CS6795 Semantic Web Techniques
Project Report

XSLT 2.0 Translation of Datalog$^+$ RuleML 1.01/XML to a Subset of the TPTP Language

Meng Luan and Changyang Liu

University of New Brunswick, Faculty of Computer Science

December 17th, 2014

Professor: Dr. Harold Boley
Advisor: Dr. Tara Athan

1. Introduction

This project is aimed at implementing an XSLT 2.0 translator to convert Datalog$^+$ Deliberation RuleML 1.01 in XML format to an equivalent representation in a subset of the TPTP (Thousands of Problems for Theorem Provers) language. The following sections in this chapter introduce some of the languages that this project uses and the objectives of this project. Chapter 2 describes the methodology for developing this project and for solving the problems we encountered. The results of the project are provided in Chapter 3, and we conclude with our major work in this project and possible future work in Chapter 4. The project website [6] provides the distribution and the full documentation of this project.

1.1. Languages

Several languages are in the center of this project. RuleML and the TPTP language are respectively the source and the target languages of the translation. XSLT is the language that we use to implement the translator. Each language is introduced briefly in the following.

RuleML. RuleML (Rule Markup Language) is a markup language designed in an XML format. It is developed by an open non-profit organization of the same name, to provide a uniform representation for all kinds of relevant rule languages. As part of semantic technology efforts, the RuleML specification has become a de facto standard for Web rule knowledge representation. In both industry and academia, RuleML is used to bridge and exchange knowledge bases and queries among different systems [3], so the translation between RuleML and other rule-based languages is prevalent. The specification of RuleML pertinent to this project is Deliberation RuleML 1.01, which is the currently released Deliberation version. Deliberation RuleML 1.01 has broad coverage across various rule logics and has a modular schema system that allows fine-grained customization [4]. This project has its focus on Datalog$^+$ Deliberation RuleML 1.01, especially on the following three defining Datalog$^+$ extensions [1], which can also be used in Hornlog$^+$:

- Existential Rules, where variables in rule conclusions are existentially quantified.
- Equality Rules, where the binary "Equal" predicate is applied in rule conclusions.
- Integrity Rules, which use the empty "Or" in rule conclusions to provide a convenient way for expressing falsity.

All these extensions have their equivalent representation in the target language of the translation in this project. The target language is a subset of the (full) first-order form (FOF) of the TPTP language.

The TPTP language. TPTP is a comprehensive library of automated theorem proving (ATP) test problems, which are available online. Its motivation is to provide support for the testing and evaluation of ATP systems [7]. The TPTP site also hosts online services for solving problems. These services are provided by the numerous individually developed ATP systems (or the TPTP systems) accessible from the TPTP portal. Standard input format is enforced by all the TPTP systems, whereby the TPTP language is defined. The input of a TPTP system in the form of the TPTP language is also called a TPTP problem, which consists of a number of TPTP formulae. A TPTP formula can be an "axiom", corresponding to the `<formula>` under `<Assert>` in RuleML, or a "conjecture", corresponding to the `<formula>` under `<Query>` in RuleML.

XSLT. XSLT (Extensible Stylesheet Language Transformations) Version 2.0 is a language to describe the ways to transform XML documents into other XML, HTML or plain text [2]. An XSLT document works as the input of an XSLT processor, together with the source XML file to be translated. The XSLT processor is typically an executable program. A new document is generated by the XSLT processor based on the content of the source XML and the definitions in the input XSLT document, which is called an XSLT translator in this report.

1.2. Objectives

As mentioned before, the task of this project is to implement an XSLT 2.0 translator to convert Datalog$^+$ Deliberation RuleML 1.01[1] knowledge bases[2] into the representation of the TPTP language, focusing on the three Datalog$^+$ extensions. As the main goal of

[1] In the following parts of this report the term "RuleML" is also used as a shortcut for "Datalog$^+$ Deliberation RuleML 1.01" when allowed by the context.

[2] Considered as the input of the translator, RuleML knowledge bases are called RuleML instances or RuleML documents in this report.

this work, the translator should work correctly. In other words, the TPTP output of the translator should be accepted and solved by the TPTP systems without error, provided the RuleML input document is properly composed. The elements in the schema of Datalog$^+$ Deliberation RuleML 1.01 should be supported by the translation as much as possible, and any deviations should be documented.

Besides, we strive to render the output TPTP documents in a "pretty-printed" manner, i.e., with proper indentation, spacing and line breaking, to enhance the human readability. The following[3] is a sample formula in the TPTP language to illustrate the layout of pretty-printing:

```
fof(property_of_integer, axiom, (
    ! [X] :
      ( integer(X)
     => ( even(X)
        | odd(X) ) )
)).
```

The output TPTP formulae in this project always start with "`fof`". In the above TPTP formula, "`!`" stands for universal quantification, "`=>`" stands for implication, and "`|`" stands for disjunction [8]. This formula means that if `X` is an integer, it is either even or odd (technically, because "`|`" is inclusive, `X` could be both even and odd).

Our last goal is to retain comments in the translation from RuleML to the TPTP language. RuleML uses XML comments and the TPTP language has its own line commenting syntax starting with a "%" (TPTP systems also support another syntax for block comments, however, we prefer to use line comments in this project). A comment in the TPTP language extends from the first "%" to the end of the line [8]. Thus, the following RuleML comment

```
<!-- A RuleML comment
is also an XML comment.
-->
```

will be converted into the following form in the TPTP language:

```
% A RuleML comment
% is also an XML comment.
```

[3]The RuleML input rule for producing this TPTP formula is given in Section 2.1.

