corresponding proof systems. For example the encoding of the reasoning rule set for *S5* is shown below.

```
reasoningRuleS5: THEORY BEGIN
  IMPORTING reasonerS5, reasoningRuleK
  Truth : THEOREM FORALL (k:Knowledge),(a:Agent): eval(K(a,k)) IMPLIES eval(k)
  Positive_Introspection : THEOREM FORALL (k:Knowledge),(a:Agent):
    eval(K(a,k)) IMPLIES eval(K(a,(K(a,k))))
  Negative_Introspection : THEOREM FORALL (k:Knowledge),(a:Agent):
    eval(knot(K(a,k))) IMPLIES eval(K(a,(knot(K(a,k)))))
END reasoningRuleS5
```

The way of encoding for the reasoning rules is slightly different from the corresponding axiom in the proof system, mainly because they are used for different purposes. The

ones in proof systems are for deriving other theorems whereas the ones here are applied to a particular model for evaluation of formulae. Hence there are two ways to construct the reasoning rules. Firstly new reasoning rules can be derived from existing ones. Secondly new reasoning rules for a logic can be obtained by translating theorems from the corresponding theorem set. Theorems in the theorem set are of the form $\text{FORALL } Q: \text{derives}(F)$ where Q is a set of bound variables and F is formula in the corresponding logic. We do not change the quantifier or the bound variables and translate only the quantified formula. The translated formula is $\text{FORALL } Q: F'$ where F' uses the evaluation function `eval`. We have implemented a simple translation program which works recursively on the propositional structure of the formula. The algorithm takes in such theorems as input and produces the corresponding reasoning rule as given in Algorithm 5.

6.2.5 Framework Workflow

Having described the components of the verification framework, we explain the framework methodology as depicted in Fig. 6.9.

When the framework is first used, the theorem sets and reasoning rule sets contain the initial theorems and reasoning rules, as discussed in the Section 6.2.2 and 6.2.4. Subsequently,

6.2. Reasoning Framework

Require: FORALL Q: $\text{derives}(F)$ is an encoded theorem in the theorem set

Ensure: FORALL Q: F' is the corresponding encoded reasoning rule

```
1: if F is of the form  $\text{knot}(F1)$  then
2:   return NOT  $\text{translate}(F1)$ 
3: else if F is of the form  $\text{kand}(F1, F2)$  then
4:   return  $\text{translate}(F1)$  AND  $\text{translate}(F2)$ 
5: else if F is of the form  $\text{kor}(F1, F2)$  then
6:   return  $\text{translate}(F1)$  OR  $\text{translate}(F2)$ 
7: else if F is of the form  $\text{kif}(F1, F2)$  then
8:   return  $\text{translate}(F1)$  IMPLIES  $\text{translate}(F2)$ 
9: else if F is of the form  $\text{kiff}(F1, F2)$  then
10:  return  $\text{translate}(F1)$  IFF  $\text{translate}(F2)$ 
11: else
12:  return  $\text{eval}(F)$ 
13: end if
```

Algorithm 5: $F' = \text{translate}(F)$

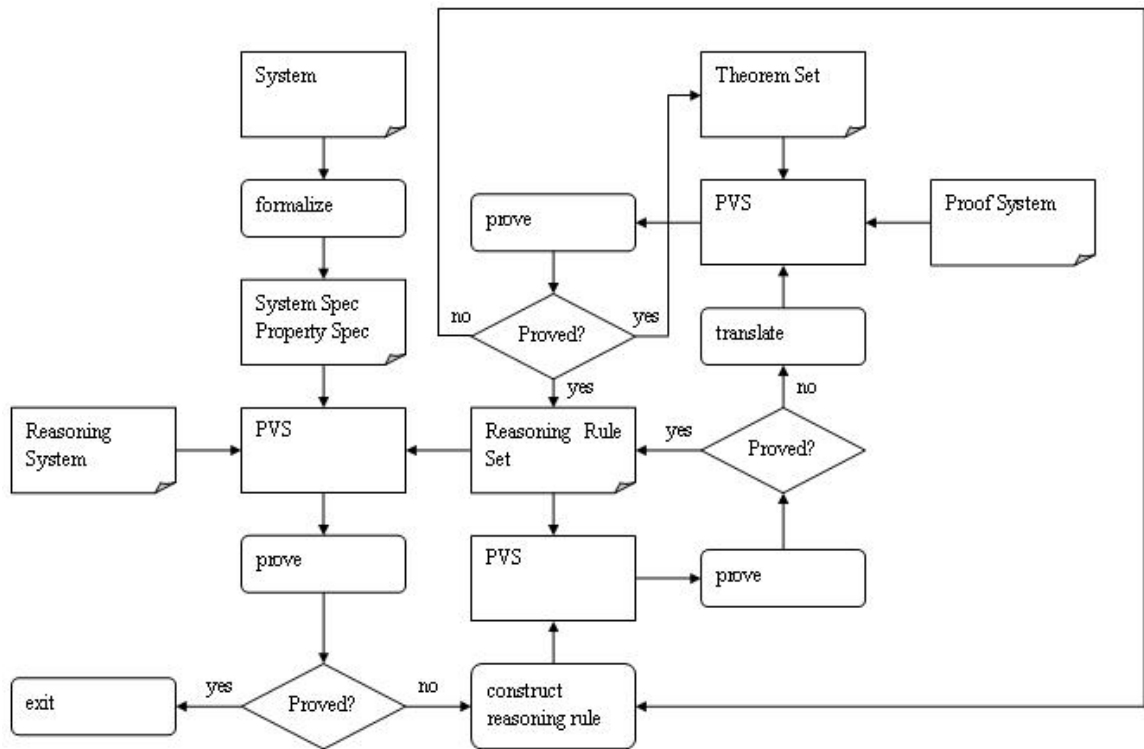


Figure 6.9: Framework Workflow

6.2. Reasoning Framework

1. Given a *system*, formalize it using an appropriate epistemic logic to produce the *system specification*. Specify the property to be proved about system.
2. With the appropriate *reasoning system* and *reasoning rule set*, try to prove the property. If the property is proved, exit.
3. Construct a reasoning rule which may help prove the property.
4. Try to prove the new reasoning rule in PVS based on the existing reasoning rules in the *reasoning rule set*. If the rule is proved, add it to the *reasoning rule set* and go to step 2. Otherwise translate it into theorem format.
5. Try to prove the translated theorem in *proof system* based on existing theorems in the *theorem set*. If successful, add the theorem to the theorem set and the reasoning rule to the reasoning rule set and go to step 2. Otherwise go to step 3.

In effect, the user keeps trying to prove the property with reasoning rules which can be proved by using either the existing reasoning rules or the existing theorems. In the process the reasoning rule set and the theorem set are incrementally constructed. Thus multiple instances of the prover can be invoked at the same time.

6.3 Examples

As a demonstration of how to use the framework, we now illustrate with the running example, the Three Wise Men problem. We also highlight the advantage of using our approach by extending the number of agents from three to an arbitrary number which is beyond the capabilities of state-of-the-arts epistemic model checkers. The procedures follow the work flow explained in Section 6.2.5.

6.3.1 Formalizing the System

We can formalize the problem in our reasoning framework as shown below. As we can see the specification is a direct translation of the model in Section 6.1.2.

