

Extending OWL with Finite Automata Constraints

A Writing Project

Presented to

The Faculty of the department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Jignesh Borisa

November 2009

© 2009

Jignesh Borisa

ALL RIGHTS RESERVED

ABSTRACT

Extending OWL with Finite Automata Constraints

by Jignesh Borisa

Over the past few years, there has been vast progress in the theory of logic programming. In particular, a new area called answer set programming has arisen, which is a fusion of logic programming with Stable Model Semantics (SLP). Answer Set Programming is used to handle search problems. The Web Ontology Language (OWL) is designed to be used by applications. It is used to define the content of information to humans. Although OWL provides bases for various semantic web applications, it lacks sufficient declarative semantics for instance recognition to support automated semantic annotation. This omission prevents OWL from being a satisfactory ontology language for automated semantic annotation. This problem can be solved by adding declarative instance recognition semantics to OWL. The first goal of my project is to develop an extension of OWL which can handle answer set problems with finite automata constraints. The second goal is to develop an inference engine for the resulting language. I have computed stable model semantics by using an efficient algorithm. I have created a test OWL document for various parts of a computer mouse. I used Java based Pellet's OWL reasoner to reason about the OWL document. It generates OWL ontology for OWL document. I have implemented a finite automata closure algorithm. It counts stable model for the rules which satisfy all of its constraints.

Table of Contents

1. Introduction.....	6
2. Deliverable 1: Compute Stable Model Semantics.....	9
3. Deliverable 2: Install and Experiment with smodels.....	14
4. Deliverable 3: Create a test OWL document and try to find out inference engine.....	16
5. Deliverable 4: Implement finite automata closure algorithm.....	19
6. Summary.....	22
7. Future Work.....	23
References.....	24

List of Figures

Figure 1: Compute Stable Model Semantics Example 1.....	11
Figure 2: Compute Stable Model Semantics Example 2	12
Figure 3: Output of smodels.....	15

1. Introduction

As preparation for the writing project titled “Extending OWL with finite automata constraints”, I have written a report which provides brief details of my work that I have done in my CS297 project. The purpose is to prepare documentation for the work that I have done in my CS297 project.

Answer Set Programming (ASP) is one kind of declarative programming. It is based on the stable model semantics of logic programming. In ASP, there is not any need to use function symbols. By avoiding the use of function symbols, we can eliminate the problem of high complexity of the program models. Adding functional symbols in answer set programming creates high complexity problems for models of programs. For example, finding the stable model of the horn program with no function symbols would be done in linear time, and finding the stable model of horn program with functional symbols would take recursively enumerable set. Stable model semantics was proposed by Gelfond and Lifschitz in the late 80s. It defines a declarative semantics for logic programs with negation as failure. Let P be a logic program. Let Q be a subset of variables of P . Let P^Q be the program. If the program contains clause C of P , which contains negated variable $\text{Not } A$ in its body such that $A \in Q$ then C is not counted. But if a body of clause contains a negated $\text{Not } A$ such that $A \notin Q$, then $\text{Not } A$ 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 .

In this project, I have computed stable model semantics for logic program. For that I have found Horn Sets from the guessed values for the variable of the rules of the program. By comparing values of Horn rules with guessed values, we can check the existence of stable model. I have used an efficient algorithm to compute stable model semantics. I have guessed for variables from 0 to maximum values of all the variables in the program.

For example, if we have following rules of program.

P:

S: P, Q.

R: P, Not Q.

Here, let us guess the maximum value of the all variables in program. It means that all variables have true value. So, the Horn set contains P and S. Q and R are not in the Horn set because we have guessed Q as true and clause R contains Not Q in its body.

As a part of my project, I have learnt Web Ontology Language (OWL). OWL is used to define content of information to human instead of directly presenting information to humans. It is supported by RDF, XML and RDF Schema. The web ontology language describes the hierarchical organization of ideas in a domain, in a way that can be parsed and understood by software. OWL can be used explicitly to represent the meaning of various terms in vocabularies. OWL provides three sublanguages: OWL Lite, OWL DL and OWL Full. OWL ontology contains a set of axioms. These axioms place constraints on sets of individuals and the types of relationships permitted between them. These axioms provide semantics by allowing additional information based on the data explicitly provided. I have created a test OWL document for the parts of Computer Mouse.