2. Methodology

The procedure of the translation in this project consists of two phases as illustrated in Figure 2.1. The first phase is to normalize the input RuleML document by another XSLT translator, which is called the normalizer. The implementation of the normalizer itself is not part of this project. In the second phase, the XSLT translator[1] implemented in the project is invoked to generate the output in the form of the TPTP document from the normalized form of the input RuleML document.

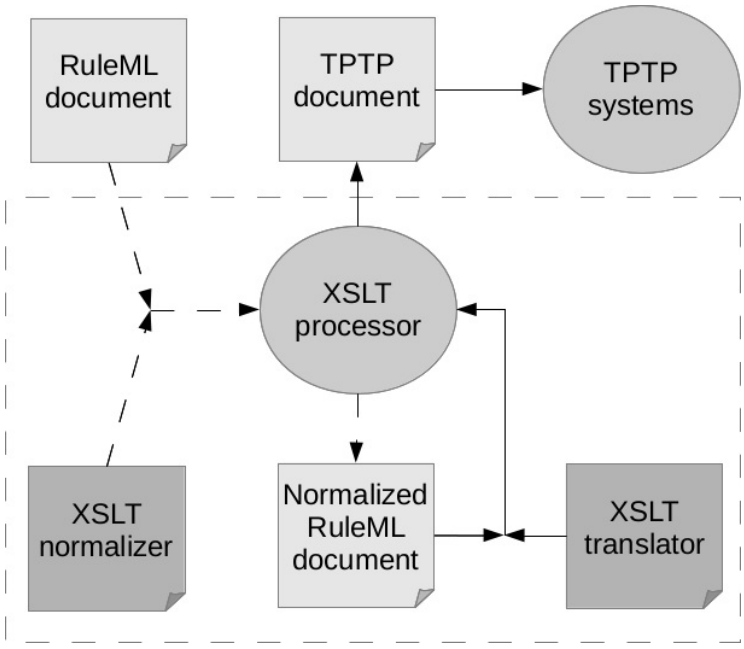

Figure 2.1.: The process of translation, where the dashed arrows indicate normalization.

[1]The term "XSLT translator" used in this report here and later on will be dedicated to denote the specific translator that is implemented in this project. The XSLT normalizer, which is also an XSLT translator in the general sense, will always be referred to as the "normalizer".

2.1. Normalization

In Deliberation RuleML 1.01 some elements can be reconstructed from other ones and thus can be omitted for brevity. The omittable elements are edge elements (with first letters in lowercase). In contrast, Node elements (with first letters in uppercase) cannot be omitted. The following[2] rule in RuleML is an example of the stripe-skipped form, i.e. without edge elements but with only Node elements:

```
<RuleML xmlns="http://ruleml.org/spec">
  <Assert>
    <Forall>
      <Var>x</Var>
      <Implies>
        <Atom>
          <Rel>integer</Rel>
          <Var>x</Var>
        </Atom>
        <Or>
          <Atom>
            <Rel>even</Rel>
            <Var>x</Var>
          </Atom>
          <Atom>
            <Rel>odd</Rel>
            <Var>x</Var>
          </Atom>
        </Or>
      </Implies>
    </Forall>
  </Assert>
</RuleML>
```

For example, in the above rule the first subelement of `<Implies>`, i.e. `<Atom>`, constitutes the premise, and the next subelement, `<Or>`, is the conclusion. Their roles under `<Implies>` are determined by their positions. The fully-striped form of the above rule, i.e. with all edge elements in place (which will be the outcome of the normalizer used in this project) is as follows:

```
<RuleML xmlns="http://ruleml.org/spec">
  <act index="1">
```

[2]This rule is the RuleML input producing the TPTP formula in Section 1.2.

```
    <Assert mapMaterial="yes" mapDirection="bidirectional">
      <formula>
        <Forall>
          <declare>
            <Var>x</Var>
          </declare>
          <formula>
            <Implies material="yes" direction="bidirectional">
              <if>
                <Atom>
                  <op><Rel>integer</Rel></op>
                  <arg index="1"><Var>x</Var></arg>
                </Atom>
              </if>
              <then>
                <Or>
                  <formula>
                    <Atom>
                      <op><Rel>even</Rel></op>
                      <arg index="1"><Var>x</Var></arg>
                    </Atom>
                  </formula>
                  <formula>
                    <Atom>
                      <op><Rel>odd</Rel></op>
                      <arg index="1"><Var>x</Var></arg>
                    </Atom>
                  </formula>
                </Or>
              </then>
            </Implies>
          </formula>
        </Forall>
      </formula>
    </Assert>
  </act>
</RuleML>
```

Note that the reconstructed elements, e.g. `<if>` and `<then>`, reveal roles explicitly without relying on their positions within `<Implies>`[3].

[3]The reconstructed attributes, e.g. `@material` and `@direction`, are not needed in our development.

