IST-2004-027655

ESTRELLA
European project for Standardised Transparent Representations in order to Extend Legal Accessibility

Specific Targeted Research Project
Information Society Technologies

Deliverable 1.1

Specification of the Legal Knowledge Interchange Format

Due date of deliverable: 31 December 2006
Actual submission date: 22 January 2007

Start date of project: 1 January 2006
Duration: 30 months

University of Amsterdam

PROJECT CO-FUNDED BY THE EUROPEAN COMMISSION WITHIN THE SIXTH FRAMEWORK PROGRAMME (2002-2006)
Executive Summary

This deliverable reports on the specification of syntax and semantics of the LKIF language, and should be read in conjunction with deliverable D1.4 on the LKIF ontologies. The proposed LKIF adds a terminology for deontic statements, and a special LKIF rule language to existing fragments of OWL and the rule language SWRL.

The Technical Annex states about this work package:

The main technical objectives of this work package are to develop a first version of a Legal Knowledge Interchange Format (LKIF), building upon emerging XML-based standards of the Semantic Web, including RDF and OWL, and Application Program Interfaces (APIs) for interacting with legal knowledge systems. The LKIF will apply the state of the art in the field of Artificial Intelligence and Law, taking into account business and application requirements. Existing Semantic Web initiatives are aimed at modelling concepts (OWL "ontologies") and rules (RuleML and SWRL). The LKIF will build on but go beyond this generic work to allow further kinds of legal knowledge to be modelled, including: meta-level rules for reasoning about rule priorities and exceptions, legal arguments, cases and case factors, values and principles, and legal procedures. In addition, an OWL ontology of basic legal concepts, such as obligations, permissions, rights and powers, will be developed, which can be reused when modelling a specific legal domain, such as tax law.

In chapter 2, we elaborate on these goals, by developing more detailed requirements for LKIF. These requirements are derived from several sources:

1. From a survey of research on computational models of legal reasoning and argumentation, from the field of Artificial Intelligence and Law.

2. From an analysis of the business requirements articulated by the participating vendors, and from the logical reconstructions of the logics used by these vendors, as reported in the companion report, deliverable D1.2. [van den Berg et al., 2006].

3. From feedback and comments by the participating user organizations and members of the observatory board on earlier versions of this report.

Particular attention is given to normative reasoning, and the question why deontic logics are not really suitable candidates for an interchange format for legal knowl-
edge. Another discussion is devoted to argumentation, argumentation schemes, argumentation protocols and all those argumentation epistemological-level phenomena characteristic of the dialectical character of law.

Appendix B introduces candidate Semantic Web standards and technologies that were seriously considered for the LKIF. The objective is that the LKIF will be compatible with existing standards and technologies, and that LKIF knowledge representations can be used by existing tools not specific to the legal field. A combination of several sublanguages of OWL (DLP, DL, Full) and two fragments of SWRL (DLP, DL-safe) was selected to build the LKIF on. The selected sublanguages are based on logical fragments identified in Semantic Web literature. In some cases there are existing reasoners for them. Chapter 3 gives a formal characterization of these sublanguages, and prescribes how these languages should be integrated into one language.

Chapter 4 discusses the structure of LKIF. LKIF can be said consist of different sublanguages (just like each standard ontology dependant of Semantic Web standards). The sublanguages differ in expressiveness, computational complexity, and available terminology. The available options can be categorized in four dimensions:

**Ontology** Which files of the LKIF ontology are used, and which not? A trivial use of LKIF would for instance consist of importing the time ontology, while ignoring norms and LKIF Rules. Of each OWL module, a DLP version and a DL version will be made available.

**Description Logic** Four fragments of OWL can be considered part of LKIF: OWL DLP, OWL DL, propositional OWL, and OWL Full. OWL DL and OWL DLP Semantics where given in chapter 3. OWL Full will not be supported by reasoners, and is merely provided as an option for building extensions to LKIF. Propositional OWL is dealt with separately.

**SWRL** Three fragments of SWRL can be considered part of LKIF: The DLP subset of SWRL and the DL-safe subset were defined in chapter 3. No SWRL is also a valid option.

**Special semantics** Two types of special semantics can currently be considered part of LKIF: LKIF rules as defined in section 6.2, and normative statements as defined in section 6.1. No use of special LKIF reasoning is also a valid option.

A choice from each dimension identifies a context of use for LKIF, ranging from mere use of LKIF defined concepts to the use of LKIF rules, and a specific LKIF reasoner. Some combinations already enjoy partial or complete support from existing software. Interesting combinations are for instance the following:

1. Propositional OWL and the DLP variant of the LKIF Ontology is the easiest to support for existing tools, and for instance the vendors part of the consortium.

2. OWL DL and the DLP variant of SWRL, with the DL variant of the ontology is already supported by existing tools like the Pellet reasoner, SWOOP, TopBraid, and Protege.
3. OWL DL and the DLP variant of SWRL, with the DL variant of the ontology, and special semantics for normative statements can be easily supported with a preprocessor and existing technology.

4. OWL DLP, the DLP variant of SWRL, with the DLP variant of the ontology, and special semantics for LKIF Rules is the most promising option for support of argumentation, and besides that many argumentation applications don’t make very high demands of the underlying reasoner.

The chapter following that summarizes of possible syntaxes for storing LKIF files. Most of these are taken from existing languages. All existing RDF syntaxes can of course be used. In addition we extended an existing linear pretty-printing syntax for description logic. This same syntax is used throughout the report. There is a graphical diagramming method for description logic, derived from UML, and a proposal for a new one for argument structures. The last one is however not part of the specification, since the argument structure has not been fixed yet.

Chapter 6 specifies the semantics of extensions for normative reasoning, or more accurately for inferring “betterness”, and LKIF Rules. The normative reasoning extensions involves just one principle: disjointness of the thing allowed and disallowed by the same norm. The LKIF Rules extension is a bit more complex, and extends SWRL. Semantics of this extension is argumentation-theoretic, and appeals to the concepts of argument scheme and argument structure.

Summarizing we can say that LKIF sofar is mostly an OWL ontology. In addition to that it is a method of using OWL and SWRL. Thirdly, it contains two new sublanguages, for defeasible rules and for subjunctive betterness. It is also unfinished, for instance because it lacks a clear explanation of the relation between OWL model theory and argumentation theory: the level at which these connect is presumably proof theory level.

In addition to that the ontology also only covers a base level of components required for explaining epistemological, situational, and mereological patterns as they occur in legal reasoning, but not (yet) the complex legal concepts identifying these patterns: this is a conscious methodological choice as deliverable D1.4 explains.
The main technical objectives of work package 1 are to develop a first version of a Legal Knowledge Interchange Format (LKIF), building upon emerging XML-based standards of the Semantic Web, including RDF and OWL, and Application Program Interfaces (APIs) for interacting with legal knowledge systems. The LKIF will apply the state of the art in the field of Artificial Intelligence and Law, taking into account business and application requirements. This deliverable reports on the specification of syntax and semantics of the LKIF language, and should be read in conjunction with deliverable D1.4 on the LKIF ontologies. The proposed LKIF adds a terminology for deontic statements, and a special LKIF rule language to existing fragments of OWL and the rule language SWRL.
Contents

1 Introduction .................................................. 2
   1.1 Intended Audience ............................................... 2
   1.2 Acronyms .......................................................... 3
   1.3 Preliminaries ................................................... 3

2 Requirements .................................................. 5
   2.1 Artificial Intelligence and Law .................................. 6
      2.1.1 Introduction to Argumentation Schemes ....................... 9
      2.1.2 Legal Argumentation Schemes .................................. 11
      2.1.3 Normative Reasoning ........................................... 17
   2.2 Legal Expert Systems ........................................... 32
      2.2.1 Normative Statements ........................................... 32
      2.2.2 Other uses of Production Rules ............................... 34
      2.2.3 Teleological Statements ....................................... 34
      2.2.4 Declarations .................................................. 35
      2.2.5 Requirements from Administrative Expert Systems .......... 36
   2.3 Vendor Requirements ........................................... 36

3 Description Logic and Rules ................................... 39
   3.1 Using a Combination of OWL DL and Rules .................... 39
   3.2 RDF/RDFS Semantics ............................................. 41
   3.3 OWL DL: Syntax and Semantics ................................ 45
      3.3.1 Why some rules cannot be represented in OWL DL? .......... 50
   3.4 Description Logic Programs .................................... 52
   3.5 SWRL: Syntax and Semantics ................................... 53
      3.5.1 The Strongly Safe Subset of SWRL ............................. 56

4 Overview of LKIF ................................................. 62
   4.1 Dimensions ..................................................... 63
   4.2 Ontologies of Law .............................................. 64
   4.3 MetaLex XML .................................................... 65

5 Syntax of LKIF ................................................... 67
   5.1 OWL Abstract Syntax of LKIF Rules .......................... 67
      5.1.1 LKIF Pretty-printing Syntax ................................. 69
      5.1.2 LKIF Rules S-expression Syntax ............................. 71
   5.2 Visualizations of LKIF ........................................... 75
D  Argument Structure Graphs  119
Chapter 1

Introduction

The main technical objectives of work package 1 are to develop a first version of a Legal Knowledge Interchange Format (LKIF), building upon emerging XML-based standards of the Semantic Web, including RDF and OWL, and Application Program Interfaces (APIs) for interacting with legal knowledge systems. The LKIF will apply the state of the art in the field of Artificial Intelligence and Law, taking into account business and application requirements.

This deliverable reports on the specification of syntax and semantics of the LKIF language, and should be read in conjunction with deliverable D1.4 on the LKIF ontologies. The proposed LKIF adds a terminology for deontic statements, and a special LKIF rule language to existing fragments of OWL and the rule language SWRL.

This section explains the structure of the deliverable, and partially restates the content of the executive summary attached to the deliverable. Chapter 2 contains an introduction into the issues that guided LKIF development.

Appendix B introduces candidate Semantic Web standards and technologies that were seriously considered for the LKIF. Read this appendix, or deliverable D1.5, first if this issue is of interest.

A combination of several sublanguages of OWL (DLP, DL, Full) and two fragments of SWRL (DLP, DL-safe) was selected to build the LKIF on. The selected sublanguages are based on logical fragments identified in Semantic Web literature. In some cases there are existing reasoners for them. Chapter 3 gives a formal characterization of these sublanguages, and prescribes how these languages should be integrated into one language, and it is therefore considered a part of the main body of the deliverable.

Chapter 4 discusses the structure of LKIF, followed by a chapter on syntax. Chapter 6 specifies the semantics of extensions for normative reasoning and LKIF Rules.

1.1 Intended Audience

This version of the LKIF Specification was created specifically for the reviewers of the Estrella project. The specification is not yet in a form that helps implementation of LKIF reasoners, or allows submission of the specification as a standard proposal to a standardization body.
The document contains long chapters (i.e. chapters 2 and B) that justify and explain design decisions. A final version for a broader audience should be transparent enough not to need these chapters. The body of the specification is in chapter 4 and the chapters following it.

Wherever a logical specification is needed, description logic syntax and terminology is favoured. If description logic is insufficient for explanation, first order logic is used. Knowledge of these two types of logics is recommended.

1.2 Acronyms

The following acronyms are used in this report:

**DL** Description Logic

**FOL** First Order Logic

**DTL** Deontic Logic

**AL** Action Logic

**DFL** Defeasible Logic

**SWRL** Semantics Web Rule Language

**OWL** Ontology Web Language

**URL** Unique Resource Locator

**URI** Unique Resource identifier

**KIF** Knowledge Interchange Format

**LKIF** Legal Knowledge Interchange Format

**DLP** Description Logic Programming

**RDF(S)** Resource Description Framework (Schema)

**HTML** HyperText Markup Language

**SGML** Standard Generalized Markup Language

**XML** eXtensible Markup language

1.3 Preliminaries

First Order Logic (\(\mathcal{FOL}\)) and Description Logic (\(\mathcal{DL}\)) are central to understanding this specification. Appendix A introduces notation and concepts. Table 1.1 explains the relation between \(\mathcal{FOL}\) and \(\mathcal{DL}\) sentences. \(\mathcal{FOL}\) is more expressive, and is used if \(\mathcal{DL}\) is unsufficient.
<table>
<thead>
<tr>
<th><strong>DL-SYNTAX</strong></th>
<th><strong>FOL-SYNTAX</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>⊤</td>
<td>$C(x) \lor \neg C(x)$</td>
</tr>
<tr>
<td>⊥</td>
<td>$C(x) \land \neg C(x)$</td>
</tr>
<tr>
<td>C</td>
<td>$C(x)$</td>
</tr>
<tr>
<td>$\neg C$</td>
<td>$\neg C(x)$</td>
</tr>
<tr>
<td>$C_1 \sqcap C_2$</td>
<td>$C_1(x) \land C_2(x)$</td>
</tr>
<tr>
<td>$C_1 \sqcup C_2$</td>
<td>$C_1(x) \lor C_2(x)$</td>
</tr>
<tr>
<td>$\forall P.C$</td>
<td>$\forall y(P(x, y) \rightarrow C(y))$</td>
</tr>
<tr>
<td>$\exists P.C$</td>
<td>$\exists y(P(x, y) \land C(y))$</td>
</tr>
<tr>
<td>$n \leq P$</td>
<td>$\forall y_1, \ldots, y_{n+1} : P(x, y_1) \land \ldots P(x, y_{n+1}) \rightarrow \lor_{i&lt;j}(y_i = y_j)$</td>
</tr>
<tr>
<td>$n \leq P.C$</td>
<td>$\forall y_1, \ldots, y_{n+1} : P(x, y_1) \land \ldots P(x, y_{n+1}) \land C(y_1) \land \ldots C(y_{n+1}) \rightarrow \lor_{i&lt;j}(y_i = y_j)$</td>
</tr>
<tr>
<td>⋇</td>
<td>$\forall x, \forall y(P(x, y))$</td>
</tr>
<tr>
<td>⋃</td>
<td>$\forall x, \forall y P^-(x, y)$</td>
</tr>
<tr>
<td>id(C)</td>
<td>$id(x, x) \land C(x)$</td>
</tr>
<tr>
<td>$\neg P$</td>
<td>$\neg P(x, y)$</td>
</tr>
<tr>
<td>$P_1 \sqcap P_2$</td>
<td>$P_1(x, y) \land P_2(x, y)$</td>
</tr>
<tr>
<td>$P_1 \sqcup P_2$</td>
<td>$P_1(x, y) \lor P_2(x, y)$</td>
</tr>
<tr>
<td>$P_1 \circ P_2$</td>
<td>$\exists z(P_1(x, z) \land P_2(z, y))$</td>
</tr>
<tr>
<td>$P^-$</td>
<td>$P^-(y, x)$</td>
</tr>
<tr>
<td>$P \upharpoonleft C$</td>
<td>$P(x, y) \land C(x)$</td>
</tr>
<tr>
<td>$P \upharpoonright C$</td>
<td>$P(x, y) \land C(y)$</td>
</tr>
</tbody>
</table>

$P^+$ and $P^*$ non first-order expressible

| C(a) | C(a) |
| C_1 \sqcap C_2 | C_1(x) \rightarrow C_2(x) |
| C_1 \equiv C_2 | C_1(x) \leftrightarrow C_2(x) |
| P(a, b) | P(a, b) |
| P_1 \sqcap P_2 | P_1(x, y) \rightarrow P_2(x, y) |
| P_1 \equiv P_2 | P_1(x, y) \leftrightarrow P_2(x, y) |

Table 1.1: Correspondence between FOL and DL
Chapter 2

Requirements

First, let us recall the goals for LKIF from the technical annex of the project [ESTRELLA, 2006, p. 3]:

The main technical objectives of the ESTRELLA project are to develop a Legal Knowledge Interchange Format (LKIF), building upon emerging XML-based standards of the Semantic Web, including RDF and OWL, and Application Programmer Interfaces (APIs) for interacting with LKIF legal knowledge systems. The LKIF will apply the state of the art in the field of Artificial Intelligence and Law, taking into account business and application requirements. Existing Semantic Web initiatives are aimed at modeling concepts (OWL “ontologies”) and rules (RuleML and SWRL). The LKIF will build on but go beyond this generic work to allow further kinds of legal knowledge to be modeled, including: meta-level rules for reasoning about rule priorities and exceptions, legal arguments, legal procedures, cases and case factors, values and principles. In addition, an OWL ontology of basic legal concepts, such as obligations, permissions, rights and powers, will be developed, which can be reused when modeling specific legal domain, such as tax law. A reference “inference engine” and run-time environment capable of processing knowledge bases using all the features of the LKIF will be implemented and validated in the pilot applications.

In this chapter, we will elaborate on these goals, by developing more detailed requirements for LKIF. These requirements are derived from several sources:

1. From a survey of research on computational models of legal reasoning and argumentation, from the field of Artificial Intelligence and Law.

2. From an analysis of the business requirements articulated by the participating vendors, drawing on their practical experience with real-world applications of legal knowledge systems, as well as from the logical reconstructions of the logics used by these vendors, as reported in the companion report, deliverable D1.2. [van den Berg et al., 2006].

3. From feedback and comments by the participating user organizations and members of the observatory board on earlier versions of this report.
2.1 Artificial Intelligence and Law

Edwina Rissland, Kevin Ashley and Ronald Loui published a good summary of the field of Artificial Intelligence and Law, as of 2003, in a special issue of the Artificial Intelligence Journal on AI and Law [Rissland et al., 2003]. A recent treatise on AI and Law is Giovanni Sartor’s “Legal Reasoning: A Cognitive Approach to the Law” [Sartor, 2005].

A major lesson from research on Artificial Intelligence and Law is that legal reasoning cannot be viewed, in general, as the application of some deductive logic, such first-order predicate logic, to some theory of the facts and relevant legal domain. In fact, no one in the field ever seriously took the position that legal reasoning in its entirety could be viewed this way, although some critics have misunderstood or misrepresented the field by assuming this to be the case. As pointed out by Rissland et al. [Rissland et al., 2003]:

Contrary to some popular notions, law is not a matter of simply applying rules to facts via modus ponens, for instance, to arrive at a conclusion. Mechanical jurisprudence, as this model has been called, is somewhat of a strawman. It was soundly rejected by rule skeptics like the realists. As Gardner puts it, law is more “rule-guided” than “rule-governed.”

The reference to Gardner here, is to Anne Gardner’s thesis “An Artificial Intelligence Approach to Legal Reasoning” [Gardner, 1987], one of the first books to be published in the field. Legal reasoning is not only deductive, because legal concepts cannot be defined by necessary and sufficient conditions. Better, one can define legal concepts this way, but such definitions are only hypotheses or theories which will not be blindly or “mechanically” followed, using deduction, when one tries to apply these concepts to decide legal issues in concrete cases. Legal concepts are, as the legal philosopher H.L.A. Hart put it, “open-textured” [Hart, 1961]. Whether or not a legal concept applies in a particular case requires the interpretation, or reinterpretation of the legal sources, such as statutes and case law, in light of such things as the history of prior precedent cases, the intention of the legislature, public policy, and evolving social values.

The process of determining whether the facts of a case can be “subsumed” under some legal concept is one of argumentation. Legal argumentation is a dialogue, guided by procedural norms, called “protocols”. Which protocol applies depends on the particular type of dialogue and the task at hand. Several of these “task orientations” are listed in Rissland et al.’s survey article [Rissland et al., 2003]:

- In advocacy, the task is to produce the best arguments possible for a particular side in the legal controversy.
- In adjudication, a court has the task of deciding the legal issues of a case and justifying this decision by publishing an opinion explaining his or her reasoning.
- In advising, planning and drafting, a lawyer assists a client with predicting the legal consequences of various alternative courses of action, and their risks and
In drafting this process results in some kind of legal document, such as a contract or will.

- In administration, a public agency has the task of applying public policy, laws and regulations to determine rights and benefits, in areas such as tax administration and social security.

In all of these tasks, argumentation plays an essential role. In advocacy, an attorney needs to construct, select and effectively present arguments promoting the interests of his client. When justifying its decision during adjudication, the court needs to consider and respond to the arguments put forward by the parties and present the arguments leading to his decision in a cogent and persuasive way. When advising a client, a lawyer needs to balance the pros and cons of various alternative courses of action. And, finally, in public administration a public agency is obliged to take all relevant legal sources into account when making its administrative decisions. Each legal source, be it case law, policy guidelines or legislation, is a source of legal arguments. These arguments can conflict and will need to be aggregated in some way. Moreover, public administrations do not make such decisions unilaterally, but in dialogues with the affected citizens and other stakeholders, who will put forward their own arguments which need to be taken into consideration.

For quite a while, the Artificial Intelligence and Law community viewed itself as being divided into two main “camps”: the case-based reasoning camp, mostly made up of researchers in the United States, and the “rule-based” camp, mostly made up of European researchers. This division was often explained by the different weight given legislation and cases by the common law and civil law legal traditions. This picture, however, was never completely accurate. Much early AI and Law research in the US applied formal logic to model legislation, including Gardner’s thesis [Gardner, 1987], which used Genesereth’s logic programming MRS logic programming language [Genesereth and Smith, 1982] to model legal rules from the Restatement of Contracts. And already in the 1950s, Layman Allen proposed the use of propositional logic to “normalize” legislation [Allen, 1957].

Conversely, the leading European researchers in AI and Law never proposed the use of classical logic to model legislation or legal reasoning. Rather, it was recognized very early that legal reasoning, unlike classical logic, is defeasible. This led to attempts to model legal reasoning using abduction [Gordon, 1991] and nonmonotonic logics [Gordon, 1987, Sartor, 1991, Prakken, 1991, Hage et al., 1993]. Jaap Hage’s work on “reason-based logic” is particularly interesting in retrospect, because it was one of the first to suggest that legal reasoning is an argumentation process, requiring the aggregation and weighing of arguments, or “reasons” as he called them, pro and con some conclusion.

In retrospect, it is becoming increasingly clear that US and European researchers shared a common view: that legal reasoning at its core is argumentation. Anne Gardner [Gardner, 1987, p. 3] wrote:

First, legal rules are used consciously by the expert to provide guidance in the analysis, argumentation, and decision of cases. Legal rea-
soning might thus be classified as a rule-guided activity rather than a rule-governed activity.

Second, and as a consequence, the experts can do more with the rules than follow them. In a field like contract law, where the rules have been developed mainly through decisions in individual cases, lawyers can argue about the rule themselves, and can propose refinements, reformulations, or even newly formulated rules to adapt the law to the case at hand. (Emphasis added.)

In [McCarty, 1997], Thorne McCarty, one of the founders of the field of AI and Law, makes it perfectly clear that argument was always central in his view of legal reasoning:

My own theory of legal argument dates back quite a number of years. It was first published in the collection of papers from the Swansea conference in 1979 . . . The problem all along has been to build a computational model of the arguments that occur in the majority and dissenting opinions in a series of corporate tax cases . . . beginning with Eisner v Macomber, 252 U.S. 189 (1920).

And Edwina Rissland and Kevin Ashley also viewed themselves as modeling legal argumentation in their research on case-based reasoning. The book version of Ashley’s thesis is entitled “Modeling Legal Argument: Reasoning with Cases and Hypotheticals” [Ashley, 1990].

Most research in AI and Law in the US in fact was not restricted to modeling arguments from case law, but rather aimed to account for how arguments from various sources, including statues as well as cases, can be combined and used together. This was the case for Gardner’s thesis [Gardner, 1987], which used cases to interpret open-textured concepts, when the “rules ran out.”, as well as for Rissland & Skalak’s CABARET system [Skalak and Rissland, 1992], which used cases to broaden and narrow the interpretation of legal rules, and Branting’s GREBE system [Branting and Porter, 1991]. Branting was one of the first, incidentally, in AI and Law to make explicit use of Toulmin’s model of argumentation, in particular his concept of a warrant [Toulmin, 1950].

For whatever reason, however, it wasn’t until two papers on computational models of legal argumentation in a special issue of the International Journal of Man-Machine Studies on AI and Law, by Gordon [Gordon, 1991] and by Routen and Bench-Capon [Routen and Bench-Capon, 1991] that argumentation became a hot topic in AI and Law and efforts began in earnest to use argumentation theory to integrate case-based, rule-based and other approaches to legal reasoning. The procedural aspects of argumentation, i.e. as a dialogue and not just a way of comparing pros and cons, began to come into focus, starting with Gordon’s Pleadings Game [Gordon, 1995]. A dialogical approach to integrating arguments from rules and cases was presented by Prakken and Sartor not much later [Prakken and Sartor, 1998].

AI and Law work on legal argumentation lead to some collaboration with experts in other fields interested in argumentation, in particular the Canadian philosopher Doug Walton, who is recognized as one of the leading experts on the philosophy of
argumentation. For example, Prakken and his colleague Bex worked with Walton on modeling arguments about legal evidence [Bex et al., 2003] and Gordon worked with Walton to develop a computational model of argument [Gordon and Walton, 2006a], called Carneades, and used this to model to reconstruct the arguments in the Pierson v. Post case, a popular benchmark in AI and Law [Gordon and Walton, 2006b]. This collaboration has been a two way street. Not only has the AI and Law community learned a great deal about argumentation theory, but Walton became interested enough in Artificial Intelligence and Law to write a book on legal argumentation informed by research results from the AI and Law field [Walton, 2005].

2.1.1 Introduction to Argumentation Schemes

One important insight from argumentation theory is the concept of an “argumentation scheme”. There are many different kinds of arguments and much research has gone into discovering and classifying various patterns of argument, based on an analysis of the structure and content of arguments reconstructed from natural language texts. Although they are the result of empirical case studies, they also have a normative side. They are a useful tool both for guiding the reconstructing of arguments put forward by other parties, so as to open them up to critical analysis and evaluation, as well supporting the construction (“invention”) of new arguments to put forward in support of one’s own claims or to counter the arguments of others.

Argumentation schemes generalize the concept of an inference rule to cover plausible as well as deductive and inductive forms of argument. Argumentation schemes are conventional patterns of argument, historically rooted in Aristotle’s “Topics” [Slomkowski, 1997]. Unlike inference rules, argumentation schemes may be domain dependent. Each scheme comes with a set of “critical questions” for evaluating and challenging arguments which use the scheme. For example, the scheme for argument from expert opinion includes a critical question about whether the expert is biased.

Since argumentation schemes may be domain dependent, there are an unlimited number of such schemes. Domain dependent schemes, in fields such as the law, may evolve along with the knowledge of some domain. Many schemes, however, are general purpose. Walton and his colleagues have taken on the project of collecting and classifying general purpose schemes. To date their collection contains about 60 schemes. Examples include Argument from Expert Opinion, Argument from Popular Opinion, Argument from Analogy, Argument from Correlation to Cause, Argument from Consequence, Argument from Sign and Argument from Verbal Classification.

We will present some specifically legal argumentation schemes, and prior work on modeling these schemes computationally within the field of AI and Law, later in this section. Before doing so, however, let us address the relevance and importance of this work for LKIF.

Argumentation Use Cases

Now that we understand that legal reasoning is based on arguments and argumentation, the question for us in ESTRELLA is what requirements does this raise for representing legal knowledge in LKIF? To understand how legal knowledge is used
in legal argumentation, let us take a look at how various argumentation tasks fit together. Based on prior analyses of argumentation tasks and their interrelationships [Brewka and Gordon, 1994, Prakken, 1995, Bench-Capon and Sartor, 2003, Gordon, 2003], we can distinguish the following three layers:

1. The “logical layer” is responsible for representing statements and argumentation schemes into order to construct or generate arguments by applying argumentation schemes to a “knowledge base” of statements.

2. The “dialectical layer” is responsible for structuring, evaluating and comparing arguments which have been put forward during the dialogue, distributing various burdens of proof, managing commitments and applying dialogue protocols to guide, moderate and facilitate the process. We also will include the task of “reconstructing” arguments from natural language texts in this level.

3. Finally, the “rhetorical layer” is responsible for helping participants to “play the game” well. Whereas the dialectical layer facilitates the normative goals of the dialogue, this layer provides a private advisor to each participant, to help participants protect and further their own interests. Tasks here include selecting among arguments which could be made and presenting these arguments clearly and persuasively, taking into consideration the intended audience, perhaps using argument visualization techniques. We have also placed in this layer the task of authorities, such as courts or public agencies, to make and justify decisions.

Figure 2.1 is a “use case” diagram showing these tasks, divided into the above layers, together with the abstract roles responsible for each task. In concrete situations, one person may have more than one role, some roles may be combined or some roles may need to be distinguished further. For example, in lawsuits the judge may have the moderator role and share the authority role with a jury.

The legal knowledge we want to model with LKIF is to be used in the first layer, the “logical” layer, for generating arguments using argumentation schemes. Logic is understood very broadly here, to cover all forms of inference and reasoning patterns: deductive, abductive, inductive, plausible, analogical, etc. The theory of argumentation schemes is intended to have this generality; argumentation schemes have been developed to be an abstraction sufficient to cover all conceivable forms of inference.

An LKIF knowledge base needs, as a minimum, to provide a representation of all the information needed for applying the argumentation schemes to be supported. Ideally, of course, LKIF would be sufficient for all argumentation schemes, both general purpose and specifically legal. From a practical standpoint, it will be necessary to start with some selection of argumentation schemes and continue to develop and evolve future versions of LKIF to handle additional schemes, as the need arises.

The working hypothesis of ESTRELLA is that description logic, in the form of the Web Ontology Language (OWL), is sufficient for the purpose of modeling and representing the information needed by legal argumentation schemes. As we
know, description logic is semantically (not syntactically) a subset of first-order predicate logic (FOL). That is, description logic inherits FOL’s classical, model theoretic semantics. All description logic entailments are deductively valid in FOL. That is, description logic, like FOL, does not itself provide any support for the forms of plausible inference required for legal reasoning.

Arguably, however, it is not necessary for the knowledge representation language itself to support plausible inference. Its job is only to serve as a kind of “deductive database” for retrieving information needed for applying argumentation schemes. It is the argumentation schemes, together with the other components of the layered model of argumentation, which are responsible for supporting and evaluating plausible inferences, in the form of defeasible arguments.

Description logic is a decidable subset of FOL and thus relatively weak in terms of the inferences it sanctions. But this limitation does not appear problematical, so long as description logic is only be used to implement a deductive database to be used by the computational models of the legal argumentation schemes.

An interesting question is whether LKIF should be expressive enough to model the argumentation schemes themselves, as well as the data required by the argumentation schemes. This would require some kind of argumentation scheme interpreter or inference engine for the LKIF representation of argument schemes. This would be an elegant and powerful device, if it is feasible. But this raises research issues which have yet to be addressed in the Artificial Intelligence and Law field, let alone solved. There are to our knowledge no proposals yet for a language capable of representing a variety of argumentation schemes.

2.1.2 Legal Argumentation Schemes
**Argument from Cases** The topic of case-based reasoning in AI can be understood as attempts to construct computational models of the scheme for arguments from analogy and related schemes. The first research on case-based reasoning was probably within the interdisciplinary field of AI and Law, at around the time the field was forming in the late 1970s. McCarty’s TAXMAN model [McCarty, 1977] was one of the seminal works in the field. McCarty’s approach to case-based reasoning, based on the idea of constructing and comparing theories of a line of cases, was much ahead of its time. According to this theory-construction model, the better arguments from cases are the ones based on the better, i.e. more “coherent”, explanatory theory of those cases.

Probably the most influential computational model of case-based reasoning is Ashley’s HYPO model [Ashley, 1990]. In HYPO, cases are represented as a set of “dimensions”, where each dimension includes information about which party is favored in each direction of the dimension. For example, in the trade secrets domain, the dimension of disclosure favors the defendant, i.e. the party who allegedly violated a trade secret, the more the plaintiff company has disclosed the (so-called) secret to third parties. HYPO formalized the relation of “on-pointedness”. One precedent case is more “on-point” than another precedent case if the first case has more dimensions in common with the current case. Arguments were constructed in HYPO by searching for analogous cases, cases with dimensions in common with the current case, which had been decided in favor of the desired party, plaintiff or defendant. (This depended of course on the role of the party trying to construct the argument.) The other party can then try to construct counterarguments, either by distinguishing the current case from precedent case, i.e. by pointing out differences between the two cases, or by searching for more on-point cases in his favor. HYPO is named after its model of reasoning with hypothetical cases. Hypotheticals are imaginary cases, constructed for the sake of argument. Typically, they are variations of the current case, constructed to test proposed interpretations of legal rules or principles. For example, if a party proposes some rule, perhaps by generalizing the decision of some precedent case, the other party could try to construct a hypothetical case showing that this proposed rule leads to some unintuitive or otherwise undesirable result.

The CATO model of case-based reasoning [Aleven and Ashley, 1997], both simplified and extended the HYPO model. It simplifies HYPO by replacing dimensions by boolean “factors”, i.e. propositions which are either true or false in a case. But CATO extends HYPO by organizing these factors into a hierarchy and using this hierarchy to support additional case-based argumentation schemes, in particular schemes for arguments from “downplaying” and “emphasizing” distinctions. A distinction between a precedent case and the current case is downplayed by showing that factors present in both cases have a common ancestor in the hierarchy and arguing that the precedent case is more general, applying to all cases in which this more abstract, common factor is present. For example, if the precedent case involved deception but the current case bribery, one might downplay the distinction between deception and bribery by noting they are both illegal means of obtaining information and arguing the precedent applies to all such illegal means, not just deception.

Other influential models of case-based reasoning in the AI and Law field include GREBE [Branting and Porter, 1991, Branting, 2000] and CABARET [Skalak and Rissland, 1992].
GREBE used semantic networks, the forerunner of ontologies modeled using Description Logic, currently popular in the context of the Semantic Web, in its model of case comparison. CABARET modeled the use of cases to construct arguments about open-textured concepts and included models of argumentation schemes for broadening and narrowing the application of legal rules using cases. Both GREBE and CABARET were early attempts to model an argumentation framework in which argumentation schemes for arguments from both rules and cases could be used together, in an integrated fashion. See also [Prakken and Sartor, 1998].

Gardner’s early model of legal reasoning [Gardner, 1987] also needs to be mentioned in this context. Although it was primarily a model of schemes for arguments from rules, it also included scheme for arguments from cases, called interpretation rules, which were applied to open-textured concepts, “when the rules ran out”.

Loui and Norman [Loui and Norman, 1995] developed a computational model of another case-based argumentation scheme, for arguments from the “rationale” of the case. The scheme exposes presuppositions of the rationale of a case and then argues that these presuppositions do not apply in the present case. For example, in a precedent case which decided that vehicles are not allowed in public parks, there may be a presupposition that the vehicles in question are privately owned. If in the current case the vehicle is not privately owned, this argumentation scheme could be applied to construct an argument that the precedent does not apply.

Finally, Bram Roth developed a model of case-based argumentation which also makes use of rationales, represented as reconstructions of the dialectical structure of the arguments in the published opinions of the cases [Roth, 2003]. In Roth’s account, the arguments in a precedent case are applied to the facts of the current case. If the current facts provide at least as much support for the conclusion of the precedent case, considering its arguments, then the conclusion of the precedent case presumptively also applies to the current case. The scheme modeled by Roth is known as argument “a fortiori” (from the stronger argument).

**Argument from Rules** Next we want to address computational models of schemes for arguments from defeasible rules or, as Walton calls them, defeasible generalizations. In the law, the idea of applying rules, by trying to “subsume” the facts of the case under the legal terms of the rule, is quite basic. In the legal philosophy known as “mechanical jurisprudence”, this process was thought to be purely deductive. In some early work in the field of AI and Law, this same insight led to experiments with using theorem provers or rule-based systems to build legal expert systems based on first-order logic [Sergot et al., 1986].

This approach is adequate for some application scenarios, especially in public administration, and is the basis for most commercial legal knowledge systems today, including those developed by the vendors in the ESTRELLA project. But models of rules based on classical logic are not well suited for capturing the defeasibility of arguments from rules, since legal concepts are open-textured, rules can be subject to exceptions, overridden by others rules, or invalid.

Nonmonotonic logics have been developed in AI to model reasoning with defeasible rules, but typically these logics do not address the issue of how to integrate reasoning with defeasible rules with other forms of plausible or presumptive rea-
soning, such as case-based reasoning. Argumentation-theoretic models of reasoning with defeasible rules can overcome these limitations. One influential argumentation-theoretic model of arguments from rules is Hage and Verheij’s “Reason-Based Logic” [Hage et al., 1993, Verheij, 1996, Hage, 1997]. Other models of defeasible arguments from rules were developed by Gordon [Gordon, 1995], as part of his Pleadings Game model of legal argumentation, and Prakken and Sartor [Prakken and Sartor, 1996].

When arguments conflict, some way is needed to resolve these conflicts. Some models of argumentation include a “built-in” method for resolving these conflicts. For example, several models always prefer arguments from cases to arguments from rules [Gardner, 1987, Branting, 2000, Skalak and Rissland, 1992]. Similarly, some nonmonotonic logics, such as Conditional Entailment [Geffner and Pearl, 1992] always prefer the more specific argument. A more general solution is to support argumentation about argument priorities or strengths [Gordon, 1995, Prakken and Sartor, 1996, Hage, 1996, Verheij, 1996, Kowalski and Toni, 1996]. These priority arguments may apply higher-level principles, such as lex superior (prefer the rule from the higher authority), lex posterior (prefer the newer rule) and lex specialis (prefer the more specific rule), which may themselves be defeasible.

**Argument from Values, Purpose, Goals and Policy** In [Bench-Capon, 2002], Bench-Capon analyzed the role of purpose (“teleology”) when interpreting a body of case law, motivated by the seminal paper by Berman and Hafner [Berman and Hafner, 1993], which identified limitations of the HYPO approach to case-based reasoning in the law.

Bench-Capon’s central idea is that the rules and rule preferences cannot be derived solely from factors in precedent cases, but must also be informed by the purposes of the rules, i.e. by the values promoted by the rules. Shortly thereafter, Bench-Capon, in collaboration with Sartor, developed this basic idea into a theory-construction model of legal argument [Bench-Capon and Sartor, 2003]. In this model, legal theories are constructed from precedent cases in a process which takes values and value preferences into consideration to derive and order rules, which may then be applied to the facts of cases to reach decisions.

