Practice Exam

15-414/614 Bug Catching: Automated Program Verification

Name: ____________________

Andrew ID: ____________________

Instructions

- This exam is closed-book.
- You have XX minutes to complete the exam.
- There are 7 problems on 12 pages.
- Read each problem carefully before attempting to solve it.
- State any assumptions that you make about a question.
- If you aren't sure about an assumption, ask the course staff.

	Max	Score
Dynamic Logic	40	
Resolution	30	
SAT Solvers	30	
SAT Encodings	20	
Arrays and Uninterpreted Functions	25	
Certificates	30	
Temporal Logic	30	
Total:	205	

1 Dynamic Logic (40 points)

This problem explores an alternative application of dynamic logic. Instead of reasoning about imperative programs, we reason about programs for a simple stack machine.

Consider the following set of *programs* α, where we replace the usual assignment with several stack operations.

$$\begin{array}{llll} \text{Programs} & \alpha, \beta & ::= & \mathbf{push}\ k \mid \mathbf{dup} \mid \mathbf{drop} \mid \mathbf{dec} \mid \mathbf{plus} \mid \mathbf{times} \\ & & & \mid \alpha\ ;\ \beta \mid \alpha \cup \beta \mid ?P \mid \alpha^* \\ \text{States} & s, t & ::= & k_1 \cdots k_n \\ \text{Formulas} & P & ::= & \mathbf{top}\ k \mid \mathbf{true} \mid \mathbf{false} \\ & & & \neg P \mid P \wedge Q \mid P \rightarrow Q \mid \forall x.\, P \mid \exists x.\, P \mid P \vee Q \mid [\alpha]P \mid \langle\alpha\rangle P \end{array}$$

States are just *stacks* of integers $k_1 \cdots k_n$ where k_1 is the top of the stack. *Formulas* no longer mention variables (which the language of programs does not have). Instead we have a single new formula $\mathbf{top}\ k$ which holds if the top of the current stack is equal to the number k. The quantifiers here range over integers, as usual, and we imagine we state additional arithmetic properties.

We give the semantic definitions for the new constructs; all the other cases remain the same.

$$\begin{array}{lll} s[\![\mathbf{push}\ k]\!]s' & \text{iff} & s' = k \cdot s \\ s[\![\mathbf{drop}]\!]s' & \text{iff} & s = k \cdot s' \quad \text{for some } k \\ s[\![\mathbf{dup}]\!]s' & \text{iff} & s = k \cdot t \text{ and } s' = k \cdot k \cdot t \quad \text{for some } k \text{ and } t \\ s[\![\mathbf{dec}]\!]s' & \text{iff} & s = k \cdot t \text{ and } s' = (k-1) \cdot t \quad \text{for some } k \text{ and } t \\ s[\![\mathbf{minus}]\!]s' & \text{iff} & s = k_1 \cdot k_2 \cdot t \text{ and } s' = (k_1 - k_2) \cdot t \quad \text{for some } k_1, k_2, \text{ and } t \\ s[\![\mathbf{times}]\!]s' & \text{iff} & s = k_1 \cdot k_2 \cdot t \text{ and } s' = (k_1 \times k_2) \cdot t \quad \text{for some } k_1, k_2, \text{ and } t \\ s \models \mathbf{top}\ k & \text{iff} & s = k \cdot t \quad \text{for some } t \end{array}$$

For example, for any stack s we will have

$$s[\![\mathbf{push}\ k\ ;\ \mathbf{times}]\!](k \times k) \cdot s$$

10 **Task 1** Describe the meaning of the following program as a relation between an empty initial stack and a final stack s' by stating the possible forms of s'.

$$(\cdot)[\![\mathbf{push}\ k\ ;\ \mathbf{push}\ i\ ;\ \mathbf{push}\ j\ ;\ \mathbf{minus}\ ;\ \mathbf{times}]\!]s' \quad \text{iff} \qquad\qquad s' = k \times (j - i) \times k$$

$\boxed{10}$ **Task 2** Describe the meaning of the following program as a relation between an initial stack just containing $n > 0$ and a final stack s', by stating the possible forms of s'.

$n[\![(?\neg(\mathbf{top}\ 1)\ ;\ \mathbf{dup}\ ;\ \mathbf{dec})^*\ ;\ ?\mathbf{top}\ 1\ ;\ \mathbf{times}^*]\!]s'$ iff s′ = n!