All in all, Deliberation RuleML 1.01 allows freedom to some extent of skipping edges, defaulting attribute values, and reordering elements. When the translation from RuleML is performed, such freedom should be taken into account. However, a normalized RuleML document with all edge elements reconstructed and canonical ordering of elements established will significantly relieve the XSLT translator from dealing with positional semantics. This is the reason why the normalization phase performs preprocessing for the translation. Note that RuleML provides several XSLT normalizers which we could utilize. The normalizer that we adopt in this project is actually one[4] for a superset of Datalog$^+$ logic, however, it serves well for this project.

2.2. XSLT Translator

Taking advantage of RuleML normalization discussed in the previous section, sophisticated conditional branching between stripe-skipped and striped forms has been avoided in the implementation of the XSLT translator Hence we developed the translator following more closely to the "push style" of XSLT, as opposed to its "pull style". The pull style usually selects XML elements from the source document iteratively, and thus uses less XSLT templates. In contrast, the push style prefers to write more XSLT templates for the XML elements in the source document, and to apply the templates recursively. In general, the push style makes an XSLT document better structured, and it is the style that we adopted in this project.

Considering the scope of this project, any elements beyond Datalog$^+$ expressivity such as `<Fun>` and `<Expr>` are ignored by not applying any XSLT template to them. Most elements defined in the schema of Datalog$^+$ Deliberation RuleML 1.01 can be translated into proper components of the TPTP language. The RuleML element for which the TPTP language has no equivalent counterpart is `<Retract>`, which is also ignored in the translation[5]. In addition, each TPTP formula has a unique name in the TPTP document, but rules in RuleML do not require names. We assume unnamed RuleML rules and use the following XSLT code to generate a unique name for every TPTP formula:

```
<!-- Formula name. -->
<xsl:value-of select="concat(
  $formula-source, '_in_act', $act-index, '_formula',
```

[4]The normalizer document named 101_nafneghornlogeq_normalizer.xslt used in this project can be found at `http://deliberation.ruleml.org/1.01/xslt/normalizer`, visited on December 3rd, 2014.

[5]It would be possible to instead produce a warning or error message like "<Retract> not allowed in TPTP" in the manner of the XSLT-implemented Schematron (`http://www.schematron.com`), but this would cross the boundary between a translator (our task) and a validator for TPTP/RuleML (a quite different task).

```
  count(preceding-sibling::r:formula) + 1)"/>
```

The above code is in the XSLT template matching the top level `<formula>` under `<Assert>` or `<Query>`. The string-type parameter `formula-source` of this template has the value of either "assert" or "query", depending on the parent of this `<formula>`. The parameter `act-index` has the same value as the attribute "index" (see the example of normalized RuleML in Section 2.1) of the ancestor element `<act>`. The value (normally an integer) of this optional attribute should be unique for each `<act>` in the RuleML document, and can be reconstructed by the normalization phase if it is not provided in the original input RuleML document. The generated name takes the form of "assert_in_act<*M*>_formula<*N*>" or "query_in_act<*M*>_formula<*N*>", where <*M*> is substituted for the value of attribute "index" and <*N*> is substituted for by the sequential index of the current `<formula>`, starting from 1, among all the elements of `<formula>` at the same level.

During the development of the XSLT translator, another problem we noticed is that the text contents of `<Rel>`, `<Var>` and `<Ind>` may not follow the required syntax by the TPTP language when they are used respectively as functors, variables and constants in the output document. In the TPTP language, functors and constants must be character sequences starting with a lowercase letter and containing only alphanumeric characters, or single-quoted character sequences that contain only a set of printable characters. Backslashes and apostrophe characters (or single quotation marks) must be escaped by a leading backslash when they are single-quoted. Variables in the TPTP language must be character sequences starting with an uppercase letter and containing only alphanumeric characters [8]. We make the translator convert the first letter of the text contents of `<Rel>`, `<Var>` and `<Ind>` into the proper case if the text contains only alphanumeric characters. Besides, if the text contents of `<Rel>` and `<Ind>` contain non-alphanumeric characters, they will be escaped by inserting backslashes properly and then single-quoted. The converted text as described above is used as functors, variables or constants in the output TPTP document, but note that it still may not conform to the TPTP syntax.

2.3. Java Wrapper

In this project, to perform the XSLT transformation, we use the Saxon XSLT processor, which can be either invoked as a Java library or run as an executable Java program [5]. However, our translation combines two phases of XSLT transformation, firstly the normalization phase and secondly the translation phase, so calling the Saxon XSLT processor directly would lead to much typing and memorizing. Also we preferred a platform-independent solution rather than maintaining multiple solutions for different platforms. Therefore we developed a Java program to wrap the functionalities of the XSLT normal-

izer and the XSLT translator into a single executable. This Java program provides specific command-line options for the translation task, reads the input RuleML document and generates the output according to the specified options. Usage instructions and more details about the Java program are provided on our project website [6].

