APPROVAL SHEET

Title of Thesis: Extending Reasoning Infrastructure for Rules on the Semantic Web: Well-founded Negation, Incremental Courteous Logic Programs, and Interoperability Tools in SweetRules

Name of Candidate: Shashidhara Ganjugunte Master of Science, 2005

Thesis and Abstract Approved: _____

Dr. Anupam Joshi
Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

Curriculum Vitae

Name: Shashidhara Ganjugunte.

Permanent Address: 4765, Chapel Square, Baltimore, MD 21227.

Degree and date to be conferred: Master of Science, August 2005.

Date of Birth: November 09, 1979.

Place of Birth: Bangalore, India.

Secondary Education: M.E.S College, Bangalore, 1997.

Collegiate institutions attended:
University of Maryland, Baltimore County, M.S. Computer Science, 2005.
University Visvesvaraiah College of Engg., Bangalore, India,
B.E. Computer Science, 2001.

Major: Computer Science.

Minor(s):

Professional publications:

Li Ding, Pranam Kolari, Shashidhara Ganjugunte, Tim Finin, Anupam Joshi
On Modeling and Evaluating Trust Network Inference, Seventh International Workshop on Trust in Agent Societies, AAMAS 2004

Professional positions held:

Research Assistant, CSEE Department, UMBC. (Jun. 04 - May. 05)

Teaching Assistant, CSEE Department, UMBC. (Aug. '03 - Jun. '04).

Software Design Engineer, Microsoft India Private Limited, (November '01 - Jul. '03).

ABSTRACT

Title of Thesis:

Extending Reasoning Infrastructure for Rules on the Semantic Web: Well-founded Negation, Incremental Courteous Logic Programs, and Interoperability Tools in SweetRules

Author: Shashidhara Ganjugunte, Master of Science, 2005

Thesis directed by: Dr. Anupam Joshi, Associate Professor
Department of Computer Science and
Electrical Engineering

Production Rules, Description Logic(DL) and Logic Programs(LP) are the key paradigms of knowledge representation. Production rules systems (particularly the JESS rule engine) are based on the Rete network and primarily support forward inferencing. However, they do not have proper semantics for negation. The Web Ontology Language (OWL) is based on description logic and cannot express rules. Several extensions to OWL including the OWL Rules Language(ORL), Semantic Web Rule Language(SWRL)have been proposed in order to overcome this deficiency. But, in order to keep the reasoning decidable we have resorted to the "Description Logic Programs(DLP)" approach, which considers the intersection of description logic and logic programs and translates OWL constructs within the DLP subset to LP. Systems based on Logic programs, especially those supporting "Well-Founded Semantics(WFS)" such as XSB have clean semantics for handling negation. However, it is harder to author rules in such systems because of lack of higher level primitives to specify priorities and conflict handling. All of the above systems do not have the ability to invoke external procedures which can perform side-effectful actions such as sensing and effecting. In this thesis, we present SweetRules which addresses all these

issues by evangelizing Well-Founded Semantics and Situated and Courteous Logic Programs.

Extending Reasoning Infrastructure for Rules on the Semantic Web: Well-founded Negation, Incremental Courteous Logic Programs, and Interoperability Tools in SweetRules

by
Shashidhara Ganjugunte

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Master of Science
2005

Dedicated to my Grandparents

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my graduate advisor Dr. Anupam Joshi for his constant support and guidance. I am also grateful to Dr. Tim Finin and Dr. Tim Oates for their pointers and guidance.

I would also like to thank Dr. Benjamin Grosof for his constant guidance and support. Dr. Grosof was the main research supervisor for this project, and it is his vision that has manifested itself as SweetRules.

I would also like to thank my close friend Pranam Kolari, for his valuable feedback during and after writing the thesis.

I would also like to thank all the members of the SweetRules team, Chitro Neogy, Said Tabet, Mike Dean, Dave Kolas, Sumit Bhansali and Dr. Benjamin Grosof for their comments and suggestions.

I would also like to thank the SweetRules users and the community members for their questions and feedback which have helped immensely in increasing the quality of SweetRules.

My sincere thanks to my parents for their constant support and encouragement during the entire course of my Master's.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Challenges in interoperability across different knowledge representation mechanisms

Ontologies are used to model the entities and their properties in the external world. In the semantic web, this is based on description logic(DL) paradigm. It has constructs that allow specification of classes of entities and their properties. These are known as T-Box (terminology) definitions. Specific instances of the entities can also be specified and these are known as the A-Box (assertion) definitions. The reasoning in these systems is based on subsumption which is derived from the class and property hierarchies. DL reasoners can also perform classification of instances by derving their membership from the T-Box and A-Box definitions. The popular language for authoring ontologies is the Web Ontology Language (OWL) [22]. These systems have an inherent difficulty in that they cannot express rules and can only perform monotonic reasoning. Several extensions to OWL including the OWL Rules Language(ORL) [15], Semantic Web Rule Language(SWRL)[17] have been proposed in order to overcome this deficiency. But, in order to keep the reasoning decidable we have resorted to the "Description Logic Programs(DLP)" approach proposed by Grosof et. al in [2], which considers the intersection of description logic and logic programs and translates OWL constructs within the DLP subset to LP.

Ordinary Logic Programs (OLP) have been in existence for close to three decades. However, the semantics of logic programs in presence of negation with cyclic dependency was not clear until the emergence of stable model semantics [16] and Well-Founded Semantics (WFS) [6]. Of these, WFS is attractive as it is tractable for datalog(i.e. function free) programs under the restriction of bounded number of variables per rule (known as the VBD restriction). WFS handles cyclic negative recursion by using a three valued logic in which atoms might be assigned t, u or f truth values which denote true, undefined or false truth values respectively. The notion of undefined truth value is used to handle negative cycles in a logic program. OLP reasoners

typically support backward reasoning and are based on a query - answer set model. Even though these systems have clean semantics, it is harder to author rules in them because of lack of higher level primitives to specify priorities and conflict handling. Both DL and OLP do not have the ability to perform side-effectfull actions such as sensing (i.e. invoking an external procedure during inferencing in order to evaluate a predicate's truth value or to get variable bindings) and effecting (i.e. invoking an external procedure after inferencing is done, based on the conclusions drawn by the reasoner).

Production Rule systems are descendents of the OPS5 family of rule engines. In general, they support specification of rules in the form implications and are capable of forward reasoning. They also use the Rete network [5] which facilitates fast pattern matching and reduces the time required to compute the conclusions. However, they do not have proper semantics for negation and in particular, some of the conclusions drawn from a logic program with negation could be unsound. We present an algorithm to overcome this limitation.

The higher level notion of prioritized conflict handling, and declarative specification of sensing and effecting is available in the powerful Situated Courteous Logic Programs(SCLP). SCLP has another distinct advantage in that it can handle negation for any category of logic programs (i.e. acyclic, stratified and cyclic) as it is based on Well Founded Semantics(WFS).

This thesis proposes an approach to unify these heterogenous knowledge representation mechanisms via SCLP evangelizing WFS. In the following sections, we highlight the importance of Well Founded Semantics and incremental reasoning and conclude by listing the primary contributions of this thesis.

1.2 Role of Well Founded Semantics

Well Founded Semantics was originally proposed by Van Gelder [6]. WFS handles all the categories of logic programs, namely:

1. Acyclic - The LP does not have any cyclic dependencies among the predicates.
2. Stratified - The LP has cycles but the cycles are not permitted to have any predicate within the scope of negation.
3. Cyclic - The LP is permitted to have cycles both among the negated and the non-negated predicates.

Moreover, reasoning in WFS is non-monotonic as it is based on Negation-As- Failure(NAF) and the presence of the undefined truth value makes it conducive for distributed and incremental reasoning.

1.3 Significance of incremental reasoning

As the semantic web evolves, it is expected that mechanisms to support incremental compilation and reasoning will be crucial to address scalability and support distributed inferencing. Also, support for incremental reasoning facilitates building of good rule authoring tools. Towards this end we present a novel design of an incremental courteous compiler and incremental courteous inferencing mechanism.

1.4 Contributions

This thesis proposes an approach to unify these heterogenous knowledge representation mechanisms via SCLP evangelizing WFS. The major contributions of this thesis are:

1. A first of it's kind toolset for translation, automatic composition and merging of heterogenous knowledge representation systems
2. A novel approach for supporting Well Founded negation in Production Rules
3. A novel algorithm to support incrementalism in courteous logic programs

Chapter 2

Related work

2.1 Platform level tools

The Well-founded semantics for the World Wide Web(W^4)[4] project attempts to develop WFS compatible tools for the World Wide Web, providing tools to interoperate between RDF and Hornlog RuleML. However, in terms of expressiveness SweetRules provides a richer platform through the following enhancements:

1. Support for RDFS and most of OWL Lite constructs.
2. Support for SCLP RuleML beyond the hornlog fragment
3. Incremental support for compiling and inferencing.
4. Enabling well founded negation in Production Rule systems.
5. Richer translator suite with supported translations to other knowledge representation mechanisms such as KIF, Smodels etc.

2.2 Computation of the Well-Founded Model for Production Rules

Even though the WFS was proposed by Van Gelder [6], the constructive formalization of WFS was done by Przymusinki's in his work on stable stationary models in [21]. The alternating fixed point formalization of WFS was first proposed by Van Gelder in [7]. Kemp et.al study the bottom-up computation of the Well-Founded Model based on magic sets (see [3] for a detailed introduction on magic sets) in [18, 19]. They also provide an algorithm to compute the Well-Founded model based on alternating fixed point formalization and observe that the computation of the Well-Founded Model can be done using strongly connected components. We adopt a similar approach for incremental reasoning once the courteous logic program has been translated to an

ordinary logic program. Although there is a wealth of literature on the computation of well-founded semantics particularly for backward reasoning, we are not aware of any related work applying computation of well-founded semantics for Production Rules systems.

2.3 Incremental compilation for Courteous Logic Programs

We believe that there is no work done in developing an incremental compilation algorithm for courteous logic programs.

Chapter 3

SweetRules - A Platform for Semantic Web Rules

3.1 Overview and vision

Traditional knowledge representation mechanisms such as description logics, logic programs and production rules have each evolved independently to support a wide range of scenarios. However, interoperability among these heterogeneous mechanisms hasn't been well understood. In order to promote interoperability, SweetRules adopts a uniform knowledge representation mechanism based on the powerful Situated and Courteous Logic Programs (SCLP), emphasizing clean semantics by evangelizing Well Founded Semantics (WFS) and provides a suite of translators which attempt to capture a significant part of the most relevant parts of the legacy knowledge representation mechanisms. The key issues addressed by these semantic preserving translators are:

- Detecting violation of expressive restrictions whenever translation of infeasible features is requested
- Raising the capabilities of the inference engines to support courteous features (such as XSB) by doing a pre-processing step of courteous compilation in conjunction with translation.
- Providing mechanism to do effecting through an action launcher.
- Handling munging of names in systems that do not support URIs.

The legacy systems also have their own semantics and inferencing engines which may not adhere to WFS. SweetRules also attempts to overcome this deficiency by wrappering around [1] the legacy inferencing engines with additional tools which en-

[1] in some cases like the XSB inferencing engine, supporting sensing would require access to the internals of the inference engine and hence is not supported by SweetRules

sure compliance with WFS. Supporting well founded negation in production rules, which discussed at length in the chapter on SweetJess is an excellent example of this wrappering.

SweetRules also extends the SCLP RuleML DTD in a way which ensures smooth integration with WSDL web services

In this chapter we present a brief overview of SweetRules Components and architecture, support for effecting via the action launcher and web services support in SweetRules, description of the KB merging feature, followed by short descriptions of SweetRules components such as SweetXSB, SweetOnto, SweetCR, SweetJess and a brief survey of analysis tools that are part of SweetRules.

3.2 SweetRules Platform - Architecture and Components

At the core of SweetRules framework is a component repository manager which manages various categories of tools such as translators and inference engines. SweetRules identifies different types of KR formalisms by descriptors. The important types of descriptors are the KR descriptors and KB descriptors which describe the Knowledge Representation mechanism and Knowledge base respectively. A KR descriptor has fields to represent the KR encoding type (which is the format such as"RuleML", "OWL" etc), and the version number. A KB descriptor contains the KR descriptor and the contents of the actual knowledge base.
The following sections describe other components of the SweetRules core infrastructure.

Relevant API documentation: *Refer to interfaces org.semwebcentral.sweetrules.infrastructure.ISweetKRDescriptor, and org.semwebcentral.sweetrules.infrastructure.ISweetKBDescriptor.*

3.2.1 Translators

The translators are responsible for conversion from one KR to another. For example, the OWL to RuleML translator converts OWL ontologies in the DLP subset to RuleML. There are two categories of translators, namely

1. Simple translator - These perform direct translation from one KR format to another
2. Composite translator - These are generated automatically by composing the simple translators.

The SweetRules version 2.1 has 21 simple translators and 72 composite translators.

3.2.2 Inference engines

The inference engine classes serve as wrappers to the underlying inference engines such as XSB, JESS, CommonRules 3.3 and Jena. SweetRules also supports indirect inferencing, in which translators are automatically invoked to convert the source data into the native inference engine format, and the result of the inferencing are converted back to the output format specified by the user.

