Pstable Tool and Its Aplication

Fernando Uceda Ponga[1] Fernando Zacarías Flores[2] Dionicio Zacarías Flores[3]

***Abstract*—We present an implementation of pstable model semantics. Our implementation uses the well known tools: MiniSat and Lparse. Also we show the utility of an automatic demonstrator implemented with these tools for applications where the logic has consistence problems.**

***Index Terms*—Satisfactibility, Pstable, Text Implication.**

I. Introduction

The stable model semantics for logic programming support it self within the non-monotonic reasoning, [1]. This approach has been widely used and it is perhaps the most well known semantics for knowledge representation. The Pstable model semantics, introduced in [2] shares several properties with stable model semantics, but it is closer to classical logic than stable. Pstable model semantics has at least the same expressiveness of stable, this means we can model anything that is modeled in stable, and how we proof with our experiments Pstable can model things, hard or may be impossible of model with stable models. This is why we are interested in a good implementation. The purpose of this paper is to show the utility and reach of this tools.

II. Background

We introduce some concepts which are needed for further understanding of the work and the results presented on this document.

Pstable Model Semantics

The definition of the pstable model semantics was shown in [2] and is the following:

Let P be a normal program, and M a set of atoms. We say that M is a pstable model of P if M is a model of P and $Red_M(P) \models M$.

Textual Implication

This term is used when from the semantic of a text in natural language, is possible to infer, another text in natural language. More specific, if the true of one sentence implies the true of the other one, also call hypothesis [3].

Paper submmited on February 26, 2007. Benemérita Universidad Autónoma de Puebla – Facultad de Ciencias de la Computación,y FCFM, Puebla, Pue., México, C.P. 72570, moderm_cruzades@hotmail.com[1], fzflores@yahoo.com.mx[2], jdzf55@yahoo.com.mx[3].

III. Pstable Solver

The previous analysis of graph problems and text implication solving, add to the intention of solving by using logic, make a big gap, mainly because the classic logic requires complex modeling, just for pre-solving inconsistencies. There's a logic extension able to solve inconsistencies, so that is the logic modeling we will use for our implementation.

Logical solution tools are available, unfortunately ASP is still weak for problems as simple as:

a→b and b→a and ¬b.

this look like a contradiction, strangely if we make a deep look of this we can see it as:

(a or ¬b) and (b or ¬a) and ¬b.

is a sentence with a main characteristic, the property of having satisfiable values for a simple solution implying ¬b and ¬a, even been false values they make true the problem, this models are stable for the solution and they are also solution of the original problem form.

Satisfiability can be done by many SAT solvers, we choose the MINISAT [4] solver, because won all the industrial categories of the SAT 2005 competition and the SAT-Race 2006. MiniSat is a good starting point both for future research in SAT, and for applications using SAT, like all sat solvers MINISAT use the DIMACS CNF format as an input, so if we wanna make a PSTALBE solver it's gonna be important first convert traditional logic input to the exclusively disjunctive form.

First approach to make this made us look to Lparse most of all because works with variable-free programs that are quite cumbersome to generate by hand also is a front-end that adds variables (and a lot of other stuff) to the accepted language of smodels [5] and generates a variable-free simple logic program that can be given to smodels and we suppose that the resulting file will be easier to convert in DIMACS CNF file. to do the parsing, but we don't get the expected result as we show as follows:

input

```
a :- b, not c.
c :- b, not a.
e :- c.
b.
```

output

```
1 1 0 0
1 2 1 0 3
1 3 1 1 4
1 4 1 1 3
0
1 b
2 e
3 c
4 a
0
B+
0
B-
0
1
```

is easy see how its too bounden in with smodels format and that only lead us to a mayor level of complexity in the final parsing, and it's gonna make harder, analysis-matching of the output with the program for performing the adjustment and actualization in the body.

Taking this experiences in consideration we decide to develop a custom parser this parsing step and conversion thread was implemented in java, because we look forward, and the further work will end-up in a online portal [6], once we choose this path, the proofs gave us java as a good environment for integrate all the work.

Later on, to parsing of a translation is done a input program like the one below

```
c->b ^ d.
b->a.
a->c.
not b.
```

At once, the conversion of program looks like this:

```
c
c comments
c 1 c
c 2 b
c 3 d
c 4 a
p cnf 4 5
1 -3 0
1 -2 0
2 -4 0
4 -1 0
-2 0
```

taking this as an example, let us show how the program work, the variables are not stored and converted by alphabetical or numeric order, they are parsed by order of apparition, the variables are stored in a modified hash table (see table 1) with sub-classing, if the memory can be seen the table be this:

Table 1. Hash table

	Var	Not	Fact	And	->	Or
c	1	False	False	0	3,2	0
b	2	True	False	3	4	0
d	3	False	False	2	0	0
a	4	False	False	0	1	0

(The table use only numeric values, we put label for easier understanding)

Structured as a dynamic structure has no limits of variables and connections between them, also make trivial, add actions or deletion, but can increase the time of searching a negated variable.

Search of a negated variable is tricky because SAT solvers use the disjunctive for of the implication this mean that a variable may be negated in the new format, so we make a extra table to index the first one:

Table 2. Table for search process

	Complement
c	False
b	True
d	True
a	True

The table is shown very simplified but, it contains the information of the row where the variable occur and link the search process with the insert process.