3. Results

Our XSLT translator as well as the Java wrapper works perfectly with all the Datalog$^+$ examples[1] accompanying the release of Deliberation RuleML 1.01. The project and the documentation are published on GitHub[2]. The TPTP output documents of our translator are well-formed and pretty-printed, and can be executed by TPTP-accepting (FOL) provers such as Vampire 3.0[3]. Here is another introductory example, showing the translation of the three defining Datalog$^+$ extensions mentioned in Section 1.1. The following document is the original RuleML input:

```
<RuleML xmlns="http://ruleml.org/spec">
<!--
     This document is an example to illustrate existential,
     equality, and integrity rules.
-->
  <Assert>
    <!-- Existential:
         Every human being has (at least) a mother.  -->
    <Forall>
      <Var>H</Var>
      <Implies>
        <Atom>
          <Rel>human</Rel>
          <Var>H</Var>
        </Atom>
        <Exists>
          <Var>M</Var>
          <Atom>
            <Rel>hasMother</Rel>
            <Var>H</Var>
            <Var>M</Var>
          </Atom>
```

[1] `http://deliberation.ruleml.org/1.01/exa/DatalogPlus`, visited on December 17th, 2014.
[2] `https://github.com/EdmonL/RuleML2TPTP`, visited on December 17th, 2014.
[3] `http://www.cs.miami.edu/~tptp/cgi-bin/SystemOnTPTP`, visited on December 17th, 2014.

```
      </Exists>
    </Implies>
  </Forall>
</Assert>
<Assert>
  <!-- Equality:
       Every human being has only one mother.  -->
  <Forall>
    <Var>H</Var>
    <Var>M1</Var>
    <Var>M2</Var>
    <Implies>
      <And>
        <Atom>
          <Rel>hasMother</Rel>
          <Var>H</Var>
          <Var>M1</Var>
        </Atom>
        <Atom>
          <Rel>hasMother</Rel>
          <Var>H</Var>
          <Var>M2</Var>
        </Atom>
      </And>
      <Equal>
        <Var>M1</Var>
        <Var>M2</Var>
      </Equal>
    </Implies>
  </Forall>
</Assert>
<Assert>
  <!-- Integrity:
       No one can be their own mother.  -->
  <Forall>
    <Var>H</Var>
    <Implies>
      <Atom>
        <Rel>hasMother</Rel>
        <Var>H</Var>
        <Var>H</Var>
      </Atom>
```

```
        <Or/>
      </Implies>
    </Forall>
  </Assert>
</RuleML>
```

Note that the above document contains comments. The TPTP output translated from this RuleML document is as follows:

```
% This document is an example to illustrate existential,
% equality, and integrity rules.
% Existential:
% Every human being has (at least) a mother.
fof(assert_in_act1_formula1, axiom, (
    ! [H] :
      ( human(H)
     => ? [M] : hasMother(H, M) )
)).
% Equality:
% Every human being has only one mother.
fof(assert_in_act2_formula1, axiom, (
    ! [H, M1, M2] :
      ( ( hasMother(H, M1)
        & hasMother(H, M2) )
     => M1 = M2 )
)).
% Integrity:
% No one can be their own mother.
fof(assert_in_act3_formula1, axiom, (
    ! [H] :
      ( hasMother(H, H)
     => $false )
)).
```

For another, complex example showing the translation of all kinds of Datalog$^+$ rules, see Appendix A and Appendix B.

4. Conclusion

In this project, we developed a translator which utilizes XSLT 2.0 and which takes Datalog$^+$ RuleML documents as input, and outputs equivalent TPTP-FOF documents. We remark that we have achieved all the objectives discussed in Section 1.2. The major work we have done in this project includes:

- Using an XSLT normalizer to preprocess Datalog$^+$ RuleML input so that the complexity of the XSLT translator is reduced.
- Developing the translator in XSLT 2.0 in push style.
- Making the TPTP output pretty-printed to improve human readability.
- Developing an Java wrapper to combine all the processes of the translation into a single executable.
- Publishing and maintaining the project on GitHub, open-source.

4.1. Future Work

Future work regarding this project may focus on supporting more expressive RuleML sublanguages, and on applying the translation tool to large rulebases. Some specific future work may include:

- Composing RuleML2TPTP with translators targeting RuleML/XML.
- Making it a "Save as ..." option of other RuleML tools (e.g., editors and engines).
- Extending the expressivity of the RuleML input sublanguage from Datalog$^+$ to Hornlog$^+$.

- Further extensions allowing first-order logic RuleML input and even higher-order logic RuleML.

- Considering an inverse TPTP2RuleML translator (with increasing expressivity subsets) for the TPTP-to-RuleML direction.

Bibliography

[1] Harold Boley. CS 6795 Semantic Web Techniques - Fall 2014 Projects. http://www.cs.unb.ca/~boley/cs6795swt/fall2014projects.html. Visited on December 17th, 2014.

[2] The World Wide Web Consortium. XSL Transformations (XSLT) Version 2.0. http://www.w3.org/TR/xslt20. Visited on December 17th, 2014.

[3] RuleML Inc. RuleML Home. http://wiki.ruleml.org/index.php/RuleML_Home. Visited on December 17th, 2014.

[4] RuleML Inc. Specification of Deliberation RuleML 1.01. http://wiki.ruleml.org/index.php/Specification_of_Deliberation_RuleML_1.01. Visited on December 17th, 2014.

[5] Saxonica Ltd. SAXON - the xslt and xquery processor. http://saxon.sourceforge.net/. Visited on December 17th, 2014.

[6] Meng Luan. RuleML2TPTP project entry page. http://edmonl.github.io/RuleML2TPTP. Visited on December 17th, 2014.

[7] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[8] Geoff Sutcliffe, Stephan Schulz, Koen Claessen, and Allen Van Gelder. Using the TPTP Language for Writing Derivations and Finite Interpretations. In *Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*, pages 67–81. Springer Berlin Heidelberg, 2006.