Relevant API documentation: *Refer to interfaces org.semwebcentral.sweetrules.infrastructure.ISweetInferenceEngine, org.semwebcentral.sweetrules.infrastructure.ISweetInferenceEngineTask*

3.2.3 Courteous compilers

A courteous compiler can be thought of as a special kind of translator that compiles the courteous features to an ordinary logic program. SweetRules has 3 courteous compilers, the IBM CommonRules 3.3 compiler, SweetRules native compiler, and SweetRules native incremental compiler. The incremental compiler, has the added advantage that it does not recompile the entire input for small changes in the rulebase. The details of the design of the incremental compiler and it's algorithm will be given in later chapters.

3.2.4 Pluggability

In addition to providing a suite of tools mentioned above, SweetRules has a pluggable architecture which makes it easy to hook up either a translator or an inferencing engine. For details, please refer to SweetRules documentation.

Relevant API documentation: *Refer to org.semwebcentral.sweetrules.infrastructure.ISweetCourteousCompiler, org.semwebcentral.sweetrules.courteouscompiler.ISweetIncrementalCourteousCompiler*

3.3 Action launcher and Web services support

SweetRules includes a native action launcher which can be used to perform effecting. This is very useful in increasing the expressiveness of inference engines like XSB which cannot perform effecting. This is done by first running the cycle of inferencing and once the conclusions are obtained the effector statements are processed to see if any action needs to be performed, if so the action launcher invokes the external procedure. In addition to supporting local method calls, the action launcher can also invoke a WSDL web service operation. In order to support specification of a web service method, the SCLP RuleML was extended by adding the wsproc element as a child of the aproc element. The DTD of the wsproc element is given below.

```
<!ENTITY % parameter.content "CDATA">
<!ELEMENT wsproc (serviceName, portName, operation,
  targetNamespace, endPointAddress)>
<!ELEMENT serviceName (#PCDATA)>
<!ELEMENT portName (#PCDATA)>
<!ELEMENT operation (operationName, parameters)>
<!ELEMENT operationName (#PCDATA)>
<!ELEMENT parameters (parameter)*>
<!ELEMENT parameter (#PCDATA)>
<!ATTLIST parameter
parameterName   %parameter.content; #IMPLIED>
<!ELEMENT targetNamespace (#PCDATA)>
<!ELEMENT endPointAddress (#PCDATA)>
<!ATTLIST endPointAddress
href %URI; #IMPLIED
>
```

For additional details, refer to the SweetRules documentation.

3.4 Knowledge base merging and RuleML includes element

SweetRules supports merging of heterogeneous knowledgebases using the RuleML "rbaseincludes" element which is a child of the top level rulebase element. The DTD for this is given below:

```
<!ENTITY % rbaseincludes.content "(#PCDATA)">
<!ELEMENT rbaseincludes % rbaseincludes.content;>
<!ATTLIST rbaseincludes
href CDATA #IMPLIED
  version CDATA #IMPLIED
  kbtype CDATA #IMPLIED
>
```

Relevant API documentation: *Refer to interface org.semwebcentral.sweetrules.infrastructure.ISweetKBMergeManager and class org.semwebcentral.sweetrules.ruleml.SweetKBMergeManager.*

As an example, the following rulebase includes an RuleML rulebase, OWL ontology and Jess facts and can be merged into a single rulebase using the **merge** command of SweetRules which automatically checks for feasibility of translation and if feasible, merges them into a single rule base after translation.

Example 1
```
<?xml version="1.0"?>
<rulebase>
<!-- The RuleML Rules -->
<rbaseincludes href="file:///C:/DEMO/orderingleadtime/OrdLeadTime.ruleml"
       version="0.8"
       kbtype="ruleml"/>

<!-- The OWL ontology -->
<rbaseincludes href="file:///C:/DEMO/orderingleadtime/OrdLeadTimeOnt.owl"
       version="1.0"
       kbtype="owl"/>

<!-- The Jess facts -->
<rbaseincludes href="file:///C:/DEMO/orderingleadtime/OrdLeadTimeFacts.jess"
       version="6.1"
       kbtype="Jess"/>
</rulebase>
```

As explained above, the first includes declaration includes the rules, the second declaration includes the OWL ontology and last declaration includes the jess facts.

For details about the content of the rulebase, ontology and the jess facts refer to the appendix. The appendix also shows the result of merging the included knowledge bases. SweetRules version 2.1 download has a "doc" folder which contains a subfolder called "demoFiles" which hosts the above example along with many other examples.

3.5 SweetXSB - Bi directional Translation from RuleML and XSB

SweetXSB is the part of SweetRules that wrappers the XSB engine and provides bi-directional translation between RuleML and XSB.

3.5.1 The RuleML *query* and *answerSet* elements

XSB is a backward reasoning engine with support query and answers. We extend the RuleML DTD, to include query and answer elements as follows.

```
<!ELEMENT query ( (_rlab,_body) | (_body,_rlab?) )>
<!ELEMENT answerSet (answer*)>
<!ELEMENT answer (binding*)>
<!ELEMENT binding ((BVar, BSubstitution)|(BSubstitution, BVar))>
<!ELEMENT BVar var>
<!ELEMENT BSubstitution cterm>
```

The query element is the child of the top level rulebase element, whereas the answerSet element is the top level element(This is bound to change later on, perhaps with the addition of a top level knowledge base element). The query element can have a label and a body. The body is same as the rule body. The answerSet element is a collection of answers, and each answer is a set of bindings. A binding is a variable and it's substitution. The following example, described in the RuleML presentation syntax, illustrates the use of the query element. For the rest of the document we present rule bases mostly in the RuleML presentation syntax or where appropriate in the RuleML XML markup.

Example 2 Consider the following sample rulebase.

```
{qua}
 pacifist(?X)
 :-   quaker(?X).

{rep}
 neg  pacifist(?X)
 :-   republican(?X).

{emptyLabel}
 quaker(nixon).

{emptyLabel}
 republican(nixon).

{emptyLabel}
 overrides(rep, qua).
```

Suppose we compile and load the above into XSB, and ask the following query, to find who is not a pacifist (notice the cneg="yes")

```
<?xml version="1.0" encoding="UTF-8"?>
<rulebase>
  <_rbaselab>
  <ind>SampleQuery</ind>
  </_rbaselab>

  <!-- Query to find who is NOT a pacifist -->
  <query>
    <_body>
       <fclit cneg="yes" fneg="no">
  <_opr>
  <rel>pacifist</rel>
```

```
          </_opr>
  <var>X</var>
        </fclit>
    </_body>
  </query>
</rulebase>
```

We get back the following answer, which says Nixon was not a pacifist

```
<?xml version="1.0"?><answerSet>
<answer>
<binding>
<BVar>
<var>X</var>
</BVar>
<BSubstitution><cterm><_opc>
<ctor>nixon</ctor>
</_opc></cterm>
</BSubstitution>
</binding>
</answer>
</answerSet>
```

The advantage of having the RuleML markup, is that one can specify the cneg attribute in the query.

Relevant API documentation: *Refer to org.semwebcentral.sweetrules.sweetxsb. SweetRuleMLQueryXSBAdapterTask, org.semwebcentral.sweetrules.sweetxsb. SweetXSBQueryXSBAdapterTask, org.semwebcentral.sweetxsb. SweetXSBLoadTask*

3.5.2 Notion of a session

In order to support backward reasoning as explained above, SweetRules has a notion of session in which the user can load a knowledge base in any format that is translateable to XSB format (including in SCLP RuleML which will be automatically compiled to OLP). The user can then query either in XSB or in RuleML.

3.5.3 Forward Reasoning with XSB

XSB only supports backward reasoning by adhering to the query-answering model. However, it can be easily made to simulate exhaustive forward inferencing by querying for every predicate, with all arguments to the predicate as variables. In general this

may not be efficient, but with tabling in XSB we believe that it is not too much overhead.

Relevant API documentation: *Refer to org.semwebcentral.sweetrules.sweetxsb.SweetXSBForwardInferencingTask*

3.5.4 Handling the undefined truth value in XSB

Even though XSB is based on Well Founded Semantics it returns the true truth value for both a true answer and an undefined answer. In order to disambiguate between these cases, for every query Q we create a dummy predicate and a rule as follows:

1. The dummy predicate "dummyPredicate" will have the same arity as the number of variables in the query.

2. We create the following rule and load it into XSB
   dummyPredicate(X) :- Q where X is a tuple of variables appearing in Q

Case of a ground query If the original query is ground, then the dummyPredicate will have zero arity. We query the XSB engine with *dummyPredicate*, if it returns true then we issue the query *dummyPredicate,tnot(dummyPredicate)* if it returns true again then the answer to original query is undefined otherwise it is true.

Case of a non-ground query For a non-ground query we first issue the query *dummyPredicate(X)*. If the XSB engine returns a non-empty binding set say B_1, we query the engine again with *dummyPredicate(X),tnot(dummyPredicate(X))* and get the binding set say B_2. The binding set corresponding to "true" truth value is B_1 - B_2.

3.6 SweetOnto - Translating the DLP subset of OWL to RuleML and SWRL

SweetOnto is based on the DLP paper [2] by Grosof et.al. This is a very important part of SweetRules as it provides interoperability between the popular OWL language based on Description Logic and other parts of SweetRules. The OWL ontology has to be in the Description Logic subset for this translation to go through.

The code originated from KAON DLP codebase from University of Karlsruhe. This was then enhanced to include:

1. Tighter validation for OWL ontologies to ensure they are in the DLP subset.

2. Translation of Datatype properties, from OWL to RuleML and SWRL. This is similar to the way object properties are handled, except that the range is a datatype. We create a unary predicate corresponding to the datatype and handle it in a way analogous to handling range in an object property. For example if the range of a property P, was

`http://www.w3.org/2001/XMLSchema#string,`

and it had an instance whose value was "abcdef", then we create a RuleML atom as follows

`_''http://www.w3.org/2001/XMLSchema#string''(abcdef).`

3. Translation of RuleML facts to RDF typed literals. This is explained, in a later section.

4. Translation of OWL within the DLP subset to SWRL. Even though SWRL rules are an expressive subset of RuleML, they make distinction between different types of atoms such as classAtom, datarangeAtom, individualPropertyAtom, dataValuedPropertyAtom, sameIndividualAtom, differentIndividualsAtom and builtinAtom. While performing the OWL to SWRL translation this meta type information is maintained before the OWL ontology is serialized to SWRL. During, the serialization phase this type information is used to serialize to appropriate category of atoms.

Relevant API documentation: *Refer to org.semwebcentral.sweetrules. sweetonto.SweetDLPCompiler,org.semwebcentral.sweetrules. SweetDLPSerializers*

3.6.1 RDF Support

SweetRules supports translation of RuleML facts to RDF typed triples and vice-versa.

RDF Facts to RuleML

Every RDF triple is converted into a set of atoms in RuleML. The conversion is done by creating a unary predicate with same URI as the class with an argument which indicates the ID. The type assertions are also done in a manner analogous to the representation of the range values in a datatype property by creating a unary atom, whose literal has the same URI as the datatype and an argument whose value denotes the value of the predicate part of the triple. The namespace prefixes for the URIs in RDF, which are very useful for the reverse translation from RuleML facts to RDF are also automatically generated.

Example 3 *The following RDF triple will be converted to RuleML as shown below.*

```
<rdf_RNSW:RDF xmlns:rdf_RNSW=
"http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:__RNSW="http://www.person.org#">
<__RNSW:Person rdf_RNSW:ID="http://www.person.org#JohnSmith">
<__RNSW:age
```

```
   rdf_RNSW:datatype=
   "http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
  30
</__RNSW:age>
</__RNSW:Person>
</rdf_RNSW:RDF>
```

```
_"http://www.person.org#Person"(_"http://www.person.org#JohnSmith").
_"http://www.person.org#age"(_"http://www.person.org#JohnSmith", 30).
_"http://www.w3.org/2001/XMLSchema#nonNegativeInteger"(30).
```

RuleML Facts to RDF

Conceptually, this translation is a many to one mapping for which we make the following assumptions:

1. A unary predicate having "XMLSchema" represents a datatype.
2. A unary predicate not having "XMLSchema" represents a class.
3. A binary predicate and it's two arguments map into a triple. That is the predicate becomes the propery, the first argument becomes the subject and the second argument becomes the fact.
4. RuleML facts of arity greater than 2 are ignored.
5. In RuleML the URIs are represented as *href* attributes which are not in the format namespace-prefix:suffix, but store the full URI. When converting to RDF if we have a URI like `http://www.person.org#Person` we cannot create an XML element using this full URI in the output RDF as it would be an invalid XML element because of presence of characters like `'/','#'`. So we need a way to separate the prefix (`http://www.person.org#`) from the suffix (Person). This separation is non-trivial, because the `#` character as a separator is not mandatory. Therefore we make the assumption that all such relevant declarations are in the RuleML file to be translated to RDF.

As an example consider the following set of RuleML facts

```
_"http://www.person.org#Person"(_"http://www.person.org#JohnSmith").
_"http://www.person.org#age"(_"http://www.person.org#JohnSmith", 30).
_"http://www.w3.org/2001/XMLSchema#nonNegativeInteger"(30).
```

is converted to a single RDF triple shown below.

```
<rdf_RNSW:RDF xmlns:rdf_RNSW=
"http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:__RNSW="http://www.person.org#">
<__RNSW:Person rdf_RNSW:ID="http://www.person.org#JohnSmith">
<__RNSW:age
   rdf_RNSW:datatype=
   "http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
  30
</__RNSW:age>
</__RNSW:Person>
</rdf_RNSW:RDF>
```

In the above, the URI prefix

‘‘http://www.person.org#’’

must be declared as a namespace prefix, otherwise the RDF elements cannot be created.