I have used Pellet's OWL reasoner to reason about OWL document. It generates OWL ontology for OWL document. It is Java based library and builds OWL ontology. OWL ontology consists of a set of axioms which place constraints on sets of individuals and the types of relationships permitted between them.

2. Deliverable 1: Compute Stable Model Semantics

Motivation

The main objective of this deliverable is to write a Java program to compute stable model semantics. Stable model semantics was introduced by Gelfond and Lifschitz. The concept of

a stable model is used to provide a declarative semantics for logic programs with negation. It is one of the standard approaches to the meaning of negation in logic programming. It is the basis of answer set programming. Consider the following program:

a:
c: a, b
d: a, not b.

Given this program, the atom 'a' will true because the program has all 'a' as a true. The given atom 'b' will false, because it does not occur in the head of any rules of the program. The atom 'c' will false because the rule with 'c' contains 'a' and 'b' and we know that 'b' is false. The atom 'd' will true because d contains 'a' and not 'b'. 'b' is false which makes 'd' as true. The given program can be represented by the following truth assignment:

a	b	c	d
T	F	F	T

Goal

The goal of computing stable model semantics is to write a Java program for computing stable model for given rules. For computing stable model semantics, it needs to parse given rules in properties file and to find horn sets from the rules of program. From horn sets of rules, we can compute stable model semantics.

Implementation and Results

I have developed a Java program to compute stable model semantics. First, I have created a properties file that contains all rules of the program. Each rule contains head, variable list and negative variable list.

For example, x1: x2, x3,-x4, x5.

Here, we have x1 as head of the rule, x2 is variable and x4 and x5 are negative variables.

1. Parse the rules of logic program

I have implemented following various functions for parsing the rules.

- `parseRule()` is used to parse the rules of the property file. It returns rule.
- `parseHead()` is used to parse the heads of all rules in the property file. It returns head of the rules.
- `parseVarlist()` is used to parse the variable lists of rules. It returns a variable list of the rules.
- `parseNegVarlist()` is used to parse the negative variable list. It returns the negative variable list of the rules.

2. Guessing for Variables

I have guessed true or false value for all variables. I guessed from 0 to `Max_Value` for all variables of the rule where `Max_Value` is maximum value of all variables. For example, if you guess 0 for all variables for the rules than all variables would have false value and if you guess `Max_Value` for all variables than all variables would have true value.

3. Find Horn Sets

From the guessed values of variables, I have predicted horn sets. I have implemented function `getHornSet()`. It predicts horn sets from the guessed values of variables.

For example, we have following two rules:

$X1:-X2$ and $X2:-X1$

If I guess X2 as false then X1 is horn sets for the rules. We can predict values for variables from Horn Sets.