A. The Business Scenario Example in RuleML

The following RuleML knowledge base is an instructive example[1] accompanying the Deliberation RuleML 1.01 release:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://deliberation.ruleml.org/1.01/relaxng/
   datalogplus_min_relaxed.rnc"?>
<RuleML xmlns="http://ruleml.org/spec">
  <!-- Some of these examples are from
    "A general Datalog-based framework for tractable query answering
    over ontologies",
    Andrea Calì,
    Georg Gottlob
    Thomas Lukasiewicz
    http://dx.doi.org/10.1016/j.websem.2012.03.001
        (preprint at http://www.websemanticsjournal.org/index.php/ps/
   article/view/289)
    -->

  <!-- This RuleML Document incrementally asserts into, retracts from and
       queries a rulebase (within the <RuleML> element) for a total of
       28 transactions: 13 Asserts, 2 Retracts and 13 Queries.
       Each Query demonstrates the semantics of the previous Assert or
   Retract
       by providing (in the XML comments) the expected answer to the Query
   .
    -->
  <Assert>
    <!--
         Equations are allowed as facts:
```

[1] http://deliberation.ruleml.org/1.01/exa/DatalogPlus/datalogplus_min.ruleml, visited on December 17th, 2014.

```
        William is an employee.
        "Bill" is an alias for "William".
        Sublanguage: datagroundfacteq
      -->
    <Atom>
      <Rel>employee</Rel>
      <Ind>William</Ind>
    </Atom>
    <Equal>
      <Ind>Bill</Ind>
      <Ind>William</Ind>
    </Equal>
</Assert>
<Query>
    <!--
        Who are the employees?
        Answers:
        x: <Ind>Bill</Ind>
        x: <Ind>William</Ind>
        Sublanguage: datalogeq
      -->
    <Atom>
      <Rel>employee</Rel>
      <Var>x</Var>
    </Atom>
</Query>

<Assert>
    <!--
        Equations may be universally quantified.
        This fact is a degenerate case, corresponding to the body of an
 implication being empty.
        The following makes the reflexive property of equality explicit (
 as built into most systems):
        Everything is equal to itself.
        Sublanguage: datalogeq
      -->
    <Forall>
      <Var>x</Var>
      <Equal>
        <Var>x</Var>
        <Var>x</Var>
```

```
    </Equal>
  </Forall>
</Assert>
<Query>
  <!--
       What is equal to itself?
       Answers:
       x: <Ind>Bill</Ind>
       x: <Ind>William</Ind>
       Sublanguage: datalogeq
    -->
  <Equal>
    <Var>x</Var>
    <Var>x</Var>
  </Equal>
</Query>

<Assert>
  <!--
       Non-ground facts are allowed.
       John is the CEO.
       John is responsible for everything.
       Sublanguage: datalogeq
    -->
  <Atom>
    <Rel>CEO</Rel>
    <Ind>John</Ind>
  </Atom>
  <Forall>
    <Var>x</Var>
    <Atom>
      <Rel>responsible_for</Rel>
      <Ind>John</Ind>
      <Var>x</Var>
    </Atom>
  </Forall>
</Assert>
<Query>
  <!--
       Is John responsible for Bill?
       Answers:
       Succeeds.
```

```
      Sublanguage: datalogeq
    -->
  <Atom>
    <Rel>responsible_for</Rel>
    <Ind>John</Ind>
    <Ind>Bill</Ind>
  </Atom>
</Query>

<Assert>
  <!--
       Rules are expressed as universally-quantified implications.
       A simple rule can be used to assert a subclass relationship:
       Every manager is an employee.
       Sublanguage: datalog
    -->
  <Forall>
    <Var>x</Var>
    <Implies>
      <if>
        <Atom>
          <Rel>manager</Rel>
          <Var>x</Var>
        </Atom>
      </if>
      <then>
        <Atom>
          <Rel>employee</Rel>
          <Var>x</Var>
        </Atom>
      </then>
    </Implies>
  </Forall>
  <Atom>
    <Rel>manger</Rel>
    <Ind>John</Ind>
  </Atom>
</Assert>
<Query>
  <!--
       Who are the employees?
       Answers:
```

```
        x: <Ind>Bill</Ind>
        x: <Ind>William</Ind>
        x: <Ind>John</Ind>
        Sublanguage: datalog
    -->
  <Atom>
    <Rel>employee</Rel>
    <Var>x</Var>
  </Atom>
</Query>

<Assert>
  <!--
       Equations are allowed in the body of (existential) implications.
       If anyone is the same as Margaret, then they supervise someone.
       (Note that this rule could be semantically simplified
       to an existential fact relying on axioms of equality.)
       Sublanguage: datalogexeq
    -->
  <Forall>
    <Var>x</Var>
    <Implies>
      <if>
        <Equal>
          <Ind>Margaret</Ind>
          <Var>x</Var>
        </Equal>
      </if>
      <then>
        <Exists>
          <Var>y</Var>
          <Atom>
            <Rel>supervises</Rel>
            <Var>x</Var>
            <Var>y</Var>
          </Atom>
        </Exists>
      </then>
    </Implies>
  </Forall>
</Assert>
<Query>
```

