IST-2004-027655

ESTRELLA

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

Specific Targeted Research Project
Information Society Technologies

Deliverable 4.1

The Legal Knowledge Interchange Format (LKIF)

Due date of deliverable:	15 September 2008
Actual submission date:	15 September 2008

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

Fraunhofer FOKUS

FINAL

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)

Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Executive Summary

This reports specifies the final version of the Legal Knowledge Interchange Format (LKIF) developed in the European ESTRELLA project.

LKIF is an XML Schema for representing theories and arguments (proofs) constructed from theories. A theory in LKIF consists of a set of axioms and defeasible inference rules. The language of individuals, predicate and function symbols used by the theory can be imported from an ontology represented in the Web Ontology Language (OWL). Importing an ontology also imports the axioms of the ontology. All symbols are represented using Universal Resource Identifiers (URIs). Other LKIF files may also be imported, enabling complex theories to modularized.

Axioms are named formulas of full first-order logic. The heads and bodies of inference rules are sequences of first-order formulas. All the usual logical operators are supported and may be arbitrarily embedded: disjunction ($\wedge$), conjunction ($\vee$), negation ($\neg$), material implication ($\rightarrow$) and the biconditional ($\leftrightarrow$). Both existential ($\exists$) and universal ($\forall$) quantifiers are supported. Free variables in inference rules represent schema variables.

Terms in formulas may be atomic values or compound expressions. Values are represented using XML Scheme Definition (XSD) datatypes. Atomic formulas are reified and can be used as terms, allowing some meta-level propositions to be expressed.

The schema for atomic formulas has been designed to allow theories to be displayed and printed in plain, natural language, using Cascaded Style Sheets (CSS). An atomic proposition may first be represented in propositional logic, using natural language, and later enriched to become a first-order model, by marking up the variables and constants of the proposition and specifying its predicate using an XML attribute. This feature of LKIF is essential for enabling domain experts, not just computer specialists, to write and validate theories.

Support for allocating the burden of proof when constructing arguments from theories in dialogues is provided. An *assumable* attribute is provided for atomic formulas, to indicate they may be assumed true until they are challenged or questioned. An *exception* attribute is provided for negated formulas, to indicate that $\neg P$ may be presumed true *unless* P has been proven. Making a negated formula, $\not P$, an exception shifts the burden of proof for P to the opponent of P in the dialogue. Unlike negation-as-failure in logic programming, exceptions do not depend on the closed-world assumption.

Arguments in LKIF link a sequence of premises to a conclusion, where both the premises and the conclusion are atomic formulas. Attributes are provided for stating the direction of the argument (pro or con), the argumentation scheme applied and the role of each premise in an argument. Arguments can be linked together to form argument graphs. The legal proof standard each proposition at issue must satisfy, such as "preponderance of the evidence" or "beyond reasonable doubt" may be specified. Attributes are provided for recording the relative weight assigned to each argument by the finder of fact, such as the jury, or some other audience, as well as the status of each issue in the proceeding.

All of the main elements of an LKIF file may be assigned Universal Resource Identifiers, allowing them to be referenced in other documents, anywhere on the World Wide Web. Cross references between elements of legal source documents and the elements of the LKIF document which model these sources may be included within the LKIF file, using a sequence of *source* elements. The scheme allows m to n relationships between legal sources and elements of the LKIF model to be represented.

LKIF builds on and uses many existing World Wide Web standards, including the XML, Universal Resource Identifiers, XML Namespaces, the Resource Description Framework (RDF) and the Web Ontology Language (OWL). However, for a variety of reasons it does not use other XML schemas for modeling legal rules, such as Common Logic, RuleML, the Semantic Web Rule Language (SWRL), or the Rule Interchange Format. Common Logic is an ISO standard for representing formulas of first-order classical logic. While LKIF includes a sublanguage for first-order logic, LKIF has been designed to allow formulas of first-order logic to be represented in human readable form in natural language, to ease development, maintenance and validation by domain experts. Moreover, the ISO Common Logic standard does not look like it will be widely adopted within the World Wide Web community, which has its own standards body, the World Wide Web Consortium. RuleML, SWRL and RIF, among other efforts, are competing to become the Web standard for rules. At the beginning of the ESTRELLA project, SWRL was the leading candidate. In the meantime, during the development of LKIF in Estrella, RIF has become the leading contender. But neither SWRL nor RIF are currently expressive enough for the legal domain. Legal rules can best be understood as domain-dependent defeasible inference rules. They cannot be adequately modeled as material implications in first-order logic. However, an LKIF theory can in principal import a first-order theory represented in any XML format, to be used as part of the axioms of the theory. This feature of LKIF enables a part of the legal theory to be represented in first-order logic, using whatever format eventually becomes the World Wide Web standard.

This final version of LKIF has changed substantially since the first version, published in ESTRELLA Deliverable D1.2 in January, 2007. LKIF is no longer defined as an OWL ontology, but as an XML schema, since LKIF's main purpose is to serve as an interchange format which is easy to translate to and from other formats. LKIF has been designed to make this task easier, for example by writing translators in the Extensible Style Sheet Language Transformations (XSLT) language. OWL documents have many serializations. LKIF serves as a kind of standard, canonical serialization. (OWL is still used for the LKIF basic ontology of legal concepts and is the LKIF standard for defining ontologies of legal domains.) The Lisp s-expression syntax for rules is still supported by the LKIF reference inference engine, but is no longer a normative part of the LKIF specification. Vendors do not need to support the s-expression syntax to be LKIF compliant. Finally, whereas Deliverable D1.3 was a lengthy report with an extensive section on requirements and a survey of existing knowledge representation languages and technologies, this report aims to be a concise specification of LKIF.

The Legal Knowledge Interchange Format (LKIF)

ESTRELLA Deliverable D4.1

Thomas F. Gordon
Fraunhofer FOKUS, Berlin

Contents

ESTRELLA
p/a Faculty of Law
University of Amsterdam
PO Box 1030
1000BA Amsterdam
The Netherlands

tel: +31 20 525 3485/3490
fax: +31 20 525 3495
`http://www.estrellaproject.org`

Corresponding author:
Thomas F. Gordon
`thomas.gordon@fokus.fraunhofer.de`

Contents

Chapter 1

Overview

The Legal Knowledge Interchange Format (LKIF) is an XML schema for representing axiomatizations of theories in formal logic and proofs of propositions constructed from these theories.

As its name suggests, LKIF has been developed primarily for applications in the legal domain. Its development was driven by the requirements of applications of legal knowledge systems, from vendors of legal knowledge systems, public administrators and experts in the field of artificial intelligence and law. These legal requirements distinguish LKIF from related efforts to develop XML formats for rules, such as RuleML, the Semantic Web Rule Language (SWRL) or the Rule Interchange Format (RIF). Whereas these other rule languages are intended for representing production rules or first-order logic, moreover usually some subset of first-order logic whose proof theory has interesting computational properties, LKIF is intended to model legal rules of the kind found in legislation and regulations. Legal rules cannot be modeled adequately as either production rules or as material implications in first-order logic without losing the structure of the rules of the legislation and abstracting away its allocation of the burden of proof. While there are several interpretations of legal rules in legal theory, in LKIF legal rules are modeled as *defeasible inference rules*, since legal rules do not contrapose, i.e. may not be used backwards, and the conclusion of an applicable legal rule is only presumptively true, not necessarily true. When several rules are applicable, conflicts among the rules are resolved by reasoning about rule priorities, using meta-level legal rules such as *lex superior* (prefer the rule from higher authority) and *lex posterior* (prefer the newer rule). Also, there are several ways to undercut the applicability of legal rules, such as by showing that the rule has been repealed and is no longer valid law.

