CSCE 420

Programing Assignment #3

due: Thursday, Nov 15, 9:35am (submit on eCampus)

Objective

The objective of this project is to implement DPLL, as described in the textbook, and then use it to solve some problems. You may use C++, Java, or Python.

The input will one or more .cnf files containing clauses. The file format is this:

- each line contains a single clause, like "-A -B C" (from ¬A v ¬B v C, or A ∧ B → C)
- the clause is given by list of literals separated by one or more spaces
- positive literals are propositions (symbols), and negated literals will be prefixed by a '-'
- symbols are sequences of characters (case-sensitive) in the following set: a-z|A-Z|0-9|_
- since clauses only have disjunctions ('v') between symbols, those will be dropped
- lines starting with a '#' are assumed to be comments
- the file can also have blank lines

Here is an example:
prob0.cnf

```
# this is the simple example from class
# it should be satisfiable
-A -B
A B
-B D
-A D
-B -C
B C
-C -A
-C -B
A B C
```

The first non-comment line is equivalent to "¬A v ¬B" and the last line is "A v B v C".
Make the program so it can read multiple .cnf files from the command line. For example, the input might be a single set of clauses, or it could be a KB and a separate file of Facts (some assertions, or the negation of a query).

The program should read-in the files, collect all the clauses and make a list of all the propositional symbols (stripping of the '-' from negative literals), and then call DPLL(). When it finishes, it should report whether the clauses are satisfiable, and return a model if they are. If there is a solution, **print out the model** (truth assignments for all propositions), and it is also helpful to print out just the subset of propositions that are True.

As DPLL runs, you should **print out tracing information**, such as what the model (or partial assignment) looks like on each pass, when the unit-clause or pure-symbol heuristics are used, when choice points are reached, and when back-tracking occurs.

Problems to Solve

1a. Use your program to show there is a way to color the map of Australia in the textbook. (hint: write a script to generate the clauses for the KB)

1b. Assuming the initial solution differs from that shown in the textbook, show that the map can be colored the same way as in the textbook. (hint: add a *minimal* set of facts like WAR as an additional input file to force DPLL to find the intended solution)

1c. Show that the map can be colored such that NT is red and V is blue. (different solution than in the book)

1d. **Re-write the map KB in FOL** (e.g. using quantifiers). Use 'has(s,c)' as a predicate to say a state s has the color c, and 'adjacent(s1,s2)' to say which states are adjacent. Facts (predicate) will include: state(WA), state(NT)... and color(r), color(g), color(b), and adjacent(WA,NT)...

2. Write a script to generate the KB for **4-queens**, and use DPLL to find a solution. Use symbols like Q13. Qij=T means there is a queen in column i in row j. Columns are numbered 1 to 4, left-to-right. Rows are number 1 to 4, top-down.

3. Generalize the script for generating clauses for **N-queens** (with N as a command-line arg). Show that the **3-queens** problem has no solution.

4a. Show that the **6-queens** problem has a solution (notice where the queen is in column 1). (optional: *How many back-tracking steps occur during the search? What are the choice-points?*)

4b. Show that the 6-queens problem has a different solution with a queen in Q13.

4c. Show that there is no solution with a queen in the upper-left corner, Q11.

5. **Re-write the KB for N-queens in FOL** (using quantifiers).
   Use 'Q(c,r)' to represent that there is a queen in column c in row r.
   Write the KB in a general way that applies to any version (N) of the problem.
   Each instance of the problem would different facts for the possible rows and columns.
   For the 4-queens problem, there would be facts like this:
   row(1),row(2),row(3),row(4),col(1),col(2),col(3),col(4)

What to Turn in

- You will submit your code for testing using **eCampus** (https://ecampus.tamu.edu/)
- You should include a Word document with **instructions on how to compile and run** your program.
- Include your **knowledge bases** for the map and queens problems.
- Include the **scripts** you used to generate the knowledge bases.
- Include **transcript** that show your solution traces**.**
- Include a type-written **document giving your FOL-encoding** of the map-coloring and N-queens problems.

Example Transcript

```
> cat prob0.cnf
# this is the simple example from class
# it should be satisfiable
-A -B
A B
-B D
-A D
-B -C
B C
-C -A
-C -B
A B C

> python dpll.py prob0.cnf
props:
A B C D
initial clauses:
0: (-A v -B)
1: (A v B)
2: (-B v D)
3: (-A v D)
4: (-B v -C)
5: (B v C)
6: (-A v -C)
7: (-B v -C)
8: (A v B v C)
-----------
model= {}
pure_symbol on D=True
model= {'D': True}
trying A=T
model= {'A': True, 'D': True}
unit_clause on (-A v -B) implies B=False
model= {'A': True, 'B': False, 'D': True}
```

```
unit_clause on (B v C) implies C=True
model= {'A': True, 'C': True, 'B': False, 'D': True}
backtracking
trying A=F
model= {'A': False, 'D': True}
unit_clause on (A v B) implies B=True
model= {'A': False, 'B': True, 'D': True}
unit_clause on (-B v -C) implies C=False
model= {'A': False, 'C': False, 'B': True, 'D': True}
-----------
nodes searched=8
solution:
A=False
B=True
C=False
D=True
-----------
true props:
B
D
```

> cat temp

```
# add a fact to try to force B to be false
-B
```

> python dpll.py prob0_kb.cnf temp

```
props:
A B C D
initial clauses:
0: (-A v -B)
1: (A v B)
2: (-B v D)
3: (-A v D)
4: (-B v -C)
5: (B v C)
6: (-A v -C)
7: (-B v -C)
8: (A v B v C)
9: (-B)
-----------
model= {}
unit_clause on (-B) implies B=False
model= {'B': False}
unit_clause on (A v B) implies A=True
model= {'A': True, 'B': False}
unit_clause on (-A v D) implies D=True
model= {'A': True, 'B': False, 'D': True}
unit_clause on (B v C) implies C=True
model= {'A': True, 'C': True, 'B': False, 'D': True}
backtracking
no solution found (unsatisfiable)
```