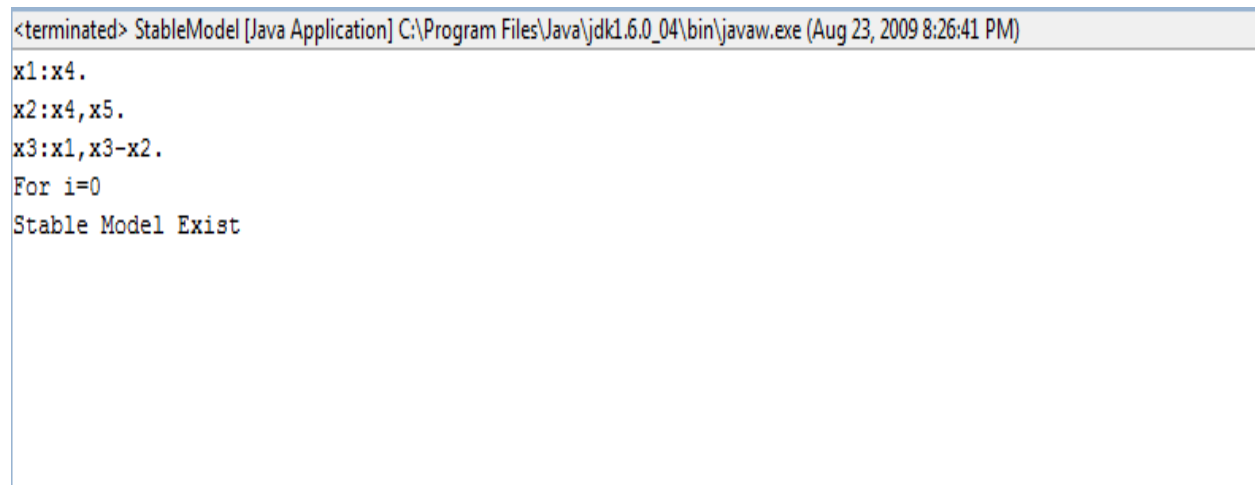
4. Check Stable Model

From the Horn Sets, we can check if the stable model is exists or not. I have implemented function `checkStableModel ()` which checks existence of a stable model for the rules. It predicts values of variables from Horn Sets. If a Horn set contains variable, then we can predict its value as true or false for other remained variables which are not in Horn sets. If variables have the same values as guessed values of variables then a Stable Model is exist for that guessed values of variables. Otherwise Stable Model does not exist.

Example 1

Rules

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



```
<terminated> StableModel [Java Application] C:\Program Files\Java\jdk1.6.0_04\bin\javaw.exe (Aug 23, 2009 8:26:41 PM)
x1:x4.
x2:x4,x5.
x3:x1,x3-x2.
For i=0
Stable Model Exist
```

Figure 1: Compute Stable Model Semantics Example 1

Example 2

Rules

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

```
<terminated> StableModel [Java Application] C:\Program Files\Java\jdk1.6.0_04\bin\javaw.exe (Aug 23, 2009 8:35:49 PM)
x1:x4,-x2.
x2:x4,-x3.
x3:-x2.
For i=0
Stable Does not Exist
For i=1
Stable Does not Exist
For i=2
Stable Does not Exist
For i=3
Stable Does not Exist
For i=4
Stable Does not Exist
For i=5
Stable Does not Exist
For i=6
Stable Does not Exist
For i=7
Stable Does not Exist
For i=8
Stable Does not Exist
For i=9
Stable Does not Exist
For i=10
Stable Does not Exist
For i=11
Stable Does not Exist
For i=12
Stable Does not Exist
For i=13
Stable Does not Exist
For i=14
Stable Does not Exist
For i=15
Stable Does not Exist
```

Figure 2: Compute Stable Model Semantics Example 2

Remarks

I concluded that the Java program that I have implemented to compute stable model semantics is working properly. I have tested the Java program as per various rules of the logic programs. I have guessed for variables of the rules from 0 to Max_Value where Max_Value is

the maximum of the variables of the rules. If guessed values of variables of the rules are same as Horn sets then a stable model exists otherwise a stable model does not exist.

3. Deliverable 2: Install and Experiment with smodels

Motivation

The main objective of this deliverable is to install and experiments with smodels. Smodels is an implementation of the stable model semantics for logic programs, which is implemented by Patrik Simons. Smodels can be used either as a C++ library or as a stand-alone program together with a suitable front-end. It can be called from user programs. Smodels extends the normal logic programs. It adds new special following rule types like constant rules, choice rules and weight rules. Smodels works with variable-free programs that are quite cumbersome to generate by hand. Lparse is a front-end. It adds variables to the accepted language and generates a variable-free simple logic program. It can be given to smodels. By translating other semantics into normal logic programs, lparse implements several them.

Goal

The goal of this deliverable is to install and experiment with smodels. Smodels is a collection of c++ programs. It computes stable model semantics for logic programs. It checks existence of the stable model for particular rules of the program.