```
  <!--
       Does Margaret supervise someone?
       Answer:
       Succeeds
       Sublanguage: datalogeq
    -->
  <Exists>
    <Var>y</Var>
    <Atom>
      <Rel>supervises</Rel>
      <Ind>Margaret</Ind>
      <Var>y</Var>
    </Atom>
  </Exists>
</Query>

<Assert>
  <!--
       Pairwise Disjoint Classes
       Nothing is both an employee and a department.
       Sublanguage: ncdatalog
    -->
  <Forall>
    <Var>x</Var>
    <Implies>
      <if>
        <And>
          <Atom>
            <Rel>employee</Rel>
            <Var>x</Var>
          </Atom>
          <Atom>
            <Rel>department</Rel>
            <Var>x</Var>
          </Atom>
        </And>
      </if>
      <then>
        <Or/>
      </then>
    </Implies>
  </Forall>
```

```
</Assert>
<Assert>
  <!-- HR is an employee.
       HR is a department.
       These facts together with the previous rule create an
 inconsistency.
       Sublanguage: datalog
    -->
  <Atom>
    <Rel>employee</Rel>
    <Ind>HR</Ind>
  </Atom>
  <Atom>
    <Rel>department</Rel>
    <Ind>HR</Ind>
  </Atom>
</Assert>
<Query>
  <!--
       Is there any inconsistency?
       Succeeds, indicating inconsistency.
       Sublanguage: ncdatalog
    -->
  <Or/>
</Query>
<Retract>
  <!--
       Remove that HR is an employee.
       Sublanguage: datalog
  -->
  <Atom>
    <Rel>employee</Rel>
    <Ind>HR</Ind>
  </Atom>
</Retract>
<Query>
  <!--
       Is there any inconsistency?
       Fails, indicating consistency.
       Sublanguage: ncdatalog
    -->
  <Or/>
```

```
</Query>
<Assert>
  <!--
       Functionality Constraint:
       Everyone (or everything) has at most one supervisor.
       Sublanguage: datalogeq
    -->
  <Forall>
    <Var>x</Var>
    <Var>y</Var>
    <Var>z</Var>
    <Implies>
      <if>
        <And>
          <Atom>
            <Rel>supervises</Rel>
            <Var>x</Var>
            <Var>z</Var>
          </Atom>
          <Atom>
            <Rel>supervises</Rel>
            <Var>y</Var>
            <Var>z</Var>
          </Atom>
        </And>
      </if>
      <then>
        <Equal>
          <Var>x</Var>
          <Var>y</Var>
        </Equal>
      </then>
    </Implies>
  </Forall>
  <Atom>
    <Rel>supervises</Rel>
    <Ind>Margaret</Ind>
    <Ind>Bill</Ind>
  </Atom>
  <Atom>
    <Rel>supervises</Rel>
    <Ind>Peggy</Ind>
```

```
    <Ind>Bill</Ind>
  </Atom>
</Assert>
<Query>
  <!--
        Is Peggy the same as Margaret?
        Answer:
        Succeeds.
        Sublanguage: datalogeq
   -->
  <Equal>
    <Ind>Peggy</Ind>
    <Ind>Margaret</Ind>
  </Equal>
</Query>

<Assert>
  <!--
        Negative Constraints are allowed.
        No one (or no thing) is their own supervisor.
        Sublanguage: ncdatalog
    -->
  <Forall>
    <Var>x</Var>
    <Implies>
      <if>
        <Atom>
          <Rel>supervises</Rel>
          <Var>x</Var>
          <Var>x</Var>
        </Atom>
      </if>
      <then>
        <Or/>
      </then>
    </Implies>
  </Forall>
</Assert>
<Assert>
  <!--
        Margaret supervises Peggy.
        Sublanguage: datalog
```

```
  -->
  <Atom>
    <Rel>supervises</Rel>
    <Ind>Margaret</Ind>
    <Ind>Peggy</Ind>
  </Atom>
</Assert>
<Query>
  <!--
        Is there any inconsistency?
        Succeeds, indicating inconsistency.
        Sublanguage: ncdatalog
  -->
  <Or/>
</Query>
<Retract>
  <!--
        Remove that Margaret supervises Peggy.
        Sublanguage: datalog
  -->
  <Atom>
    <Rel>supervises</Rel>
    <Ind>Margaret</Ind>
    <Ind>Peggy</Ind>
  </Atom>
</Retract>
<Query>
  <!--
        Is there any inconsistency?
        Fails, indicating consistency.
        Sublanguage: ncdatalog
  -->
  <Or/>
</Query>
<Assert>
  <!--
        Equations may appear in the body of negative constraints.
        The simplest case is the assertion that two individuals
        are different (as built into systems making the unique name
 assumption).
        Sublanguage: ncdatalogeq
```

```
    -->
  <Implies>
    <if>
      <Equal>
        <Ind>Sue</Ind>
        <Ind>Maria</Ind>
      </Equal>
    </if>
    <then>
      <Or/>
    </then>
  </Implies>
</Assert>
<Query>
  <!--
        Is Sue the same as Maria?
        Answer:
        Fails.
        Sublanguage: datalogeq
    -->
  <Equal>
    <Ind>Sue</Ind>
    <Ind>Maria</Ind>
  </Equal>
</Query>