This theory construction approach, first advocated in AI and Law by McCarty [McCarty, 1977], can be viewed as a complex argumentation scheme. The scheme is complex compared to other case-based schemes, because it depends not only on the features of a single case, but rather an analysis of a whole line of cases. The idea of the scheme is to construct a theory capable of explaining the decisions in this line of cases and then to apply this theory to the facts of the current case.

Of course several competing theories are possible. Thus the scheme can produce several competing arguments. The conflict between these arguments is resolved by comparing these theories: the better the theory, the better the argument. Which theory is better, or most “coherent” can be debatable and thus an issue to be addressed by further argumentation. There are different criteria for evaluating the quality of theories. Bench-Capon and Sartor have addressed the issue of how to define and model coherence [Bench-Capon and Sartor, 2003]. And they have proposed quantitative metrics of theory coherence [Bench-Capon and Sartor, 2001]. See also [Hage, 2001].
Atkinson (formerly Greenwood), Bench-Capon and McBurney [Atkinson et al., 2005] did further work on modeling teleological reasoning in the law, in which they develop a formal model of the argument scheme for practical reasoning, based on Bench-Capon’s and Sartor’s Value-Based Argumentation Framework [Bench-Capon and Sartor, 2003].

**Argument from Evidence**  Computational models of argumentation schemes for reasoning with evidence have long been neglected. One of the first models [Lutomski, 1989], represented a number of argumentation schemes for arguments from statistical evidence in the domain of employment discrimination law, including critical questions. Chris Reed, Douglas Walton and Henry Prakken have more recently been working together on computational models of arguments from evidence [Prakken et al., 2003, Bex et al., 2003, Prakken, 2004], based on John Pollock’s work on a scheme for argument from perception and related schemes [Pollock, 1987].

**Requirements from Legal Argumentation Schemes**

From the legal argumentation schemes which have been investigated in the field of AI and Law, summarized above, we can derive the following requirements for concepts which must be representable in LKIF.

- **Case factors and factor hierarchies**, needed by the HYPO/CATO style of argument from analogy. Ideally, LKIF would also support case *dimensions* as well. Factors are boolean attributes of cases which lend support to some conclusion; Dimensions are scalar attributes which lend support to some conclusion. Thus both factors and dimensions can be modeled as a binary relation between an attribute and a statement.

- All case-based argumentation schemes operate on a body of precedent cases, i.e. on a *set of cases*. To support case-based reasoning, LKIF must allow a whole database of cases to be represented within a single LKIF knowledge base. Some of these cases will be *hypotheticals*; other will be *precedent cases*.

- The *on-pointedness* relation among cases is needed for resolving conflicts among precedent cases using the HYPO style of case-based reasoning.

- Many legal argumentation schemes operate on rules. Some argumentation schemes operate on both rules and cases. Thus, rules must be *reified* in LKIF. They must be modeled, not as formulas, but as objects with properties and relations to other objects. Example properties include various dates, such as the date of enactment, period of *validity* and date of repeal, the *authority* which issued the rule, the *backing* of the rule (i.e. a reference to the legal source), *applicability* and *exclusionary conditions* or *exceptions*. Monica Palmirani and her colleagues at the University of Bologna have done some research in connection with Estrella on the temporal properties of legal rules [Palmirani and Brighi, 2006].

- An important relation for defeasible reasoning is the *priority relation* between rules. This priority relation can be derivable from the properties of the rules,
using higher-level legal principles, such as lex posterior or lex posterior. It must be possible to represent such principles about rule priorities in LKIF, as well as other legal principles, such as estoppel.

- To support the theory-construction style of case-based reasoning, LKIF would have to allow multiple theories of a body of case law to be representable in a single LKIF knowledge base. Moreover, the concept of theory coherence must be modeled in some way, to enable arguments about which theory to prefer when alternatives have been proposed. It is not clear that coherence needs to be modeled explicitly in an LKIF knowledge base. Coherence could be a property derivable from other information in the LKIF knowledge base. Perhaps the models of schemes for argument from theory construction should be responsible for defining coherence.

- To reconstruct the McCarty style of legal argument from theory construction, LKIF would need to support prototypes and deformations of cases.

- To support theory construction arguments from rule preferences and social values, as in Trevor Bench-Capon’s model of argument from theory construction, LKIF would need to support the representation of values and value preferences. (Rule preferences have covered previously.)

- To support arguments from practical reasoning, as in Atkinson’s work, LKIF would need to support the modeling of goals and actions, as well as the values already mentioned.

- To aggregate, evaluate and compare arguments, it would be useful if LKIF included some concept of statement and of conflict between statements. This would allow the concept of pro and con arguments to be modeled at the dialectical level.

- One question is whether arguments themselves, and relations among arguments (such as attack or defeat relations), need to be representable in LKIF. This could be the responsibility of the dialectical layer, rather the the logic layer of which LKIF is a part. However, some argumentation schemes may be viewed as arguments about other arguments. Arguments from rationales, as studied for example by Ronald Loui and his colleagues [Loui and Norman, 1995] and Bram Roth [Roth, 2003], are of this kind. The “rationales” in this form of argumentation can be viewed as arguments, or argument graphs. To be able to handle such argumentation schemes, LKIF would have to support the representation of arguments and argument graphs.

- To support arguments from evidence, LKIF would of course have to allow evidence to be represented. It may be that evidence is a relationship between an agent, some object and an event, representing the fact that the agent has proffered the object as evidence in some legal proceeding.
2.1.3 Normative Reasoning

In the previous sections it has been explained why the view that each normative statement can be treated the same from a logical point of view is can be considered mechanical jurisprudence. The existing, extensive, literature on deontic logic, the field of logic that deals with normative reasoning should nevertheless be taken into account, and developers of deontic logics for the legal field should be encouraged to use LKIF for defining logics. This section deals with why no complete deontic logic can be part of LKIF.

Norms guide or mandate conduct in a given type of situation, but this is not the only possible way to explain their purpose. The following rationales are usually associated with the use of norms to guide conduct:

**Accumulating Knowledge** Accumulated knowledge from conduct in the past is written down in the form of norms to guide conduct in the future. The norms set out the relevant criteria, guide collection of information, decrease the amount of mistakes, and generally allow relatively stupid people to solve complex social problems.

**Consistency** Writing down norms supposedly encourages consistency, fairness, equality of treatment of persons, groups, and organizations in different places and at different times. Norms limit the discretion of the decision maker in treating specific persons, groups, and organizations differently, and therefore reduce bias and corruption. Written norms also make behaviour of others predictable, which reduces conflicts.

**Democracy** The process of writing down norms allows for greater public involvement than the mere making of a decision. The use of norms is in this sense a precondition for effective democracy on a large scale.

**Legitimacy** Written norms contribute to the perceived legitimacy of decisions, because of the reasons above, and allow the decision-maker to cite the source to justify the decision.

In the previous section the use of norms to justify has been discussed at length in the context of argumentation. This section deals with a certain approach to explaining how norms guide conduct.

The orthodox treatment of both monadic and dyadic deontic logic fails to deal with what deontic concepts actually represent. Deontic logics explain certain properties of a family of axiological concepts centering on subjunctive betterness (cf. [Dayton, 1981, Makinson, 1999]).

Subjunctive betterness is however no single concept for which a single logic exists. There are a variety of such concepts which are related to each other in ways explicable within a larger theory of rational behaviour, but not within a deontic logic itself. As a result any deontic logic is bound to depend upon particular normative theses limited to particular types of problems, particular circumstances, or for instance a particular style of legislating.
Obligation is traditionally represented by an operator $O$, "it ought to be that". The modal interpretation of $O$ calls for a nonempty class of ideal worlds (or deontic alternatives) related to each possible world in such a way that what ought to be in a given world actually comes about in its deontic alternatives. This ideal standard for a given world is not a deontological one: worlds are ideals in the sense of being subjunctively best worlds. Thus if $O\alpha$ is true at a world, it means that it would better if $\alpha$ were true at that world. It does not mean that it is obligatory for anyone to do something which might make $\alpha$ true. It does not deal with the circumstance that the only way to make $\alpha$ true involves forbidden actions, that making $\alpha$ true is impossible, or that although bringing $\alpha$ about would certainly be better, it is not my obligation but somebody else’s.

The basic intuition behind a dyadic $O$ operator also involves appeal to the idea of the subjunctive better. $O(\alpha | \beta)$ is true at a world if the condition given by $\beta$ determines some non-empty class of deontic ideals in which $\alpha$ is true. It means that given that $\beta$, it would be better if $\alpha$ were true. It does not follow from this that given that $\beta$ it is obligatory for anyone to do something which would bring about $\alpha$. In the literature this distinction is known as ough-to-be vs. ought-to-do (cf. [d’Altan et al., 1996]), and they are widely assumed to be mere reformulations of the same thing. The assumption that if it ought to be that $\alpha$ (given $\beta$) is equivalent to saying that it would be better if $\alpha$ were true (given $\beta$) is built into all orthodox deontic logics.

Obligations express not the goodness of the action but the imperativeness of the decision to perform the action referred to as obligatory. Interpretation of the violation of the obligation cannot be separated from a theory of intelligent decision making that answers questions such as what was within the power of the decision maker, what did the decision maker intend, what did he foresee, and what did he try to do?

Deontic logics do not explain obligations in terms of decision making, and therefore cannot be considered viable candidates for a generic theory explaining decision making – or argumentation – in law. We must at the same time recognize that law is not the only field concerned with decision making. It would be unrealistic to require that a standard for legal knowledge representation has to standardize knowledge representation of epistemology, and the standard cannot succeed if it does prescribe such a theory.

The best the standard can achieve is to explain deontic concepts in terms of betterness, of intention, of beliefs, of decisions, of violations, of sanctions, etc. This is a purely terminological approach. This results in a sort of logic, described in section 6.1, but not one that is sufficient for solving legal cases.

**Subjunctive Betterness**

Because betterness is central to the field of deontic logic this section treats it separately. From this treatment several requirements on LKIF follow. Central to betterness is the notion of choice:

---

1. This clearly separates legal obligation from ends-oriented accounts of morality.
Observation 1. Deontic choice \( O(\alpha | \beta) \): if an agent has the choice between \((\alpha \land \beta)\) and \((\neg \alpha \land \beta)\) then the agent should choose \((\alpha \land \beta)\).

Let \( \beta \) be a situation, and \( \alpha \) an alternative in a menu. Beware of interpreting \( \alpha \) as an action: the alternatives may concern both descriptions of actions and situations, as long as situations can be conceived of as productive characterizations in the sense that social and legal norms only speak about situations brought about by intelligent human action. The deontic operators are reduced to preference statements as follows:

\[
O(\alpha | \beta) : \beta,\alpha \succ \beta,\neg \alpha \\
F(\alpha | \beta) : \beta,\neg \alpha \succ \beta,\alpha \\
P(\alpha | \beta) : \beta,\alpha \succeq \beta,\neg \alpha
\]

These can form the basis of preference-based reasoning system that can meet at least the following desirable characteristics of a deontic knowledge representation:

Observation 2. What is obligatory is permitted: \( O(\alpha | \beta) \to P(\alpha | \beta) \)

Observation 3. The impossible and the meaningless are not obligatory: \( \neg O(\alpha | \alpha) \) and \( \neg O(\neg \alpha | \alpha) \) are axioms.

Observation 4. There are no conflicting obligations. The obligations \( O(\alpha | \beta) \) and \( O(\neg \alpha | \beta) \) are inconsistent: \( \neg (O(\alpha | \beta) \land O(\neg \alpha | \beta)) \) is an axiom. Idem for \( O(\alpha | \beta) \) and \( P(\neg \alpha | \beta) \).

Observation 5. If \( \phi \) is the set of worlds \( w \) such that \( M,w \models \phi \), then the sentences \( O(\alpha | \top) \), \( O(\beta | \alpha) \), \( O(\neg \beta | \neg \alpha) \) are only satisfied by the ordering \( \neg \alpha \land \beta \succ \neg \alpha \land \neg \beta \succ \alpha \land \neg \beta \succ \alpha \land \beta \).

A similar characterization can be given that depends on a functional mapping from propositions to the set \{allowed, disallowed, silent\} as in [Valente and Breuker, 1995]:

\[
O(\alpha | \beta) : f(\{\beta,\alpha\}) = allowed, f(\{\beta,\neg \alpha\}) = disallowed \\
F(\alpha | \beta) : f(\{\beta,\neg \alpha\}) = allowed, f(\{\beta,\alpha\}) = disallowed \\
P(\alpha | \beta) : f(\{\beta,\alpha\}) = allowed, f(\{\beta,\neg \alpha\}) = silent
\]

This representation is less powerful for a number of reasons. For one, it cannot be used to detect contrary-to-duty obligations.

One of the attractive features of the representation in the form of preferences is that it produces triangles between a ‘context’ \( \beta \), and two good \((\alpha \land \beta)\) and bad \((\alpha \land \beta)\) alternatives that are complete partitions of the context. It naturally fits in a graphical representation of taxonomies, and knowledge acquisition methods like the repertory grid.

A point of contention may be the translation of a permission to a statement of qualified indifference \((\alpha \succeq \neg \alpha)\). It is a weakening of \((\alpha = \neg \alpha)\). A permission for \( \alpha \) to be the case appears to imply a permission for \( \neg \alpha \) to be the case, and this seems
too strong since an intuition among deontic logicians is that if \( O(\alpha) \) then \( P(\alpha) \) and \( F(\neg\alpha) \). If something is obliged, then it should also be allowed. The asymmetric statement \( (\alpha \geq \neg\alpha) \) leaves room for a prohibition or obligation \( (\neg\alpha \succ \alpha) \) without creating a conflict, and retains the information that the permission was about \( \alpha \) and not \( \neg\alpha \).

It is by the way not clearcut that the intuition of the deontic logician is correct in contexts outside of legislation. It is perhaps caused by extended exposure to alethic modal logics. There is ample evidence that points the other way. For instance:

- it is an intuition that stating that \( \alpha \) is allowed while it is already understood that \( \alpha \) is obliged is at least an odd and confusing use of language,
- it is also an intuition that a norm system that only contains \( P(\alpha|\top) \) is not a well-formed norm system at all since it serves no purpose in guiding and evaluating behaviour,
- it is also an intuition that operator \( P \) serves no real purpose in a deontic reasoning system (like SDL) that does not allow for conflicts between norms in guiding and evaluating behaviour,
- and it is a fact that people generally attribute an attitude of indifference towards \( \alpha \) and \( \neg\alpha \) to others who express \( P(\alpha|\top) \). At least children and child psychologists do it in [Keller et al., 2004].

The operator \( P \) cannot be understood in any other way than a superfluous utterance stating indifference towards \( \alpha \) and \( \neg\alpha \) if it is evaluated without a context. The \( P \) only becomes relevant if:

- it interferes with one or more obligations or prohibitions,
- and it is used in a reasoning system that chooses between incompatible deontic statements, and for some reason prefers the permission.

The explicitly stated permission clearly has another function than the derived permission following from the absence of prohibition. Since permissions are usually uttered with the explicit intention of amending a specific obligation, often to be found nearby in the same legislative text, it makes sense to ‘localize’ them to some extent by adding the asymmetry. The asymmetry is also necessary for using the representation in combination with common deontic reasoning systems.

Some broad permissions can be interpreted as symmetrical. It is for instance generally accepted that freedom of expression includes the freedom to keep your opinion to yourself, and freedom of religion includes both the freedom to exercise (any) religion and the freedom to reject religion altogether. On the other hand a freedom from slavery does not typically include the right to sell yourself into slavery, so we cannot say that ‘rights’ in general include a symmetrical permission\(^2\). We might add a fourth dyadic operator for freedom of choice (liberty) with limited application:

\(^2\)In addition to correlated duties to refrain from infringing on the right.
\[ L(\alpha|\beta) : \beta,\alpha = \beta,\neg\alpha \]

The purpose of this characterization is not to invent yet another deontic logic, but to give general constraints on a representation. The advantage of this characterization is that it fits the monotonic description logic that OWL and indirectly the Semantic Web are based on. The problem is that we need an external mechanism to deal with conflicts, or cycles in the preference ordering.

**Conflicting Normative Statements**

The main reason for translating deontic norms to preferences between the thing regulated and its logical complement is that it allows for more complex ordering of alternatives to choose from in norm systems with interacting norms. Unfortunately it is the case that norm systems sometimes impose preferences on alternatives that are incompatible.

As we will see this is not unreasonable, and there are rules for resolving such circularities. The legislator does not usually include pairs of directly opposite norms like \( O(\alpha) \) and \( P(\alpha) \), but pairs \( O(\alpha) \) and \( P(\alpha') \) where \( \alpha' \) is subsumed by \( \alpha \), or \( \alpha' \sqsubseteq \alpha \), do occur quite regularly. Because subsumption relations play a key role in this discussion, we will introduce an ad hoc graphical convention for displaying a subsumption relation \( \alpha' \sqsubseteq \alpha \):

\[
\begin{array}{c}
\alpha \\
\uparrow \\
\alpha'
\end{array}
\]

We have to introduce another notational convention to make clear that a statement \((\beta,\alpha) \succ (\beta,\neg\alpha)\) is an ordering on \( \alpha \) and \( \neg\alpha \) within the context of \( \beta \) for pragmatic purposes:

\[
\beta : \begin{array}{c}
\alpha \\
\uparrow \\
\neg\alpha
\end{array}
\]

An instrumental concept in explaining conflicts between norms is that of realizability; A norm is realized if the state of affairs it permits or mandates is the case. There is a conflict between a pair of norms if they are not jointly realizable.

A distinction is usually made between so-called conflicts of disaffirmation and compliance conflicts. Lindahl (cf. [Lindahl, 1992]) defines disaffirmation as follows: “a relation between two norms of different deontic mode, one being permissive and the other mandatory”.

A disaffirmation conflict is in our context a circularity between a permission and either an obligation or prohibition. This occurs if a state of affairs is simultaneously permitted and prohibited. The first such situation – a conflict of disaffirmation between \( O(\alpha|\beta) \) and \( P(\neg\alpha'|\beta') \) where \( \alpha' \sqsubseteq \alpha \) and \( \beta' \sqsubseteq \beta \) – can be displayed as follows:

\[
\beta : \begin{array}{c}
\alpha \\
\uparrow \\
\neg\alpha
\end{array}
\]

\[
\beta' : \begin{array}{c}
\alpha' \\
\downarrow \\
\neg\alpha'
\end{array}
\]
We call this a disaffirmation of an imperative. Intuitively it is meant as an exception and takes precedence to the primary obligation, but this is not necessarily the case as we will see in section 2.1.3. A simple example:

Parking is prohibited in this neighbourhood. License holders are excepted.

The second such situation – a conflict of disaffirmation between $P(\alpha|\beta)$ and $O(\neg\alpha'|\beta')$ where $\alpha' \subseteq \alpha$ and $\beta' \subseteq \beta$ – can be displayed as follows:

$$
\begin{array}{c}
\beta : \\
\uparrow \\
\beta' : \\
\end{array}
\begin{array}{c}
\alpha \\
\downarrow \\
\alpha' \\
\downarrow \\
\neg\alpha \\
\downarrow \\
\neg\alpha' \\
\end{array}
$$

We call this a disaffirmation of a permission. Intuitively this case should be handled in the same way as the previous one, with the disaffirming norm taking precedence. A simple example:

Parking is allowed in the parking bays. Exception for cars wider than 1.85m.\(^3\)

Disaffirmation of permissions and imperatives is a very useful tool for legislators. It allows the legislator to create exceptions to provisions, without amending the provision with yet another sentence fragment. Since sentences in legislation are already too long as it is, according to many, this is certainly a useful thing. Disadvantage of doing this is that it makes it harder to apply provisions, because the task of checking whether there are any such exceptions in the legal corpus is left as an exercise to the reader.

Another purpose of the disaffirmation is to influence the legislation of another legislator without actually touching its legislation. If legislator A is superior to legislator B, a disaffirmation in the legislation of A will effectively amend the legislation of B. How this works is explained in section 2.1.3. Because the amendment now ends up in a completely different part of the corpus, this type of exception will be hard to find.

This does not mean that disaffirmation conflicts are always merely a design feature of legislation, or that the disaffirmation conflict is the same thing as an exception. There are several other types of disaffirmation conflicts that have no intuitive solution, or clear purpose. The first such situation – a conflict of disaffirmation between $O(\neg\alpha|\beta')$ and $P(\alpha'|\beta)$ where $\alpha' \subseteq \alpha$ and $\beta' \subseteq \beta$ – can be displayed as follows:

$$
\begin{array}{c}
\beta : \\
\uparrow \\
\beta' : \\
\end{array}
\begin{array}{c}
\alpha \\
\downarrow \\
\alpha' \\
\downarrow \\
\neg\alpha' \\
\end{array}
$$

The following two simple rules exemplify this case:

\(^3\)Inspired by the SUV amendment of 20 oktober 2004 of the town council of Nijmegen.
1. Using network facilities in the classrooms is prohibited.

2. Using WiFi in the university building is permitted.

It is not clear which of the two rules is an exception to the other one, granted that we share the belief that using WiFi is subsumed by using network facilities, and the classrooms are subsumed by the university building. The second such situation is created by reversing the deontic modalities in the example:

1. Using network facilities in the classrooms is permitted.

2. Using WiFi in the university building is prohibited.

This is a conflict of disaffirmation between $O(\lnot \alpha'|\beta)$ and $P(\alpha|\beta')$ – where $\alpha' \sqsubseteq \alpha$ and $\beta' \sqsubseteq \beta$ – and it can be displayed as follows:

$$
\begin{array}{c}
\beta: \\
\uparrow \\
\beta': \\
\downarrow \\
\end{array}
\begin{array}{c}
\lnot \alpha' \\
\lnot \alpha \\
\alpha' \\
\alpha \\
\end{array}
$$

These situations are certainly conflicts from a *joint realization* point of view, and would certainly have been recognized as such by Lindahl (viz. [Lindahl, 1992]) or Hill (viz. [Hill, 1987]). Arguably they belong to the Hill’s “intersection” conflicts (cf. [Elhag et al., 1999, Hill, 1987]). These cases defy Valente’s formalization of normative conflicts (cf. [Valente and Breuker, 1995]) because they rely on the distinction between context and that which is required that is typical for an ought-to-do perspective on norms (viz. []). The reader can decide for himself whether this is a genuine conflict. Hill’s “intersection” conflicts (cf. [Elhag et al., 1999, Hill, 1987]) are however too crude as a concept: one cannot generally claim a normative conflict between norms that are only tangentially related.

A special case is the *explicit disaffirmation*: the permission and the imperative disaffirm each other directly. This obviously happens if and only if $\alpha' = \alpha$ and $\beta' = \beta$. All four of the previous characterizations are applicable to this pair of norms. Intuitively this does not make sense at all. It is however in principle valid, provided that the two norms have been enacted by different legislators that are apparently disputing jurisdiction.

As we will see, the compliance conflict is even stranger. The compliance conflict is defined by Lindahl as a relation between two mandatory norms, both of which are individually realizable, but not jointly realizable. Simply put, the compliance conflict gives you a choice where none of the options is allowed, or all of them are obliged.

This situation – a *conflict of compliance* between $O(\alpha)$ and $O(\lnot \alpha')$ where $\alpha' \subset \alpha$ – can be displayed as follows:

$$
\begin{array}{c}
\alpha \\
\uparrow \\
\alpha' \\
\downarrow \\
\lnot \alpha \\
\lnot \alpha' \\
\end{array}
$$
An interesting real-world example is when the Amsterdam police ordered nightclub owners to lock the emergency exits to keep drugs out, while the fire department ordered the same nightclub owners to unlock them to allow for escape in case of disaster. Whether this is a compliance conflict depends on how we conceptualize the options of the nightclub owners. Strictly speaking, the nightclub owners did have the possibility of admitting much less customers, or even to stop running a nightclub. If regulations on running nightclubs are so restrictive that it is in fact impossible to run a nightclub, does that mean that therefore running a nightclub effectively becomes prohibited? Do these rules then form a disaffirmation of a permission (or permit) to run a nightclub?

The treatment of compliance conflicts in actual reasoning systems is for this reason never really adequate, because the judgment call depends on a potentially huge amount of external knowledge.

As argued in for instance [Boer, 2000], it is also extremely useful to recognize affirmation of an imperative and affirmation of a permission as a tool for structuring legislation. These closely related concepts are also based on subsumption, but do not involve a circularity. An affirmation of an imperative assumes two norms of the same mandatory deontic mode in a subsumptive relationship, and affirmation of a permission involves two norms of permissive mode.

An imperative for designing ships for instance states that equipment of any kind on ships should designed “in such a way as to minimize potential damage of mishandling”. This norm is also affirmed in many places by other, more detailed, imperatives relating to specific bad user interfaces of specific types of equipment. It is for instance not allowed to use a single piping system with faucets to transport both water for the fire pumps and fuel oil for the propulsion system. Mistakes with faucets are easily made in a panic. The more general norm has been added because new ways of mishandling equipment are invented regularly as technology progresses, and designers are admonished to think of these scenario’s before the ship is build.

An important point is that these conflicts or circularities between norms do not depend on any case at hand. The subsumption relationship between two norms does not come into existence in the context of a specific case. The simple fact that the same situation description is permitted by one norm and prohibited by another, does not in any way imply a conflict. Translating norms to preferences, and therefore conflicts to circularities, makes this clear, but the idea that a conflict arises between two norms with respect to a specific situation is widespread.

In [Valente and Breuker, 1995], Valente argued that the conflict should be considered to apply to a “minimal situation description” involving only the shared facts necessary for retaining the applicability of both involved norms. For this purpose he proposed a computation of the “prime implicant” of the conflict. While Valente was right to observe that not any pair of norms of different deontic mode applying to the same situation point to conflict, it is unnecessary to involve an actual situation to be assessed in the computation, as showed in for instance [Winkels et al., 1999]. The situation description we are looking for that gives rise to the conflict is always simply the more specific one.

The previous characterizations are adequate from a deontic logic point of view, but there are arguably other situations which are also conceptualized as a conflict
that are not covered by this characterization. In some cases both mandatory norms can be realized, but only in a way that is not satisfactory with respect to preferences underlying the involved norms. One such scenario, constructed by Elhag et al. (viz. [Elhag et al., 1999]), involves two permissions:

There seem to be other types of conflict as that between the permission for A to live in a certain house and a permission for B to destroy that same house. These conflicts need our attention and have to be embodied in a theory on normative conflicts.

It is trivial to think of similar situations involving two different agents and two mandatory norms. This situation is actually jointly realizable (to the detriment of A), and neither of the agents is confronted by a circularity in making his choice. In addition, both agents are free to act or not to act on the permission. As such, it is not really a conflict between the two involved norms, but everyone would agree that the resulting situation is awkward if both agents act on the permissions.

Another situation of non-deontic conflict arises if the legislator issues a norm that cannot be complied with because it is impossible to realize or its realization does not depend on any agent choice. The legislator cannot order that dice should always land on six eyes, it cannot prohibit volcanoes exploding, it cannot amend the law of gravity, it cannot prohibit fatal accidents, and it cannot order you not to offend anyone.

**Implied Repeal: Choice between Normative Statements**

We all know that beliefs can sometimes be wrong, so intelligent beings need to be able to revise beliefs when they acquire new information that contradicts their old beliefs. Reasoning systems modeling this phenomenon are called belief revision systems.

One common way of determining which beliefs should be surrendered is to use a so-called epistemic entrenchment ordering (viz. [Nebel, 1992]). This ordering expresses the idea that some of our beliefs are more fundamental than others; It is a preference ordering on beliefs.

This preference ordering is distinct from the preference ordering defined by a norm system. The preference ordering on beliefs guides us in surrendering beliefs and adopting new beliefs as our understanding of a situation improves. The preference ordering alluded to by the norm system orders situations and guides us in determining whether the situation is desirable, and how the situation is to be avoided or reached.

Legal reasoning, like most domains of reasoning, involves both types of preference-based inferences. A reasoning system that tries to mimic real world argumentation must combine both types of reasoning. Since both types of reasoning are independent, to the extent that no deontic preference ordering ever follows from an epistemic preference ordering and vice versa, we can limit our attention to each type of reasoning individually.

Makinson and Färdenfors ([Gärdenfors and Makinson, 1988]) postulated that there is a tight connection between belief revision and nonmonotonic logics. Be-
lief revision leads to temporal nonmonotonicity, i.e. the set of beliefs does not grow monotonically with time. Default reasoning leads to logical nonmonotonicity, i.e., the set of consequences does not grow monotonically with the set of premises.

In deontic reasoning we find a parallel process: over time we refine our norm systems. In legal theory we find the principles of *lex posterior derogat legi priori* (Lex Posterior) and *lex Specialis derogat legi generali* (Lex Specialis).

The Lex Posterior principle entails that, in case of irreconcilable conflict, the later provision will take precedence over the earlier provision, and the Lex Specialis principle entails that, in case of irreconcilable conflict, the more specific provision will take precedence over the more general provision. There is also the third principle of *Lex Superior*, or *lex superior derogat legi inferiori*, which states that the higher provision will take precedence over the lower provision. Irreconcilable conflict is understood in this book as the cases of conflict between preference statements introduced in section 2.1.3.

Lex Superior is a fundamentally different kind of notion. The Lex Specialis and Lex Superior principles describe certain phenomena of reasoning in general. They do not sanction the preference of the newer over the older, or the specific over the general, but merely observe that it usually is so.

The Lex Specialis and Lex Posterior principles do not have to be explained in many words to laymen: They will be naturally applied even by children in contexts outside Law. The Lex Specialis principle, as a kind of logical nonmonotonicity, is based on the principle of parsimony in communication. We expect the reader or listener to infer – to some degree – that our more specific statements are exceptions to the more general ones if they appear to be in conflict. The Lex Posterior principle, as a kind of temporal nonmonotonicity, is based on the assumption of improvement: if later and earlier statements from the same source appear to be in contradiction, the reader or listener will generally speaking assume that the later statement reflects a better understanding of the issue by the author than the earlier statement, and that the author of the statement is aware of the fact that he is revising an earlier statement. We take for instance for granted that the latest publication discussing a theory reflects the best understanding of that theory.

As explained in [Boer and Winkels, 2005], the situation is more complicated in Law when reasoning over cases that happened in the past. In normal circumstances we would never consider applying the scientific theories of the 10th century to the reconstruction of an event in the 10th century, but in Law we are often asked to apply the provisions of the past to a case of the past. Logical and temporal nonmonotonicity are explicit in Law, as the legal principles of Lex Specialis and Lex Posterior, because Law can manipulate these principles by for instance instructing the reader to not apply them.

It is exactly because legal reasoning so often seems to deviate from the intuitions embodied by Lex Specialis and Lex Posterior that in the context of Law lay people will start doubting their common sense judgment, and start making mistakes.

Conceptually, application of the principles is easy to understand. Any provision obviously has a history – when did it enter into the norm system and when where what changes made to the text – and propositional content. A Lex Specialis ordering between two provisions can be discovered by comparing the provisions content-wise.
A Lex Posterior ordering between two provisions can be discovered by comparing the history of the provisions.

The Lex Superior principle is also ‘intuitive’, but it is a design principle for complex, layered legal systems. An Act of Parliament for instance takes priority over a Royal Decree, which takes priority over a Municipal Decree, etc. This hierarchy is not something which is discovered by application of the principle to a pair of provisions: it has been designed into the legal system. Provisions overruling the normal activity of Lex Specialis and Lex Posterior usually define a preference ordering on legal sources that can be used for choosing between norms.

The Lex Superior ordering is a consequence of the provisions that manipulate the application of Lex Specialis and Lex Posterior. It does not only apply to categorical distinctions (Act vs. Royal Decree etc.), but also to individual ordering constraints created by certain types of entrenched and self-entrenched clauses, sunset clauses, effective-date clauses etc. Suber described many examples of such clauses in [Suber, 1990].

The logic of the legal system dictates that Lex Superior should take precedence over Lex Specialis and Lex Posterior. The relative priority of the Lex Posterior and Lex Specialis principles among themselves is however not necessarily settled.

The reason for this is that they are implicitly assumed to reinforce each other. Assuming that the legislator refines his expressed preferences over time, is aware of his own acts in the past, and intends it’s new provisions to be compatible with the existing corpus, it is only reasonable to expect that new preference statements refine the existing system. The Lex Posterior ordering and the Lex Specialis ordering are in other words expected to point the same way in most cases, while the Lex Superior ordering is expected to point in the opposite way.

In the case where an older rule is more specific than a newer rule, the older rule seems to amend the newer one – which is absurd – and their are good arguments for precedence of both principles: the legislator could have repealed the older rule but didn’t, and the older rule is not likely to have been intended as an amendment to the first one. To this author the first argument is more convincing, and therefore Lex Posterior should defer to Lex Specialis. Do note that the status of this type of conflict has to be verified for each legal system.

A special case is the conflict between two provisions where a Lex Posterior ordering exists, but a Lex Specialis ordering does not. This is the case in a situation of symmetric disaffirmation described in section 2.1.3. To this situation there is no satisfactory prima facie solution, because the legislator is able to directly repeal the offending older provision and yet failed to do so. This suggests that the legislator did not explicitly intend the ordering that is imposed by Lex Posterior. Still one usually favors the ordering imposed by Lex Posterior if there are no other clues, because the choice between norms has to be made in some way.

Absurd is the case of a symmetric disaffirmation where even the Lex Posterior principle doesn’t offer a solution because the involved provisions where adopted at the same time. This case cannot be solved by interpreting the intent of the legislator, since a rational legislator cannot possibly have intended to contradict himself.

It does happen, and must be resolved. Any criterium will do: For example, one U.S. court held that when two amendments adopted at the same time are irrecon-
cilable, the one receiving the greater number of votes in the Senate takes priority over the other (viz. [Suber, 1990]). The absurdity of this situation becomes clear if one realizes that at least some senators voted to support two amendments with the exact opposite propositional content. This exotic principle has not been accepted in jurisprudence anywhere. In for instance the Netherlands this case is handled (for formal law and decrees) by a provision published in 1990 that proposes decision rules like the sequential order in which provisions were published and the order in which the monarch signed documents.

We prefer to look on this principles of choice as a kind of rule for aggregating (possibly conflicting) normative statements. We therefore do not want things like Lex Specialis logic built into the reasoning system.

Contrary-to-Duty Obligations

The contrary-to-duty (CTD) obligation arises in a sub-ideal situation, brought about by the violation of a primary obligation. This is not allowed in SDL because it is based on a modal distinction between the actual world and the ideal one, and treats normative sentences as constraints on the ideal world. Treatment of obligations as preferences between types of situations orders these worlds.

The relation between the primary obligation and CTD-obligation can be represented according to the following schema:

\[
\begin{array}{c}
\alpha \\
\downarrow \searrow \nearrow \\
\neg \alpha, \beta \\
\downarrow \\
\neg \alpha, \neg \beta
\end{array}
\]

The CTD-obligation in the bottom row only distinguishes cases in which \(\neg \alpha\) is already the case, in both proscribed and prohibited situations.

It is very important to realize that the relation between the primary obligation and the CTD-obligation is not one of compliance or disaffirmation conflict. As long as you don’t get into the sub-ideal situation, the CTD-obligation is silent – neither complied with nor violated. The CTD-obligation also does not imply a permission to enter into the sub-ideal situation.

The Chisholm paradox (viz. [Chisholm, 1963]) is an instructive example of how to use preferences between types of situations to analyze complex sets of logical relations between primary obligations and CTD-obligations. Since we are not building a logic we are obviously not ‘solving’ the paradox, but merely representing the situation that gives rise to it. The Chisholm set consists of the following norms:

1. \(O(\alpha | \top)\)
2. \(O(\beta | \alpha)\)
3. \(O(\neg \beta | \neg \alpha)\)

The paradox arises in standard deontic logic (SDL) when \(\alpha\) is the case. \(\alpha\) reads “a man goes to the assistance of his neighbours”, and \(\beta\) reads “the man tells his neighbours that he will come”. Opinions are divided over whether this paradox
already exists in natural language, or only in the logics that give rise to it. If one reads norms as preference statements, one takes the position that they merely disqualify certain logical representations of the Chisholm situation.

The Chisholm set translates to the following set of preferences:

1. $\alpha \succ \neg \alpha$
2. $(\beta, \alpha) \succ (\neg \beta, \alpha)$
3. $(\neg \beta, \neg \alpha) \succ (\beta, \neg \alpha)$

Since the first preference expresses a preference for both $(\beta, \alpha)$ and $(\neg \beta, \alpha)$, the imposed ordering – from most to least ideal – is the desired one (cf. [van der Torre and Tan, 1998]):

$$\alpha, \beta \succ \alpha, \neg \beta \succ \neg \alpha, \neg \beta \succ \neg \alpha, \beta$$

This ordering is consistent with preference and has no effects on the validity of norms. An obligation is not cancelled by adding a second obligation telling us what to do if the first one is violated. If you tell your neighbours that you will come to their assistance, and don’t do it, then you have violated two obligations. As a general rule, CTD-obligations do not require a choice between the involved norms.

Only because the Chisholm situation involves a third norm ordering the two situations not ordered by the first two, we are able to construct a complete ordering. This is not generally the case with CTD-obligations.

Time adds another interesting twist to the Chisholm situation. Note that the transitions that can be effected by an agent are only those from $(\neg \alpha, \neg \beta)$ to either $(\alpha, \neg \beta)$ or $(\beta, \neg \alpha)$, and from $(\beta, \neg \alpha)$ to $(\alpha, \beta)$. The choice for $\beta$ can only be realized before the choice for $\alpha$.

This makes a temporal reading, where norms only become visible at certain time points and norm 2 only becomes visible because norm 1 was violated, impossible (cf. [van der Torre and Tan, 1998]). This shows that the Chisholm example, although a bit contrived, has been chosen very well by Roderick Chisholm.

