Answer Set Programming[*]

Answer set programming (ASP) is a form of declarative programming oriented towards difficult combinatorial search problems. It has been applied, for instance, to plan generation and product configuration problems in artificial intelligence and to graph-theoretic problems arising in VLSI design and in historical linguistics.

Syntactically, ASP programs look like Prolog programs, but the computational mechanisms used in ASP are different: they are based on the ideas that have led to the development of fast satisfiability solvers for propositional logic.

Answer set programming has emerged from interaction between two lines of research — on the semantics of negation in logic programming [Gelfond and Lifschitz, 1988] and on applications of satisfiability solvers to search problems [Kautz and Selman, 1992]. It was identified as a new programming paradigm in [Lifschitz, 1999; Marek and Truszczyński, 1999; Niemelä, 1999].

Computing Answer Sets with LPARSE and SMODELS

In this section we discuss the use of the answer set solver created by Ilkka Niemelä's group at the Helsinki University of Technology. The system consists of two programs, LPARSE and SMODELS; the former is a preprocessor. It can be downloaded from its web site,

`http://www.tcs.hut.fi/Software/smodels/ .`

The syntax of the input language of LPARSE is more limited in some ways than the class of programs defined in the previous lecture notes ("Answer Sets"), but it includes many useful special cases. In the input language of LPARSE, the head of a rule can be

- an atom,
- empty (that is, $\perp$),
- an expression of one of the forms

$$\{F_1, \dots, F_n\}^c$$

$$l \leq \{F_1, \dots, F_n\}^c$$
$$\{F_1, \dots, F_n\}^c \leq u$$
$$l \leq \{F_1, \dots, F_n\}^c \leq u$$

[*]Some parts of this note are taken from the survey "Introduction to Answer Set Programming" by Vladimir Lifschitz.

provided that the formulas F_i are atoms.

In the last case, the superscript ^{c} and the sign $\leq$ are dropped. For instance,

$$\texttt{1}\{\texttt{p},\texttt{q}\} \tag{1}$$

in the head of a rule stands for

$$1 \leq \{p, q\}^c. \tag{2}$$

The body can be a conjunction (possibly empty) of

- atoms possibly preceded with *not*,
- expressions of the forms

$$l \leq \{F_1, \ldots, F_n\}$$
$$\{F_1, \ldots, F_n\} \leq u$$
$$l \leq \{F_1, \ldots, F_n\} \leq u$$

provided that each F_i is an atom possibly preceded with *not*.

In the last case, the sign $\leq$ is dropped. For instance, expression (1), when it occurs in a body of an LPARSE rule, stands for

$$1 \leq \{p, q\}.$$

Note that this expression is different from (2).

In the input language of LPARSE, `:-` stands for $\leftarrow$, and each rule is followed by a period.

If we want to find, for instance, the answer sets for the program

$$\begin{array}{l} p \leftarrow \textit{not } q \\ q \leftarrow \textit{not } p \\ r \leftarrow p \\ r \leftarrow q \end{array}$$

we create the file

```
p :- not q.
q :- not p.
r :- p.
r :- q.
```

called, say, `input.lp` . Then we invoke LPARSE and SMODELS as follows:

```
% lparse input.lp | smodels 0
```

The zero at the end indicates that we want to compute all answer sets; a positive number k would tell SMODELS to terminate after computing k answer sets. The default value of k is 1. The main part of the output generated in response to this command line is the list of the program's answer sets:

```
Answer: 1
Stable Model: r p
Answer: 2
Stable Model: r q
```

("stable model" means "answer set").

Problem 1 Use SMODELS to find the answer sets for the program

$$\begin{array}{l}\{p,q\}^c \\ \leftarrow p,q.\end{array}$$

A group of rules that follow a pattern can be often described in a compact way using variables. Variables must be capitalized. Consider, for instance, the programs Π_n

$$p_i \leftarrow \textit{not}\; p_{i+1} \qquad (1 \le i \le n).$$

To describe Π_7, we don't have to write out each of its 7 rules. Let's agree to use the symbol `index` to represent a number between 1 and 7. We can write Π_7 as

```
index(1..7).
p(I) :- not p(I+1), index(I).
```

The auxiliary symbols used in the input language of LPARSE to describe the ranges of variables, such as `index`, are called domain predicates. The "definitions" of domain predicates, such as `index(1..7)`, tell LPARSE how to substitute specific values for variables in schematic expressions, such as `p(I) :- not p(I+1)`. Grounding—translating schematic expressions into sets of rules—is the main computational task performed by LPARSE.