3.7 SweetCR - Bi-directional translation from RuleML to CommonRules 3.3

SweetCR is a wrapper on top of IBM CommonRules 3.3. In order to interoperate other rule systems with it we wrote bi-directional XSLT translators to translate from RuleML to a format native to CommonRules known as “Business Rules Markup Language (BRML)”. One limitation of this translation has been that BRML supports a notion of typing in terms of it’s elements IntegerTerm, StringTerm, DoubleTerm etc. Since RuleML has no notion of types, there will be loss of information during the BRML to RuleML translation.

3.7.1 SweetKIF and SweetSModels- Bi-directional Translation from BRML to KIF or Smodels via CommonRules 3.3

CommonRules 3.3 provides translators from BRML to KIF and Smodels. SweetRules provides a wrapper on top these translators. This enables bi-directional translation from RuleML to KIF or Smodels.

3.8 SweetJess - Partial Bi-directional translation from RuleML to Jess

Jess is a representative of the production rule family of rule engines and is capable of extremely fast forward inferencing. However, semantically there are problems with the way Jess handles negation. This aspect along with expressive improvements from the previous version is dealt with in detail in a later chapter.

3.9 RuleML Object model and associated tools

SweetRules includes other tools which are very useful for analyzing Rulebases. These tools are all based on the RuleML object model, which is generated from JAXB. The important ones are:

1. *Predicate dependency graph builder*: This component can take a RuleML rulebase and construct a predicate dependency graph from the Rules in the RuleML rulebase.
   Relevant API documentation: *Refer to org.semwebcentral.sweetrules.analyzer.SweetPDGBuilder*

2. *Predicate Stratifier*: This component tries to compute the stratification of a RuleML rulebase, based on Ullman's algorithm [23]. If the rulebase is not stratifiable it throws an error.
   Relevant API documentation: *Refer to org.semwebcentral.sweetrules.analyzer.SweetPredicateStratifier*

3. *Element Equality checker*: This utility can compare two RuleML elements in the object model and detect if they are equal. This is useful because the RuleML object model was autogenerated from JAXB and .equals method was not overridden.
   Relevant API documentation: *Refer to org.semwebcentral.sweetrules.ruleml.SweetEqualityChecker*

4. *Diff Facts*: This is utility that can take a diff of two fact bases in RuleML and is built on top of equality checker. This is implemented in the routines of the class SweetDiffFacts.
   Relevant API documentation: *Refer to org.semwebcentral.sweetrules.ruleml.SweetDiffFacts*

5. *Duplicate Filter*: This another utility built on top of Equality checker and filters duplicate elements from a rule base.
   Relevant API documentation: *Refer to org.semwebcentral.sweetrules.ruleml.SweetDuplicateFilter*

6. *LloydTopor transformer, class SweetLloydToporTransformer*: This tool performs the head conjunction elimination from the rules of a rule base.
   Relevant API documentation: *Refer to org.semwebcentral.sweetrules.courteouscompiler.SweetLloydToporTransformer*

7. *Unifier, Substitutor and Unifier composer*: The unification and unification composition are based on the corresponding Lloyd's algorithms in [20]. The substitutor substitues the variables in the elements of a rule base with their bindings.
   Relevant API documentation: *Refer to org.semwebcentral.sweetrules.courteouscompiler.ISweetUnifier, org.semwebcentral.sweetrules.courteouscompiler.SweetSubstitutor*

8. *RuleML Presentation Syntax generator*: This utility generates RuleML Presentation syntax from the RuleML object model.
   Relevant API documentation: *Refer to org.semwebcentral.sweetrules.ruleml.SweetRuleML2RuleMLPSTranslator*

Chapter 4

SweetJess

4.1 Overview of Production Rules systems

JESS (Java Expert System Shell) is a descendant of OPS5 family of production rules systems. It is built on top of the RETE network [5] to support extremely fast forward inferencing. JESS supports rules in a fashion similar to that of Prolog, but also comes with a lot of agenda and control features which tend to make it procedural and less declarative. The interoperability between SCLP (RuleML) and Production Rules (Jess) was studied as part of the initial SweetJess release by Grosof et. al in [13]. The primary changes [1] in this version include:

- Architecture and integration
  1. Integration with the courteous compiler
  2. RuleML to Jess rewritten with RuleML Object model
  3. Jess to RuleML, rewritten using Jess object model, to avoid parsing.
- Situated support Support for multiple sensors and effector statements per predicate
- Negation and courteous support
  1. Limited support for the NOT element of JESS (Jess to RuleML translation)
  2. Invocation of the inverse ECN transform in the Jess to RuleML translation to reverse translate classical negation.
  3. Support for well-founded negation both for stratified and cyclic cases.

[1] The implementation of the previous version was also revamped to support the new RuleML object model, and to parse the jess code in a more generic setting

4.2 Well founded negation

JESS does not handle negation properly as illustrated by the example below.

Example 4 ***A sample rule base in which negation semantics are not proper*** *(assert (f c))*
(defrule rk1 (and (f ?x) (not (h ?x))) => (assert (k ?x)))
(defrule rh2 (and (f ?x) (not (g ?x)) => (assert (h ?x)))

The expected conclusions are
(f c) and *(h c)* .
But JESS also gives *(k c)* .
This can be attributed to the fact that the rule rk1 was fired too early, without waiting for the result of the rule rh2. If we run the same rule base with ordering of rk1 and rh2 reversed, we get the correct conclusions.

4.2.1 Well-Founded negation in Stratified Logic Programs

Intuitively, if we reorder the rules in the rulebase so that the rules are fired in the correct order, then the conclusions drawn will be sound. For the stratified case this is possible, because stratification defines an predicates of rules in which the predicates in the higher strata depend on the rules in the lower strata. In order to compute stratification we use Ullman's algorithm [23].

JESS has a notion of rule salience, which represents the priority of a rule. The rules are fired in the decreasing order of the salience (i.e. rule with the highest salience is fired first). Once the stratification of the rule base is computed, we assign salience to the rules as explained below.

Let the total number of strata be T

1. If the head of a rule is not conjunctive and it belongs to stratum S then we compute salience as
   $salience \leftarrow (T + 1 - S) * \omega$
   where ω is a suitable scaling factor(default is 1000 in SweetRules) chosen for the purpose of robustness.

2. If the head of a rule is conjunctive, we compute the salience for each of the conjuncts as explained above and choose the maximum among them as the salience of the rule.

SweetRules implicitly uses the above approach of setting the salience whenever exhaustive forward inferencing with SweetJess as the inference engine is invoked on a rulebase which is stratified.

4.2.2 Well-Founded negation in general Logic Programs

In general for a logic program, stratification may not be defined. In such cases the JESS engine has to be guided externally in order to ensure correctness of the conclusions drawn. We adopt a slight variation of the alternating fixed point formalization of Well Founded Semantics to compute the conclusions. Before describing the actual algorithm, we briefly review the constructive formalization of WFS in the next section.

4.2.3 A brief review of the constructive formalization of WFS

In this section we briefly review the constructive formalization of WFS by Przymusinski in [21]. At a very high level, WFS defines a three valued logic model comprising of truth values t, f and u which denote true, false and undefined respectively.

The interpretation I of a logic program, is defined in terms of it's ground instance as the set of atoms which have been assigned true and those that have been assigned false (in terms of negation as failure). Those atoms in the Herbrand base that are not assigned either true or false are assigned the undefined truth value. The set of atoms assigned the "t" truth value are denoted by Pos(I) and the set of atoms assigned the "f" truth value are denoted by Neg(I).

Definition 1 ***NAF****: is defined as an operator to denote negation-as-failure.*

Definition 2 ***Positive LP****: A positive LP is one in which NAF and u do not appear in the body.*

Definition 3 ***Non-negative LP****: A non-negative LP is one in which NAF does not appear in the body, but u may appear in the body (i.e., as a body atom).*

The most straight forward way to implement forward chaining, is to execute the following steps, after starting with an empty interpretation in which all atoms assigned false i.e. the literals under the scope of NAF are true.

1. Derive the next set of immediate conclusions
2. Use these conclusions as premises and repeat the above steps until no new conclusions are drawn

The need for the undefined truth value, can be understood if we use the above algorithm to a logic program with cycles. The following examples illustrate the execution of this algorithm to different logic programs

Example 5 *A positive logic program.*

p(a) :- q(a).
q(a) :- r(a).
r(a) :- p(a).
p(a).

If we execute the above steps we will get a trace similar to the one below.

I_0 = { ***naf*** *p(a)*, ***naf*** *q(a)*, ***naf*** *r(a)* }
I_1 = { *p(a)*, ***naf*** *q(a)*, ***naf*** *r(a)* }
I_2 = { *p(a)*, ***naf*** *q(a)*, *r(a)* }
I_3 = { *p(a)*, *q(a)*, *r(a)* }
I_4 = { *p(a)*, *q(a)*, *r(a)* }

It is clear that the fixed point is reached in iteration I_3. This approach also works if there are no cycles involving NAF as shown below.

Example 6 *A negative logic program without cycles.*

p(a) :- q(a).
q(a) :- ***naf*** *r(a).*
I_0 = { ***naf*** *p(a)*, ***naf*** *q(a)*, ***naf*** *r(a)* }
I_1 = { ***naf*** *p(a)*, *q(a)*, ***naf*** *r(a)* }
I_2 = { *p(a)*, *q(a)*, ***naf*** *r(a)* }
I_3 = { *p(a)*, *q(a)*, ***naf*** *r(a)* }

It is again clear that quiescence (i.e. fixed point) is reached in I_2.

The examples given above illustrate that if a given logic program does not have cycles, then it has a two-valued model i.e. each of the atoms are either assigned a true or a false value. Such a model is called a total model, since every atom is assigned either true or false.

Example 7 *A negative logic program with cycles.*

p(a) :- q(a).
q(a) :- ***naf*** *p(a).*
I_0 = { ***naf*** *p(a)*, ***naf*** *q(a)* }
I_1 = { ***naf*** *p(a)*, *q(a)* }
I_2 = { *p(a)*, *q(a)* }
I_3 = { *p(a)*, ***naf*** *q(a)* }
I_4 = { ***naf*** *p(a)*, ***naf*** *q(a)* }

It is clear from the example given above that the computation will not terminate, as I_4 is same as I_0 and the whole process of inferencing is done repeatedly. In such cases, WFS makes the value of both p(a) and q(a) undefined and the computation stops and we have a partial model, i.e. a model in which some of the atoms are not assigned either true or false truth value.

The simple algorithm for forward chaining given above is formalized as an operator below.

Definition 4 ***Immediate consequence operator*** ***($\hat{T}$(P(I)))****: is the immediate transform operator on an "input" partial interpretation I, taking it to an "output" partial interpretation in which all immediate positive conclusions, and all "immediate" negative conclusions are generated.*

Two interpretations can be compared using the truth ordering operator $\preceq$ as follows.

Definition 5 *If I and J are two interpretations then $I \preceq J$ if $Pos(I) \subseteq Pos(J)$ and $Neg(I) \supseteq Neg(J)$.*

Models which are least with respect to $\preceq$ will be called as **least models**. A model is **total** if all the literals in the rule base are assigned either t or f truth values, and it is **partial** if atleast one literal is assigned the u truth value. A least model which is total will be referred to as a least model, whereas a least model which is partial will be referred to as a least partial model (LPM).

According to ([21]; Theorem 8) the least fixed point of $\hat{T}$ coincides with the LPM for a non-negative program P.

Also by ([21]; Proposition 9), we have If P is a non-negative program:

$$Pos(LPM(P)) = Pos(LM(P_{u \leftarrow f})) \tag{4.1}$$

$$Neg(LPM(P)) = Neg(LM(P_{u \leftarrow t})) \tag{4.2}$$

where $P_{u \leftarrow f}$ is defined as logic program obtained by replacing every undefined truth value in P by false and $P_{u \leftarrow t}$ is defined as thee logic program obtained by replacing the undefined truth value by true.

The characterization of WFS is given in terms of the Quotient operator and the ψ transform which are reproduced here.

Definition 6 *The quotient of a logic program P with respect to a partial interpretation I, is denoted by $\frac{P}{I}$ is defined as the program obtained by replacing every negative body literal with it's truth value.*

Definition 7 *The ψ transform of a logic program P with respect to an interpretation I, is defined as*

$$\psi_P(I) = LPM(\frac{P}{I}) \tag{4.3}$$

The well founded model of a logic program is defined in ([21]; Theorem 13) as the iterated least fixed point of the ψ transform, given by the equation below:

$$J_k = \psi_P(J_k) = LPM(\frac{P}{J_k}) \tag{4.4}$$

We denote the well-founded model of a logic program P, by WFM(P).

4.3 Well Founded Model computation in Production Rules systems

We use equations 4.1, 4.2 and 4.4 to derive an algorithm to compute the well founded model.

We can compute the WFM(P) via a depth-2-nesting iteration (we use k to denote the outer loop iteration count), where the outer is an iteration on ψ_P, with iterates J_k, and the inner iteration computes LPM(P/J_k), where that is generated via two parallel iterations: $Pos(LM((P/J_k)_{u\leftarrow f}))$ and $Neg(LM((P/J_k)_{u\leftarrow t}))$, Also note that we don't directly compute P/J_k since P is non-ground but J_k is ground, rather test P's negative literals against J_k as a positive and negative "fact" store.