In this report, Relax NG's compact, human readable syntax [Clark, 2003] will be used to present the grammar of the LKIF schema. The grammar can be automatically converted into an Extensible Schema Definition (XSD).

The top-level structure of an LKIF file has this form:

```
element lkif {
    attribute version { xsd:string }?,
    Sources?, Theory?, ArgumentGraphs?
}
```

The version attribute is used to provide the version number of the LKIF schema used by the model, not the version number of the model. LKIF, as an interchange format, is intended to be translated to and from other formats, such as the proprietary formats of vendors. The version attribute enables translators to evolve to support future versions of LKIF while remaining backwards compatible, to continue to support models represented using prior LKIF versions.

The three top elements of an LKIF file are all optional. The Sources element is for linking elements of an LKIF file to the legal source texts modeled by the element. The Theory element provides a way to model the axioms and inferences rules of a legal theory. Finally, the ArgumentGraphs element provides a way to model one or more sets of interlinked arguments constructed from the theory. An ArgumentGraph models a legal proof of a proposition at issue, whereby 'proof', we mean proof in the legal sense, not in the sense of deductive proofs in classical logic. The proposition proved need not be necessarily true. It is sufficient that arguments pro and con the proposition satisfy the applicable legal proof standard, such as "preponderance of the evidence", in civil cases, or "beyond reasonable doubt", in criminal cases.

The rest of this report is organized as follows. In Chapter 2, the LKIF format for formulas in propositional and predicate logic is presented. The grammar of formulas is presented first, since formulas are used in both theories and argument graphs. Chapter 3 presents the LKIF format for defeasible inference rules, intended mainly for modeling legal rules. Chapter 4 presents the LKIF grammar for axiomatizations of theories, consisting of a axioms and a set of defeasible inference rules for deriving conclusions from the axioms. Chapter 5 presents the LKIF formats for arguments and argument graphs, which are used to represent proofs. Finally, Chapter 6 describes how to link elements of an LKIF model to the source documents being modeled.

Chapter 2

Formulas

LKIF can represent both propositional and first-order logic formulas. All of first-order logic is supported, with extensions for representing meta-leval propositions about atomic propositions and annotating formulas with information needed for allocating the burden of proof. The schema for atomic formulas enables formulas to be entered and displayed in readable, semi-natural language, using Cascaded Style Sheets (CSS). This feature of LKIF is designed to make it easier for legal experts, with little specialized training in computer technology, to construct, maintain and validate formal models of the law.

2.1 Terms

A logical Term is a variable, individual, constant, expression or an atom.

```
Term = Variable | Individual | Constant | Expression | Atom
```

Variables represent both schema variables in inference rules and logical variables in first-order formulas.

```
Variable = element v { xsd:Name }
```

The content of a variable is restricted to be a symbol complying with the lexical syntax of the XSD Name datatype.[1]. That is, the variable name must begin with a letter, underscore character or colon and may be followed by any number of alphanumeric charcters. White space is not allowed in variable name. Here are two examples of variables:

```
<v>x</x>
<v>buyer</v>
```

There are two ways to represent logical constants, using the c and i elements. The c element allows XSD symbols (names), URIs, strings, integers and floating point numbers to be used as constants. The i, mnemonic for 'individual', is an alternative way to represent constants using URIs which has the advantage of providing a way to describe the object denoted by the constant in natural language, to facilitate readability.

[1]http://www.w3.org/TR/xmlschema-2/

```
Constant = element c {
    xsd:Name | xsd:anyURI | xsd:string |
    xsd:boolean | xsd:integer | xsd:float
}

Individual = element i {
    attribute value { xsd:anyURI },
    text
}
```

Here are some examples of constants:

```
<c>John</c>
<c>http://www.estrellaproject.org</c>
<c>34.6</c>
<c>true</c>
<c>110</c>
<c>"this is a string"</c>
<i value="http://www.estrellaproject.org">The Estrella Website.</i>
```

Compound terms can be represented using expr (expression). The functor of a compound term is represented as a URI, using an attribute of the expression element. A compound term consists of zero or more subterms.

```
Expression = element expr {
    attribute functor { xsd:anyURI },
    Term*
}
```

Here is an example, showing how to represent the mathematical expression $\sqrt{x+y}$:

```
<expr functor="sqrt">
  <expr functor="sum"><v>x</v><v>y</v></expr>
</expr>
```

In this example, the URIs of the functors naming the square root and summing functions are local names. A dictionary of functors could be modeled using an OWL ontology, in which case the URI of the functor attribute of an expression should reference the functor in the ontology.

2.2 Atomic Formulas

Atomics formulas are represented using s elements, where ‘s’ is intended to be mnemonic for ‘statement’. The predicate symbol of an atomic formula is represented by a URI, using an attribute. The content model of the s element consists of zero or more terms interspersed with free text, allowing statements to be formulated in semi-natural language. This free text is intended to make statements easier for humans to read and understand, but has no formal meaning. It serves the same purpose as comments in computer programs. It is the responsibility of the modeler or "knowledge engineer" to use this free text in a meaningful and consistent way.

```
Atom = element s {
    attribute pred { xsd:anyURI }?,
    attribute assumable { xsd:boolean}?,  # default: false
    ((text | Term)*)
}
```

The assumable attribute of an atomic formula provides one way to annotate formulas with meta-level information about the burden of proof in legal proceedings. Depending on the applicable rules of legal procedure, an assumable formula may be assumed to be true without proof or evidence until a party has made an issue out of the statement using the appropriate procedure, motion or speech act. The assumable attribute has no affect on logical, model-theoretic semantics of formulas.

The syntax of atomic formulas enables statements to first be represented in natural language, formalized only at the level of propositional logic, and later enriched to a first-order predicate logic representation. For example, the statement "Joe is the parent of Sally" can first be represented in propositional logic as:

```
<s>Joe is the parent of Sally</s>
```

and later enriched to a first-order logic representation to become:

```
<s pred="family:parent">
  <c>Joe</c> is the parent of <c>Sally</c>
</s>
```

Here are two further examples of atomic formulas, illustrating the use of variables:

```
<s pred="family:parent">
  <v>Person1</v> is a parent of <v>Person2</v>
</s>

<s pred="family:obligatedToSupport">
  <v>Person1</v> is obligated to support <v>Person2</v>
</s>
```

Atomic formulas are reified in LKIF and may also be used as terms. This enables meta-level propositions about atomic propositions to be expressed, such as the following event calculus [Kowalski and Sergot, 1986] statement about a security interest s being perfected at time t:

```
<s pred="holds"><s pred="perfected"><v>s</v></s><v>t</v></s>
```

2.3 Compound Formulas

Compound formulas can be formed from atomic formulas using all the usual logical operators as well as universal and existential quantifiers. A well-formed formula (Wff) is an atomic formula or a compound formula constructed using one of these operators or quantifiers:

```
Wff = Atom | Or | And | Not | If | Iff | All | Exists
```

The logical disjunction $P \vee Q$ is represented using an or element.

```
Or = element or {
  attribute assumable { xsd:boolean }?,
  Wff, Wff+
}
```

The assumable attribute of all formulas, not just disjunctions, is optional. If the attribute is not supplied, its default value is false. An assumable formula may be added to the set of axioms of the theory under construction during a dialogue. Questioning this assumption would cause the formula to be removed from the set of axioms.

