4 Methodology of Answer Set Programming

Recall that to solve a problem using ASP means to write a logic program whose answer sets correspond to solutions, and then find solutions using an answer set solver (Section 1.4). The basic approach to writing such programs consists in combining choice rules with constraints (Section 3). In this section we discuss a few examples of this "generate-and-test" strategy and its enhancement that involves "defined" atoms.

4.1 Example: N Queens

Our 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 on the same row).
3. X does not contain a pair of different atoms of the form $q(i,j)$, $q(i,j')$ (two queens on 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 rules similar to the program from Problem 3.7:

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

(exactly one queen on 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

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

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

```
number(1..n).
#domain number(I;I1;J;J1).

1{q(K,J) : number(K)}1.

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

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

The expression `number(I;I1;J;J1)` is the LPARSE abbreviation for the list

`number(I), number(I1), number(J), number(J1).`

Problem 4.1 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.

Problem 4.2 (a) Check how long it takes for SMODELS to find one solution to the 16 queens problem using the program above. (b) The constraint corresponding to Condition 3 can be alternatively written using a cardinality expression:

$$\leftarrow 2 \le \{q(i,1), \dots, q(i,n)\} \qquad (1 \le i \le n).$$

Modify the program accordingly and check how this modification affects the computation time for 16 queens.

4.2 Example: Schur Numbers

We will show now how to use ASP to estimate the size of Schur numbers. 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 $\{1,\dots,n\}$ can be partitioned into k sum-free sets.[1]

For specific values of k, $S(k)$ can be computed (or estimated) using a program whose answer sets correspond to the partitions of $\{1,\dots,n\}$ into k

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

(possibly empty) sum-free sets. We will use the atoms $a_i(x)$ ($1 \leq i \leq k$, $1 \leq x \leq n$) to express that x belongs to the i-th set A_i. The choice rules

$$1 \leq \{a_1(x), \dots, a_k(x)\} \leq 1 \qquad (1 \leq x \leq n)$$

describe "potential solutions"—partitions of $\{1, \dots, n\}$ into k sets. The constraints

$$\leftarrow a_i(x), a_i(y), a_i(x+y) \qquad (x, y \geq 1,\ x+y \leq n,\ 1 \leq i \leq k)$$

eliminate the sets A_i that are not sum-free.

Here is how this program can be written in the input language of LPARSE:

```
number(1..n).
#domain number(X;Y).

subset(1..k).

1{a(I,X) : subset(I)}1.

:- a(I,X), a(I,Y), a(I,X+Y), subset(I), X+Y<=n.
```

Problem 4.3 Use this program to find $S(3)$.

The number of answer sets for the ASP program above is greater than the number of partitions of $\{1, \dots, n\}$ into k sum-free sets, because the sets in any such partition can be arranged in a sequence $A_1, \dots, A_k$ in many possible ways. In particular, the subsets in a partition can be always ordered in such a way that

$$\begin{aligned} &1 \in A_1, \\ &2 \in A_1 \cup A_2, \\ &\dots \\ &k \in A_1 \cup \dots \cup A_k. \end{aligned}$$

Consequently, if we add to the program above a constraint exressing these conditions then the property of the program that we are interested in will be preserved—it will have an answer set iff $\{1, \dots, n\}$ can be partitioned into k sum-free sets. Adding this "symmetry breaking" constraint will actually improve the computation time of SMODELS for many values of k and n, because, generally, SMODELS is good at exploiting constraints for restricting search.

Problem 4.4 Investigate the effect of this optimization on the computation time of SMODELS for $k = 4$, $n = 38$.

4.3 Generate-Define-Test

In more complex uses of ASP, constraints in the test part of the program use auxiliary atoms that are 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, that is, a closed path that visits every vertex of the graph 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 u_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

$$\{in(u, v)\}^c \tag{1}$$

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

$$adj(u, v), adj(v, u) \leftarrow in(u, v) \tag{2}$$

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

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

for all $u \in V$. For Condition 2, we introduce the auxiliary atoms $r(u)$ that express the reachability of u from u_0. They are "defined recursively" by the rules

$$\begin{array}{l} r(u_0) \\ r(u) \leftarrow r(v), adj(u, v) \end{array} \tag{4}$$

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

$$\leftarrow not\ r(u) \tag{5}$$

for all $u \in V$.

Answer sets for program (1)–(5) are in a 1–1 correspondence with the Hamiltonian circuits in G. Rules (2) and (4) form the define part of the program, and rules (3) and (5) form the test part.

The following file `hc` is an LPARSE encoding of program (1)–(5):

```
{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. They make the use of the option `-d none` to suppress information on the domain predicates redundant

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(u0;u1;u2;u3;v0;v1;v2;v3).
edge(u0,u1). edge(u1,u2). edge(u2,u3). edge(u3,u0).
edge(v0,v1). edge(v1,v2). edge(v2,v3). edge(v3,v0).
edge(u0,v0). edge(u1,v1). edge(u2,v2). edge(u3,v3).
```

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
```

Syntactically, rules in the generate and test parts of an ASP program are somewhat different from Prolog rules: Prolog has neither choice rules nor constraints. On the other hand, the define part of an ASP program does look similar to a Prolog program, and writing define rules in ASP is based on the same intuition as writing Prolog programs.