We define OLR to be Old Lower-bounding Rulebase used for generating t conclusions in the previous k's (outer loop) iterate. In terms of Przymusinski, OLR is $(\frac{P}{J_{k-1}})_{u\leftarrow f}$.
OHR is the Old Higher-bounding Rulebase used for generating f conclusions (via its complement) in the previous k's (outer loop) iterate. OHR is $(\frac{P}{J_{k-1}})_{u\leftarrow t}$.
CLR is the Current Lower-bounding Rulebase used for generating t conclusions in the current k's (outer loop) iterate. CLR is $(\frac{P}{J_k})_{u\leftarrow f}$.
CHR is the Current Higher-bounding rulebase used for generating f conclusions (via its complement) in the current k's (outer loop) iterate. CHR is $(\frac{P}{J_k})_{u\leftarrow t}$.
Note that each of these rulebases above is a positive LP. OLA, OHA, CLA, and CHA are each the set of true ground Atoms concluded, respectively, from OLR, OHR, CLR, and CHR. I.e., the final working fact sets generated in each of those rulebases after each rulebase has been executed until quiescence.

OLA is Pos(LPM($\frac{P}{J_{k-1}}$)). OHA is Complement(Neg(LPM($\frac{P}{J_{k-1}}$))). CLA is Pos(LPM($\frac{P}{J_k}$)). CHA is Complement(Neg(LPM($\frac{P}{J_k}$))).

We will treat OLA, OHA, CL, and CH as modules. Each module contains premise rules and also conclusions. Note that in our terminology, some of the premise rules have the form of facts.

In CLR, a NAF literal should succeed when the NAF'd atom is not in OHA, i.e., when the atom is in Complement(OHA), i.e., when the atom is in Neg(LPM($\frac{P}{J_{k-1}}$)).

In CHR, a NAF literal should succeed when the NAF'd atom is not in OLA, i.e., when the atom is in Complement(OLA), i.e., when the atom is in

Complement(Pos(LPM($\frac{P}{J_{k-1}}$))). Complement(Pos(LPM($\frac{P}{J_{k-1}}$))) - Neg(LPM($\frac{P}{J_{k-1}}$)) Note that Complement(Pos(LPM($\frac{P}{J_{k-1}}$))) - Neg(LPM($\frac{P}{J_{k-1}}$)) is the set of atoms that have undefined truth value in LPM($\frac{P}{J_{k-1}}$).

In sections that follow, we describe how to accomplish the computations described above. We also assume that modules can be assigned from one to another, by a statement of the form $module_i := module_j$ which copies all the facts and rules in $module_j$ to $module_i$, and they can be compared for equality, i.e. $module_i == module_j$ is true if rules or facts in $module_i$ are same as those in $module_j$. In the comparison for equality the module prefixes are ignored.

4.3.1 Initialization (1st iteration, k=1)

We use the suffix to 1 denote the fact that k = 1. In CLR_1, every NAF literal should fail. OHA_1 is thus in effect the set of all ground atoms. But this is not practical to represent as a fact set. It's simpler instead to do the following rewriting to create CLR_1. We take the original rulebase P and rewrite it by dropping every rule that contains a NAF literal.

In CHR_1, every NAF literal should succeed. OLA_1 is thus in effect the empty set. i.e. $OLA_1 := \emptyset$

4.3.2 Post initialization iterations

1. The previous outer loop iterate's (k-1's) CLR, CLA, CHR, and CHA are overwrite-copied to become, respectively, the current outer loop iterate's (k's) OLR, OLA, OHR, and OHA. Actually, we only need to keep OLA and OHA, not OLR and OHR.

2. To create CLR (for k>1), we take the original rulebase P and rewrite it as follows:

   - Replace every NAF'd atom q(T) by OHA::q(T).
   - Copy OLA into CLR as an additional set of premise facts. (performance tweak)

   Here, T is a tuple of argument terms of appropriate arity/slots for q.

3. To create CHR, we take the original rulebase P and rewrite it as follows:

   - Replace every NAF'd atom q(T) by OLA::q(T).
   - Copy OLA into CLR as an additional set of premise facts. (performance tweak)

4. To compute CLA, execute exhaustive forward inferencing on CLR. I.e., put module focus on CL, and execute it till quiescence

5. To compute CHA, execute exhaustive forward inferencing on CHR. I.e., put module focus on CL, and execute it til quiescence (e.g., until detect an empty execution agenda)

4.3.3 Termination (last iteration m)

When the outer loop is quiescent, we say that the final iteration has been reached. Let's call that iteration k=m. This quiescence can be detected as follows.

If ((CLA == OLA) and (CHA == OHA)) then outer loop has quiesced.

Notationally, let FLA and FHA stand for the final iteration's CLA and CHA, respectively: FLA is CLA_m and FHA is CHA_m.

Note that Complement(FHA) are the atoms that have f truth value, and that (FHA - FLA) are the atoms that have u truth value.

4.3.4 Extending to effecting

We need to defer effecting until the outer loop iteration is done. We could do this effecting by calling an "action launcher" set of rules at the end. "Deferring" here means that no effecting should happen during the inner loop CLR_k and CHR_k executions. If effecting were to occur during the inner loop executions, it might happen repeatedly, which would be unintended and also unsoundly (in the case of CHR_k). We can do the deferring by rewriting the rules to have separate effecting rules, essentially corresponding to effector statements in the manner of Situated LP as explained in the SweetJess working paper [14]. The pure rules (possibly with sensors) would be executed, determining the final FLA and FHA. Then effector rules would be executed, generating actions. This rewriting would only be necessary if we started with Jess/CLIPS form rules. If starting from Situated OLP (SOLP), i.e. OLP with sensor and effector statements, we would not need to rewrite. For simplicity, we will assume we are starting with SOLP, e.g., RuleML.

4.3.5 Supporting Sensing

The key semantic issue is that the sensor predicates/expressions may appear within the scope of **naf** . In CLR and CHR, the NAF'd atoms should get rewritten only if they are not sensor atoms, and in the initialization of CLR a rule should be eliminated only if there is a non-sensor NAF literal.

4.3.6 Detailed Algorithm

Algorithm 1 Computation of the Well Founded Model of a logic program P

SP := the set of sensor predicates that is those that appear in a sensor statement
NAFP := the set of NAF'd predicates.
{The set of NAF'd non-sensor predicates}
NP := NAFP - SP.
{declare modules CL, CH, OLA, OHA, EF}
k := 1. {k is the counter of outer loop iterations}
quiesced_outerloop := false.
CLR_init := (the rules in P, rewritten to drop every rule containing a NAF'd non-sensor atom).
CLR_basic := (the rules in P, rewritten to replace every NAF'd non-sensor atom q(t) by OHA::q(t)).
CHR_basic := (the rules in P, rewritten to replace every NAF'd non-sensor atom q(t) by OLA::q(t)).
EFR_basic := (the effector statements in P, rewritten in Jess/CLIPS style as action-head rules).
CLR := CLR_init.
CHR := CHR_basic.
OLA := emptyset.
{Execute CLR in module CL, until its quiescence.}
CLA := the resulting set of conclusions.
{Execute CHR in module CH, until its quiescence. OK to compute this concurrently with previous step }
CHA := the resulting set of conclusions.
CLR := CLR_basic. { in prep for k=2 }
{Main outer loop}
if k greaterthanorequalto 1 **then**
  while quiesced_outerloop == false **do**
    k := k+1.
    OLAN := (CLA - OLA).
    OLA := CLA.
    OHA := CHA.
    CLR := (CLR union OLAN).
    CHR := (CHR union OLAN).
    {Execute CLR in module CL, until its quiescence. I.e., put module focus on CL, and execute it til quiescence (e.g., until detect an empty execution agenda).}
    CLA := the resulting set of conclusions.

{Execute CHR in module CH, until its quiescence. OK to compute this concurrently with previous step}

25: CHA := the resulting set of conclusions.

26: quiesced_outerloop := ((CLA == OLA) and (CHA == OHA)). {NB: to implement efficiently, perhaps could compare counts of conclusions}

27: **end while**

28: **end if**

29: **if** quiesced_outerloop == true **then**

30: FLA := CLA

31: FHA := CHA

32: EFR := (EFR_basic $\bigcup$ CLA).

33: Execute EFR in module EF, until its quiescence. {run the effecting rules on the WFM's true conclusion facts} {I.e., put module focus on CL, and execute it til quiescence } {e.g., until detect an empty execution agenda}

34: **end if**

Chapter 5

Overview of Courteous Logic Programs

Ordinary Logic Programs(OLP) particularly those based on Well Founded Semantics offer an attractive platform for specification of policies as they are tractable[1] and offer clean semantics for negation. However, in terms rule authoring they are highly unintuitive because of lack of primitives for specification of high level concepts such as rule priority, opposition and classical negation. Courteous logic proposed by Grosof [8–11] offers an attractive platform for Rule authoring as it has highly expressive primitives for rule authoring, with added support for situated features such as sensing and effecting.

5.1 Generalized Courteous Logic Programs(GCLP)

The extensions that courteous logic programs define over ordinary logic programs are defined in [11] and are reviewed below.

Definition 8 ***Labelled Rule****: Courteous logic Programs extend the notion of a rule in an OLP, by adding a label as shown below.*
{LABEL} L_0 :- L_1 and L_2 and ... ***naf*** *L_i and ... L_n*
where "naf" stands for negation-as-failure. The L_i are literals which can be either positive or be under the scope of classical negation operator "neg".

Definition 9 ***Mutual Exclusion Constraint or Mutex****: A general mutex, specifies scope of opposition among literals. This can conditional or unconditional. The number of the opposers is restricted to be 2.*
A mutex is specified as:
!- L_1 and L_2 | G_1 and G_2 ... G_k
where L_i are literals which can be in the scope of "neg", G_i are literals which can be

[1] under the Datalog and Variable Bound restrictions

in the scope of "neg" or "naf".
There is also an implicit mutex (also known as classical mutex) between every atom and the classical negation of the atom, which indicates unconditional oppostion between an atom and it's classical negation.

Definition 10 ***Overrides predicate****: The reserved overrides predicate, specifies strict prioritization between it's arguments. For example, overrides(L_i, L_j) specifies that rules with label L_i have strictly higher priority over rules with label L_j.*

The situated features such as sensing and effecting perform side effectful operations and are described in detail in [12].

5.2 Static courteous transformations

Here we briefly review the courteous transformations defined in [11] to translate a GCLP to OLP. The transformation proceeds in two phases, first phase eliminates classical negation and the second one performs courteous transformations taking into account the prioritization predicate (i.e. overrides) and the mutexes.

5.2.1 Elimination of Classical Negation

In this phase, if any literal appears within the scope of classical negation it is replaced by an adorned literal with the classical negation removed. A mutex between the adorned literal and unadorned literal is also added to the output of the transform, signifying oppostion between the two literals. Such a mutex is called as implict or classical mutex. For example, if the predicate P appears within the scope of "neg" then it is replaced by "n_P" (which is assumed not occur any where else in the rule base), and a mutex of the form shown below is added to the output.

!- P(X) and n_P(X)
where X is a tuple of variables of same arity as P.

A mutex between a literal and it's classically negated version is called as an implict or classical mutex.

5.2.2 Compiling the courteous features Post ECN

For any predicate q, let the RuleLocale(q) denote the set of rules that mention q in the head.

Let M_i be a mutex defined as:
$! - P1(u1)$ `and` $P2(u2)$ $|Ei[zi]$
where P1, P2 are predicates and u1, u2 are term tuples, Ei[zi] represents the condition under which opposition holds.

We introduce the notions of a RelOppTriple, Semi-RelOppTriple, ordered mutexes and mutex duals below.

Definition 11 ***Relevant Opposition Triple(RelOppTriple)*** *Let r1 be a rule with head P1(t1) and r2 be a rule with head P2(t2), where t1, t2 are term tuples and let M_i be a mutex as define above.*
$< r1, \theta, r2 >$ is a relevant opposition triple or RelOppTriple iff there is a maximum general unifier θ such that $< t1, t2 >=< u1, u2 >$ Notationally, it is convenient to consider $< r1, M_i, r2 >$ as a RelOppTriple and ignore the unifier. We make use of both the notations in the description of incremental compilation.

Definition 12 ***Semi Relevant Opposition Triple(Semi-RelOppTriple)*** *This is similar to the RelOppTriple defined above, except that now we consider a single rule R_k and it's unification with one of the opposer literals of a mutex M_i. We denote this as $< R_k, M_i >$, if a unifier exists. These are pre-computed in order to make the incremental compilation fast.*

Definition 13 ***Ordered Mutexes****: An ordered mutex is one in which the permutation of the opposers is not allowed, i.e. even if the only difference between two opposers is in the ordering of their opposers then those mutexes are considered different.*

Definition 14 ***Mutex dual****: For an ordered Mutex M_i = !- P1(t1) and P2(t2) | B i.e. is P1 is the first opposer predicate, P2 is second opposer predicate, t1 and t2 are term tuples, and B is the given condition, the dual $Mdual_i$ is constructed as $Mdual_i$ = !- P2(t2) and P1(t1) | B.*

For the purposes of discussion of the incremental algorithms, we consider only ordered mutexes, by applying the proposed algorithms first to the mutex and then to it's dual. But this is transparent from an end-user perspective.