Belzer’s Reykjavic set (cf. [Belzer, 1987]) requires a partial ordering of the involved types of situation. The Reykjavic set consists of the following norms:

1. $O(\neg \alpha, \neg \beta|\top)$
2. $O(\beta|\alpha)$
3. $O(\alpha|\beta)$

$\alpha$ reads “tell Reagan the secret”, and $\beta$ reads “tell Gorbachov the secret”. This one is slightly more complicated to translate to preferences. The Reykjavic set translates to the following set of preferences:

1. $(\neg \alpha, \neg \beta) \succ (\neg \beta, \neg \alpha)$
2. $(\beta, \alpha) \succ (\neg \beta, \alpha)$
3. \((\beta, \alpha) \succ (\beta, \neg \alpha)\)

This shows us it is possible that there are more than one CTD-obligations pertaining to one primary obligation. The relation between this primary obligation and the two CTD-obligations can be represented in the following schema:

\[
\begin{align*}
\alpha, \beta & \succ \alpha, \neg \beta \\
\neg \alpha, \neg \beta & \succ \neg (\neg \beta, \neg \alpha) \\
\alpha, \beta & \succ \neg \alpha, \beta
\end{align*}
\]

If CTD-sets get even more complex, it is obviously no longer possible to represent them graphically in this way.

In reality one would prefer to define a concept \((\beta \cap \alpha)\) instead of the cumbersome \(\neg (\neg \beta, \neg \alpha)\), but for the sake clarity I do not rewrite statements. The first preference clearly expresses a preference for \((\neg \beta, \neg \alpha)\) over \((\beta, \alpha)\), \((\neg \beta, \alpha)\), and \((\beta, \neg \alpha)\). The imposed ordering is a partial one:

\[
\begin{align*}
\neg \alpha, \neg \beta & \succ \alpha, \beta \\
\alpha, \beta & \succ \neg \alpha, \beta
\end{align*}
\]

The last two situations are left to personal preference for either Reagan or Gorbachov. Note that the transitions that can be effected by an agent are those from \((\neg \alpha, \neg \beta)\) to either \((\alpha, \neg \beta)\) or \((\beta, \neg \alpha)\), and from those to \((\alpha, \beta)\). There is no ordering constraint on choices between \(\alpha\) and \(\beta\).

One might speculate that the primary obligation and the contrary-to-duty obligation involve a substantially different decision point. The decision to tell or apologize is substantially different from the decision to come or break a promise. This intuition may be dispelled by Forrester’s situation (viz. [Forrester, 1984]), in which one is admonished to kill gently, if one kills. Killing gently is directly subsumed by killing, so it is not possible to kill gently while not killing. \(\alpha\) and \(\beta\) are not at all independently realizable in this case.

This type of situation should not be considered merely theoretical. Real legislation does actually create many situations reminiscent of the Chisholm and Forrester situation. Contrary-to-duty imperatives are quite common, although they are usually far more complex to analyze in legislation. All norms regulating punishment by the legal system, and contractual remedial and reparational obligations and liabilities arising from contract violations follow this general pattern.

There is a very different way of handling (some kinds of) CTD-obligations that should be mentioned. Instead of viewing a CTD-obligation as an obligation arising from the sub-ideal situation in which one norm has already been violated, one could see the CTD-obligation a arising from the violation itself. Consider the following formulations in natural language:

1. One ought not to break a promise.
2. One may break a promise if one apologizes.
3. One ought to break promises and apologize.

4. If one breaks a promise, then one ought to apologize.

5. If item 1 is violated, then one ought to apologize.

Item 2 is clearly a permission that is an exception to item 1. Item 3 is ridiculous, and creates a compliance conflict with item 1. Item 4 is clearly a CTD-obligation to item 1. Item 5 logically expresses the same thing as item 4, but still works somewhat differently in a legal system.

Item 5 is only active as a CTD-obligation if item 1 exists. If the text of item 1 changes, then the preference expressed in item 5 changes as well. Governatori et al. (cf. []) solve this by the introduction of an operator $O(\alpha) \otimes O(\beta)$, which is read as “$O(\beta)$ is the reparation of the violation of $O(\alpha)$”. This makes the obligation $O(\beta)$ inapplicable to any situations in which the opposite of $\alpha$ is the case, but $O(\alpha)$ is nevertheless not violated. The operator is therefore not suitable for replacement of a more general way of dealing with CTD-obligations. Governatori et al. may however have a valid argument that some CTD-obligations do actually require the violation. Arguably this is the case in the following set:

1. One ought not to break a promise.
2. One may break a promise if one pays a 100$ fee to the government.
3. If item 1 is violated, then one ought to apologize.

The question is whether item 1 is still violated if one pays the fee. Does one still need to apologize? We have a problem if that is not the case; We need to distinguish between CTD-obligations that are only applicable if the primary obligation is, and CTD-obligations that are applicable regardless of the primary obligation. This is a case of confluence of a CTD-relation on the logical level, and an intratextual conditional validity relation.

Apparently this is not the case if item 3 would have been formulated as “If the conditions of item 1 are met, then one ought to apologize.”, but I do not dare to guarantee that it will be read this way. These examples are provided as is, without warranty of any kind.

**Requirements from Deontic Logic**

From research in Deontic Logic we take the following requirements:

1. There should be some account of deontic concepts based on choice and subjunctive betterness of the things described by the concept;
2. This account should take into account the meaning of the deontic verbs, and the way these are typically used in law;
3. Contrary-to-duty situations should not lead to logical inconsistency;
4. Normative conflict should not lead to logical inconsistency;
5. It should be possiblt to define rules for resolving normative conflict.
2.2 Legal Expert Systems

It is generally taken for granted in Legal Knowledge Engineering that – given some assumptions on context of use – it is possible to represent normative knowledge in a language with poor expressiveness like production rules. The knowledge representation languages of Ruleburst and Knowledge Tools can both be characterized as production rules, but with different degrees of expressiveness. There are two obvious mappings, each of them represents a different aspect of the full apparent meaning of the normative statement, and only the aspect relevant for solving the problem at hand is usually represented. This discussion is limited to obligations and prohibitions only, and the possibility of conflicts and exceptions is only dealt with cursorily.

2.2.1 Normative Statements

The two mappings for a normative statement $O_i(\alpha \mid \beta)$ (i.e. you ought to $\alpha$ if $\beta$, equivalent to the prohibition $F(\neg\alpha \mid \beta)$) are the following ones:

Violation: $O_i(\alpha \mid \beta) = \neg\alpha \land \beta \rightarrow V_i$, where $V_i$ is the violation of norm $i$ and $O_i$ an obligation following from $i$. Each obligation and prohibition is mapped to a violation, and the violation can itself be condition to another rule.

Compulsion: $O_i(\alpha \mid \beta) = \beta \rightarrow_i \alpha$, where $\beta \rightarrow_i \alpha$ is taken as a computer-assisted uncontrollable urge to take the action required by norm $i$. Implicitly it also means that in this conception $\alpha$ is an action and $\beta$ a situation description – but $\beta$ may just as well be your own or someone else’s preceding action or the violation of another norm – and the performance of $\alpha$ is seen as unproblematic in the sense that performance of $\alpha$ is dependent only on the will to perform $\alpha$ and there are no (failed) attempts to $\alpha$ and no relevant external conditions to be verified before $\alpha$ can be performed. If there is such a condition $\gamma$ one adds it to the rule, resulting in $\beta \land \gamma \rightarrow_i \alpha$. This rule mixes a classical action schema – defining the preconditions and consequences of the action $\alpha$ itself – with the compulsion to perform $\alpha$.

The choice between these two representations is motivated by whom is addressed by the norm. In the simplest terms this is either you or me. If it is you, I will tend to use the violation format for the norm. If it is me, I will tend to use the compulsion format. In an administrative decision support system $J$ will generally speaking take the civil servant role, and you take the client role. It is perfectly possible to build systems that work the other way around: check whether the public body you are interacting with violates any rules.

A number of problems immediately becomes apparent when we consider the possibility of reusing these production rules in other contexts. Firstly, it is not possible to infer the violation rules from the compulsion rules if the compulsion rule cannot be separated from other production rules with the same structure. This is why the $\rightarrow_i$ is used. The most obvious solution is to identify and classify the rules themselves ($i : \beta \rightarrow \alpha$), but this is not an elegant solution from a semantic point
of view. If one needs to add conditions $\gamma$ constraining the possibility of the action $(\beta \land \gamma \rightarrow \iota \alpha)$ then separating the normative knowledge from background knowledge becomes even harder.

Another problem is that if $\beta \rightarrow \iota \alpha$ and $\gamma$ is a precondition of $\alpha$ one may derive a compulsion to to $\gamma$, or $\beta \rightarrow \iota \gamma$, for pragmatic and context-dependent reasons.

A special case of the compulsive interpretation of norms is the following:

**Compulsion to Sanction:** $O_i(\alpha \mid V_j) = V_j \rightarrow \iota \alpha$, where $V_j$ is a violation. Note that in reality there is also often the possibility that there is only a permission to sanction or search remedy or reparation for a certain class of agents. The sanction is in any case never *implied* or *caused* by the violation.

Consider the example of theft. Theft $\tau$ is defined in legislation, and the definition can be captured itself in the form of a rule. Theft is, on the other hand, at the same time a violation proposition. This relation between action description $\alpha$, classification as theft $\tau$, qualification as a violation $V_i$, and obligation $j$ to sanction $\beta$ may be encoded as $\{\alpha \rightarrow \beta\}$, or $\{\alpha \rightarrow \tau, \tau \rightarrow \beta\}$, or $\{\alpha \rightarrow \tau, \tau \rightarrow V_i, V_i \rightarrow j \beta\}$. Only in the most explicit representation a mapping to norms can be made. The relation between theft and a sanction is obvious, but the possibility to search remedy, reparation, or administrative sanction is very often not considered to be a sanction following from a violation. Modelers should keep in mind that wherever someone is sanctioned, someone else is empowered to sanction.

In the previous note on LKIF by the Leibniz Center we included a detailed description of normative conflict and normative conflict resolution. It is generally accepted that *most* permissions function as *exception* to a more general obligation. This means that a permission to $\alpha \land \beta$ and obligation to $\alpha$:

1. are in normative conflict because $\alpha \land \beta \rightarrow \alpha$; and
2. the normative conflict is resolved in favour of the permission because the applicable conflict resolution rule is Lex Specialis or Lex Posterior$^4$.

In a production rule system this is usually handled by the following representation pattern:

**Exception** $O_i(\alpha \mid \top) \land P_j(\alpha \land \beta \mid \top) = \alpha \land \neg \beta \rightarrow V_{ij}$, where $V_{ij}$ is the violation of $i$, taking into account $j$, and $\top$ can be anything.

While this compound rule can work out correctly in a specific context, it is incorrect in assuming that 1) the exception only applies to nearby legal provision $i$, and 2) conflicts between both are always resolved by the default application of Lex Specialis$^5$. Since the rule represents an intersection of the meaning of two participating normative statements, it cannot be taken as a general representation of either norm.

---

$^4$I.e. the permission is generally speaking more specific or was enacted after the obligation to which it is an exception.

$^5$The only conflict resolution principle that can be applied by only looking at the logical content of the normative statement.
2.2.2 Other uses of Production Rules

Since the production rules with a normative character are often mixed with production rules expressing background knowledge, this is also the place to make a superficial classification of other purposes of production rules:

**Subsumption:** $\alpha \rightarrow \beta$ expresses that $\beta$ subsumes $\alpha$: $\beta$ and $\alpha$ are predicated of the same thing. A car is for instance a vehicle. If necessary and sufficient conditions of a concept are known (e.g. $\alpha \leftrightarrow \beta$), it is also possible to make a rule in both directions and use the one which is most likely to apply to your problem context.

**Heuristic:** A heuristic rule $\alpha \rightarrow \beta$ in the sense of Clancey's Heuristic Classification (cf. [Clancey, 1985]) relates an abstract problem category $\alpha$ to an abstract solution category $\beta$. Heuristic classification consists of a phase of abstraction, where subsumption rules generalize concepts to more abstract problem concepts higher in the taxonomy, a mapping from problem to solution, and the refinement downward in the solution taxonomy by asking for more differentiating features. Concept definitions translate to other subsumption production rules (upward, downward) depending on whether they are most likely to be applied in the phase before or after the heuristic match. Heuristic rules are usually not reusable outside of the specific context for which they were written.

**Causality:** Similar to subsumption rules, except that it required that the things on both sides of the $\rightarrow$ are causally connected in some way, and that the cause exists before the effect.

**Action-schema:** The action schema defines the preconditions and consequences of some action $\alpha$. But the action itself obviously does not happen automatically, unless the designers of the system decide that the action is in the interest of the user (see compulsion). A simple solution is to use some kind of intention proposition: if $\alpha$ is the precondition for doing $\beta$, and $I_\beta$ is the intention to do $\beta$ as soon as it is possible, then $\alpha \land I_\beta \rightarrow \beta$.

The action-schema is the most insidious type of rule, since it can be used to impose an implicit control structure on the system as a whole. It is often used to make the system reason in a goal-directed way.

2.2.3 Teleological Statements

In most administrative systems we find two involved parties – usually a civil servant and a citizen in some role (home owner, unemployed person, disabled person, etc) – and a purpose of the system: issuing permits or granting a benefit. It is implicitly assumed that everyone involved with the system in the capacity of citizen wants the permit or benefit. This is obviously a plausible assumption in most cases, but not always\(^6\).

\(^6\)Someone who was fired unjustly and goes to the Legal Services Counter described in [van Engers et al., 2004] might want to file an official complaint against being fired without checking for eligibility for unemployment benefits.
This bias towards the goal of the system often leads to a mixup of teleological reasoning (the intention $I_{beta}$ to achieve $\beta$) with the compulsion pattern ($\top \rightarrow i \beta$). Take the example of filing a complaint against a decision. It is good advice to say: you should file your complaint within six weeks after the decision ($if$ you want to complain). The corresponding normative statement means something completely different: the civil servant is obligated to accept the complaint if it is filed within six weeks of the decision. Very often normative statements are applied to the wrong agent! The complaining citizen is very often not obliged to complain within six weeks, not obliged to complain at all, and not obliged to finish an administrative procedure he started. The citizen can change his mind at any point, and retains his freedom to act all the time.

Let’s identify this pattern as follows:

**Compulsion-to-achieve-benefit:** $I_{\alpha} \rightarrow i \beta$, or according to our misinterpretation of $i$ you ought do $\beta$ because $I_{\alpha}$.

This happens because much of the things legislator’s are involved with are *transactions* between two parties $p_1$ and $p_2$. The legislator imposes obligations on $p_1$ because the legislator believes the interest of $p_2$ are served by doing so. In application contexts the norms which address $p_1$ are interpreted in the context of the perceived interests of $p_2$. The note written by Giovanni Sartor on Fundamental Legal Concepts ([Sartor, 2006]) describes in a clear way the so-called Hohfildian concepts of $p_2$’s right (correlative to $p_1$’s duty), $p_2$’s power (correlative to $p_1$’s liability), etc.

The norms are usually written in terms of generic agents playing certain roles (home owner, driver, citizen, civil servant, etc.). The norm can explicitly mention the addressee of the norm, the intended beneficiary, both, or neither of them. If a party is not explicitly mentioned, this can mean it is obvious from context or that it applies to anyone within the reach of the jurisdiction.

### 2.2.4 Declarations

Last but not least, there is also the declaration of the legislator. This is not a norm, in the sense that it cannot be violated, but it does have a certain normative aspect as an act of communication. The definition in legislation *must* be adopted.

**Declaration:** $\beta \rightarrow i \alpha$, where $\beta \rightarrow i \alpha$ – here meaning $\beta$ is deemed to be $\alpha$ for the purposes of $i$ – is taken as a compulsion to make the inference required by norm $i$. This format is very similar to the compulsion interpretation described earlier, and behaves in all other respects like subsumption. The performance of $\alpha$ is again seen as unproblematic, and in this case it really cannot be violated.

Note that declarations by the legislator can be both on the concept and on the instance level. The legislator for instance defines public body in an intensional sense, but can also declare that agency $X$ is or is not a public body. Because the legislator

---

7The General Administrative Act may say in its first chapter: “This act applies to public bodies. For the purposes of this act, public body is defined as [...] and excludes [...]”

8Criminal Law is a typical example of the universal addressee.
can redefine existing concepts it is useful to distinguish the defined concept from the natural one\(^9\).

The declaration itself is not complicated to deal with, but the use of declarations does make it much harder to interpret the law. For one, it is not always obvious whether some statement should be read as declarative or imperative. Secondly, any complication arising from the mixup of norms and other knowledge can also happen with declarations. Thirdly, the declaration often function as conceptual glue to link for instance norm, violation, and sanction (as in theft), or declares *positional* concepts (sale, tort, marriage) used by the counterparty/beneficiary to infer that he has a certain right or power vis à vis some other agent.

### 2.2.5 Requirements from Administrative Expert Systems

The representation of norms as production rules strongly depends on 1) whose perspective you take in a transaction, 2) the assumed goal of the parties using the system, and 3) who is expected (or in some sense *allowed*) by the system to occasionly violate the norms, and who is expected to comply with them. A *neutral* representation removes this bias from the interpretation, and LKIF should be based on a neutral perspective. At the same time this perspective itself obviously contains important and useful knowledge often forgotten in deontic logics, in particular with respect to the beneficiary of the norms as described by Giovanni Sartor in [Sartor, 2006]. This view of the norm as regulating a transaction between two kind of parties – addressee and beneficiary – is central to capturing both perspectives, and should be a focus for the development of LKIF.

It is not to be expected that an LKIF processor will be able to discover semantic distinctions between production rules that are not explicit in some way. This means that information must be added to point out which parties are involved, which rules are normative in character, what role the involved parties play, etc. Because expert systems are usually highly focused and involved few parties interacting with eachother, this should not be a big problem.

Since there are multiple ways to generate a rule from a norm, generating an expert system from LKIF will also require choices. If we conceptualize the norm as a structure relating knowledge items that play certain roles, translating from rules to LKIF consists of filling in the slots we can infer from the rule, and translating from LKIF to rules consists of checking which slots are filled and which type of interpretation can therefore be generated.

### 2.3 Vendor Requirements

One goal of ESTRELLA is to develop translators between the knowledge representation formats used by the vendors in the project (KnowledgeTools, RuleBurst and

---

\(^9\)Simple example: X offers to sell his house to Y if Y accepts before next tuesday. Y accepts, but X retracts the offer. According to legislator L X has now sold the house to Y, according to X he hasn’t, and according to the uninformed opinion of Y X has violated the law by not selling him the house.
RuleWise) and LKIF, to enable the development of legal knowledge systems using their products without the risk of “vendor lock-in”.

Towards this end, as a necessary first step, the logics underlying the systems used by the vendors have been reconstructed, in task T1.2, and reported in deliverable D1.2 [van den Berg et al., 2006]. This work has made it clear that there are significant differences between the logics used by the vendors which make it impossible to develop an interchange format which would enable translation between these systems without significant losses of information.

This risk was anticipated in the Technical Annex of Estrella [ESTRELLA, 2006, p. 51]:

The formalisms used for representing legal knowledge by the industrial partners in the project is quite diverse. It may be difficult or indeed impossible to develop an Legal Knowledge Interchange Format (LKIF) which allows knowledge represented in these diverse formalisms to be interchanged without loss of information. . . .

ESTRELLA will deal with these technological risks in a pragmatic way. LKIF, if necessary, will consist of a number of sublanguages, possibly layered, sufficient to cover a range of approaches to representing legal knowledge. Even though it may not be possible to exchange knowledge between knowledge bases encoded in these sublanguages without loss of information, it will be possible to exchange knowledge bases between the proprietary formats of the main vendors of legal knowledge systems and the open LKIF format, using one or more of these sublanguages. This will provide vendor neutral formats for exchanging legal knowledge, for each approach to representing legal knowledge, meeting the primary goal of the project.

Thus the fallback plan of the technical annex is to export information from the formats of the vendors to LKIF, with little loss, but import only as much information from LKIF to the format used by each vendor as is feasible, considering the limitations of each format.

However, LKIF would not be of much practical use as an interchange format for legal knowledge if it were not possible to transfer at least some legal knowledge represented using one vendor’s format to the format used by some other vendor. It is not clear at this point just how much knowledge must be transferable in order for LKIF to be considered useful and successful for the purpose of knowledge interchange among the current systems used by the participating vendors.

However, whether or not LKIF will be successful as a knowledge interchange format among the formats currently used by the vendors, the ESTRELLA project will strive to make LKIF serve as a target and model towards which the vendors will want to migrate their products and services, so as to enable future versions of their products, and the products of other vendors in the legal knowledge systems market, to exhibit a high degree of interoperability.

For the details of the knowledge representations formats currently used by the vendors, so as to determine which hurdles must be overcome to meet the requirements articulated here, see deliverable D1.2.
The logical reconstruction of the vendor formats in D1.2 has shown that, for the most part, the semantics of these formats are various subsets of first-order logic. Since we know from legal theory and AI and Law that legal reasoning and argumentation cannot generally be understood as deduction, this raises the question about how nonetheless demonstrably useful applications for public administration have been able to be developed and deployed using these formats. The answer is that applications of legal knowledge systems for public administration have a greatly reduced scope. The legal knowledge system represents the public agency’s theory of the legal domain. Public clerk’s apply this theory to help analyze a legal situation. It is clear that this theory may be insufficient for analyzing some cases. However, rather than designing the legal knowledge system to support other, nondeductive forms of legal argumentation, the expectation is that these problems will be resolved outside of the system. For example, a citizen who feels that a wrong determination of his or her rights to social benefits has been made, can initiate some other legal proceeding to challenge this decision, perhaps by filing a court complaint. If the citizen is able to convince the public agency, or the court, that an error has been made, the agency will then take steps to revise its theory of the legal domain, and modify the legal knowledge system accordingly.

LKIF should support this workflow. Indeed, the first priority should be to assure that LKIF is sufficient as a knowledge interchange format for this kind of application scenario in public administration, since this is currently the market in which the vendors do bulk of their business. But this goal is not in conflict with the more ambitious goal of supporting other forms of legal reasoning and argumentation, as these have been studied and modeled in the field of Artificial Intelligence and Law, since deduction is not superseded by these other forms, but rather complemented by them.
Chapter 3

Description Logic and Rules

In this chapter, we present a way to integrate OWL DL with rules. The plan is as follows: Preliminarily, we will address the issue of why the integration is necessary within our research project (3.1). Then, we will spell out the semantics of OWL DL on the background of the semantics for RDF/RDFS (3.2 and 3.3). Finally, in 3.4 and 3.5 we will present and discuss two proposals for integrating OWL and rules: DLP (Description Logic Program) and SWRL (Semantic Web Rule Language).

3.1 Using a Combination of OWL DL and Rules

Why OWL DL, and why rules? There are several reasons to adopt OWL DL as a language formalism to model (legal) knowledge. Among others, interchangeability and decidability play a key role. OWL has become the standard language in which an increasing amount of sharable knowledge is being formalized; investing time in designing a new representational language will bring the further problem of translating the existing formalized knowledge into our new language, whence the choice to adhere to the existing OWL standard. Secondly, among the OWL family of languages, OWL DL is the most expressive dialect that remains decidable.

There is one drawback to OWL DL, however—its limited expressiveness. Hence, it needs to be extended. One of the proposal in the KR community is to extend it with rules. A rule is a formula of the form:

\[ \varphi_1 \land \cdots \land \varphi_k \rightarrow \psi, \]

where \( \varphi_1 \land \cdots \land \varphi_k \) is the antecedent (body) and \( \psi \) the consequent (head) of the rule. The meaning of a rule is typically ‘whenever the antecedent is true, the consequent is true’.

The choice of rules as extension for OWL is not accidental and it is relevant to us. The general motivation is that integrating OWL (DL) with rules yields a language that is as expressive as FOL, which is a very expressive language, if not the most expressive language we may aim at, within a formal standpoint. Transcending the limit of FOL is not advisable, since it usually leads to incompleteness (which in turn means that we do not have a logic anymore). Yet a language almost as expressive as FOL is desirable.

A more precise motivation for the choice of rules is that OWL DL is well suited to express taxonomical, terminological or generic knowledge, whereas rules may express...
configurations of concepts and properties\textsuperscript{1} that cannot be reduced to taxonomical classification—and we need to express these configurations that OWL DL cannot express.

An example can make this apparent. Consider the (informally expressed) rule:

*If a judge condemns an innocent person, then he is unjust toward that person,*

whose translation in FOL looks like:

$$\forall x, y : \text{Judge}(x) \wedge \text{condemn}(x, y) \wedge \text{Innocent}(y) \rightarrow \text{unjust}(x, y).$$

The example contains a relation between concepts and properties that we do want to be able to express in our representational language (it should be fairly obvious why), but that OWL DL is not capable to express. The intuitive reason is relatively simple: Ascertaining that something is a judge, an innocent, etc. (the atoms in the antecedent of the rule) is a matter of taxonomical knowledge and reasoning. However, attributing a property to individuals since a configuration of properties happens to be true of them, is not a matter of taxonomical reasoning.

From a formal point of view, the deficiency is due to OWL inability to handle variables. For in the rule above the variables \(x, y\) are passed on from the antecedent to the consequent, and OWL DL is not able to handle such a passing on. By contrast, a rule such as

*If someone performed an action that is not allowed, then he has committed a violation,*

which is translatable in FOL as

$$\forall x, y : \text{Person}(x) \wedge \text{Action}(y) \wedge \text{perform}(x, y) \wedge \text{Disallowed}(y) \rightarrow \text{commit}(x, \text{violation}),$$

can be expressed in OWL, namely as:

$$\text{Person} \sqcap \exists \text{perform. Action} \sqcap \text{Disallowed} \sqsubseteq \text{Person} \sqcap \exists \text{commit} \sqcap \text{Violation}.$$  

Notice that in the last example only one variable is shared between antecedent and consequent of the rule. So these two tentative remarks can be made:

- if at least one variable is shared between antecedent and consequent of a rule, then the rule is OWL DL expressible;
- if more than one variable is shared, then the rule may not be OWL DL expressible.\textsuperscript{2}

However, to fully understand why this is so, we need much more formal machinery—i.e., the OWL DL semantics but also a formal characterization of the OWL DL language as (a superset of) the bi-simulation invariant fragment of FOL (see 3.3).

\textsuperscript{1}In accordance to the terminological conventions adopted in KR, concepts and properties are denoted by one-place and 2-place relations, respectively.

\textsuperscript{2}This crucial observation is taken from [Parsia et al., 2004], which uses it in another but similar context.
Our preliminary discussion should stop here. It suffices to have shown that there are rules that cannot be expressed in OWL DL. What we need is an investigation concerning how to integrate OWL and rules. In section 3.5, we provide an overview of the problem, by looking at the already ample literature on the topic (see, among others, [Donini et al., 1998], [Horrocks et al., 2004a], [Parsia et al., 2004], [Motik et al., 2005], [Rosati, 2006]).

Before getting started, one may legitimately be puzzled by the question: If OWL plus rules gives FOL, why don’t we just stick to the old and well-known FOL? Why did we need to select a fragment of FOL (namely OWL) and then extend it to get FOL back? It seems we are moving around in a circle, but fortunately we are not.

There are at least three good reasons why not: First, FOL is not decidable, and thus we need to single out a fragment of FOL that is almost as expressive as FOL but still decidable (and this is not an easy task). Second, it is often the case that new logics are invented, and that after all they boil down to be (equivalent to) subsets of FOL. Modal logic itself—which might be seen as a revolution in the way of doing logic—can be translated into a language that is a subset of FOL (see [Blackburn et al., 2001], Chapter 2.4.). Nonetheless, modal logic allowed logicians to model and formalize many aspects they couldn’t think of until they were limited to FOL. Third (and most important point), FOL and knowledge representation languages, such as OWL, have significantly different syntaxes: the former has a human readable syntax (which is formal only in a weak sense), and the latter has a machine readable syntax (which makes the language fully formal). Moreover, OWL is downward compatible with other standard representational languages, such as RDF (see [Antoniou and van Harmelen, 2004]).

In sum, we are concerned with expressiveness (and FOL is expressive enough) but also decidability (and FOL is not decidable). Second, shifting between (equivalent) formalisms may help us look at the (same) problems from different perspectives and find better solutions. Third, our concern is not only theoretical—we are not simply choosing a (decidable) logic, but a computer-usable language to interchange knowledge.

### 3.2 RDF/RDFS Semantics

Let us begin with the RDF vocabulary and then move to the RDF semantics.\(^3\)

The **RDF vocabulary** \( V := V_{URI} \cup V_L \) is a set of RDF names. A **RDF name** is a URI reference in \( V_{URI} \) or a literal in \( V_L \). A URI can be a URL or URN, which are both registered strings of symbols that uniquely identify an entity. Names can occur as subject, predicate, or object of an a **RDF triple**, which has the shape \(<subject, predicate, object>\). A set of RDF triples is a **RDF graph** \( \mathcal{G} \).

The term graph is used since any RDF triple can be visualized as a simple directed graph where subject and object label the nodes, and the predicate labels the edge connecting them.

Notice that not every node of a graph \( \mathcal{G} \) needs to be labeled by a name, as some nodes may be blank. Call \( B_\mathcal{G} \) or simply \( B \) the set of blank nodes of \( \mathcal{G} \). If \( B_\mathcal{G} = \emptyset \),

\(^3\)We will follow [Heyes, 2004].
then $G$ is a *RDF ground graph*, and if a triple has no blank nodes, then it is a *RDF ground triple*. In terms of FOL, triples are like atomic formulas; graphs like RDF sets of atomic formulas; and blank nodes are similar to variables.

**Definition 1.** A *RDF model*—denoted by $\mathcal{M}_{\text{RDF}}$—is a tuple $\langle R, P, I_L, I_{\text{URI}}, I_P, g \rangle$, where $R$ is a domain of resources,$^4$ $P$ is the set of properties, and the different $I_i$’s are interpretation functions defined as:

- $I_L : L \mapsto \rightarrow R$, where it is clear that $L \in V$, but also $L \in R$.
- $I_{\text{URI}} : \text{URI} \mapsto \rightarrow R \cup P$;
- $I_P : P \mapsto \rightarrow R \times R$;
- $g : B \mapsto \rightarrow R$.

In other words: Any literal is mapped to itself, since literals are by themselves resources in $R$. Secondly, URIs are mapped to the domain of resources $R$ and properties $P$. This *intensional* level of interpretation given by $I_L \cup I_{\text{URI}}$ is integrated with an *extensional* level of interpretation, which is taken care by the interpretation $I_P$ that gives the extension of a property in terms of pairs of resources. Finally the function $g$ takes care of blank nodes by assigning to them elements in the domain $R$ (think of $g$ as the assignment function$^5$ for variables in FOL).

Notice that, since a property $p$ in $P$ may also be a resource in $R$, its extension $I_P(p) \in \wp(R \times R)$ may contain the property $p$ as element of its pairs $\langle r, r' \rangle \in R \times R$. This especially is the case if a name $n$ occurs as subject or object in a triple and as predicate in another triple, so that $I_{\text{URI}}(n) \in R \cap P$, where $R \cap P$ must be non-empty. At first, this seems a violation of well-foundedness, in the sense that the (interpretation of a) property $p$ may happen to be an element of itself, i.e., $p \in p$, thus giving rise to a loop or infinite descending chain with respect to the $\in$ relation.

Fortunately, this is not the case, since RDF—as pointed out before—distinguishes between intensional and extensional interpretation of properties. The intension of a property may be contained in its own extension, while rising no problem with respect to well-foundeness. Nonetheless, the duality intension/extension has pros and cons, but however interesting we cannot discuss them here.

We are now ready to give definitions for the truth relation and the entailment relation between graphs. They both will be denoted by $\models$. We will denote a generic triple by $\langle s, p, o \rangle$.

**Definition 2.** The *RDF truth* relation is defined as follows:

$$\mathcal{M}_{\text{RDF}} \models \langle s, p, o \rangle \quad \text{iff} \quad I_{\text{URI}}(p) \in P \text{ and } \langle I_{\text{URI}} \cup g(s), I_{\text{URI}} \cup g(o) \rangle \in I_P(I_{\text{URI}}(p)) \quad \text{for some } g$$

$$\mathcal{M}_{\text{RDF}} \models G \quad \text{iff} \quad \mathcal{M}_{\text{RDF}} \models \langle s, p, o \rangle, \text{ for all } \langle s, p, o \rangle \in G \text{ and for some } g$$

$$G \models G' \quad \text{iff} \quad \mathcal{M}_{\text{RDF}} \models G \Rightarrow \mathcal{M}_{\text{RDF}} \models G', \text{ for all } \mathcal{M}_{\text{RDF}}$$

$^4$Resource can be taken to be synonymous of entity or individual in a logical domain.

$^5$Assignment functions are generally used by formal semanticists. See [Gamut, 1991], vol. I.
The first clause describes when a triple is true in a RDF model under a certain assignemnt $g$. Note that if the predicate $p$ is denotation-less, the triple is false (not senseless, as one may expect). Subject $p$ and object $o$ are interpreted by the function $I_{URI} \cup g$, as $p, o$ might stand for URIs but also blank nodes. The second clause describes when a graph (set of triples) is satisfied in a model under a certain $g$. The third clause describes the semantic relation of logical entailment between graphs, for an arbitrary $g$.

Beyond the RDF simple or generic vocabulary, we have the RDF specific vocabulary, which consists of URI references beginning with rdf. Elements of this vocabulary are, e.g., rdf:type, rdf:property, rdf:Statement, rdf:subject, rdf:Predicate, rdf:object. Since these URIs are not just any name, their possible interpretations must be constrained. There are two ways to do so: one is by stating semantic restrictions on the interpretation functions $I$; the other is by using axiomatic triples. Let us look at two examples. The semantic constraint

$$x \in P \text{ iff } \langle x, I_{URI}(\text{rdf:property}) \rangle \in I_P(I_{URI}(\text{rdf:type}))$$

requires any element of $P$ to be a (type of) property. The following axiomatic triple:

$$(\text{rdf:type}, \text{rdf:type}, \text{rdf:property})$$

requires types to be a (type of) property. Indeed, an axiomatic triple can receive a straightforward equivalent semantic formulation. For example, the above axiomatic triple could have been phrased as:

$$\langle I_{URI}(\text{rdf:type}), I_{URI}(\text{ref:property}) \rangle \in I_P(I_{URI}(\text{rdf:type})) \text{ for all } I_P, I_{URI}.$$

In RDF semantics we find many other axiomatic triples that we will not list here for lack of space (see [Heyes, 2004] for a complete exposition).

Another interesting feature of the RDF language is reification of triples. Reification works as follows: First, a triple $\langle s, p, o \rangle$ is associated to its token, call it $t$, whose interpretation $I(t)$ is a member of the set of resources $R$. Second, four triples that contain $t$ as subject (and, thus, that are indirectly—via $t$—about the triple $\langle s, p, o \rangle$) are added:

$$(t, \text{rdf:type}, \text{rdf:statement})$$

$$(t, \text{rdf:subject}, s)$$

$$(t, \text{rdf:predicate}, p)$$

$$(t, \text{rdf:object}, o)$$

The four triples above ensure that the resource $t$ is a statement (first triple), and that the subject, predicate and object of $t$ are $s$, $p$, and $o$, respectively (second, third and fourth triples). As a result, the triple $\langle s, o, p \rangle$ and its reification $t$ will have the same subject, predicate and object.

We now move to the RDFS language and its semantics. We have already seen that RDF comprises a specific vocabulary whose names have a fixed or constrained meaning. The language RDF Schema (or simply RDFS) enlarges the RDF specific vocabulary with names beginning with rdfs. Some examples are rdfs:domain,
rdfs:range, rdfs:Resource, rdfs:Class, rdfs:subClassOf. In order to spell out what these URIs actually mean, we need to define the RDFS semantics.

The RDFS semantics is an extension of the RDF semantics (simple and with RDF specific vocabulary). The RDF model is augmented with the interpretation function \( I_C : \mathbb{C} \rightarrow \wp(\mathbb{C}) \), where \( \mathbb{C} \) is the set of all classes (one-place relations), in the same way as \( \mathbb{P} \) is the set of all properties (two-place relations). The function \( I_C \) defines the extension of a class \( c \in \mathbb{C} \), in the same way as \( I_P \) defines the extension of a property \( p \in \mathbb{P} \). The function \( I_{URI} \) should be redefined as \( I_{URI} : URI \rightarrow \wp \cup \mathbb{R} \cup \mathbb{C} \). Once again, we have a duality between intensional and extensional point of view, which are taken care by \( I_{URI} \) and \( I_C \), respectively.

Let us now see how the meanings of some RDFS-specific URIs mentioned above are spelled out:

1. \( x \in I_C(y) \iff \langle x, y \rangle \in I_P(I_{URI}(\text{rdf:type})) \)
2. \( \mathbb{C} = I_C(I_{URI}(\text{rdfs:Class})) \)
3. \( \mathbb{R} = I_C(I_{URI}(\text{rdfs:Resource})) \)
4. \( \langle x, y \rangle \in I_P(I_{URI}(\text{rdfs:domain})) \land \langle x', y' \rangle \in I_P(x) \Rightarrow x' \in I_C(y) \), for any \( x, x', y, y' \).
5. \( \langle x, y \rangle \in I_P(I_{URI}(\text{rdfs:range})) \land \langle x', y' \rangle \in I_P(x) \Rightarrow y' \in I_C(y) \), for any \( x, x', y, y' \).
6. \( x \in \mathbb{C} \Rightarrow \langle x, I_P(\text{rdfs:Resource}) \rangle \in I_P(I_{URI}(\text{rdfs:subClassOf})) \)
7. \( \langle x, y \rangle \in I_P(I_{URI}(\text{rdfs:subClassOf})) \Rightarrow x, y \in \mathbb{C} \land I_C(x) \subseteq I_C(y) \)

A few comments are in order. Condition 1 secures that \( x \) is in the extension of a concept \( y \) if and only if \( x \) is a (type of) \( y \). The second condition connects the intension and the extension of \textit{rdfs:Class}—respectively, \( I_{URI}(\text{rdfs:Class}) \) and \( I_C(I_{URI}(\text{rdfs:Class})) \)—by equating \( \mathbb{C} \) to the extension of the intension of \textit{rdfs:Class}. The third condition defines \( \mathbb{R} \) along the same lines. Condition 4 and 5 take care of the meaning of \textit{rdfs:domain} and \textit{rdfs:range}, by stating that if \( y \) is the domain (range) of \( x \)—which is thus composed of 2-place tuples of resources—then every resource that is the first (second) argument of any tuple in \( x \) must be in the extension of \( y \). Finally, the last two conditions determine the meaning of \textit{subClass}, by stating that if something is a class, then it is a subclass of the class of resources; and that if the intensions of two URIs are in the subclass relation, then their respective extensions follow a relation of set-theoretical inclusion.