6.3. Examples

```
twm: THEORY BEGIN
  IMPORTING reasonerPAC
  IMPORTING reasoningRulePAC
  m1, m2, m3: Agent
  p1, p2, p3: (base?)
  g: set[Agent] = {a : Agent | a = m1 OR a = m2 OR a = m3}
  init: AXIOM eval(c(g,kor(kor(p1,p2),p3)))
  init_m1: AXIOM
    eval(c(g,kif(p1,k(m2,p1)))) AND eval(c(g,kif(knot(p1),k(m2,knot(p1))))) AND
    eval(c(g,kif(p1,k(m3,p1)))) AND eval(c(g,kif(knot(p1),k(m3,knot(p1)))))
  init_m2: AXIOM
    eval(c(g,kif(p2,k(m1,p2)))) AND eval(c(g,kif(knot(p2),k(m1,knot(p2))))) AND
    eval(c(g,kif(p2,k(m3,p2)))) AND eval(c(g,kif(knot(p2),k(m3,knot(p2)))))
  init_m3: AXIOM
    eval(c(g,kif(p3,k(m2,p3)))) AND eval(c(g,kif(knot(p3),k(m2,knot(p3))))) AND
    eval(c(g,kif(p3,k(m1,p3)))) AND eval(c(g,kif(knot(p3),k(m1,knot(p3)))))
  conclude: THEOREM
    eval(pa(knot(kor(k(m1,p1),k(m1,knot(p1)))),
      pa(knot(kor(k(m2,p2),k(m2,knot(p2)))),k(m3,p3))))
END twm
```

As we are going to use the logic PAL-C for reasoning, we import the reasoning system `reasonerPAC` and the reasoning rule set `reasoningRulePAC`. We first define the three wise men as agents and define three pieces of ground knowledge `p1`, `p2` and `p3` each of which corresponds to the proposition meaning that the *i*-th man is wearing a red hat. Then the negation of them means that the *i*-th man is wearing a white hat. For ease of specifying the system we define the group of agents `g` containing the three agents. The fact that initially it is commonly known that at least one of them is wearing a red hat (implied from the fact that there are two white hats and three red

hats) is captured by the axiom `init`. The fact that the colour of one's hat is known to the others is captured by the three axioms `init_1`, `init_2` and `init_3`. Then `conclude` is the property we want to prove, that is, after the first two men declared their ignorance about the colour of their hat, the third knows his hat is red. Formally this is

$$[\neg (K_1 p_1 \vee K_1 \neg p_1)][\neg (K_2 p_2 \vee K_2 \neg p_2)]K_3 p_3$$

6.3.2 Constructing and Proving Reasoning Rules

The PVS prover cannot prove the property automatically. Hence according to the workflow we need to construct some reasoning rules. There are 9 reasoning rules that we need for proving the property. Some of these reasoning rules are from the original reasoning rule set, while others cannot be proved directly from other reasoning rules. Therefore we input them into the translation program to obtain the corresponding theorems and then prove them in the proof system **PAC**. Due to space limitation, simple proofs for the lemmas are omitted.

Let $\varphi, \varphi_1, \varphi_2, \psi$ and ω be arbitrary formulae, p a ground proposition, B an arbitrary set of agents, and a an agent in B then the following lemmas hold.

Lemma 6.3.1 $\vdash C_B \varphi \rightarrow K_a \varphi \wedge C_B K_a \varphi$.

Lemma 6.3.2 $\vdash [\varphi \wedge [\varphi]\psi]\omega \rightarrow [\varphi][\psi]\omega$.

6.3. Examples

Lemma 6.3.3 $\vdash (\varphi \rightarrow K_a[\varphi]\psi) \rightarrow [\varphi]K_a\psi$.

Lemma 6.3.4 $\vdash (\varphi \rightarrow p) \rightarrow [\varphi]p$.

Lemma 6.3.5 $\vdash \varphi \wedge [\varphi]\neg\psi \rightarrow \neg[\varphi]\psi$.

Lemma 6.3.6 $\vdash \varphi \wedge [\varphi]K_a\psi \rightarrow K_a[\varphi]\psi$.

Lemma 6.3.7 $\vdash [\varphi](\psi \wedge \omega) \rightarrow [\varphi]\psi \wedge [\varphi]\omega$.

Lemma 6.3.8 $\vdash (\varphi_1 \rightarrow \varphi_2) \rightarrow (K_a\varphi_1 \rightarrow K_a\varphi_2)$.

Lemma 6.3.9 $\vdash K_a\varphi_1 \rightarrow (K_a\varphi_2 \rightarrow K_a(\varphi_1 \wedge \varphi_2))$.

With the reasoning rules, it is now sufficient to prove the property. Due to space limitation, we would not show the proof details such as proof commands used in this proof. The simplified proof tree is shown in Fig. 6.10. To improve readability and save space in the proof tree, $\neg K_1p_1 \wedge \neg K_1\neg p_1$ is renamed to φ_1 and $\neg K_2p_2 \wedge \neg K_2\neg p_2$ to φ_2 in some parts of the proof without loss of correctness.

6.3.3 Proof Strategies

PVS proof strategies provide an accessible means of increasing the degree of automation available to PVS users. A proof strategy is intended to capture patterns of

$$\begin{array}{c}
 \frac{p_1 \vee p_2 \vee p_3, \neg p_3, \neg p_2 \vdash p_1,}{K_1(p_1 \vee p_2 \vee p_3), K_1 \neg p_3, K_1 \neg p_2 \vdash K_1 p_1,} \text{[k_elim]} \\
 \frac{K_1(p_1 \vee p_2 \vee p_3), \neg p_i \rightarrow K_j \neg p_i, \neg p_3, \neg p_2 \vdash K_1 p_1,}{K_1(p_1 \vee p_2 \vee p_3), \neg p_i \rightarrow K_j \neg p_i, \neg K_1 p_1, \neg K_1 \neg p_1, \neg p_3 \vdash p_2} \\
 \frac{K_1(p_1 \vee p_2 \vee p_3), \neg p_i \rightarrow K_j \neg p_i, \neg K_1 p_1 \wedge \neg K_1 \neg p_1, \neg p_3 \vdash p_2}{K_1(p_1 \vee p_2 \vee p_3), \neg p_i \rightarrow K_j \neg p_i, \neg p_3 \vdash (\neg K_1 p_1 \wedge \neg K_1 \neg p_1) \rightarrow p_2} \text{[Lem. 4]} \\
 \frac{K_1(p_1 \vee p_2 \vee p_3), \neg p_i \rightarrow K_j \neg p_i, \neg p_3 \vdash [\neg K_1 p_1 \wedge \neg K_1 \neg p_1] p_2}{K_2 K_1(p_1 \vee p_2 \vee p_3), K_2(\neg p_i \rightarrow K_j \neg p_i), K_2 \neg p_3 \vdash K_2[\neg K_1 p_1 \wedge \neg K_1 \neg p_1] p_2} \text{[k_elim]} \\
 \frac{K_2 K_1(p_1 \vee p_2 \vee p_3), K_2(\neg p_i \rightarrow K_j \neg p_i), \neg p_i \rightarrow K_j \neg p_i, \neg p_3 \vdash K_2[\neg K_1 p_1 \wedge \neg K_1 \neg p_1] p_2}{K_2 K_1(p_1 \vee p_2 \vee p_3), K_2(\neg p_i \rightarrow K_j \neg p_i), \neg p_i \rightarrow K_j \neg p_i, \neg K_2[\neg K_1 p_1 \wedge \neg K_1 \neg p_1] p_2 \vdash p_3} \\
 \frac{K_2 K_1(p_1 \vee p_2 \vee p_3), K_2(\neg p_i \rightarrow K_j \neg p_i), \neg p_i \rightarrow K_j \neg p_i, \varphi_1, \neg K_2[\varphi_1] p_2, \neg K_2[\varphi_1] \neg p_2 \vdash p_3}{K_2 K_1(p_1 \vee p_2 \vee p_3), K_2(\neg p_i \rightarrow K_j \neg p_i), \neg p_i \rightarrow K_j \neg p_i, \varphi_1, [\varphi_1] \neg K_2 p_2, [\varphi_1] \neg K_2 \neg p_2 \vdash p_3} \text{[Lem. 5,6]} \\
 \frac{K_2 K_1(p_1 \vee p_2 \vee p_3), K_2(\neg p_i \rightarrow K_j \neg p_i), \neg p_i \rightarrow K_j \neg p_i, \varphi_1, [\varphi_1](\varphi_2) \vdash p_3}{K_2 K_1(p_1 \vee p_2 \vee p_3), K_2(\neg p_i \rightarrow K_j \neg p_i), \neg p_i \rightarrow K_j \neg p_i, \varphi_1 \wedge [\varphi_1] \varphi_2 \vdash p_3} \text{[Lem. 7]} \\
 \frac{K_2 K_1(p_1 \vee p_2 \vee p_3), K_2(\neg p_i \rightarrow K_j \neg p_i), \neg p_i \rightarrow K_j \neg p_i, \varphi_1 \wedge [\varphi_1] \varphi_2 \vdash p_3}{K_2 K_1(p_1 \vee p_2 \vee p_3), K_2(\neg p_i \rightarrow K_j \neg p_i), \neg p_i \rightarrow K_j \neg p_i \vdash [\varphi_1 \wedge [\varphi_1] \varphi_2] p_3} \text{[Lem. 4]} \\
 \frac{K_2 K_1(p_1 \vee p_2 \vee p_3), K_2(\neg p_i \rightarrow K_j \neg p_i), \neg p_i \rightarrow K_j \neg p_i \vdash [\varphi_1 \wedge [\varphi_1] \varphi_2] p_3}{K_3 K_2 K_1(p_1 \vee p_2 \vee p_3), K_3 K_2(\neg p_i \rightarrow K_j \neg p_i), K_3(\neg p_i \rightarrow K_j \neg p_i) \vdash K_3[\varphi_1 \wedge [\varphi_1] \varphi_2] p_3} \text{[k_elim]} \\
 \frac{K_3 K_2 K_1(p_1 \vee p_2 \vee p_3), K_3 K_2(\neg p_i \rightarrow K_j \neg p_i), K_3(\neg p_i \rightarrow K_j \neg p_i), \varphi_1 \wedge [\varphi_1] \varphi_2 \vdash K_3[\varphi_1 \wedge [\varphi_1] \varphi_2] p_3}{K_3 K_2 K_1(p_1 \vee p_2 \vee p_3), K_3 K_2(\neg p_i \rightarrow K_j \neg p_i), K_3(\neg p_i \rightarrow K_j \neg p_i) \vdash \varphi_1 \wedge [\varphi_1] \varphi_2 \rightarrow K_3[\varphi_1 \wedge [\varphi_1] \varphi_2] p_3} \text{[Lem. 3]} \\
 \frac{K_3 K_2 K_1(p_1 \vee p_2 \vee p_3), K_3 K_2(\neg p_i \rightarrow K_j \neg p_i), K_3(\neg p_i \rightarrow K_j \neg p_i) \vdash \varphi_1 \wedge [\varphi_1] \varphi_2 \rightarrow K_3[\varphi_1 \wedge [\varphi_1] \varphi_2] p_3}{K_3 K_2 K_1(p_1 \vee p_2 \vee p_3), K_3 K_2(\neg p_i \rightarrow K_j \neg p_i), K_3(\neg p_i \rightarrow K_j \neg p_i) \vdash [\varphi_1][\varphi_2] K_3 p_3} \text{[Lem. 2]} \\
 \frac{K_3 K_2 K_1(p_1 \vee p_2 \vee p_3), K_3 K_2(\neg p_i \rightarrow K_j \neg p_i), K_3(\neg p_i \rightarrow K_j \neg p_i) \vdash [\varphi_1][\varphi_2] K_3 p_3}{C(p_1 \vee p_2 \vee p_3), C(p_i \rightarrow K_j p_i), C(\neg p_i \rightarrow K_j \neg p_i) \vdash [\varphi_1][\varphi_2] K_3 p_3} \text{[Lem. 1]}
 \end{array}$$

