Assignment 5
(I can't get no) Satisfaction

15-414: Bug Catching: Automated Program Verification

Due Thursday, March 30, 2023
80 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/s22/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.tex` as a template and submit `asst5.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` with 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 the propositional variable p_{ij} asserts that pigeon i is placed in hole j. There are two conditions that a correct assignment should satisfy.

- *Pigeon clauses*: Each pigeon $1 \leq i \leq n$ is placed in a hole.
- *Hole clauses*: Each hole $1 \leq j < n$ contains at most one pigeon.

Task 1 (5 pts). Write down a CNF for the pigeonhole problem for $n = 3$.

Task 2 (10 pts). Using your pigeonhole CNF for $n = 3$, use the resolution rule to prove that the formula is unsatisfiable. You should state your answers as shown below for the example formula $(p \lor \neg q) \land \neg p \land q$:

(1)	$p \lor \neg q$	(Given)
(2)	$\neg p$	(Given)
(3)	q	(Given)
(4)	$\neg q$	(1, 2)
(5)	$\bot$	(3, 4)

Task 3 (10 pts). Using your pigeonhole CNF for $n = 3$, apply the DPLL algorithm with clause learning to it. You should write down the steps of your evaluation in the following form, as illustrated in the example from Section 7.1 of Lecture 14.

(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 $\neg p \lor r$

(8) ...

2 Pure SAT (55 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 $nvars - 1$ and that the cnf formulas have 0 or more variables.

```
type var = int
type lit = { var : var ; sign : bool }
type clause = list lit
type cnf = {
  clauses : array clause ;
  nvars : int ;
  mutable ghost model : list clause
}
type valuation = array bool
```

An example of how a CNF is represented using this type is provided Figure 1 (note that for brevity we do not specify the ghost model, as it is just a functional copy of the clause array). 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; sign=false} ];
    [ {var=0; sign=true}; {var=2; sign=false}; {var=3; sign=true} ];
    [ {var=1; sign=false}; {var=2; sign=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 4 (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 5 (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. In lecture, we proved such a property:

```
let lemma pure_literal_safe (clauses : list clause) (l : lit) =
  requires { is_pure_literal l clauses }
  requires { 0 <= l.var }
  ensures { clauses_unsat clauses <-> clauses_unsat (remove_lit l clauses) }
```

However, there were two parts of the proof that remained unfinished.

Task 6 (10 pts). Show that it is always possible to extend a valuation so that it satisfies a given literal by implementing `extend_valuation`.

Task 7 (25 pts). While we were able to prove `pure_literal_safe` without induction, we found that it was necessary to introduce the following lemma, which does (in all likelihood) require induction.

$$\forall L, T, M \,.\, \texttt{is\_pure\_literal}(L, T) \wedge M \vDash \texttt{remove\_lit}(L, T) \wedge M(L.\texttt{var}) = L.\texttt{sign} \rightarrow M \vDash T$$

Prove this lemma by completing `remove_l_diff_sat` in `baby-sat.mlw`.