The above conditions are not exhaustive as to fully describe the RDFS semantics (for this, see [Heyes, 2004]). Here we only aimed to give a short overview of how the relatively complex expressiveness of the RDF(S) language can be characterized via a formal semantics. Our exposition was only meant to provide the adequate background to introduce the OWL DL semantics.
3.3 OWL DL: Syntax and Semantics

OWL DL\(^6\) (hereafter, simply OWL) is in general more expressive than RDF(S), but two features of RDF(S) are lacking: The duality between extension and intension, and the reification mechanism that renders RDF(S) able to express a form of second-order predication. This is a consequence of the fact that, while OWL DL has an equivalent logic, RDF(S) does not (for reification and second-order predication are missing in FOL, the logic par excellence).

The *OWL Vocabulary* \(V := V_L \cup V_{URI}\), where (similarly to the RDF vocabulary) \(V_L\) is a set of literals and \(V_{URI}\) a set of URI references. The set \(V_{URI}\) is composed as follows:

- \(V_I\), set of individual names, e.g. *Peter*
- \(V_O\), set of ontology names, generally consisting of URL addresses indicating where ontologies are stored;
- \(V_{IC}\), set of (individual-)class names, including e.g. *owl:Thing, owl:Nothing*;
- \(V_{DC}\), set of datatype-class names, including e.g. *rdfs:Literal, xsd:gDay, xsd:integer*;
- \(V_{IP}\), set of (individual-)property names, e.g. *has_father*;
- \(V_{DP}\), set of datatype-property names, e.g. *height_in_meters*;
- \(V_{AP}\), set of annotation property names, including e.g. *owl:label, owl:seeAlso*;
- \(V_{OP}\), set of ontology property names, including e.g. *owl:import*.

The mnemonics here are: for class use \(C\); for property \(P\); for individual object \(I\); for annotation \(A\); for ontology \(O\). The reader familiar with FOL may find helpful to see that the set \(V_I\) is a set of individual constants; \(V_{IC}\) and \(V_{DC}\) are sets of one-place predicates (classes); \(V_{DP}, V_{IP}, V_{AP}\) are sets of two-place predicates (properties). The only complication of the OWL DL vocabulary is that one- and two-place predicates are subdivided depending on whether they refer to objects/individuals, datatypes, annotations, or ontologies.

Additionally, the *OWL logical vocabulary* consists of symbols for classes construction, i.e. \(\sqcup, \sqcap, \neg, \forall, \exists, \le\), and also symbols for formula construction, i.e. \(\sqsubseteq\) and \(\equiv\). Let us now turn to the *OWL syntax*, i.e., how to construct terms (atomic classes or complex classes), and formulas (OWL axioms and facts).

**Definition 3 (OWL-Class).** The set of *OWL classes* is recursively defined by the following rule:

\[
\text{OWL-Class} := C | \top | \bot | \neg C | C_1 \sqcap C_2 | C_1 \sqcup C_2 | \forall P.C | \exists P.C | \forall T.D | \exists T.D | \le n^{P^{int}} | \le nT | OneOf(i_1, \ldots, i_k) | OneOf(l_1, \ldots, l_k)
\]

\(^6\)For thorough treatment of OWL DL semantics that we are going to present, see [Patel-Schneider and Horrocks, 2004].
where $C$ stands for a class (atomic or complex); $P \in V_{IP}$ an atomic property, $D \in V_{DC}$ a data type class; and $T \in V_{DP}$ a data type property. The symbols $\top$ and $\bot$ are short for \texttt{owl:Thing} and \texttt{owl:Nothing}. $\textit{P}^\text{int}$ means that the property $P$ is not a transitive property or is not a sub-property of a transitive property.\footnote{This is one of the expressive limitations of OWL DL.} The predicate $\textit{OneOf}$ is a concept constructor that gives lists of individual names $i$ or literals $l$.

Paraphrasing the above definition, an OWL class can be the universal class, the empty class, but it can also be obtained via alike set-theoretical operations of complementation, union and intersection performed upon perviously defined classes; existential, universal and cardinal value restrictions can also be used in class construction. It should be already clear that while class construction may involve datatype-classes or properties, we will always have an individual-class as a result of the construction. In other terms, datatype-classes cannot be complex. Why this is so will become clear once the semantics is spelled out. But now the OWL axioms or formulas.

{	extit{OWL axioms}} are formulas expressing that individuals belong to classes, that two classes are equivalent, or that one is a subset of the other (class inclusion axioms), and similarly for properties; equality and inequality between individuals can also be expressed. A finite sets of axioms concerning class and property inclusion is called a \textit{T-Box} (terminological knowledge), whereas a finite set of axioms concerning individuals is called \textit{A-Box} (assertive knowledge, or knowledge about state of affairs in the world).

\textbf{Definition 4 (OWL-Axiom).} The set of OWL axioms is divided into $\mathcal{T}$-axioms and $\mathcal{A}$-axioms (thus, mirroring the distinction between T-Box and A-Box):

\begin{align*}
\mathcal{T}\text{-Axiom} &::= C_1 \sqsubseteq C_2 \mid C_1 \equiv C_2 \mid P_1 \sqsubseteq P_2 \mid P_1 \equiv P_2 \mid D_1 \sqsubseteq D_2 \mid D_1 \equiv D_2 \mid T_1 \sqsubseteq T_2 \mid T_1 \equiv T_2 \\
\mathcal{A}\text{-Axiom} &::= C(i) \mid P(i_1, i_2) \mid i_1 = i_2 \mid i_1 \neq i_2 \mid D(l) \mid T(l_1, l_2) \mid l_1 = l_2 \mid l_1 \neq l_2
\end{align*}

Besides OWL axioms there are some more OWL sentences that can be expressed, call them \textit{OWL facts}. They concern classes being disjoint, or properties being transitive, symmetric inverse or functional. They will simply be referred to by \texttt{Disjoint($C_1, C_2$)}, \texttt{Trans($P$)}, \texttt{Symm($P$)} and \texttt{Func($P$)} (for individual classes and properties); \texttt{Disjoint($D_1$)} (for datatype classes.) The inverse of an individual property $P$ will be referred to by $P^{-1}$. Notice that datatype properties cannot be said to be inverse, symmetric or transitive.\footnote{This is another OWL DL expressive limitation.}

We are now ready to present the \textit{OWL semantics}. We will proceed in three steps. First, we will define an OWL model, namely an interpretation function for atomic classes, datatypes, properties, etc. Then, we will extend this interpretation function to complex classes. Finally, we will define truth conditions for the OWL axioms and entailment relation between ontologies.
Definition 5. A OWL model $\mathcal{M}_{\text{OWL}}$ is a tuple $(R, R_D, R_O, I_C, I_P, I_I, I_L)$. The set $R$ is the domain of resources, with $R_O \subseteq R$ the set of objects or individuals, and $R_D \subseteq R$ the set of data types or literal values (i.e. either the literal itself, if it is a plain literal, or its proper value, if it is a typed literal). Notice that $R_D \cap R_O = \emptyset$. Each $I_i$ is an interpretation function for classes, properties, individual names and literals. More precisely:

- $I_C(\text{owl:Thing}) = R_O \subseteq R$
- $I_C(\text{owl:Nothing}) = \{\} \subseteq R$
- $I_C(\text{owl:Literal}) = R_D \subseteq R$
- $I_C: V_C \mapsto \varphi(R_O)$
- $I_C: V_D \mapsto \varphi(R_D)$
- $I_P: V_{DP} \mapsto \varphi(R_O \times R_D)$
- $I_P: V_{IP} \mapsto \varphi(R_O \times R_O)$
- $I_P: V_{AP} \cup \{\text{rdf:type}\} \mapsto \varphi(R \times R)$
- $I_P: V_{OP} \cup \{\text{rdf:type}\} \mapsto \varphi(R \times R)$
- $I_I: V_I \mapsto R_O$
- $I_L: V_L \mapsto R_D$

In the above definition for the OWL model, the interpretation for atomic classes and properties is given. The interpretation of an atomic class (individual or datatype) is a subset of $R_O$ or $R_D$. Roughly, the interpretation of an atomic property (individual or data type) is a subset of $R_O \times R_O$ or $R_O \times R_D$. Table 3.1 gives a recursive extension to complex classes (second step in our exposition of the OWL DL semantics).
<table>
<thead>
<tr>
<th><strong>AWL-Syntax</strong></th>
<th><strong>Semantics</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg C$</td>
<td>$\mathbb{R}_O \setminus I_C(C)$ or simply $\overline{I_C(C)}$</td>
</tr>
<tr>
<td>$C_1 \cap C_2$</td>
<td>$I_C(C_1) \cap I_C(C_2)$</td>
</tr>
<tr>
<td>$C_1 \cup C_2$</td>
<td>$I_C(C_1) \cup I_C(C_2)$</td>
</tr>
<tr>
<td>$\forall P.C$</td>
<td>${o \in \mathbb{R}_O: \langle o, o' \rangle \in I_P(P) \Rightarrow o' \in I_C(C), \text{for all } o' }$</td>
</tr>
<tr>
<td>$\exists P.C$</td>
<td>${o \in \mathbb{R}_O: \langle o, o' \rangle \in I_P(P) &amp; o' \in I_C(C), \text{for some } o' }$</td>
</tr>
<tr>
<td>$n \leq P$</td>
<td>${o \in \mathbb{R}_O: #{o' : \langle o, o' \rangle \in I_P(P)} \leq n}$</td>
</tr>
<tr>
<td>$\forall T.D$</td>
<td>${o \in \mathbb{R}_O: \langle o, d \rangle \in I_P(T) \Rightarrow d \in I_C(D), \text{for all } d}$</td>
</tr>
<tr>
<td>$\exists T.D$</td>
<td>${o \in \mathbb{R}_O: \langle o, d \rangle \in I_P(T) &amp; d \in I_C(D), \text{for some } d}$</td>
</tr>
<tr>
<td>$n \leq T$</td>
<td>${o \in \mathbb{R}_O: #{d : \langle o, d \rangle \in I_P(T)} \leq n}$</td>
</tr>
<tr>
<td>$\text{OneOf}(i_1, \ldots, i_k)$</td>
<td>${I_T(i_1), \ldots, I_T(i_k)}$</td>
</tr>
<tr>
<td>$\text{OneOf}(l_1, \ldots, l_k)$</td>
<td>${I_L(l_1), \ldots, I_L(l_k)}$</td>
</tr>
</tbody>
</table>

**Table 3.1:** Semantics for OWL DL concept constructors.
From Table 3.1 it is apparent that classes may be constructed using datatype-properties or classes in $V_D$ and $V_{DP}$, but the denotation of class constructs is always a subset of $\mathbb{R}_O = I_C(owl:Thing) = I_C(\top)$. Hence, the object domain $\mathbb{R}_O$ is the proper domain and the datatype domain $\mathbb{R}_D$ is auxiliary. We now turn to the final step—giving truth-conditions for OWL axioms.

**Definition 6.** The OWL truth relation is defined for OWL axioms as follows:

\[
\begin{align*}
\mathfrak{M}_{OWL} \models C(i) & \text{ iff } I_I(i) \in I_C(C) \\
\mathfrak{M}_{OWL} \models C_1 \subseteq C_2 & \text{ iff } I_C(C_1) \subseteq I_C(C_2) \\
\mathfrak{M}_{OWL} \models C_1 \equiv C_2 & \text{ iff } I_C(C_1) = I_C(C_2) \\
\mathfrak{M}_{OWL} \models P(i_1, i_2) & \text{ iff } \langle I_I(i_1), I_I(i_2) \rangle \in I_P(P) \\
\mathfrak{M}_{OWL} \models P_1 \subseteq P_2 & \text{ iff } I_P(P_1) \subseteq I_P(P_2) \\
\mathfrak{M}_{OWL} \models P_1 \equiv P_2 & \text{ iff } I_P(P_1) = I_P(P_2) \\
\mathfrak{M}_{OWL} \models i_1 = i_2 & \text{ iff } \langle I_I(i_1), I_I(i_2) \rangle \in I_P(\equiv) \\
\mathfrak{M}_{OWL} \models i_1 \neq i_2 & \text{ iff } \langle I_I(i_1), I_I(i_2) \rangle \notin I_P(\equiv) \\
\mathfrak{M}_{OWL} \models D(l) & \text{ iff } I_L(l) \in I_C(D) \\
\mathfrak{M}_{OWL} \models D_1 \subseteq D_2 & \text{ iff } I_C(D_1) \subseteq I_C(D_2) \\
\mathfrak{M}_{OWL} \models D_1 \equiv D_2 & \text{ iff } I_C(D_1) = I_C(D_2) \\
\mathfrak{M}_{OWL} \models T(l_1, l_2) & \text{ iff } \langle I_L(l_1), I_L(l_2) \rangle \in I_T(T) \\
\mathfrak{M}_{OWL} \models T_1 \subseteq T_2 & \text{ iff } I_P(T_1) \subseteq I_P(T_2) \\
\mathfrak{M}_{OWL} \models T_1 \equiv T_2 & \text{ iff } I_P(T_1) = I_P(T_2) \\
\mathfrak{M}_{OWL} \models l_1 = l_2 & \text{ iff } \langle I_L(l_1), I_L(l_2) \rangle \in I_T(\equiv) \\
\mathfrak{M}_{OWL} \models l_1 \neq l_2 & \text{ iff } \langle I_L(l_1), I_L(l_2) \rangle \notin I_T(\equiv)
\end{align*}
\]

Additionally, we need to specify the truth relation for OWL fact:

\[
\begin{align*}
\mathfrak{M}_{OWL} \models \text{DISJOINT}(C_1, C_2) & \text{ iff } I_C(C_1) \cap I_C(C_2) = \emptyset \\
\mathfrak{M}_{OWL} \models \text{TRANS}(P) & \text{ iff } \langle o_1, o_2 \rangle \in I_P(P) \land \langle o_2, o_3 \rangle \Rightarrow \langle o_1, o_3 \rangle \in I_P(P) \text{ for all } o_i \in \mathbb{R}_O \\
\mathfrak{M}_{OWL} \models \text{FUNC}(P) & \text{ iff } \langle o_1, o_2 \rangle \in I_P(P) \land \langle o_1, o_3 \rangle \Rightarrow o_2 = o_3 \\
\mathfrak{M}_{OWL} \models \text{SYMM}(P) & \text{ iff } \langle o_1, o_2 \rangle \in I_P(P) \Leftrightarrow \langle o_2, o_1 \rangle \in I_P(P) \text{ for all } o_i \in \mathbb{R}_O \\
\mathfrak{M}_{OWL} \models P' = P^{-1} & \text{ iff } \langle o_1, o_2 \rangle \in I_P(P') \Leftrightarrow \langle o_2, o_1 \rangle \in I_P(P) \text{ for all } o_i \in \mathbb{R}_O \\
\mathfrak{M}_{OWL} \models \text{DISJOINT}(D_1, D_2) & \text{ iff } I_C(D_1) \cap I_C(D_2) = \emptyset
\end{align*}
\]

**Definition 7.** The ontology satisfaction, consistency and entailment is defined as follows: Given a vocabulary $V$, and an ontology $\mathcal{O} \models T\text{-axiom} \cup A\text{-axioms}$, we
have

\[
\mathcal{M}_{\text{OWL}} \models \sigma \quad \text{iff} \quad \mathcal{M}_{\text{OWL}} \models \varphi, \text{ for all } \varphi \in \sigma
\]

and any linguistic item in \( \sigma \) is backed in \( V \)

\[
\sigma \nvdash \bot \quad \text{iff} \quad \mathcal{M}_{\text{OWL}} \models \sigma, \text{ for some } \mathcal{M}_{\text{OWL}}
\]

\[
\sigma \models \sigma' \quad \text{iff} \quad \mathcal{M}_{\text{OWL}} \models \sigma \Rightarrow \mathcal{M}_{\text{OWL}} \models \sigma', \text{ for all } \mathcal{M}_{\text{OWL}}
\]

A remark about the relationship between OWL DL and DLs is in order. We know that OWL DL is equivalent to a DL, namely \( \text{SHOIN}(D) \). The mnemonics here are very useful (watch the capital letters!): \( S \) stands for role tranSitivity; \( H \) for properties Hierarchy; \( O \) for nOminals (i.e. sets consisting of a list of individual names or literals); \( I \) for property Inverse; \( N \) for unqualified Number restriction; \( D \) for Datatype. If we add these expressive features to the standard DL language, denoted by \( \text{ALC} \), which consists of class negation, class intersection, class union, existential and universal restriction, class inclusion and assertive axioms —e.g. \( C_1 \sqsubseteq C_2, (C(a)) — \) then we roughly get the same as the OWL DL language. Some formulas in OWL DL might be accommodated by using syntactic sugar. For a formal proof of the equivalence between OWL DL and \( \text{SHOIN}(D) \), see [Volz, 2004a].

Despite OWL DL being equivalent to the DL \( \text{SHOIN}(D) \), we preferred to describe OWL DL in terms of OWL DL itself, whenever possible. The syntax was stated in a human readable way. The semantics was, instead, stated from the standpoint of OWL itself, not its equivalent DL. The difference between the two ways is fortunately minimal, namely that the domain of the interpretation function can be an abstract set of constants or number names, or a set of OWL DL concrete syntax strings (given by URIs and literals). The second option is the one we chose, since we intended to give a semantics that was as much uniform to the RDF(S) semantics as possible.

### 3.3.1 Why some rules cannot be represented in OWL DL?

We will now address more precisely the question we posed at the beginning—why OWL DL cannot represent certain rules and which they are. It is not enough to say that one is not able to see any way to express a rule in OWL DL. What we need is a semantic characterization of the expressive limits of OWL DL. Here we sketch how this can be done, while limiting our attention to the Description Logic \( \text{ALC} \) (the simplest DL).

\[
\text{ALC-Class} ::= C | \neg C | C_1 \cap C_2 | \forall P.C | \exists P.C.
\]

The DL \( \text{ALC} \) can be translated into the multi-modal logic \( \text{K}_m \),\(^9\) whose language is defined as follows:

\[
\text{ML} ::= p | \neg \varphi | \varphi_1 \land \varphi_2 | \langle P^i \rangle \varphi | [P^i] \varphi.
\]

The semantics for non-modal formulas is standard (see FOL), while for modal formulas we have:

\(^9\)See [de Rijke, 1998] for an overview of the translation. For a more precise treatment, see [Baader et al., 2003], Chapter 4.
**Definition 8.** Let $\mathcal{M}_{\mathcal{ML}} = \langle W, I, \{P^i : i \in \mathbb{N}\} \rangle$, where $W$ is a set of possible worlds, $I$ is the interpretation function and $\{P^i : i \in \mathbb{N}\}$ is a family of accessibility relations between worlds. Let $P^i(w) := \{v : \langle w, v \rangle \in P^i\}$. The truth-conditions for modal formulas are as follows:

$$
\mathcal{M}_{\mathcal{ML}}, w \models \langle P^i \rangle \varphi \quad \text{iff} \quad \mathcal{M}_{\mathcal{ML}}, v \models \varphi, \text{ for some } v \in P^i(w);
$$

$$
\mathcal{M}_{\mathcal{ML}}, w \models [P^i] \varphi \quad \text{iff} \quad \mathcal{M}_{\mathcal{ML}}, v \models \varphi, \text{ for all } v \in P^i(w).
$$

One can easily see that the modal formulas $[P] \varphi$ and $\langle p \rangle \varphi$ are equivalent to the DL concepts $\forall P.C$ and $\exists P.C$. Furthermore, there is a meaning-preserving translation between $\mathcal{ML}$ and $\mathcal{ALC}$ (see [Baader et al., 2003] for details). Now, the point is the following: Modal formulas are invariant under bi-simulation,\(^{10}\) i.e., if there is a bi-simulation between two models and yet these are different, such a difference cannot be witnessed by modal formulas nor DL formulas (see [Blackburn et al., 2001], Theorem 2.20).

The result of modal and DL formulas as invariant under bi-simulation allows us to explain why the rule:

$$
\forall x, y : \text{Judge}(x) \land \text{condemn}(x,y) \land \text{Innocent}(y) \rightarrow \text{unjust}(x,y).
$$

cannot be expressed in OWL DL (and a fortiori neither in $\mathcal{ALC}$). Consider these two different Kripke models:

```
\begin{tikzpicture}
    \node (x) at (0,0) {$x$};
    \node (y) at (1,1) {$y$};
    \node (y1) at (2,0) {$y_1$};
    \node (y2) at (2,2) {$y_2$};
    \node (x') at (4,0) {$x'$};
    \draw (x) -- (y) node [midway, above] {unjust};
    \draw (y) -- (y1) node [midway, above] {condemn};
    \draw (y) -- (y2) node [midway, above] {condemn};
    \draw (x') -- (y1) node [midway, above] {unjust};
    \draw (x') -- (y2) node [midway, above] {condemn};
\end{tikzpicture}
```

Clearly, there is a bi-simulation between $\mathcal{M}$ and $\mathcal{M'}$, and so the two models must make true the same modal formulas (and also DL formulas, because of the meaning-preserving translation). However, the rule we are considering is true in $\mathcal{M}$ but false in $\mathcal{M'}$, and hence it cannot be (expressed as) a modal or DL formula.

The above sketchy explanation needs to be refined and extended to the entire OWL DL or $\mathcal{SHOIN}(D)$. For reasons of time and space, we leave the treatment of this matter at this stage for now.

---

\(^{10}\)The definition of bi-simulation runs as follows: A non-empty relation $Z \subseteq W \times W$ between possible worlds $w \in W$ is a bi-simulation between models $\mathcal{M}, \mathcal{M}'$ iff

(i) if $\langle w, w' \rangle \in Z$, then $w, w'$ make true the same propositional letter;

(ii) if $wZw'$ and $wRv$, then there is a $v'$ in $\mathcal{M}'$, such that $vZv'$ and $w'Rv'$ (zig condition);

(iii) if $wRv'$ and $w'Rv'$, then there is a $v$ in $\mathcal{M}$, such that $vRv'$ and $wRv$ (zag condition).
3.4 Description Logic Programs

At the beginning of this chapter, we motivated why OWL DL needs to be integrated with a rule language. A relevant proposal for integrating OWL and rules is DLP (Description Logic Program), which is a fragment of DL that can be translated into a rule language. Note that this proposal is not particularly attractive with regard to expressiveness, and yet it is interesting because it singles out a fragment that can be fully translated from DL to a rule language, and vice versa.

Roughly, DLP is the intersection of two fragments of FOL, namely DL and Horn Logic programs (that are function-free). Horn logic programs are languages composed by rules of the shape:

\[ A_1 \land \ldots \land A_{k-1} \rightarrow A \]

where \( A_1, \ldots, A_{k-1} \) and \( A \) are FOL atoms. \( A \) is the head (consequent) of the rule; and the (possibly empty) finite conjunction \( A_1 \land \ldots \land A_k \) is the body (antecedent) of the rule. Rules will be denoted by \( r \). There are \( k \) universal quantifiers that scope over the entire rule and that binds the variables in the rule. Note that Horn Logic programs are a subset of LP (Logic Programs), in the sense that the latter allows negated atoms\(^{11} \) to occur in the antecedent of rules.

A precise syntactic characterization of the language DLP is difficult to give, because of the different limitation that the intersection of DL and LP imposes on DLP. The limitation that DL imposes are as follows:

- no function symbol should be used;
- the predicates in the atoms of a rule can have at most arity 2;
- any two-place predicate should be guarded by a one place predicate (e.g. \( P(x, y) \land A(y) \));
- the language is equality-free.

On the other hand, here is a significant sample of the DL axioms that cannot be expressed in DLP:

\[
\begin{align*}
A & \subseteq B_1 \cup B_2; \\
\forall P. A & \subseteq B; \\
\neg A & \subseteq B; \\
\leq 2P. 
\end{align*}
\]

One can easily see that the above axioms, when translated in FOL, do not correspond to the shape of a rule:

\[
\begin{align*}
\forall x : A(x) & \rightarrow B_1(x) \lor B_2(x); \\
\forall x : (\forall y P(x, y) & \rightarrow A(y)) \rightarrow .B(x);
\end{align*}
\]

\(^{11}\)However, at the semantic level, negation as failure is used and not classical negation.
\[ \forall x : \neg A(x) \rightarrow B(x); \]
\[ \forall x : (P(x, y_0) \land p(x, y_1) \land p(x, y_2)) \rightarrow (y_0 = y_1 \lor y_1 = y_2 \lor y_0 = y_2). \]

The above rules cannot be expressed in LP forth following reasons: The first rule contains a disjunction in the consequent; the second one contains an embedded implication in the antecedent; the third one contains negations: and the last one contains equality.

The semantics of rules in DLP is standard: whenever the antecedent (body) is true, the consequent (head) is true.

### 3.5 SWRL: Syntax and Semantics

In this section, OWL DL is extended to SWRL (Semantic Web Rule Language), which is a combination of OWL DL and Rule ML (Rule Markup Language). For those who want to to gain more expressiveness, SWRL is more interesting than DLP, because it preserves the full expressiveness of DL augmented with a rule language. This has the costs of rendering SWRL an undecidable language. For this reason, we present the the strongly safe subset of SWRL as the best combination of OWL and rules, in so far as it preserves decidability while minimizing losses in expressiveness.

The main reason to choose SWRL as a working starting point, and not other rule languages, is that SWRL is becoming a new standard, since the W3C consortium has suggested it as the language for incorporating rules into OWL (see [Horrocks et al., 2004a]). Similarly to OWL DL, it is wiser to place ourselves within a recognized standard than to aim at designing our own language from scratch.

The syntax of \( SHOIN(D) \)—the DL equivalent to OWL DL whose syntax we used to describe the OWL syntax—is certainly not enough to write rules. So we need to rely on the FOL syntax. However, given that SWRL is a proper extension of the OWL DL language, it makes no sense to express SWRL formulas partly in DL syntax and partly in FOL syntax. Fortunately enough, there is a straightforward translation between DL formulas and FOL formulas. The reader may profit from consulting Table 3.2, where the FOL equivalent formulas to DL formulas are given. For a formal treatment of the translation between DL and FOL, see [Volz, 2004a]. Hereafter, we then assume that all SWRL formulas are expressed in the syntax of FOL, according to the equivalence given in Table 3.2.
<table>
<thead>
<tr>
<th>DL-SYNTAX</th>
<th>FOL-SYNTAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊤</td>
<td>( C(x) \lor \lnot C(x) )</td>
</tr>
<tr>
<td>⊥</td>
<td>( C(x) \land \lnot C(x) )</td>
</tr>
<tr>
<td>( \neg C )</td>
<td>( \neg C(x) )</td>
</tr>
<tr>
<td>( C_1 \sqcap C_2 )</td>
<td>( C_1(x) \land C_2(x) )</td>
</tr>
<tr>
<td>( C_1 \sqcup C_2 )</td>
<td>( C_1(x) \lor C_2(x) )</td>
</tr>
<tr>
<td>( \forall P.C )</td>
<td>( \forall y(P(x, y) \rightarrow C(y)) )</td>
</tr>
<tr>
<td>( \exists P.C )</td>
<td>( \exists y(R(x, y) \land C(y)) )</td>
</tr>
</tbody>
</table>
| \( n \leq P \) | \( \forall y_1, \ldots, y_{n+1}: \)  
| | \( P(x, y_1) \land \ldots P(x, y_{n+1}) \rightarrow \)  
| | \( \bigvee_{i<j}(y_i = y_j) \) |
| \( n \leq P.C \) | \( \forall y_1, \ldots, y_{n+1}: \)  
| | \( P(x, y_1) \land \ldots R(x, y_{n+1}) \land \)  
| | \( C(y_1) \land \ldots C(y_{n+1}) \rightarrow \)  
| | \( \bigvee_{i<j}(y_i = y_j) \land \) |
| \( C(a) \) | \( C(a) \) |
| \( C_1 \sqsubseteq C_2 \) | \( C_1(x) \rightarrow C_2(x) \) |
| \( C_1 \equiv C_2 \) | \( C_1(x) \leftrightarrow C_2(x) \) |
| \( P(a, b) \) | \( P(a, b) \) |
| \( P_1 \sqsubseteq P_2 \) | \( P_1(x, y) \rightarrow P_2(x, y) \) |
| \( P_1 \equiv P_2 \) | \( P_1(x, y) \leftrightarrow P_2(x, y) \) |

**Table 3.2:** Correspondence between DL and FOL syntax
The **SWRL vocabulary** is the OWL DL vocabulary $V$ with the addition of the sets:

- $V_{IX}$ for individual variables, denoted by $x, y, z$;
- $V_{DX}$ for data type variables, denoted by $m, n$;
- $V_{\text{built-in}}$ for built-in names.

It is useful to introduce two notational shortcuts. Any variable in $V_{IX}$ or name in $V_I$ will be called **object term**, and denoted by $t$, with indices if necessary. Likewise, a **datatype term**, denoted by $v$, is either a literal in $V_L$ or a datatype variable in $V_{DX}$.

The **SWRL logical vocabulary** extends the OWL logical vocabulary with $\rightarrow$ and $\land$.

Strictly speaking, since SWRL is stated in FOL, its logical vocabulary is equivalent to that of FOL, namely truth-functional connectives ($\land, \lor, \rightarrow, \neg$) and quantifiers ($\exists, \forall$).

**Definition 9.** The set of **SWRL atoms** is defined by the following rule:

\[
\text{SWRL-Atom} ::= C(t) \mid D(v) \mid P(t_1, t_2) \mid T(t, v) \mid t_1 = t_2 \mid t_1 \neq t_2
\]

where $C$ is a OWL class (atomic or complex); $P \in V_{IP}$ a OWL atomic property, $D \in V_{DC}$ a OWL datatype class; and $T \in V_{DP}$ a OWL datatype property.

**Definition 10.** The set of **SWRL rules** is the smallest set constructed out of the SWRL atoms such such every element has the shape:

\[
A_1 \land \ldots \land A_k \rightarrow A,
\]

where $A_i$’s and A are SWRL atoms. $A$ is the head (consequent) of the rule; and the (possibly empty) finite conjunction $A_1 \land \ldots \land A_k$ is the body (antecedent) of the rule. Rules will be denoted by $\mathcal{r}$. There are $k$ universal quantifiers that scope over the entire rule and that binds the variables in the rule. They can be ordered as one pleases, since they are only universal quantifiers, and there is no harm is swapping them.

**Definition 11.** The set of **SWRL formulas** is the smallest that includes the sets:

- OWL classes (i.e., their FOL translation);
- OWL axioms and facts (i.e., their FOL translation);
- SWRL rules.

**Definition 12.** A **SWRL model**—denoted by $\mathfrak{M}_{\text{SWRL}}$—extends the $\mathfrak{M}_{\text{OWL}}$ models with assignment function $g := g_I \cup g_D$, where:

- $g_I : V_{IX} \rightarrow \mathbb{R}_O$
- $g_D : V_{DX} \rightarrow \mathbb{R}_D$
**Definition 13.** The SWRL truth relation extends the OWL truth relation with the following clauses:

\[
\mathcal{M}_{SWRL} \models C(t) \quad \text{iff} \quad g_I \cup I_I(t) \in I_C(C)
\]
\[
\mathcal{M}_{SWRL} \models P(t_1, t_2) \quad \text{iff} \quad \langle g_I \cup I_I(t_1), g_I \cup I_I(t_2) \rangle \in I_P(P)
\]
\[
\mathcal{M}_{SWRL} \models t_1 = t_2 \quad \text{iff} \quad \langle g_I \cup I_I(t_1), g_I \cup I_I(t_2) \rangle \in I_P(=)
\]
\[
\mathcal{M}_{SWRL} \models t_1 \neq t_2 \quad \text{iff} \quad \langle g_I \cup I_I(t_1), g_I \cup I_I(t_2) \rangle \notin I_P(=)
\]
\[
\mathcal{M}_{SWRL} \models C(v) \quad \text{iff} \quad g_D \cup I_L(v) \in I_C(C)
\]
\[
\mathcal{M}_{SWRL} \models T(t, v) \quad \text{iff} \quad \langle g_I \cup I_I(t), g_D \cup I_L(v) \rangle \in I_P(D)
\]
\[
\mathcal{M}_{SWRL} \models A_1 \land \ldots \land A_2 \rightarrow A \quad \text{iff} \quad \mathcal{M}_{SWRL} \models A_1 \land \ldots \land A_2 \Rightarrow \mathcal{M}_{SWRL} \models A
\]

The style of the truth clauses does not present any substantial difference from truth clauses of FOL (see, e.g., [van Dalen, 2004], [Gamut, 1991]). The only non-standard notation are the function symbols \(g_I \cup I_I\) and \(g_D \cup I_L\), which take care respectively of individual names and variables, and datatype names (literals) and variables. Any individual \(t\) is pure, in the sense that does contain any datatype term, and vice versa for datatype terms. The interpretation of the symbol = is to be understood as a set of pairs of the form \(\langle r, r \rangle\) for any \(r \in \mathbb{R}\).

3.5.1 The Strongly Safe Subset of SWRL

It is known that the arbitrary combination of OWL DL and a rule language, resulting e.g. in SWRL, is undecidable (see [Horrocks and Patel-Schneider, 2004a]).

How do we retain decidability while keeping the language sufficiently expressive? This question follows a traditional leitmotiv in KR, i.e., the tradeoff between expressiveness and tractability/decidability. The way to regain decidability is obviously by limiting the formation of rules in certain ways. How?

Preliminarily, some additional notation and terminology is useful: A program \(\mathcal{P}\) is a set of rules; a terminological knowledge base or ontology or T Box will be denoted by \(T\); a assertive knowledge base or A Box will be denoted by \(A\); finally, a OWL atom is a SWRL atom (see Definition 9) that occurs in \(T\).

One may ask: If everything in SWRL (or any subset of it) is expressed in FOL, then syntactically there is no difference between OWL axioms and rules, as they both have the form of an implication. This is correct. Let us say, then, that a rule is a formula in SWRL that is not expressible in OWL DL (recall 3.3.1).

Concerning a decidable combination of OWL DL and rules, there are at least three possible guiding questions:

1. What kind of atoms to place in rules?
2. Where to place atoms in rules, i.e., head, body or both?
3. How to handle variables (safety conditions)?

In response to question 1, one may allow only a restricted range of SRWL atoms to appear in rules, e.g., only classes and not properties, or only object classes or properties and not datatypes, etc. A systematic treatment of this issue is missing.
from the literature, but it is not clear whether an investigation in such direction
will be at all productive—on the decidability side and on the expressiveness side.
At first, it seems we cannot avoid the use of both classes and properties in rules.
Hence, limiting the construction of rules this way does not seem to be fruitful.

To question 2 there are roughly three possible responses, which give immediately
different alternatives as to how \( \mathcal{T} \) and \( \mathcal{P} \) reciprocally relate. The first alternative is
that OWL atoms are allowed to occur only in the antecedent (body) of rules, so that
\( \mathcal{P} \) is built on the top of \( \mathcal{T} \). This proposal can be found, e.g., in [Donini et al., 1998].
The other alternative is symmetric—allowing OWL atoms only in the consequent
of rules, so that \( \mathcal{T} \) is built on the top of \( \mathcal{P} \). The previous two options allow only
a one-way interaction between \( \mathcal{P} \) and \( \mathcal{T} \), whereas the third possibility allows a bi-
directional interaction between \( \mathcal{T} \) and \( \mathcal{A} \). Which one to choose?

If OWL atoms are allowed only in the consequent (head) of rules, or viceversa,
it seems we are restricting the language too much. Intuitively we want rules whose
antecedent contains concepts and properties already defined in \( \mathcal{T} \), otherwise what
would these rules be devised for? Hence, the second alternative is to be discarded.

What about the first? That OWL atoms occur only in the antecedent of rules
seems more plausible. However, it is reasonable to assume that what we derive from
a rule is something which our terminological knowledge \( \mathcal{T} \) can say what it is, oth-
wise the terminological knowledge would be blind with respect to the derivations
in \( \mathcal{P} \). This line of reasoning leads us to the conclusion that we may require any
atoms occurring in a rule to be already specified in \( \mathcal{T} \). This is feasible, since \( \mathcal{T} \)
defines the ontological “horizon”, and the limits of it should not be transgressed.
Unluckily enough, it is exactly this requirement that makes the integration of \( \mathcal{T} \) and
\( \mathcal{P} \) particularly problematic, and that makes, in turn, the language undecidable.

The moral is that it is preferable to allow OWL atoms to appear in both the
consequent and the antecedent (hence alternatives one and two must be discarded).
Nevertheless, one should be aware that this choice is very problematic, and so one
should be prone to accept expressive limitations of other kind (i.e., safety conditions
on rules; see later on).

As a gloss to the possible responses to question 2, we note that \( \mathcal{T} \) and \( \mathcal{P} \) do
interact, in the sense that they may update the other component with a rule or an
axiom. Here are two simple examples. Consider the following two rules:

\[
\forall x, y : A(x) \land R(x,y) \rightarrow B(y) : \\
\forall x, y : A(x) \land R(x,y) \rightarrow C(y).
\]

One can see that we can prove:

\[
\exists x : B(x) \land C(x),
\]

that is, in OWL DL:

\[
B \sqcap C \subseteq \top.
\]

To see that, assume, for contradiction, \( \exists x : (B(x) \land C(x)) \rightarrow \bot \). Moreover, assume for
arbitrary \( a, b \) that \( A(b) \) and \( R(b,a) \). Thus, by the two rules, we have \( B(a) \) and \( C(a) \),
and by the assumption for contradiction we get \( \bot \). Thus, \( \neg A(b) \) or \( \neg R(b,a) \). Since
a, b were arbitrary, we get $\forall x : \neg A(x)$, which means that $A$ is itself inconsistent. Thus—unless we assume that the rules are vacuously satisfied (and thus the rules are meaningless)—we have a contradiction, whence $B \sqcap C \subseteq \top$.