$\boxed{20}$ **Task 3** Let f be a mathematical function from an integer to an integer. Write a formula computes $f\ \alpha$ such that for every integer k and stack s we have $k \models \mathsf{computes}\ f\ \alpha$ iff $k[\![\alpha]\!]f(k) \cdot s'$ for some s'. Your formula may mention f applied to an argument.

$\mathsf{computes}\ f\ \alpha =$ $\forall x.\, \mathbf{top}\ x \longrightarrow \langle\alpha\rangle\mathbf{top}\ (f(x))$

Prove the correctness of your definition

Solution: 10em

$$
\begin{array}{lll}
k \models \mathsf{computes}\ f\ \alpha & \text{iff} & k \models \forall x.\, \mathbf{top}\ x \longrightarrow \langle\alpha\rangle\mathbf{top}\ (f(x)) \\
 & \text{iff} & k \models \langle\alpha\rangle\mathbf{top}\ (f(k)) \quad (\text{since } k \models \mathbf{top}\ x \text{ iff } k = x) \\
 & \text{iff} & \text{there is an } s \text{ such that } k[\![\alpha]\!]s \text{ and } s \models \mathbf{top}\ f(k) \\
 & \text{iff} & \text{there is an } s \text{ such that } k[\![\alpha]\!]s \text{ and } s = (f(k) \cdot s') \text{ for some } s' \\
 & \text{iff} & k[\![\alpha]\!]f(k) \cdot s' \text{ for some } s'
\end{array}
$$

2 Resolution (30 points)

15 **Task 1** A *tautology* is a clause that contains an atom p and also its negation $\neg p$. Let $\mathcal{T}$ be a set of propositional clauses. Prove that if we delete all tautologies from $\mathcal{T}$ to obtain $\mathcal{S}$, then $\mathcal{T}$ and $\mathcal{S}$ have the same set of satisfying assignments.

Solution:

1. Let $M \models \mathcal{T}$. This means $M \models C$ for every $C \in \mathcal{T}$ and $\mathcal{T} \supseteq \mathcal{S}$. So $M \models C$ for every $C \in \mathcal{S}$.
2. Let $M \models \mathcal{S}$ and $\mathcal{T} = \mathcal{S} \cup \mathcal{R}$ where $\mathcal{R}$ consists entirely of tautologies. Then $M \models C$ for $C \in \mathcal{R}$ because every M satisfies every tautology. That's because either $M \models p$ or $M \models \neg p$ for the complementary pair or literals in C. Since also $M \models \mathcal{S}$ we have $M \models \mathcal{T}$.

15 **Task 2** We say clause C *subsumes* clause D if $C \subseteq D$ and *strictly subsumes* clause D if $C \subsetneq D$. Let $\mathcal{T}$ be a set of propositional clauses. Let $\mathcal{S}$ be the result of deleting all clauses from $\mathcal{T}$ that are strictly subsumed by other clauses in $\mathcal{T}$. Prove that $\mathcal{T}$ and $\mathcal{S}$ have the same set of satisfying assignments.

Solution: We have $\mathcal{T} = \mathcal{S} \cup \mathcal{R}$ where every clause $D \in \mathcal{R}$ we have $C \in \mathcal{S}$ with $C \supsetneq D$.

1. Assume we have $M \models \mathcal{T}$ so $M \models \mathcal{S} \cup \mathcal{R}$ so $M \models \mathcal{S}$.
2. Assume $M \models \mathcal{S}$ so $M \models C$ for every $C \in \mathcal{S}$. If $C \subsetneq D$ then also $M \models D$ because clauses are interpreted disjunctively. Therefore $M \models \mathcal{R}$ since every clause in $\mathcal{R}$ is subsumed by one in $\mathcal{S}$.

3 SAT Solvers (30 points)

20 **Task 1** DPLL learns new clauses that help it avoid entering conflicts similar to those it has already encountered. Consider the following alternative method, which is easier to implement.

1. Let $(l_1, \ldots, l_n)$ be a partial assignment that results in a conflict.
2. Add the clause $\neg l_1 \vee \neg l_2 \vee \cdots \vee \neg l_n$ to the set of clauses as a learned clause.

(10 points) Is this approach sound, i.e. will adding these clauses potentially change the satisfiability of the original formula? Justify your answer.

