Programming Directions for Assignment 1

CSC486/2506 - Knowledge Representation and Reasoning

Fall 2006

1 Platforms to be used

Your final code *must* run on either the CDF or CSLab machines. If you download software, beware of differences in versions and leave yourself enough time to test your final code on one of these machines.

The preferred programming languages for the course, and specifically for this assignment Are as follows:

1. **Prolog:** your program has to run under SWI=Prolog, which is available for Linux and Windows at http://www.swi-prolog.org/ together with full documentation.

   You can run SWI-Prolog in CDF (execute `pl`) and CSLAB (execute `/pkgs/pl-5.6.17/bin/pl` on a linux machine).

2. **Scheme:** your program has to run under MIT Scheme. You can download a copy from `http://www.gnu.org/software/mit-scheme/` for Linux or Windows.

   MIT Scheme is installed in both CS and CDF. Just type `scheme` to run it.

The reason for using Scheme or Prolog is that they are natural programming languages for the kind of problems we will be dealing with. Programming in another language will require *a lot* more work on your behalf. With this caveat, you can still use **Java** or **C** as long as your program compiles and runs on CDF or CSLab machines. In the documentation, you must specify whether your program runs in CDF or CSLab and clearly state the exact **command used to compile** your source files.

2 Input Format

Question 4 requires you to write a satisfiability procedure for propositional clauses using the described *tableau* method. Here, we will discuss a uniform way of representing the input to your program. You **must** use this representation.
We represent *literals* as follows:

1. **Positive literals** are represented as **positive integers**. E.g., instead of literals p and q, we will refer to literals 2 and 5;

2. **Negative literals** are represented as **negative integers**. E.g., instead of literals $\neg p$ and $\neg q$, we will refer to literals -2 and -5.

We represent a *clause* as follows:

- In Scheme, we will represent a clause as a Scheme list of literals. So a clause will have this format: `(l1 l2 ... ln)` where *li* is the representation of a literal (that is, an integer different from zero). For instance, `(2 -3 4)` stands for the clause $(2 \vee \neg 3 \vee 4)$;

- In Prolog, we will represent a clause as a Prolog list of literals. So a clause will have this format: `[l1,l2,...,ln]` where *li* is the representation of a literal (that is, an integer different from zero). For instance, `[2,-3,4]` stands for the clause $(2 \vee \neg 3 \vee 4)$.

```
(1 -2)
(2 -8)
(-5 -4 3)
(5)
(4 -2 -6)
(-8 -7 6 -10)
(6 -10)
(end)
```

(a) Scheme representation

```
[1,-2].
[2,-8].
[-5,-4,3].
[5].
[4,-2,-6].
[-8,-7,6,-10].
[6,-10].
[end].
```

(b) Prolog representation

Figure 1: Two files containing a set of propositional clauses

Finally, a set of propositional clauses in a file will be represented as a sequence of clauses representations as described. The sequence will finish with a list containing a single element "`end`" meaning there are no more clauses. In particular, in the Prolog representation each clause will be followed by a dot `'.'` symbol stating the end of the clause. For example, Figure 1(a) shows a file containing a set of propositional horn clauses for Scheme and Figure 1(b) shows a file representing the same set for Prolog. Your program will take files of this sort as the input. If you do not use Prolog or Scheme you should chose one of the two representations and state which one you use in your documentation.

3 What your program should provide

If you use Prolog or Scheme we provide sample files (`tabsat.pl` and `tabsat.cl` respectively [1]) which already have the code necessary for reading a set of clauses from a file. These two files define a top-level procedure called `tabSat` whose only argument is the name of a file containing a set of clauses.

Procedure `tabSat` reads the clauses in the file and calls `solve_tab_sat` while passing the corresponding set of clauses as a *list of clauses*. It is **this** procedure `solve_tab_sat` that you have to implement by modifying the sample file. `solve_tab_sat` takes a list of clauses in the corresponding form and prints '`YES`' if such clauses are satisfiable, or '`NO`' otherwise.

Notice that you do not need to program the top-level predicate `tabSat` as it is already implemented. Given a file containing a set of clauses, you should be able to test your Scheme program as follows: `(tabSat "<filename>")`. Similarly, in Prolog the command will look as follows: `tabSat(<filename>).` Notice also that you will have to write extra code to perform your statistical experiments so as to support or refute the easy-hard-easy pattern for the tableau method. Such code will include a procedure for generating random sets of clauses and it will use procedure `solve_tab_sat` to test the satisfiability of those sets.

Finally, if you are using Java or C, your program itself should be named `tabSat` and it should take as its only argument the name of the file where it should read the set of clauses. As there is no sample file provided for these programming languages, you are responsible for programming both the task of reading the clauses from a file as well as solving the satisfiability problem. A call to your program should have the following form: `tabSat <filename>`.

4 How to submit your program

You should submit an *electronic copy* of your code to `jabaier /AT/ cs /DOT/ toronto.edu` as an attachment file. You should submit a *printed copy* of your experimental results and your program with the rest of your paper assignment. Note that code received *after the due date* will count as a late assignment. Recall that your program documentation should include the exact command used for compiling/running your program. It should also indicate which machine (CSLab or CDF) your code runs on.

No matter what platform you choose, remember that your program has to run error-free in CDF or CSLab. So, if you worked elsewhere, we recommend you test your program on either of these machines well before the due date of the assignment.

[1] These are available from the website