The courteous transforms are defined for each predicate q, in terms of it's rule locale. If a rule does not have any opposers then it is copied to the output without any change. For every rule $rule_j$ in the locale of q that has opposers, the rules produced by the courteous transforms explained below are included in the output. For the purposes of these transforms some adorned predicates are created which are assumed not to exist anywhere else in the rulebase. The transforms are described below.

The first transform states that the head of q(tj) is true whenever there is an unrefuted candidate for it, and it is not skeptically defeated.

$$q(tj) : -q_u(tj) \texttt{ and } \mathbf{naf} \;\; q_s(tj) \tag{5.1}$$

where q_u and q_s are newly introduced predicates of same arity as q, and tj is term tuple of q in the head of the rule $rule_j$.

The second transform creates a place holder candidate which fires whenever the body of the original rule (before compilation) is satisfied.

$$q_{cj}(tj) : -Bj[yj]. \tag{5.2}$$

where q_{cj} is also a newly introduced predicate, Bj[yj] denotes the body of the rule $rule_j$.

The third transform defines the notion of an unrefuted predicate, which is defined to be true whenever the candidate fires and it is not refuted by a higher priority predicate.

$$q_u(tj) : -q_{cj}(tj) \texttt{ and } \mathbf{naf} \ \ q_{rj}(tj) \tag{5.3}$$

The concept of a refuted literal and that of a skeptically defeated literal are defined on a per relevant opposition triple basis. For each relevant opposition triple jikTriple $< rule_j, \theta_{jik}, rule_k >$, in which $rule_j$ is involved, the following rules are included in the output.

The fourth transform states that a literal is refuted if it has a candidate, and it's opposer has a candidate and the priority of the opposer strictly greater than that of the candidate and condition of opposition specified in the mutex is true.

$$q_{rj}(tj.\theta_{jik}) : -q_{cj}(tj.\theta_{jik}) \texttt{ and } p^i_{ck}(wk.\theta_{jik}) \texttt{ and } overrides(lab_k, lab_j) \texttt{ and } Ei[(Zi.\theta_{jik})]. \tag{5.4}$$

The fifth transform states that a literal is skeptically defeated if it is unrefuted and the opposer is also refuted and the condition for opposition holds.

$$q_{sj}(tj.\theta_{jik}) : -q_u(tj.\theta_{jik}) \texttt{ and } p^i_u(wk.\theta_{jik}) \texttt{ and } Ei[(Zi.\theta_{jik})]. \tag{5.5}$$

The transforms given above 5.1, 5.2, 5.3, 5.4, 5.5 will be referred to as courteous transforms 1j, 2j, 3j, 4j, 5j respectively. For detailed explanation of these transforms refer to [11].

Chapter 6

Incremental Courteous Compiler

6.1 Scenarios for incremental compilation and reasoning

In an open and dynamic environment like the web, we can envision scenarios in which small updates in the form of rule (including facts) or mutex addition or deletion happening frequently. Under these circumstances, it will be very helpful to have a tool which incrementally compiles and derives conclusions based on the new update, instead of recompiling the whole rulebase and deriving all conclusions from it. The presence of an incremental compiler also facilitates development of good rule authoring tools, which enable the rule author to incrementally view the impact of small changes. In order to support these scenarios we have developed an incremental courteous compiler to perform the courteous transforms described above.

6.2 Incremental Courteous Transforms

Let CR (R) represent the result of courteous transformation on a GCLP rulebase R, and CR_{123} represent the result applying the transforms 1j, 2j, 3j and CR_{45} represent the result of applying the transforms 4j, 5j. We denote any change to the rulebase by δR. For an add update the resulting GCLP rulebase will be denoted by $R + \delta R$, and for a delete update it will be denoted by $R - \delta R$. For sake of simplicity, in describing the transforms we assume that each update is on either a single rule or single mutex.

6.2.1 Compilation-Soundness and Compilation-completeness properties of the courteous transform

In order to prove the correctness of the incremental algorithm, we need define the compilation-soundness and compilation-completeness of courteous compilation below.

Definition 15 ***Compilation-Soundness****: The result of the courteous transform CR is compilation-sound if*

1. *Every rule in the output OLP which is the result of transforms 1j or 2j or 3j is dervied from a rule which has opposers*
2. *Every rule in the output OLP which is the result of transforms 4j or 5j is derived from a valid RelOppTriple, i.e. a triple* $< Rule_j, \theta_{jik}, Rule_k >$ *where* θ_{jik} *unifies the head of* $Rule_j$ *with first opposer literal of* $Mutex_i$ *and simultaneously unifies head of* $Rule_j$ *with second opposer literal of* $Mutex_i$
3. *Every rule in the output OLP that is but a replica of a corresponding rule in GCLP does not have any opposers.*

Definition 16 ***Compilation-Completeness****: The result of the courteous transform CR is compilation-complete if every rule that has no opposers is replicated just replicated without adornment in the output OLP, and every rule that has opposers is subject to transforms*

1. *1j, 2j, 3j once*
2. *4j, 5j once for each RelOppTriple in which the rule is a part of.*

We characterize the incremental compilation with the following lemmas.

Lemma 1 *The set of relevant opposition triples of a courteous rulebase is non-decreasing for a sequence of add updates. This also means that the number of opposers for any rule and the number of opposers produced by a mutex is non-decreasing.*

Lemma 2 *On a rule add, there can be no new RelOppTriple in which the newly added rule is not a part of.*

Lemma 3 *On a mutex add, the new RelOppTriples must involve the rules only from the locales of the two opposer literals' predicates*

Lemma 4 *On a rule delete, the rules that have to be retracted are:*
case (1): The rule had no opposers
Only the rule itself.
case (2): The rule had opposers
(a) those produced from transforms 1j, 2j, 3j on the deleted rule, and any other rule which becomes newly unopposed.
(b) those produced from any RelOppTriples in which the rule is a member of as a result of transforms 4j,5j.

Lemma 5 *On a mutex delete, the rules that have to be retracted are those produced via transforms 4j, 5j from the RelOppTriples in which the mutex was a member of. Also, if mutex deletion creates new unopposed rules then the rules produced by applying transforms 1j, 2j, 3j on these unopposed rules must be retracted.*

6.3 Incremental courteous transforms

In the transforms described below we do not consider simultaneous addition or removal of the classical mutexes, as it makes the description of the transforms more complicated. The addition or removal of classical mutexes can be modelled as separate steps performed serially with the actual rule or mutex update, without affecting the correctness of the transforms. This is described in detail in the algorithms which implement the courteous transforms. The courteous transforms are illustrated with a fictitious scenario in which a rule base and an inference engine are used for prescription of medicine.

Example 8 *Assume a patient can not take aspirin and thyomine together if the patient is female who is pregnant, here is the rule set that excludes the prescription of both aspirin and thyomine (drug names are hypothetical).*
Prescription Rule base

```
{rule_minorpain}
 prescription(aspirin, ?Patient)
 :-   symptom(minorpain, ?Patient).

{rule_ulcer}
 prescription(thyomine, ?Patient)
 :-   symptom(ulcer, ?Patient).

!-  prescription(aspirin, ?Patient) and
    prescription(thyomine, ?Patient)
  |   pregnant(?Patient).

symptom(minorpain, sue).
symptom(ulcer, sue).
pregnant(sue).
```

In the description below, we assume that a part of the above rule base is compiled and incrementally add the remaining parts and demonstrate how incremental compilation is performed. The mutex of this rule base will be referred to as prescriptionMutex

Transform 1 ***Courteous transform for an add rule update*** *Let the new rule being added be subject to ECN.*
Let R_k stand for the result of ECN applied to that new rule.
Case 1. The Rule does not have any opposers

$$CR(R + \delta R) = CR(R) \bigcup R_k \tag{6.1}$$

Case 2. The Rule has opposers The following equations describe the transform if the new Rule being added has opposers.

$$T123R_k = CR_{123}(R_k) \tag{6.2}$$

$$T45R_k = \bigcup_{rt \in RelOppTriples(R_k)} CR_{45}(rt) \tag{6.3}$$

$$TNewOppoAdd = \bigcup_{R_l \in NewlyOpposedRuleSet} CR_{123}(R_l) \tag{6.4}$$

$$NewOppoDel = NewlyOpposedRuleSet \tag{6.5}$$

$$CR\ (R + \delta R) = \Big(CR\ (R) \bigcup T123R_k \bigcup T45R_k \bigcup TNewOppoAdd\Big) - NewOppoDel \tag{6.6}$$

$T123R_k$ is the result of applying courteous transforms 1j,2j,3j to rule R_k
$T45R_k$ is the result of applying courteous transforms to RelOppTriples generated by R_k. This includes the RelOppTriples generated both from the mutexes for which the head of R_k unifies with second opposer literal and the dual of each of these mutexes.
NewlyOpposedRuleSet is the set of rules that previously did not have any opposers, but now have an opposer in the form of R_k
TNewOppoAdd is the result of applying courteous transforms 1j,2j,3j to rules in the NewlyOpposedRuleSet.
NewOppoDel is the NewlyOpposedRuleSet itself.

Sketch of proof of correctness of transform 1 If the rule does not have any opposers, by lemma 2 no additional RelOppTriple will be created and we just need to replicate the rule in the output without running transforms 1j, 2j, 3j. Intuitively, if the rule has opposers we need to apply transforms 1j, 2j, 3j, this is done by $T123R_k$. For every new RelOppTriple generated we need to apply transforms 4j,5j. By lemma 1 and lemma 2 the RelOppTriples are increasing and the new ones come only from the Rule added. This is done by $T45R_k$.
$T123R_k$ and $T45R_k$ alone do not ensure compilation-completeness however as we also need to ensure that we run transforms 1j, 2j, 3j on any previously unopposed rules which now have R_k, as the opposer. This is done by TNewOppoAdd. To ensure compilation-soundness, we need to remove the previously unopposed rules from the output, so we remove the rules in the set NewOppoDel.

Therefore the transform 1 is compilation-sound and compilation-complete.

Example 9 *Suppose we started from the rulebase in example 8, but without the rule "rule_ulcer", then there would be no opposers and the rule base would be replicated in the output OLP. But if we add the rule "rule_ulcer", then it will oppose the rule "rule_minorPain". This leads to the generation of RelOppTriples <rule_minorpain,* prescriptionMutex, *rule_ulcer> and <rule_ulcer,* prescriptionMutex, *rule_minorpain>. The unopposed facts are replicated as they do not have opposers. Since the added rule rule_ulcer has opposers, transforms in case 2 of the add rule incremental compilation are applied, $T123R_k$, TNewOppoAdd are generated by applying courteous transforms 1j, 2j, 3j to the rules rule_ulcer and rule_minorPain respectively. $T45R_k$ is generated by applying the courteous transforms to each of the RelOppTriples generated above. The previously replicated rule_minorPain becomes NewOppoDel and is deleted from the OLP(replaced by the result of applying transforms 1j,2j,3j on it). The resulting OLP is shown below:*

```
{1_1j}
 prescription(thyomine, ?Patient)
 :-   prescription_u(thyomine, ?Patient) and
 naf   prescription_s(thyomine, ?Patient).

{2_2j}
 prescription_c_2(thyomine, ?Patient)
 :-   symptom(ulcer, ?Patient).

{3_3j}
 prescription_u(thyomine, ?Patient)
 :-  prescription_c_2(thyomine, ?Patient) and
 naf   prescription_r_2(thyomine, ?Patient).

{4_4j}
 prescription_r_2(thyomine, ?Patient)
 :-   prescription_c_2(thyomine, ?Patient) and
  prescription_c_1(aspirin, ?Patient) and
  overrides(rule_minorpain, rule_ulcer) and
  pregnant(?Patient).


{5_5j}
 prescription_s(thyomine, ?Patient)
 :-  prescription_u(thyomine, ?Patient) and
  prescription_u(aspirin, ?Patient) and
  pregnant(?Patient).

{6_1j}
```

```
 prescription(aspirin, ?Patient)
 :-   prescription_u(aspirin, ?Patient) and
 naf   prescription_s(aspirin, ?Patient).

{7_2j}
 prescription_c_1(aspirin, ?Patient)
 :-   symptom(minorpain, ?Patient).

{8_3j}
 prescription_u(aspirin, ?Patient)
 :-  prescription_c_1(aspirin, ?Patient) and
 naf   prescription_r_1(aspirin, ?Patient).

{9_4j}
 prescription_r_1(aspirin, ?Patient)
 :-   prescription_c_1(aspirin, ?Patient) and
  prescription_c_2(thyomine, ?Patient) and
  overrides(rule_ulcer, rule_minorpain) and
  pregnant(?Patient).

{10_5j}
 prescription_s(aspirin, ?Patient)
 :-  prescription_u(aspirin, ?Patient) and
  prescription_u(thyomine, ?Patient) and
  pregnant(?Patient).

 pregnant(sue).

 symptom(ulcer, sue).

 symptom(minorpain, sue).
```

In the above OLP, the rules are labelled only for illustrative purposes. Every label is of the form <serial number>_<courteous transform applied>. The unopposed facts are just replicated in the OLP.

Transform 2 ***Courteous transform for an add mutex update*** *Let the new mutex being added be M_i.*

$$CR\ (R+\delta R) = \left(CR\ (R) \bigcup_{rt \in RTGen_{Mi}} CR\ _{45}(rt) \bigcup TNewOppoAdd \right) - NewOppoDel \tag{6.7}$$

$RTGen_{Mi}$ represents the relevant opposition triples generated by the mutex being added.
$TNewOppoAdd$ and $NewOppoDel$ are defined as above except that they are generated from rules for which opposers were created due to mutex addition.

