Assignment 5
(I can't get no) Satisfaction

15-414: Bug Catching: Automated Program Verification

Due 23:59pm, Tuesday, April 6, 2021
85 pts

This assignment is due on the above date and it must be submitted electronically on Gradescope. Please carefully read the policies on collaboration and credit on the course web pages at http://www.cs.cmu.edu/~15414/assignments.html.

What To Hand In

You should hand in the following files on Gradescope:

- Submit the file `asst5.zip` to Assignment 5 (Code). You can generate this file by running `make handin`. This will include your solutions `baby-sat.mlw`, and the proof sessions in `baby-sat/`.
- Submit a PDF containing your answers to the written questions to Assignment 5 (Written). You may use the file `asst5-sol.tex` as a template and submit `asst5-sol.pdf`.

  Make sure your session directories and your PDF solution files are up to date before you create the handin file.

Using LaTeX

We prefer the answer to your written questions to be typeset in LaTeX, but as long as you hand in a readable PDF with your solutions it is not a requirement. We package the assignment source `asst5.tex` and a solution template `asst5-sol.tex` in the handout to get you started on this.

1 Propagations and Conflicts (25 pts)

The pigeonhole problem asks us to find a one-to-one mapping between n pigeons and m holes. Obviously, this isn't possible when $n > m$. Consider an encoding of this problem as SAT for n pigeons and $n - 1$ holes, where we have the following CNF clauses and propositional variables p_{ij} which assert that pigeon i is placed in hole j.

- *Pigeon clauses*: For each pigeon $1 \leq i \leq n$, assert that it is placed in some hole.

$$p_{i,1} \vee \ldots \vee p_{i,n-1}$$

- *Hole clauses*: For each hole $1 \leq j < n$ and each pair of pigeons $1 \leq i < k \leq n$, these two pigeons aren't placed in the same hole:

$$\neg p_{i,j} \vee \neg p_{k,j}$$

Task 1 (15 pts). Write down a CNF for the pigeonhole problem for $n = 3$ and apply the DPLL algorithm with clause learning to it. You should write down the steps of your evaluation in the following form:

(1) Decide p

(2) Unit propagate q from clause C_2

(3) Decide $\neg r$

(4) Unit propagate s from clause C_1

(5) Conflicted clause C_1

(6) Backtrack to r

(7) Learn conflict clause $\neg p \vee r$

(8) ...

Task 2 (10 pts). Consider the following CNF formula:

$$\underbrace{(\neg x_1 \vee \neg x_2 \vee \neg x_3)}_{C_1} \wedge \underbrace{(\neg x_1 \vee \neg x_3 \vee \neg x_4)}_{C_2} \wedge \underbrace{(\neg x_1 \vee \neg x_5)}_{C_3} \wedge \underbrace{(x_2 \vee x_4 \vee x_5)}_{C_4} \wedge \underbrace{(x_6 \vee x_7)}_{C_5}$$

Suppose you make the following decisions:

- Decide $x_6 = 0$
- Unit propagate x_7 from clause C_5
- Decide $x_1 = 1$
- Unit propagate $\neg x_5$ from clause C_3
- Decide $x_3 = 1$
- Unit propagate $\neg x_2$ from clause C_1
- Unit propagate $\neg x_4$ from clause C_2
- Conflicted clause C_4

Show a learned clause that you can derive from this conflict by either showing a sequence of resolution steps or drawing the implication graph and a possible separating cut. Please refer to Lecture 12 on ways to generate conflict clauses. [1]

2 Encodings (25 pts)

Task 3 (10 pts). For Boolean variables, x_1, x_2, x_3, and x_4, write a CNF which is satisfied if and only if at least two of the variables are set to *true*.

Colorings. Consider a 2-coloring problem for numbers 1 to 5 such that for every integer solution $a + b = c$ with $1 \leq a < b < c \leq n$ holds that a, b, and c do not have the same color. Note that the possible sums with numbers 1 to 5 under these conditions are:

- 1 + 2 = 3
- 1 + 3 = 4
- 1 + 4 = 5
- 2 + 3 = 5

Task 4 (8 pts). Write a CNF encoding for this problem where each color is represented in unary, i.e., use one Boolean variable per color and number. Explain your reasoning for the different clauses that you added to the formula.

Task 5 (7 pts). Write a CNF encoding for this problem where each color is represented in binary, i.e., use one Boolean variable per number representing its color. Explain your reasoning for the different clauses that you added to the formula.

[1] Available at https://www.cs.cmu.edu/~15414/lectures/12-sat-solving.pdf

3 Baby SAT steps (35 pts)

In this assignment, we will explore simple operations that can be performed over formulas in the conjunctive normal form before we build our first verified SAT solver.

Consider the following types that define a variable (var), literal (lit: which is define as a positive or negative variable), clause, cnf formula, and valuation. Assume that the variables range from 0 to *nvars* and that the cnf formulas have 0 or more variables.

```
type var = int
type lit = { var : var ; polarity : bool }
type clause = list lit
type cnf = { clauses : array clause ; nvars : int }
type valuation = array bool
```

An example of how a CNF is represented using this type is provided Figure 1. Besides, a valuation (also called *interpretation* in some lecture notes) is represented as an array of booleans whose i^{th} component is the value of x_i.

```
{ nvars = 4;
  clauses = [
    [ {var=3; value=false} ];
    [ {var=0; value=true}; {var=2; value=false}; {var=3; value=true} ];
    [ {var=1; value=false}; {var=2; value=true} ] ] }
```

Figure 1: Representation of the formula $\neg x_3 \wedge (x_0 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2)$.

Task 6 (10 pts). Specify and implement a function `eval_clause` that takes a valuation ρ and a clause c as its arguments and returns `true` if c is true for valuation ρ and `false` otherwise.

Task 7 (10 pts). Specify and implement a function `eval_cnf` that takes a valuation ρ and a formula cnf in conjunctive normal form as its arguments and returns `true` if cnf is true for valuation ρ and `false` otherwise.

Pure Literals

Any variable that only appears in either positive or negative literals is called *pure*, and their corresponding variables can always be assigned in a way that satisfies the literal. Thus, they do not constrain the problem in a meaningful way, and can be assigned without making a choice. This is called *pure literal elimination* and is one type of simplification that can be applied to CNF formulas. Consider the following CNF formula:

$$\underbrace{(x_1 \vee x_2)}_{C_0} \wedge \underbrace{(\neg x_1 \vee x_2)}_{C_1} \wedge \underbrace{(x_1 \vee \neg x_2 \vee x_3)}_{C_2} \wedge \underbrace{(\neg x_1 \vee x_2 \vee x_3)}_{C_3}$$

Notice that x_3 appears only as a positive literal in this formula. Hence, we can assign x_3 to *true* and satisfy the literal. This procedure will simplify the above formula into:

$$\begin{aligned} & (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_1 \vee x_3) \\ \leftrightarrow\ & (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee \top) \wedge (\neg x_1 \vee x_1 \vee \top) \\ \leftrightarrow\ & (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \end{aligned}$$

Note that if a formula is satisfiable and if a literal l is pure, then it is always possible to have an interpretation that satisfies the literal, i.e., assigns l to *true* if l is positive or to *false* if l is negative.

Task 8 (15 pts). Specify and implement a function `pure_literal` that takes a formula cnf in conjunctive normal form and a literal l as its arguments and returns `true` if l is a pure literal and `false` otherwise.