When the input file uses domain predicates, the output of SMODELS lists the objects satisfying each domain predicate along with the elements of the answer set:

```
Answer: 1
Stable Model: p(1) p(3) p(5) p(7) index(1) index(2)
index(3) index(4) index(5) index(6) index(7)
```

Information on the extents of domain predicates in the output can be suppressed by including a `hide` directive in the input file:

```
hide index(_).
```

The family of programs Π_n can be described by a schema with the parameter n:

```
index(1..n).
p(I) :- not p(I+1), index(I).
```

When this schematic description is given to LPARSE as input, the value of the constant n should be specified in the command line using the option `-c`, as follows:

```
% lparse -c n=7 p.lp | smodels 0
```

Problem 2 Consider the program obtained from Π_n by adding the rule

$$p_{n+1} \leftarrow \textit{not}\, p_1.$$

How many answer sets does this program have, in your opinion? Check your conjecture for $n = 7$ and $n = 8$ using SMODELS.

Variables can be also used "locally" to describe the list of formulas in a cardinality expression. The domain predicate characterizing the range of a local variable is placed inside the braces. For instance, the rule

$$1 \leq \{p_1, \ldots, p_n\} \leq 1$$

can be represented in an LPARSE input file by the lines

```
index(1..n).
1{p(I) : index(I)}1.
```

Problem 3 Consider the program

$$1 \leq \{p_{i1}, \ldots, p_{in}\}^c \leq 1 \qquad (1 \leq i \leq n),$$

where n is a positive integer. How many answer sets does this program have, in your opinion? Check your conjecture for $n = 3$ using SMODELS.

A detailed description of the input language of LPARSE, as well as the complete list of options of LPARSE and SMODELS, can be found in the system's manual, available at its web site.

Besides SMODELS, there are some other answer set solvers that use LPARSE for grounding, such as ASSAT[1] and CMODELS[2]. Among other existing systems for computing answer sets, DLV[3] stands out by virtue of two things: it uses a very efficient grounding mechanism, and it can process rules containing disjunctions of several atoms in the head.

Methodology of ASP

To solve a problem using ASP means to write a program whose answer sets correspond to solutions, and then find a solution using an answer set solver, such as SMODELS. The basic approach to writing such a program is known as the "generate-and-test" strategy. First we write a group of rules whose heads contain choice expressions so that the

[1]`http://assat.cs.ust.hk/` .
[2]`http://www.cs.utexas.edu/users/tag/cmodels/` .
[3]`http://www.dbai.tuwien.ac.at/proj/dlv/` .

answer sets for this group of rules correspond to "potential solutions"—an easy-to-describe superset of the set of solutions. Then we add a group of constraints that weed out the potential solutions that are not solutions.

Consider, for instance, the use of this method to solve the n-queens problem. The goal is to place n queens on an $n \times n$ chessboard so that no two queens would be placed on the same row, column or diagonal. A solution can be described by a set of atoms of the form $q(i, j)$ $(1 \leq i, j \leq n)$; including $q(i, j)$ in the set indicates that there is a queen at position (i, j). A solution is a set X satisfying the following conditions:

1. The cardinality of X is n.
2. X does not contain a pair of different atoms of the form $q(i, j)$, $q(i', j)$ (two queens in the same row).
3. X does not contain a pair of different atoms of the form $q(i, j)$, $q(i, j')$ (two queens in the same column).
4. X does not contain a pair of different atoms $q(i, j), q(i', j')$ with $|i' - i| = |j' - j|$ (two queens on the same diagonal).