For "generate-and-test" programs, such as the examples discussed in Sections 4.1 and 4.2, proving correctness is usually not difficult: it requires no theory of answer sets beyond the basic properties of choice formulas and constraints discussed in Section 3. But if a program contains definitions, as the Hamiltonian circuit example, then more complex mathematical tools are required for a correctness proof.

4.4 Programming Exercises

In each of the following exercises, show how to solve the given computational problem using SMODELS. Test your programs using data of your choice.

Problem 4.5 A graph G is said to be *n-colorable* if there exists a function f from its vertices to numbers $1, \ldots, n$ such that $f(x) \neq f(y)$ for every pair of adjacent vertices x, y. Problem: determine whether G is n-colorable.

Problem 4.6 A *clique* in a graph G is a subset of its vertices in which every two elements are adjacent. Problem: determine whether G has a clique of cardinality n.

Problem 4.7 An *independent set* in a graph G is a subset of its vertices in which every two elements are not adjacent. According to Ramsey's theorem, for any positive integers m, n there exists a positive integer r such that every graph with at least r vertices contains either a clique of cardinality m or an independent set of cardinality n. The smallest r with this property is called the *Ramsey number* $R(m,n)$.[2] For instance, $R(3,3) = 6$. Problem: estimate $R(m,n)$.

Problem 4.8 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 $\{1, \ldots, mn\}$, 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 $\{1, \ldots, mn\}$, and, for every $\{i,j\}$ in B, guests i and j should be assigned different tables. Problem: find such a seating arrangement or determine that this is impossible.

[2] See `http://mathworld.wolfram.com/RamseyNumber.html` .

Problem 4.9 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 $\{1, \ldots, n\}$ 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. Problem: find such an assignment of papers to referees or determine that this is impossible.

Problem 4.10 By G_n we denote the $n \times n$ grid of unit squares (that is, the graph whose vertices are the points (x, y) with coordinates $x, y \in \{0, \ldots, n\}$, with two vertices considered adjacent if the distance between them is 1). A *wire routing problem* is determined by a list

$$\{v_1, v'_1\}, \ldots, \{v_k, v'_k\}$$

of disjoint pairs of vertices of G_n. A *solution* to a wire routing problem is a list of paths $\alpha_1, \ldots, \alpha_k$ in G_n such that

- α_i connects v_i with v'_i ($i = 1, \ldots, k$), and
- no two paths on the list pass through a common vertex.

Problem: find a solution such that the length of each path in it does not exceed a given upper bound u, or determine that this is impossible.

Problem 4.11 Problem: cover an 8×8 chessboard by twenty-one 3×1 tiles and one 1×1 tile.

Problem 4.12 Each of four men owns a different species of exotic pet. Here is what we know about them:

1. Mr Engels (whose pet is named Sparky), Abner and Mr. Foster all belong to a club for owners of unusual pets.
2. The iguana isn't owned by either Chuck or Duane.
3. Neither the jackal nor the king cobra is owned by Mr. Foster.
4. The llama doesn't belong to Duane (whose pet is named Waggles).

5. Abner, who doesn't own the king cobra, isn't Mr. Gunter.
6. Bruce and Mr. Foster are neighbors.
7. Mr. Halevy is afraid of iguanas.

Problem: Find each man's full name and determine what kind of pet he owns.

Problem 4.13 There are five houses of five different colors. In each house lives a person of a different nationality. Each of these five men drinks a certain beverage, smokes a certain brand of cigarettes, and keeps a certain pet. No two men have the same pet, drink the same drink or smoke the same brand. We also know the following:

1. The Brit lives in the red house.
2. The Swede keeps a dog.
3. The Dane drinks tea.
4. The green house is on the left of the white house.
5. The owner of the green house drinks coffee.
6. The person who smokes Pall Mall rears birds.
7. The owner of the yellow house smokes Dunhill.
8. The man living in the house right in the center drinks milk.
9. The Norwegian lives in the first house.
10. The man who smokes Blend lives next to the one who has cats.
11. The man who has horses lives next to the Dunhill smoker.
12. The man who smokes Bluemaster drinks beer.
13. The German smokes Princess.
14. The Norwegian lives next to the blue house.
15. The man who smokes Blend has a neighbor who drinks water.

Problem: determine who owns the fish.