Solution: This approach is sound. Each clause that is learned in this way lists a non-satisfying partial assignment. Let P be the formula given to the solver. Because $l_1 \wedge \cdots \wedge l_n$ led to a conflict, $P \wedge l_1 \wedge \cdots \wedge l_n$ is unsatisfiable, or equivalently, $\neg(P \wedge l_1 \wedge \cdots \wedge l_n)$ is valid. Applying DeMorgan's gives us $\neg P \vee \neg l_1 \vee \cdots \vee \neg l_n$, which is equivalent to $P \rightarrow (\neg l_1 \vee \cdots \vee \neg l_n)$.
Thus, if P is satisfiable, then any satisfying assignment will also satisfy $\neg l_1 \vee \cdots \vee \neg l_n$. If P is not satisfiable, i.e. equivalent to $\perp$, then $\perp \wedge (\neg l_1 \vee \cdots \vee \neg l_n)$ is still equivalent to $\perp$, and thus unsatisfiable.

(10 points) Is this approach useful, i.e., will learning these clauses ever prevent the solver from exploring an assignment that it wouldn't have otherwise? If so, provide an example; if not, explain why.

Solution: This approach is not especially useful, because even without clause-learning, DPLL does not explore the same partial assignment more than once, so adding these clauses will not cause the solver to avoid any conflicts that it has not already seen.
The normal approach for learning clauses applies resolution, and produces clauses that lead to unit-propagating an assignment that would not necessarily have been decided next when backtracking. Suppose that backtracking happens on the variable corresponding to l_i. The only assignment that will be "undone" by backtracking would be l_i, and this would result in propagating $\neg l_i$ from the newly-learned clause. This is redundant with DPLL's normal behavior. Similarly, the clause will not become unit again afterwards, because future assignments to a variable x_j in the clause will always satisfy $\neg l_j$.

10 **Task 2** Given a partial interpretation a clause can be either satisfied, conflicting, unit or unresolved. For the partial interpretation $I = \{a, \neg c, d\}$ identify the status of each of the following clauses:

$(a \vee \neg a)$ $\equiv$ Satisfied

$(\neg a \vee b \vee c)$ $\equiv$ Unit

$(b \vee \neg b \vee \neg a)$ $\equiv$ Satisfied

$(\neg d \vee a)$ $\equiv$ Satisfied

$(\neg a \vee b \vee e)$ $\equiv$ Unresolved

4 SAT Encodings (20 points)

20 **Task 1** An isomorphism of undirected graphs G_1 and G_2 is a bijection f between their vertices such that any two vertices $v, v' \in G_1$ are connected by a single edge if and only if $f(v)$ and $f(v')$ are connected by a single edge in G_2. Describe how to encode this as a propositional formula that is satisfiable if and only if G_1 and G_2 are isomorphic.

- Explain how many variables are required, and how the variables are interpreted.
- Likewise, explain which clauses are necessary and what they mean.

Demonstrate your encoding on the graphs shown below. Note that they are not isomorphic, because the second graph does not have a self-edge on the left node.

Solution: Assume that both graphs have the same number n of nodes; if they do not, then output a trivial formula that is equivalent to false. The encoding then has n^2 variables x_{ij}, which should be true whenever $f(v_i) = w_j$, where $v_i \in G_1$ and $w_j \in G_2$. There are three groups of clauses.

1. Every vertex in G_1 is mapped to some vertex in G_2.
$$x_{i1} \vee \cdots \vee x_{in} \quad \text{for} \quad 0 < i \leq n$$
2. Distinct vertices in G_1 aren't mapped to the same one in G_2.
$$\neg x_{i,k} \vee \neg x_{j,k} \quad \text{for} \quad 0 < i, j, k \leq n \quad \text{where} \quad i \neq j$$
3. The mapping preserves connectivity.
$$\neg x_{i,i'} \vee \neg x_{j,j'} \text{ for } 0 < i \leq j \leq n, 0 < i' \neq j' \leq n \text{ where } (v_i, v_j) \in G_1, (w_{i'}, w_{j'}) \notin G_2$$

Note that for the third group, we have $i \leq j$ because the graphs are undirected, so their edge relation is symmetric; adding clauses for symmetric $(v_i, v_j) \in G_1$ is redundant.