Here's the LKIF representation of $P \vee Q$:

```
<or>
  <s>P</s>
  <s>Q</s>
</or>
```

The logical conjunction $P \wedge Q$ is represented using a and element.

```
And = element and {
  attribute assumable { xsd:boolean }?,
  Wff, Wff+
}
```

Here's the LKIF representation of $P \wedge Q$:

```
<and>
  <s>P</s>
  <s>Q</s>
</and>
```

Notice that both or and and elements consists of two or more well-formed formulas, not just two. Thus, for example, $P \vee Q \vee R$ can be represented as:

```
<or>
  <s>P</s>
  <s>Q</s>
  <s>R</s>
</or>
```

Negated formulas, such as $\neg P$, are represented using not elements.

```
Not = element not {
  attribute exception { xsd:boolean }?,
  attribute assumable { xsd:boolean }?,
  Wff
}
```

Using this syntax, $\neg P$ is represented in LKIF as:

```
<not><s>P</s></not>
```

The semantics of negation, indeed for all well-formed formulas in LKIF, is classical. In particular, not does not mean "negation-as-failure", as used in logic programming languages such as Prolog [Clocksin and Mellish, 1981]. The exception attribute of a negated formula $\neg P$ can be used to assign the burden of proof for P to the party who opposes $\neg P$. An exception $\neg P$ is presumably true *unless* the opponent of $\neg P$ has proven P. The proof-theoretic effect is similar to negation-as-failure in logic programming, but without defining a nonmonotonic inference relation or making the closed-world assumption. The assumable and exception attributes of a negated formula are orthogonal. If $\neg P$ is assumable, then $\neg P$ may be added to the axioms of the theory under construction. Merely questioning the assumption is enough to cause the formula to be removed from the axioms, without having to prove P. If $\neg P$ is an exception, used as a premise in a defeasible inference rule $Q \Leftarrow \neg P$, then the burden of proving P is on the opponent of Q. That is, the rule may be used to conclude that Q is presumably true without first having to prove P. But the opponent of Q can later defeat the presumption by proving P.

Material implications, such as $P \rightarrow Q$ are represented in LKIF using if elements.

```
If = element if {
  attribute assumable { xsd:boolean }?,
  Wff, Wff
}
```

Here is the example $P \rightarrow Q$ using this syntax:

```
<if>
  <s>P</s>
  <s>Q</s>
</if>
```

Biconditionals, such as $P \leftrightarrow Q$, are represented in LKIF using iff elements.

```
Iff = element iff {
  attribute assumable { xsd:boolean }?,
  Wff, Wff
}
```

The assumable attribute is optional. If the attribute is not supplied, its default value is false.

Here's the $P \leftrightarrow Q$ example in LKIF:

```
<iff>
  <s>P</s>
  <s>Q</s>
</iff>
```

Finally, the universal and existential quantifiers, $\forall x.P(x)$ and $\exists x.P(x)$, respectively, are represented in LKIF using all and exists elements.

```
All = element all {
 attribute assumable { xsd:boolean }?,
 Variable+, Wff
}

Exists = element exists {
  attribute assumable { xsd:boolean }?,
  Variable+, Wff
}
```

For example:

```
<all>
   <v>x</v>
   <s pred="P"><v>x</v></s>
</all>

<exists>
   <v>x</v>
   <s pred="P"><v>x</v></s>
</exists>
```

Well-formed formulas Wffs in LKIF are just an XML concrete syntax for first-order predicate logic, extended with some attributes for annotating formulas with information about the burden of proof. These extensions have no effect on the first-order semantics of formulas. The concepts of logical consequence and contradiction have their usual, classical meaning without change.

Chapter 3

Rules

There are many kinds of rules and thus many kinds of rule languages. In logic, rules may be interpreted as material implications, i.e. as a kind of logical operator for constructing formulas, or as inference rules for deriving formulas from axioms. In computer science, production rules have been developed as a part of a programming paradigm in which rules are applied to data stored in working memory. Formal languages are defined using grammar rules. In law, rules provide one way to express norms, principals and regulations.

Since LKIF is designed as an XML format for interchanging models of legal knowledge, the kind of rules we are interested in modeling are legal rules. Legal rules express public policy, not only about how to act, but also about how to reason legally about situations and actions. 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.

Legal rules can be viewed as domain dependent inference rules, complementing and extending the universally applicable and generic inference rules of classical logic, such as modus pollens or modus tollens. Aside from being domain-dependent, another difference between legal rules and the inference rules of classical logic is that the conclusion of a legal rule need not be necessarily true. Typically, the conclusion of a legal rule is only *presumptively* true. The rules of legal procedure regulate the construction and evaluation of *arguments* from legal rules. It may be possible to construct conflicting arguments from several rules. The law provides ways to resolve these conflicts, using meta-leval principals, such as lex superior (prefer the rule from the higher authority), and by using procedural measures, such as allocating the burden of proof.

LKIF has been designed to be sufficiently expressive to support the *isomorphic modeling* [Bench-Capon and Coenen, 1992] of legislation, at a very high level, in order the facilitate the development, validation and maintenance of knowledge bases by legal experts, the rule language is more expressive than formulas of first-order logic, let alone subsets of first-order logic, such as Horn clause logic or description logic, which have been developed due to interesting computational properties, such as (semi-)decidability or even tractability. LKIF has been designed to optimize expressiveness for the legal domain, not computational efficiency. LKIF is designed for use in interactive systems which help users to construct theories and arguments, as well as traditional expert systems, which interactively acquire facts and deduce propositions from these facts by applying rules. For argument construction tasks, the expressivity of the knowledge representation language is more important than the computational properities of the inference relation, since users are reponsible for controlling the search for arguments. For traditional expert system, the computational properties of the inference relation may be more important, since the inference engine is expected to fully automatically derive logical conclusions from the facts input by the user. LKIF has been designed to be expressive enough for both kinds of systems, but when using LKIF for traditional expert systems, the knowledge engineers should take case to use a subset of LKIF which can be handled by inference engines with the desired computational properties.

From a computational perspective, the semantics of LKIF rules is based on the dialectical and argumentation-theoretic approach articulated by Ron Loui in “Process and Policy: Resource-Bounded Non-Demonstrative Reasoning” [Loui, 1998]. Essentially, legal rules are interpreted as policies for reasoning in resource-limited, decision-making processes. In argumentation theory such reasoning policies are viewed as inference rules for presumptive reasoning, called *argumentation schemes* [Walton, 2006]. Arguments are instances of argumentation schemes, constructed by substituting variables of a scheme with terms of the object language. An *argument graph* is a proof constructed from a set of arguments. A set of argumentation schemes defines a search space over argument graphs. Reasoning with argumentation schemes can be viewed as heuristic search in this space, looking for argument graphs in which some disputed claim is acceptable or not given the proof represented by the argument graph. In dialogues, the parties take turns searching this space, looking for counterarguments. Turn-taking, termination conditions, resource limitations and other procedural parameters are determined by the rules of the legal proceeding, i.e. by the argumentation protocol for the particular type of proceeding. See [Gordon, 2007a] for further details about the semantics of LKIF rules.

A rule in LKIF consists of an identifier, a head and an optional body, where both the head and the body, if there is one, consist of one or more well-formed formulas of first-order predicate logic.

```
Rule = element rule {
  attribute id { xsd:ID },
  attribute strict { xsd:boolean }?,
  Head, Body?
}

Head = element head { Wff+ }
Body = element body { Wff+ }
```

The strict attribute of rules provides a way to state, for instances of the rule, that the formulas in the head of an instance of the rule are necessarily true when the formulas in the body of the instance of the rule are true. Whether this is in fact the case, or whether instead the rule is subject to exceptions or capable of being overridden by other rules, is a modeling issue. It is the responsibility of author of the rule to model the rules of the legal domain correctly. The strict attribute is a meta-leval annotation which has no affect on the argumentation-theoretic semantics of rules but may be useful for heuristic purposes.

