Using DLV

DLV is a system for computing answer sets. In its input program, rules may be represented by schematic expressions with variables, but without function symbols. Like Prolog systems, DLV expects that $\leftarrow$ is represented by `:-` , that each rule is followed by a period, and that every variable begins with a capital letter. Classical negation is represented by `-` and negation as failure by `not` . The system can be invoked by typing

```
dlv <filename>
```

on the command line.

Assignment DLV-1. Download the version of the system that will run on your platform from the DLV homepage (see the *Relevant Internet Sites* section of this seminar's homepage) and call it `dlv`. Use it to solve Problem 3.1 from *Foundations of Logic Programming* and Problems 24 and 26 from the handout on answer sets.

Note that in the case of Problem 26 the output does not show the atoms beginning with e. This is because those atoms are "extensional"—they are defined in the program by rules with empty bodies.

If you try to solve Problem 23a using DLV in a similar way, an error message will be produced. This is because DLV expects the given schematic representation of the input program to be *safe* (or *range-restricted*): every variable occurring in a schematic rule is required to occur also in one of the its body literals not preceded by negation as failure. The schematic rule

$$\neg Prime(x) \leftarrow not \;\; Prime(x) \qquad (1 \le x \le 10) \tag{1}$$

violates this condition.

There are two ways to use DLV to solve Problem 23a. One is to rewrite the program without variables, that is, to replace (1) by 10 rules

$$\begin{array}{l} \neg Prime(1) \leftarrow not \;\; Prime(1) \\ \dots \\ \neg Prime(10) \leftarrow not \;\; Prime(10) \end{array}$$

The other way is to replace (1) by

$$\neg Prime(x) \leftarrow not \;\; Prime(x),\; Number(x)$$

and to define *Number* to hold for $1, \ldots, 10$. This modification changes the program's answer set, but in a very simple way: it adds atoms

$$Number(1), \ldots, Number(10)$$

to it. DLV allows us to give a compact definition of *Number* using "built-in predicates." One of these predicates, `#int`, represents membership in a fixed initial segment of nonnegative integers. The upper bound `#maxint` of this segment is specified in the command line using option `-N`. Thus the input file for Problem 23a may look like this:

```
prime(2).
prime(3).
prime(5).
prime(7).
-prime(X) :- not prime(X), number(X).

number(X) :- #int(X), X!=0.
```

and DLV should be invoked as follows:

```
dlv <filename> -N=10 .
```

In the output of DLV, we will read:

```
{-prime(1), -prime(4), -prime(6), -prime(8), -prime(9),
-prime(10), number(1), number(2), number(3), number(4),
number(5), number(6), number(7), number(8), number(9),
number(10)}
```

The output does not include the atoms beginning with `prime` because they are extensional. Displaying the auxiliary atoms that begin with `number` can be suppressed by including `-filter="prime"` in the command line.

Assignment DLV-2. Use DLV to solve Problem 23b.

DLV is capable of computing answer sets for logic programs of a more general kind that we have seen so far—for "disjunctive" programs. In a disjunctive rule, the head may contain several literals, rather than one. The notion of an answer set is extended to disjunctive programs in Section 5.1 of *Introduction to Logic Programming*.

In particular, the head of a disjunctive rule can be empty:

$$\leftarrow L_1, \ldots, L_m, not \;\; L_{m+1}, \ldots, not \;\; L_n \;. \tag{2}$$

Such disjunctive rules are called *constraints*. A set X of literals *violates* a constraint (2) if

$$L_1, \ldots, L_m \in X, \;\; L_{m+1}, \ldots, L_n \notin X.$$

The effect of adding a constraint to a program is to eliminate the answer sets violating that constraint.

Assignment DLV-3. Instruct DLV to find the answer sets for the program from Problem 24 that (a) include neither p nor $\neg q$, (b) include both p and $\neg q$, and (c) include at least one of p, $\neg q$.