The *aspps* system

Deborah East[1] and Mirosław Truszczyński[2]

[1] Department of Computer Science, Southwest Texas State University,
San Marcos TX 78666, USA
[2] Department of Computer Science, University of Kentucky,
Lexington KY 40506-0046, USA

1 Introduction

The *aspps* system is an answer-set programming system based on the extended logic of propositional schemata [2], which allows variables but not function symbols in the language. We denote this logic PS^+. A theory in the logic PS^+ is a pair (D, P), where D is a set of ground atoms (only constant symbols as arguments) representing an *instance of a problem* (input data), and P is a set of PS^+-clauses representing a *program* (an abstraction of a problem). The meaning of a PS^+-theory $T = (D, P)$ is given by a *family* of PS^+-models [2].

The *aspps* program and an associated module *psgrnd* allow one to compute models of PS^+ theories. First, a theory is grounded using the program *psgrnd*. The theory obtained in this way is an input to the *aspps* solver, which computes models of the ground theory and, hence, of the original theory, as well.

The language of logic PS^+ accepted by *psgrnd* includes special constructs, such as those to model cardinality constraints on sets. The theories produced by *psgrnd* maintain the structure of these constructs. The *aspps* solver is based on the Davis-Putnam algorithm for satisfiability checking. There is however a major difference. The solver accepts and takes advantage of special constructs in the ground theories to improve search.

Both *psgrnd* and *aspps*, examples of PS^+-programs and the corresponding performance results are available at `http://www.cs.uky.edu/ai/aspps/`.

In this document we update an earlier note describing our system [1]. We present the syntax of PS^+-programs that is accepted by *psgrnd*. We also provide instructions for executing *psgrnd* and *aspps* and discuss some of the available options. For theoretical underpinnings of *aspps*, we refer the reader to [2].

2 PS^+-theories

The language of PS^+ contains variable, constant and predicate symbols but not function symbols. Predicates in a PS^+ theory are classified as *data* and *program* predicates. Ground atoms built of data predicates represent a problem instance. They form the component D (data) of a theory (D, P). The program P consists of clauses built of atoms involving both data and program predicates. Clauses are written as implications and explicit negation of atoms is not allowed (the

implication symbol is omitted if a clause has an empty conjunction of atoms as the antecedent). The program is written to capture all the relevant constraints specifying the problem to be solved.

The key difference between the logic PS^+ and the logic of propositional schemata is in the definition of a model. Following the intuition that computation must not modify the data set, a set of ground atoms M is a model of a PS^+ theory (D, P) if M is a propositional model of the grounding of (D, P) and if it coincides with D on the part of the Herbrand Universe given by data predicates.

Data for an instance is given in one or more data files. The following set of atoms is an example of a data file for an instance of graph coloring.

$vtx(1).\ \ vtx(2).\ \ vtx(3).\ \ vtx(4).$
$edge(1,4).$
$edge(1,2).$
$edge(3,2).$
$clr(r).$
$clr(g).$
$clr(b).$

The rule file (only one rule file is allowed) includes a preamble, where program predicates are defined and variables declared. The preamble in the program file restricts the way program predicates are grounded. The following lines form an example of the preamble for a graph-coloring program.

pred $color(vtx, clr).$
var $vtx\ X,\ Y.$
var $clr\ K,\ C.$

The program predicate *color* is a binary predicate. Its first argument must be a constant from the extension of the data predicate *vtx*, as defined in the data file. Its second argument must be a constant from the extension of the data predicate *clr*. Only unary data predicates can be used to define *types* of arguments of program predicates. Binary data predicates can also be used in the predicate definitions to restrict their extensions. An example (not related to graph coloring) is:

pred $hc(vtx, vtx) : edge.$

This statement restricts the extension of the program predicate *hc* to a subset of the extension of the data predicate *edge*. The preamble also declares types of variable symbols used in the program. Variables are declared by means of unary data predicates (examples are given above). The declaration of variables allows for more efficient grounding and further error checking.

The preamble is followed by clauses describing constraints of the problem. An example of a program for graph coloring follows.

Clause 1. $color(X, r)|color(X, g)|color(X, b).$
Clause 2. $color(X, K), color(X, C) \rightarrow K == C.$
Clause 3. $color(X, K), color(Y, K), edge(X, Y) \rightarrow .$

Clause 1 ensures that each vertex is assigned at least one color. Clause 2 enforces that at most one color is assigned to each vertex. The last clause prohibits assigning the same color to vertices connected by an edge.