Notice that LKIF rules are much more expressive than Horn clauses of the kind used in logic programming languages such as Prolog. The body consists of arbitrary formulas of first-order logic, not just literals. The head also consists of arbitrary formulas of first-order logic, rather than, as in Horn clauses, a single positive literal. Negated formulas may appear in both the body and head of rules. Negation is interpreted classically, not as negation as failure. (However, the exception attribute of negated formulas can be used to allocate the burden of proof to the desired party.) Rule identifiers can be used as to refer to rules in other rules. The properties of rules, such as their date of enactment, may be expressed; and, using these properties, conflicts among rules can be resolved by using rules to reason about priorities among rules.

All of the extensions are useful for modeling law, particularly if one is interested, as we are, in modeling law in a way which preserves its organization and structure, called 'isomorphic modeling" in the AI and Law field [Bench-Capon and Coenen, 1992]. Legislation is typically organized as general rules with separately stated exceptions. Other rules, such as the legal principals of lex superior (prefer the rule from the higher authority) and lex posterior (prefer the later rule) are used to resolve conflicts among rules. Simple general rules are important, in order to make it feasible for persons to learn and remember the law. It is better to know the basic rules than to not be able to know any of the rules at all, due to their overwhelming complexity.

Here is first example of rule in LKIF, 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 id="Sect-9-105h">
  <head>
    <s pred="goods"><v>c</v> is goods</s>
  </head>
  <body>
    <s pred="movable"><v>c</v> is movable</s>
    <not exception="true">
      <s pred="money"><v>c</v> is money</s>
    </not>
  </body>
</rule>
```

Sect-9-105h is an XML name, which may be used as a term denoting the rule in other rules.

Legal rules are defeasible generalizations. Showing that some exception applies, such as the exception for money in the above example, is one way to defeat a rule, by *undercutting* [Pollock, 1987] it. A rule applies if its conditions are true, *unless* some exception applies. A party who wants to apply this rule need not show that the exception does not apply. The burden of producing evidence that the exception does apply is on the other party. In the example, the party which wants to prove that the object is not goods has the burden of proving that the object is money.

Another source of defeasibility is conflicting rules. Two rules conflict if one can be used to derive P and another $\neg P$. To resolve these conflicts, we need to be able to reason (i.e. argue) about which rule has priority. To support reasoning about rule priorities, the rule language includes a built-in predicate over rules, prior, where

```
<s pred="prior">
  <v>r1</v> has priority over <v>r2</v>
</s>
```

means that rule r1 has priority over rule r2. If two rules conflict, the arguments constructed using these rules are said to *rebut* each other, following Pollock [Pollock, 1987].

The priority relationship on rules is not defined by the system. Rather, priority is a substantive issue to be reasoned (argued) about just like any other issue. One way to construct arguments about rule priorities is to apply the argumentation scheme for arguments from legal rules to meta-level rules, i.e. rules about rules, using information about properties of rules, such as their legal authority or date of enactment. The reification of rules and the built-in priority predicate make this possible. In knowledge bases for particular legal domains, represented using this rule language, rules can be prioritized both *extensionally*, by asserting facts about which rules have priority over which other rules, and *intensionally*, using meta-rules about priorities.

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 in LKIF as:

```
<rule id="lex-posterior">
  <head>
    <s pred="prior"><v>r1</v> has priority over <v>r2</v></s>
  </head>
  <body>
    <s pred="enacted"><v>r1</v> was enacted on <v>d1</v></s>
    <s pred="enacted"><v>r2</v> was enacted on <v>d2</v></s>
    <s pred="later"><v>d1</v> is later than <v>d2</v></s>
   </body>
 </rule>
```

Rules can be defeated in two other ways: by challenging their validity or by showing that some exclusionary conditions apply. These are modeled with rules about validity and exclusion, using two further built-in predicates: valid(rule id) and excluded(rule id, atom).

The valid and excluded relations, like the prior relation, are to be defined in domain models. Rules can be used for this purpose. For example, the exception in the previous example about money not being goods, even though money is movable, could have been represented as an exclusionary rule as follows:

```
<rule id="Sect-9-105h2">
  <head>
    <s pred="excluded">
      <c>Sect-9-105h</c> is excluded from
      <s pred="goods"><v>c</v> is goods</s>
    </s>
  </head>
  <body>
    <s pred="money"><v>c</v> is money</s>
  </body>
</rule>
```

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:

```
<rule id="repealed">
  <head>
    <not><s pred="valid"><v>r1</v></s></not>
  </head>
  <body>
    <s pred="repealed"><v>r1</v></s>
   </body>
</rule>
```

This example also illustrates the use of negation in the head of a rule.

Further examples of rules, showing how rules are used in the axiomatization of a legal theory, are provided in Chapter 4.

Chapter 4

Theories

By ‘theory’ we mean here simply a set of propositions. The theory need not be about anything or be consistent or coherent or have any other desirable properties.

Theories may consist of an infinite number of propositions, so some way is needed to represent an infinite number of propositions in a finite text. The usual way this is done in logic is via an *axiomatization* of the theory. Some finite subset of the propositions of the theory are chosen as *axioms*. The theory is completed using a finite set of inference *rules*, which can be applied to the axioms and, recursively, the consequences of these rules, to derive the remaining propositions of the theory.

When axiomatizing theories using classical logic, we say an axiomatization is *correct* if and only if all of the axioms and all of the propositions derivable using the inference rules are members of the theory and *complete* if every proposition which is a member of the theory is either an axiom or derivable using the inference rules. As discussed in Chapter 3, however, LKIF rules model legal rules, whose conclusions are typically only presumptively true, not necessarily true. Thus a set of LKIF rules may enable conflicting, contradictory propositions to be derived from a set of premises. An inconsistent theory has no models, i.e. it describes no possible set of objects and relations among these objects, in any situation, case or domain. Thus, the theory represented in LKIF is *not* intended to be a coherent theory of some case, but rather a base of raw material for constructing and arguing about, in dialogues, theories of the case, to find the most coherent ones. This is typically done by constructing, comparing and evaluating arguments from the rules and other sources, such as the evidence and precedent cases. The result of this process is a consistent theory, a set of issues, and proofs showing how each of the issues can be resolved using arguments constructed from the theory. In summary, legal reasoning in general is an argumentation process in which theories and proofs are constructed in dialogs. Legal reasoning cannot be reduced to a mechanical, deductive application of rules to facts.

Formally, a theory in LKIF has this grammatical form:

```
Theory = element theory {
  attribute id { xsd:ID },
  Imports?, Axioms?, Rules?

Imports = element imports { Import+ }
Axioms = element axioms { Axiom+ }
Rules = element rules { Rule+ }
}
```

The id of a theory, as usual, provides a way to assign a theory a Universal Resource Identifier (URI), which can be used to reference the theory on the World Wide Web.

A theory may optionally import propositions from other XML files on the Web.