Sketch of proof of correctness of the transform 2 The proof is similar to the case of add rule, and lemma 1 still applies. The only difference being the new RelOppTriples generated are due to the new mutex only (from lemma 2). So, we need to run courteous transforms 4j, 5j on each of the RelOppTriples generated by the mutex. In order to ensure compilation-completeness we need to run transforms on previously unopposed rules which now have opposers, and to ensure compilation-soundness we need to remove the unopposed rules from the output. If the mutex does not result in any new opposers, then $RTGen_{Mi}$, TNewOppoAdd, NewOppoDel will be empty and as expected nothing in the output changes. These make the transform compilation-sound and compilation-complete.

Example 10 *As illustrated for the add rule update if we load the prescription rule base in example 8 without the mutex and then add the mutex for incremental compilation, the new RelOppTriples will be <rule_minorpain, prescriptionMutex, rule_ulcer> and <rule_ulcer, prescriptionMutex, rule_minorpain>. These are stored in $RTGen_{Mi}$, and TNewOppoAdd and NewOppoDel are analogous to the add rule case.*

Transform 3 ***Courteous transform for a delete rule update*** *Let the Rule being deleted be R_k*

Case 1: The rule being deleted was unopposed

$$CR(R - \delta R) = CR\ (R) - R_k \tag{6.8}$$

Case 2: The rule being deleted had opposers

$$CR\ (R - \delta R) = CR\ (R) \bigcup NewNoOppoAdd - \Big(T123R_k \bigcup T45R_k \bigcup TNewNoOppoAdd\Big) \tag{6.9}$$

where
$NewNoOppoAdd$ represents the set of rules whose sole opposer was R_k
$TNewNoOppoAdd$ represents the rules obtained by applying the courteous transforms 1j,2j,3j to rules in $NewNoOppoAdd$

Sketch of proof of correctness for transform 3 Lemma 4 characterizes the delete rule update and the proof is similar to the add rule case, as delete is the inverse of add.

Example 11 *If we start with the full rule base as in example 8 and remove either of the rules "rule_minorPain" or "rule_ulcer" the other rule becomes unopposed, i.e. the other rule becomes part of NewNoOppoAdd and the rules generated using transforms 1j,2j,3j,4j,5j must be deleted. The result will be that all the rules of the CLP will be replicated in the OLP.*

Transform 4 ***Courteous transform for a delete mutex update*** *Denoting the mutex update by δR the result of CR $(R + \delta R)$ can be expressed as*

$$CR\ (R-\delta R) = CR\ (R) \bigcup NewNoOppoAdd - \left(\bigcup_{rt \in RTGen_{Mi}} CR\ _{45}(rt) \bigcup TNewNoOppoAdd \right) \tag{6.10}$$

Sketch of proof of correctness for transform 4 From lemma 5, the deletion of a mutex will result in removal of only those RelOppTriples in which the mutex was a member of. If there were no such RelOppTriples, then $RTGen_{Mi}$, TNewNoOppoAdd, NewNoOppoAdd would be empty. The rest of the proof follows from the proof of the add mutex case.

Example 12 *Removal of mutex from example 8 will result in no opposers and the output as in the case of rule removal will just be the set of all rules in the CLP.*

6.4 Incremental algorithms

The incremental algorithms based on the incremental courteous transforms are described below. In general, these algorithms make use of the notion of semi-RelOppTriples by computing the semi-RelOppTriples for every Mutex and Rule pair. On a mutex or rule addition, the new RelOppTriples are computed by composing the semi-RelOppTriples. These algorithms use ordered mutexes, i.e. both a mutex and it's dual when computing the RelOppTriples. All these algorithms also rely on maintaining certain additional data structures between updates and that information includes a list of current RelOppTriples, a table of semiRelOppTriples indexed by Mutexes, a table of meta information about the predicates such as a count of the number of times it appears within the scope of cneg, a map of RelOppTriples to the output rules, a table of semiRelOppTriples organized and indexed by mutexes. These data structures are assumed to be global to the algorithms. In the description of these algorithms the phrase "SCLP Rulebase" will depending on the context, mean the SCLP Rulebase obtained after applying the transform to eliminate classical negation(ECN).

The algorithms are summarized, for details refer to the SweetRules V2.1 source code.

6.4.1 Rule addition

The incremental algorithm for compiling a newly rule is given in algorithm 2. It first runs the ECN transform on this new rule R_{new} to obtain R_k, and checks if an implicit mutex needs to be added for any of the predicates in the new rule. If required it adds them and compiles the new mutexes using the algorithm 3. The algorithm proceeds by incrementally computing the semi-RelOppTriples and using the semi-RelOppTriples computes the relOppTriples. If no opposers exist for the new rule, it is just replicated, else the courteous transforms are applied. In this process if any previously unopposed rule now has an opposer in the form of the rule being added, it is retracted from the output and courteous transforms 1j,2j,3j are performed on it.

Algorithm 2 Algorithm for incrementally compiling a new rule R_{new}=`{Rlab}` P_H`[`t_h`] :- B[`t_Y`]`, where Rlab is the rule label P_H is the head predicate, B is the body and t_Y is the tuple of all variables in B and t_h is the argument tuple.

1: Run ECN on R_{new} to obtain R_k
2: Add the Rule R_k to the SCLP rulebase
3: **for all** Predicates q part of R_k **do**
4: **if** q is in scope of CNEG for the first time **then**
5: Create an implicit mutex compile it with Algorithm 3
6: **end if**
7: **end for**
8: **if** Implicit mutex for P_H was added **then**
9: Add results courteous transforms 1j,2j,3j on R_k to the output.
10: Maintain a map of R_k and the resulting rules in the output
11: Return
12: **end if**
13: $semiRelOppTripleList \leftarrow \emptyset$
14: **for all** Mutexes M_i whose second opposer literal unifies with head of R_k **do**
15: Create a semi-RelOppTriple st from mutex M_i with rule R_k, and the unifier θ_{ik}
16: Add st to $semiRelOppTripleList$
17: Update the global Semi-RelOppTriple table
18: **end for**
19: $relOppTripleList \leftarrow \emptyset$
20: **for all** Semi-RelOppTriples st in $semiRelOppTripleList$ **do**
21: Get the mutex M_i of the semi-relOppTriple st
22: Get the dual of the mutex i.e. $Mdual_i$
23: **for all** Semi-relOppTriple $stDual$ corresponding to the mutex dual **do**
24: Get the rule R_j in $stDual$
25: **if** Can unify $< R_j, M_i, R_k >$ **then**
26: Add the above RelOppTriple to $relOppTripleList$

Also add $< R_k, Mdual_i, R_j >$ to $relOppTripleList$
Update the global RelOppTripleList with above.
end if
end for
end for
if No opposers exist for R_k **then**
replicate the rule in the output and mark it as unopposed
end if
if Opposers exist for R_k **then**
Add results courteous transforms applied to R_k 1j,2j,3j to the output.
Maintain a map of R_k and the resulting rules in the output
for all RelOppTriple rt in $relOppTripleList$ **do**
Add results courteous transforms 4j,5j to the output
Maintain a map of rt and the rules produced in the output
if The other rule R_j in rt is not equal to R_k, and was previously unopposed **then**
Remove the rule from the output, add result of courteous transforms 1j,2j,3j on R_j to the output.
Maintain a map of R_j and the resulting rules in the output
end if
end for
end if

6.4.2 Mutex addition

For a mutex addition, we compute the RelOppTriples by iterating over Rule locales of the opposer literals. The algorithm given below first performs the ECN transform, and adds any required implicit mutexes. It then iterates over the rule locales of the opposers to compute the new relevant opposition triples. Once the new relevant opposition triples are computed transforms 4j, 5j are applied to each of them. If any previously unopposed rule now has an opposer it is unmarked and transforms 1j, 2j, 3j are applied to it.

Algorithm 3 Algorithm for incrementally compiling a new mutex M_{new}=`!- P1(u1) and P2(u2) | Ei[Zi]`, where P1 is the first opposer literal, P2 is the second opposer literal, u1, u2 are arguments of opposers, Ei is the condition under which the oppostion holds, Zi is argument of the body of the condition

Run ECN on M_{new} to get M_i
Add the mutex M_i to the SCLP rulebase
for all Predicates q part of R_k **do**

4: **if** q is in scope of CNEG for the first time **then**
5: Create an implicit mutex, compile it with this algorithm
6: **end if**
7: **end for**
8: $relOppTripleList = \emptyset$
9: **for all** Rule R_k in the locale of second opposer of M_i **do**
10: Try to unify the second opposer with any rule head of R_k
11: **if** unifierFound **then**
12: Compute semi-relopptriple involving the mutex being added and the rule R_k
13: Update the global semi-RelOppTriple table
14: **for all** Rules R_j in the first opposer′s locale, try to find rules which fully unify with R_k **do**
15: **if** unifierFound **then**
16: Create the rel opp triple $rt \leftarrow < R_j, M_i, R_k >$
17: Add rt to $relOppTripleList$
18: Update global RelOppTripleList with rt
19: **end if**
20: **end for**
21: **end if**
22: **end for**
23: **for all** RelOppTriple rt in $relOppTripleList$ **do**
24: perform courteous transforms 4j,5j
25: Maintain a map of rt and the rules generated
26: **end for**
{Mutex addition might create new opposed rules}
27: **for all** Rules R_l part of some RelOppTriple in $relOppTripleList$ and previously unopposed **do**
28: Unmark R_l
29: Remove R_l from the output
30: Apply transforms 1j, 2j, 3j
31: Maintain a map R_l and the results of 1j,2j,3j
32: **end for**

6.4.3 Rule deletion

The rule deletion algorithm reverses the steps of Rule addition. As a first step it checks to see if the rule is unopposed, if it is unopposed, it is just removed and the algorithm terminates.
Otherwise, it proceeds by removing the rules generated by transforms 1j, 2j, 3j. For every RelOppTriple in which this rule is a member of, rules generated by transforms 4j, 5j are removed. Each opposer rule, is then checked to see if it has any other

opposers, if not results of running 1j, 2j, 3j on these opposers is reversed by deleting the generated rules from the output. This algorithm also handles removal of classical mutexes for predicates whose only occurence in the scope of classical negation after rule removal happens to be only in classical mutexes.

Algorithm 4 Algorithm for incrementally compiling a rule delete. Let the rule being deleted be R_k and be of the form {Rlab} P_H[t_h] :- B[t_Y], where Rlab is the rule label P_H is the head predicate, B is the body and t_Y is the tuple of all variables in B and t_h is the argument tuple.

1: **if** R_k is unopposed **then**
2: remove it from the SCLP rulebase and the output OLP rulebase
3: Return
4: **end if**
5: Remove the rules generated from R_k using transforms 1j,2j,3j
6: Update the map of R_k and the rules generated from transforms 1j,2j,3j
7: $impMutexRemovalList \leftarrow \emptyset$
8: **for all** p predicate in R_k **do**
9: **if** The only occurence of p within the scope of cneg, other than in R_k, is in classical mutexes **then**
10: Add p to $impMutexRemovalList$
11: **end if**
12: **end for**
13: **for all** p in $impMutexRemovaList$ **do**
14: Get the classical mutex M_{clas} corresponding to p
15: Apply the mutex removal algorithm to remove M_{clas}
16: **end for**
17: $candidateUnOpposedRules \leftarrow \emptyset$
18: **for all** RelOppTriples rt in global RelOppTripleList, in which R_k is present **do**
19: **if** Other opposer in rt say R_j is not same as R_k **then**
20: Add R_j to $candidateUnOpposedRules$
21: **end if**
22: Remove the rules generated by rt using transforms 4j,5j
23: Update the Map of RelOppTriples and rules generated from transforms 4j,5j
24: Update the global semi-RelOppTriple table corresponding to the mutex in rt
25: Delete rt from the global RelOppTripleList
26: **end for**
27: **for all** R_l in $candidateUnOpposedRules$ **do**
28: **if** R_l does not have any opposers other than R_k **then**
29: Remove the rules generated from R_l using transforms 1j,2j,3j
30: Update the map of R_l and the rules generated from transforms 1j,2j,3j

31: Mark R_l as unopposed
32: Replicate R_l in the output rulebase
33: **end if**
34: **end for**
35: Remove R_k from the SCLP rulebase

In the above algorithm the line 15 could be modified to just remove the mutex, instead of invoking the mutex deletion algorithm. If this is done the lines 7-16 should be moved after line 35 to ensure correctness.

6.4.4 Mutex deletion

The mutex deletion algorithm iterates over all the RelOppTriples in which the mutex is a member of and removes the result of transforms 4j and 5j. If the removal of the mutex makes some of the rules unopposed, the rules generated by applying transforms 1j, 2j, 3j on such rules are removed, and the rule is replicated in the output. The algorithm is then repeated for the dual of the mutex.