Figure 6.10: Simplified proof tree for the three wise men problem

6.3. Examples

$$[K] \frac{\neg p_2, \neg p_3, p_1 \vee p_2 \vee p_3 \vdash p_1}{K_1 \neg p_2, K_1 \neg p_3, K_1(p_1 \vee p_2 \vee p_3) \vdash K_1 p_1}$$

Figure 6.11: Proof Fragment for K elimination

inference steps. A defined proof rule is a strategy that is applied in a single atomic step so that only the final effect of the strategy is visible and the intermediate steps are hidden from the user. PVS provides strong support for writing strategies. Therefore being able to use proof strategies to increase the degree of automation is a major motivation for using PVS.

To illustrate how proof strategies can be useful, consider the proof fragment we have discussed in the Three Wise Men example shown in Fig. 6.11.

This figure represents the situation whereby there are n antecedent formulae and 1 consequent formula, all of which are of the form $K_a \varphi$. We want to simulate the effect of dashed boxes in natural deduction (as shown in Fig. 6.12) by stripping away K_a , so that the sequent can be simplified.

The following would be the naive method.

1. apply the `k_collect` reasoning rule $n-1$ times to merge the antecedent formulae into one formula (in this case, $K_1(\neg p_2 \wedge \neg p_3 \wedge (p_1 \vee p_2 \vee p_3))$),
2. apply the `kbox` rule to remove the K operator from both the antecedent and the consequent

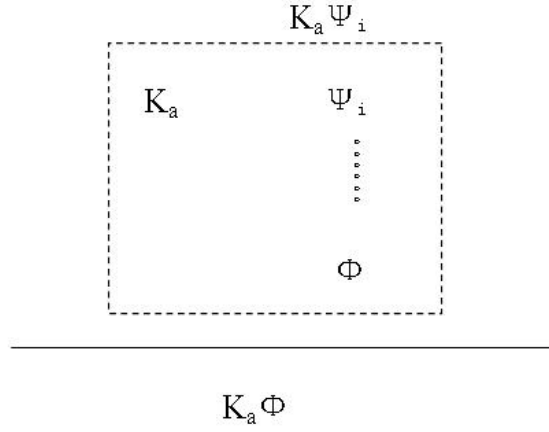


Figure 6.12: K introduction in natural deduction

3. apply evaluation axioms $n - 1$ times to break the single antecedent formula into n antecedent formulae

The problem with the naive method is that in the general case the number of antecedent formulae is neither fixed nor known. It is desired that we can achieve the effect with a single proof command regardless of the number of antecedent formulae. However it is important to note that such strategy is applicable only when *all* antecedent and consequent formulae are of the same modal operator (in this case K_1) to ensure the soundness of the deduction. The proof strategies are shown below. The strategy simply repeatedly applies the `k_collect` rule until there is no further effect to combine the formulae and recursively applies the `kand_collapse` rule to split the formulae.

6.3. Examples

```
(defstep k_elim ()
  (try (try (forward-chain "k_collect")
    (then (hide -2) (try (forward-chain -1)
      (then (hide -2) (k_elim))
      (then (fail) (fail))))))
    (skip))
  (kand_collapse)
  (skip))
  "k box: eliminating k operator" "k box: eliminating k operator")

(defstep kand_collapse ()
  (try (forward-chain "kand_ax")
    (then (hide -2) (kand_collapse))
    (skip))
  "collapsing kand clause" "collapsing kand clause")
```

In a similar way, we defined proof strategies for eliminating all types of modal operators such as shared knowledge, common knowledge and public announcement. Because the strategies are similar, we show only the `k_elim` strategy for illustration.

6.3.4 Generalizing the Example

An observation from the three wise men problem is that a generalized version of the problem, where there are n , an arbitrary number, wise men presents a system with an arbitrary number of epistemic states. In fact it is true that the last wise man would be able to know that he or she has a red hat if the previous $n - 1$ wise men had declared their ignorance. It is clear that such a system is beyond the expressiveness of

the spectrum of epistemic logics discussed in this paper. It is not possible to represent the system in the model checkers we have discussed, let alone to prove the property by using them. In this section, we demonstrated how such system can be represented and how the property can be proved using our framework.

```

n: nat
atLeastTwoWiseMen: AXIOM n >= 2
Range: TYPE+ = {m: nat | 1 <= m AND m <= n}
wm: [Range -> Agent]
p: [Range -> (base?)]
getAgents(wm: [Range -> Agent], i: Range, j: Range): set[Agent] =
  {a: Agent | EXISTS (m: Range): m >= i AND m <= j AND wm(m) = a}

```

We first define an arbitrary natural number constant `n` for the number of wise men and then define an axiom `atLeastTwoWiseMen` to state that there are at least two wise men in this problem. The type `Range` is a subtype of natural number between 1 and `n`. `wm` is a function which maps each value in `Range` to an agent. This makes it easy to refer to every single agent in the problem. `p` is a function which maps each value in `Range` to a base formula such that `p(i)` denotes that the *i*th wise man is wearing a red hat. So both the agents and the propositions are indexed by the values in `Range`. Then the function `getAgents` returns the set of agents indexed between `i` and `j` inclusive.

