Final Exam

15-414/614 Bug Catching: Automated Program Verification

Name: _____

Andrew ID: _____

Instructions

- You have 180 minutes to complete the exam.
- This exam is closed-book and closed-note.
- Read each problem carefully before attempting to solve it.
- State any assumptions that you make about a question.
- **Write your Andrew ID at the top of each page.**
- **If you remove the staple, place the pages in the correct order when you hand the exam in.**

	Max	Score
Dynamic Logic	40	
Resolution	30	
SAT Solving	25	
Propositional Encodings	25	
Congruence	20	
Certificates	25	
Temporal Logic	35	
Total:	200	

1 Dynamic Logic (40 points)

This problem explores an alternative application of dynamic logic. Instead of reasoning about imperative programs over a store, we reason about particular kinds of string-processing programs.

Consider the following set of *programs* α, where we replace the usual assignment $x \leftarrow e$ with the action "**read** c" that reads the character c from the input.

$$\begin{array}{llll} \text{Programs} & \alpha, \beta & ::= & \mathbf{read}\ c \mid \alpha\ ;\ \beta \mid \alpha \cup \beta \mid ?P \mid \alpha^* \\ \text{States} & w & ::= & c_1 \ldots c_n \\ \text{Formulas} & P, Q & ::= & \mathbf{equal}\ w \mid \mathbf{true} \mid \mathbf{false} \\ & & & \neg p \mid P \wedge Q \mid P \rightarrow Q \mid P \vee Q \mid [\alpha]P \mid \langle\alpha\rangle P \end{array}$$

States are just words formed from the characters of some alphabet, where ϵ denotes the empty word. Formulas no longer mention variables (which the language of programs does not have). Instead we have a single new formula **equal** w which holds if the current state is equal to the word w. We give the semantic definitions for the new constructs; all the other cases remain the same.

$$\begin{array}{lll} w[\![\mathbf{read}\ c]\!]w' & \text{iff} & w = c\,w' \\ w \models \mathbf{equal}\ w' & \text{iff} & w = w' \end{array}$$

For a more realistic language we would have other predicates than **equal** to reason about the state of the computation. For the tasks below, we fix the alphabet to be $\{0, 1\}$.

5 **Task 1** Write a program alt such that $w[\![\mathsf{alt}]\!]\epsilon$ iff $w = 010101\ldots01$ where

$$\mathsf{alt} = \quad (\mathbf{read}\ 0\ ;\ \mathsf{read}\ 1)^*$$

5 **Task 2** Write a program dbl such that $w[\![\mathsf{dbl}]\!]\epsilon$ iff in w, a 1 is followed by one or more 1s before the next 0. For example, the relation should hold for $w = 00$ and $w = 111011$ but not for $w = 10$ or $w = 0001$.

$$\mathsf{dbl} = \quad (\mathbf{read}\ 0 \cup (\mathbf{read}\ 1\ ;\ \mathbf{read}\ 1\ ;\ (\mathbf{read}\ 1)^*))^*$$

10 **Task 3** Write a formula "accepts α" such that $w \models \mathsf{accepts}\ \alpha$ iff $w[\![\alpha]\!]\epsilon$

$$\mathsf{accepts}\ \alpha = \quad \langle\alpha\rangle\ \mathbf{equal}\ \epsilon$$

Prove the correctness of your definition

Solution:

$$\begin{array}{lll} w \models \mathsf{accepts}\ \alpha & \text{iff} & w \models \langle\alpha\rangle\ \mathbf{equal}\ \epsilon \\ & \text{iff} & \exists w'.\, w[\![\alpha]\!]w' \wedge w' \models \mathbf{equal}\ \epsilon \\ & \text{iff} & \exists w'.\, w[\![\alpha]\!]w' \wedge w' = \epsilon \\ & \text{iff} & w[\![\alpha]\!]\epsilon \end{array}$$

10 **Task 4** Provide a translation $\mathsf{prog}(r)$ from regular expressions r to programs α such that $w \in \mathcal{L}(r)$ iff $w \models \mathsf{accepts}\ (\mathsf{prog}(r))$. You can find the definition of the language $\mathcal{L}(r)$ generated by regular expression r in the formula sheet.