Algorithm 5 Algorithm for incremental compilation for a deleting a mutex M_i, which is of the form `!- P1(u1) and P2(u2) | Ei[Zi]`, where P1 is the first opposer literal, P2 is the second opposer literal, u1, u2 are arguments of opposers, Ei is the condition under which the oppostion holds, Zi is argument of the body of the condition

1: $candidateUnOpposedRules \leftarrow \emptyset$
2: **for all** rt such that rt is a relopptriple in which the mutex M_i is a member of **do**
3: remove the rules generated by rt by transforms 4j,5j from the compiled OLP
4: Update the map of rt and the rules generated by transforms 4j,5j
5: **for all** Rules r that are part of rt **do**
6: $candidateUnOpposedRules \leftarrow candidateUnOpposedRules \cup r$
7: **end for**
8: Update the global RelOppTriple List by removing the entry corresponding to rt
9: **end for**
10: Update the global semi-RelOppTriple table, by removing all semi-RelOppTriples in which M_i appears.
11: **for all** rc in $candidateUnOpposedRules$ **do**
12: **if** rc is not part of any relopptriple **then**
13: remove the rules generated from rc using transforms 1j, 2j and 3j
14: Update the Map of rc and the rules generated from 1j,2j,3j
15: Replicate rule rc in the output
16: Mark the rule as unopposed
17: **end if**

18: **end for**
19: repeat the above steps of the algorithm for the dual of the mutex
20: remove the mutex and it's dual from the SCLP rulebase

Chapter 7

Design of the Incremental inferencing algorithm for courteous logic programs

7.1 Inferencing after incremental updating

An incremental update such as a rule add (or mutex add) or rule delete (or mutex delete), will be subject to incremental compilation as explained in chapter 6 to produce an OLP. Once an OLP rulebase is produced it must be input to an inferencing engine which can then derive the conclusions entailed by the rulebase. There are several well-known alternatives for reasoning including:

1. Backward reasoning
2. Forward reasoning
3. Mixed reasoning

In each of these cases we can incrementally compile the update and do inferencing on the complete OLP. But, for the forward or mixed reasoning scenarios, the inference engines typically store the conclusions drawn from the previous OLP (i.e. the OLP compiled from the SCLP before the incremental update was processed). Therefore it is advantageous to compute only the modified conclusions. This is more challenging because a given update affects only a few of the old conclusions. In the rest of the chapter we address how to perform this task of incremental inferencing on an incremental update. We present a set of algorithms to narrow down the set of conclusions that need to be reconsidered and then recompute the ‘ ‘region” that needs to be modified.

7.2 Incremental forward reasoning

In order to support incremental reasoning, we first compile the incremental premises update using the techniques of chapter 6, especially the algorithms from section 6.4. This compilation determines the modifications to the output OLP. We add/delete each OLP rule as apppropriate and perform incremental forward inferencing on the OLP rulebase. In order to do this, we need maintain the predicate dependency graph (PDG) of the OLP and strongly connected components (SCC) of the PDG.

7.3 Correspondence of SCCs and the Well-Founded Model

If the rule base is predicate stratified, then the we can compute the Well-Founded model by iterating from the lowest stratum and proceeding towards higher strata. Similarly, for a general rule base we can build it's predicate dependency graph and determine the strongly connected components of the graph. Consider the graph in which every SCC is collapsed into a single node. The dependency graph for those nodes is acyclic. This means that we can compute the well-founded model one SCC at a time starting with SCCs that are not dependent on any other and then proceeding to other SCCs in the direction of dependency. On an incremental "add" update, we first build the new SCC graph from the rulebase which has the added premises and then recompute the conclusions starting from the SCC to which the update belongs and proceed to the other SCCs dependent on the affected SCC. A delete update is similar, except that we use the dependency information from the rulebase prior to performing the delete update.

7.4 Incremental reasoning algorithms

Each update to GCLP might cause multiple rules to be added or deleted from the OLP. Therefore we define a generic high level algorithm to perform incremental reasoning on a set of rule updates.

Algorithm 6 Incremental inferencing via SCC for a set of rules being added and a set of rules being deleted to the OLP post compilation

1: $R_{add} \leftarrow$ Set of rules being added to the OLP
2: $R_{del} \leftarrow$ Set of rules being deleted from the OLP
3: Add the rules in R_{add} to the OLP
4: Compute the new PDG and SCC taking into account rules in R_{add} (but without yet deleting the rules in R_{del}

5: RULHEAD ← set of all head predicates of rules in R_{add}
6: RULHEAD ← RULHEAD $\bigcup$ set of all head predicates of rules in R_{del}
7: Remove the rules in R_{del} from the OLP
{Incrementally compute the new conclusions, using the SCCs}
8: Remove the old conclusions from locales of predicates in $RULHEAD$ and those that depend on these predicates
9: Determine the SCCs corresponding to each of the predicates in the $RULHEAD$
10: Recompute the new conclusions starting from the SCCs which do not depend on any other SCC and proceed in the direction of dependency.
{Update the PDG and the SCC}
11: Compute the new PDG and SCCs considering the fact that rules in R_{del} have now been deleted

---

7.4.1 Incremental reasoning for Rule addition or deletion

First compile the rule being added/deleted and determine the set of rules to be added to the output OLP and the set of rules that have to be deleted from the OLP. Then apply algorithm 7.4 to get the new set of conclusions.

7.4.2 Incremental reasoning for mutex addition or deletion

Similar to Rule addition, first compile and then determine the set of rules to be added to output OLP and the set of rules being deleted from the OLP and apply algorithm 7.4.

Chapter 8

Future work

SweetRules can be made more expressive and powerful by enhancing it as described below.

1. Distributed inferencing: SweetRules currently runs on a standalone machine working with a single instance of inference engines. It will be interesting both theoretically and from an implmentation perspective to explore the issues in distributed inferencing.

2. Support for conflictful sensing: SweetRules assumes that sensing is always conflict free. But, there will be real world scenarios in which there are multiple sources for sensing and we need a protocol to resolve any conflicts that might arise.

3. Handling Equality in DLP: Handling equality will enable translation of DL constructs such as functional property and inverse functional property.

4. Support for Hi-Log and F-Logic: Supporting Hi-Log and F-Logic would enable interoperability with Flora [1].

5. General Lloyd Topor: The current implementation performs head conjunction elimination only. It will be useful to generalize to other cases such as elimination of body disjunction.

6. Generalized Mutexes: The current SCLP semantics is defined for mutexes with only two opposers. In general, we need to develop theory for more than two opposers.

7. Generalizing the incremental courteous compilation and inferencing to batch updates: For a batch update, the incremental compilation is supported by applying the courteous transforms as though a sequence of single rule/mutex updates were presented.

But if inferencing is also involved, we can save a lot of work if we perform the compilation of the entire batch and run incremental inferencing only once.

Chapter 9

Conclusion

In this thesis we outlined an approach to perform knowledge integration, evangelizing Well Founded Semantics and Situated and Courteous Logic Programs. We presented the architecture and technical details of the SweetRules platform which is capable merging heterogeneous knowledge bases, by performing appropriate translations emphasizing clean and uniform semantics. The primary contributions in terms of preserving semantics include handling of negation in Production Rules, and disambiguating the undefined truth value in XSB. We also showed how OWL ontologies within the DLP subset can be translated to RuleML, and then be merged with rules which refer to ontological entities. This elevates the static ontologies to be used in conjunction with rules.

In terms of extensibility, we showed how the SCLP RuleML can be enhanced to support web services and backward reasoning via the query and answer elements. We also proposed the design of an incremental courteous compiler and reasoner to facilitate development of applications for rule authoring and distributed inferencing.

Appendix

The Orderingleadtime example demonstrating KB merging

In this section the contents of the included files in example 1 are shown along with the result of the merge.

The RuleML rulebase has the following rules:

```
<?xml version="1.0" encoding="UTF-8"?>
<rulebase>
    <_rbaselab>
        <cterm>
            <_opc>
                <ctor>OrderingLeadTimeRules</ctor>
            </_opc>
        </cterm>
    </_rbaselab>
    <imp>
        <_head>
            <cslit cneg="no">
                <_opr>
                    <rel>orderModificationNotice</rel>
                </_opr>
                <var>Order</var>
                <cterm>
                    <_opc>
                        <ctor>days14</ctor>
                    </_opc>
                </cterm>
            </cslit>
        </_head>
        <_body>
            <andb>
                <fclit cneg="no" fneg="no">
                    <_opr>
                    <rel href="http://www.ordermanagement.org/
                                OrderingLeadTimeOnt.owl#preferredCustomerOf"/>
                    </_opr>
```

```
                <var>Buyer</var>
                <var>Seller</var>
            </fclit>
            <fclit cneg="no" fneg="no">
                <_opr>
                    <rel>purchaseOrder</rel>
                </_opr>
                <var>Order</var>
                <var>Buyer</var>
                <var>Seller</var>
            </fclit>
        </andb>
    </_body>
    <_rlab>
        <cterm>
            <_opc>
                <ctor>leadTimeRule1</ctor>
            </_opc>
        </cterm>
    </_rlab>
</imp>
<imp>
    <_head>
        <cslit cneg="no">
            <_opr>
                <rel>orderModificationNotice</rel>
            </_opr>
            <var>Order</var>
            <cterm>
                <_opc>
                    <ctor>days30</ctor>
                </_opc>
            </cterm>
        </cslit>
    </_head>
    <_body>
        <andb>
            <fclit cneg="no" fneg="no">
                <_opr>
                <rel href="http://www.ordermanagement.org/
                            OrderingLeadTimeOnt.owl#MinorPart"/>
                </_opr>
```

```
                <var>Order</var>
            </fclit>
            <fclit cneg="no" fneg="no">
                <_opr>
                    <rel>purchaseOrder</rel>
                </_opr>
                <var>Order</var>
                <var>Buyer</var>
                <var>Seller</var>
            </fclit>
        </andb>
    </_body>
    <_rlab>
        <cterm>
            <_opc>
                <ctor>leadTimeRule2</ctor>
            </_opc>
        </cterm>
    </_rlab>
</imp>
<imp>
    <_head>
        <cslit cneg="no">
            <_opr>
                <rel>orderModificationNotice</rel>
            </_opr>
            <var>Order</var>
            <cterm>
                <_opc>
                    <ctor>days2</ctor>
                </_opc>
            </cterm>
        </cslit>
    </_head>
    <_body>
        <andb>
            <fclit cneg="no" fneg="no">
                <_opr>
                <rel href="http://www.ordermanagement.org/
                            OrderingLeadTimeOnt.owl#preferredCustomerOf"/>
                </_opr>
                <var>Buyer</var>
```

```
                <var>Seller</var>
            </fclit>
            <fclit cneg="no" fneg="no">
                <_opr>
                    <rel>orderModificationType</rel>
                </_opr>
                <var>Order</var>
                <cterm>
                    <_opc>
                        <ctor>reduce</ctor>
                    </_opc>
                </cterm>
            </fclit>
            <fclit cneg="no" fneg="no">
                <_opr>
                    <rel>orderItemIsInBacklog</rel>
                </_opr>
                <var>Order</var>
            </fclit>
            <fclit cneg="no" fneg="no">
                <_opr>
                    <rel>purchaseOrder</rel>
                </_opr>
                <var>Order</var>
                <var>Buyer</var>
                <var>Seller</var>
            </fclit>
        </andb>
    </_body>
    <_rlab>
        <cterm>
            <_opc>
                <ctor>leadTimeRule3</ctor>
            </_opc>
        </cterm>
    </_rlab>
</imp>
<imp>
    <_head>
        <cslit cneg="no">
            <_opr>
                <rel>overrides</rel>
```

```
            </_opr>
            <cterm>
                <_opc>
                    <ctor>leadTimeRule3</ctor>
                </_opc>
            </cterm>
            <cterm>
                <_opc>
                    <ctor>leadTimeRule1</ctor>
                </_opc>
            </cterm>
        </cslit>
    </_head>
    <_rlab>
        <cterm>
            <_opc>
                <ctor>emptyLabel</ctor>
            </_opc>
        </cterm>
    </_rlab>
</imp>
<mutex>
    <_oppo>
        <ando>
            <cslit cneg="no">
                <_opr>
                    <rel>orderModificationNotice</rel>
                </_opr>
                <var>Order</var>
                <var>X</var>
            </cslit>
            <cslit cneg="no">
                <_opr>
                    <rel>orderModificationNotice</rel>
                </_opr>
                <var>Order</var>
                <var>Y</var>
            </cslit>
        </ando>
    </_oppo>
    <_mgiv>
        <fclit cneg="no" fneg="no">
```

```
                <_opr>
                    <rel>notEquals</rel>
                </_opr>
                <var>X</var>
                <var>Y</var>
            </fclit>
        </_mgiv>
    </mutex>
</rulebase>
```

The OWL ontology for the example is shown below:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#"
    xml:base="http://www.ordermanagement.org/OrderingLeadTimeOnt.owl">
  <owl:Ontology rdf:about=""/>


  <owl:class rdf:ID="Buyer"/>
  <owl:class rdf:ID="Seller"/>
  <owl:class rdf:ID="Product"/>

  <owl:class rdf:ID="MinorPart">
  <rdfs:subClassOf rdf:resource="#Product"/>
  </owl:class>

  <owl:ObjectProperty rdf:ID="preferredCustomerOf">
  <rdfs:domain rdf:resource="#Buyer"/>
  <rdfs:range rdf:resource="#Seller"/>
  </owl:ObjectProperty>

  <!--TODO: preferredCustomerOf should be preferredCustomer
    property stored in the seller -->
  <Seller rdf:ID="FrysElectronics"/>
  <Seller rdf:ID="CompUSA"/>

  <Buyer rdf:ID="IBM">
  <preferredCustomerOf rdf:resource="#FrysElectronics"/>
```

```
  <preferredCustomerOf rdf:resource="#CompUSA"/>
  </Buyer>
  <Buyer rdf:ID="ReqOrg">
  <preferredCustomerOf rdf:resource="#FrysElectronics"/>
  <preferredCustomerOf rdf:resource="#CompUSA"/>
  </Buyer>


