Extending OWL with Finite Automata Constraints

Jignesh Borisa
December 15, 2010
Advisor
Dr. Chris Pollett
Committee Members
Dr. Jon Pearce
Dr. Robert Chun
Agenda

• What the project is about?
• Introduction
• Motivation
• Design and Specification
• Tools Used
• Implementation
• Test Case Outputs
• Demo
• Conclusion
What the project is about?

Developed an extension to OWL that allows one to support collections and constraints where membership in the collections can be computed by finite automata.
What is an ontology?

- Formal representation of the knowledge by a set of concepts within a domain and relationships between those concepts.
- Used to reason about the properties of that domain and describe the domain.
- Ontology consists of
  - Classes: sets, collections, concepts
  - Relations between classes (consists of, must be preceded by, etc)
  - Functions (relation with 1 result)
  - Individuals (instances or objects)
  - Axiomata (knowledge on concepts/relations that can be checked on its logics)
  - Attributes: properties, parameters or characteristics that objects can have
XML and RDF

- **Extensible Markup Language (XML)**
  - Defines rules to mark-up a document in a way that allows the author to express semantic meaning in the mark-up
  - Data format used primarily for sharing data

- **Resource Description Framework (RDF)**
  - Framework for describing resources on the web
  - Designed to be read and understood by computer applications
  - RDF descriptions are not designed to be displayed on the web
  - Datamodel for objects ("resources") and relations between them
  - RDF is a collection of triples, each consisting of a subject, a predicate, and an object.
  - An open-world framework that allows anyone to make statements about any resources
Web Ontology Language (OWL)

- Markup language for sharing and publishing data using ontologies on the Internet
- It belongs to a family of knowledge representation languages for writing ontologies
- Provides more vocabulary for describing properties and classes than XML and RDF like cardinality (i.e. exactly one), relations between classes (i.e. disjointness), etc.
- Used for representing the meaning of the terms in vocabularies and their interrelationships
Answer Set Programming

- Logic programming paradigms are also used for knowledge representation.
- We considered answer set programming, a particular form of logic programming.
- It is a declarative programming approach to knowledge representation.
- It is oriented towards difficult search problems.
- It is based on the stable model semantics of logic programming.
What is Logic Program?

- It is the use of mathematic logic for computer program.
- The logic programming is the use of logic as both a declarative and procedural representation language.
- It is based upon the fact that a backwards theorem-prover applied to declarative sentences in the form of implications:
  \[
  \text{If } B_1 \text{ and } \ldots \text{ and } B_n \text{ then } H
  \]
- It also treats the implications as goal-reduction procedures: to show/solve $H$, show/solve $B_1$ and $\ldots$ and $B_n$. 
Example of Logic Program

- As the implication:
  
  If you press the alarm signal button, then you alert the driver of the train of a possible emergency.

- As the procedure:
  
  To alert the driver of the train of a possible emergency, press the alarm signal button.
Stable Model Semantics

- This model was proposed by Gelfond and Lifschitz in 1988.
- It defines a declarative semantics for logic program with negation as failure.
- Let P be a logic program and Q be a subset of variables of P.
- Let $P^Q$ be the program.
- If the program contains clause C of P, which contains the negated variable Not A in its body such that $A \in Q$, then C is not counted.
- If a body of clause contains a negated Not A such that $A \notin Q$, then Not A is not counted from the clause body.
- If Q is a least Herbrand model of $P^Q$, then Q is a stable model of P.
Computing Stable Model

Consider the following logic program:

\[ x_2 : - \neg x_1 \]
\[ x_1 : - \neg x_2 \]

Truth Table for Computed Stable Model

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>Stable Model exist?</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>No</td>
</tr>
</tbody>
</table>
Computing Stable Model (Cont...) 