$$
\begin{array}{lcl}
\mathsf{prog}(c) & = & \mathbf{read}\ c \\
\mathsf{prog}(r_1 \cdot r_2) & = & \mathsf{prog}(r_1)\ ;\ \mathsf{prog}(r_2) \\
\mathsf{prog}(1) & = & ?\mathbf{true} \\
\mathsf{prog}(r_1 + r_2) & = & \mathsf{prog}(r_1) \cup \mathsf{prog}(r_2) \\
\mathsf{prog}(0) & = & ?\mathbf{false} \\
\mathsf{prog}(r^*) & = & \mathsf{prog}(r)^*
\end{array}
$$

You do not need to prove your interpretation correct, except as required in the next task.

10 **Task 5** Prove the correctness of your interpretation of $\mathsf{prog}(1)$. You may use the property you proved in Task 3.

Solution:

$$
\begin{array}{lll}
w \models \mathsf{accepts}\ (\mathsf{prog}(1)) & \text{iff} & w[\![\mathsf{prog}(1)]\!]\epsilon \\
& \text{iff} & w[\![?\mathbf{true}]\!]\epsilon \\
& \text{iff} & w = \epsilon \\
& \text{iff} & w \in \mathcal{L}(1)
\end{array}
$$

2 Resolution (30 points)

We like to think of a clause as a *set of literals*, that is, a clause does not contain any repeated literals. If we instead think of it as a *multiset of literals*, that is, some literals may be repeated, then the resolution rule might be stated in the form

$$\frac{p, C \quad \neg p, D}{C, D}$$

where the comma represents multiset union. This means, for example, if p has $n > 0$ occurrences in C' and $C' = (p, C)$, then p has $n - 1$ occurrences in C. Similarly, if q has i occurrences in C and j occurrences in D, then q has $i + j$ occurrences in C, D.

15 **Task 1** Prove that this form of resolution is still *sound*, that is, the inference preserves the set of satisfying assignments.

Solution: Let $M \models p, C$ and $M \models \neg p, D$. We have to show that $M \models C, D$. We consider two cases:

1. $M(p) = \mathbf{true}$. Then $M \models p$ and $M \not\models \neg p$. Regardless of additional copies of $\neg p$ in D, we therefore have $M \models D$. Hence also $M \models C, D$ since the multiset is interpreted disjunctively.
2. $M(p) = \mathbf{false}$. Then $M \models \neg p$ and $M \not\models p$. Regardless of additional copies of p in C, we therefore have $M \models C$. Henc also $M \models C, D$.

15 **Task 2** Provide a counterexample showing that this form of resolution is *incomplete*. That is, show a set of clauses that is unsatisfiable and prove that one cannot derive the empty clause from it.

Solution: Consider clauses p, p and $\neg p, \neg p$. We calculate:

$$\begin{array}{ll} p, p & (C_1) \\ \neg p, \neg p & (C_2) \\ p, \neg p & C_3 = C_1 \bowtie_p C_2 \end{array}$$

At this point we have $C_1 \bowtie_p C_3 = (p, p) = C_1$ and $C_3 \bowtie_p C_2 = (\neg p, \neg p) = C_2$ and, finally $C_3 \bowtie_p C_3 = (p, \neg p) = C_3$ so our collection of clauses is saturated without deriving an empty clause.

3 SAT Solving (25 points)

10 **Task 1** Given a partial interpretation, a clause can be either satisfied, conflicting, unit or unresolved. Consider the formula:

$$\underbrace{(a \vee b \vee \neg c)}_{C_1} \wedge \underbrace{(\neg a \vee \neg b \vee \neg c)}_{C_2} \wedge \underbrace{(\neg a \vee b \vee c)}_{C_3} \wedge \underbrace{(\neg b \vee c)}_{C_4} \wedge \underbrace{(\neg b \vee \neg c)}_{C_5} \wedge \underbrace{(\neg a \vee \neg c)}_{C_6}$$

Identify a clause with each status for the partial assignments listed below.