  <Product rdf:ID="po1234567"/>

  <MinorPart rdf:ID="po3456789"/>

  <MinorPart rdf:ID="po7890123"/>

  <Product rdf:ID="po5678901"/>

</rdf:RDF>
```

The JESS facts are:

```
; purchaseOrder (productName, Buyer, Seller)
(assert (purchaseOrder
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#po1234567
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#ReqOrg
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#CompUSA))

(assert (purchaseOrder
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#po7890123
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#IBM
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#CompUSA))

(assert (purchaseOrder
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#po3456789
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#ReqOrg
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#CompUSA))

(assert (purchaseOrder
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#po5678901
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#IBM
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#CompUSA))

(assert (purchaseOrder
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#po1234567
```

```
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#ReqOrg
  http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#CompUSA))

; orderItemIsInBacklog (product)
(assert (orderItemIsInBacklog
   http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#po3456789))
(assert (orderItemIsInBacklog
   http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#po5678901))
(assert (orderModificationType
           http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#po3456789
   reduce))

(assert (orderModificationType
   http://www.ordermanagement.org/OrderingLeadTimeOnt.owl#po5678901
           reduce))
```

After executing KB Merging on the above KBs, by processing the "rbaseincludes" statements, we get the following combined KB:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rulebase>
    <_rbaselab>
        <ind/>
    </_rbaselab>
    <imp>
        <_head>
            <cslit cneg="no">
                <_opr>
                    <rel>orderModificationNotice</rel>
                </_opr>
                <var>Order</var>
                <cterm>
                    <_opc>
                        <ctor>days14</ctor>
                    </_opc>
                </cterm>
            </cslit>
        </_head>
        <_body>
            <andb>
                <fclit cneg="no" fneg="no">
                    <_opr>
```

```
                        <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#preferredCustomerOf"/>
                    </_opr>
                    <var>Buyer</var>
                    <var>Seller</var>
                </fclit>
                <fclit cneg="no" fneg="no">
                    <_opr>
                        <rel>purchaseOrder</rel>
                    </_opr>
                    <var>Order</var>
                    <var>Buyer</var>
                    <var>Seller</var>
                </fclit>
            </andb>
        </_body>
        <_rlab>
            <cterm>
                <_opc>
                    <ctor>leadTimeRule1</ctor>
                </_opc>
            </cterm>
        </_rlab>
    </imp>
    <imp>
        <_head>
            <cslit cneg="no">
                <_opr>
                    <rel>orderModificationNotice</rel>
                </_opr>
                <var>Order</var>
                <cterm>
                    <_opc>
                        <ctor>days30</ctor>
                    </_opc>
                </cterm>
            </cslit>
        </_head>
        <_body>
            <andb>
                <fclit cneg="no" fneg="no">
                    <_opr>
```

```
                        <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#MinorPart"/>
                    </_opr>
                    <var>Order</var>
                </fclit>
                <fclit cneg="no" fneg="no">
                    <_opr>
                        <rel>purchaseOrder</rel>
                    </_opr>
                    <var>Order</var>
                    <var>Buyer</var>
                    <var>Seller</var>
                </fclit>
            </andb>
        </_body>
        <_rlab>
            <cterm>
                <_opc>
                    <ctor>leadTimeRule2</ctor>
                </_opc>
            </cterm>
        </_rlab>
    </imp>
    <imp>
        <_head>
            <cslit cneg="no">
                <_opr>
                    <rel>orderModificationNotice</rel>
                </_opr>
                <var>Order</var>
                <cterm>
                    <_opc>
                        <ctor>days2</ctor>
                    </_opc>
                </cterm>
            </cslit>
        </_head>
        <_body>
            <andb>
                <fclit cneg="no" fneg="no">
                    <_opr>
                        <rel href="http://www.ordermanagement.org/
```

```
OrderingLeadTimeOnt.owl#preferredCustomerOf"/>
                    </_opr>
                    <var>Buyer</var>
                    <var>Seller</var>
                </fclit>
                <fclit cneg="no" fneg="no">
                    <_opr>
                        <rel>orderModificationType</rel>
                    </_opr>
                    <var>Order</var>
                    <cterm>
                        <_opc>
                            <ctor>reduce</ctor>
                        </_opc>
                    </cterm>
                </fclit>
                <fclit cneg="no" fneg="no">
                    <_opr>
                        <rel>orderItemIsInBacklog</rel>
                    </_opr>
                    <var>Order</var>
                </fclit>
                <fclit cneg="no" fneg="no">
                    <_opr>
                        <rel>purchaseOrder</rel>
                    </_opr>
                    <var>Order</var>
                    <var>Buyer</var>
                    <var>Seller</var>
                </fclit>
            </andb>
        </_body>
        <_rlab>
            <cterm>
                <_opc>
                    <ctor>leadTimeRule3</ctor>
                </_opc>
            </cterm>
        </_rlab>
    </imp>
    <imp>
        <_head>
```

```
            <cslit cneg="no">
                <_opr>
                    <rel>overrides</rel>
                </_opr>
                <cterm>
                    <_opc>
                        <ctor>leadTimeRule3</ctor>
                    </_opc>
                </cterm>
                <cterm>
                    <_opc>
                        <ctor>leadTimeRule1</ctor>
                    </_opc>
                </cterm>
            </cslit>
        </_head>
        <_rlab>
            <cterm>
                <_opc>
                    <ctor>emptyLabel</ctor>
                </_opc>
            </cterm>
        </_rlab>
    </imp>
    <mutex>
        <_oppo>
            <ando>
                <cslit cneg="no">
                    <_opr>
                        <rel>orderModificationNotice</rel>
                    </_opr>
                    <var>Order</var>
                    <var>X</var>
                </cslit>
                <cslit cneg="no">
                    <_opr>
                        <rel>orderModificationNotice</rel>
                    </_opr>
                    <var>Order</var>
                    <var>Y</var>
                </cslit>
            </ando>
```

```
        </_oppo>
        <_mgiv>
            <fclit cneg="no" fneg="no">
                <_opr>
                    <rel>notEquals</rel>
                </_opr>
                <var>X</var>
                <var>Y</var>
            </fclit>
        </_mgiv>
    </mutex>
    <imp>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#Buyer"/>
                </_opr>
                <var>X</var>
            </atom>
        </_head>
        <_body>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#preferredCustomerOf"/>
                </_opr>
                <var>X</var>
                <var>Y</var>
            </atom>
        </_body>
    </imp>
    <imp>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#Seller"/>
                </_opr>
                <var>Y</var>
            </atom>
        </_head>
```

```
        <_body>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#preferredCustomerOf"/>
                </_opr>
                <var>X</var>
                <var>Y</var>
            </atom>
        </_body>
    </imp>
    <imp>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#Product"/>
                </_opr>
                <var>X</var>
            </atom>
        </_head>
        <_body>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#MinorPart"/>
                </_opr>
                <var>X</var>
            </atom>
        </_body>
    </imp>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#Product"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#po5678901"/>
            </atom>
        </_head>
```

```
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#Product"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#po1234567"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#Seller"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#FrysElectronics"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#Seller"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#CompUSA"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
```

```
OrderingLeadTimeOnt.owl#Buyer"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#ReqOrg"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#Buyer"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#IBM"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#MinorPart"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#po3456789"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#MinorPart"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#po7890123"/>
            </atom>
        </_head>
```

```
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#preferredCustomerOf"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#IBM"/>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#CompUSA"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#preferredCustomerOf"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#ReqOrg"/>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#FrysElectronics"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#preferredCustomerOf"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#IBM"/>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#FrysElectronics"/>
            </atom>
        </_head>
```

```
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#preferredCustomerOf"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#ReqOrg"/>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#CompUSA"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.w3.org/2002/07/owl#class"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#MinorPart"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.w3.org/2002/07/owl#class"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#Seller"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.w3.org/2002/07/owl#class"/>
```

```
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#Product"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <atom>
                <_opr>
                    <rel href="http://www.w3.org/2002/07/owl#class"/>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#Buyer"/>
            </atom>
        </_head>
    </fact>
    <fact>
        <_head>
            <cslit cneg="no">
                <_opr>
                    <rel>purchaseOrder</rel>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#po1234567"/>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#ReqOrg"/>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#CompUSA"/>
            </cslit>
        </_head>
    </fact>
    <fact>
        <_head>
            <cslit cneg="no">
                <_opr>
                    <rel>purchaseOrder</rel>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#po7890123"/>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#IBM"/>
```

```
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#CompUSA"/>
            </cslit>
        </_head>
    </fact>
    <fact>
        <_head>
            <cslit cneg="no">
                <_opr>
                    <rel>purchaseOrder</rel>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#po3456789"/>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#ReqOrg"/>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#CompUSA"/>
            </cslit>
        </_head>
    </fact>
    <fact>
        <_head>
            <cslit cneg="no">
                <_opr>
                    <rel>purchaseOrder</rel>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#po5678901"/>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#IBM"/>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#CompUSA"/>
            </cslit>
        </_head>
    </fact>
    <fact>
        <_head>
            <cslit cneg="no">
                <_opr>
                    <rel>orderItemIsInBacklog</rel>
                </_opr>
                <ind href="http://www.ordermanagement.org/
```

```
OrderingLeadTimeOnt.owl#po3456789"/>
            </cslit>
        </_head>
    </fact>
    <fact>
        <_head>
            <cslit cneg="no">
                <_opr>
                    <rel>orderItemIsInBacklog</rel>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#po5678901"/>
            </cslit>
        </_head>
    </fact>
    <fact>
        <_head>
            <cslit cneg="no">
                <_opr>
                    <rel>orderModificationType</rel>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#po3456789"/>
                <ind>reduce</ind>
            </cslit>
        </_head>
    </fact>
    <fact>
        <_head>
            <cslit cneg="no">
                <_opr>
                    <rel>orderModificationType</rel>
                </_opr>
                <ind href="http://www.ordermanagement.org/
OrderingLeadTimeOnt.owl#po5678901"/>
                <ind>reduce</ind>
            </cslit>
        </_head>
    </fact>
</rulebase>
```

Bibliography

[1] Flora - an object-oriented knowledge base language. http://flora.sourceforge.net/.

[2] Benjamin N. Grosof an Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 48–57, New York, NY, USA, 2003. ACM Press.

[3] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. *J. Log. Program.*, 10(3-4):255–299, 1991.

[4] Carlos Viegas Damsio. The w4 project well-founded semantics for the world wide web. http://centria.di.fct.unl.pt/ cd/projectos/w4/.

[5] Charles L. Forgy. Rete: A fast algorithm for the many pattern/ many object pattern match problem. *Journal of Artificial Intelligence*, 19:17–37, 1982.

[6] A. V. Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38:620–650, 1991.

[7] A. Van Gelder. The alternating fixpoint of logic programs with negation. In *PODS '89: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 1–10, New York, NY, USA, 1989. ACM Press.

[8] Benjamin N. Grosof. Courteous logic programs: Prioritized conflict handling for rules. Technical report, IBM T.J. Watson Research Center, http://www.research.ibm.com , search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, Dec. 1997. IBM Research Report RC 20836., 1997.

[9] Benjamin N. Grosof. Prioritized conflict handling for logic programs. In *Logic Programming: Proceedings of the International Symposium (ILPS-97), (Also, an extended version is available as IBM Research Report RC 20836 at http://www.research.ibm.com)*, pages 197–211, 1997.

[10] Benjamin N. Grosof. Compiling prioritized default rules into ordinary logic programs, 1999.

[11] Benjamin N. Grosof. A courteous compiler from generalized courteous logic programs to ordinary logic programs, 1999.

[12] Benjamin N. Grosof. Representing e-commerce rules via situated courteous logic programs in ruleml. *Electronic Commerce Research and Applications*, 3(1):2–20, 2004.

[13] Benjamin N. Grosof, Mahesh D. Gandhe, and Timothy W. Finin. Sweetjess: Translating damlruleml to jess. In *RuleML*, 2002.

[14] Benjamin N. Grosof, Mahesh D. Gandhe, and Timothy W. Finin. Sweetjess: Inferencing in situated courteous ruleml via translation to and from jess rules. 2003.

[15] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an owl rules language. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 723–731, New York, NY, USA, 2004. ACM Press.

[16] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an owl rules language. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 723–731, New York, NY, USA, 2004. ACM Press.

[17] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. Swrl: A semantic web rule language combining owl and ruleml. http://www.daml.org/2003/11/swrl/, 2003.

[18] David B. Kemp, Divesh Srivastava, and Peter J. Stuckey. Magic sets and bottom-up evaluation of well-founded models. In *ISLP*, pages 337–351, 1991.

[19] David B. Kemp, Divesh Srivastava, and Peter J. Stuckey. Bottom-up evaluation and query optimization of well-founded models. *Theor. Comput. Sci.*, 146(1-2):145–184, 1995.

[20] J. W. Lloyd. *Foundations of logic programming; (2nd extended ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[21] Teodor C. Przymusinski. Well-founded and stationary models of logic programs. *Ann. Math. Artif. Intell.*, 12(3-4):141–187, 1994.

[22] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. Owl web ontology language guide. http://www.w3.org/TR/2004/REC-owl-guide-20040210/, 2004.

[23] Jeffrey D. Ullman. *Principles of database and knowledge-base systems, Vol. I.* Computer Science Press, Inc., New York, NY, USA, 1988.