6.3. Examples

```

atLeastOneWhite(i : Range) : RECURSIVE Knowledge =
  IF i = 1 THEN p(1) ELSE kor(p(i), atLeastOneWhite(i-1)) ENDIF
  MEASURE i
init_1 : AXIOM eval(c(getAgents(wm, 1, n), atLeastOneWhite(n)))
init_2 : AXIOM FORALL (i,j : Range) : i /= j IMPLIES
  eval(c(getAgents(wm, 1, n), kif(p(i), k(wm(j), p(i)))))
init_3 : AXIOM FORALL (i,j : Range) : i /= j IMPLIES
  eval(c(getAgents(wm, 1, n), kif(knot(p(i)), k(wm(j), knot(p(i)))))

```

Our generalized problem is one with an arbitrary number of agents. We cannot specify the fact that at least one of the agents is wearing a white hat ($p_1 \vee p_2 \vee \dots p_n$) *explicitly* like what we did to the three wise man problem, because we simply cannot list them all. Here we use the recursive definition in PVS. The function `atLeastOneWhite` is recursively defined with a `measure` expression which is normally used to ensure termination. Then we can easily specify the three initial axioms as `init_1`, `init_2` and `init_3`.

```

ignorance(i : Range) : Knowledge =
  kand(knot(k(wm(i), p(i))), knot(k(wm(i), knot(p(i)))))
cons1(i : Range, e : Knowledge) : RECURSIVE Knowledge =
  IF i = 1 THEN pa(ignorance(1), e) ELSE cons1(i-1, pa(ignorance(i), e)) ENDIF
  MEASURE i
nwmProperty : THEOREM eval(cons1(n-1, k(wm(n), p(n))))

```

For reusability and easy readability, we define a function called `ignorance` specifying that a given agent does not know the colour of his or her hat. The recursive function `cons1` serves a constructive purpose. It simply constructs the arbitrary-length formula $[\neg K_1 p_1 \wedge \neg K_1 \neg p_1][\neg K_2 p_2 \wedge \neg K_2 \neg p_2] \dots [\neg K_i p_i \wedge \neg K_i \neg p_i]e$ where e is an arbitrary formula given as an argument. Then the property we want to prove can be

easily formulated as a theorem called `nwmProperty`.

```

cons2(i: Range): RECURSIVE Knowledge =
  IF i = n THEN p(n) ELSE kor(p(i), cons2(i+1)) ENDIF
  MEASURE n - i
cons3(i: Range, j : ({l: Range | i <= l}), e: Knowledge): RECURSIVE Knowledge =
  IF (i = j) THEN k(wm(i), e) ELSE cons3(i+1, j, k(wm(i),e))  ENDIF
  MEASURE j - i
lemma1: LEMMA FORALL (i: Range): i < n IMPLIES
  eval(cons1(i, cons3(i+1, n, cons2(i+1))))

```

In fact the theorem is the result an application of a general theorem.

$$\begin{aligned}
 & \forall i : 1..n, \\
 & [\neg K_1 p_1 \wedge \neg K_1 \neg p_1] [\neg K_2 p_2 \wedge \neg K_2 \neg p_2] \dots [\neg K_i p_i \wedge \neg K_i \neg p_i] \\
 & K_n K_{n-1} \dots K_{i+1} (p_{i+1} \vee p_{i+2} \vee \dots \vee p_n)
 \end{aligned}$$

This is specified as a lemma called `lemma1` which utilizes two more recursively defined functions called `cons2` and `cons3`.

We prove by using induction on i which leaves us with two proof obligations, the base case and the inductive case. The base case is

$$\vdash [\neg K_1 p_1 \wedge \neg K_1 \neg p_1] K_n K_{n-1} \dots K_2 (p_2 \vee p_3 \vee \dots \vee p_n)$$

and the inductive case is

$$\begin{aligned}
 & [\neg K_1 p_1 \wedge \neg K_1 \neg p_1] [\neg K_2 p_2 \wedge \neg K_2 \neg p_2] \dots [\neg K_i p_i \wedge \neg K_i \neg p_i] \\
 & K_n K_{n-1} \dots K_{i+1} (p_{i+1} \vee p_{i+2} \vee \dots \vee p_n) \\
 & \vdash \\
 & [\neg K_1 p_1 \wedge \neg K_1 \neg p_1] [\neg K_2 p_2 \wedge \neg K_2 \neg p_2] \dots [\neg K_{i+1} p_{i+1} \wedge \neg K_{i+1} \neg p_{i+1}] \\
 & K_n K_{n-1} \dots K_{i+2} (p_{i+2} \vee p_{i+3} \vee \dots \vee p_n)
 \end{aligned}$$

The simplified proof trees for the base case and the inductive case are shown in Fig.

6.13 and Fig. 6.14

6.3. Examples

$$\begin{array}{c}
\frac{p_1 \vee p_2 \vee \dots \vee p_n, \neg(p_2 \vee p_3 \vee \dots \vee p_n) \vdash p_1}{K_1(p_1 \vee p_2 \vee \dots \vee p_n), K_1 \neg(p_2 \vee p_3 \vee \dots \vee p_n) \vdash K_1 p_1} \\
\frac{K_1(p_1 \vee p_2 \vee \dots \vee p_n), \neg p_i \rightarrow K_j \neg p_i, \neg(p_2 \vee p_3 \vee \dots \vee p_n) \vdash K_1 p_1}{C(p_1 \vee p_2 \vee \dots \vee p_n), C(\neg p_i \rightarrow K_j \neg p_i), \neg K_1 p_1, \neg K_1 \neg p_1 \vdash p_2 \vee p_3 \vee \dots \vee p_n} \\
\frac{C(p_1 \vee p_2 \vee \dots \vee p_n), C(\neg p_i \rightarrow K_j \neg p_i) \varphi_1 \vdash p_2 \vee p_3 \vee \dots \vee p_n}{C(p_1 \vee p_2 \vee \dots \vee p_n), C(\neg p_i \rightarrow K_j \neg p_i) \vdash \varphi_1 \rightarrow (p_2 \vee p_3 \vee \dots \vee p_n)} \\
\frac{C(p_1 \vee p_2 \vee \dots \vee p_n), C(\neg p_i \rightarrow K_j \neg p_i) \vdash [\varphi_1](p_2 \vee p_3 \vee \dots \vee p_n)}{C(p_1 \vee p_2 \vee \dots \vee p_n), C(\neg p_i \rightarrow K_j \neg p_i) \vdash [\varphi_1] K_{n-1} \dots K_2 (p_2 \vee p_3 \vee \dots \vee p_n)} \\
\frac{K_n C(p_1 \vee p_2 \vee \dots \vee p_n), K_n C(\neg p_i \rightarrow K_j \neg p_i) \vdash K_n [\varphi_1] K_{n-1} \dots K_2 (p_2 \vee p_3 \vee \dots \vee p_n)}{C(p_1 \vee p_2 \vee \dots \vee p_n), C(\neg p_i \rightarrow K_j \neg p_i) \vdash K_n [\varphi_1] K_{n-1} \dots K_2 (p_2 \vee p_3 \vee \dots \vee p_n)} \\
\frac{C(p_1 \vee p_2 \vee \dots \vee p_n), C(\neg p_i \rightarrow K_j \neg p_i) \vdash \varphi_1 \rightarrow K_n [\varphi_1] K_{n-1} \dots K_2 (p_2 \vee p_3 \vee \dots \vee p_n)}{C(p_1 \vee p_2 \vee \dots \vee p_n), C(\neg p_i \rightarrow K_j \neg p_i) \vdash [\varphi_1] K_n K_{n-1} \dots K_2 (p_2 \vee p_3 \vee \dots \vee p_n)}
\end{array}$$