In some cases, the consequent of a clause must be a disjunction of a set of atoms that depends on a particular data instance. To build such disjunctions, we introduced in the language of the logic PS^+ the notion of an *e-atom*. An example of an e-atom (in the context of our graph-coloring setting) is $color(X, \_)$. It stands for the disjunction of all atoms of the form $color(X, c)$, where c is a constant from the extension of the data predicate *clr*. The use of the special construct for existential atoms (e-atoms) allows us to rewrite Clause 1 as "$color(X, \_)$.". Our current version of logic PS^+ allows also for more complex variants of e-atoms.

Another powerful modeling concept in the language of logic PS^+ is that of a cardinality atom. An example of a cardinality atom is $k\{color(X, \_)\}m$. We interpret the expression within the braces as a specification of the *set* of all ground atoms of the form $color(X, c)$, where c is a constant from the extension of the data predicate *clr*. The meaning of the atom $k\{color(X, \_)\}m$ is: at least k and no more than m atoms of the form $color(X, c)$ are true.

Using the concept of a cardinality atom, we can replace Clause 1 and 2 with a single clause "$1\{color(X, \_)\}1$.". We preserve the structure of cardinality atoms in ground theories and take advantage of the structure in the solver.

In addition to the program and data predicates, the *aspps* implementation includes *predefined* predicates and function symbols such as the equality operator ==, arithmetic comparators <=, >=, < and >, and arithmetic operations +, −, ∗ ,/, $abs()$ (absolute value), $mod(N, b)$, $max(X, Y)$ and $min(X, Y)$. We assign to these symbols their standard interpretation. We emphasize that the domains are restricted only to those constants that appear in a theory.

3 Grounding PS^+-theories

The grounding of logic PS^+ programs is performed by the module *psgrnd*. When grounding, we first evaluate all expressions built of predefined operators. We then form ground instantiations of all program clauses. Next, we evaluate and simplify away all atoms built of predefined predicates. We also simplify away all atoms built of data predicates (as they are fully determined by the data part). Therefore, the ground PS^+-theory contains only ground atoms built of program predicates.

The required input to execute *psgrnd* is a single program file, one or more data files and optional constants. If no errors are found while reading the files and during grounding, an output file is constructed. The output file is a machine readable file. Included in the output file is a mapping from the program predicates to machine readable form for each predicate. A list of all ground atoms built of program predicates that were determined during grounding is also given. The name of the output file is a catenation of the constants and file names with the extension **.aspps**.

***psgrnd* -r rfile -d dfile1 dfile2** ... [**-c c1=v1 c2=v2** ...]

Required arguments

-r **rfile** is the file describing the problem. There must be exactly one program file.
-d **datafilelist** is a list of one or more files containing data that will be used to instantiate the theory.

Optional arguments

-c **name=value** This option allows the use of constants in both the data and rule files. When **name** is found while reading input files it is replaced by **value**; **value** can be any string that is valid for the data type. If **name** is to be used in a range specification, then **value** must be an integer.

4 Solving PS^+-theories

The solver *aspps* is used to compute models of the ground PS^+-theory produced by *psgrnd*. The solver takes advantage of the special constructs by allowing search to branch based on the status of cardinality atoms. This possibility often significantly reduces the size of the search space. In addition, the heuristics for selecting atoms on which to branch are guided by the structure of the original problem, preserved by the grounding process.

The name of the file containing the theory is input on the command line. After executing the *aspps* program, a file named aspps.stat is created or appended with statistics concerning this execution of *aspps*.

***aspps* -f filename** [**-A**] [**-P**] [**-C** [**x**]] [**-L** [**x**]] [**-S name**]

Required arguments

-f **filename** is the name of the file containing a theory produced by *psgrnd*.

Optional arguments

-A Prints atoms that are true in the computed model in readable form.
-P Prints the input theory and then exits.
-C [**x**] Counts the number of solutions. This information is recorded in the statistics file. If **x** is specified it must be a positive integer; *aspps* stops after finding **x** solutions or exhausting the whole search space, whichever comes first.
-S **name** Shows positive atoms with predicate **name**.

References

1. D. East and M. Truszczyński. *aspps* — an implementation of answer-set programming with propositional schemata. In *Proceedings of Logic Programming and Nonmonotonic Reasoning Conference, LPNMR 2001*, LNAI 2173, pages 402–405, Springer Verlag, 2001.
2. D. East and M. Truszczyński. Propositional satisfiability in answer-set programming. In *Proceedings of Joint German/Austrian Conference on Artificial Intelligence, KI'2001*, LNAI 2174, pages 138–153, Springer Verlag, 2001.