This shows that from some rules in $\mathcal{P}$ a new and non-trivial OWL axiom in $\mathcal{T}$ may be derivable. For the other direction—OWL DL axioms giving rise to new rules—suppose in $\mathcal{T}$ we have

$$A \sqsubseteq B,$$

and that in $\mathcal{R}$ we have

$$B(x) \land R(x, y) \rightarrow C(y).$$

Then, we also get the new rule:

$$A(x) \land R(x, y) \rightarrow C(y).$$

Great attention in the literature is paid towards the possible responses to question 3, concerning how to handle variables. This issue falls under the heading of safety condition or safe-rules condition. Following [Parsia et al., 2004], we may distinguish two types of safety conditions, to which the general safety condition must be added.

**Definition 14.** A rule $r \in \mathcal{P}$ is said to be safe iff, whenever a variable (individual or datatype) occurs in the consequent (body) of $r$, it also occurs in the antecedent (body) of $r$. To this general condition, two refinements can be added:

- **weak:** if a variable occurs in $r$, then it occurs in some non-OWL atom in $r$;
- **strong:** if a variable occurs in $r$, then it occurs in some non-OWL atom of the antecedent (body) of $r$.

As a consequence, a program $\mathcal{P}$ is said to be safe (weakly or strongly) iff all its rules respect the safety condition (weakly or strongly). While guiding questions 1 and 2 did not deliver any substantial insight into how to preserve decidability in SWRL—better: they did deliver constraints that we believed to be too restrictive with respect to expressiveness—the safety conditions are crucial for retaining decidability, and deserve careful consideration.

We now give examples of rules that do not comply with the safety conditions. As a reminder, note that here we are only concerned with rules that are not expressible in OWL DL, i.e., rules where at least two variables are shared between antecedent and consequent (recall 3.1).

The general safety conditions is only violated when a new fresh variable is introduced in the consequent of a rule that was not already in the antecedent of the rule. This condition boils down to two major cases: Either an existential quantifier is introduced in the consequent (head) of a rule; or a universally quantified variable is used only in the consequent of a rule. The former case implies that the formula is not a SWRL rule anymore. Thus, we only need to take care of the latter case.

Consider the example:

$$\forall x, y, z, u : P_1(x, y) \land P_2(y, z) \rightarrow P_3(u, y) \land C(y),$$

This is not to say that a alike rule cannot be expressed in SWRL. It can be, since OWL concept can express existential quantification.
which is equivalent to

$$\forall x, y, z : P_1(x, y) \land P_2(y, z) \rightarrow \forall u : P_3(u, y) \land C(y).$$

Now, we assume that $P_1, P_2$ refer to non-empty relations—they could refer to the empty set, but then these properties would be useless for our knowledge modelling, since they would refer to nothing. Thus, we can safely assume that there are some individuals, denoted by $a, b, c$, such that $P_1(a, b) \land P_2(b, c)$. Given that, we immediately have that $\forall u : P_3(u, b)$. Thus the property $P_3$—one may say—is almost tautological, once we assume that the properties $P_1, P_2$ do not refer to the empty set.

What does the rule we are considering actually express? Simply, the rule says that whenever an individual $x$ satisfies certain conditions, $x$ is in $P_3$-relation with any individual in the domain (e.g., if $x$ is hotter than the sun, $x$ can melt everything). If so, one easily sees that the rule can be safely rephrased as:

$$\forall x, y, z, u : P_1(x, y) \land P_2(y, z) \land \top(u) \rightarrow P_3(u, y) \land C(y),$$

where $\top$ is the concept under which any individual in the domain falls. Hence, the general safety condition does not limit the expressiveness of SWRL, while weak and strong safety do, as we shall now see.

Concerning the weak safety condition, consider the rule we used at the beginning, call it $r_1$:

$$\text{JUDGE}(x) \land \text{condemn}(x, y) \land \text{INNOCENT}(y) \rightarrow \text{unjust}(x, y). \quad (r_1)$$

Rule $r_1$ is not by itself a non-weak safe rule. It would be non-weak safe, provided that $\text{JUDGE}$, $\text{INNOCENT}$ $\text{condemn}$, $\text{unjust}$ were already in the terminological knowledge $T$. Instead, if at most $\text{JUDGE}$, $\text{INNOCENT}$ and $\text{condemn}$ were, then $r_1$ would be weak safe. However, $r_2$ would still not meet the strong safety condition. To meet such a condition, we should have e.g. that at most $\text{JUDGE}$, $\text{INNOCENT}$ are in $T$. Obviously, the strong version of safety entails the weak version.

Concerning the strong weakness condition there are two results in the literature that are worth mentioning, and that address our problem of retaining decidability.

**Theorem 1.** [Motik et al., 2005] If a terminological knowledge base $T$ is in $\text{SHOIN}$, and a program $P$ contains only strongly safe rules, then query answering in $\langle T, P \rangle$ is decidable.

**Theorem 2.** [Rosati, 2006] If consistency checking is decidable in $T$ (where $T$ is any theory written in any logical language), and if a program $P$ contains only strongly safe rules, then consistency checking in $\langle T, P \rangle$ is still decidable.

The first result is limited to $\text{SHOIN}$. We need an analogous result for $\text{SHOIN}(\text{D})$. Fortunately, from the second result and from the fact that the reasoning in $\text{SHOIN}(\text{D})$ is decidable, we get the following corollary:

**Corollary 1.** [Rosati, 2006] If a terminological knowledge base $T$ is in $\text{SHOIN}(\text{D})$, and a program $P$ contains only strongly safe rules, then consistency checking in $\langle T, P \rangle$ is decidable.
The above results (though they do not guarantee tractability) provide a strong argument to choose the strongly safe subset of SWRL (hereafter, ssafe-SWRL), i.e., the subset of the SWRL language containing only rules that are strongly safe.

We started our investigation with the goal of preserving decidability and minimizing losses in expressive power. Hence, we ask: How much expressive power do we loose by adopting ssafe-SWRL? Let us look at an extended example. Consider a knowledge base $\mathcal{KB}$, where $\mathcal{A}, \mathcal{T}, \mathcal{P}$ as usual denote assertive and terminological knowledge base, and a program. We mix OWL DL syntax for axioms and facts with FOL syntax for rules, to make clear what the rules are. Assume we have:

$$
C(a) \in \mathcal{A}
$$

$$
[C \sqsubseteq D] \in \mathcal{T}
$$

$$
[\forall x, y : C(x) \land D(y) \rightarrow R(x, y)] \in \mathcal{P}
$$

The above rule is not safe since the variables $x, y$ do not occur in a non OWL atom in the body of the rule. That is the kind of limitation that ssafe-SWRL will confront us with. At first, this is a very annoying limitation. However, the first discouragement can be wiped out if some syntactic tricks are used, as presented in [Motik et al., 2005].

The trick consists of two things: (a) adding a ad hoc non-OWL atom, call it $O$, to the antecedent of the rule, and (b) adding $O(a)$ to $\mathcal{A}$, for every individual name $a$. Thus, we have, call it $\mathcal{KB}'$:

$$
O(a), C(a) \in \mathcal{A}
$$

$$
[C \sqsubseteq D] \in \mathcal{T}
$$

$$
[C(x) \land D(y) \land O(x) \land O(y) \rightarrow R(x, y)] \in \mathcal{P}
$$

Now the strongly safety condition is met. What is the semantic difference that the addition of the $O$-atom gives rise to? Does the trick affect the inferences we can make? The answer is ‘yes and no’. It does not, if we are making inferences about named individuals (individual which we refer to by individual names). It does, if we are making inferences about unknown (unnamed) individuals.

**Claim 1.** Given a $\mathcal{KB}$, in which we can derive the conclusion: $\psi(X)[N]$, for $\psi$ an arbitrary $n$-place relation, $X$ a sequence of variables and $N$ a sequence of individual names by which some variables in $X$ are replaced. Then, in $\mathcal{KB}'$, we can derive the same conclusion, provided $B \subseteq V_I$, where $B$ is the set of constants occurring in the derivation of the conclusion and $V_I$ is the set of fixed individual names.

**Proof.** Straightforward. If we can derive $\psi(X)(N)$, then we have $\mathcal{KB} \vdash \Phi(Y)[B] \rightarrow \psi(X)[N]$ (where $\Phi$ is a conjunction of atoms), and in $\mathcal{A}$, there are a number of assertions of the type $P(a_i)$, for $a \in N$. Suppose, in $\mathcal{A}$, we add $O(i_i)$, for every
$i_i \in V_I$, and we change the rule into $O(b_i) \land \ldots \land O(b_k) \land \Phi(Y)[B] \rightarrow \psi(X)[N]$. But we know $B \subseteq V_I$, thus we can also write the rule as $O(i_i) \land \ldots \land O(i_k) \land \Phi(Y)[B] \rightarrow \psi(X)[N]$, for $i_i = b_i$. By modus ponens, we get the original rule back, and thus also the original conclusion.

The main point is that $B \subseteq V_I$, i.e. the fixed individual names $V_I$ be at least as numerous as the constants $B$ used in the derivation of the conclusion. This is not always the case, since in a derivation we might use the rule of $\exists$-elimination, which introduces new arbitrary names, which were not in $V_I$. That the problem lies in rules containing the existential quantifier can be clarified with a new example that was inspired by [Motik et al., 2005], call it $\mathcal{KB}2$.

$$\exists \text{caused}\_by.\text{HELP}\_\text{ByBob}(\text{fire}) \in \mathcal{A}$$

$$\text{caused}\_by(z,y) \land \text{help}(x,y) \rightarrow \text{coresponsible}(x,z) \in \mathcal{R}$$

The content of $\mathcal{A}$ in FOL is equivalent to

$$\exists x \text{caused}\_by(\text{fire},x) \land \text{help}\_\text{by}(\text{Bob},x)$$

We instantiate $x$ with a fresh constant $b$, so we have:

$$\text{caused}\_by(\text{fire},b) \land \text{help}\_\text{by}(\text{Bob},b)$$

By the rule, we conclude:

$$\text{coresponsible}(\text{Bob},\text{fire}).$$

This conclusion cannot be drawn, if we adopt a strong safe rule, because we would lack $O(b)$, given that $b$ is not a part of the fixed constants $C$. I will call this problem the unknown individual problem.

The most obvious way to tackle this problem is to say the the set of individual constant $V_I$ is augmented with as many arbitrary constants as there are existentially quantified variables. This solution seems fine as long as the augmented knowledge base can derive at least as much as the initial knowledge base. Complications may arise, because the augmented knowledge base might actually derive more than expected. We know that a when a theory $T$ is modified such that constants are added for each existential variable, the resulting theory $T'$ is conservative over $T$. Our situation is more complicated, however, since the safety conditions must be taken care of as well. We leave the details of this discussion for future work.

To conclude, in this chapter we presented the semantics of RDF(S), OWL DL and SWRL. We also provided a strategy to characterize the expressive limitations of OWL DL (see 3.3.1), and we argued for the choice of ssafe-SWRL as the best (known) rule-extension of OWL DL. This choice is strengthened by the fact that some existing reasoning implementations which support ssae-SWRL are already available, for example, the reasoners KAON2 (which does not support nominals, though) and the latest release of Pellet (see B.3.1).
Chapter 4

Overview of LKIF

The Legal Knowledge Interchange Format (LKIF) is an OWL ontology of legal concepts, allowing legal knowledge bases to be represented in OWL. LKIF will define epistemological concepts for legal reasoning and argumentation, such as legal rules, meta-level rules for reasoning about rule priorities and exceptions, legal arguments, legal procedures, cases and case factors, values and principles. In addition, LKIF will define an ontology of generally useful substantive legal concepts, such as obligations, permissions, rights and powers, which can be reused when modeling specific legal domain, such as tax law. Here, in this report, some of the epistemological concepts of LKIF have been defined and illustrated. The ontology of substantive legal concepts is presented in a separate report, Deliverable D1.4.

Conceptually, the epistemological concepts of LKIF can be separated into the following layers:

**Terminological Layer (T)** is OWL, the description logic subset of OWL or less. OWL Full can be used, but has no semantics\(^1\).

**Extended Terminological Layer (ET)** is SWRL, the Semantic Web Rule Language, on top of OWL. This extends OWL with the datalog subset of first-order predicate logic (FOL). Although datalog, like OWL-DL, is a decidable subset of FOL, full SWRL is not decidable, since it provides the union of datalog and description logic.

**Rules Layer (R)**, which we will also call LKIF Rules, extends SWRL with support for negation and defeasible reasoning. The Rules layer provides a language expressive enough to model legal rules in an way which comes much closer to the ideal of *isomorphic modeling* [Bench-Capon and Coenen, 1992], i.e. a language which allows legal rules to be written in a way which reflects their structure in legislation. The rules layer supports rules with exceptions, assumptions, and exclusionary conditions, and enables meta-level information about rules to be represented, such as the date of enactment, and used in other rules, such as *lex posterior*, to reason about rule priorities.

**Extended Rules Layer (ER)** will add OWL classes for modeling legal precedents, such as factors or dimensions, as well as the reasons, arguments or

---

\(^1\)This comes down to saying it is possible to store OWL Full files, but a fragment of the contained RDF Graph must be selected before it can be reasoned about.
This report presents LKIF Rules, the LKIF language for modeling legal rules (LKIF Rules), and the terminological layers below that. The Extended Rules Layer will be handled in the next version of LKIF, to be developed in Work Package 4 of ESTRELLA. The rest of the report is organized as follows. First, we provide an overview of LKIF, and different dimensions in which choices can be made. The next chapter presents an abstract and several concrete syntaxes for parts of LKIF, including some graphical representations of LKIF.

Chapter 6 describes the semantics of epistemological concepts that have associated semantics that extends the simple terminological meaning that the ontology provides for it.

The concept of the four layers serves to identify the degree of support (T, ET, R, ER) of LKIF by software tools. This is separate from using LKIF, which is restricted to these layers. Let us introduce the notion of an LKIF reasoner, an LKIF file, and of using LKIF:

**Definition 15** (Using LKIF). To use LKIF is to refer to an entity in the LKIF namespace.

**Definition 16** (LKIF File). An LKIF file is a file which uses one of the concrete syntaxes described in this chapter, is limited to one of the expressive fragments identified in this chapter, and refers to an entity in the LKIF namespace.

**Definition 17** (LKIF Reasoner). An LKIF reasoner is a reasoner that completely and correctly applies the semantics of an LKIF layer (T, ET, R, ER) to an LKIF file.

### 4.1 Dimensions

LKIF consist of different sublanguages. The sublanguages differ in expressiveness, computational complexity, and required modelling effort. The user of LKIF can choose a sublanguage depending on his requirements. The available options can be separated in four dimensions.

The dimensions are:

**Ontology** Which files of the LKIF ontology are used, and which not? A trivial use of LKIF would for instance consist of importing the time ontology, while ignoring norms and argumentation. Of each OWL module, a DLP version and a DL version will be made available.

**Description Logic** Four fragments of OWL can be considered part of LKIF: OWL DLP, OWL DL, propositional OWL, and OWL Full. OWL DL and OWL DLP Semantics where given in chapter 3. OWL Full is not and will not be supported by reasoners, and is merely provided as an option for building extensions to LKIF. Propositional OWL is dealt with separately: it is merely a syntactical fragment of OWL DL, and it supported by OWL DL reasoners.
SWRL  Three fragments of SWRL can be considered part of LKIF: The DLP subset of SWRL and the DL-safe subset were defined in chapter 3, and no SWRL is also a valid option.

Special semantics  Two types of special semantics can currently be considered part of LKIF: argumentation as defined in section 6.2, and normative reasoning as defined in section 6.1.

The combination of four options, one selected from each dimension, results in an LKIF variant. Some combinations result in impossible or ineffectual logics; this section will describe some of the valid combinations.

The description logic dimension consists of layers with increasing expressiveness, that the different OWL sub-languages provide. OWL consists of the following sub-languages: OWL Lite, OWL DLP, OWL DL, and OWL Full. OWL Lite is not of interest for LKIF. We propose to add a fifth layer, propositional OWL, to the existing ones in the future, as a special option for existing expert systems that can be considered fragments of propositional logic. Sections B.1 and B.1.1 describe OWL and its fragments.

Propositional OWL is a way of using OWL. OWL can be used to represent (and reason with) conjunctions, disjunctions, and negations of propositions, all referring to one constant. Propositional OWL does not use properties. This is a limited from of using OWL: it does not take advantage of all capabilities of an OWL reasoner, but it can ease transition from the existing proprietary formats that are fragments of propositional logic. Therefore we recognize this use as the propositional OWL option. The expressiveness of the propositional OWL fragment is a subset of OWL DLP.

Interesting combinations are the following:

1. Propositional OWL and the DLP variant of the LKIF Ontology is the easiest to support for existing tools.
2. OWL DL and the DLP variant of SWRL, with the DL variant of the ontology is already supported by existing tools like the Pellet reasoner, SWOOP, TopBraid, and Protege.
3. OWL DL and the DLP variant of SWRL, with the DL variant of the ontology, and special semantics for normative reasoning can be easily supported with a preprocessor and existing technology.
4. OWL DLP, the DLP variant of SWRL, with the DLP variant of the ontology, and special semantics for argumentation is the most promising option for support of argumentation, and besides that many argumentation applications don’t make very high demands of the underlying reasoner.

4.2 Ontologies of Law

The LKIF ontology currently consists of the following modules, available as different OWL ontology files:
Expression covers a number of representational primitives necessary for describing relational mental states that connect a person to a proposition: e.g. beliefs, statements etc;

Norm describes the concepts most central to LKIF: e.g. norm, obligation, prohibition etc;

Argumentation describes concepts central to LKIF argumentation, and the components of LKIF Rules: e.g. assumption, exception, rule;

Processes describes concepts related to (involuntary) change;

Action covers concepts related to actions and their relation to physical change and intentions, e.g. action, agent etc;

Role describes constructs that underlie the roles being played by agents;

Place defines representational primitives for describing places, locations and the relations between them;

Time covers representational primitives for describing time intervals;

Mereology describes classes and properties that allow to express mereological relations, e.g. parthood, components etc.

These are considered the necessary components for explaining the types of frameworks, whether epistemological, situational, or mereological that are typical in law. Typical legal concepts (like burden of proof, potestative right, administrative appeal) can be described in terms of them, but are, maybe contrary to expectations of some readers, not themselves the focus of the work on LKIF.

4.3 MetaLex XML

In addition to specifying a standard for legal knowledge representation, the Estrella consortium has been heavily involved with the CEN/ISSS standardization effort around MetaLex XML\(^2\), and XML language for sources of law. MetaLex XML uses OWL DL to specify metadata of sources of law, which are organized around actions: for instance the action of publication of an act happens at a certain date, which is the date of publication of the act. MetaLex XML also standardizes the method to attach knowledge representation in OWL to sources of law. This subject is not discussed separately in this deliverable. It suffices to know that the Estrella project has contributed much to the CEN/ISSS MetaLex proposal, and that the definition of action in Estrella and the concept of action in MetaLex are very similar.

The CEN/ISSS proposal for MetaLex makes a distinction between versions of the expression – a substantially different text – and different versions of an XML manifestation of that same expression. A new manifestation version may be required because there is a modified interpretation, in particular when the former

\(^2\)http://www.metalex.eu/wiki/index.php/Main_Page
manifestation was encoded before the contents goes into force. The interpretation of force and efficacy of the content is different, and therefore metadata changes. The same thing applies to self-contained RDF files or other self-contained serializations of LKIF\(^3\): there is a modified LKIF interpretation of the same expression. It is therefore required that the ontology be dated and the distinction between an extension of the same interpretation (backwards compatible) and a different one (not backwards compatible) should be made, in accordance with the W3’s interpretation of this issue\(^4\). Note that this only applies to different interpretations of the same source, and not to interpretations of different expressions of the same work of law: these are simply different things.

The CEN/ISSS proposal for MetaLex and the deliverables of work package 3 are authoritative on this issue. See [Noy and Musen, 2004, Klein and Noy, 2003] for more information.

\(^3\)Any form besides embedded RDF/A statements in the MetaLex file.
\(^4\)http://www.w3.org/2001/sw/WebOnt/webont-issues.html#I5.14-Ontology-versioning
Chapter 5

Syntax of LKIF

All parts of LKIF can be represented in RDF, the abstract syntax used by OWL and SWRL. The main unit for defining RDF abstract syntax are RDF triples (see 3.2). A triple has three components:

- subject, which can be a RDF URI reference or a blank node;
- predicate, which can only be a URI reference;
- object, which can be a RDF URI reference, a blank node or a literal.

An RDF triple can be written in the linear form as \( \langle \text{subject}, \text{predicate}, \text{object} \rangle \). A set of triples is called an RDF Graph.

This means that any concrete syntax relates to RDF – for instance the standard XML serialization, RDF/A, N3, and ntriples. The next section discusses an open issue that could be approached as both an ontological one and a syntactical one: the way LKIF Rules fit within the whole RDF framework.

In addition to that, LKIF defines two linear syntaxes. The first one is closely related to a (non-authoritative) method already used in papers to describe description logic Tbox axioms, and a translator for OWL to this syntax already exists. LKIF extends it. The second one is a Lisp-style syntax for LKIF rules.

5.1 OWL Abstract Syntax of LKIF Rules

In this section, the OWL ontology for LKIF Rules is presented. This ontology is a simple extension of SWRL and will serve as an abstract syntax for LKIF rules.\(^1\) The ontology will be presented here as a set of RDF triples, using s-expressions of the form \( \langle \text{predicate} \text{subject} \text{object} \rangle \) for RDF triples.

Let us first define some prefixes for the XML namespaces we will be using:

\begin{verbatim}
  rdfs = http://www.w3.org/2000/01/rdf-schema
  owl = http://www.w3.org/2002/07/owl
  swrl = http://www.w3.org/2003/11/swrl
\end{verbatim}

\(^1\)As OWL has a concrete representation using XML, this OWL ontology also indirectly provides a concrete syntax for LKIF Rules. However OWL’s XML syntax is too obscure and verbose to be written or read directly by humans. This XML syntax is suitable as an machine readable interchange format for use by computer programs, but another, more readable, concrete syntax is defined in Section 5.1.2.
Symbols without a prefix represent URIs in the namespace of the LKIF Rules ontology being defined here. Here is the OWL ontology for LKIF Rules:

```
(rdfs:subClassOf Rule swrl:Imp)
(rdfs:subClassOf Rule swrl:Imp)
(rdfs:subClassOf ValidRule Rule)
(rdfs:subClassOf Assumption swrl:Atom)
(rdfs:subClassOf Exception swrl:Atom)
(rdfs:subClassOf NegatedAtom swrl:Atom)
(rdfs:type prior owl:ObjectProperty)
(rdfs:domain prior Rule)
(rdfs:range prior Rule)
(rdfs:type excluded owl:ObjectProperty)
(rdfs:domain excluded Rule)
(rdfs:range excluded Atom)
```

The `Rule` class is defined to be a subclass of SWRL’s `implication` class, `swrl:Imp`. The concept of a valid rule in LKIF is modeled in OWL as a subclass of the LKIF `Rule` class, named `ValidRule`.

Assumptions, exceptions and negated atoms are all defined to be subclasses of SWRL atoms, `swrl:Atom`.

The priority relation over rules, `prior`, is modeled as an OWL `ObjectProperty`. Notice that `prior` is not declared to be a transitive property in OWL. This is because the priority or rules is defeasible, to be defined in the models of substantive legal domains using defeasible rules. Although rule priority may be transitive, in fact, in some legal domains, this need not be the case in all legal domains.

The excluded relation is also modeled as an OWL `ObjectProperty`, over `Rule` and `Atom` instances.

Recall the example rule, above, for UCC § 9-105-h about movable things being goods:

```
(rule §-9-105-h
  (if (and (Movable ?c)
            (unless (Money ?c)))
    (Goods ?c)))
```

The XML source code for this rule, using the LKIF Rules ontology, is as follows:

```
<Rule rdf:ID="§-9-105-h">
  <swrl:body>
    <swrl:AtomList>
      <rdf:first>
        <swrl:ClassAtom>
          <swrl:classPredicate rdf:resource="#Movable"/>
        </swrl:ClassAtom>
      </rdf:first>
      <rdf:rest>
```

Section 5.1 OWL Abstract Syntax of LKIF Rules

Clearly, no one is going to be writing LKIF rules directly in XML, using this verbose syntax. Not even an XML structure editor would make this task much easier. An LKIF Rules plugin for a graphical ontology editor, comparable to the SWRL plug-in for Protege, would be ideal. (In fact, the above rule was first entered using the SWRL plug-in for Protege and then edited within Protege to make it an LKIF rule and to make the antecedent about money in the body an LKIF exception.)

Still, even with a nice graphical editor for LKIF Rules, it will be useful to have one or more simple, human writable and readable concrete syntaxes for rules, such as the s-expression syntax we have been using in this report. In section 5.1.2, the concrete syntax for this s-expression representation of LKIF rules will be formally defined.

5.1.1 LKIF Pretty-printing Syntax

Logical Vocabulary

The logical vocabulary of LKIF in pretty-printing syntax is as follows:
- $\forall, \exists, \cap, \cup, \neg$ (as concepts constructors);
- $\equiv, \subseteq$ (for OWL formulas);
- $\equiv, \forall, \exists, \ass, \unless$ (as connectives for legal reasoning);
- $(?\text{variable}_1, ?\text{variable}_2, \ldots ?\text{variable}_k)$ (as a universal quantifier for a $k$ number of variables; to be used in rules).

**Terms**

Terms are divided into individual variables, individual constants, classes and properties. Variables are lower-case letters preceded by $\?$, individual constants are lower-case letters preceded by $\!$, concepts are printed and properties are in lower-case. That is:

$\?\text{variable}$

$\!\text{constant}$

$\text{CONCEPT}$

$\text{PROPERTY}$

Concepts can be atomic and complex. Complex concepts are constructed by using concept constructors: $\neg, \cap, \cup, \forall, \exists$. The precise grammar for constructing concepts is given in 3.3. The following are some significant examples:

$\neg\text{CONCEPT}$

$\text{CONCEPT}_1 \cap \text{CONCEPT}_2$

$\text{CONCEPT}_1 \cup \text{CONCEPT}_2$

$\forall\text{PROPERTY}.\text{CONCEPT}$

$\exists\text{PROPERTY}.\text{CONCEPT}$

**Formulas**

LKIF is capable of expressing standard OWL TBox axims:

$\text{CONCEPT}_1 \subseteq \text{CONCEPT}_2$

$\text{CONCEPT}_1 \equiv \text{CONCEPT}_2$

Furthermore, LKIF can also express OWL ABox axioms, enriched with variables, propositional negation and the intensional operators $\ass, \unless$ (see 6 for a semantics). Here is a significant sample:

$\text{CONCEPT}(\!\text{individual})$

$\text{PROPERTY}(\!\text{individual}_1, \!\text{individual}_2)$
The propositional negation \( \sim \) or the operators \texttt{ass} and \texttt{unless} can be placed before any of the above formulas (but not before TBox axioms), and this will yield a new LKIF formula.

### Rules

Wewill now introduce a pretty-printing syntax for LKIF rules. Antecedent (body) and consequent (head) of rules are connected by arrow symbols \( \Rightarrow \). The blank space marked by \( \ldots \) is a place-holder for rule names. (For in LKIF rules can be named and referred to as if they were individual objects). Variables in rules are universally quantified, and so we need to introduce a notation for the universal quantifier: \( (\text{?variable}_1, \text{?variable}_2, \ldots) \).

One express that any car coming from the left has to slow down (unless an exception applies):

\[
(\text{?variable}_1) \\
\text{Car}(\text{?variable}_1) \land \text{coming from left}(\text{?variable}_1) \land \text{unless}(\text{Apply}(!r2)) \\
\Rightarrow \\
\text{Obligation_to_stop}(\text{?variable}_1)
\]

The exception introduced by \texttt{unless} refers to another rule, named \( r2 \), which states that any car on a main road has no obligation to stop or slow down:

\[
(\text{?variable}_1, \text{?variable}_2) \\
\text{Car}(\text{?variable}_1) \land \text{on}(\text{?variable}_1, \text{?variable}_2) \land \text{Main_road}(\text{?variable}_2) \\
\Rightarrow \\
\text{Has_Precendence}(\text{?variable}_1)
\]

Finally, there are certain concepts and properties (which only apply to rule names) that have a fixed semantics. These are: \texttt{valid}, \texttt{rebut}, and \texttt{exclude}. Section 6.2.2 offers an explanation of their semantics.

### 5.1.2 LKIF Rules S-expression Syntax

This section presents a formal definition of the s-expression syntax for LKIF-Rules, in Extended Backus-Naur Form (EBNF)\(^2\). This syntax is inspired by the Common Logic Interchange Format (CLIF) for first-order predicate logic, which is part of the draft ISO Common Logic standard.\(^3\) While inspired by CLIF, no attempt is made

---

\(^2\)EBNF is specified in the ISO/IEC 14977 standard. 
\(^3\)http://philebus.tamu.edu/cl/
to make LKIF Rules conform to Common Logic standard. The syntax uses the Unicode character set. White space, delimiters, characters, symbols, quoted strings, boolean values and numbers are lexical classes, not formally defined in this version of the specification of LKIF Rules. For simplicity and to facilitate the development of a prototype inference engine for LKIF Rules, for the time being we will use the lexical structure of the Scheme dialect of Lisp, as defined in the R5RS standard, extended to support the Unicode character set.

Variable and Name

\[
\begin{align*}
\text{variable} &::= \text{symbol} \\
\text{name} &::= \text{symbol}
\end{align*}
\]

A variable is a symbol beginning with a question mark character. Here are some examples:

- `?x`
- `?agreement`
- `?person`

A name is a symbol used to denote a Uniform Resource Identifier (URI). URIs are the World Wide Web standard way to identify or name things, also used by OWL and SWRL. A name may not begin with a question mark character, since such symbols are reserved for variables. Names are case-sensitive.

Names may include a prefix denoting a namespace. Some mechanism for binding prefixes to namespaces is presumed in this report, rather than being defined here. The prefix of a name is the part of the name up to the first colon. Only a single colon is allowed in a name. The part of the name after the colon is the local identifier of the object named, within this namespace.

Here are some example names:

- `agreement`
- `Agreement1`
- `lkif:Permission`
- `lkif:Event`

Term

A term is either atomic or compound. An atomic term is either a variable, name, string, number, or boolean value. A compound term consists of a functor symbol and a list of terms.
term ::= variable | name | string | number | boolean |
   | '(' name term* ')'
   | """, term

Quoted terms, the last case in the definition above, are syntactic sugar, as in Lisp and Scheme, for the form (quote <term> ...). One use of this feature is to denote lists or sets of terms.

Here are some example terms:

Joe
contract1
"Falkensee, Germany"
12.345
678
#t
(+ 34.5 123)

Sentence

All sentences in LKIF Rules are either atomic or negations of atomic sentences; no other logical operators are allowed within sentences.

atom ::= name | '(' name term* ')
sentence ::= atom | '(' 'not' atom ')

The SWRL forms of atomic sentences are special cases of this syntax, where the predicate name is an OWL class or property name, sameAs, differentFrom or some other SWRL built-in, such as swrlb:NotEqual or srwlrb:multiply.

Notice that LKIF Rules allows names alone to be used as atomic sentences. In this case, the name denotes a proposition. This provides LKIF Rules with a convenient syntax for propositional logic, to better support vendors, such as KnowledgeTools, whose knowledge representation formats are based on propositional logic.

The following are examples of sentences:

(initiates event1 (possesses ?p ?o))
(holds (perfected ?s ?c) ?p)
(children Ines '(Dustin Caroline))
(not (children Tom '(Sally Susan)))
(applies UCC-§-306-1 (proceeds ?s ?p))

Rule

LKIF rules generalize the syntax of Horn clause logic in the following ways:

1. Rules are named.

2. Multiple sentences are allowed in the head (conclusion) of rules.
3. Sentences in both the body and head may be negated. (Negation does not, however, have the Prolog semantics of negation-as-failure. See Section 6.2.2.)

4. Exceptions and assumptions are supported in the body of rules.

Here is the definition of the syntax of LKIF rules:

```plaintext
rule ::= '(': 'rule' name '(': 'if' body head ')') ':
       | '(' 'fact' name conclusion+ ')

head ::= conclusion | '(' 'and' conclusion conclusion+ ')

body ::= condition | '(' 'and' condition condition+ ')

conclusion ::= sentence

condition ::= sentence
            | '(' 'unless' sentence ')
            | '(' 'assuming' sentence ')
```

The `fact` form is syntactic sugar for a rule with an empty body.

Here are a few examples of rules and facts, reconstructed from the Article Nine World of the Pleadings Game [Gordon, 1995]:

```plaintext
(rule §-9-306-1
   (if (and (goods ?s ?c)
            (consideration ?s ?p)
            (collateral ?si ?c)
            (collateral ?si ?p)
            (holds (perfected ?si ?c) ?e)
            (unless (applies §-9-306-3-2 (perfected ?si ?p))))
    (holds (perfected ?si ?c) ?e)))

(rule §-9-306-2a
   (if (and (goods ?t ?c)
            (collateral ?s ?c))
    (not (terminates ?t (security-interest ?s))))

(fact F1 (not (terminates T1 (security-interest S1))))
(fact F2 (collateral S1 C1))
```

**Reserved Symbols**

LKIF Rules reserves the following predicate symbols, which have special meaning in the semantics, as explained in Section 6.2.2: prior, excluded, valid, applies and rebuts. These reserved symbols may be used in LKIF applications by defining them outside of the LKIF namespace.
5.2 Visualizations of LKIF

The specification of RDF proposes a standard graphical visualisation. RDF triples can be, and often are, visualized as directed graphs, in which two nodes (corresponding to subject and predicate) are connected by a directed arrow (corresponding to the predicate).

So far only a method to visualize Description Logic axioms has been decided. Another very interesting subject is the visualisation of argument structures, as these are sometimes used in end user software. The visualisation preferred is one taken from CARNEADES (cf. [Gordon and Walton, 2006a]), which is introduced in appendix D. It cannot be considered part of LKIF for now, as no definitive decision has been taken on the ontology of argument structures. The argument structure plays a central role in the explanation of the semantics of LKIF Rules in section 6.2.

5.2.1 Description Logic Graphs

The method of displaying description logic in graphical form that is adopted by LKIF is based on UML, following the mapping from OWL DL to the UML MOF defined in [Brockmans et al., 2006]. Figure 5.1 shows an example from the LKIF ontology, specifically a part of the module of the ontology discussed in section 6.1.

Figure 5.1: How DL is displayed in graphical form.
Most of the concepts defined in LKIF are only defined, in OWL DLP or OWL DL, in the ontology. Few of them are associated to LKIF specific semantics – i.e. are themselves part of the expressive power of the language, and warrant extra explanation. In most cases these additional semantics do not conflict with the interpretation of the underlying language (being OWL or SWRL). Only the LKIF Rules extension goes beyond the normal bounds of SWRL. The next two sections give an overview of normative statements, and of LKIF Rules, and the semantics of concepts specific to LKIF.
6.1 Normative Statements

6.1.1 Overview

A Legal Knowledge Interchange Format must deal with the issue of deontic reasoning, as exemplified by the field of deontic logic (cf. generally [Dayton, 1981, Makinson, 1999]). At the same time, it must be recognized that deontic logic is generally speaking not used by the industry. In an industrial setting, only certain aspects of the deontic reasoning are taken into account: for the purposes of some expert system it may for instance be relevant that a norm directs one to make a certain choice, but not that a violation can take place, or the other way around. In the field of deontic logic there is little convergence to a standard system of reasoning. There is such a system as Standard Deontic Logic, but it functions as a straw man, to be struck down by superior proposals, and not as a serious candidate in its own right. In addition to the disagreements about deontic reasoning, there is also a sizable community in AI & which rejects the very notion of mechanical jurisprudence, as noted in chapter 2.

In the Legal Knowledge Interchange format we take a cautious approach. LKIF specifies a standard structural representation of normative statements, without committing to a restrictive semantics. LKIF also doesn’t allow to derive violation of norms automatically. Instead normative statements are interpreted as giving constraints on what can be considered a violation of the statement. The act of ascribing a normative qualification to something is essentially abductive.

The key function of the norm is to qualify – as either allowed or disallowed. This is expressed in the ontology by noting that the norm is a Qualification, and the thing it qualifies has the quality Qualified. At this point we again stress that the decision that a norm qualifies something is an abductive one. Axioms relating to these qualities are listed in the table below:

\[
\begin{align*}
\text{Norm} & \equiv \exists \text{qualifies}.\text{Normatively}\_\text{Qualified} \\
\text{Norm} & \subseteq \text{Declaration} \\
\text{Norm} & \subseteq \text{Qualification} \\
\text{Permission} & \subseteq \text{Norm} \\
\text{Permission} & \equiv \exists \text{allows}.\text{Allowed} \land \neg \exists \text{allows}.\text{Allowed} \\
\text{Prohibition} & \subseteq \text{Permission} \\
\text{Prohibition} & \equiv \exists \text{allows}.\text{Obliged} \land \neg \exists \text{allows}.\text{Obliged} \land \exists \text{disallows}.\text{Disallowed} \\
\text{Obligation} & \equiv \text{Prohibition} \\
\text{Normatively}\_\text{Qualified} & \equiv \exists \text{qualified by}.\text{Norm} \\
\text{Allowed} & \subseteq \text{Normatively}\_\text{Qualified} \\
\text{Allowed} & \equiv \exists \text{allowed by}.\text{Permission} \\
\text{Obliged} & \subseteq \text{Allowed} \\
\text{Disallowed} & \subseteq \text{Normatively}\_\text{Qualified} \\
\text{Disallowed} & \equiv \exists \text{disallowed by}.\text{Prohibition} \\
\text{Strictly Allowed} & \subseteq \text{Allowed} \\
\text{Strictly Disallowed} & \subseteq \text{Disallowed} \\
\text{Allowed And Disallowed} & \equiv \text{Disallowed} \land \text{Allowed} \\
\text{disallows} & \subseteq \text{qualifies}
\end{align*}
\]
allows □ qualifies
allowed_by ⇔ allows⁻¹
disallowed_by ⇔ disallows⁻¹

The quality of being allowed or disallowed can not only be applied to real objects, but also to hypothetical one. It is for instance possible to state that the object of one’s intentions or expectations is disallowed, without this leading to a liability to be sanctioned: the qualification disallowed in itself does not entail the qualification violation. The thing that is disallowed is the object of an observation of, or a belief in, a violation, as expressed in the table below. The connection between certain actions and the observation of violation may be sufficient for some modeling purposes.