Figure 6.13: Simplified proof tree for the base case

$$\begin{array}{c}
\frac{p_{j+1} \vee \dots \vee p_n, p_{j+2} \vee \dots \vee p_n \vdash p_{j+1}}{K_{j+1}(p_{j+1} \vee \dots \vee p_n), K_{j+1}(p_{j+2} \vee \dots \vee p_n) \vdash K_{j+1} p_{j+1}} \\
\frac{K_{j+1}(p_{j+1} \vee \dots \vee p_n), \neg K_{j+1} p_{j+1}, \neg p_x \rightarrow K_y \neg p_x \vdash p_{j+2} \vee \dots \vee p_n}{K_{j+1}(p_{j+1} \vee \dots \vee p_n), \neg K_{j+1} p_{j+1} \vdash p_{j+2} \vee \dots \vee p_n} \\
\frac{K_{j+1}(p_{j+1} \vee \dots \vee p_n) \vdash \varphi_{j+1} \rightarrow (p_{j+2} \vee \dots \vee p_n)}{K_{j+1}(p_{j+1} \vee \dots \vee p_n) \vdash [\varphi_{j+1}](p_{j+2} \vee \dots \vee p_n)} \\
\frac{K_{n-1} \dots K_{j+1}(p_{j+1} \vee \dots \vee p_n) \vdash [\varphi_{j+1}] K_{n-1} \dots K_{j+2}(p_{j+2} \vee \dots \vee p_n)}{K_n \dots K_{j+1}(p_{j+1} \vee \dots \vee p_n) \vdash K_n [\varphi_{j+1}] K_{n-1} \dots K_{j+2}(p_{j+2} \vee \dots \vee p_n)} \\
\frac{K_n \dots K_{j+1}(p_{j+1} \vee \dots \vee p_n) \vdash \varphi_{j+1} \rightarrow K_n [\varphi_{j+1}] K_{n-1} \dots K_{j+2}(p_{j+2} \vee \dots \vee p_n)}{K_n \dots K_{j+1}(p_{j+1} \vee \dots \vee p_n) \vdash [\varphi_{j+1}] K_n \dots K_{j+2}(p_{j+2} \vee \dots \vee p_n)} \\
\frac{[\varphi_2] \dots [\varphi_j] K_n \dots K_{j+1}(p_{j+1} \vee \dots \vee p_n) \vdash [\varphi_2] \dots [\varphi_{j+1}] K_n \dots K_{j+2}(p_{j+2} \vee \dots \vee p_n)}{[\varphi_1] \dots [\varphi_j] K_n \dots K_{j+1}(p_{j+1} \vee \dots \vee p_n) \vdash [\varphi_1] \dots [\varphi_{j+1}] K_n \dots K_{j+2}(p_{j+2} \vee \dots \vee p_n)}
\end{array}$$

Figure 6.14: Simplified proof tree for the inductive case

6.4 Chapter Summary

In this chapter we presented a formal hierarchical framework for specifying and reasoning about higher-order agent knowledge. We encoded a hierarchy of epistemic logics K , $S5$, $S5C$, PAC and $PAL-C$ in the PVS specification language. The framework mainly consists of four components: *Proof Systems* for the ability to completely derive theorems of a particular logic, *Theorem Sets* for storing the theorems derived from the proof systems, *Reasoning Systems* for evaluating a formula of a concrete model, and *Reasoning Rule Sets* for storing reasoning rules for better proof automation. *Proof Systems* and *Theorem Sets* work on the meta-level while *Reasoning Systems* and *Reasoning Rule Sets* work on the object level. We demonstrated the idea by solving the classical Three Wise Men problem. One of the important contributions of this chapter is that we are able to use the proposed framework to specify systems with an arbitrary number of agents and epistemic states, which are not possible in the state-of-the-art epistemic model checkers. We have illustrated the advantage with the generalized version of the wise men problem.

Some researchers have done related work along a similar line. Kim and Kowalski used meta-reasoning with common knowledge based on a Prolog implementation to solve the same Three Wise Men puzzle [68]. Compared with their work, our approach has the advantage of being able to quantify over agent, knowledge and even functions, i.e., offering higher-order logic benefits. In [9], Basin *et al.* presented a theoretic-

6.4. Chapter Summary

cal and practical approach to the modular natural deduction presentation of a class of modal logics using Isabelle [89]. In [96], the sequent calculus of classical linear logic $KDT4_{lin}$ is coded in the higher order logic using the proof assistant COQ [21] with two-level meta-reasoning. These two pieces of work include neither the common knowledge operator nor the public announcement operator which adds much complexity to the reasoning process. A similar approach to our work was taken by Arkoudas and Bringsjord in [5]. Instead of encoding the logic and the axiomatization, they encode the sequent calculus for an epistemic logic in Athena [4], an interactive theorem prover too, and reason about the reasoning process in the logic. Two other major differences are that their work did not provide support for public announcement operators and that they did not comment on the completeness of their system.

Chapter 7

Conclusion

7.1 Main Contribution of the Thesis

Agent software technologies are currently still in an early stage of market development, where, arguably, the majority of users adopting the technology are visionaries who have recognized the long-term potential of agent systems. Despite the potential, one of the key problems awaiting to be resolved in agent-based technology is how to represent the agent's knowledge about the world and other interacting agents in such a way that effective and efficient reasoning about complex properties of agent knowledge can be performed.

More specifically, as we have identified in Chapter 1, the following four challenges,

among others, in the field of knowledge representation and reasoning in the intelligent agent domain exist and are of special interest to us. Here we iterate them again.

- **Interoperability and Heterogeneity:** Agent-based research is only just beginning to grapple with problems associated with the inevitable heterogeneity of its problem solving components. The basic problem is how agents with different domains of discourse, employing different knowledge representation schemes, different problem solving paradigms, and with different assumptions about their world and each other, can be made to interact in an effective and scalable manner.
- **Uncertainty, Vagueness and Incompleteness:** As agents have a necessarily partial perspective of their world, and because their problem domain is open, complex and distributed, they require sophisticated mechanisms for reasoning with uncertain, incomplete and contradictory information if they are to exhibit the desired degree of flexibility and robustness.
- **Rules-based Agent Knowledge and Reasoning:** Agents are situated in an environment and exhibit reactive and possibly proactive behavior. Rules are natural means to specify these forms of agent behavior. It is a challenge for agents to perform reasoning on and with such rules.
- **Multi-agent Knowledge Representation and Reasoning:** The area of

7.1. Main Contribution of the Thesis

multi-agent systems is traditionally concerned with formal representation of the mental state of autonomous agents in a distributed setting. The knowledge of an intelligent agent typically deals with what agents consider possible given their current information. This includes knowledge about facts as well as higher-order information about information that other agents have. It is a challenging task to enable systematic design of such intelligent agents as the reasoning process of interacting agents can be extremely complex.

This thesis presents our research works in tackling these challenges. The four main contributions of this thesis can be summarized as follows.

1. We have defined the PVS semantics for the ontology languages OWL DL and SWRL (Chapter 3), making it possible to use the PVS theorem prover to perform complex reasoning tasks on Semantic Web ontologies. We have shown that properties crucial to the validity of ontology-based knowledge can be checked by PVS. Some of these properties are beyond the expressiveness of the current ontology languages, including SWRL.
2. We propose Belief-augmented OWL (Chapter 4), as an ontology language enriched with belief information. As an extension of OWL DL, BOWL can be used to associate belief and disbelief factors directly with web resources, enabling software agents to perform more flexible and accurate reasoning. We

define the abstract syntax of BOWL and augment the model-theoretic semantics of OWL to incorporate belief values. We also define the reasoning tasks and algorithms for BOWL and present a prototype implementation using the constraint logic programming technique.

3. We defined a set of notions for the anomalies that may exist in a set of rules that an intelligent agent may possess (Chapter 5). Such anomalies include rule inconsistency, redundancy and circularity. We propose to use a combination of standard DL reasoning and the constraint logic programming techniques to discover anomalies in a SWRL agent rule base.
4. We presented a formal hierarchical framework for specifying and reasoning about higher-order agent knowledge (Chapter 6). We encoded a hierarchy of epistemic logics in the PVS specification language. The framework mainly consists of four components: *Proof Systems* for the ability to completely derive theorems of a particular logic, *Theorem Sets* for storing the theorems derived from the proof systems, *Reasoning Systems* for evaluating a formula of a concrete model, and *Reasoning Rule Sets* for storing reasoning rules for better proof automation. *Proof Systems* and *Theorem Sets* work on the meta-level while *Reasoning Systems* and *Reasoning Rule Sets* work on the object level. We demonstrated the idea by solving the classical multi-agent epistemic problem. One important contribution is that we are able to use the proposed framework

7.2. Future Work Directions

to specify systems with an arbitrary number of agents and epistemic states, which are not possible in the state-of-the-art epistemic model checkers.

