Logic Programs with Tests *

Marion Mircheva

Institute of Mathematics & Computer Science
Bulgarian Academy of Sciences
"Acad. G.Bonchev" str. bl. 8, Sofia 1113 , Bulgaria
E-mail: marion@bgearn.bitnet

Abstract. We extend logic programming to deal with logic programs that include new truth functional connectives called tests. Stable Model Semantics and Three Valued Stable Model Semantics are extended to give meaning to programs with tests.
We consider three possible applications of such programs. It is shown how to define a particular semantics in terms of another one with the help of an appropriate transformation of normal programs into programs with tests. Our approach can be applied for resolving inconsistency in logic programs with explicit negation. Programs with tests can serve as a promising tool for eliminating or encoding integrity constraints within particular background semantics.

1 Introduction

In this paper we propose a general framework for extending two valued and three valued stable model semantics to programs with new truth functional connectives called tests. We consider at least three possible applications of logic programs with such metatools.

For some well known semantics we present a method to calculate a particular semantics (set of models) of a program in terms of some other semantics of the program obtained from the initial one after a suitable transformation. The resulting program however includes a new test connective. For example the stable models of the program

\[ a \leftarrow \sim b \]
\[ b \leftarrow \sim a \]

\[ M_1 = \{a, \sim b\} \text{ and } M_2 = \{b, \sim a\} \] coincide with the three valued stable models of the following program

\[ a \leftarrow \sim b, \sim t(a) \]
\[ b \leftarrow \sim a, \sim t(b) \]

* This paper has been developped during my visit at the Institute for Logic, Complexity and Deduction systems, University of Karlsruhe, Germany. It was supported by the European Economic Community within the framework of the Community’s Action for Cooperation in Science and Technology with Central and Eastern European Countries under the contract number CIPA3510CT924716.
The modal\textsuperscript{2} operator $t(a)$ has the meaning:

\[
\text{if } a = u \text{ then } t(a) = \text{true}, \text{ otherwise } t(a) = \text{false}.
\]

This test operator distinguishes undefined value from the Boolean ones. Obviously its meaning can be given by the truth table:

<table>
<thead>
<tr>
<th>$a$</th>
<th>$t(a)$</th>
<th>$\sim t(a)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$u$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>


The next application concerns extended logic programs, that are programs with default ($\sim$) and explicit ($\neg$) negations. We propose a method for resolving inconsistency caused by the interaction between default and explicit negations within three valued stable semantics. We present an alternative extension of three valued stable models to programs with explicit negation that gives meaning to every logic program. Our models can be calculated within three valued stable semantics over logic programs with a special test connective.

Within our framework, any background semantics could be easily adapted to deal with programs augmented with integrity constraints presented as denials $a_1, \ldots, a_n$. This can be done either by withdrawing the models that brake up the constraints or by revising the models in a satisfactory way. The revision strategy has to be encoded in the meaning of the test connectives that has to be used (witness section 5).

The paper is organized as follows. Section 2 involves the syntax of our framework. In Section 3 we formally describe our extension of two valued stable models to programs with tests and present a case study example. Section 4 extends Stable3 of Przymusinski to include programs with tests and presents two case study examples. Section 5 describes a new extension of three valued stable models to programs with explicit negation. An appropriate transformation into a class of programs with guard connective is considered here. Section 6 discusses related papers and gives some conclusive remarks.

2 General Framework: Normal Logic Programs with Test Formulae

In this section we extend the propositional logic programming to deal with programs that include some new truth functional connectives, called test operators.

\textsuperscript{2} We use the term "modal" according to Polish logical tradition: each connective that transforms multi valued domain into Boolean one is called modal.