Observation_of_Violation ≡ Observation ⊓∃ expresses.Disallowed
Observation_of_Violation ⊑ Problem
Belief_IN_Violation ≡ Belief □∃ expresses.Disallowed
Belief_IN_Violation ⊑ Belief

Central to deontic reasoning is the notion of deontic choice (cf. [Hansson, 2001]), which states that if \( O(α | β) \) and an agent has the choice between bringing about \( (α ∧ β) \) and \( (¬α ∧ β) \) then the agent should choose \( (α ∧ β) \). In terms of a menu of alternatives this means that if \( | α ∧ β | \) and \( | ¬α ∧ β | \) are subsets of the menu \( A_1 \) and \( A_1 \in | α ∧ β | \) and \( A_2 \in | ¬α ∧ β | \) then \( A_1 > A_2 \). With \( > \) is meant normatively_strictly_better, which orders alternatives that are ALLOWED and DISALLOWED, respectively. Similarly the properties normatively_equivalent_or_better, normatively_not_equivalent, and strictly_equivalent are defined, all as subtypes of normatively_comparable, which applies to things that are NORMATIVELY_QUALIFIED. The table below lists most axioms.

NORMATIVELY_QUALIFIED ≡ ∃normatively_comparable.NORMATIVELY_QUALIFIED
ALLOWED ≡ ∃normatively_equivalent_or_worse.NORMATIVELY_QUALIFIED
OBLIGED ≡ ∃normatively_strictly_worse.DISALLOWED
DISALLOWED ≡ ∃normatively_strictly_better.ALLOWED
normatively_strictly_better ≡ normatively_strictly_worse⁻¹
normatively_equivalent_or_worse ≡ normatively_equivalent_or_better⁻¹
normatively_strictly_worse ≡ normatively_strictly_better⁻¹
normatively_equivalent_or_better ≡ normatively_equivalent_or_worse⁻¹
strictly_equivalent ≡ strictly_equivalent⁻¹
normatively_not_equivalent ≡ normatively_not_equivalent⁻¹
normatively_strictly_better ≡ normatively_equivalent_or_better
normatively_strictly_better ≡ normatively_not_equivalent
normatively_equivalent_or_worse ≡ normatively_comparable
normatively_strictly_worse ≡ normatively_equivalent_or_worse
normatively_strictly_worse ≡ normatively_not_equivalent
normatively_equivalent_or_better ≡ normatively_comparable
strictly_equivalent ≡ normatively_equivalent_or_better
strictly_equivalent ≡ normatively_equivalent_or_worse
normatively_not_equivalent ≡ normatively_comparable

T □ ∀normatively_strictly_better⁻¹.DISALLOWED
T □ ∀normatively_comparable⁻¹.NORMATIVELY_QUALIFIED
T □ ∀normatively_equivalent_or_worse⁻¹.ALLOWED
T □ ∀normatively_strictly_worse⁻¹.OBLIGED
T □ ∀strictly_equivalent⁻¹.ALLOWED
normatively_not_equivalent ≡ normatively_not_equivalent⁻¹
T ⊑ ∀normatively_strictly_better.OBLIGED
T ⊑ ∀normatively_comparable.NORMATIVELY_QUALIFIED
T ⊑ ∀normatively_strictly_worse.DISALLOWED
T ⊑ ∀normatively_equivalent_or_better.ALLOWED
T ⊑ ∀strictly_equivalent.ALLOWED

And the properties are of course transitive:

\[
\begin{align*}
&\text{TRANS}(\text{normatively_strictly_better}) \\
&\text{TRANS}(\text{normatively_strictly_worse}) \\
&\text{TRANS}(\text{normatively_equivalent_or_better}) \\
&\text{TRANS}(\text{strictly_equivalent})
\end{align*}
\]

Figure 6.1: An entity-relationship diagram describing the salient structure of obligations and prohibitions.

Putting them together results in the structure described in figure 6.1. A norm applies to (or qualifies) a certain case, allows a certain case - the obliged case or allowed case - and disallows a certain case - the prohibited or disallowed case. The obliged and prohibited case are both subsumed by the case to which the norm applies. Besides that they by definition form a complete partition of the case to which the norm applies, i.e. all cases to which the norm applies are either a mandated case or a prohibited case. This is true of the obligation and the prohibition: they are simply two different ways to put the same thing into words. Note that we can derive the nature of the comparative relation between mandated and prohibited case from the norm, or postulate the existence of the norm from the comparative relation.

The permission, shown in figure 6.2, is different. The permission allows something, but it doesn’t prohibit anything. The logical complement of the mandated case is here an opposite qualified case, about which we know only that it cannot be obliged.

Some things cannot be expressed in OWL DL. Stating for instance that \( \text{normatively_strictly_better} \cap \text{normatively_equivalent_or_worse} = \emptyset \) is not possible in the current OWL DL standard, but
Figure 6.2: An entity-relationship diagram describing the salient structure of a permission. Obligation and prohibition are subsumed by permission.

will be in the currently proposed OWL 1.1 update. As will become clear in section 6.1.3 LKIF depends on these properties not being disjoint.

6.1.2 Betterness

This section proposes a representation for normative systems and normative statements, and gives a semantic account of normative statements in OWL in terms of possible worlds. The possible worlds represent the concrete alternatives in a menu to consider for some agent who has to make a choice. LKIF is agnostic as to whether these ‘worlds’ represent situations, actions, plans, or design options. The statements of the legislator are interpreted as meaning that there is a world better than or at least as good as one in which the opposite is true. The statements of the legislator in other words generate and order categories of alternatives to be considered.

The perspective of the legislator cannot be directly transferred to the addressee of legislation, because the agent confronted with a menu of alternatives is not necessarily the party who brings about the violation of the normative statement, even if he foresees that it will be violated in some alternative futures. Pragmatically, we can use the rule of thumb that the agent choosing between alternatives is usually responsible for the situations directly resulting from his actions. In a planning environment characterized by the absence of other agents this will usually work, and many authors (cf. generally [d'Altan et al., 1996]) indeed seem to assume that there must be some standard method – however complex it may be – for relating productive and logically equivalent behavioural characterizations of the undesirable.

This rule of thumb is of course based on the idea that the situation resulting from your action is solely the product of your action and that it is static and completely predictable, which is of course often not the case in real life. A collision between two cars is for instance undesirable, but not necessarily the ‘product’ of the actions of both drivers, or even just one of them.

In many situations the representation of normative statements and the alternatives to be considered is sufficient to solve the decision making problems of the addressee as far as
the Law is concerned. In some cases we need more complex reasoning involving goals and intention, foreseeability, and causality.

To enforce the Law it is generally sufficient to know that certain norms have been violated. A system that supports this task collects information to establish what the settled facts are, and decide on the legality of the settled facts. To comply with the Law, on the other hand, we have to deal with the fact that decisions take place in a context of settled facts. This is obviously the case in relation to the timeline: history is the context, and the future holds the alternatives. In other cases an agent may consciously choose to plan for an illegal goal. Consider for instance an agent who has the goal to steal a huge amount of money, and is considering alternative ways to achieve this goal without physically harming or endangering people. He may treat the theft as a settled fact in his decision problem, and the goal of the system is to order the alternatives and to select the best one. The legality of the context is not at stake, and the evaluation of the context should not interfere with the ordering of alternatives.

Note that this account of deontic reasoning makes no commitments on things like actions, situations, states, etc. It simply divides reality into the part that is to be considered settled and the part that is open to manipulation by agents. Decision problems can be classified into the following four categories:

**Assessment:** Does a constellation of facts $F$ violate norms in normative system $N$?

**Envisioning:** What categories of cases does normative system $N$ distinguish, and how does it order them?

**Ordering the alternatives:** Given a context description $C$, and a menu $A$, and any $A_1 \in A$, $A_2 \in A$, order $A_1$ and $A_2$ in accordance with normative system $N$.

**Choosing the best alternative:** Given a context description $C$, and a menu $A$, choose all $A_1 \in A$ for which there exists no $A_2 \in A$ such that $A_1 \succ A_2$ in normative system $N$.

The two differentiating features are whether one is using the normative system with or without a context, and whether one is ordering or minimizing alternatives. On the logical level we also have to distinguish ordering categories or concrete alternatives. In a normative theory that does not contain conflicts this distinction turns out to be of less importance than it would be in a true logic of preference, because we are sure – given the definitions of deontic operators in terms of preference – that the preferences are all properly partitioned (if $\alpha \succ \neg \alpha$ then by definition $\models \alpha \cap \neg \alpha = \emptyset$). As long as we are able to determine in which order to consider conflicting normative statements, these do not pose a complex problem.

On the other hand the property of preference independence does not hold. Given concepts $P$ and $Q$, it is not generally the case that $Q$ is preferentially independent from $P$. Let $P$ mean *running over a pedestrian in your car* and $Q$ mean *calling 112 and telling them you ran over a pedestrian*: observe that it is entirely possible and reasonable that $P$ is disallowed, and $Q$ is allowed if $P$ and disallowed if $\neg P$. This is essentially what happens in so-called contrary-to-duty situations like the Chisholm paradox discussed later. This precludes the use of additive value functions, multiplicative utility functions, and other economic concepts based on the assumption of preference independence (cf. [Keeney and Raiffa, 1976, Russell and Norvig, 2003]). The use of decision support methods like Multiattribute Utility Theory (MAUT; cf. [Dyer, 2005]) will not help policy makers to make more rational decisions if this assumption is violated.

The statements of the legislator must be consistent in one normative system: conflicts of compliance and disaffirmation are truth-functional contradictions if they are in the same system. To solve this problem, a legal system needs a method for partitioning normative
statements into different subsystems, and then it needs to order the subsystems, and use
the ordering to determine the verdict the composite normative system yields on a decision
problem.

If the alternative chosen by the agent violates normative system \( N_i \) but uses \( N_j \), and
\( N_j \) is higher than \( N_i \), then the alternative is allowed.

To determine this verdict for case \( C \) given two properly partitioned normative systems
\( N_i \) and \( N_j \) and \( N_i \succ N_j \), we have to be able to establish that, although case \( C \) violates \( N_j \),
it uses \( N_i \). The agent uses the norms by choosing an alternative such that an alternative is
accessible that is not (strictly) better. The agent complies with the legislator by choosing
an alternative in which no alternative is accessible that is strictly better than the one being
violated. Violation of the norms, by way of a choice for an alternative even though there
is a better alternative, is obviously not a truth-functional contradiction. The notion of using
a normative statement has to be introduced to make a distinction between situations the
norms are silent on, and situations explicitly allowed by the norms. What we want to know
is not that \( C \) is not disallowed in \( N_i \), but that \( N_i \) explicitly allows it.

For this reason the properties \textbf{normatively} \textbf{strictly worse} and
\textbf{normatively} \textbf{equivalent or better} cannot be considered disjoint, and whether a substantial
violation takes place is not an issue that can be answered in OWL DL alone.

### 6.1.3 Semantics

The mapping of deontic statements into OWL DL will be explained in terms of frame
semantics. OWL DL can be understood as a labeled modal logic with some extras (cf.
[Horrocks et al., 2003, Haarslev and Møller, 2001]) and an odd abstract syntax. A separate
property can be described by frame axioms. This mapping is non-standard from a deontic
logics point of view, and has some interesting characteristics.

As a point of departure we use dyadic obligations, prohibitions, and permissions of the
form \( O_i(\alpha \mid \beta) \), intuitively meaning that \( \alpha \) is obliged if \( \beta \) is true according to theory \( i \). These
are \textit{prima facie} obligations, in the sense that violation of theory \( i \) is not defined in terms of
\( \neg(\alpha \cap O(\alpha \mid \top)) \). Whether something is a violation depends on context, and is decided by
a specific reasoning protocol.

We work with modal frames of the form \( M = \langle W,N,\{\preceq_i\}_{i \in N},V \rangle \) where \( W \) is a set of
worlds, \( N \) a set of preference theories, the \( \preceq_i \) are transitive relations, and \( V \) is a propositional
valuation. Intuitively, this structure should be understood as the ABox (assertion box) of
the description classifier. Each \( i \in N \) is a (consistent) subtheory in the preference system,
and can be intuitively understood as a kind of agent that imposes its own preferences on
worlds \( w \in W \). In a description logic context the worlds should be understood as constants
declared in the ABox.

Read \( x \preceq_i y \) as “preference theory \( i \) considers \( y \) as
\textbf{normatively} \textbf{equivalent or better} \( x \)”. Read \( x \prec_i y \) as “preference theory \( i \) considers \( y \) \textbf{normatively} \textbf{strictly better}
than \( x \)”. This is shorthand for \( (x \preceq_i y) \cap \neg(y \preceq_i x) \). Read \( x \asymp_i y \) as “preference theory \( i \) is
considers \( y \) and \textbf{strictly equivalent} \( x \)”. This is shorthand for \( (x \preceq_i y) \cap (y \preceq_i x) \). I omit the
labels \( i \) in some cases, but it is essential to define a subclass of the LKIF relations for any
large scale modeling project, to prevent undesirable interactions with other models.

The language \( L \) is a set of declarations, implicitly joined by a \( \cap \), modeled on the de-
scriptions of concepts in a description logic TBox (terminological box) of the form \( \phi_1 \subseteq \phi_2 \)
or \( \phi_1 \equiv \phi_2 \) where \( \phi_1 \) and \( \phi_2 \) are concept definitions conforming to the syntax:

\[
\phi := p \mid \neg \phi \mid \phi \cup \phi \mid \phi \cap \phi \mid \exists \preceq_i \phi \mid \forall \preceq_i \phi
\]

As usual \( \forall \) is defined in terms of \( \exists \) as follows: \( \forall \preceq_i \phi \equiv \neg \exists \preceq_i \neg \phi \). The letter \( p \) represents
an atomic proposition, or \textit{concept name} in description logic parlance. The truth definition
for this language, and semantic notions like frame, satisfiability, and validity, are entirely standard (see Blackburn et al., 2001) for a normative reference:

\[ M, w \models \exists i \preceq_i \phi \text{ if } \exists w' : w \preceq_i w' \text{ and } M, w' \models \phi \]

This says that \( \phi \) is true in at least one alternative of \( w \) which \( i \) considers at least as good as \( w \). The preference modality \( \exists \preceq \) constrains a preference order at the level of worlds, for each separate preference theory. The language has a complete axiomatization over preference models: the standard modal logic \( K4 \) for each separate preference theory.

Note that the axiom \( T : \square A \subseteq A \), which would turn \( \preceq \) into a reflexive relation, is absent. Addition of this axiom would turn this logic into the S4 system used by i.a. Van Benthem et al. as a preference logic (cf. [van Benthem et al., 2005]) and from there to more complex deontic logics like for instance Boutilier’s CT4O (cf. [Boutilier, 1994]) or Van der Torre’s 2DL (cf. [der Torre, 1997]).

A normative statement translates to a preference expression in \( L \). On the propositional level an obligation \( O_i(\alpha | \beta) \), or a prohibition \( P_i(\neg \alpha | \beta) \), translates to:

\[(\neg \beta \preceq \alpha) \equiv (\exists i) (\beta \preceq \neg \alpha) \]

This says that in a world where \( (\beta \preceq \alpha) \) is true, there is no world equal or better accessible where \( (\beta \preceq -\alpha) \) is true, and in a world where \( (\beta \preceq -\alpha) \) is true, there is some world equal or better accessible where \( (\beta \preceq \alpha) \) is true, according to \( i \). Note that obligation is not defined in terms of the \( \square \) operator. It is possible to include other worlds where \( (\beta \preceq -\alpha) \) is true in the ordering as long as one does not claim they are better than a world in which \( (\beta \preceq \alpha) \) is true, and obviously a world where \( \neg \beta \) is true can be placed anywhere in the ordering.

**Observation 1.** In the description logic OWL DL the obligation \( O_i(\alpha | \beta) \) translates to a slightly different syntactical structure, involving several auxiliary terms representing complex sentences: \( \beta \equiv (C_1), (\beta \preceq \alpha) \equiv (C_2), (\beta \preceq -\alpha) \equiv (C_3), (C_2) \subseteq (C_1), (C_3) \subseteq (C_1) \), and a complement of statement \( \neg(C_2) \equiv (C_3) \), in addition to the essential \( (C_2) \subseteq (C_3) \) or \( (C_3) \subseteq (C_2) \). In other words, \( \alpha \) as differentia, or distinguishing characteristic, between \( (C_2) \) and \( (C_3) \) is left implicit.

The permission \( P_i(\alpha | \beta) \) translates to:

\[(\beta \preceq -\alpha) \equiv (C_2) \equiv (C_3) \equiv (C_2) \equiv (C_3) \subseteq (C_2) \]

This says that in a world where \( (\beta \preceq -\alpha) \) is true, there is some world equal or better accessible where \( (\beta \preceq \alpha) \) is true. Intuitively this may suggest that it is advisable to choose \( (\beta \preceq \alpha) \) over \( (\beta \preceq -\alpha) \), but that is not intended. It is obviously consistent to add an edge from a world in which \( (\beta \preceq \alpha) \) is true to a world where \( (\beta \preceq -\alpha) \) is true, establishing that both alternatives are of equal value.

**Observation 2.** In the description logic the permission \( P_i(\alpha | \beta) \) translates to: \( \beta \equiv (C_1), (\beta \preceq \alpha) \equiv (C_2), (\beta \preceq -\alpha) \equiv (C_3), (C_2) \subseteq (C_1), (C_3) \subseteq (C_1) \).

In propositional modal logic these statements are not very easy to read. They are certainly not an improvement over \( O(\alpha | \beta) \) or \( P(\alpha | \beta) \), but in a graphical representation for OWL they work quite well.

Note that the characterization of permission is just strong enough to alert us of any conflicts between obligations and permissions. It is not necessarily the case that the presence
of $x \preceq y$ indicates a strong permission for that ordering. Trying to strengthen the definition of permission within the syntactic limits of OWL DL (for instance $(\beta \cap \neg \alpha) \subseteq (\exists \preceq_i (\beta \cap \alpha) \cap \forall \preceq (\alpha \cup \neg \beta))$ is not going to solve the problem that we are dealing with an underspecified frame that includes unintended orderings.

The following is the only thing that cannot be expressed in OWL DL itself and must be checked by a preprocessor:

**Definition 18** (Disjointness of allowed and disallowed case). If $(C_1) \subseteq \exists \preceq_i (C_2)$ or $(C_1) \subseteq \neg \exists \preceq_i (C_2)$ then $C_1 \cap C_2 = \emptyset$.

Note that the relation $\preceq$ is not complete. Some worlds are genuinely incomparable. This means that the $\preceq$ relation does not meet the property of trichotomy, requiring that it should be possible to choose between any two alternatives, as is often required of preference relations. Hansson defends the invalidity of the trichotomy property in [Hansson, 2001], but this argument is not universally accepted.

**Observation 3.** Some worlds are incomparable.

**Proof.** Assume the obligations $O(\alpha \mid \top) \cap O(\beta \mid \top)$. $\models \phi$ is the set of worlds $w$ such that $M, w \models \phi$. Assume $a \in \neg \alpha \cap \beta$ and a $y \in \alpha \cap \neg \beta$. Observe that both $x \preceq y$ and $y \preceq x$ are contradictions.

To be viable as a deontic knowledge representation system, the system has to meet some requirements. Minimally, the system should capture two properties: that what is obliged should be permitted and the impossible should not be obligatory.

The intuition for the first property is that you cannot be obliged to do something without at the same time being permitted to do that something. The property is usually expressed by way of the following axiom: $OA \subseteq PA$. This characterization does meet the axiom $OA \subseteq PA$, but it is not defined in terms of the frame axiom $\forall \preceq A \subseteq \exists \preceq A$:

**Observation 4.** What is obligatory is permitted. The axiom $O(\alpha \mid \beta) \subseteq P(\alpha \mid \beta)$ is true.

**Proof.** Trivial from $((\beta \cap \alpha) \subseteq \neg \exists \preceq_i (\beta \cap \neg \alpha)) \cap ((\beta \cap \neg \alpha) \subseteq \exists \preceq_i (\beta \cap \alpha))$ and $(\beta \cap \neg \alpha) \subseteq \exists \preceq_i (\beta \cap \alpha)$.

Note that it is perfectly possible to weaken the obligation so that it does not meet this property. The obligation translates to two statements in this system, connected here by a $\cap$. Omitting the preference statement $((\beta \cap \neg \alpha) \subseteq \exists \preceq_i (\beta \cap \alpha))$ will do the trick.

A nice feature of this system is that it does not only make the impossible obligation $O(\neg \alpha \mid \alpha)$ inconsistent, but also the unintuitive and meaningless obligation $O(\alpha \mid \alpha)$ mentioned by [Makinson, 1999].

**Observation 5.** The impossible and the meaningless are not obligatory: $\neg O(\alpha \mid \alpha)$ and $\neg O(\neg \alpha \mid \alpha)$ are true.

**Proof.** Observe that $\bot$ follows from $((\alpha \cap \alpha) \subseteq \neg \exists \preceq_i (\alpha \cap \neg \alpha)) \cap ((\alpha \cap \neg \alpha) \subseteq \exists \preceq_i (\alpha \cap \alpha))$ and from $((\neg \alpha \cap \alpha) \subseteq \neg \exists \preceq_i (\neg \alpha \cap \neg \alpha)) \cap ((\neg \alpha \cap \neg \alpha) \subseteq \exists \preceq_i (\neg \alpha \cap \alpha))$.

As mentioned before, the conflicts of disaffirmation and compliance do raise a contradiction, as they arguably should. It is important to be able to calculate all prima facie obligations, even if we apply a method to resolve them automatically. The axiom that captures this notion for conflicts of compliance is generally called the no-dilemma assumption.

**Observation 6.** There are no conflicting obligations. The obligations $O(\alpha \mid \beta)$ and $O(\neg \alpha \mid \beta)$ are inconsistent: $\neg(O(\alpha \mid \beta) \cap O(\neg \alpha \mid \beta)))$ is true.
Proof. If $M, w \models \beta \land \alpha$ then $M, w \models \exists i (\beta \land \neg \alpha) \land \neg \exists i (\beta \land \alpha)$ if both statements are true.

The conflict between a permission and an obligation follows a very similar pattern. As stated earlier the obligation consists of two separate statements. One captures the permission for a choice that it entails ($x \sqsubseteq y$), and the other, negative, statement can be thought of as blocking a permission for a competing choice ($\neg (y \sqsubseteq x)$).

**Observation 7.** The obligation $O(\alpha \mid \beta)$ and permission $P(\neg \alpha \mid \beta)$ are inconsistent: $\neg (O(\alpha \mid \beta) \land P(\neg \alpha \mid \beta))$ is true.

Proof. If $M, w \models \beta \land \neg \alpha$ then $M, w \models \exists i (\beta \land \alpha) \land \neg \exists i (\beta \land \alpha)$ if both statements are true.

Finally, the contrary-to-duty situations are also a classical test of the viability of a deontic knowledge representation system. The contrary-to-duty situations are no problem for this system, as I will show with the Chisholm paradox.

**Observation 8.** The sentences $O(\alpha \mid T)$, $O(\beta \mid \alpha)$, $O(\neg \beta \mid \neg \alpha)$ are only satisfied by the ordering identified by [Chisholm, 1963].

Proof. If $w \models \phi$ is the set of worlds $w$ such that $M, w \models \phi$, then the desirable ordering is represented as $| \neg \alpha \land \beta \mid \prec \neg \alpha \land \neg \beta \mid \prec \alpha \land \neg \beta \mid \prec \alpha \land \beta |$. Verify for each set $| \phi |$ that:

1. that it cannot be empty if some other set $| \phi' |$ is nonempty and $| \phi' \mid \prec \phi |$;
2. that $x \sqsubseteq y$ is not possible for any $x \in | \phi |$ and $y \in | \phi' |$ and $| \phi' \mid \prec \phi |$.

Details are left to the reader.

The essence of the Chisholm paradox is of course not that to comply with the obligations you have to choose a world $w \in | \alpha \land \beta |$. The knowledge representation system should be able to establish that if only a fixed context $| \neg \alpha |$ is given, then the optimal choice is a $w \in | \neg \alpha \land \neg \beta |$.

### 6.1.4 Discussion

For decidability and practical algorithms we refer to Baader et al. in [Baader and Sattler, 2001]. Note that description logics of this kind been used in practice on very large knowledge bases (cf. for instance generally [Mcguinness et al., 2000]), although these may exhibit other structural properties than legal knowledge bases built with the method outlined in this document. It is known that certain mereological and temporal constructs do not scale very well, and that may also be the case for this preference relation.

The modal notion of bisimulation makes sense in this interpretation and can be used to establish that two preference structures are the same from a modal standpoint. This notion is central to the semantics of comparison of preference structures but we will not develop this thought here.

A disadvantage of the mapping is of course the lack of flexibility of a black box OWL reasoner: it is not possible to freely add axioms to the language. The result is a weak logic that has to be augmented with intelligent queries to get meaningful results. As such, it is probably not a very good starting point for more complicated logics.

Notably, even the reflexive frame (the axiom $T : \Box A \sqsubseteq A$) cannot be expressed in OWL, even though reflexivity is a common property of mereological, topological, spatial, and temporal relations. Some workarounds have been proposed in i.a. [Grau et al., 2005].
(submissions 26 and 33), and the OWL specifications propose some alternative methods for approximating reflexivity in the definition of parts and wholes.

One of the arguments made for the omission of reflexive frames, besides the apparent computational attractiveness of doing so, is that its primary function is as an abbreviation anyway: instead of stating that for some $w$ there is some $w'$ such that $wRw'$ that has some property we are looking for, we have to state that $w$ has the property we are looking for or there is some $w'$ such that $wRw'$ that has the property we are looking for.

In [Rescher, 1955] Rescher made the point that the property of reflexivity is actually ontologically suspect: nothing is a “part of itself” and well-formed wholes should consist of at least two parts. In a mereological, topological, spatial, or temporal context this observation is not of great importance. If identity is not regarded as a limit case of parthood, a stronger relation can obviously be defined in terms of the weaker one without invalidating important results.

Rescher’s point also holds for the $\preceq$ relation: it is odd to say that something is as good as itself, or that we are indifferent between something and itself. Since we are mostly concerned with the irreflexive counterpart of $\preceq$ we simply adopt the limitations of OWL.
6.2 LKIF Rules

6.2.1 Overview

This section specifies LKIF Rules. LKIF rules are defeasible rules that amongst other suggest arguments to be made, and policies to be followed. It is used to express "rules" that are non-terminological and (therefore) defeasible in character. The LKIF rules extension is both a expressive extension of the LKIF logical language, as explained in 4, and a specific epistemological terminology, defining concepts such as EXCEPTION, ASSUMPTION, etc.

The SWRL specification includes the following usage suggestion:

A useful restriction in the form of the rules is to limit antecedent and consequent classAtoms to be named classes, where the classes are defined purely in OWL (in the same document or in external OWL documents). Adhering to this format makes it easier to translate rules to or from existing (or future) rule systems, including: Prolog, production rules (descended from OPS5), event-condition-action rules; and SQL (where views, queries, and facts are all rules).

Adhering to this form also maximises reuse and interoperability of the ontology knowledge in the rules with other OWL-speaking systems that do not necessarily support SWRL.

Following this suggestion, LKIF Rules syntax limits predicates in atomic sentences to the names of classes defined in OWL. OWL descriptions of classes may not be used directly in rules.

A term is a variable or a constant. Constants are symbols, used as names of instances of classes, or values of some datatype (string, boolean, number, etc.). Since LKIF is based on OWL and SWRL, symbols are Universal Resource Identifiers (URIs) and the set of datatypes available are those provided by OWL. Variables will be represented by SWRL variables.

For simplicity and readability, we will be using in this report a concrete syntax, explained in 5.1.2, based on s-expressions to represent rules. Variables will be represented as symbols beginning with a question mark, e.g. ?x or ?y. Other symbols, as well as numbers and strings, represent constants, e.g. contract, 23.1, or "Jane Doe".

An atomic sentence is a simple declarative sentence, containing no logical operators, such as negation, conjunction or disjunction. Since the LKIF rules sublanguage is based on SWRL, its sentence forms are also supported: (C x), (P x y), (sameAs x y), (differentFrom x y) and (builtin r x ..), where C is an OWL class name, P is an OWL property and r is a built-in OWL property.

The natural language sentence “Ines is the mother of Caroline.” would be represented in LKIF Rules as (mother Caroline Ines). (Notice the order of the terms here. OWL atoms are represented using RDF triples of the form (<predicate> <subject> <object>). In the example, mother is the predicate, Caroline, the subject, and Ines the object of the sentence.)

In LKIF, OWL is used to define the concept of a rule. Individual rules are instances of this rule concept. Thus, rules are verified objects in this model, with an identifier and a set of properties, enabling any kind of meta-data about rules to be represented, such as the rule's date of enactment, issuing governmental authority, legal source text, or its period of validity. We do not define these properties. Here our focus is on defining the structural components of rules. Our starting point again is SWRL. Rules in SWRL have a body and a head, each of which are a list of atomic sentences. The terms 'head' and 'body' are from logic programming, where they mean the conclusions and antecedents of the a rule, respectively. Notice that, unlike Horn clause logic, a rule may have more than one conclusion.

Here is an example SWRL rule, from the SWRL specification:
(rule r1
  (if (and (artist ?x)
            (artist-style ?x ?y)
            (style ?y)
            (creator ?y ?x))
    (style/period ?z ?y)))

Here, r1 is an OWL identifier naming the rule.

LKIF Rules extend this rule language to support negation, exceptions and assumptions, represented informally here as (not P), (unless P) and (assuming P), respectively, where P is an atomic sentence. We use the term condition to cover both atomic sentences and these forms.

Negated conditions may be used in both the body and head of rules. Exceptions and assumptions may be used only within the body of rules.

Here an example rule illustrating the use of an exception. This is a simplified reconstruction of a rule from the Article Nine World of the Pleadings Game [Gordon, 1995], meaning that all movable things except money are goods.

(rule §-9-105-h
  (if (and (movable ?c)
          (unless (money ?c)))
    (goods ?c)))

Rules are defeasible in argumentation. Exceptions are one form of defeasibility. A rule applies if its conditions are acceptable, unless some exception applies. The burden of proving the exception is on the party who wishes to show that the rule is not applicable. The party who wants to apply this rule need not first show that the exception does not apply.

Another source of defeasibility is conflicting rules. If one rule allows us to derive P and another (not P), which rule has priority? (Such conflicts cannot occur in SWRL, since it provides no support for negation.) LKIF extends SWRL with another feature to resolve these conflicts. Rules are defined to have an OWL property, prior, where (prior r1 r2) is intended to mean that rule r1 has priority over rule r2.

This priority relationship among rules is not defined in LKIF, but rather in applications of LKIF in particular legal domains. This can be done both extensionally, by asserting a bunch of facts about which rules have priority over which other rules, as in Nute’s Defeasible Logic [Nute, 1994] and intensionally, using meta-rules about priorities, as in PRATOR [Prakken and Sartor, 1996]. For example, assuming metadata about the enactment dates of rules has been modeled, the legal principle that later rules have priority of earlier rules, lex posterior, can be represented as:

(rule lex-posterior
  (if (and (enacted ?r1 ?d1)
           (enacted ?r2 ?d2)
           (later ?d2 ?d1))
    (prior ?r2 ?r1)))

Legal rules can be defeated in two other ways: by challenging their validity and by showing that some exclusionary conditions apply. These are modeled in LKIF with rules about validity and exclusion. For this purpose, two further predicates are pre-defined in LKIF (valid <rule>) and (excluded <rule> <atom>). The valid predicate is represented as an OWL class; the excluded predicate as an OWL property. Notice that the second argument of the excluded predicate is an atomic sentence. This is possible in SWRL, and
thus LKIF, because any OWL object may be referenced in the object or value position of an atomic sentence. This is a very powerful feature that LKIF uses extensively for representing meta-level knowledge.

The excluded property, like the prior property, is not defined in LKIF, but rather in the models of particular legal domains. Thus, the exception in the example above about money not being goods, even though money is movable, can also be represented using an exclusionary rule as follows:

\[(\text{rule } \S 9-105-h-i)\]
\[(\text{if (money ?c)})\]
\[(\text{excluded } \S 9-105-h \text{ (goods } ?c)))\]

To illustrate the use of the validity property of rules, imagine a rule which states that rules which have been repealed are no longer valid:

\[(\text{rule repeal})\]
\[(\text{if (repealed ?r1)})\]
\[(\text{not (valid ?r1))))\]

Notice also the use of negation in the conclusion (head) of this rule.

This completes our informal presentation of the LKIF rule language.

6.2.2 Semantics

Legal rules express public policy. These are not only policies about how to act, but also policies about how to reason when planning actions or determining the legal consequences of actions after they have been performed. For example, the definition of murder as the “unlawful killing of a human being with malice aforethought” expresses both the policy against the intentional killing of another human being and the reasoning policy to presume that a murder has been committed if it has been proven that a human being has been killed intentionally. Such presumptions are not sufficient for making legally correct decisions. Having proven that the defendant has killed someone intentionally is not sufficient for proving guilt or convicting him of committed murder. Rather, such a decision is legally correct only if it has been made at the end of the proper and fair legal procedure, i.e. via “due process”, in which the parties have been given a fair opportunity to produce evidence and make arguments and these have been properly taken into consideration in the justification of the decision.

Conversely, legal rules are not statements about what is or is not true in some domain or world. Perhaps legal rules can be considered to be declarative sentences, but what they declare are policies, not truths.

Due to these properties of legal rules, modeling them as material implications in classical propositional or predicate logic is inadequate. Material implications express constraints on some domain which are universally true, i.e. without exception. Legal rules, on the contrary, express defeasible generalizations. Applying a legal rule only gives rise to a presumption. Or, as Jaap Hage puts it in his work on Reason-Based Logic [Hage et al., 1993], applying a legal rule gives us a ‘reason’ to accept its conclusion, which must be ‘weighed’ against other reasons, including reasons from other rules.

Viewing legal reasoning as deduction using a classical logic also has problems from a procedural perspective. Let us suppose, for the sake of argument, that some legal domain, such as tax law, can be modeled as a set of formulas in first-order logic, \(\Lambda\). Let us suppose further that the facts of some particular case to be decided can also be modeled this way, in \(\Phi\). Can the union of these two set of formulas, \(\Lambda \cup \Phi\), be viewed as a theory from which
legally correct conclusions can be drawn, using only deduction? Suppose some formula \( \rho \) is logical consequence of this theory, \( \Lambda \cup \Phi \models \rho \). Is this fact alone sufficient to say with certainty that \( \rho \) is a legally correct conclusion? The legal philosophy which claims this is the case is known as mechanical jurisprudence.\(^1\) Since \( \rho \) is a logical consequence of \( \Lambda \cup \Phi \), it is necessarily true, given this theory. One attractive feature of mechanical jurisprudence is that it gives us a feeling of security, of having a method or procedure for making correct decisions. But this is a false sense of security, because each of the steps in this procedure is fraught with uncertainties which have been obscured by this presentation:

1. The model of the legal domain in first-order logic, in \( \Lambda \), may be faulty, i.e. invalid, in any number of ways. For example, the natural language texts of the legal sources needed to be interpreted during the modeling process. These interpretations may be incorrect. And the law may have changed in the meantime, so the model may be out of date. The model may be incomplete. Some exceptions to rules, for example, may not have been taken into consideration. This kind of incompleteness, incidentally, also calls into question the correctness of any conclusions drawn deductively applying the model.

2. Similar problems exist for the model of the facts. To what extent are the facts supported by evidence? If there was conflicting evidence, were these conflicts resolved in a legally correct way? There is always a gap between the legal terms used in legal rules and the nonlegal description of the facts. For example, suppose a boy rode his bike through a park and was cited for violating a rule prohibiting ‘vehicles’ in the park. (This is of course a variation of Hart’s famous example of the open-texture of legal concepts.) How was the gap between ‘bike’ and ‘vehicle’ resolved? This is known as the subsumption problem in German legal theory. The model of the legal domain can be extended with interpretation rules to deal with this problem. But one can always question whether or not such an interpretation rule is legally valid. Is it supported by case law? Even if it is, other cases my point in the other direction.

3. Finally, a further assumption of the mechanical jurisprudence model is that it was possible to determine that \( \rho \) is a logical consequence of \( \Lambda \cup \Phi \), ignoring the well-known fact that first-order predicate logic is undecidable. Even if it were possible to find a decidable subset of first-order logic which was sufficiently expressive for modeling legal problems, which is highly doubtful, we would still have to overcome computational complexity problems. There may not exist a tractable decision procedure for the logic.

These procedural issues are orthogonal to the problem previously discussed, about the defeasibility of legal rules. In AI, there have been numerous attempts to model reasoning with defeasible rules using nonmonotonic logics. While these logics may or may not adequately model reasoning with priority or defeat relationships between conflicting rules, or conflicting ‘arguments’ constructed using these rules, they do not address any of the procedural issues. They are purely relational models of reasoning which presume that the knowledge of the domain, including any defeasible rules and the facts of some case, have already been adequately modeled. Moreover, nonmonotonic logics often have even worse computational properties than classical logic, thus exacerbating the third procedural issue about effective procedures for applying such a theory to make decisions.

For all of these reasons, the semantics of legal rules adopted here is based on another approach, one articulated well by Ron Loui in his landmark article “Process and Policy: Resource-Bounded Non-Demonstrative Reasoning” \([\text{Loui, 1998}]\). Essentially, legal rules are interpreted as policies for reasoning in resource-limited, decision-making processes. From a

\(^1\)Mechanical Jurisprudence, however, is a kind of straw-man philosophy. It is unclear whether any respectable legal theorist has ever taken this position.
classical logic perspective, this approach synthesizes proof theory and model theory. (Critics might say that it inadmissibly mixes up or conflates these two kinds of theories.) The ‘knowledge’ expressed by a theory made up of defeasible rules is not knowledge about the domain, at least not directly so, but rather knowledge about how to reason in the domain. To use the standard example from the field of nonmonotonic logic, the rule “birds fly” does not state that all birds fly, but rather that an agent should presume that some bird flies, not knowing anything else. The rule constrains the agent’s reasoning about the domain, not the domain of discourse itself.

