CS 161: Fundamentals of Artificial Intelligence

Spring 2021

Programming Assignment 4 - Due 11:55pm Wednesday, April 28

1 Background

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. The SAT problem can be easily formulated as a *constraint satisfaction problem* (CSP).

Conjunctive Normal Form (CNF): A propositional sentence Δ is in CNF if and only if it is a conjunction of *clauses*, where a clause is a disjunction of literals (a literal is a variable or its negation).

Example 1 *The following sentence Δ is a CNF with three clauses, which is defined over binary variables X, Y, and Z.*

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

Clauses:
(1) $X \vee \neg Y \vee Z$
(2) $\neg X$
(3) $\neg Y \vee \neg Z$
Variables: X, Y, Z
Literals: $X, \neg Y, Z, \neg X, \neg Z$

Satifiability (SAT): A CNF is *satisfiable* if and only if there exists a *complete* variable assignment that satisfies the CNF (otherwise, it is unsatisfiable). In this case, the corresponding variable assignment is called a *model* of the CNF. A CNF is satisfied if and only if each of its clauses is satisfied.

Example 2 Δ *from Example 1 is* $True$ *given the variable assignment*

$$\omega = \{X = False,\ Y = False,\ Z = True\}.$$

Therefore, Δ is satisfiable, and the complete variable assignment ω is a model of Δ (note that there is another model of Δ).

In this homework, given a propositional sentence in CNF, you need to decide whether this sentence is satisfiable or not. We refer to this problem as the SAT problem.

SAT 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.

2 Your task

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 techniques discussed in class to improve the performance of backtracking search, such as variable ordering and forward checking in particular.

Representation of CNF in LISP: You are supposed to represent a CNF in LISP as follows:

- A variable is an integer indexing from 1 to n, where n is the number of variables of the CNF. So, a positive literal can be represented by a positive integer. Respectively, a negative literal can be represented by a negative integer. (e.g., Positive literal of variable 2 is 2, and the 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.

Example 3 *Δ from Example 1 will be represented by the list*

```
((1 -2 3) (-1) (-2 -3))
```

where variables X, Y and Z are indexed by 1, 2, and 3, respectively.

Function signature: 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. If delta is satisfiable, function `sat?` returns a list of n integers, representing a model of delta, otherwise it returns `NIL`.

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

Example 4

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

Explanation:

- *For CNF* $(X \vee \neg Y \vee Z) \wedge \neg X \wedge (\neg Y \vee \neg Z)$*, one of its model is* $\{X = False, Y = False, Z = True\}$*, which can be represented by* `(-1 -2 3)`*.*
- *The order of literals in the model is arbitrary, so* `(-2 -1 3)` *is also a valid result to return.*
- $\{X = False, Y = False, Z = False\}$ *also satisfies this CNF, so* `(-1 -2 -3)` *is a valid result, too.*

Example 5

`(sat?  1 '((1) (-1)))` `returns NIL`

Explanation: $X \wedge \neg X$ *is not satisfiable.*

3 Reading CNF files to test your program

The following files coming with the assignment may help you test your program.

(1) Folder `cnfs/`: To help you test your program with bigger CNFs, We provide you with some CNF files in DIMACS format (see below for details about this format), where the CNFs become harder as the number of variables increases. See the folder `cnfs/` coming with the assignment.

(2) `hw4-skeleton.lsp`: We provide you with the LISP code that can parse the CNF files in DIMACS format in `hw4-skeleton.lsp`. In particular, after

completing your `sat?` implementation, you can call `solve-cnf` to test your program on the CNF files. Usage:

```
(solve-cnf <path-to-cnf-file>)
```

e.g.,`(solve-cnf "./cnfs/f1/sat_f1.cnf")`

DIMACS format: A common and easy way to represent CNFs is through DIMACS file format. Consider the following CNF which is over binary variables 1, 2, and 3:

$$(1 \vee \neg 2 \vee 3) \wedge -1 \wedge (\neg 2 \vee \neg 3) \wedge 3$$

This CNF can be represented using DIMACS format as follows

1. c this is a comment line
2. p cnf 3 4
3. 1 -2 3 0
4. -1 0
5. -2 -3 0
6. 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 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. In this homework, you can use the provided functions in `hw4-skeleton.lsp` to read and parse DIMACS files.

4 Grading

Your submission will be evaluated by two measures: (i) correctness and (ii) speed.

(i) 75% of the grade will be based on correctness.
(ii) The last 25% will be based on whether you can correctly solve the SAT problems in files f1, f2, f3, f4, and f5 within 10 minutes each (5 points for each problem).

5 Submission & Rules

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

- The functions you are allowed to use are the same as those allowed in past assignments.
- You are allowed to use as many helper functions as you want.
- All input to your function will be legal (i.e., you do not need to validate inputs)

6 Honor Code

Remember that you **cannot** use any outside references for this or any assignment. However, you are allowed (and encourage) to experiment with actual SAT solver (which can be found online). Obtaining test problems and testing SAT solver from the Internet are acceptable. It is **not acceptable** to copy solution for any function you have to write. In general, any idea that is not originally yours must be attributed to the appropriate sources. If you have any questions, please contact the TA.