7.2 Future Work Directions

Based on the work presented in this thesis, there are a number of directions of future research that may be pursued. In this section, some of these possible research works are briefly discussed.

7.2.1 Reasoning about Semantic Web Services

The Semantic Web should enable greater access not only to content but also to services. Users and software agents should be able to discover, invoke, compose, and monitor Web resources offering particular services and having particular properties, and should be able to do so with a high degree of automation if desired.

The OWL-S [106] ontology has been developed to enrich Web Services with semantics. The semantic markup of OWL-S enables the automated discovery, invocation, composition, interoperation and monitoring of Web services. This automation is achieved by providing a standard ontology for declaring and describing Web Services.

Being an OWL ontology, OWL-S defines a set of essential vocabularies to describe the

three components of a service: profile, model and grounding, as shown in Figure 7.1. A service can have several profiles and one service model. The service model, in turn, may have one or more service groundings. In summary, a service profile describes what the service does; the service model describes how the service works and the grounding provides a concrete specification of how the service can be accessed.

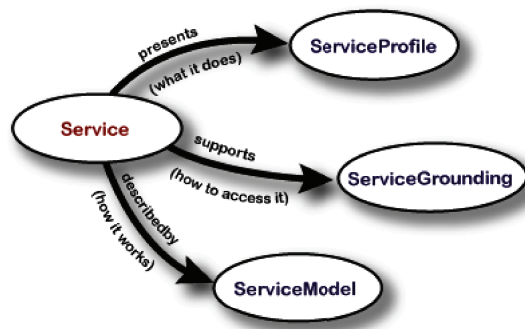


Figure 7.1: The main components of the OWL-S ontology.

The *ServiceProfile* class provides a bridge between service requesters and service providers. The instances of this class advertise an existing service by describing it in a general way that can be understood both by humans and computer agents. It is also possible to use a service profile to advertise a needed service request.

OWL-S provides a subclass of *ServiceProfile*, *Profile*. This default class should include provider information, a functional description and host properties of the described ser-

7.2. Future Work Directions

vice. It is possible to define other profile classes that specify the service characteristics more precisely.

The *ServiceModel* class uses the subclass *Process* to provide a process view on the service. This view can be thought of as a specification of the ways a client may interact with a service. The service model defines the inputs, outputs, preconditions and effects (IOPEs) and the control flow of composite processes.

The *ServiceGrounding* class provides a concrete specification of how the service can be accessed. Of main interest here are subjects including protocol, message formats, serialization, transport and addressing. The grounding can be thought of as the concrete part of the Semantic Web service description, compared to the service profile and service model which both describe the service on an abstract level.

Model checking techniques [20] may prove to be applicable in this domain. Communicating Sequential Processes (CSP) [53] is a well-known event-based formal notation primarily aimed at describing the sequencing of behavior within a process and the synchronization of behavior between different processes. We plan to apply a recently developed model checker, PAT [73, 104], to reason about semantic-based services. The analysis will include the checking of reachability, deadlock and refinement, and the verification of LTL formulae. Furthermore, PAT's strength is in verifying liveness properties under fairness assumptions, such as weak fairness and strong local/global fairness.

Complex Semantic Web services may have intricate data state, autonomous process behavior and concurrent interactions. The design of such systems requires precise and powerful modeling techniques to capture not only the ontology domain properties but also the services' process behavior and functionalities. Another approach is to apply an integrated formal modeling language, Timed Communicating Object-Z (TCOZ) [75, 76], to design SW services [113].

7.2.2 Combining Knowledge Uncertainty and Rules

We have seen in Chapter 4 and 5 how uncertainty reasoning and rule reasoning are possible in the context of ontology-based knowledge representation. In particular we have extended the ontology language OWL DL with belief factors. As SWRL is an extension of OWL DL with rule axioms. It is natural to provide reasoning support for belief-augmented rules. Certainty factors can be augmented to both the rule atoms or the whole implication.

7.2.3 Higher Automation for PVS Verification

Proof Strategies

We defined 30 proof strategies in Chapter 3 and 24 in Chapter 6 to make PVS proofs more automated. These strategies were designed according to the encoding of the

7.2. Future Work Directions

corresponding semantics in PVS. Currently, we are in the process of developing more intelligent strategies to implement heuristics found from our experiments, such as backtracking.

We are inspired by the work [2, 3, 82] which developed a tool TAME (Timed Automata Modeling Environment) based upon PVS, allowing users to specify and prove properties of three classes of automata without much effort. TAME utilized the capability of proof strategies in PVS and provided a collection of high level strategies dedicated to assisting mechanized proofs of particular properties, specifically, proofs of invariants and of weak refinement by means of induction.

Reasoning Rule Management

The frameworks we have proposed achieves a higher space bound than the current state-of-the-art model checkers for both ontology languages and epistemic logic, at the expense of automation. From time to time, the user has to select one from a set of rules that is applicable in some stage of the reasoning process. This requires much human expertise. Currently the reasoning rules are simply collected in a set which grows with the use of the systems. A better rule management will be able to categorize the rules according to some criteria such as the type of formula involved, the number of premises or the number of bound variables.

Bibliography

- [1] K. P. Adlassing, G. Kolarz, W. Scheithauern, H. Effenberger, and G. Grabner. CADIAG: Approaches to Computer-assisted Medical Diagnosis. *Computers in Biology and Medicine*, 15(5):315–335, 1985.
- [2] M. Archer. TAME: Using PVS Strategies for Special-purpose Theorem Proving. *Annals of Mathematics and Artificial Intelligence*, pages 139 – 181, 2000.
- [3] M. Archer, C. Heitmeyer, and E. Riccobene. Proving Invariants of I/O Automata with TAME. *Automated Software Engineering*, 9(3):201–232, 2002.
- [4] K. Arkoudas. Athena. <http://www.cag.csail.mit.edu/~kostas/dpls/athena/>.
- [5] K. Arkoudas and S. Bringsjord. Metareasoning for Multi-agent Epistemic Logics. In *Proceedings of the 5th International Conference on Computational Logic in Multi-Agent Systems*, pages 50–65, 2004.

- [6] R. J. Aumann. Agreeing to Disagree. *The Annals of Statistics*, 4(6):1236–1239, 1976.
- [7] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook – Theory, Implementation and Applications*. Cambridge University Press, 2003.
- [8] A. Baltag, L. S. Moss, and S. Solecki. The Logic of Public Announcements, Common Knowledge, and Private Suspicions. In *Proceedings of the 7th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 43–56, 1998.
- [9] D. Basin, S. Matthews, and L. Viganò. A Modular Presentation of Modal Logics in a Logical Framework. In *The Tbilisi Symposium on Logic, Language and Computation: Selected Papers*, pages 293–307. CSLI Publications, 1998.
- [10] J. Baumeister and D. Seipel. Verification and Refactoring of Ontologies With Rules. In *Proceedings of the 15th International Conference on Knowledge Engineering and Knowledge Management*, pages 82–95, 2006.
- [11] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference.
- [12] S. Bechhofer, R. Volz, and P. W. Lord. Cooking the Semantic Web with the OWL API. In *Proceedings of the 1st International Semantic Web Conference 2003*, pages 659–675, 2003.