In the graphs above, let v_1, w_1 be the nodes on the left of each graph, and v_2, w_2 be the nodes on the right of each graph. This gives:

$$\begin{array}{l}
x_{11} \vee x_{12} \\
x_{21} \vee x_{22} \\
\neg x_{11} \vee \neg x_{21} \\
\neg x_{12} \vee \neg x_{22} \\
\neg x_{11} \vee \neg x_{11} \text{ (simplifies to } \neg x_{11}) \\
\neg x_{12} \vee \neg x_{12} \text{ (simplifies to } \neg x_{12})
\end{array}$$

Note that this is unsatisfiable, because the last two clauses conflict with the first.

5 Arrays and Uninterpreted Functions (25 points)

$\boxed{10}$ **Task 1** Provide a formula in the theory of equality and uninterpreted functions that is valid if an only if the following formula in the theory of arrays is valid:

$$\mathsf{read}\,(\mathsf{write}\,a\,i\,(\mathsf{read}\,b\,j))\,j = x \land \mathsf{read}\,b\,i \neq x \land i = j$$

If you introduce any uninterpreted functions in your solution, explain what they correspond to in the original formula.

Solution: First we remove the read-over-write terms by case-splitting on $i = j$ and $i \neq j$. Because $i = j$ is in the formula, we know that the latter case will be unsatisfiable, so we do not need to include it.

$$\mathsf{read}\,b\,j = x \land \mathsf{read}\,b\,i \neq x \land i = j$$

Now we replace $\mathsf{read}\,b\,\cdot$ terms with uninterpreted function applications $f(\cdot)$:

$$f(j) = x \land f(i) \neq x \land i = j$$

$\boxed{15}$ **Task 2** Compute the congruence closure of your solution for Task 1, and state whether the congruence classes satisfy the equality and uninterpreted functions formula.

Solution: Start with the most granular set of congruence classes on the subterms appearing in the formula (x, i, j, $f(i)$, $f(j)$):

$$\{\{x\}, \{i\}, \{j\}, \{f(i)\}, \{f(j)\}\}$$

Then we account for the equalities listed in the formula:

$$\{\{x, f(j)\}, \{i, j\}, \{f(i)\}\}$$

And propagate congruences; $i = j$ so $f(i) = f(j)$, we must merge $\{x, f(j)\}$ with $\{f(i)\}$:

$$\{\{x, f(i), f(j)\}, \{i, j\}\}$$

There are no further congruences to propagate, so this is the closure. These classes do not satisfy the formula that we started out with, because they contradict $f(i) \neq x$.

6 Certificates (30 points)

Consider the formula:

$$\underbrace{(\neg p_1 \lor \neg p_2)}_{C_1} \land \underbrace{(\neg p_2 \lor p_3)}_{C_2} \land \underbrace{(p_1 \lor \neg p_3 \lor \neg p_5)}_{C_3} \land \underbrace{(\neg p_5 \lor p_2)}_{C_4} \land \underbrace{(p_5 \lor p_2)}_{C_5} \land \underbrace{(p_1 \lor \neg p_3 \lor p_5)}_{C_6}$$

$\boxed{10}$ **Task 1** Which of the following are correct clausal certificates for this formula? Explain your answer in terms of the reverse unit propagation property.

(5 points) $[p_5 \lor p_2, \neg p_5 \lor p_2, \bot]$

> **Solution:** We see that $\neg(p_5 \lor p_2)$ unit-propagates to a conflict via C_4 and C_5. Likewise, conjoining $p_5 \lor p_2$ and asserting $\neg(\neg p_5 \lor p_2)$ leads to the same conflict via unit propagation. However, adding both $p_5 \lor p_2$ and $\neg p_5 \lor p_2$ to C_1-C_6 does not result in any unit propagations, so this is not a valid clausal certificate.
> Generally, a valid clausal certificate will need to have a unit clause in the penultimate position (or redundantly, earlier in the certificate) in order to be valid. Otherwise it will not be possible to unit-propagate to conflict after asserting $\neg\bot$.

(5 points) $[\neg p_1, \neg p_2, \bot]$

> **Solution:** Asserting p_1 unit-propagates $\neg p_2$ from C_1, which leads to a conflict on C_5 after propagating $\neg p_5$ from C_4. Adding $\neg p_1$ and asserting p_2 propagates p_3 from C_2, which then propagates $\neg p_5$ from C_3, and conflicts on C_6. Finally, adding $\neg p_1$ and $\neg p_2$ to C_1 - C_6 will propagate $\neg p_5$ from C_4, which will conflict will C_5.
> Thus, this is a valid clausal certificate.