- Consider the following logic program:
  \[ x_1: \neg x_4, \neg x_2 \]
  \[ x_2: \neg x_4, \neg x_3 \]
  \[ x_3: \neg x_2 \]

- Reduced model is derived from this logic program. It is as follow:
  \[ x_1: x_4 \]
  \[ x_2: x_4 \]
  \[ x_3: x_4 \]

- Let us look at truth table of this program for computing Stable model
<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>Stable Model exist?</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>Yes</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>No</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td>No</td>
</tr>
</tbody>
</table>
OWL Capabilities

- Three sublanguages: OWL Lite, OWL DL and OWL Full.
- While OWL Lite supports cardinality constraints, it only permits cardinality values of 0 or 1.
- OWL DL includes all OWL language constructs, but they can be used only under certain restrictions.
- For example, while a class may be a subclass of many classes, a class cannot be an instance of another class.
- In OWL Full, a class can be treated simultaneously as a collection of individuals and as an individual.
- OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary.
Denotation in Description Logic

- Consider following denotation in Description Logic:

  \[( \text{union Male Female} ) \quad \land \quad ( \text{subClass} \leq 2 \text{hasChild} ) \quad \Rightarrow \quad ( \text{class Person} )\]
An Example OWL Document

The denotation in DL is equivalent to following OWL code:

```xml
<owl:Class rdf:about="#Person">
  <owl:disjointUnionOf rdf:parseType="Collection">
    <owl:Class rdf:ID="Male"/>
    <owl:Class rdf:ID="Female"/>
  </owl:disjointUnionOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild"/>
      <owl:mincardinality rdf:datatype="&xsd;string">1</owl:mincardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```
One way to Extend OWL

Consider the following denotation in Description Logic:

\[(\text{subClass} \ (\exists \text{hasDNASequence.}\ \{A*TA*G\}) \ ) \Rightarrow (\text{Class Person})\]

- Here \(A*TA*G\) is some regular expression such that only humans have DNA sequences of this type.
- This is something you could do in OWL. What we can’t do is come up with a parameterized family of subclasses of this…
- i.e., we might want to define:

\[(\text{subClass} \ (\text{hasDNASequence.}\ \{A*TA*G\}) \ ) \Rightarrow (\text{Class Person}\{A*TA*G\})\]

And have \((\text{Class Person}\{AATAAAG\})\) be class which is an instance of this family. For example, this might represent Person’s with Dwarfism.
Let Person be an OWL class and hasDNASequence be its data property. The value of hasTelphone is A*TA*G which is regular expression.
Person Example in Our Extended Syntax

<owl:CollectionClass rdf:about="#Person">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasDNASequence"/>
            <owl:hasValue rdf:datatype="xsd:string">
                A*TA*G
            </owl:hasValue>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
Motivation: Incapability of OWL

• The previous experiment with OWL does not work because
  • Regular expression provides concise and flexible means of matching strings of text.
  • OWL can only support inflexible or fixed value for data property.
  • OWL cannot allow set of values for property.
• We decided to extend OWL to support collections.
  • Collections are a natural part of the world that we need to model such as protein sequences, SSN, email ID, etc.
Motivation: Extension to Answer Set Programming

- A recent extension to answer set programming is to support constraints where membership in the sets can be computed by finite automata.
- In this extension, new types of constraints are introduced that allow for a more compact representation of problem in answer set programming.
- We attempted to extend OWL with feature from this new approach to answer set programming.
Design

- Created the following language constructs to support collections and finite automata constraints
  1. CollectionClass
  2. memberClassOf
  3. collectionClassOf
  4. instanceOf
CollectionClass

- DataRange
  - +arity:int

- OWLEntity
  - +URI:String

- Datatype

- CollectionClass

- DataProperty
CollectionClass Specification

- Defined by URI
  
  - datatypeURI : = URI
  - dataPropertyURI := URI
  - CollectionClassURI : = URI

- Syntax for CollectionClass
  
  - entity : = datatype | CollectionClass | dataProperty
  - datatype := 'Datatype' '(' datatypeURI ')'‘
  - CollectionClass : = ‘ CollectionClass’ ‘(‘ CollectionClassURI ‘)’
  - dataProperty : = ‘DataProperty’ ‘(‘ dataPropertyURI ‘)’
memberClassOf axiom
memberClassOf Specification

- Syntax for memberClassOf axiom
  
  \[
  \text{owlClass} : = \text{description} \\
  \text{CollectionClass} : = \text{description} \\
  \text{memberClassOf} : = \text{'MemberClassOf'} \text{'}\{\text{ annotation }\} \text{ } \text{owlClass } \text{CollectionClass } \text{'}\}
  \]
collectionClassOf axiom
collectionClassOf Specification

- Syntax for collectionClassOf axiom

  memberClass : = description
  CollectionClass : = description
  collectionClassOf : = 'CollectionClassof' '(' '{ annotation }'
                   memberClass CollectionClass ')

instanceOf axiom
instanceOf Specification

- Syntax for instanceOf axiom
  
  instance : = individual

  owlClass : = description

  instanceOf : = ‘InstanceOf’ ‘(‘ { annotation} instance
    owlClass ‘)’
Tools Used

- OWL 2.0
- Pellet
  - Java-based open source OWL reasoner
  - Provides various features like data type reasoning, ontology analysis, ontology debugging etc.
  - Used Pellet to reason about OWL document
- DOM API
  - Component API of the Java API for XML processing
  - Allows programs to dynamically access and update the content of documents
  - Used DOM API to parse an extended OWL document
Initial Research

1. Compute Stable Model Semantics

Rule:
- x1:- x4.
- x2:- x4,x5.
- x3:- x1,x3,-x2.

For i=0
Stable Model Exist
Initial Research (Cont...)  

2. Created and reasoned about OWL document