BIBLIOGRAPHY

- [13] F. Bellifemine and G. Rimassa. Developing Multi-agent Systems with a FIPA-compliant Agent Framework. *Software – Practice & Experience*, 31(2):103–128, 2001.
- [14] M. Benerecetti, F. Giunchiglia, and L. Serafini. Model Checking Multiagent Systems. *Journal of Logic and Computation*, 8(3):401–423, 1998.
- [15] T. Berners-Lee. Notation 3 – Ideas about Web Architecture. <http://www.w3.org/DesignIssues/Notation3.html>, 1998.
- [16] T. Berners-Lee. Uniform Resource Identifiers (URI): Generic Syntax. <http://www.ietf.org/rfc/rfc2396.txt>, 1998.
- [17] T. Berners-Lee. CWM - A General Purpose Data Processor for the Semantic Web. www.w3.org/2000/10/swap/doc/cwm, 2004.
- [18] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):35–43, 2001.
- [19] J. Broekstra, M. Klein, S. Decker, D. Fensel, and I. Horrocks. Adding Formal Semantics to the Web: Building on top of RDF Schema. In *Proceedings of ECDL Workshop on the Semantic Web: Models, Architectures and Management*, 2000.
- [20] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite State Concurrent System using Temporal Logic Specifications: a Practical

- Approach. In *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 117–126. ACM, 1983.
- [21] T. Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76(2-3):95–120, 1988.
- [22] Dan Brickley and R.V. Guha (editors). Rdf vocabulary description language 1.0: Rdf schema. <http://www.w3.org/TR/rdf-schema/>, 2004.
- [23] B. V. Dasarathy. *Decision Fusion*. IEEECS, 1994.
- [24] A. P. Dempster. Upper and Lower Probabilities Induced by Multivalued Mapping. *Annals of Mathematical Statistics*, 38, 1967.
- [25] Z. Ding and Y. Peng. A Probabilistic Extension to Ontology Language OWL. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, page 40111.1, 2004.
- [26] J. S. Dong, Y. Feng, and H. fung Leung. A Verification Framework for Agent Knowledge. In *Proceedings of the 9th International Conference on Formal Engineering Methods*, pages 57–75, 2007.
- [27] J. S. Dong, Y. Feng, and H. fung Leung. A Verification Framework for Agent Knowledge. Submitted to Formal Aspects of Computing, 2009.

BIBLIOGRAPHY

- [28] J. S. Dong, Y. Feng, and Y. F. Li. Verifying OWL and ORL Ontologies in PVS. In *Proceedings of the 1st International Colloquium on Theoretical Aspects of Computing*, pages 201–210, 2004.
- [29] J. S. Dong, Y. Feng, Y. F. Li, and J. Sun. A Tools Environment for Developing and Reasoning about Ontologies. In *Proceedings of the 12th Asia-Pacific Software Engineering Conference*, pages 465–472, 2005.
- [30] J. S. Dong, Y. Feng, Y.-F. Li, C. K.-Y. Tan, B. Wadhwa, and H. H. Wang. Belief-augmented OWL (BOWL) - Engineering the Semantic Web with Beliefs. Submitted to *Innovations in Systems and Software Engineering*, A NASA Journal., 2010.
- [31] J. S. Dong, Y. Feng, J. Sun, and J. Sun. Context Awareness Systems Design and Reasoning. In *Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods*, pages 335–340, 2006.
- [32] J. S. Dong, Y. Feng, J. Sun, and J. Sun. Sensor Based Design Techniques for Smart System Space. In *Proceedings of the 1st International Conference on Mobile Computing, Communications and Applications*, 2006.
- [33] J. S. Dong, C. H. Lee, Y. F. Li, and H. Wang. A Combined Approach to Checking Web Ontologies. In *Proceedings of the 13th International World Wide Web Conference*, pages 714–722, 2004.

- [34] J. S. Dong, C. H. Lee, Y. F. Li, and H. Wang. Verifying DAML+OIL and Beyond in Z/EVES. In *Proceedings of the 26th International Conference on Software Engineering*, pages 201–210, 2004.
- [35] J. S. Dong, J. Sun, and H. Wang. Checking and Reasoning about Semantic Web through Alloy. In *Proceedings of the 12th International Symposium on Formal Methods Europe*, pages 796–813, 2003.
- [36] T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. DLV-HEX: A System for Integrating Multiple Semantics in an Answer-Set Programming Framework. In *Proceedings of the 20th Workshop on Logic Programming*, pages 206–210, 2006.
- [37] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.
- [38] Y. Feng, Y.-F. Li, C. K.-Y. Tan, B. Wadhwa, and H. H. Wang. Belief-augmented OWL (BOWL) - Engineering the SemanticWeb with Beliefs. In *Proceedings of the 12th International Conference on Engineering of Complex Computer Systems*, pages 165–174, 2007.
- [39] Y. Feng, Y. Liu, Y.-F. Li, and D. Zhang. Discovering Anomalies in Semantic Web Rules. In *Proceedings of the 4th IEEE International Conference on Secure Software Integration and Reliability Improvement*. Accepted.

BIBLIOGRAPHY

- [40] Y. Feng, Y. Liu, Y.-F. Li, and D. Zhang. Discovering Rule Anomalies in Ontology-based Pervasive, Context-aware Systems. Submitted to Transactions on Autonomous and Adaptive Systems - Special Issue on Formal Methods for Pervasive, Self-Adaptive, and Context-Aware Systems, 2010.
- [41] S. Ferndrigger, A. Bernstein, J. S. Dong, Y. Feng, Y.-F. Li, and J. Hunter. Enhancing Semantic Web Services with Inheritance. In *Proceedings of the 7th International Semantic Web Conference*, pages 162–177, 2008.
- [42] E. Friedman-Hill. *Jess in Action: Java Rule-Based Systems*. Manning Publications Co., 2003.
- [43] P. Gammie and R. van der Meyden. MCK: Model Checking the Logic of Knowledge. In *Proceedings of the 16th International Conference on Computer Aided Verification*, pages 479–483, 2004.
- [44] J. Gerbrandy. *Bisimulations on Planet Kripke*. PhD thesis, University of Amsterdam, 1999.
- [45] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. More Evaluation of Decision Procedures for Modal Logics. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pages 626–635, 1998.

- [46] M. J. Gordon. HOL: A Proof Generating System for Higher-order Logic. In G. Birwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 73–127. Academic Press, 1988.
- [47] M. J. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer, 1979.
- [48] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of the 12th International World Wide Web Conference*, pages 48–57, 2003.
- [49] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
- [50] V. Haarslev and R. Möller. RACER System Description. In *Proceedings of the 1st International Joint Conference on Automated Reasoning*, pages 701–706, 2001.
- [51] V. Haarslev and R. Möller. Practical Reasoning in Racer with a Concrete Domain for Linear Inequations. In *Proceedings of the International Workshop on Description Logics*, pages 91–98, 2002.
- [52] D. Heckerman, E. Horvitz, B. Nathwani, D. E. Heckerman, E. J. Horvitz, and B. N. Nathwani. Update on the Pathfinder Project. In *In Proceedings of the*

BIBLIOGRAPHY

- 13th Symposium on Computer Applications in Medical Care*, pages 203–207, 1989.
- [53] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [54] I. Horrocks. The FaCT System. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 307–312, 1998.
- [55] I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *Proceedings of the 13th International World Wide Web Conference*, pages 723–731, 2004.
- [56] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosz, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. <http://www.w3.org/Submission/SWRL/>, 2004.
- [57] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From *SHIQ* and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26, 2003.
- [58] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.
- [59] HP Labs. Jena 2 - A Semantic Web Framework. www.hpl.hp.com/semweb/jena2.htm.

- [60] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 1999.
- [61] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proceedings of the 14th Annual ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.
- [62] J. Jaffar, S. Michaylov, P. J. Stuckey, and R. Yap. The CLP(\mathcal{R}) Language and System. *Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.
- [63] J. Jaffar, A. Santosa, and R. Voicu. A CLP Proof Method for Timed Automata. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium*, pages 175–186, 2004.
- [64] J. Jaffar, A. Santosa, and R. Voicu. Modeling Systems in CLP with Coinductive Tabling. In *Proceedings of the 21st International Conference on Logic Programming*, pages 412–413, 2005.
- [65] Jeff Heflin (editor). OWL Web Ontology Language Use Cases and Requirements. <http://www.w3.org/TR/webont-req/>, 2004.
- [66] N. R. Jennings and M. Wooldridge. Applications of Intelligent Agents. pages 3–28, 1998.