The sets satisfying Conditions 1 and 2 can be described by the rules

$$1 \leq \{q(1, j), \ldots, q(n, j)\}^c \leq 1 \qquad (1 \leq j \leq n)$$

(exactly one queen in each row). These rules form the "generate" part of our program. The "test" part consists of the constraints expressing Condition 3

$$\leftarrow q(i, j), q(i, j') \qquad (1 \leq i, j, j' \leq n;\ j < j')$$

and Condition 4

$$\leftarrow q(i, j), q(i', j') \qquad (1 \leq i, i'j, j' \leq n;\ j < j';\ |i' - i| = j' - j).$$

Here is a representation of this program in the input language of LPARSE:

```
number(1..n).

1{q(I,J) : number(I)}1 :- number(J).

:- q(I,J), q(I,J1),
   number(I;J;J1), J<J1.

:- q(I,J), q(I1,J1),
   number(I;I1;J;J1), J<J1, abs(I1-I)==J1-J.
hide number(I).
```

The expression `number(I;J;J1)` is the LPARSE abbreviation for

```
number(I), number(J), number(J1).
```

Note the conditions `J<J1` and `abs(I1-I)==J1-J` in the bodies of rules. By including them, we tell LPARSE how to limit the choice of values used in the process of grounding.

Problem 4 Use SMODELS to find all solutions to the 8 queens problem that (a) have a queen at (1,1); (b) have no queens in the 4×4 square in the middle of the board.

In many cases, the constraints in the test part of the program use auxiliary atoms, defined in terms of the atoms occurring in the generate part. The definitions of the auxiliary atoms form then a third component of the program, besides the "generate" and "test" parts—its "define" part.

Consider, for instance, the problem of finding a Hamiltonian circuit in a given graph—a closed path that visits every node exactly once. A Hamiltonian circuit in a graph G can be thought of as a subgraph C of G that has the same set V of vertices as G and satisfies two conditions:

1. Every vertex is adjacent in C to exactly two vertices.
2. Every vertex is reachable in C from some fixed vertex v_0.

The program below represents an edge $\{u, v\}$ of G by the atom *in*(u, v), where $u < v$ ($<$ is a fixed total order on V). The presence of this atom in an answer set indicates that the edge $\{u, v\}$ is included in C. The generate part of the program consists of the rules

$$\{in(u, v)\}^c \tag{3}$$

for all edges $\{u, v\}$ of G $(u < v)$. Thus any set of edges is a potential solution. To encode the two conditions that characterize Hamiltonian circuits, we introduce the auxiliary atoms *adj*(u, v) $(u < v)$ that represent adjacency in C. These atoms are "defined" in terms of *in*(u, v) by the rules

$$adj(u, v), adj(v, u) \leftarrow in(u, v) \tag{4}$$

for all edges $\{u, v\}$ of G $(u < v)$. Then Condition 1 above can be expressed by the constraints

$$\begin{array}{l} \leftarrow \{adj(u, v)\ :\ v \in V\} \leq 1 \\ \leftarrow 3 \leq \{adj(u, v)\ :\ v \in V\} \end{array} \tag{5}$$

for all $u \in V$. For Condition 2, we introduce the auxiliary atoms $r(u)$ that express the reachability of u from v_0. They are "defined recursively" by the rules

$$\begin{array}{l} r(v_0) \\ r(u) \leftarrow r(v), adj(u, v) \end{array} \tag{6}$$

for all edges $\{u, v\}$ of G. Using these atoms, we express Condition 2 by the constraints

$$\leftarrow \textit{not}\; r(u) \tag{7}$$

for all $u \in V$.

Answer sets for program (3)–(7) are in a 1–1 correspondence with the Hamiltonian circuits in G. Rules (4) and (6) form the define part of the program, and rules (5) and (7) form the test part.

The following file `hc` is an LPARSE encoding of program (3)–(7), in which we assume that the vertices of G are integers, and $v_0 = 0$:

```
{in(U,V)} :- edge(U,V).

adj(U,V) :- in(U,V), vertex(U;V).
adj(V,U) :- in(U,V), vertex(U;V).

:- {adj(U,V) : vertex(V)}1, vertex(U).
:- 3{adj(U,V) : vertex(V)}, vertex(U).

r(0).
r(U) :- r(V), adj(U,V), vertex(U;V).

:- not r(U), vertex(U).

hide.
show in(_,_).
```

(The last two lines tell SMODELS to display the `in` atoms only.)