```xml
<terminated> ExplanationExample (2) [Java Application] C:\Program Files\Java\jdk1.6.0_04\bin\javaw.exe

Why is optical+led subclass of mouse+led?
Explanation:
  has_part range optical
  mouse+led equivalentTo mouse
  and has_part some optical
  optical+led equivalentTo mouse
  and has_part min 5
```
Initial Research (Cont...)

3. Implemented Finite Automata Closure Algorithm

```plaintext
<terminated> StableModel1 [Java Application] C:\Program Files\Java\jdk1.6.0_04\bin\javaw.exe
x1:|43R0;
x2:x3|213R0,34R1;
For i=0
  Stable Model does not Exist
For i=1
  Stable Model does not Exist
For i=2
  Stable Model does not Exist
For i=3
  Stable Model does not Exist
For i=4
  Stable Model Exist
```
Venn Diagram for Extended OWL

The Collection Class Person \{ A^*TA^*G \}

Person\{AATAG\}

- frank
- bob
- alice

Person\{AATAAAG\}

- trudy
- tom
- mark
Implementation

- To support collections
  - We added CollectionClass entity to OWL to represent collections.
    \[(\text{subClass}\ (\exists\text{hasTelephone.}\ \{[0-9]\{3\}-[0-9]\{3\}-[0-9]\{4\}\})\)\]
    \[\Rightarrow (\text{CollectionClassTelephonePattern})\]

```xml
<owl:CollectionClass rdf:about="#TelephonePattern">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasTelephone"/>
      <owl:hasValue rdf:datatype="xsd:string">
        [0-9]\{3\}-[0-9]\{3\}-[0-9]\{4\}
      </owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:CollectionClass>
```
Implementation (Cont..)

• Now that we extended OWL, we need to add support for reasoning about these kind of extended documents. This is where incorporated idea from Remmel and Marek paper for answer set programming
• We extended subclass definition. We added positive variables and constraints.
The semantics for extension to subclass is as follows:

\[ e \sqsubseteq \text{subclass} : -e_1 \in c_1, e_2 \in c_2, \ldots, e_n \in c_n, c_1 \in \lambda_1, c_2 \in \lambda_2, \ldots, c_n \in \lambda_n \]

Here, \( e_1, e_2, \ldots, e_n \) are instances of any OWL class in ontology
\( c_1, c_2, \ldots, c_n \) are OWL classes in ontology
\( \lambda_1, \lambda_2, \ldots, \lambda_n \) are CollectionClasses in ontology
\( e_1 \in c_1, e_2 \in c_2, \ldots, \) are positive variables
\( c_1 \in \lambda_1, c_2 \in \lambda_2, \ldots, c_n \in \lambda_n \) are constraints
Implementation of Semantics

- We guessed true or false value for every constraint from $0$ to $2^{\text{max}}$ times where max is number of constraints.
- We checked membership of OWL class in CollectionClasss. If the property values of all the instances of OWL class match the regular expression of property of CollectionClass then we can derive true as a value for that constraint.
- If the derived values of all the constraints are same as guessed values for those constraints then we can say that stable model exist.
Implementation (Cont...) 

- Consider following denotation of extended class in Description Logic (DL):

\( (\text{member maddox Male}) \)
\[ \land (\text{member braddpitt Adult}) \]
\[ \land (\text{member angelinajolie PersonWithAtLeastTwoChildren}) \]
\[ \land (\text{memberClass Daughter FatherDNASequence}) \]
\[ \land (\text{memberClass Son MotherDNASequence}) \]
\[ \Rightarrow (\text{class Son}) \]
The denotation in DL is equivalent to following description of class in OWL