Satisfied $M = \{a\}$ C_1

Conflicting $M = \{a, b, \neg c\}$ C_4

Unit $M = \{a\}$ C_6

Unresolved $M = \{\neg b, c\}$ None. All clauses must be decided or unit when two variables are assigned.

Satisfied $M = \{b, c\}$ C_1

15 **Task 2** For the following, consult the formula from the previous task. Which of the following are valid examples of learned clauses that DPLL might generate on its search? Provide a justification for your answer in each case.

(5 pts) $\neg a \vee b \vee c$

Solution: This is not a valid learned clause, because it appears in the CNF. It is not the result of a resolution chain involving other clauses (it contains all the variables), and it is not the negation of another clause.

(5 pts) $\neg b$

Solution: This is a valid learned clause, and would result from deciding c, which would unit-propagate from C_5, leading to conflict with C_4. $\neg b$ is the resolvent of C_4 and C_5.

(5 pts) b

Solution: This is not a valid learned clause, because all assignments that have b lead to conflicts stemming either from C_4 or C_5 (or, redundantly, C_6).

4 Propositional Encodings (25 points)

15 **Task 1** A set with n elements can be partitioned into sets of size 2 if and only if 2 divides n. Suppose that we wish to encode this as an instance of satisfiability with n^2 propositional variables x_{ij}, for $0 < i, j \leq n$. If x_{ij} is true in a satisfying assignment, then elements i and j are in the same partition.

Complete the encoding by providing a set of disjunctive clauses.

- Your clauses should ensure that each element is grouped with at least one other, and that no element is grouped with more than one other.
- Your encoding does not need to be minimal.

Solution: The encoding has n^2 variables x_{ij}, for $0 < i, j \leq n$ and $0 < j \leq 2$. If variable x_{ij} is true, then the ith and jth elements of the set are assigned to the same partition.

We need clauses which assert that each element is grouped with another or itself.

$$x_{i1} \vee \cdots \vee x_{in} \quad \text{for} \quad 0 < i \leq n$$

And clauses which state that each element is grouped with exactly one other, not including itself.

$$\neg x_{ij} \vee \neg x_{ik} \quad \text{for} \quad 0 < i, j, k \leq n \quad \text{for} \quad j \leq k$$

Note that the lax inequality is intentional, so that we end up with clauses $\neg x_{ii} \vee \neg x_{ii}$. Alternatively, we could make this a strict inequality and seperately assert $\neg x_{ii}$, or modify the first set of clauses to exclude x_{ii} as a positive literal in each. Regardless of which solution is chosen, the solution has to ensure that $x_{11} = \mathit{true}, x_{22} = \mathit{true}, \ldots$ with all other variables false is not a satisfying assignment.

10 **Task 2** Demonstrate your encoding for $n = 2$ by providing the necessary clauses.

Solution: Note that the only satisfying assignment is x_{12}, x_{21}.

$$\begin{aligned}
&x_{11} \vee x_{12} \\
&x_{21} \vee x_{22} \\
&\neg x_{11} \vee \neg x_{11} \\
&\neg x_{11} \vee \neg x_{12} \\
&\neg x_{22} \vee \neg x_{22} \\
&\neg x_{21} \vee \neg x_{22}
\end{aligned}$$

5 Congruence (20 points)

The congruence closure algorithm for deciding conjunctive formulas in the theory of equality and uninterpreted functions first constructs a congruence relation, and then checks that it demonstrates the formula's satisfiability by comparing it against any negated equalities.

For each formula and relation (given by congruence classes) below, state whether the relation is the congruence closure of the equalities given in the formula.

- If it is, then state whether the formula is satisfiable. Justify your answer in terms of the congruence closure.
- If it is not, then identify at most one literal to add and at most one to remove for the given relation to be the resulting formula's congruence closure.

For example, given the formula $x = f(y) \wedge f(x) \neq f(y)$, the relation $\{\{x, y\}, \{f(x), f(y)\}\}$ is not the congruence closure of the equality $x = f(y)$. Removing $x = f(y)$ and adding $x = y$ would yield this relation as the congruence closure.