<Assert>
  <!--
        Existential (Head) Rules
        Every manager directs at least one department.
        Maria is a manager.
        Sublanguage: datalogex
    -->
  <Forall>
    <Var>M</Var>
    <Implies>
      <if>
        <Atom>
          <Rel>manager</Rel>
          <Var>M</Var>
        </Atom>
      </if>
```

```
      <then>
        <Exists>
          <Var>P</Var>
          <Atom>
            <Rel>directs</Rel>
            <Var>M</Var>
            <Var>P</Var>
          </Atom>
        </Exists>
      </then>
    </Implies>
  </Forall>
  <Atom>
    <Rel>manager</Rel>
    <Ind>Maria</Ind>
  </Atom>
</Assert>
<Query>
  <!--
        Does Maria direct a department?
        Answer:
        Succeeds.
        Sublanguage: datalog
    -->
  <Exists>
    <Var>P</Var>
    <Atom>
      <Rel>directs</Rel>
      <Ind>Maria</Ind>
      <Var>P</Var>
    </Atom>
  </Exists>
</Query>

<Assert>
  <!--
        The heads and bodies of existential rules can contain
 conjunctions.
        Every employee who directs a department is a manager, and
 supervises at
        least another employee who works in the same department.
        Sublanguage: datalogexcon
```

```
    -->
  <Forall>
    <Var>E</Var>
    <Var>P</Var>
    <Implies>
      <if>
        <And>
          <Atom>
            <Rel>employee</Rel>
            <Var>E</Var>
          </Atom>
          <Atom>
            <Rel>directs</Rel>
            <Var>E</Var>
            <Var>P</Var>
          </Atom>
        </And>
      </if>
      <then>
        <Exists>
          <Var>E'</Var>
          <And>
            <Atom>
              <Rel>manager</Rel>
              <Var>E</Var>
            </Atom>
            <Atom>
              <Rel>supervises</Rel>
              <Var>E</Var>
              <Var>E'</Var>
            </Atom>
            <Atom>
              <Rel>works_in</Rel>
              <Var>E'</Var>
              <Var>P</Var>
            </Atom>
          </And>
        </Exists>
      </then>
    </Implies>
  </Forall>
</Assert>
```

```
  <Query>
    <!--
          Does Maria supervise someone?
          Answer:
          Succeeds.
          Sublanguage: datalog
      -->
    <Exists>
      <Var>E'</Var>
      <Atom>
        <Rel>supervises</Rel>
        <Ind>Maria</Ind>
        <Var>E'</Var>
      </Atom>
    </Exists>
  </Query>
</RuleML>
```

B. Translating the Business Scenario Example to TPTP

The TPTP document below is the output of the translation from the RuleML document in Appendix A. Note that the RuleML input contains comments as well as relators and variables that do not follow the syntax of the TPTP language, for example, `<Rel>CEO</Rel>` and `<Var>E'</Var>`. After the translation, the former one is converted into a valid functor, `cEO`, in the TPTP language, but the translation of the latter one, still `E'`, will not be executable by TPTP-accepting provers due to the apostrophe character. To execute this TPTP output on a TPTP-accepting prover, it has to be revised further and manually. Below is the TPTP output as the result of the translation:

```
% Some of these examples are from
% "A general Datalog-based framework for tractable query answering
% over ontologies",
% Andrea Calì,
% Georg Gottlob
% Thomas Lukasiewicz
% http://dx.doi.org/10.1016/j.websem.2012.03.001
% (preprint at http://www.websemanticsjournal.org/index.php/ps/article/
   view/289)
% This RuleML Document incrementally asserts into, retracts from and
% queries a rulebase (within the <RuleML> element) for a total of
% 28 transactions: 13 Asserts, 2 Retracts and 13 Queries.
% Each Query demonstrates the semantics of the previous Assert or Retract
% by providing (in the XML comments) the expected answer to the Query.
% Equations are allowed as facts:
% William is an employee.
% "Bill" is an alias for "William".
% Sublanguage: datagroundfacteq
fof(assert_in_act1_formula1, axiom, (
    employee(william)
)).
fof(assert_in_act1_formula2, axiom, (
```

```
    bill = william
)).
% Who are the employees?
% Answers:
% x: <Ind>Bill</Ind>
% x: <Ind>William</Ind>
% Sublanguage: datalogeq
fof(query_in_act2_formula1, conjecture, (
    employee(X)
)).
% Equations may be universally quantified.
% This fact is a degenerate case, corresponding to the body of an
   implication being empty.
% The following makes the reflexive property of equality explicit (as
   built into most systems):
% Everything is equal to itself.
% Sublanguage: datalogeq
fof(assert_in_act3_formula1, axiom, (
    ! [X] : X = X
)).
% What is equal to itself?
% Answers:
% x: <Ind>Bill</Ind>
% x: <Ind>William</Ind>
% Sublanguage: datalogeq
fof(query_in_act4_formula1, conjecture, (
    X = X
)).
% Non-ground facts are allowed.
% John is the CEO.
% John is responsible for everything.
% Sublanguage: datalogeq
fof(assert_in_act5_formula1, axiom, (
    cEO(john)
)).
fof(assert_in_act5_formula2, axiom, (
    ! [X] : responsible_for(john, X)
)).
% Is John responsible for Bill?
% Answers:
% Succeeds.
% Sublanguage: datalogeq
```