BIBLIOGRAPHY

- [67] John McCarthy. *Formalization of Common Sense, Papers by John McCarthy edited by V. Lifschitz*. Ablex, 1990.
- [68] J.-S. Kim and R. A. Kowalski. An Application of Amalgamated Logic to Multi-Agent Belief. In *Proceedings of the 2nd Workshop on Metaprogramming in Logic*, pages 272–283, 1990.
- [69] H. Knublauch, R. W. Ferguson, N. F. Noy, and M. A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *Proceedings of the 3rd International Semantic Web Conference*, pages 229–243, 2004.
- [70] D. Koller, A. Levy, and A. Pfeffer. P-CLASSIC: A Tractable Probabilistic Description Logic. In *Proceedings of the 14th National Conference on AI*, pages 390–397, 1997.
- [71] J. E. Laird, A. Newell, and P. S. Rosenbloom. SOAR: An Architecture for General Intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.
- [72] D. K. Lewis. *Convention: A Philosophical Study*. Harvard University Press, 1969.
- [73] Y. Liu, J. Sun, and J. S. Dong. An Analyzer for Extended Compositional Process Algebras. In *Companion of the 30th International Conference on Software Engineering*, pages 919–920. ACM, 2008.

- [74] C. Lutz. Complexity and Succinctness of Public Announcement Logic. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 137–143, 2006.
- [75] B. P. Mahony and J. S. Dong. Blending Object-Z and Timed CSP: An Introduction to TCOZ. In *Proceedings of the 20th International Conference on Software Engineering (ICSE'98)*, pages 95–104, 1998.
- [76] B. P. Mahony and J. S. Dong. Timed Communicating Object Z. *IEEE Transactions on Software Engineering*, 26(2), Feb. 2000.
- [77] F. Manola and E. M. (editors). RDF Primer. www.w3.org/TR/rdf-primer/, 2004.
- [78] W. McCune. Otter 2.0. In *Proceedings of the 10th International Conference on Automated Deduction*, pages 663–664, London, UK, 1990. Springer-Verlag.
- [79] D. L. McGuinness and F. van Harmelen (editors). OWL Web Ontology Language Overview. <http://www.w3.org/TR/owl-features/>, 2003.
- [80] J.-J. C. Meyer and W. V. D. Hoek. *Epistemic Logic for AI and Computer Science*. Cambridge University Press, 1995.
- [81] M. Minsky. A Framework for Representing Knowledge. In *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, New York, 1975.

BIBLIOGRAPHY

- [82] S. Mitra and M. Archer. PVS Strategies for Proving Abstraction Properties of Automata. *Electronic Notes in Theoretical Computer Science*, 125(2):45–65, 2005.
- [83] B. Motik, I. Horrocks, R. Rosati, and U. Sattler. Can OWL and Logic Programming Live Together Happily Ever After? In *Proceedings of the 5th International Semantic Web Conference*, pages 501–514, 2006.
- [84] D. Nardi and R. J. Brachman. An Introduction to Description Logics. In *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 1–40. 2003.
- [85] K.-C. Ng and B. Abramson. Uncertainty Management in Expert Systems. *IEEE Intelligent Systems*, 5:29–48, 1990.
- [86] H. Nottelmann and N. Fuhr. pDAML+OIL: A Probabilistic Extension to DAML+OIL Based on Probabilistic Datalog. In *Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems*, 2004.
- [87] S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In *Proceedings of the 11th International Conference on Automated Deduction*, volume 607, pages 748–752, June 1992.

- [88] P. F. Patel-Schneider, P. Hayes, and I. H. (editors). OWL Web Ontology Language Semantics and Abstract Syntax. <http://www.w3.org/TR/owl-semantics/>, 2004.
- [89] L. C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer, 1994.
- [90] W. Penczek and A. Lomuscio. Verifying epistemic properties of multi-agent systems via bounded model checking. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 209–216, 2003.
- [91] J. Picard. *Probabilistic Argumentation Systems Applied to Information Retrieval*. PhD thesis, Universite de Neuchatel, Suisse, 2000.
- [92] J. A. Plaza. Logics of Public Communications. In *Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems*, pages 201–216, 1989.
- [93] S. Poslad, P. Buckle, and R. Hadingham. The FIPAOS Agent Platform: Open Source for Open Standards. In *Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, pages 355–368, 2000.
- [94] A. D. Preece and R. Shinghal. Foundation and Application of Knowledge Base Verification. *International Journal of Intelligent Systems*, 9(8):683–701, 1994.

BIBLIOGRAPHY

- [95] J. D. Roo. Euler Proof Mechanism. www.agfa.com/w3c/euler, 2002.
- [96] M. Sadrzadeh. Modal Linear Logic in Higher Order Logic, an Experiment in Coq. In *Proceedings of Theorem Proving in Higher Order Logics*, pages 75–93, 2003.
- [97] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.
- [98] M. Sintek and S. Decker. TRIPLE – A Query, Inference, and Transformation Language for the Semantic Web. In *Proceedings of the 1st International Semantic Web Conference*, pages 364–378, 2002.
- [99] E. Sirin and B. Parsia. Pellet: An OWL DL Reasoner. In *Proceedings of the 2004 International Workshop on Description Logics*, 2004.
- [100] A. Sloman and B. Logan. Building Cognitively Rich Agents Using the Sim Agent Toolkit. *Communications of the Association of Computing Machinery*, 42:71–77, 1999.
- [101] P. Smets. Belief Functions and the Transferable Belief Model. <http://ippserv.rug.ac.be/documentation/belief/belief.pdf>, 2000.
- [102] U. Straccia. Reasoning within Fuzzy Description Logics. *Journal of Artificial Intelligence Research*, 14(14):137–166, 2001.

- [103] U. Straccia. A Fuzzy Description Logic for the Semantic Web. In *Fuzzy Logic and the Semantic Web, Capturing Intelligence, Chapter 4*, pages 167–181. Elsevier, 2005.
- [104] J. Sun, Y. Liu, J. S. Dong, and J. Pang. PAT: Towards Flexible Verification under Fairness. In *Proceedings of the 21th International Conference on Computer Aided Verification*, pages 709–714. Springer, 2009.
- [105] C. K.-Y. Tan. *Belief Augmented Frames*. PhD thesis, National University of Singapore, 2003.
- [106] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. <http://www.daml.org/services/owl-s/>, 2004.
- [107] C. Tresp and R. Molitor. A Description Logic for Vague Knowledge. In *Proceedings of the 13th Biennial European Conference on Artificial Intelligence*, pages 361–365, 1998.
- [108] D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, pages 292–297, 2006.
- [109] W. van der Hoek and M. Wooldridge. Model Checking Knowledge and Time. In *Proceedings of the 9th International SPIN Workshop on Model Checking of Software*, pages 95–111, 2002.

BIBLIOGRAPHY

- [110] H. P. van Ditmarsch, W. van der Hoek, and B. P. Kooi. *Dynamic Epistemic Logic*. Springer, 2006.
- [111] J. van Eijck. Dynamic Epistemic Modelling. CWI Technical Report SEN-E0424, Centrum voor Wiskunde en Informatica, Amsterdam, 2004.
- [112] F. van Harmelen, P. F. Patel-Schneider, and I. H. (editors). Reference Description of the DAML+OIL Ontology Markup Language. <http://www.daml.org/2001/03/reference.html>, 2001.
- [113] H. H. Wang, J. S. Dong, J. Sun, and Y.-F. Li. TCOZ Approach to OWL-S Process Model Design. In *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering*, pages 354–359, 2005.
- [114] X. Wang, J. S. Dong, C. Chin, S. Hettiarachchi, and D. Zhang. Semantic Space: An Infrastructure for Smart Spaces. *IEEE Pervasive Computing*, 03(3):32–39, 2004.
- [115] X. H. Wang, D. Zhang, T. Gu, and H. K. Pung. Ontology Based Context Modeling and Reasoning using OWL. In *2nd IEEE Conference on Pervasive Computing and Communications Workshops*, pages 18–22, 2004.
- [116] M. Wooldridge, M. Fisher, M.-P. Huget, and S. Parsons. Model Checking Multi-agent Systems with MABLE. In *Proceedings of the 1st International Joint*

- Conference on Autonomous Agents and Multiagent Systems*, pages 952–959, 2002.
- [117] M. Wooldridge and N. R. Jennings. *Intelligent Agents: Theory and Practice*. *Knowledge Engineering Review*, 10:115–152, 1995.
- [118] World Wide Web Consortium (W3C). Extensible Markup Language (XML). <http://www.w3.org/XML>, 2000.
- [119] World Wide Web Consortium (W3C). XML Schema. <http://www.w3.org/XML/Schema>, 2000.
- [120] L. A. Zadeh. Fuzzy Sets. *Information and Control*, 8:338–353, 1965.