```
Import = element import { attribute uri { xsd:anyURI } }
```

Syntactically, *any* resource with a URI can be imported. However, in this report we semantically constrain the URIs to those which reference XML documents which are instances of one of the following XML schemas:

- the Resource Description Forma (RDF)
- the Web Ontology Language (OWL), and recursively
- other LKIF files.

In principle, any XML file which can be interpreted as containing a set of propositions could be imported. Future versions of the LKIF specification may extend the list of supported document types. One possible candidate for inclusion would be the Rule Interchange Format (RIF), which is currently being developed by the World Wide Web Consortium as a standard format for (some subset of) first-order logic.

As a matter of methodology, we expect OWL to be used to declare the symbols, i.e. the terminology, to be used for concepts and relations in theories.[1] OWL axioms may be used to provide some constraints on the meanings of these symbols, but constraints from legislation would normally be represented using LKIF rules, since LKIF rules has been designed to facilitate the isomorphic modeling of legislation, easing validation and maintenance.

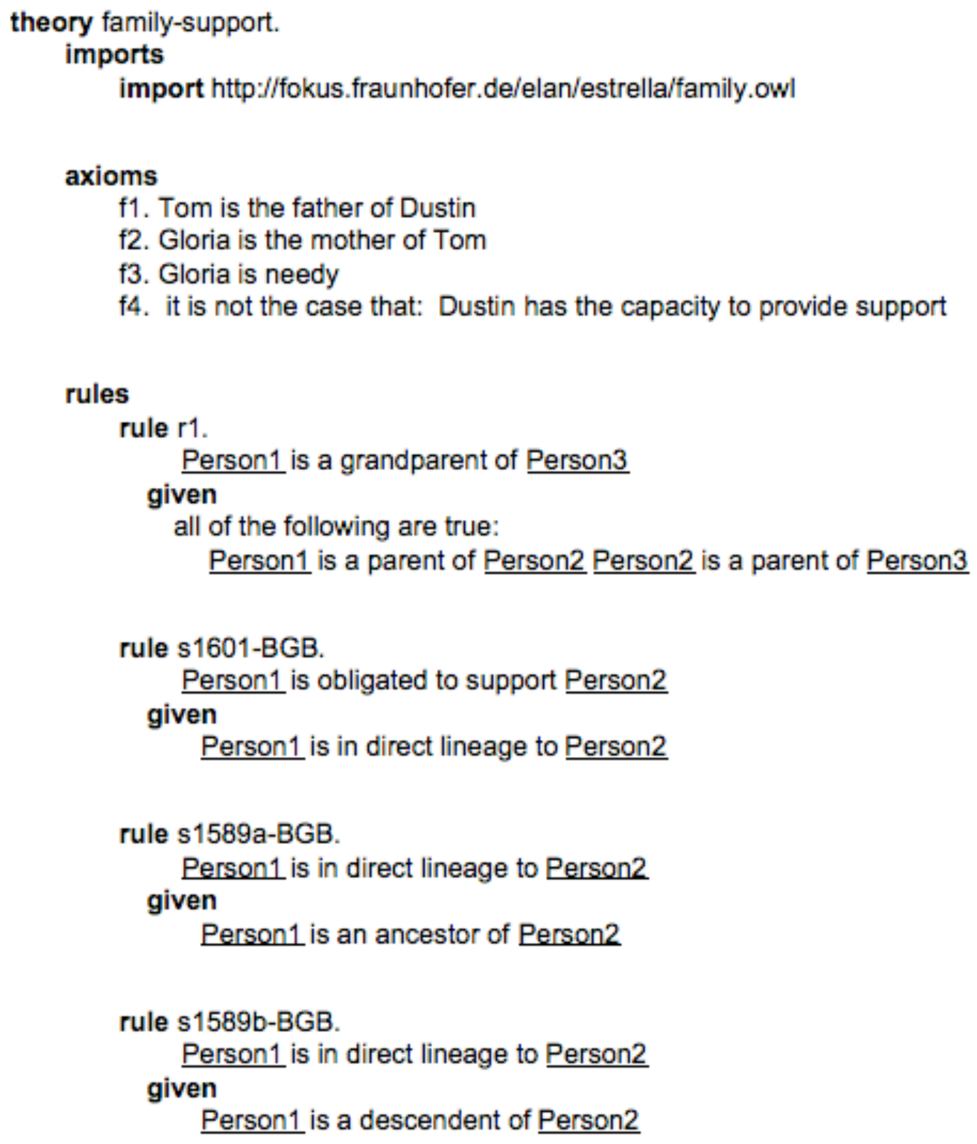
The syntax for LKIF rules was defined in Chapter 3. The syntax for LKIF axioms follows:

```
Axiom = element axiom {
 attribute id { xsd:ID },
 Wff
}
```

Essentially, an LKIF axiom simply provides a way to assign an URI to an LKIF Wff, which represent well-formed formulas of first-order logic. See Chapter 2 for the specification of the syntax of well-formed formulas, along with examples.

We finish this chapter with an example theory, from the domain of German family law. Below is the source code in LKIF. Figure 4.1 shows a more readable, formatted version of the first part of the theory, as it would appear to the author of the model when using an XML editor which supports Cascaded Style Sheets (CSS).

[1]To declare a URI is to use it in an OWL axiom, minimally an rdf:type axiom.



theory family-support.
    imports
        import http://fokus.fraunhofer.de/elan/estrella/family.owl

    axioms
        f1. Tom is the father of Dustin
        f2. Gloria is the mother of Tom
        f3. Gloria is needy
        f4. it is not the case that: Dustin has the capacity to provide support

    rules
        rule r1.
            Person1 is a grandparent of Person3
          given
            all of the following are true:
              Person1 is a parent of Person2 Person2 is a parent of Person3

        rule s1601-BGB.
            Person1 is obligated to support Person2
          given
            Person1 is in direct lineage to Person2

        rule s1589a-BGB.
            Person1 is in direct lineage to Person2
          given
            Person1 is an ancestor of Person2

        rule s1589b-BGB.
            Person1 is in direct lineage to Person2
          given
            Person1 is a descendent of Person2

Figure 4.1: Formatted Version of the Family Law Theory

```
<lkif
  xmlns:family="http:://fokus.fraunhofer.de/elan/estrella/family.owl"
  xmlns:rules="http://www.estrellaproject.org/lkif-core/lkif-rules.owl">

  <theory id="family-support">
    <imports>
      <import uri="http://fokus.fraunhofer.de/elan/estrella/family.owl"/>
    </imports>

    <axioms>
      <axiom id="f1">
        <s pred="family:father">
          <c>Tom</c> is the father of <c>Dustin</c>
        </s>
      </axiom>

      <axiom id="f2">
        <s pred="family:mother">
          <c>Gloria</c> is the mother of <c>Tom</c>
        </s>
      </axiom>

      <axiom id="f3">
        <s pred="family:needy"><c>Gloria</c> is needy</s>
      </axiom>

      <axiom id="f4">
        <not>
          <s pred="family:hasCapacityToSupport">
            <c>Dustin</c> has the capacity to provide support
          </s>
        </not>
      </axiom>
     </axioms>

     <rules>
       <rule id="r1" strict="true">
         <head>
           <s pred="family:grandparent">
             <v>Person1</v> is a grandparent of <v>Person3</v>
           </s>
         </head>
         <body>
           <and>
             <s pred="family:parent">
               <v>Person1</v> is a parent of <v>Person2</v>
             </s>
             <s pred="family:parent">
               <v>Person2</v> is a parent of <v>Person3</v>
             </s>
           </and>
         </body>
       </rule>

       <rule id="s1601-BGB">
         <head>
```

```
   <s pred="family:obligatedToSupport">
     <v>Person1</v> is obligated to support
     <v>Person2</v>
   </s>
  </head>
  <body>
    <s pred="family:directLineage">
      <v>Person1</v> is in direct lineage to <v>Person2</v>
    </s>
  </body>
</rule>

<rule id="s1589a-BGB">
  <head>
    <s pred="family:directLineage">
      <v>Person1</v> is in direct lineage to
      <v>Person2</v>
    </s>
   </head>
   <body>
     <s pred="family:ancestor">
       <v>Person1</v> is an ancestor of <v>Person2</v>
     </s>
   </body>
 </rule>

 <rule id="s1589b-BGB">
   <head>
     <s pred="directLineage">
       <v>Person1</v> is in direct lineage to <v>Person2</v>
     </s>
   </head>
   <body>
     <s pred="family:descendent">
       <v>Person1</v> is a descendent of <v>Person2</v>
     </s>
   </body>
 </rule>

 <rule id="s1741-BGB">
   <head>
     <s pred="family:ancestor">
       <v>Person2</v> is an ancestor <v>Person1</v>
     </s>
   </head>
   <body>
```