This file needs to be appended to a description of G in the form of a definition of the domain predicates `vertex` and `edge`. For instance, the file

```
vertex(0..7).

edge(U,U+1) :- vertex(U), U!=3, U!=7.
edge(0,3).
edge(4,7).
edge(U,U+4) :- vertex(U), U<4.
```

describes the vertices and edges of a cube. Assuming that this file is called `cube`, SMODELS can be instructed to find a Hamiltonian circuit by the command

```
% lparse cube hc | smodels
```

Problem 5 The vertices of graph G_n are the points (x, y) with coordinates $x, y \in \{0, \ldots, n-1\}$; two vertices are adjacent if the distance between them is 1. Use SMODELS to find out whether G_5 has a Hamiltonian circuit.

In each of the following exercises, show how to solve the given computational problem using SMODELS. For the first three exercises below, as in the Hamiltonian circuit example, use specific data (of your choice) to test your program.

Problem 6 A *clique* in a graph G is a subset of its vertices whose elements are pairwise adjacent.[4] Determine whether G has a clique of cardinality n.

Problem 7 You are organizing a large New Year's Eve party. There will be n tables in the room, with m chairs around each table. You need to select a table for each of the guests, who are assigned numbers from 1 to mn, so that two conditions are satisfied. First, some guests like each other and want to sit together; accordingly, you are given a set A of two-element subsets of $\{1, \dots, mn\}$, and, for every $\{i, j\}$ in A, guests i and j should be assigned the same table. Second, some guests dislike each other and want to sit at different tables; accordingly, you are given a set B of two-element subsets of $\{1, \dots, mn\}$, and, for every $\{i, j\}$ in B, guests i and j should be assigned different tables. The goal is to find such a seating arrangement or determine that this is impossible.

Problem 8 You are in charge of assigning referees to the papers submitted to a conference. Each of the n submissions needs to be assigned to a few referees from among the m members of the Program Committee. The PC members have read the abstracts of all submissions, and each of them gave you the numbers of the submissions that he is qualified to referee; let A_i ($1 \leq i \leq m$) be the subset of $\{1, \dots, n\}$ given to you by the i-th committee member. You need to select, for each i, a set X_i of papers to be assigned to the i-th committee member so that three conditions are satisfied. First, each X_i should be a subset of A_i. Second, the cardinality of each X_i should be between a lower bound l and an upper bound u. Third, each paper should be assigned to exactly k referees. The goal is to find such an assignment of papers to referees or determine that this is impossible.

Problem 9 Each of four men owns a different species of exotic pet. From the following clues, can you figure out each man's full name and what kind of pet he owns?

1. Mr. Engel (whose pet is named Sparky), Abner and Mr. Foster all belong to a club for owners of unusual pets.
2. The iguana is not owned by either Chuck or Duane.
3. Neither the jackal nor the king cobra is owned by Mr. Foster.
4. The llama does not belong to Duane (whose pet is named Waggles).
5. Abner, who does not own the king cobra, is not Mr. Gunter.
6. Bruce and Mr. Foster are neighbors.

[4] `http://mathworld.wolfram.com/Clique.html` .

7. Mr. Halevy is afraid of iguanas.

Problem 10 A set A of integers is called *sum-free* if there are no numbers x, y in A such that $x + y$ is in A also (x and y do not need to be different). The *Schur number* $S(k)$ is the largest integer n for which the interval $\{1, \ldots, n\}$ can be partitioned into k sum-free sets.[5] Write a program that computes $S(k)$, and try it for $k = 1, 2, 3$.

References

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080, 1988.

[Kautz and Selman, 1992] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*, pages 359–363, 1992.

[Lifschitz, 1999] Vladimir Lifschitz. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.

[Marek and Truszczyński, 1999] Victor Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

[Niemelä, 1999] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

[5] See `http://mathworld.wolfram.com/SchurNumber.html` .