```
fof(query_in_act6_formula1, conjecture, (
    responsible_for(john, bill)
)).
% Rules are expressed as universally-quantified implications.
% A simple rule can be used to assert a subclass relationship:
% Every manager is an employee.
% Sublanguage: datalog
fof(assert_in_act7_formula1, axiom, (
    ! [X] :
      ( manager(X)
     => employee(X) )
)).
fof(assert_in_act7_formula2, axiom, (
    manger(john)
)).
% Who are the employees?
% Answers:
% x: <Ind>Bill</Ind>
% x: <Ind>William</Ind>
% x: <Ind>John</Ind>
% Sublanguage: datalog
fof(query_in_act8_formula1, conjecture, (
    employee(X)
)).
% Equations are allowed in the body of (existential) implications.
% If anyone is the same as Margaret, then they supervise someone.
% (Note that this rule could be semantically simplified
% to an existential fact relying on axioms of equality.)
% Sublanguage: datalogexeq
fof(assert_in_act9_formula1, axiom, (
    ! [X] :
      ( margaret = X
     => ? [Y] : supervises(X, Y) )
)).
% Does Margaret supervise someone?
% Answer:
% Succeeds
% Sublanguage: datalogeq
fof(query_in_act10_formula1, conjecture, (
    ? [Y] : supervises(margaret, Y)
)).
% Pairwise Disjoint Classes
```

```
% Nothing is both an employee and a department.
% Sublanguage: ncdatalog
fof(assert_in_act11_formula1, axiom, (
    ! [X] :
      ( ( employee(X)
        & department(X) )
     => $false )
)).
% HR is an employee.
% HR is a department.
% These facts together with the previous rule create an inconsistency.
% Sublanguage: datalog
fof(assert_in_act12_formula1, axiom, (
    employee(hR)
)).
fof(assert_in_act12_formula2, axiom, (
    department(hR)
)).
% Is there any inconsistency?
% Succeeds, indicating inconsistency.
% Sublanguage: ncdatalog
fof(query_in_act13_formula1, conjecture, (
    $false
)).
% Is there any inconsistency?
% Fails, indicating consistency.
% Sublanguage: ncdatalog
fof(query_in_act15_formula1, conjecture, (
    $false
)).
% Functionality Constraint:
% Everyone (or everything) has at most one supervisor.
% Sublanguage: datalogeq
fof(assert_in_act16_formula1, axiom, (
    ! [X, Y, Z] :
      ( ( supervises(X, Z)
        & supervises(Y, Z) )
     => X = Y )
)).
fof(assert_in_act16_formula2, axiom, (
    supervises(margaret, bill)
)).
```

```
fof(assert_in_act16_formula3, axiom, (
    supervises(peggy, bill)
)).
% Is Peggy the same as Margaret?
% Answer:
% Succeeds.
% Sublanguage: datalogeq
fof(query_in_act17_formula1, conjecture, (
    peggy = margaret
)).
% Negative Constraints are allowed.
% No one (or no thing) is their own supervisor.
% Sublanguage: ncdatalog
fof(assert_in_act18_formula1, axiom, (
    ! [X] :
      ( supervises(X, X)
     => $false )
)).
% Margaret supervises Peggy.
% Sublanguage: datalog
fof(assert_in_act19_formula1, axiom, (
    supervises(margaret, peggy)
)).
% Is there any inconsistency?
% Succeeds, indicating inconsistency.
% Sublanguage: ncdatalog
fof(query_in_act20_formula1, conjecture, (
    $false
)).
% Is there any inconsistency?
% Fails, indicating consistency.
% Sublanguage: ncdatalog
fof(query_in_act22_formula1, conjecture, (
    $false
)).
% Equations may appear in the body of negative constraints.
% The simplest case is the assertion that two individuals
% are different (as built into systems making the unique name assumption).
% Sublanguage: ncdatalogeq

fof(assert_in_act23_formula1, axiom, (
    ( sue = maria
```

```
   => $false )
)).
% Is Sue the same as Maria?
% Answer:
% Fails.
% Sublanguage: datalogeq
fof(query_in_act24_formula1, conjecture, (
    sue = maria
)).
% Existential (Head) Rules
% Every manager directs at least one department.
% Maria is a manager.
% Sublanguage: datalogex
fof(assert_in_act25_formula1, axiom, (
    ! [M] :
      ( manager(M)
     => ? [P] : directs(M, P) )
)).
fof(assert_in_act25_formula2, axiom, (
    manager(maria)
)).
% Does Maria direct a department?
% Answer:
% Succeeds.
% Sublanguage: datalog
fof(query_in_act26_formula1, conjecture, (
    ? [P] : directs(maria, P)
)).
% The heads and bodies of existential rules can contain conjunctions.
% Every employee who directs a department is a manager, and supervises at
% least another employee who works in the same department.
% Sublanguage: datalogexcon
fof(assert_in_act27_formula1, axiom, (
    ! [E, P] :
      ( ( employee(E)
        & directs(E, P) )
     => ? [E'] :
          ( manager(E)
          & supervises(E, E')
          & works_in(E', P) ) )
)).
% Does Maria supervise someone?
```

```
% Answer:
% Succeeds.
% Sublanguage: datalog
fof(query_in_act28_formula1, conjecture, (
    ? [E'] : supervises(maria, E')
)).
```