Implementations and Results

Smodels contains C++ program files. It is used to compute stable model semantics. I have installed smodels-2.33 with lparse-1.1.1. I have made changes as per logic program and set the values of objects in the programs as per logic programs. I have measured outputs of smodels as per various logic programs. It gives the existence of stable model for logic program as an output. It is an efficient program to compute stable model semantics.

```

jignesh@jignesh-laptop:~$ cd Desktop
jignesh@jignesh-laptop:~/Desktop$ cd examples/
bash: cd: examples/: No such file or directory
jignesh@jignesh-laptop:~/Desktop$ cd smodels-2.33/
jignesh@jignesh-laptop:~/Desktop/smodels-2.33$ cd examples/
jignesh@jignesh-laptop:~/Desktop/smodels-2.33/examples$ make -I example
g++ -o example example.o ../smodels.o ../stack.o ../dcl.o ../atomrule.o ../read
.o ../queue.o ../timer.o ../list.o ../improve.o ../program.o ../api.o ../stable
.o ../tree.o ../denant.o ../restart.o
g++ -o example2 example2.o ../smodels.o ../stack.o ../dcl.o ../atomrule.o ../re
ad.o ../queue.o ../timer.o ../list.o ../improve.o ../program.o ../api.o ../stabl
e.o ../tree.o ../denant.o ../restart.o
g++ -o example3 example3.o ../smodels.o ../stack.o ../dcl.o ../atomrule.o ../re
ad.o ../queue.o ../timer.o ../list.o ../improve.o ../program.o ../api.o ../stabl
e.o ../tree.o ../denant.o ../restart.o
g++ -o example4 example4.o ../smodels.o ../stack.o ../dcl.o ../atomrule.o ../re
ad.o ../queue.o ../timer.o ../list.o ../improve.o ../program.o ../api.o ../stabl
e.o ../tree.o ../denant.o ../restart.o
g++ -o example5 example5.o ../smodels.o ../stack.o ../dcl.o ../atomrule.o ../re
ad.o ../queue.o ../timer.o ../list.o ../improve.o ../program.o ../api.o ../stabl
e.o ../tree.o ../denant.o ../restart.o
jignesh@jignesh-laptop:~/Desktop/smodels-2.33/examples$ ./example
a :- b, c, not d, not e.
b :- a, not d.
c :- a, not b.
Stable Model:
a is not in the stable model
jignesh@jignesh-laptop:~/Desktop/smodels-2.33/examples$

```

Figure 3 Output of smodels

Remarks

I conclude that smodels is an efficient program to compute stable model semantics. It gives output for logic program as per guessed values of variables of the rules.

4. Deliverable 3: Create a test OWL document and try to find a logic programming inference engine

Motivation

The main objective of this deliverable is to create an OWL document and try to find out a logic programming inference engine. The web ontology language (OWL) is designed to be used by applications that need to access the content of information instead of simply representing to humans. OWL provides interoperability of web content. It is supported by XML, RDF and RDF Schema (RDF-S). OWL is a family of knowledge representation languages for writing ontologies. OWL is endorsed by the World Wide Web Consortium. I used Pellet's OWL reasoner which is Java based open source OWL DL reasoner and used for reasoning about an OWL document.

Goal

The goal of this deliverable is to learn the Web Ontology Language (OWL) and then create a test OWL document of Computer Mouse. After creating OWL document, I have to use Pellet's OWL reasoner to reason about OWL document and generate OWL ontology.

Implementation and Results

I have learned OWL and used it to create an OWL document on computer mouse. OWL includes languages based on HTML, XML and various frame-based KR languages and knowledge acquisition approaches. OWL provides various capabilities for creating classes, subclasses, properties, defining instance and its operations. I have defined classes and subclasses for those classes for computer mouse in an OWL document. Also I have defined various properties and instances for computer mouse. OWL ontology described the data as a set of individuals and a set of property assertions which are related to each other. OWL ontology consists of a set of

axioms. These axioms place constraints on sets of individuals. These axioms provide semantics on the data which are explicitly provided. I have used OWL Lite. Here are following OWL Lite RDF schema features:

- Class: Class defines a group of individuals which are share same properties.
- rdfs:subClassOf: You can create a subclass of any class by using this feature. One class has one or more subclasses.
- rdf:Property: Property is used to define relations between classes.
- rdfs:subPropertyOf: Property is a sub property of one or more properties.
- rdfs:range: Range of properties limits individuals by initializing its value.

After creating test OWL document of Computer Mouse, I used Pellet's OWL reasoner to reason OWL document. It builds OWL ontology of OWL document. I used OWL reasoner's explanation example. It explains about the concept that I wrote in my OWL document.

Output

Why is led+scroll concept unsatisfiable?