In Loui’s ‘process and policy’ approach, policies are expressed as defeasible rules in the knowledge base of the domain. The process part of this model is handled by a normative model of the procedure for using this knowledge. These procedural models can take various forms. One common approach is to model legal reasoning as a dialogue game, where the rules of the game are commonly called a protocol. This approach fits in well with the modern theory of argumentation, from the field of philosophy [Walton, 2005]. According to this theory, which protocol is appropriate can depend on the type of the dialogue. Different types of dialogues have different purposes, and protocols are designed to achieve these purposes, or at least facilitate their achievement. This view also coincides with legal practice, where each kind of legal proceeding is governed by its own procedural rules.

The interesting and important thing about these dialogue protocols, for our purposes, is that they are responsible for addressing the issues raised above, about how to distribute the burden of representing the knowledge of the domain (known as the ‘knowledge acquisition bottleneck’ in AI), the problem of how to establish the facts of the case, as well as the problem of how to deal with computational complexity issues. Rather than sweeping these problems under the rug, as in relational models, this approach requires these problems to be explicitly addressed when designing dialogue protocols. The practical result of this exercise, if done well, is an efficient, fair, effective and transparent decision-making procedure. Applying such a procedure results in a decision which is presumably correct.

In the case of legal dialogues, these protocols, i.e the procedural rules applicable to some particular type of legal proceeding, define what it means for a decision to be legally correct: a decision is legally correct if and only if it is the result of the appropriate legal procedure, correctly executed. This view of legal correctness has been formulated by John Rawls, in his “Theory of Justice” [Rawls, 1971], where he distinguishes various forms of procedural justice: pure, perfect and imperfect justice. In general, legal decision-making is of the imperfect kind, since there is no objective, independent standard for evaluating the correctness of most legal decisions.

In addition to the knowledge of the domain, including defeasible rules, and the appropriate protocol from some proceeding, there is another component to this semantic model, called argumentation schemes. Argumentation schemes are presumptive rules of inference. They can be generic or domain dependent. They are orthogonal to the types of dialogues and their protocols. Argumentation schemes may be applied in many kinds of dialogues. Which argumentation schemes may be used, however, may be constrained by the protocol. Not all argumentation schemes may be used at all times, in all types of dialogues. Walton and his colleagues have begun collecting and classifying common argumentation schemes. Their current catalog consists of about 60 such schemes.

Admittedly, there is some hand-waving going on here, because we have yet to design such a protocol, let alone prove that it is efficient or leads to fair, effective and transparent decisions. We will not do so in this report, but for present purposes will only assume this is possible. This is no more egregious than the assumptions made by the relational model, as described above, about for example the feasibility of developing a valid model of some legal domain in first-order logic. Moreover, this issue is only postponed, not ignored. We do intend to address this issue during the course of the ESTRELLA project, as part of Task T1.7, “Defining a set of Application Programmer Interfaces for interacting with LKIF legal knowledge systems”.

2
Notice that there are at least three kinds of rules in this model:

1. The defeasible rules in the knowledge base modeling the domain of discourse;
2. The procedural rules defining the protocol for some type of dialogue; and
3. The presumptive inference rules, called “argumentation schemes”.

One question is whether any of these types of rules could be reduced to another. For example, some authors have hypothesized that all argumentation schemes can be modeled as defeasible rules. Although such a simplification would be elegant and powerful, since it would reify schemes, allowing them to become subject to debate and argued about, as well as being introduced during the dialogue, rather than defined statically as a part of the definition of the protocol for some dialogue type, we will keep these concepts separate in the semantics presented here.

Neither argumentation schemes nor dialogue types should be confused, by the way, with problem-solving methods. These are orthogonal ideas. Walton and Krabbe [Walton and Krabbe, 1995] distinguish six main types of dialogue, including for example persuasion dialogues, negotiation dialogues and deliberation dialogues. In each type of dialogue, several roles for participants are defined, such as the proponent and proponent in persuasion dialogues. To participate in a dialogue, participants must perform various tasks and solve various problems. Here is where problem-solving methods come into play. Some of these tasks, such as constructing or finding arguments, may be common to several dialogue types. To construct arguments, argumentation schemes are applied to various sources of information. A knowledge-base could serve as such a source. Thus, argumentation schemes are applied when performing some problem-solving tasks faced by participants in dialogues.

Given this background, we can now proceed to present the semantics of LKIF Rules. An LKIF rule denotes a set of argumentation schemes, one for each conclusion of the rule, all of which are subclasses of a generic argumentation scheme for arguments from defeasible rules. Applying an LKIF rule thus is a matter of instantiating one of these argumentation schemes to produce a particular argument. Reasoning which LKIF rules is a process of applying these schemes to produce arguments to put forward in dialogues.

**Definition 19 (Premises).** There are four kinds of argument premises:

1. If \( s \) is a sentence, then \( s \) is a **positive premise**.
2. If \( s \) is a sentence, then \( \neg s \) is a **negative premise**.
3. If \( s \) is a sentence, then \( \bullet s \) is an **assumption**.
4. If \( s \) is a sentence, then \( \circ s \) is an **exception**.
5. Nothing else is a premise.

**Definition 20 (Premises of Conditions).** Let \( p \) be a function mapping conditions of LKIF rules to argument premises, defined as follows:

\[
p(c) = \begin{cases} 
  c & \text{if } c \text{ is an atomic sentence} \\
  \neg s & \text{if } c \text{ is } (\text{not } s) \\
  \bullet s & \text{if } c \text{ is } (\text{assuming } s) \\
  \circ s & \text{if } c \text{ is } (\text{unless } s)
\end{cases}
\]

3Recall, however, that we are not claiming that argumentation schemes can be modeled as or reduced to defeasible rules. Here we go in the other direction: each defeasible rule is mapped to a set of argumentation schemes. This leaves open the possibility of there being other kinds of argumentation schemes.
Arguments are of two kinds, pro arguments and con arguments. Argumentation schemes will be denoted as (presumptive) inference rules, by prefixing the conclusion of the inference rule with the direction of the argument. For this purpose, the following helping function maps the conclusions of LKIF rules to the appropriate argument conclusion. If a conclusion of an LKIF rule is an atomic sentence, $s$, then the rule is mapped to an argumentation scheme $\text{pro } s$. If a conclusion of the rule is a negated sentence, $(\text{not } s)$, then the rule is mapped to an argumentation scheme $\text{con } s$.

**Definition 21** (Direction of Argument Conclusions). Let $d$ be a function mapping a conclusion of an LKIF rule to an atomic sentence prefixed by the direction of an argumentation scheme, as follows:

$$d(c) = \begin{cases} \text{pro } c & \text{if } c \text{ is an atomic sentence} \\ \text{con } s & \text{if } c = \text{(not } s) \end{cases}$$

Now we are ready to define the semantics of LKIF rules, by mapping each rule to a set of argumentation schemes.

**Definition 22** (Schemes for Arguments from LKIF Rules). Let $r$ be an LKIF rule, with conditions $a_1 \ldots a_n$ and conclusions $c_1 \ldots c_n$. Three premises, implicit in each LKIF rule, are made explicit here. The first, $\bullet v$, where $v = \text{(valid } r \text{)}$, makes the assumption that $r$ is a valid legal rule explicit. The second, $\circ b$, where $b = \text{(rebuts } r_2 \text{ } r \text{ } c_i \text{)}$, expresses the exception where $r$ is rebutted by some other rule of higher priority. Finally, $\circ e$, where $e = \text{(excluded } r \text{ } c_i \text{)}$, expresses the exception where $r$ is excluded with respect to $c_i$.

For each $c_i$ in $c_1 \ldots c_n$ of $r$, $r$ denotes the following argumentation scheme:

$$p(a_1) \ldots p(a_n), \bullet v, \circ b, \circ e$$

To construct an argument from one of these argumentation schemes, the variables in the scheme need to be systematically renamed and then instantiated using a substitution environment, i.e. a mapping from variables to terms, constructed by unifying the conclusion of the argumentation scheme with some goal atomic statement, as in logic programming.

The valid and excluded relations used in the argumentation scheme are to be defined in the models of legal domains, as explained in Section 6.2.1. The rebuts relation, on the other hand, is defined with the following LKIF rules, implicit in every domain model:

(rule Rebuts1)

(if (and (prior ?r2 ?r1)
        (applies ?r2 (not ?p)))
    (rebuts ?r2 ?r1 ?p))

(rule Rebuts2)

(if (and (prior ?r2 ?r1)
        (applies ?r2 ?p))
    (rebuts ?r2 ?r1 (not ?p)))

Intuitively, these rules mean that one rule, $r_1$ is rebutted by another, $r_2$, if $r_2$ applies to the complement of the goal to be proved using $r_1$, and $r_2$ has priority over $r_1$. The prior relation referred to here is, like the excluded and valid relation, to be defined in the models of particular legal domains. Legal domains can define priority relationships between rules, by formalizing principles such as lex posterior using LKIF rules.

The applies relation however, is a ‘built-in’, meta-level relation which cannot be defined directly in LKIF Rules. It is defined as follows:
Definition 23 (Application of an LKIF Rule). Let $\sigma$ be a substitution environment and $G$ be an argument graph. Let $r$ be an LKIF rule with all of its variables systematically renamed. $r$ applies to a sentence $p$ in $\sigma$ and $G$ if some conclusion of $r$ is unifiable with $p$ in $\sigma$ and every condition of $r$ holds in $G$.

The concept of a sentence holding in a argument graph used in the above definition is from the Carneades model of argument [Gordon and Walton, 2006a]. The formalization of argumentation schemes used here is based on Carneades, with one minor extension to support negated sentences. In Carneades an atomic sentence is acceptable if it satisfies the proof standard assigned to it in the context of the proceeding, given all the arguments put forward thus far in the proceeding. For example, in a criminal proceeding the applicable proof standard may be ‘beyond a reasonable doubt’. We now extend the definition of acceptability to cover negated sentences as follows.

Definition 24 (Complementary Proof Standard). Let us call the proof standard for (not $P$) constructed from the proof standard $S$ for $P$ by reversing the roles of the pro and con arguments in $S$, the complementary proof standard of $S$.

For example, suppose a proof standard $S$ is satisfied if and only if there is at least one defensible argument for the sentence at issue, and no defensible con arguments. Then the complementary proof standard of $S$ is satisfied if and only if there is at least one defensible con argument, and no defensible pro arguments.

Definition 25 (Acceptability of Negated Sentence). Let $S$ be the proof standard assigned to some atomic sentence $P$. Then (not $P$) is acceptable if and only the complementary proof standard of $S$ is satisfied by $P$.

6.2.3 Discussion

Note that the argument and argument structure are essential to explaining LKIF Rules, but they are (at the moment) not part of LKIF Rules, or LKIF in general (see also the remark in 5.2) because no decision has been taken on these concepts.

Once there is such a thing as an argumentation structure that can be instantiated, the question of the model-theoretic implications of the content of premises and argument conclusions in OWL arises. This topic is subject to debate, and is not addressed in this specification.

Some believe that there simply should be no model theoretic reading of argument structures. Another proposal, in appendix C, does require consistency of this content in the OWL model, but with the addition of an identity criterium external to the OWL model.
Appendix A

Preliminaries

A.1 Notations for First-order logic: Syntax and Semantics

**Definition 26.** (*FOL*-Vocabulary)

**Individual Constants** are denoted by lower-case letters, starting from the beginning of the English alphabet: \(a, b, c\). If necessary constants are indexed: \(a_1, a_2, \ldots, b_1, b_2, \ldots, c_1, c_2, \ldots\).

**Individual Variables** are denoted by lower-case letters, starting from the end of the English alphabet: \(z, x, y\), with indexes if necessary. It is assumed that we have a countable supply of variables.

**Function symbols** are denoted by lower-case letters. Common candidates are \(f, g, h\), with indexes if necessary. Function symbols may have arity \(i\) such that \(0 \leq i \leq n\), for \(n \in \mathbb{N}\). For this reason, a precise notation would require to specify the arity of the function symbol, e.g. \(h^2_4\) refers to the function symbol \(h\) with index 4 and arity 2. Notice that the upper-index refers to the arity of the function symbol. Following common practice, one can leave apart the specification of the arity, unless required by the context.

**Relation Symbols** are denoted by upper-case letters, e.g. \(A, B, C, P\), with indexes if necessary. Predicates can have arity and the same conventions for function symbols apply to predicates as well.

**Connectives** are denoted by the symbols \(\neg\) (negation), \(\land\) (conjunction), \(\lor\) (disjunction), \(\to\) (implication), \(\forall\) (universal quantifier), \(\exists\) (existential quantifier).

**Auxiliary symbols** are the parenthesis (,).

Let \(V_I\) be the set of individual constants, \(V_X\) the set of variables, \(V^i_F\) the set of functions symbols of arity \(i\), and \(V^n_R\) the set of \(n\)-place relation symbols, with \(i \in \mathbb{N}\).

**Definition 27 (*FOL*-Term).** The set \(\text{FOL-TERM}\) is the smallest set such that

**Base:** \(c \in \text{L-TERM}\) and \(x \in \text{L-TERM}\), for any \(c \in V_I\) and any \(x \in V_X\).

**Step:** \(f^i(t_1, \ldots, t_i) \in \text{L-TERM}\), with \(f^i\) any function such that \(0 \leq i \leq n\), for \(n \in \mathbb{N}\), and \(t_1, \ldots, t_i\) a possibly empty sequence of terms.

The set \(\text{FOL-TERM}_c\) of closed terms is given by \(\text{FOL-TERM}\) minus all the terms where a variable occur.

**Definition 28 (*FOL*-Formula).** The set \(\text{FOL-FORM}\) of first-order formulas \(\varphi\) is given by the rule:

\[
\text{FOL-FORM} ::= P_i(t_1, \ldots, t_i) | t_i = t_j | \neg \varphi | \varphi_1 \lor \varphi_2 | \varphi_1 \land \varphi_2 | \varphi_1 \to \varphi_2 | \exists x(\varphi(x)) | \forall x(\varphi(x)),
\]

where \(P_i\) is any relation symbol in such that \(0 \leq i \leq n\), for \(n \in \mathbb{N}\).
Observation 9 (Abbreviation). It is always better to prefer a notation that is the least heavy one. To this end, there are a number of common abbreviations:

- ⊥ abbreviates any contradictory formula, and ⊤ abbreviates any tautology.
- if \( P^0 \), then it can be abbreviated by \( p \), where \( p \) is a propositional variable;
- \( \leftrightarrow \) is an abbreviation of connectives \( ←, ∧ \) and \( → \), in the obvious way;
- \( t_i ≠ t_j \) abbreviates \( ¬(t_i = t_j) \).

Definition 29 (\( \mathcal{FOL} \)-Model). A model for our language \( \mathcal{FOL} \), call it \( \mathcal{M}_\mathcal{FOL} \) is a pair \( ⟨\mathbb{D}, I⟩ \), where

- \( \mathbb{D} \) is a possibly infinite domain of individuals;
- \( I = I_1 ∪ I_2 ∪ I_3 ∪ g \), where:
  
  - \( I_1 : V_I \rightarrow \mathbb{D} \);
  - \( I_2 : V_R^i \rightarrow \wp(\mathbb{D}^i) \);
  - \( I_3 : V_F^i \rightarrow \wp(\mathbb{D}^i) \);
  - \( g : V_X \rightarrow \mathbb{D} \).

In other terms, \( I \) assigns to every individual constant in \( V_I \) an element of the domain \( \mathbb{D} \); to any relation symbol or function symbol with arity \( i \) a subset of the powerset of \( \mathbb{D}^i \); and to any variable in \( V_I \) an element of the domain.

Definition 30. \( I \) can be recursively defined to deliver the denotation of terms:

Base: \( I(c) = I_1(c) \) and \( I(x) = I_g(x) \), with \( c, x \) an individual constant and a variable, respectively.

Step: \( I(f_i(t_1, ..., t_i)) = ⟨I(t_1), ..., I(t_i)⟩ ∈ I_3(f_i) \), with \( f_i \) a function with arity \( i \) and \( t \) terms.

Definition 31 (\( \mathcal{FOL} \)-Truth). The writing \( \mathcal{M}_\mathcal{FOL} \models \varphi \) means that a formula \( \varphi \) is true in the model \( \mathcal{M}_\mathcal{FOL} \). The truth relation \( \models \) is recursively defined in the usual way.

Definition 32 (\( \mathcal{FOL} \)-Logical entailment). The relation of logical entailment is denoted and defined as follows:

\[ \Gamma \models \varphi :⇔ \text{ for all } \mathcal{M}_\mathcal{FOL} \text{ and for all } \psi ∈ \Gamma \text{ if } \mathcal{M}_\mathcal{FOL} \models \psi \text{, then } \mathcal{M}_\mathcal{FOL} \models \varphi, \]

where \( \Gamma \) is a set for formulas.

Observation 10 (Ambiguity of \( \models \)). There is a fundamental equivocality in the use of \( \models \) for designating the truth-relation and the logical entailment relation. This is in line with common practice, and thus we will stick to this convention.

Observation 11 (Sloppy notation for \( \models \)). Strictly speaking the relation of logical entailment should be indexed, e.g. \( \models_{\mathcal{FOL}} \). Following the rule of using the lightest notation, the index has been dropped.
A.2 Notation for Description Logic: Syntax and Semantics

Definition 33 (DL-Vocabulary).

Individual Constants are denoted by lower-case letters, starting from the beginning of the English alphabet: \( a, b, c \), with indices if necessary.

Class symbols are denoted by upper-case letters, starting from the beginning of the English alphabet: \( A, B, C \), with indices if necessary. Notice that concepts have only arity 1.

Property symbols are denoted by upper-case letters. Usual candidate is \( P, S, T \), with indexes if necessary. Notice that roles have only arity 2.

Constructors for concepts and roles are denoted by the symbols \( \neg, \sqcap, \sqcup, \forall \) and \( \exists \).

Formula Constructors (which apply to concept and properties) are denoted by \( \equiv \) and \( \sqsubseteq \).

Auxiliary symbols are the parenthesis ( ).

Let \( V_I \) be the set of individual constants, \( V_C \) the set of classes, and \( V_P \) the set of properties.

Definition 34 (DL-Class). The set \( \mathcal{DL}-\text{CLASS} \) is defined by the following rule:

\[
\mathcal{DL}-\text{CLASS} ::= A \mid \top \mid \bot \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall P.C \mid \exists P.C \leq n P
\]

Definition 35 (DL-Properties). The set \( \mathcal{DL}-\text{PROP} \) is defined by the following rule:

\[
\mathcal{DL}-\text{PROP} ::= P \mid \neg P \mid P_1 \sqcap P_2 \mid P_1 \sqcup P_2 \mid P_1 \circ P_2 \mid P_1 - P^+ \mid P_1 \circ P_2 - P^+ \mid P \mid C \mid P \mid \top
\]

A few clarifications on perhaps unclear properties: \( P_1 \circ P_2 \) is the composition of \( P_1, P_2 \); \( P^1 \) is the inverse role of \( P \); \( P^+ \) is the reflexive-transitive closure of \( P \); \( P \mid C \) is \( P \) with its domain restricted to \( C \); and \( P \mid C \) is \( P \) with its range restricted to \( C \).

Definition 36 (DL-Axioms). The set \( \mathcal{DL}-\text{AXIOM} \) for \( \mathcal{DL} \)-formulas \( \varphi \) is defined by the following rule:

\[
\mathcal{DL}-\text{AXIOM} ::= C(a) \mid C_1 \equiv C_2 \mid C_1 \equiv C_2 \mid R(a, b) \mid P_1 \equiv P_2 \mid P_3 \equiv P_4
\]

where \( C, C_1, C_2 \) are any class in \( \mathcal{DL}-\text{CLASS} \), \( a, b \) are any constant in \( I_I \), and \( P, P_1, P_2 \) any property in \( \mathcal{DL}-\text{PROP} \).

Definition 37 (DL-Model). A model \( \mathfrak{M}_{\mathcal{DL}} \) for our language \( \mathfrak{M}_{\mathcal{DL}} \) is a pair \( \langle \mathbb{D}, I \rangle \), where

- \( \mathbb{D} \) is a possibly infinite domain of individuals;
- \( I = I_I \cup I_2 \cup I_3 \), where:
  - \( I_I : V_C \mapsto \varphi(\mathbb{D}) \);
  - \( I_2 : V_P \mapsto \rho(\mathbb{D} \times \mathbb{D}) \);
  - \( I_3 : V_I \mapsto \mathbb{D} \).

In other terms \( I \) assigns to every concept in \( V_C \) a subset of the domain \( \mathbb{D} \), and to any property in \( V_P \) a subset of the cartesian product \( \mathbb{D} \times \mathbb{D} \); and to any constant an element of the domain \( \mathbb{D} \).

Observation 12 (Recursive extension of \( I_I \) and \( I_2 \)). The interpretation function for concepts in \( V_C \) and properties in \( V_P \) have an obvious recursive extension to \( \mathcal{DL}-\text{CLASS} \) and \( \mathcal{DL}-\text{PROP} \).
**Definition 38 (DL-Truth).** The writing $\mathcal{M}_{DL} \models \varphi$ means that a formula $\varphi$ is true in the model $\mathcal{M}_{DL}$.

**Definition 39 (DL-Logical entailment).** The property of logical entailment is denoted and defined as follows:

$$\Gamma \models \varphi \iff \text{ for all } \mathcal{M}_{DL} \text{ and all } \psi \in \Gamma, \text{ if } \mathcal{M}_{DL} \models \psi, \text{ then } \mathcal{M}_{DL} \models \varphi,$$

where $\Gamma$ is set for formulas.

**A.3 On the Relation between FOL and DL**

**Observation 13.** (On Table 1.1) It is assumed that for any concept $C$ or property $P$ in $DL$, there is an homonymous one-place predicate $C$ and two-place predicate $P$ in $L$. Strictly speaking, this correspondence should be given by a recursive translation, e.g. $C \mapsto \varphi_C(x)$, where $C$ is in $DL$ and $\varphi_C(x)$ is in $FOL$. For a precise treatment of this, see [van Dalen, 2004].
Appendix B

Existing Languages and Technologies

In this chapter we give a detailed introduction to available knowledge representation languages and technologies which should be considered as building blocks of the LKIF standard.

At first we discuss description logics and rule languages, two well described methods for representing domain knowledge when creating knowledge representation systems.

From the technological side available reasoning engines and editors are examined, as these are crucial for creating and handling complex knowledge bases.

B.1 Description logics

Description logics are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way. Description Logics have initially been designed to fit object-centric knowledge representation formalisms like semantic networks and frame systems with a formal and declarative semantics. The name description logics is motivated by the fact that, on the one hand, the important notions of the domain are described by concept descriptions, i.e., expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates) using the concept and role constructors provided by the particular DL. On the other hand, DLs differ from their predecessors, such as semantic networks and frames, in that they are equipped with a formal, logic-based semantics. Consequently, most description logics turn out to be subsets or variants of C2, the subset of first-order logic extended with counting quantifiers, where formulae with no function symbols and maximum two variables, which is known to be decidable.

In addition to the basic description formalism of DLs, they are usually equipped with a terminological and an assertional formalism. In its simplest form, terminological axioms can be used to introduce names (abbreviations) for complex descriptions. A TBox is constituted by a finite set of terminological axioms which define subsumption and equivalence relations on classes and properties. The assertional formalism can be used to state properties of individuals. Assertional axioms or Assertions introduce Individuals, i.e. instances of a class, into the knowledge base and relate individuals with each other and the introduced terminology. [Volz, 2004b]

Description logic systems provide their users with various inference capabilities that deduce implicit knowledge from the explicitly represented knowledge. In order to ensure a reasonable and predictable behavior of a DL system, these inference problems should at least be decidable for the DL employed by the system, and preferably of low complexity. Consequently, the expressive power of the DL in question must be restricted in an appropriate way. If the imposed restrictions are too severe, however, then the important notions of the application domain can no longer be expressed. Investigating this trade-off between the expressivity of DLs and the complexity of their inference problems has been one of the
most important issues in DL research.

The complexity of the S family of languages that are relevant for the Semantic Web was listed by [Volz, 2004b]:

<table>
<thead>
<tr>
<th>Description Logic</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>SI</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>SH</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>SHIF</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>SHIQ</td>
<td>EXPTIME-complete</td>
</tr>
<tr>
<td>SHIOQ</td>
<td>NEXPTIME-hard</td>
</tr>
</tbody>
</table>

B.1.1 OWL

The OWL Web Ontology Language is intended to provide a language that can be used to describe the classes and relations between them that are inherent in Web documents and applications. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S, and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web. OWL is a revision of the DAML+OIL web ontology language incorporating lessons learned from the design and application of DAML+OIL [Mcguinness and van Harmelen, 2004]. Detailed specification is provided in the following papers: [Heflin et al., 2002], [Patel-Schneider et al., 2003], [Smith et al., 2004], [Mcguinness and van Harmelen, 2004].

OWL is syntactically layered on RDF, extending it with additional vocabulary, but it differs from Description Logics in several ways:

1. N-Ary language constructs such as conjunction (u) have to be encoded into several RDF statements.
2. OWL graphs can contain circular syntactic structures which are not possible in Description Logics.
3. Due to the circular meta-model of RDF, classes and properties can be made instances of themselves. For example, it would be possible to assert the statement rdfs:subClassOf(owl:Class,owl:Class).
4. The identifiers used for classes, properties and individuals do not have to be disjoint.

OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

**OWL Full** is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full. [Mcguinness and van Harmelen, 2004] Inference in OWL Full is clearly undecidable as OWL Full does not include restrictions on the use of transitive properties which are required in order to maintain decidability [Horrocks et al., 1999].

**OWL DL** supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions. [Mcguinness and van Harmelen, 2004] So OWL DL is a subset of OWL Full that makes the following restrictions:
1. The OWL vocabulary can not be used as identifiers for classes, properties or individuals;

2. Class constructors with syntactic cycles are disallowed.

3. The sets of identifiers used for classes, properties, and individuals must be disjoint;

4. Cardinality restrictions may not be stated on properties which are transitive.

OWL DL is so named due to its correspondence with description logics, a field of research that has studied a particular decidable fragment of first order logic. OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems. OWL DL is supported by theorem provers Pellet, Racer, FaCT, Hoolet.

**OWL Lite** supports those users primarily needing a classification hierarchy and simple constraints. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies. Owl Lite also has a lower formal complexity than OWL DL. [Mcguinness and van Harmelen, 2004] OWL Lite is created from OWL DL by imposing the following restrictions:

1. Disallows the usage of the following elements of the OWL DL vocabulary:
   - owl:intersectionOf, owl:unionOf, owl:complementOf, owl:oneOf, owl:hasValue
2. Limits the values of stated cardinalities to 0 and 1;
3. Requires the usage of class identifiers in OWL restrictions;

OWL Full language is undecidable. Therefore complexity results are only of interest for OWL DL and OWL Lite. OWL DL and OWL Lite are both based on Description Logics with an RDF syntax. They can therefore exploit the considerable existing body of description logic research, e.g., to define the semantics of the language and to understand its formal properties, in particular the decidability and complexity of key inference problems [Nutt et al., 1997]. Since OWL DL (and even OWL Lite) suffer from a very high computational complexity, and efficient reasoners able to deal with them in a sufficiently scalable way are still one of the main challenges for the community.

Complexity follows directly from the correspondence to Description Logics As the translation from OWL DL ontologies to SHINO(D) knowledge bases can be performed in polynomial time [Horrocks and Patel-Schneider, 2004b], the results for the complexity of knowledge base satisfiability in SHINO(D) still hold, so OWL DL is NEXPTIME complete. Similarly, since the translation of OWL Lite axioms to SHIF(D) knowledge bases can be computed in polynomial time and results in a linear increase in size of the knowledge base, OWL Lite entailment has the same complexity as SHIF(D) knowledge base satisfiability, which is EXPTIME complete. [Volz, 2004b]

OWL as certain variant of a Description Logic [Horrocks and Patel-Schneider, 2004b], it shares its benefits and limitations with those logics: Intractability is one of the main limitations of Description Logics (such as OWL) Another major problem is the lack of algorithms which allow to reason with non-trivial ABoxes in practice.

The relation of Description Logic with other modeling languages is not clearly defined and discussed in the literature. Therefore it would be useful to facilitate the transition by relating the elements of the language with the elements of other well-known modeling languages. This relation can also be used to bootstrap [Stojanovic et al., 2002] the Semantic Web by reusing available information models.

The next limitation is related to the system base of the language. Only three systems, FaCT [Horrocks et al., 1999], Racer [Haarslev and Moller, 2001] and Cerebra [Network Inference Inc., 2003],
are able to provide limited support for reasoning with OWL today. All three systems are incomplete and do not support all language features. For example, FaCT (Horrocks et al., 1999) offers no support at all for ABoxes, while Racer (Haarslev & Moller, 2001) departs from the OWL semantics by taking the Unique Names Assumption (UNA).

The majority of data on the Web is no longer static but resides in databases. Exposing this data to the Semantic Web requires access to the content of underlying databases [Volz, 2004b]. Description Logic reasoners such as FaCT [Horrocks et al., 1999] or Racer [Haarslev and Moller, 2001] currently require the replication of that data to enable ontology-based processing. Logic databases, however, can access relational databases directly through built-in predicates. [Volz, 2004b]

Mediating between distributed data using mappings between ontologies is becoming a necessary task in Semantic Web communities [Rousset, 2002]. Different proposals [Maedche et al., 2002], [Omelayenko, 2002], [Noy and Musen, 2000] have already been made to specify mappings between ontologies. Logic databases can provide the functionality needed to execute the necessary transformations specified by such mappings.

Volz has observed some additional limitations of OWL [Volz, 2004b]:

Ill-defined layering is one of the main limitations of OWL that derives from layering the language on RDF and defining the layering based on political aspects. For the latter case, OWL Lite does not meet its initial anticipation to be an “easy” language.

Another problem is that OWL Ontologies are monolithic, meaning they can be separated into a set of ontologies, but have to be interpreted as one unified ontology. In OWL the the only way of modularization is to include entire ontologies. This inclusion is fragile, since the included part is specified by location (which is usually different from the ontology identifier).

Datatyping in OWL is still not worked out that is further limitation. Finally the complexity of the language is high and this way “empirical” tractability could not be achieved for ABox problems up till now. Moreover, no “practical” algorithm is currently known for any SHINO(D) reasoning problem.

B.1.2 Description Logic Programs

DLPs – Description Logic Programs – are the subset of Description Logic which can be represented in the decidible Datalog subset of Logic Programming. DLP is attractive for two reasons: 1) efficient reasoners for DLP can be easily implemented by translating DL expressions to rules for existing rule-based inference engines, such as CLIPS or Jess, or logic programming languages in the Prolog family, such as XSB; and 2) more importantly, ontologies represented in DLP can be easily combined with rules without leaving the computationally useful subsets of First-Order Logic.

In [Grososf et al., 2003] the authors have shown how DLP can capture a significant fragment of DAML+OIL, including the whole of the DAML+OIL fragment of RDFS, simple frame axioms and more expressive property axioms. The RDFS fragment of DL permits: stating that a class D is a Subclass of a class E; stating that the Domain of a property P is a class C; stating that the Range of a property P is a class C; stating that a property P is a Subproperty of a property Q; stating that an individual b is an Instance of a class C; and stating that a pair of individuals (a,b) is an Instance of a property P. Additional DLP expressively permits (within DL): using the Intersection connective (conjunction) within class descriptions (i.e., in C, D, or E above); using the Union connective (disjunction) within subclass descriptions (i.e., in D above); using (a restricted form of) Universal quantification within superclass descriptions (i.e., in E above); using (a restricted form of) Existential quantification within subclass descriptions (i.e., in D above); stating that a property P is Transitive; stating that a property P is Symmetric; and stating that a property P is the Inverse of a property Q.
Section B.1  Description logics

B.1.3  OWL DLP

OWL DLP is one name for the DLP subset of OWL. Although it is not one of the sub-
languages formally defined in the OWL W3C standard (OWL Lite, OWL DL, and OWL
Full), users of OWL need only to follow a few simple guidelines to assure their ontologies
are within the DLP subset.

Description Logic Programs (DLPs) are ontological knowledge bases which lie within
the intersection of OWL and Logic Programming. They could be described as a naive
Horn fragment of OWL DL. As such they provide a formal link between different ontology
representation paradigms, and serve multiple purposes in theoretical research and practical
applications. They are also a fragment of the more expressive Horn-SHIQ, which is more
powerful but follows similar intuitions [Vrandecic et al., 2005].

DLP is not yet another knowledge representation language, but rather a bridge to bring
two paradigms together. As such it imposes certain constraints on OWL DL in order to
guarantee that all axioms stated are transformable in an efficient way to Horn clauses, i.e.
rules in the sense of traditional logic programming, but it does not define new semantics or
syntax. [Vrandecic et al., 2005] Every OWL DLP ontology is a OWL DL ontology as well.
In other words DLP is a proper subset of OWL DL, that is able to express the following
features of OWL-DL:

• concept disjointness,
• domains and ranges of properties,
• inverse and symmetric properties,
• functional and inverse-functional properties,
• subproperty and equivalence relations between object properties,
• transitive properties, and
• a limited form of General Concept Inclusion axioms (GCIs).

The constraints imposed on DLP regarding the more expressive OWL DL, lead to DLP
enjoying a much better data complexity (polynomial) and combined complexity (exptime).
This allows expecting more efficient and responsive tools than full OWL DL reasoning will
ever be able to achieve due to the complexity of DL reasoning algorithms.

So reasoning over DLP will always lead to a lower complexity than reasoning over
DL could. [Vrandecic et al., 2005] At the same time, the semantic expressivity lost due to
the restriction to DLP hardly matters in practice: [Volz, 2004b] analysed that 99% of
the axioms in ontologies taken from the daml.org repository are within the DLP fragment.
[Vrandecic et al., 2005] provided a description of an easy to use sublanguage of DLP by
listing all the constructors one may use freely in an OWL ontology without running the
risk of leaving DLP. Allowed OWL constructors: Class, Thing, subClassOf, Property, sub-
PropertyOf, domain, range, Individual, equivalentClass, equivalentProperty, sameAs, different-
From, AllDifferent, ObjectProperty, DatatypeProperty, inverseOf, TransitiveProperty,
SymmetricProperty, FunctionalProperty. InverseFunctionalProperty, intersectionOf. This
doesn’t mean that other constructors are forbidden in OWL DLP. It is just that their use
underlies further constraints.

At the same time, it should be considered that; the logical consequences that an OWL
1.1 reasoner would draw from a DLP ontology differs from the ones that would be ob-
tained using an LP engine. Typically, LP reasoners adopt the closed world assumption and
are, consequently, non-monotonic. OWL 1.1, however, is monotonic and adopts the open-
world assumption. DLP adopts the semantics of OWL 1.1 and not the semantics of LP
[Vrandecic et al., 2005].

A further advantage of DLP is that standard OWL editors can be used to create valid
DLP ontologies.
B.1.4 OWL Horst

OWL Horst\(^1\) is a dialect of OWL developed by Hermant ter Horst who defined RDFS extensions aiming at more general rule support and a fragment of OWL, which is more expressive than DLP and fully compatible with RDFS. As it was discussed in [ter Horst, 2005], this language has a number of important features:

- It is a proper (backward-compatible) and natural extension of RDFS. In contrast to OWL DLP, it imposes no constraints on the RDFS semantics. Meta-classes (classes as instances of other classes) are not disallowed in OWL Horst. Furthermore, OWL Horst does not encompass the unique name assumption;
- Unlike the DL-based rule languages, like SWRL, R-entailment provides a formalism for rule extensions without the DL-related constraints;
- Its complexity is not as extreme as that of SWRL or of any of the other approaches that combine DL ontologies with rules. OWL Horst makes possible decidable rule extensions of OWL.

First, Horst defines R-entailment, which extends the RDFS-entailment in the following ways:

- it can operate on the basis of any set of rules R (i.e. allows for extension or replacement of the standard set, defining the semantics of RDFS);
- it operates over the so-called generalized RDF graphs, where blank nodes can appear as predicates (a possibility disallowed in RDF);
- rules without premises are used to declare axiomatic statements;
- rules without consequences imply inconsistency.

Horst extends and modifies the D-entailment rules from Hayes in two steps as follows:

- D\(^*\) adds to RDFS support for reasoning with typed literals;
- pD\(^*\) adds rules that provide partial support for OWL.

The following primitives are supported: FunctionalProperty, InverseFunctionalProperty, SymmetricProperty, TransitiveProperty, sameAs, inverseOf, equivalentClass, equivalentProperty, onProperty, hasValue, someValuesFrom, allValuesFrom, differentFrom, disjointWith. The last two primitives are supported through inconsistency rules, which “fire” in case of the so-called P-clashes. It is important to bear in mind that some of the primitives are only partially supported; most notable, the standard OWL entailments related to someValuesFrom and allValuesFrom are supported in one direction only (i.e. there is no full support for the iff-semantics of these OWL primitives).

B.1.5 OWL 1.1 / SROIQ

OWL 1.1 provides extra Description Logic expressive power, moving from the SHOIN Description Logic that underlies OWL DL to the SROIQ Description Logic. The development of OWL 1.1 is currently not a W3C activity, but a Working Group is involved in the development. The additions of OWL1.1 are [Mcguinness and van Harmelen, 2004]:

\(^1\)http://www.ontotext.com/inference/rdfs_rules_owl.html
1) **Qualified cardinality restrictions** using the same syntactic tokens as regular cardinality restrictions (minCardinality, maxCardinality, and cardinality) but adding a dataRange or a description, as in

```
restriction(friend minCardinality(2 hacker));
```

Syntax: 'minCardinalityQ('non-negative-integer description')'
Syntax: 'maxCardinalityQ('non-negative-integer description')'
Syntax: 'cardinalityQ('non-negative-integer description')'

2) **Local reflexivity restrictions** on non-complex properties only, allowing constructs like `restriction(likes self)` for narcissists;
Syntax: new IndividualRestrictionComponent self

3) **Reflexive, irreflexive, symmetric, and anti-symmetric properties** for non-complex properties only, allowing constructs like `ObjectProperty(knows reflexive)` and `ObjectProperty(husband irreflexive antisymmetric)`. (e.g.: relatedTo is reflexive; spouse irreflexive; husband anti-symmetric)
Syntax: new Reflexive, Irreflexive, and AntiSymmetric tags for ObjectProperty axiom

4) **Disjoint properties** for non-complex properties only, allowing constructs like `DisjointProperties(child spouse)`
Syntax: 'DisjointProperties('propertyID propertyID {propertyID}')'

5) **Property chain inclusion axioms** allowing constructs like

```
SubPropertyOf(propertyChain(owns part) owns), provided that no cyclic inclusions result.
Syntax: 'SubPropertyOf('roleChain individualvaluedPropertyID')'
roleChain ::= 'roleChain('individualvaluedPropertyID
{individualvaluedPropertyID}')'
```

In particular additions 3, 4, 5 make OWL1.1 a lot more useful for knowledge representation than the existing OWL standard.