These two structures make easy the iterative process of calculate PSTABLE models. The process is simple and have these steps:

I. Parsing
II. Build Dynamics tables
III. Convert dynamics tables in to DIMACS CNF file
IV. Invoke minisat
V. Call answer analyst
VI. search negated variables that are not in the solution model or in the assuming set
VII. insert negated row to the program
VIII. show PSTABLE model

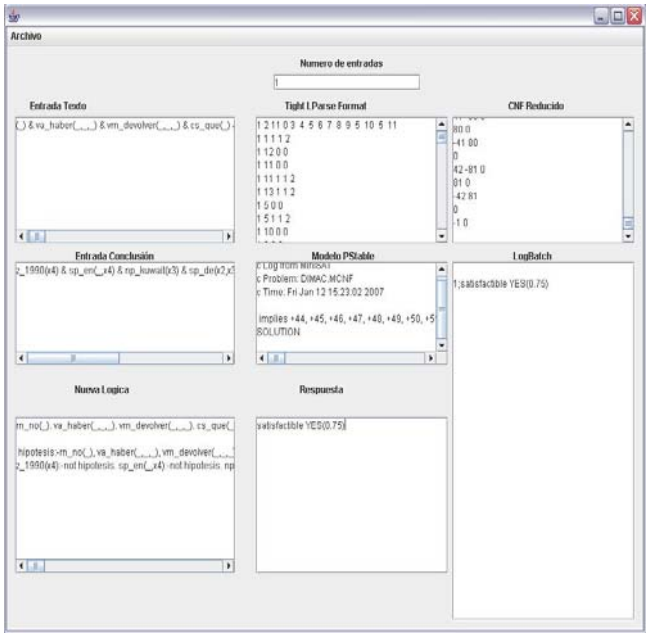
Steps IV to VII are performed until the sat solver is unable to find a solution that can satisfy the program and all possible models where checked, in this moment all models that can't be satisfied

IV. Text Implication Plug in

Taking the Pstable-Solver program as a core-system with the ability of accept plug-ins we develop a plug-in with the characteristic of convert logical expressed text in to a logical inconsistent program, this because we want to express the program as a automatic theorem demonstrator using the contradiction way. With this, we obtain the PStable model of the text and the implication we want to prove is or not true, obviously if the Pstable model have only true atoms and is

equal to the facts in the logical program, we can say that the logic is tight and the text truly implies what the input suppose. By the other side is all the atoms from the supposition are false and the minimal model can not make true any of its atoms, we can know for sure that the logic is not tight and is not adventure to say "the text implication is false", how ever if part of the atoms are true and other false, the logic is obviously weak, in this cases we get advantage of the Pstable characteristics for the analysis of the false atoms, the goal is find a model where this atoms can be true. If the atoms can be true in a model the conjunction of the models gave us an answer true of false as a value of the implication.

The application with the plug in looks like this:



The application can support batching and multiple log solving.

To illustrate the complexity of this calculus we compare our demonstrator with the automatic demonstrator Otter. This demonstrator is well known as one of the best automatic tools.

For the challenge we take 322 text implications of different type in logic form, in 77 the implication was true, 235 was false and 10 where so complex that even people could not say if there's or not implication.

With this set of proofs Otter answer to 3 problems with a yes, those were corrects, but the 319 remaining problems could not been solve and was unable to gave an answer, this results was very poor but the tools we develop gave much better results, anwer 32 correct yes 235 corrects no and 3 corrects unknown , to 45 problems answer unknown but the answer must be yes, we think that this is because the logic was very weak, even with that the pstable model proof to be a better aproch to the solution of this problem.

V. Conclusion

We decide to do this investigation, and gave this direction, because, hybrid systems and fuzzy solutions are taking lead in this day so we want to prove that, classic logic is a prower full tool if it's uses properly. During the realization of this experiments we discover that our Textual Implication tool depend by the moment of the accuracy, existed in the parsing tool which translate natural language to logic, we really think that this tool can be improve with Pstable or Stable models, Any way our tool show to be more precise, and we came to the conclusion that is because the logic we are using and the way we pass the problem to the logic, in fact the more important thing is the right modeling of the problem that why our tool is less sensitive to a bad parsing than, ordinal logic demonstrators.

References

[1] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, Proceedings of the Fifth International Conference on Logic Programming, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press, URL citeseer. ist.psu.edu/gelfond88stable.html.

[2] Mauricio Osorio Galindo and Juan Antonio Navarro Pérez and José R. Arrazola Ramírez and Verónica Borja Macías". Logics with common weak completions. Journal of Logic and Computation, 2006. URL doi: 10.1093/logcom/exl013

[3] Técnicas Aplicadas al Reconocimiento de Implicación Textual, Jesús Herrera, Anselmo Peñas, Felisa Verdejo, Departamento de Lenguajes y Sistemas Informáticos, Universidad Nacional de Educación a Distancia Madrid, España, {jesus.herrera, anselmo, felisa}@lsi.uned.es

[4] MiniSat is a minimalistic, open-source SAT solver, developed to help researchers and developers alike to get started on SAT. It is released under the MIT licence, and is currently used in a number of projects

[5] The program smodels is an implementation of the stable model semantics for logic programs. Smodels can be used either as a C++-library that can be called from user programs or as a stand-alone program together with a suitable front-end. The main front-end is lparse.

[6] Portal can be available for every one to solve their problems, portal will be capable of learning so if a solution is already done we don't calculate again, or if a problem is not solved propertly the user can enter the right solution (this is a scenario that rarely occur, and we suppose will have a probability very near to 0).