```xml
<owl:Class rdf:about="#Person">
  <owl:instanceOf rdf:ID="maddox">
    <owl:Class rdf:ID="Male"/>
  </owl:instanceOf>
  <owl:instanceOf rdf:ID="bradpitt">
    <owl:Class rdf:ID="Adult"/>
  </owl:instanceOf>
  <owl:instanceOf rdf:ID="angelinajolie">
    <owl:Class rdf:ID="PersonWithAtLeastTwoChildren"/>
  </owl:instanceOf>

  <owl:memberClassOf rdf:ID="Daughter">
    <owl:CollectionClassOf rdf:about="#FatherDNASequence"/>
  </owl:memberClassOf>

  <owl:memberClassOf rdf:ID="Son">
    <owl:CollectionClassOf rdf:about="#MotherDNASequence"/>
  </owl:memberClassOf>
</owl:Class>
```
Implementation (Cont...)  

- Reason about extended OWL document  
  - Extended Pellet to parse an extended OWL document.  
  - Added DOMParser class to Pellet which can parse extended OWL document.  
  - DOMParser can compute stable model by guessing and deriving values for constraints.  
  - Guessed values for constraints from $0$ to $2^{\text{max}}$ times  
  - Checked membership of OWL class in CollectionClasses for given constraints and computed stable model  
  - If stable model exists then it can remove all the extended tags from the OWL document and write a new reduced OWL document.  
  - Pellet can reason about this reduced OWL document.
Test Case Output

- Explanation Inference

```plaintext
<terminated> ExplanationExample (3) [Java Application] C:\Program Files\Java\jdk1.6.0_04\bin\javaw.exe
OWL Document is ready for answering
Output File has been written
Why is Son subclass of Child+Male?
Explanations (2):
1) Male subClassOf Child+Male
   Son equivalentTo Male
      and hasChild min 2

2) Child+Male equivalentTo Male
   and hasChild some Father+Son
   hasChild range Father+Son
   Son equivalentTo Male
      and hasChild min 2
```
Test Case Output (Cont...)

- Query Subsumption Inference

```
<terminated> QuerySubsumptionExample (1) [Java Application] C:\Program Files\Java\jdk1.6.0_04\bin\javaw.exe

OWL Document is ready for answering
Output File has been written
Example 1
==========
Query 1: query(?x) :- family:Male(?x).
Query 2: query(?x) :- family:Person(?x).

Query 1 is subsumed by query 2: true
Query 2 is subsumed by query 1: false
```
Test Case Output (Cont...)

- Logical Inference

```text
<terminated> MembershipTest (4) [Java Application] C:\Program Files\Java\jdk1.6.0_04\bin\javaw.exe (Nov 20, 2010 7:11:49 PM)

OWL Document is ready for answering
Output File has been written

Is PersonWithAtLeastTwoFemaleChildren subClass of PersonWithAtLeastTwoChildren?
Yes, PersonWithAtLeastTwoFemaleChildren is subClass of PersonWithAtLeastTwoChildren
```
DEMO...
Conclusion

• Experimented and observed that OWL did not support infinite sets of collections.
• Created new entity called CollectionClass to support collections.
• Created new axioms called memberClassOf, instanceOf and collectionClassOf for adding constraints and positive variables to OWL.
• Extended subclass definition that can support constraints where membership in CollectionClass could be computed by finite automata.
• Developed three inferences to reason about extended OWL document.
References

Thank you
Question