(SubPropertyOf(propertyChain(parent parent) grandparent) is still not possible).

**Expanded datatype expressiveness** Another new feature in QWL 1.1 is the expanded datatype expressiveness. The definition of SROIQ does not provide for datatypes and metamodeling. Therefore, the semantics of OWL 1.1 is defined a direct model-theoretic way, by interpreting the constructs of the functional-style syntax from [Patel-Schneider et al., 2003].

OWL 1.1 provides several ways to define a range over data values. A datatype is a fundamental type of data range that is defined by a URI. Each datatype URI is associated with a predefined arity (note that the same datatype URI cannot be used with different arities). The following datatypes are supported by OWL 1.1: xsd:string, xsd:boolean, xsd:decimal, xsd:float, xsd:double, xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName, xsd:integer, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:nonNegativeInteger, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte, and xsd:positiveInteger. [Patel-Schneider et al., 2006]

Complex data ranges can be constructed from the simpler ones using the dataComplemementOf constructor, which takes a data range and returns its complement (with the same
arity). Furthermore, data ranges consisting exactly of the specified set of constants can be formed using the dataOneOf constructor (and it has the arity one). Finally, the datatype-Restriction constructor creates a data range by applying a facet to a particular data range.

**Meta-modeling**  The application of meta-ontology constructs is also ensured by OWL1.1 through punning. Punning is a limited form of metamodeling that pushes annotation properties to the next level. This meant that the same name could be used as any or all of a class, property or individual.

The semantics of OWL 1.1 will break the connection between classes and individuals that share a name. This diverges from the situation in RDF. The effect is that the following will be legal

\[
\text{Class(Person)} \\
\text{Individual(Person)} \\
\text{Individual(John Person)} \\
\text{SameIndividualAs(Person Rock)}
\]

but that it does not entail

\[
\text{Individual(John Rock)}
\]

The use of annotations is also more flexible in OWL1.1, than in other OWL dialects. With punning, general properties can be placed on class names when used as individuals, as in

\[
\text{Individual(John)} \\
\text{Class(Person)} \\
\text{ObjectProperty(createdBy range(Person))} \\
\text{Individual(Person restriction(createdBy value(John)))}
\]

It would also be possible to allow restrictions in class axioms as syntactic sugar, allowing

\[
\text{Individual(John)} \\
\text{ObjectProperty(createdBy range(Person))} \\
\text{Class(Person restriction(createdBy value(John)))}
\]

**Additional features of OWL1.1**  Both Pellet and FaCT implement OWL 1.1. An integrated toolset was produced: FaCT++ was extended to provide reasoning support and a new version of Protégé-OWL was produced to support the editing of OWL 1.1 ontologies. [Horridge and Tsarkov, 2006]

Requirement of the separation of names for classes, properties and individuals was lifted. This mechanism, which is also known as punning, was already used by FaCT++.

**B.1.6 Summary**

Figure B.1 represents a simplified map of the complexity of a number of OWL dialects and related languages, as well as of their disposition towards DL and LP-based semantics. [http://www.ontotext.com/inference/rdfs_rules_owl.html]

**B.2 Rule Languages**

In the following sections we will have a look at the most popular rule-based knowledge representations available. Rule languages in general build from rules, implications between an antecedent and a consequent.
B.2 Rule Languages

We will focus on four different rule languages. Datalog is an early approach well studied but also very limited, stemming from the Prolog programming language. RuleML is a markup language for standardization, developed collecting lots of experience in rule-based systems. SWRL stems from RuleML, providing a rule extension for the OWL ontology language. WRL is a rule-based ontology language developed for the Semantic Web, but independent from OWL.

B.2.1 Datalog

Datalog is a database query language based on the Logic Programming paradigm, designed and studied a lot in the mid 1980’s. It stems from the efforts to integrate artificial intelligence (AI) with large databases. When Datalog was created AI systems using existing data were only connected with ad hoc interfaces with loose coupling, e.g. connecting Prolog systems to relational databases.

Datalog is a query and rule language for deductive databases that syntactically is a subset of Prolog. In many respects Datalog is a simplified version of generic Logic Programming [Ceri et al., 1989].

A logic program consists of a finite set of facts and rules. In the formalism of Datalog both facts and rules are represented as Horn clauses of the general shape:

\[ L_0 : \neg L_1, \ldots, \neg L_n \]

where each \( L_i \) is a literal of the form \( p_i(t_1, \ldots, t_{R_i}) \) such that \( p_i \) is a predicate symbol and \( t_j \) are terms. A term is either a constant or a variable.

The left-hand side of a Datalog clause is called its head and the right-hand side is called its body. Clauses with an empty body represent facts, also called extensional predicates; clauses with at least one literal in the body represent rules. Applying rules computed relations, intensional predicates are derived.

Any Datalog program \( P \) must satisfy two safety conditions:

- Each fact of \( P \) is ground.
- Each variable which occurs in the head of a rule of \( P \) must also occur in the body of the same rule.

These conditions guarantee that the set of all facts that can be derived from \( P \) is finite.
Let’s have a simple Datalog example. The fact “John is the parent of Bob” can be represented as:

\[ \text{parent}(\text{bob, john}) \]

Now we can introduce a rule “If X is a parent of Y and if Y is a parent of Z, then X is a grandparent of Z”:

\[ \text{grandparent}(Z, X) : \neg \text{parent}(Y, X), \text{parent}(Z, Y) \]

Query evaluation with Datalog is sound and complete and can be done efficiently even for large databases. For restricted forms of Datalog that don’t allow any function symbols, safety of query evaluation is guaranteed. The exact expressive power of Datalog is completely understood in certain cases [Kolaitis and Vardi, 1990], also has a meaningful message for the scope of applications. If only ordered databases are considered, then a query is expressible by a Datalog program if and only if it is computable in polynomial time.

Extensions to Datalog were made to make it object-oriented, or to allow disjunctions as heads of clauses [Eiter et al., 1997]. Both extensions have major impacts on the definition of the semantics and the implementation of a corresponding Datalog interpreter.

There are many implementations available for Datalog or one of its extensions. An example is bddbddb\(^2\), which stands for BDD-Based Deductive DataBase. It is an implementation of Datalog done at the Stanford University.

### B.2.2 RuleML

RuleML\(^3\) is a markup language developed as the canonical Web language for rules since August of 2000 [Boley et al., 2001]. It covers both forward and backward rules for deduction, rewriting and transformation tasks.

RuleML is defined by the Rule Markup Initiative, a group of academic and industrial partners, who are not focusing on academic research, but inter operation between industry standards. The standards taken into consideration include JSR 94, SQL’99, OCL, BPMI, WSFL, XLang, XQuery, RQL, OWL, DAML-S, and ISO Prolog. The new rule language is aimed to provide interoperability between established systems, such as CLIPS, Jess, ILOG JRules, Blaze Advisor, Versata, MQWorkFlow, BizTalk and Savvion. The Initiative develops the specification of RuleML and provides transformations from and to other rule systems. Also use cases are collected on the aimed application domains, e.g. business rules in e-commerce and diagnosis rules in engineering.

Rules can be stated in natural language or in some formal notation. RuleML is an XML-based markup language supporting both approaches. Rule-specific notations are made available in the RuleML namespace, which can be mixed with a namespace for natural-language (XHTML) texts and possible domain-specific namespaces (much like MathML is mixed into such domain texts).

The current RuleML design encompasses a hierarchy of rule types. The root node is “general rules”, and are specialized into subcategories as shown below:

- **reaction rules** return no value
- **transformation rules** ‘event’ trigger is always activated
  - **derivation rules** like characteristic functions, on success just return true
  - **facts** have an empty (hence, ‘true’) conjunction of premises
  - **queries** have an empty (hence, ‘false’) disjunction of conclusions

\(^2\)http://bddbddb.sourceforge.net/
\(^3\)http://www.ruleml.org/
When formulating rules, we need a content model with strict semantics. RuleML is a family of sublanguages whose root allows access to the language as a whole and whose members allow to identify customized subsets of the language. Therefore, RuleML’s specification employs modular XML Schemas as pioneered by XHTML. Users of RuleML are able to specify which sublanguage in the family (each corresponding to an expressive class, e.g. Datalog and Hornlog) best suits their needs.

According to the current specification RuleML includes two family of sublanguages, Derivative and Production (PR) RuleML. The former includes Datalog, Horn logic and first-order logic, Production RuleML is still under development.

### B.2.3 SWRL

The Semantic Web Rule Language (SWRL, pronounced “Swirl”) is a proposal for a rules-language for the Semantic Web, combining sublanguages of the OWL Web Ontology Language (OWL DL and OWL Lite) with those of the Rule Markup Language (Unary/Binary Datalog) [Horrocks et al., 2004b].

Since May 2004 the SWRL specification is a W3C member submission, offered by the National Research Council of Canada, Network Inference and Stanford University, in association with the Joint US/EU ad hoc Agent Markup Language Committee.

SWRL combines OWL DL with the Unary/Binary Datalog RuleML sublanguages. The proposal extends the set of OWL axioms to include Horn-like rules. It thus enables Horn-like rules to be combined with an OWL knowledge base.

The proposed rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Both the antecedent and consequent may contain zero or more conditions (atoms). An empty antecedent is treated as trivially true (i.e. satisfied by every interpretation), so the consequent must also be satisfied by every interpretation. An empty consequent is treated as trivially false (i.e. not satisfied by any interpretation), so the antecedent must also not be satisfied by any interpretation. These can be used to provide unconditional facts. More than one atoms are treated as conjunctions, i.e. the antecedent or consequent holds iff all of their constituent atoms hold.

Atoms can be of the form $C(x)$, $P(x, y)$, sameAs(x, y), differentFrom(x, y), or builtIn(r, x,...) where $C$ is an OWL description or data range, $P$ is an OWL property, $r$ is a built-in relation, $x$ and $y$ are either variables, OWL individuals or OWL data values, as appropriate. The sameAs and differentFrom two forms can be seen as “syntactic sugar”: they are convenient, but not necessary, as they can already be expressed using the combined power of OWL and rules without explicit (in)equality atoms.

Atoms may refer to individuals, data literals, individual variables or data variables. Variables are universally quantified, with their scope limited to a given rule. Only variables that occur in the antecedent of a rule may occur in the consequent (a condition usually referred to as “safety”). This safety condition does not, in fact, restrict the expressive power of the language, because existentials can already be captured using OWL someValuesFrom restrictions.

It is easy to see that OWL DL becomes undecidable when extended with SWRL as rules can be used to simulate role value maps [Schmidt-Schauß, 1989].

SWRL built-ins provide a large set of functions organized in a hierarchical taxonomy. Many existing built-ins are reused from XQuery and XPath. The available modules from the specification are built-ins for comparison, math functions, strings, date, time, duration, URI and list functions.
SWRL has an abstract syntax extending the abstract syntax of OWL. An XML and RDF syntax is also given for these rules based on RuleML, OWL XML and OWL RDF presentation syntax.

The following example from the SWRL specification demonstrates a very common use for rules to move property values from one individual to a related individual. The example describes the fact that the style of an art object is the same as the style of the creator. Using an informal representation:

\[
\text{Artist}(x) \land \text{artistStyle}(x, y) \land \text{Style}(y) \land \text{creator}(z, x) \Rightarrow \text{style/period}(z, y)
\]

The same rule in SWRL abstract syntax:

\[
\text{Implies}(
\text{Antecedent(Artist(I-variable(x)))}
\text{artistStyle(I-variable(x) I-variable(y))}
\text{Style(I-variable(y))}
\text{creator(I-variable(z) I-variable(x)))}
\text{Consequent(style/period(I-variable(z) I-variable(y))))}
\]

Providing inference services with SWRL is hard, as when supporting the full specification the reasoning becomes undecidable. There are different approaches available:

- OWL DL and SWRL can be translated to First Order Logic and reasoning tasks can be handled with a FOL theorem prover. Hoolet\(^4\) is an experimental implementation for this approach.
- Expanding an OWL DL reasoner based on the tableaux algorithm to support a subset of SWRL. Lately Pellet (section B.3.1) supports the DL-safe subset of SWRL, also current implementations are far from efficient.
- Using a forward chaining reasoning engine like Bossam\(^5\), translating OWL DL axioms into rules. This approach restricts expressivity due to many incompatibilities between description logics and Horn rules.

B.2.4 WRL

OWL and WRL (Web Rule Language) differ in that the former is a DL-based ontology language, whereas the latter is a rule-based ontology language which is independent from any logical language paradigm. Similarly to OWL, WRL consists of three dialects—WRL-core, -light, and -full—which are progressively more expressive.

WRL can express subset hierarchies between concepts (classes in OWL) and relations (properties or roles in OWL). For example:

\[
\text{concept Duke subConceptOf\{Aristocrat\}},
\]

expresses that all dukes are aristocrat, where concept and subConceptOf are WRL keywords. As for relations, expressing that ‘distance’ is a 3-place relation between two cities that yields a numerical value and that is a sub-relation of ‘measurement’, amounts to:

\[
\text{distance (ofType city, ofType city, impliesType _decimal)
subRelationOf measurement.}
\]

\(^4\)Hoolet, http://owl.man.ac.uk/hoolet/
\(^5\)Bossam Rule/OWL Reasoner, http://mknows.etri.re.kr/bossam
The arity of a relation can be specified by writing, e.g., `relation distance/3`. In WRL-core the arity is limited to 2.

Instances or individuals can be said to belong to a concept and/or occur in some relations. For example,

```
instance Beatrix memberOf{DutchQueen}
    hasBirthdate hasValue _date{1938, 1, 31}
    hasSon hasValue{Willem, Friso, Constantijn}
```

expresses that Beatrix is a Dutch queen who was born on January 31, 1983, and Willem, Friso, and Constantijn are her sons. WRL keywords, in this case, are `instance`, `memberOf` and `hasValue`.

Thus far, no significant difference in expressiveness from OWL can be noticed. In fact, the difference lies in the ability of WRL to handle variables and rules in full generality, as well as conjunction, negation and disjunction of formulas. Let $\alpha, \beta$ be WRL well-formed atomic formulas as the the ones above. Then,

$\alpha$ and $\beta$

$\alpha$ or $\beta$

$naf \alpha$

$\alpha :- \beta$

are well-formed formulas as well. The connectives `and`, `or` are interpreted as in FOL, `naf` stands for negation as failure, and `:-` connects the head and body of a rule. Similarly to FOL, atomic formulas and their negations are referred to as literals. The precise semantics of the connectives is that of DATALOG (see []).

Not every rule is allowed in WRL, however. A safety condition—i.e., every variable occurring in the body must occur in a non built-in literal in the head of a rule—applies to rules in WRL-core and -flight, but not to rules in WRL-full. Variables are referred to as e.g. `? myvariable` or `? x`, where `?` is the WRL keyword for variables.

WRL was designed to overcome the restricted expressive power of OWL. In terms of interoperability, OWL and WRL (after proper translation) share a common language subset, identifiable with WRL-core or OWL-DL. It is noteworthy that the document containing the WRL specifications is still under revision by W3C consortium (see [Angele et al., 2005]).

### B.3 Reasoners

Now we present some inference engines for the aforementioned knowledge representations. Pellet, Racer and KAON2 are primarily supporting the description logic behind OWL DL, whereas the SweetRules tool suite includes a rule-based inference engine. Also the OWL DL reasoners try to open in the direction of rule languages with some support for rule extensions of OWL.

#### B.3.1 Pellet and Swoop+Rules

Pellet is an open-source OWL DL reasoner written in Java, originally developed at the University of Maryland’s Mindswap Lab. Pellet is based on the tableaux algorithms developed for expressive Description Logics. It supports the full expressivity OWL DL including

For a discussion on this, see 3.3.1.

reasoning about nominals (enumerated classes). In addition to OWL DL, as of version 1.4, Pellet supports all the features proposed in OWL 1.1, with the exception of n-ary datatypes.

Pellet offers the following reasoning services, traditionally provided by DL reasoners:

**Consistency checking** Ensures that an ontology does not contain any contradictory facts, following the OWL semantics specification [Patel-Schneider et al., 2003].

**Concept satisfiability** Checks if it is possible for a class to have any instances. If a class is unsatisfiable, then defining an instance of that class will cause the whole ontology to be inconsistent.

**Classification** Computes the subclass relations between every named class to create the complete class hierarchy.

**Realization** Finds the most specific classes that an individual belongs to; in other words, computes the direct types for each of the individuals.

A handy feature from Pellet is the ability to not only detect OWL dialect, but assists in converting an OWL Full ontology to OWL DL. Many authors have to deal with the situation that some definitions are missing from the ontology to validate as OWL DL. Pellet incorporates heuristics to automatically derive such definitions.

Besides OWL DL Pellet supports different extensions in expressive power, especially for rules.

Pellet has support for **AL-log**, an extension of OWL DL with Datalog via coupling with a Datalog reasoner. It incorporates the traditional algorithm and a new pre-compilation technique that is incomplete but more efficient. The key idea of this implementation is to pre-process all of the DL atoms that appear in the Datalog rules and include them as facts in the Datalog subsystem. Once the preprocessing is done, queries can be answered by the Datalog component using any of the known techniques for Datalog query evaluation.

Pellet also has an experimental implementation of a direct tableau algorithm for the **SWRL DL-safe rules** extension to OWL DL [Kolovski et al., 2006]. A SWRL rule is called DL-safe if each variable in it occurs in a non-DL-atom in the rule body. This means rules can only introduce relations on individuals. A program is DL-safe iff all of its rules are DL-safe. The DL-safe subset of SWRL is decidable.

The Pellet implementation for DL-safe rules is still experimental and not efficient, but the developers think the approach is practical for mid-sized ontologies.

The developers of Pellet implemented a hybrid system for combining OWL and rules: **SweetRules** is based on their Web ontology browsing and editing tool, SWOOP. SweetRules handles ontologies with rules in DLP, AL-log and DL-safe representations. The new services compared to Pellet and SWOOP include dialect verification, rules editing and query answering for AL-log.

### B.3.2 Racer

The RACER System\(^8\) started as a scientific project at Hamburg University of Technology developed by Ralf Mller and Volker Haarslev. Since summer of 2005 the software renamed to RacerPro is developed, maintained and supported as a commercial application by Racer Systems GmbH, a Hamburg, Germany based company.

RACER was developed as a description logics reasoning engine even before the OWL standard emerged. The researchers had the same goal as with OWL DL: to develop a tractable reasoning engine for a decidable DL with as much expressive power as possible, RACER turned out to become an OWL DL reasoning engine.

---

\(^8\)RACER, Renamed ABox and Concept Expression Reasoner, http://www.racer-systems.com/
The RacerPro system implements a highly optimized tableau calculus for the description logic $\text{ALCQHI}^R_{\neg}$ (Liebig, 2006). In concrete, RacerPro can reason about OWL Lite knowledge bases, as well as OWL DL with approximations for nominals, together with some algebraic reasoning beyond the scope of OWL. Nominals in class definitions are approximated in a way which provides sound but incomplete reasoning. It offers reasoning services for multiple T-boxes and for multiple A-boxes, and implements the DIG standard for interfacing with applications using DL systems.

RacerPro can answer usual DL queries like concept consistency and subsumption, or instance retrieval. The reasoning engine provides its own semantically well-defined query language nRQL, new Racer Query Language, which also supports specific services, e.g. negation as failure or numeric constraints w.r.t. attribute values of different individuals.

The current version of RacerPro supports OWL DL almost completely, the system has two limitations. First, individuals in class expressions (nominals) are only approximated. Second, currently, RacerPro cannot process user-defined datatype types given as external XML Schema specifications (although all required datatypes of OWL DL are properly supported).

Being a commercial product RacerPro is well documented, and also an application with graphical user interface called RacerPorter is distributed with the system.

**B.3.3 KAON2**

KAON1 (formerly referred to as KAON) is an ontology language developed by the University of Karlsruhe and the Research Center for Information Technologies in Karlsruhe. The language is a somewhat restricted and also extended version of RDFS. The main goal of KAON1 was to provide simple reasoning facilities for large knowledge bases, mainly in industrial environments. This version is discontinued.

KAON2\(^9\) is a joint effort of the mentioned institutions with the University of Manchester, a completely new infrastructure for managing OWL DL, SWRL, and F-Logic ontologies.

Main features include an API for the management of the mentioned ontology languages and an inference engine. Queries expressed using SPARQL can be answered and the DIG interface is also supported allowing integration with tools such as Protegé.

The KAON2 DL reasoner is not based on the tableaux algorithm, as most of the other engines. Rather, reasoning in KAON2 is implemented by novel algorithms which reduce a SHIQ(D) knowledge base to a disjunctive datalog program. KAON2 supports the so-called DL-safe subset of the Semantic Web Rule Language. SHIQ(D) is a subset of OWL DL, including all features apart from enumerated classes ($\text{owl:oneOf}$ class and $\text{owl:hasValue}$ restriction). Another limitation of the reasoning engine is a practical problems with large cardinality statements. Even a maximum cardinality restriction of two may slow down the algorithm in special cases, however this is an open issue, and later may be fixed.

**B.3.4 SweetRules**

SweetRules\(^10\) is a powerful set of tools for semantic web rules and ontologies, revolving around the RuleML standard. SweetRules also supports SWRL and OWL in some extent. The tool is part of SWEET (“Semantic WEb Enabling Technology”), an overall set of tools that Benjamin Grosof’s group (with collaborators) has been developing since 2001. Other components in it include the SweetDeal e-contracting system approach and prototype, and the SweetPH system for business process ontologies drawn from the Process Handbook.

---

\(^9\)http://kaon2.semanticweb.org/  
SweetRules supports the Situated Courteous Logic Programs (SCLP) extension of RuleML, including prioritized conflict handling and procedural attachments for actions and tests [Grosof, 2004]. SweetRules can perform semantics-preserving translation and interoperability between a variety of rule and ontology languages, including XSB Prolog, Jess production rules, HP Jena-2, and IBM CommonRules. Inferencing services are also available through forward-direction inferencing and for backward-direction answering, i.e., query-answering.

The SweetRules toolset consists of three kinds of components: translators, inferencing engines, and front-ends. Translators translate between different rule languages and systems, using RuleML as an interchange format. The supported rule languages include: IBM CommonRules, XSB Prolog logic programming system, Jess production rules system, Knowledge Interchange Format (KIF) language. All translators are bidirectional.

RuleML inferencing is performed indirectly. Sweetrules translates the given rulebase to the rule language of another rule system, Jess or CommonRules, for example. The results of inferencing is translated back to RuleML.

SweetRules is implemented in Java and XSLT. In particular, the translator from RuleML to IBM CommonRules is implemented in XSLT. SweetRules makes use of several components from CommonRules, including its Courteous Compiler and its translators to XSB and KIF.

B.3.5 Comparison

The OWL reasoners we survey in this section are Pellet, RacerPro, KAON2 and FACT++. All of them offer standard reasoning services, such as consistency checking, concept satisfiability, calculation of implicit sub-classes relationships, instance checking and instance enumeration.

All reasoners support the DL language $\mathcal{SHIQ(D)}$, that is, OWL DL without nominals (namely, classes that are constructed by enumeration of the individuals belonging to them). Pellet and FACT++, in addition, supports nominals, and hence they support full OWL DL, or the equivalent DL $\mathcal{SHOIQ(D)}$ (notice the extra O); the support of nominals is also one of the upcoming features of RacerPro. The latest stable release of Pellet (November 7, 2006) and FACT++ (December 6, 2006) can also support OWL 1.1 or the equivalent DL language $\mathcal{SROIQ(D)}$ (see []).

As far as we can tell, FACT++ does not offer any additional reasoning service. The other reasoner, instead, can be differentiated with respect to query languages or additional reasoning services they support. Pellet, RacerPro and KAON2 support a query language, which is SPAQL in Pellet and KAON2, and the proprietary query language nRQL in RacerPro. A detailed presentation of these query languages can found in [Eric Prud’hommeaux, 2006] and [Haarslev et al., 2004].

In addition, Pellet and KAON2 support rules. Pellet (version 1.3) supports the generalized rule language $\mathcal{AL-log}$, as presented in [Donini et al., 1998]. The expressive restriction of $\mathcal{AL-log}$ is that concepts, which are not defined in the ontology written in OWL DL, cannot occur in the antecedent (head) of rules. Pellet generalizes $\mathcal{AL-log}$ by using $\mathcal{SHOIN(D)}$, instead of the minimal DL language $\mathcal{ALC}$. KAON2 support an even larger rule-language, i.e., the so-called DL-safe subset of SWRL (see Chapter 3), which is currently the largest subset of SWRL known to be decidable. Note that the experimental version 1.4. of Pellet also supports the same DL-safe subset of SWRL.

There are other features that are peculiar of Pellet only. It includes an implementation of the $\mathcal{ALCK}$ language that extends $\mathcal{ALC}$ with an epistemic operator $K$. The operator $K$ can be used in queries, e.g., $?(a : KC)$ meaning ‘is the individual $a$ known to be in the class $C$?’ Another use of the operators $K$ is within terminological axioms, such as $KC_1 \sqsubseteq C_2$, where $C_1, C_2$ are OWL DL concepts.

The most commonly used reasoning algorithm is based on the DL tableaux calculus (see [Baader and Sattler, 2001]). By contrast, KAON2 uses a rather non-standard reasoning
algorithm that reduces any $SHIQ(D)$ knowledge base to a disjunctive datalog program.

Finally, each reasoner is distributed under a different license: Pellet and FACT++ are distributed under the GNU public license and they have been developed by the University of Maryland (USA) and Manchester (UK). Their open source license makes them particularly appealing. Racer is a commercial product distributed by Racer System (Germany) and it may cost, depending on the license, up to 9,990.00 Euro. KAON2 is a pay license closed source and it has been developed by the University of Manchester (UK) AIFB – Universität Karlsruhe (Germany) and Forschungszentrum Informatik (Germany).

## B.4 Editors

For the visualization and maintenance of knowledge bases we would introduce Protégé, an advanced ontology editor providing full support for OWL DL.

### B.4.1 Protégé

Protégé\(^\text{11}\) is a free, open source suite of tools to construct domain models and knowledge-based applications with ontologies, developed at Stanford Medical Informatics. Although the development of Protégé has historically been mainly driven by biomedical applications, the system is domain-independent and has been successfully used for many other application areas as well.

Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. The Protégé platform supports two main ways of modeling ontologies: via the Protégé-Frames and Protégé-OWL editors. Protégé can be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications.

The Protégé-Frames editor enables users to build and populate ontologies that are frame-based, in accordance with the Open Knowledge Base Connectivity protocol (OKBC). The Protégé-OWL editor is Protégé extended by the OWL Plugin, enables users to build ontologies in the W3C’s Web Ontology Language (OWL).

Protégé’s model is based on a simple yet flexible metamodel, which is comparable to object-oriented and frame-based systems [Knublauch et al., 2004a]. The OWL Plugin extends the Protégé model and its API with classes to represent the OWL specification. The OWL Plugin supports RDF(S), OWL Lite, OWL DL (except for anonymous global class axioms, which need to be given a name by the user) and significant parts of OWL Full (including metaclasses).

In order to be able to reuse as much of the existing Protégé features as possible, a careful mapping between the Protégé metamodel and OWL had to be created that maintains the

\(^{11}\)Protégé, http://protege.stanford.edu/
traditional Protégé semantics where possible. OWL is an extension of RDF Schema, which extends RDF with metamodel classes and properties which can be mapped into the Protégé metamodel. As a result, the extensions that OWL adds to RDF(S) can be reflected by extensions of the Protégé metamodel.

Valuable features and generic services provided by Protégé include an event mechanism, undo capabilities, and a plugin system. The platform also supports a client-server-based multi-user mode that allows multiple people to edit the same ontology at the same time. Protégé also provides a highly scalable database back-end, allowing users to create ontologies with hundreds of thousands of classes. The Protégé community has contributed with lots of plugins, most of those can be also used for OWL directly or after adaptation.

Protégé has an advanced graphical user interface (GUI) for the visualization and manipulation of ontologies. The OWL Plugin preserves these features of the GUI and also extends it with OWL specific features such as a comfortable expression editor which allows users to quickly assemble DL expressions when defining classes.

The OWL Plugin provides direct access to DL reasoners through the DIG interface [Knublauch et al., 2004b]. Supported reasoning tasks are consistency checking, classification and realization. The reasoner is integrated into the user interface on a high level. For example the OWL Plugin displays both the asserted (stored in the OWL files) and inferred (result of classification) hierarchies of classes and makes available extensive information on the inferences made during classification. These features help to maintain ontologies, as the semantically correct position of the classes can be easily determined.

B.5 Evaluation

In this section we have described two types of knowledge representation languages: description logics and rule languages. Most of the languages are proposals for the Semantic Web.

Description logics are a formally well-understood way of represent the terminological knowledge of an application domain. OWL DL is a strong candidate for the Semantic Web with the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability.

Rule languages have a completely different approach, as they provide flexible control of the reasoning process.

The candidates integrating rules with description logics, DLP and SWRL are of particular interest, also the two take a diametrically opposed integration approach. DLP is an intersection of Horn logic and OWL DL, whereas SWRL is the union of them. As a result DLP has a very limited expressive power, while SWRL has at the price of decidability and practical implementations.

None of the mentioned knowledge representations can fulfill all requirements of ESTRELLA, a possible solution may integrate them some way, or propose a parallel use based on the actual problem to be solved.
Appendix C

LKIF Rules and OWL

In the case of normal SWRL rules used in combination with OWL DL, a conclusion follows and becomes an OWL DL axiom if the premises were entailed by the OWL DL model.

LKIF rules are not standard SWRL rules, as has become clear in the previous section. Argumentation schemes do not fire automatically because their premises are met by the OWL model. Instead the OWL model serves as a very weak constraint on what arguments can be made.

The problem of defeasibility is solved by presuming that in argumentation it is possible to argue about substantially the same entity, without those entities being sameAs in OWL terms. The standard for establishing that two sentences \( s_1 \) and \( s_2 \) are equivalent is different in argumentation, as follows:

**Definition 40 (Co-referents).** Is \( c_1 \) is a constant and \( M_{\text{OWL}} \vDash \text{coreferent}(c_1, c_2) \), then \( c_2 \) is the same constant in an argumentation structure.

The property \( \text{coreferent} \) is symmetric. An argumentation scheme can be instantiated from a rule when the following conditions on the premises and argument conclusions are met:

**Definition 41 (OWL Truth condition of premises and argument conclusions).** That determine whether an LKIF Rule is applicable:

1. If \( \varphi \) is a premise, then \( M_{\text{OWL}} \not\vDash \neg \varphi \)
2. If (not \( \varphi \)) is a premise, then \( M_{\text{OWL}} \not\vDash \varphi \)
3. If (unless \( \varphi \)) is a premise, then \( M_{\text{OWL}} \not\vDash \varphi \)
4. If (assuming \( \varphi \)) is a premise, then \( M_{\text{OWL}} \not\vDash \neg \varphi \)
5. If the direction of argument conclusion is pro \( \varphi \), then \( M_{\text{OWL}} \vDash \neg \varphi \)
6. If the direction of argument conclusion is con \( \varphi \), then \( M_{\text{OWL}} \not\vDash \varphi \)

When an argumentation scheme is instantiated it is asserted into the OWL model, as follows:

**Definition 42 (OWL interpretation of argument instantiation).** The following is the OWL interpretation of the instantiation of an argument scheme:

1. If \( \varphi \) is a premise, then \( M_{\text{OWL}} \vDash \varphi \)
2. If (not \( \varphi \)) is a premise, then \( M_{\text{OWL}} \vDash \neg \varphi \)
3. If (unless \( \varphi \)) is a premise, then \( M_{\text{OWL}} \vDash \neg \varphi \)
4. If (assuming \( \varphi \)) is a premise, then \( M_{\text{OWL}} \vDash \varphi \)
5. If the direction of argument conclusion is pro $\varphi$, then $\mathcal{M}_{\text{OWL}} \models \varphi$

6. If the direction of argument conclusion is con $\varphi$, then $\mathcal{M}_{\text{OWL}} \models \neg \varphi$
Appendix D

Argument Structure Graphs

Figure D.1 presents a first, simple example of the CARNEADES method. It is a reconstruction of the ‘Nixon diamond’ benchmark from the Artificial Intelligence field of non-monotonic reasoning.

Figure D.1: Nixon Diamond

Recall that argument graphs are bipartite, consisting of statement nodes and argument nodes. As shown in the figure, statements are diagrammed using rectangular boxes; arguments using rounded boxes. The label of statement nodes shows the content of the statement (or a concise identifier).

In diagrams, various kinds of lines are used to link statements to an argument. A line with a closed or open arrowhead from the argument to its conclusion is used to indicate a pro or con argument, respectively. In the Nixon example, argument a1 is a pro argument and argument a2 is a con argument. Recall that premises are a binary relation between statements and arguments. Thus premises are visualized as lines linking statements to arguments. The kind of premise is indicated by the line’s arrowhead. An ordinary premise has no arrowhead. Assumptions are indicated with a closed, circular arrowhead; exceptions with an open, circular arrowhead. Negative premises are displayed with a short, perpendicular line breaking the line linking the premise to the argument.

Solid and dashed lines are used to distinguish defensible from indefensible arguments and premises which hold from premises which do not hold. The role of each premise in the argumentation scheme applied is shown as a label on the line for the premise. In the example, the minor and major premises are clearly marked in this way.

Figure D.2 presents a second example, a reconstruction of Toulmin’s leading example,
about whether or not Harry is a British citizen, illustrating some further conventions.

---

Figures D.2, D.3, and D.4 present reconstructions of Toulmin diagrams for evidential reasoning. These diagrams illustrate the structure of arguments and the interplay between evidence, conclusions, and warrants. The reconstructions are based on different cases and demonstrate how Toulmin's framework can be applied in legal reasoning.

---

**Figure D.2:** Reconstruction of Toulmin Diagrams in Carneades

A more extensive example of the diagrams: figure D.3 presents a reconstruction of Wigmore’s chart of the evidence in the case of Commonwealth v. Umilian (1901, Supreme Judicial Court of Massachusetts, 177 Mass. 582), in the version presented by Bex, et al. [Bex et al., 2003].

Figure D.4 shows a reconstruction of this Wigmore chart with this method, using the diagramming method proposed here.

A more detailed description of this diagramming method is forthcoming.
Figure D.3: Wigmare Chart for the Umilian Case

Figure D.4: Carneades Diagram for the Umilian Case


BIBLIOGRAPHY


exceptions, 15
A-Box, 46
action schema, 34
actions, 16
adjudication, 6
administration, 7
advising, 6
advocacy, 6
affirmation of a permission, 24
affirmation of an imperative, 24
AL-log, 112
applicability, 15
argument structures, 75
argumentation, 7
arguments, 16
assertive knowledge base, 56
authority, 15
axiomatic triples, 43
backing, 15
belief revision, 25
Common Logic Interchange Format, 71
complexity, 101
concept name, 82
conflict, 16
conflict of compliance, 23
datatype term, 55
Datatyping, 102
decidability, 39
defeasible generalizations, 89
disaffirmation of a permission, 22
disaffirmation of an imperative, 22
drafting, 6
epistemic entrenchment, 25
evidence, 16
exception, 33
exceptions, 26
exclusionary, 15
explicit disaffirmation, 23
extensional predicates, 107
general safety, 58
goals, 16
Ill-defined layering, 102
intensional predicates, 107
interchangeability, 39
joint realization, 23
jointly realizable, 21
justifying, 6
lex posterior, 88
lex posterior derogat legi priori, 26
lex Specialis derogat legi generali, 26
Lex Superior, 26
lex superior derogat legi inferiori, 26
license, 115
limits of OWL DL, 50
Lisp-style syntax, 67
LKIF, 67
LKIF rules, 71
Mechanical jurisprudence, 6
object term, 55
ontology satisfaction, consistency and entailment, 49
OWL atom, 56
OWL axioms, 46
OWL classes, 45
OWL facts, 46
OWL logical vocabulary, 45
OWL model, 47
OWL semantics, 46
OWL syntax, 45
OWL truth, 49
OWL Vocabulary, 45
planning, 6
preference independence, 81
pretty-printing syntax, 69
priority relation, 15
program, 56
public body, 35
RDF Graph, 67
RDF graph, 41
RDF ground graph, 42
RDF ground triple, 42
RDF model, 42
RDF name, 41
RDF specific vocabulary, 43
RDF triple, 41
RDF triples, 67
RDF truth, 42
RDF vocabulary, 41
RDFS, 43
RDFS semantics, 44
realizability, 21
realized, 21
reasoning algorithm, 114
reasoning services, 114
reification, 43
reified objects, 87
rules, 39, 114

s-expression syntax, 71
safe-rules, 58
semantics, 53
statement, 16
strong safety, 59
strongly safe subset of SWRL, 60
SweetRules, 112
SWRL atoms, 55
SWRL DL-safe rules, 112
SWRL formulas, 55
SWRL logical vocabulary, 55
SWRL model, 55
SWRL rules, 55
SWRL truth, 56
SWRL vocabulary, 55
symmetric disaffirmation, 27
syntactic characterization, 52

T-Box, 46
terminological knowledge base, 56
tractability, 102
transactions, 35

unknown individual problem, 61

validity, 15

weak safety, 59