10 **Task 2** Recall that resolution certificates are composed of a list of proof steps, which are of the form:

$$\begin{array}{lll} \textit{Step}\ \#: & \textbf{Assume} & C \\ \textit{Step}\ \#: & \textbf{Resolve} & C \quad [\textit{Step}\ \#, \textit{Step}\ \#, \ldots] \end{array}$$

The **Resolve** steps give the result C of applying resolution on the sequence of clauses, identified by step numbers, obtained at earlier steps. The last step should be $\perp$.

Explain how to obtain a resolution certificate from a clausal certificate. That is, explain how each step of the clausal proof corresponds to a sequence of resolution steps involving clauses from the original formula, as well as earlier clauses in the certificate.

Solution: Each clause C in the clausal certificate corresponds to a resolution chain, which can be obtained by asserting its negation, noting the clauses $C_1, \ldots, C_n$ that it makes unit (in the order that they become unit), and the clause $C_\perp$ that is ultimately conflicting. Then the resolution chain yielding C is $C_\perp \bowtie C_n \bowtie \cdots \bowtie C_1$. This is represented in the certificate as:

$$\begin{array}{llll} \textit{Step}\ i: & \textbf{Assume} & C_1 & \\ \vdots & & & \\ \textit{Step}\ i+n: & \textbf{Assume} & C_n & \\ \textit{Step}\ i+n+1: & \textbf{Assume} & C_\perp & \\ \textit{Step}\ i+n+2: & \textbf{Resolve} & C & [i+n+1, i+n, \ldots, i] \end{array}$$

Note that the **Assume** steps arise when C_i was a clause in the original formula; if a C_i is a clause appearing earlier in the clausal certificate, then no **Assume** step is needed, and the **Resolve** step can refer to the step number that appeared earlier in the resolution certificate for that clause.

When the final step $\perp$ of the clausal proof is checked, it will yield a similar chain $\perp = C' = C'_\perp \bowtie C'_{n'} \bowtie \cdots \bowtie C'_1$, which will make the certificate a refutation.

10 **Task 3** Provide a resolution certificate corresponding to the clausal certificate $[p, \perp]$ on the following formula:

$$\underbrace{(p \vee q)}_{C_1} \wedge \underbrace{(\neg p \vee q)}_{C_2} \wedge \underbrace{(\neg r \vee \neg q)}_{C_3} \wedge \underbrace{(r \vee \neg q)}_{C_4}$$

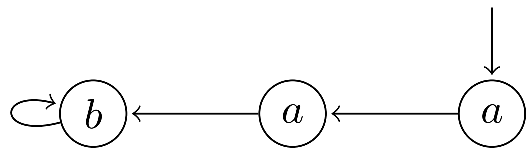
Solution: Begin by noting the resolution chains: asserting $\neg p$ yields $p = C_4 \bowtie C_3 \bowtie C_1$, and then adding p yields $\perp = C_4 \bowtie C_3 \bowtie C_2 \bowtie p$.

$$\begin{array}{llll} \textit{Step}\ 1: & \textbf{Assume} & p \vee q & \\ \textit{Step}\ 2: & \textbf{Assume} & \neg r \vee \neg q & \\ \textit{Step}\ 3: & \textbf{Assume} & r \vee \neg q & \\ \textit{Step}\ 4: & \textbf{Resolve} & p & [3, 2, 1] \\ \textit{Step}\ 5: & \textbf{Assume} & \neg p \vee q & \\ \textit{Step}\ 6: & \textbf{Resolve} & \perp & [3, 2, 5, 4] \end{array}$$

7 Temporal Logic (30 points)

$\boxed{15}$ **Task 1** Draw a Kripke structure that satisfies the formula $\mathbf{A}[a \, \mathbf{U} \, \mathbf{AF} \, b] \wedge \mathbf{EX} \, \neg b$.

Solution:



$\boxed{15}$ **Task 2** For each state in your answer to Task 1, label which of the formulas $\mathbf{AF} \, b$, $\mathbf{EX} \, \neg b$, and $\mathbf{A}[a \, \mathbf{U} \, \mathbf{AF} \, b]$ are satisfied. You may refer to them as P, Q, and R, respectively.

Solution: Let the initial state be w_0, the state to the left of it w_1, and the leftmost state w_2.

$\mathbf{AF} \, b$	w_0, w_1, w_2
$\mathbf{EX} \, \neg b$	w_0
$\mathbf{A}[a \, \mathbf{U} \, \mathbf{AF} \, b]$	w_0, w_1, w_2