CS 161 – FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE

Fall 2016
Assignment 4 – Due 11:55pm Thursday, October 27

In this assignment, you will write a `LISP` program to solve the *satisfiability* (SAT) problem. In particular, given a propositional sentence Δ in *conjunctive normal form* (CNF), you will decide whether Δ is satisfiable. A propositional sentence Δ is in CNF iff it is a conjunction of *clauses*, where a clause is a disjunction of literals (a literal is a variable or its negation). For instance, the following sentence is a CNF with three clauses, which is defined over binary variables X, Y, and Z.

$$\Delta = (X \vee \neg Y \vee Z) \wedge \neg X \wedge (\neg Y \vee \neg Z).$$

A CNF is *satisfiable* if there exists a complete variable assignment that satisfies each clause of the CNF (otherwise, it is unsatisfiable). In this case, the corresponding variable assignment is called a *model* of the CNF. For example, CNF Δ above is satisfiable as the complete variable assignment $\{X = \mathsf{False}, Y = \mathsf{False}, Z = \mathsf{True}\}$ is a model of CNF Δ (note that there is another model of Δ).

Indeed, one can easily formulate a SAT problem as a constraint satisfaction problem (CSP). Basically, variables of the CNF will correspond to the variables of the CSP, each having a domain with two values (i.e., True and False), and each clause of the CNF will represent a constraint of the CSP. Then, a solution to the CSP will correspond to a model of the CNF, and vice versa. In this assignment, your task is to treat the SAT as a CSP and solve it using backtracking search while detecting states that violate constraints. You are encouraged to use some of the techniques discussed in class for improving the performance of backtracking search, including variable ordering and forward checking in particular. To do that, you will represent a CNF in `LISP` as follows:

- A variable is an integer indexed from 1 to n, where n is the number of variables of the CNF. So, a positive (resp., negative) literal can be represented by a positive (resp., negative) integer. For example, positive literal of variable 2 is 2, and negative literal of variable 2 is -2.
- A clause is a list of integers. For example, the list (1 -2 3) represents the clause $1 \vee \neg 2 \vee 3$. Note that a unit clause is also represented as a list, e.g., the list (-2) represents the unit clause -2.
- A CNF is a list of lists. For example, the list ((1 -2 3) (-1) (-2 -3)) represents CNF Δ above, where variables X, Y, and Z are indexed by 1, 2, and 3, respectively.

Given this representation, your top-level function must have the following signature:

```
(defun sat? (n delta) ...)
```

where `n` is an integer and `delta` is a CNF defined over `n` variables. The function `sat?` returns a list of `n` integers, representing a model of `delta`, if `delta` is satisfiable, otherwise it returns `NIL`, e.g.,

```
(sat? 3 '((1 -2 3) (-1) (-2 -3))) returns (-1 -2 3).
(sat? 1 '((1) (-1))) returns NIL.
```

When a CNF has more than one model, `sat?` can return any one of the models, where the order of literals in the model can be arbitrary.

Reading CNF files

A common and easy way to represent CNFs is through DIMACS file format (see below for details). To help you test your program with bigger CNFs, we provide you with `LISP` code that can parse CNF files in DIMACS format (see the file `parse_cnf.lsp`). In particular, given such a CNF file as input, the function `parse-cnf` will return a list of two elements: the number of variables of the CNF and its `LISP` representation, respectively. It should then be trivial to call the `sat?` function. We also provide you with some CNF files in DIMACS format, where the CNFs become harder as the number of variables increases (see the folder `cnfs/` coming with the assignment).

DIMACS format: Consider the following CNF which is over binary variables 1, 2, and 3:

$$(1 \lor \neg 2 \lor 3) \land \neg 1 \land (\neg 2 \lor \neg 3) \land 3.$$

This CNF can be represented using DIMACS format as follows:

```
c this is a comment line
p cnf 3 4
1 -2 3 0
-1 0
-2 -3 0
3 0
```

In general, a CNF file may start with a number of comment lines, where each such line must begin with lowercase `c`. Next, we must have what is known as the "problem line", which begins with lowercase `p`, followed by `cnf`, followed by the number of variables n, followed by the number of clauses m. This is followed by clause lines. A clause line is defined by listing clause literals one by one, where a negative literal is preceded by a `-` sign. The end of a clause is defined by `0`. Note that variables are indexed from 1 to n. There can also be comments in between clause lines.

Grading

Your submission will be evaluated by two measures: correctness and speed. 80% of the grade will be based on correctness and 20% will be based on the rank of your submission's speed with respect to the other correct solutions submitted by your fellow students.

Submission & Rules

(1) Submit your commented `LISP` program in a file named **hw4.lsp** via **CCLE**.

(2) Your programs will be evaluated under `CLISP` interpreter. In order to get any scores, you need to make sure the following `LISP` command does not produce any errors in `CLISP` interpreter.

`> (load "hw4.lsp")`.

(3) The functions you are allowed to use are the same as those allowed in past assignments.

(4) You are allowed to use as many helper functions as you want.

(5) All input to your functions will be legal, i.e., you do not need to validate inputs.