Explanation:

```
led+scroll equivalentTo scroll
                        and uses some led
                        and part_of some optical
scroll subClassOf ps2_mouse
ps2_mouse equivalentTo mouse
                        and uses only not part_of some optical
                        and uses only not optical
```

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
```

Remarks

Pellet's OWL reasoner has provided an online demo to reason about OWL document. It provides various kinds of OWL documents reasoning examples. I recommend its use for reasoning OWL document as Pellet's reasoner is open source.

5. Deliverable-4: Implement finite automata closure algorithm

Motivation

The main objective of this deliverable is to make some changes in stable model program that I implemented in deliverable and implement finite automata closure algorithm. For implementing this algorithm, I need to make constraints for every rule. Constraints are effectively global requirements for every rule. I have added various models in property files. Constraints are part of rule which contains list of variables and models.

Goal

The goal is to implement finite automata closure algorithm and test output for various rules with various constraints.

Implementation

I have made some changes in my stable model program of deliverable-1. As Professor Pollet has suggested me to remove negative variable and add constraints in the rules, I have removed negative variable from the rules and added new constraints in the rules in the property file of my project. It contains regular expression in the form of 0 and 1. If any rule satisfies all of its constraints, it will use for finding stable model. If rule does not stratify all of its constraints, it will not use for finding stable model.

For example, property file look like following

```
R0=0*1?  
R1=1*01  
rule[0]=x1:|43R0;  
rule[1]=x2:x3|213R0,34R1;
```

Here, R0 and R1 are models and it contains regular expression 0*1? and 1*01 respectively. rule[0] and rule[1] are rules and it contains variable list and constraints. These Constraints have two parts: One is a list of variable and second is a model.

I need to check if the model contains a variable or not. From last example you can say that we need to check if R0 contains 43 or not. If it contains 43 then rule[0] is used to count stable model, which then we need to follow the same procedure that I have implemented in deliverable-1 which counts stable model for the rules which satisfy all of its constraints.

Result

1.

```
R0=0*1?  
R1=1*01  
rule[0]=x1:|43R0;  
rule[1]=x2:x3|213R0,34R1;
```

Output

```
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
```

2.

```
R0=0*  
R1=1*  
rule[0]=x2:|43R0;  
rule[1]=x2:x3|14R0;
```

Output

```
x2:|43R0;  
x2:x3|14R0;  
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 does not exist
For i=5
Stable Model does not exist
For i=6
Stable Model does not exist
For i=7
Stable Model does not exist
For i=8
Stable Model exist

Remark

If there are not any rules that satisfy all of their constraint then you will get an output like stable model exist for $i=0$. For $i=0$, all variable are guessed as false. It means that it shows false stable model. If there are not any rules in the model then all variable of rules are guessed false and it will count as false and output will be false stable model. So, I have checked cases which are not giving false stable model.

6. Summary

Answer Set programming is declarative programming. It is based on Stable Model. Stable Model is used for defining a declarative semantics for the rules of logic programs with negation as failure. I have computed Stable Model Semantics by using assume and reduce algorithm for computing stable model. I have tested my program for computing stable model semantics with various logic programs. I have learnt OWL which is designed to be used by an application for procession content of information. I have created a test OWL document. I used Pellet's OWL reasoned which is open source java based OWL-DL reasoner. It is basically building OWL ontology of OWL document. There are various examples available in Pellet's library which helped me very much to reason about OWL document.

7. Future Work

For my CS 298, I will extend the deliverables that I have worked on my CS 297 course. I will create and extend OWL inferences to support collections. I will extend OWL with finite automata constraints. The deliverables that I have worked in this semester have helped me to implement finite automata closure algorithm.

References

- [2009] Automata and Answer Set Programming. Victor Marek, Jeffery B. Remmel.
- [2000] Smodels: A System for Answer Set Programming. Llkka Niemela, Patrick Simons, Tommie Syrjanen.
- [2005] Logic programming with infinite sets. Douglas Cenzer, Jeffery B. Remmel, Victor Marek.
- [2003] Putting OWL in order: Pattern for sequences in OWL. Nice Drummond, Alan Rector, Robert Stevent, Georgina Moulton.
- [2005] OWL-AA: Enriching OWL with instance recognition semantics for automated semantic annotation. Yihong Ding, David Embley, Stephen Liddle.