```
    <s pred="family:adoptedBy">
      <v>Person1</v> was adopted by <v>Person2</v>
    </s>
  </body>
</rule>

<rule id="s1590-BGB">
  <head>
    <not>
      <s pred="family:obligatedToSupport">
        <v>Person2</v> is obligated to support <v>Person2</v>
      </s>
    </not>
  </head>
  <body>
    <s pred="family:relativeOfSpouse">
      <v>Person1</v> is a relative of the spouse of  <v>Person2</v>
    </s>
  </body>
</rule>

<rule id="s1602a-BGB">
  <head>
    <not>
      <s pred="family:obligatedToSupport">
        <v>Person2</v> is obligated to support <v>Person1</v>
      </s>
    </not>
  </head>
  <body>
    <not>
      <s pred="family:Needy"><v>Person1</v> is needy </s>
    </not>
  </body>
</rule>

<rule id="s1602b-BGB">
  <head>
    <s pred="family:Needy"><v>Person1</v> is needy</s>
  </head>
  <body>
    <not>
      <s pred="AbleToSupportSelf">
        <v>Person1</v> is able to support himself
      </s>
    </not>
  </body>
</rule>

<rule id="s1602c-BGB">
  <head>
    <not>
      <s pred="family:Needy"><v>Person1</v> is needy</s>
    </not>
  </head>
  <body>
    <s pred="family:AbleToSupportSelf">
      <v>Person1</v> is able to support to  himself
    </s>
  </body>
</rule>

<rule id="s1603-BGB">
  <head>
    <not>
      <s pred="family:obligatedToSupport">
        <v>Person1</v> is obligated to support <v>Person2</v>
      </s>
    </not>
  </head>
  <body>
    <not>
      <s pred="family:HasCapacityToSupport">
        <v>Person1</v> has the capacity to provide support
      </s>
    </not>
  </body>
</rule>

<rule id="s1611a-BGB">
  <head>
    <s pred="rule:excluded">
      <c>s1601-BGB</c> excludes
      <s pred="family:obligatedToSupport">
        <v>Person2</v> is obligated to support <v>Person1</v>
      </s>
    </s>
  </head>
  <body>
    <s pred="family:NeedyDueToImmoralBehavior">
      <v>Person1</v> is needy due to his
      own immoral behavior
```

```
        </s>
      </body>
    </rule>

    <rule id="s91-BSHG">
      <head>
        <s pred="rule:excluded">
          <c>s1601-BGB</c> excludes
          <s pred="family:obligatedToSupport">
            <v>Person1</v> is obligated to support <v>Person2</v>
          </s>
        </s>
      </head>
      <body>
        <s pred="family:undueHardship">
          <s pred="family:obligatedToSupport">
            <v>Person1</v> is obligated to support <v>Person2</v>
          </s>
          would cause <v>Person1</v> undue hardship
        </s>
      </body>
    </rule>
  </rules>
</theory>
</lkif>
```

Chapter 5

Argument Graphs

Argument graphs, also called ‘inference graphs’, are a generalization of ‘and/or’ graphs useful for representing proofs and providing a basis for generating explanations. Diagramming or ‘visualizing’ argument graphs is a common method for helping people to understand complex chains of reasoning or arguments.

Figure 5.1 shows an example argument graph, from [Gordon et al., 2007], using a diagramming method developed in [Gordon, 2007b].

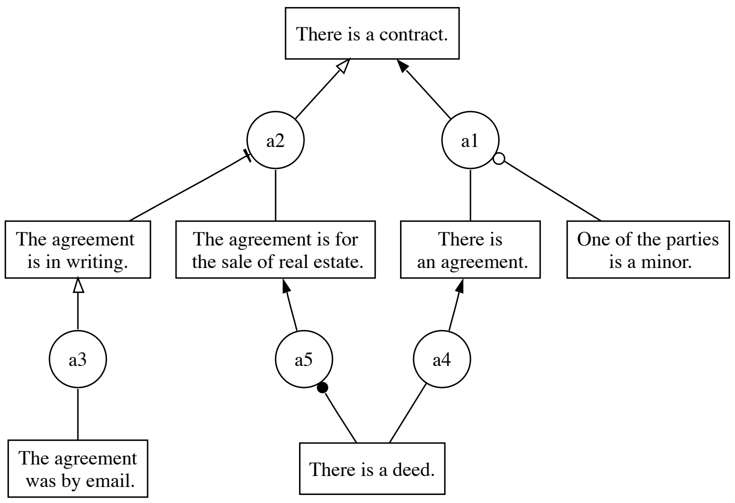

There is a contract.
a2
a1
The agreement is in writing.
The agreement is for the sale of real estate.
There is an agreement.
One of the parties is a minor.
a3
a5
a4
The agreement was by email.
There is a deed.

Figure 5.1: Argument Graph

The LKIF model of argument graphs is based on the conceptual model of the Argument Interchange Format [Carlos Ches et al., 2006], which was developed in the European ASPIC project [1] We stress that the Argument Interchange Format (AIF) is a *conceptual* model because, despite its name, it is not and interchange format and does not have a normative concrete syntax.[2]

LKIF instantiates the abstract AIF model of argument graphs in a way which allows the *acceptability* of a proposition, given a set of assumptions and an assignment of a *proof standard* to each proposition, to be computed [Gordon et al., 2007].

[1]ASPIC (IST-002307) was an Integrated Project of the European Unions 6th Framework.

[2]Three concrete syntaxes for AIF are presented in [Carlos Ches et al., 2006] for the sake of illustration.

Let us present the grammar of LKIF argument graphs top-down, beginning with the argument-graph element.

```
 ArgumentGraph = element argument-graph {
      attribute id { xsd:ID }?,
      attribute title { xsd:string }?,
      attribute main-issue { xsd:anyURI }?,
      Statements, Arguments
 }
Statements = element statements { Statement+ }
Arguments = element arguments { Argument* }
```

As with all top-level LKIF elements, an argument-graph has an id attribute, allowing it to be assigned an URI. The title attribute is a string for providing a longer, more descriptive label for the graph. The main-issue attribute is a URI pointing to the statement of the argument graph which is of primary interest in a case.

The content model of an argument-graph consists of one or more statement elements followed by zero or more argument elements. The grammar of the statement element is defined as follows:

```
 Statement = element statement {
   attribute id { xsd:ID },
   attribute value { "unknown" | "true" | "false" }?,
   attribute assumption { xsd:boolean }?,
   attribute standard { "SE" | "DV" | "BA" | "PE" | "BRD" | "CCE" }?,
   Atom
}
```

Statements serve two purposes: 1) they provide a way to assign a URI to an atomic formula, i.e. an LKIF atom, and 2) they provide a place to store additional information about the status of each atom of the argument graph in a *stage* of a dialogue:

value. The value attribute expresses whether the atomic proposition true, false or at issue ('unknown') in the case. Argument graphs do not depend on or use a three-valued logic. The underlying logic is classical two-valued logic. Each proposition is either true or false. The value assigned to an atomic proposition in a statement expresses meta-level, epistemological information about whether or not the truth value of the proposition is known in the case and , if so, which truth value applies. The default value of the value attribute is unknown.

assumption. The assumption attribute is used to express whether the value attribute has been only assumed, temporarily or hypothetically, or whether a final decision has been taken to accept the value as a fact of the case. The default value of the assumption attribute is false. Taken together, the value and assumption attributes provide four different states:

- **value=unknown, assumption=true.** By convention, we take this mean that the proposition has been stated in the dialogue, but not questioned.
- **value=unknown, assumption=false.** This means the truth of the proposition has been questioned by a party in the dialogue.
- **value=true, assumption=true.** The proposition is assumed to be true.
- **value=true, assumption=false.** A decision has been taken to accept the statement as true.
- **value=false, assumption=true.** The proposition is assumed to be false.
- **value=false, assumption=false.** A decision has been taken that the statement is false, i.e. to accept the negation of the statement as being true.

standard. The standard attribute provides a way to assign a *proof standard* to each proposition in he dialogue. This is a very general mechanism, as it allows a different proof standard to be assigned to each proposition. The proof standard must be selected from the following set:

- **SE.** Scintilla of the evidence.
- **PE.** Preponderance of the evidence.
- **CCE.** Clear and convincing evidence.
- **BRD.** Beyond reasonable doubt.
- **DV.** Dialectical validity
- **BA.** Best argument.

The first four proof standards are well known legal proof standards, suitable for issues of fact. The *dialectical validity* and best argument proof standards are suitable for legal issues. The default proof standard is 'best argument', BA. All of these proof standards are supported the reference inference engine.

The purpose of a proof standard is to provide a systematic way to aggregate conflicting pro and con arguments. There is no consensus in the computational models of argument community about how to aggregate arguments or model proof standards. Indeed, the precise meaning of the legal proof standards is an open issue of legal theory [Anderson et al., 2005]. A formal model of the scintialla of the evidence, dialectical validity and best argument standards compatible with LKIF argument graphs has been proposed [Gordon et al., 2007]. Formal models of the preponderance of the evidence, clear and convincing evidence and beyond reasonable doubt legal proof standards for LKIF are under development. There is no doubt that these are important legal proof standards, even if there is disagreement about how to model them computationally. For this reason, LKIF provides syntax for assigning these legal proof standards to propositions, even though a precise formal model of their meaning is still lacking.

Finally, we now turn to the LKIF grammar for arguments. A set of arguments links up the statements, using their URIs, into a graph.

```
Argument = element argument {
  attribute id { xsd:ID },
  attribute title { xsd:string }?,
  attribute direction { "pro" | "con" }?,
  attribute scheme { xsd:anyURI | xsd:string }?,
  attribute weight { xsd:float }?,
  Conclusion, Premises
}

Premises = element premises { Premise* }
Conclusion = element conclusion { attribute statement { xsd:anyURI }  }

Premise = element premise {
  attribute polarity { "positive" | "negative" }?,
  attribute exception { xsd:boolean }?,
  attribute role { xsd:anyURI | xsd:string }?,
  attribute statement { xsd:anyURI }
}
```

An argument links a sequence of zero or more premise elements to exactly one conclusion element. Each argument can be assigned a URI as its id. The title attribute provides a way to label an argument with a descriptive name. This is useful, for example, for defining a set of *argumentation schemes*, where each argument is intended to serve as a template for other arguments. The direction attribute is for stating whether the argument is pro or con its conclusion. Arguments are pro arguments by default, if this attribute is left unspecified. The scheme attribute provides a place to reference the argument which was used as a template. The value of the scheme argument may be either a URI or a string. This allows the common name of the scheme to be provided if, for example, the scheme has not been modeled formally in LKIF. The weight attribute of an argument is a real number in the range of 0.0 to 1.0. The default weight of an argument is 0.5. Weights are not relevant for the *scintilla of evidence*, *dialectical validity* or *best argument* standards. Whether or not weights are useful in computational models of the *preponderance of evidence*, *clear and convincing evidence* or *beyond reasonable doubt* proof standards is an open research question. LKIF provides a way to assign weights to arguments so as to be suitable for use in computational models of proof standards which use weights.

A semantic constraint on argument graphs, not expressible in the grammar, is that they may not contain cycles. This restriction is reasonable, since argument graphs model proofs and cyclic arguments are not persuasive. Moreover we are aware of no application scenarios requiring support for cyclic arguments. Syntactically, LKIF is capable of supporting cyclic arguments, since the grammar does not and cannot forbid them.

Another semantic constraint is that the statement attributes of the premise and conclusion elements should reference a statement element defined in the same argument graph containing this argument.

Premises also provide several attributes. The polarity attribute provide a way to negate a premise. Recall that all statements in an argument graph are atomic propositions, i.e. positive literals. That is, no statement in the argument graph is negative. Negation is modeled in two ways, depending on whether the premise or the conclusion of the argument is to be negated. Premises are negated by assigning the premise a negative polarity. (By default, the polarity is positive.) A negated premise holds only if its statement is *not* acceptable. Conclusions are negated by using con arguments. An argument con a proposition P is equivalent to an argument pro the proposition $\neg P$ and vice versa. The exception attribute of premise is for allocating the burden of proof for the premise to the party interested in undercutting the argument. An exceptional premise is assumed to hold unless the negation of the statement of the premise is acceptable. The role attribute of a premise element provide a way to annotate the premise with its role in the argumentation scheme applied to construct the argument. For example, if Toulmin's famous scheme is used [Toulmin, 1958], the role might be 'data', 'warrant' or perhaps 'backing'. Finally, as already suggested, the statement attribute is used to reference the propositional content of the premise, using the URI assigned the atomic proposition by a statement element of the argument graph.

Finally, let us close this chapter with an example, an LKIF version of the argument graph shown in Figure 5.1 at the beginning of this chapter.

```
<lkif>
  <argument-graphs>
    <argument-graph>
      <statements>
        <statement id="contract" value="unknown" assumption="true">
          <s>There is a contract</s>
        </statement>
        <statement id="writing" value="unknown" assumption="true">
          <s>The agreement is in writing.</s>
        </statement>
        <statement id="real-estate" value="unknown" assumption="true">
          <s>The agreement is for the sale of real estate.</s>
        </statement>
        <statement id="agreement" value="unknown" assumption="true">
          <s>There is an agreement.</s>
        </statement>
        <statement id="minor" value="unknown" assumption="true">
          <s>One of the parties is a minor.</s>
        </statement>
        <statement id="email" value="unknown" assumption="true">
          <s>The agreement was by email.</s>
        </statement>
        <statement id="deed" value="true" assumption="true">
          <s assumable="true">There is a deed.</s>
        </statement>
      </statements>
      <arguments>
        <argument id="a1" direction="pro">
          <conclusion statement="contract"/>
          <premises>
            <premise statement="agreement"/>
            <premise exception="true" statement="minor"/>
          </premises>
        </argument>

        <argument id="a2" direction="con">
          <conclusion statement="contract"/>
          <premises>
            <premise polarity="negative" statement="writing"/>
            <premise statement="real-estate"/>
          </premises>
        </argument>

        <argument id="a3" direction="con">
          <conclusion statement="writing"/>
          <premises>
            <premise statement="email"/>
          </premises>
        </argument>

        <argument id="a4" direction="pro">
          <conclusion statement="agreement"/>
          <premises>
            <premise statement="deed"/>
          </premises>
        </argument>

        <argument id="a5" direction="pro">
          <conclusion statement="real-estate"/>
          <premises>
            <premise statement="deed"/>
          </premises>
        </argument>
      </arguments>
    </argument-graph>
  </argument-graphs>
</lkif>
```

Chapter 6

Sources

LKIF provides a mechanism, using URIs, for associating each of the major elements of an LKIF model (theories, axioms, rules, statements, arguments, and argument graphs) with one or more XML resources on the World Wide Web. This feature is needed to enable elements of an LKIF model of legislation to be linked to XML representations of the legal source texts, or parts of these texts, in particular legal sources marked up using the MetaLex XML schema developed in Work Package 3 of the ESTRELLA project, and delivered as Report D3.1, the companion report of this deliverable defining LKIF.

The mechanism is very simple but also very powerful and general. At the beginning of an LKIF document, an optional sources element may be included, consisting of one or more source elements. The formal grammar of these elements follows:

```
Sources = element sources { Source+ }

Source = element source {
  attribute element { xsd:anyURI },
  attribute uri { xsd:anyURI }
}
```

Each source element consists of a pair of element and uri attributes, both of which have URI values. The element attribute should point to an element of the LKIF model in the same file as the source element. The uri attribute should point to the element in an external file containing a legal source text interpreted by the element of the LKIF model.

Acknowledgements

The work reported in this report was conducted as part of the European ESTRELLA project (IST-2004-027655). The content of this report is the result of work done by many members of the ESTRELLA consortium, not just the author of this report. In particular I would like the acknowledge the contributions of, in alphabetical order, Trevor Bench-Capon, Stephan Breidenbach, Marcello di Bello, Kasper van den Berg, Matthew Bickham, Alexander Boer, Joost Breuker, Giuseppe Contissa, Tom van Engers, Andras Forhecz, Paul Hancock, Steffen Heitkam, Szymon Klarman, Rinke Hoekstra, Ulrike Liss, Andras Millinghofer, Monica Palmirani, Samir Sekkat, Reka Vas, Radboud Winkels, and Adam Wyner.

Appendix A

Acronyms

CSS. Cascaded Style Sheet

LKIF. Legal Knowledge Interchange Format

OWL. Web Ontology Language

Relax NG. Regular Language for XML Next Generation

RDF. Resource Description Format

RIF. Rule Interchange Language

RuleML. Rule Markup Language

SWRL. Semantic Web Rule Language

URI. Universal Resource Identifier

XML. Extensible Markup Language

XSD. XML Schema Definition

XSLT. Extensible Stylesheet Language Transformations

Appendix B

LKIF Schema

```
# Copyright (C) 2008 Thomas F. Gordon, Fraunhofer FOKUS, Berlin
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License version 3 (LGPL-3)
# as published by the Free Software Foundation.
#
# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# General Public License for details.
#
# You should have received a copy of the GNU Lesser General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.

# Version:  2.0.4

datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"

grammar {
    start = element lkif {
        attribute version { xsd:string }?,  # version of the LKIF schema
        Sources?, Theory?, ArgumentGraphs?
    }

    Sources = element sources { Source+ }

    Source = element source { attribute element { xsd:anyURI },
                              attribute uri { xsd:anyURI } }

    Theory = element theory {
        attribute id { xsd:ID },
        Imports?, Axioms?, Rules?
    }
```

```
Imports = element imports { Import+ }
Axioms = element axioms { Axiom+ }
Rules = element rules { Rule+ }

Import = element import { attribute uri { xsd:anyURI } }

Axiom = element axiom {
    attribute id { xsd:ID },
    Wff
}

ArgumentGraphs = element argument-graphs { ArgumentGraph+ }

Rule = element rule {
    attribute id { xsd:ID },
    attribute strict { xsd:boolean }?, # default: false
    Head, Body?
}

Head = element head { Wff+ }
Body = element body { Wff+ }

Atom = element s {
    attribute pred { xsd:anyURI | xsd:Name }?,
    attribute assumable { xsd:boolean}?,  # default: false
    ((text | Term)*)
}

Wff = Atom | Or | And | Not | If | Iff | All | Exists
Or = element or { attribute assumable { xsd:boolean }?,  #default: false
                  Wff, Wff+ }
And = element and { attribute assumable { xsd:boolean }?,  #default: false
                    Wff, Wff+ }
Not = element not {
    attribute exception { xsd:boolean }?,  # default: false
    attribute assumable { xsd:boolean }?,  # default: false
    Wff
}

If = element if {
    attribute assumable { xsd:boolean }?,  #default: false
    Wff, Wff
 }

Iff = element iff {
    attribute assumable { xsd:boolean }?,  #default: false
    Wff, Wff
}

All = element all {
    attribute assumable { xsd:boolean }?,  #default: false
    Variable+, Wff
}

Exists = element exists {
    attribute assumable { xsd:boolean }?,  #default: false,
    Variable+, Wff
 }

Term = Variable | Individual | Constant | Expression | Atom

Variable = element v { xsd:Name }

Individual = element i {
    attribute value { xsd:anyURI },
    text
}

Constant = element c {
    # attribute value { xsd:anySimpleType }
    xsd:Name | xsd:anyURI | xsd:string | xsd:boolean | xsd:integer | xsd:float
 }

Expression = element expr {
    attribute functor { xsd:anyURI },
    Term*
}

ArgumentGraph = element argument-graph {
    attribute id { xsd:ID }?,
    attribute title { xsd:string }?,
    attribute main-issue { xsd:anyURI }?,
    Statements, Arguments
}

Statements = element statements { Statement+ }

Statement = element statement {
    attribute id { xsd:ID },
    attribute value { "unknown" | "true" | "false" },  # default: unknown
    attribute assumption { xsd:boolean }?, # default: false
    attribute standard { "SE" | "DV" | "BA" | "PE" | "BRD" | "CCE" }?, # default: BA
```

```
        Atom
    }

    Arguments = element arguments { Argument* }
    Argument = element argument {
     attribute id { xsd:ID },
     attribute title { xsd:string }?,
        attribute direction { "pro" | "con" }?,
        attribute scheme { xsd:anyURI | xsd:string }?,
        attribute weight { xsd:float }?,  # range: 0.0 bis 1.0; default: 0.5
        Conclusion, Premises
    }

    Premises = element premises { Premise* }

    Premise = element premise {
        attribute polarity { "positive" | "negative" }?, # default: positive
        attribute exception { xsd:boolean }?,  # default: false, i.e. ordinary premi
        attribute role { xsd:anyURI | xsd:string }?,
        attribute statement { xsd:anyURI }
     }

    Conclusion = element conclusion { attribute statement { xsd:anyURI }  }
}
```

Bibliography

[Anderson et al., 2005] Anderson, T., Schum, D., and Twining, W. (2005). *Analysis of Evidence*. Cambridge University Press, 2nd edition.

[Bench-Capon and Coenen, 1992] Bench-Capon, T. and Coenen, F. P. (1992). Isomorphism and legal knowledge based systems. *Artificial Intelligence and Law*, 1(1):65–86.

[Carlos Ches et al., 2006] Carlos Ches n., McGinnis, J., Modgil, S., Rahwan, I., Reed, C., Simari, G., South, M., Vreeswijk, G., and Willmott, S. (2006). Towards an argument interchange format. *Knowledge Engineering Review*, 21(4):293–316.

[Clark, 2003] Clark, J. (2003). Relax ng. `http://relaxng.org`.

[Clocksin and Mellish, 1981] Clocksin, W. F. and Mellish, C. S. (1981). *Programming in Prolog*. Springer-Verlag.

[Gordon, 1995] Gordon, T. F. (1995). *The Pleadings Game; An Artificial Intelligence Model of Procedural Justice*. Springer, New York. Book version of 1993 Ph.D. Thesis; University of Darmstadt.

[Gordon, 2007a] Gordon, T. F. (2007a). Constructing arguments with a computational model of an argumentation scheme for legal rules. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Law*, pages 117–121.

[Gordon, 2007b] Gordon, T. F. (2007b). Visualizing Carneades argument graphs. *Law, Probability and Risk*, 6(1-4):109–117.

[Gordon et al., 2007] Gordon, T. F., Prakken, H., and Walton, D. (2007). The Carneades model of argument and burden of proof. *Artificial Intelligence*, 171(10-11):875–896.

[Kowalski and Sergot, 1986] Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, 4:67–95.

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

[Pollock, 1987] Pollock, J. (1987). Defeasible reasoning. *Cognitive Science*, 11(4):481–518.

[Toulmin, 1958] Toulmin, S. E. (1958). *The Uses of Argument*. Cambridge University Press, Cambridge, UK.

[Walton, 2006] Walton, D. (2006). *Fundamentals of Critical Argumentation*. Cambridge University Press, Cambridge, UK.

ESTRELLA