10 **Task 1** $f(x, y) = f(y, x) \wedge x = f(x, y) \wedge f(f(x, y), y) \neq f(y, f(y, x))$
Congruence classes: $\{\{y\}, \{x, f(x, y), f(y, x), f(f(x, y), y), f(y, f(y, x)))\}\}$

Solution: This is the congruence closure of the equalities in the formula. The formula is not satisfiable because $f(f(x, y), y)$ and $f(y, f(y, x))$ are related, but appear in a negated equality literal in the formula.

10 **Task 2** $g(x) = y \wedge f(f(x)) = x \wedge f(f(f(x))) = x \wedge g(f(f(x))) \neq y$
Congruence classes: $\{\{f(f(x)), g(x), y\}, \{f(x)\}, \{g(f(f(x)))\}, \{f(f(f(x))), x\}\}$

Solution: This is not the congruence closure of the equalities in the formula. Removing $f(f(x)) = x$ and adding $f(f(x)) = y$ (resulting in the formula below) would yield this relation as the congruence closure.

$$g(x) = y \wedge f(f(x)) = y \wedge f(f(f(x))) = x \wedge g(f(f(x))) \neq y$$

6 Certificates (25 points)

Suppose that given the following formula:

$$\underbrace{(a \vee \neg b)}_{C_1} \wedge \underbrace{(\neg a \vee c \vee \neg d)}_{C_2} \wedge \underbrace{(a \vee c \vee \neg d)}_{C_3} \wedge \underbrace{(\neg c \vee \neg e)}_{C_4} \wedge \underbrace{(\neg c \vee e)}_{C_5} \wedge \underbrace{(c \vee d)}_{C_6}$$

DPLL produces the following trace:

Step	**Partial valuation**
Start with an empty valuation.	$\{\}$
Decide a.	$\{a \mapsto \texttt{true}\}$
Decide c.	$\{a \mapsto \texttt{true},\ c \mapsto \texttt{true}\}$
Propagate $\neg e$ from unit clause C_4.	$\{a \mapsto \texttt{true},\ c \mapsto \texttt{true}, e \mapsto \texttt{false}\}$
C_5 conflicts. Learn $C_7 = \neg c$. Backtrack.	$\{a \mapsto \texttt{true}\}$
Propagate $\neg c$ from unit clause C_7.	$\{a \mapsto \texttt{true}, c \mapsto \texttt{false}\}$
Propagate d from unit clause C_6.	$\{a \mapsto \texttt{true}, c \mapsto \texttt{false}, d \mapsto \texttt{true}\}$
C_2 conflicts. Learn $C_8 = \neg a$. Backtrack.	$\{\}$
Propagate $\neg a$ from unit clause C_8.	$\{a \mapsto \texttt{false}\}$
Propagate $\neg c$ from unit clause C_7.	$\{a \mapsto \texttt{false}, c \mapsto \texttt{false}\}$
Propagate $\neg d$ from unit clause C_6.	$\{a \mapsto \texttt{false}, c \mapsto \texttt{false}, d \mapsto \texttt{true}\}$
C_3 conflicts.	**Unsat**

10 **Task 1** Provide a clausal certificate for this trace. Recall that a clausal certificate for a formula P is composed of a sequence of clauses $[C_1, C_2, \ldots, C_n = \bot]$ with the property:

$$P \wedge C_1 \wedge \cdots \wedge C_{i-1} \wedge \neg C_i \text{ unit propagates to } \bot, \text{ for all } i \in \{1, \ldots, n\}$$

Briefly explain how you obtained the clauses in this certificate.

Solution: The clausal certificate is $[\neg c, \neg a, \bot]$. This is a list of the clauses learned by DPLL during its search, with $\bot$ corresponding to the fact that its final unit propagation resolved to $\bot$.

15 **Task 2** Provide a resolution certificate for this trace. Recall that resolution certificates are composed of a list of proof steps, which are of the form:

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

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

Briefly describe how you obtained this certificate.

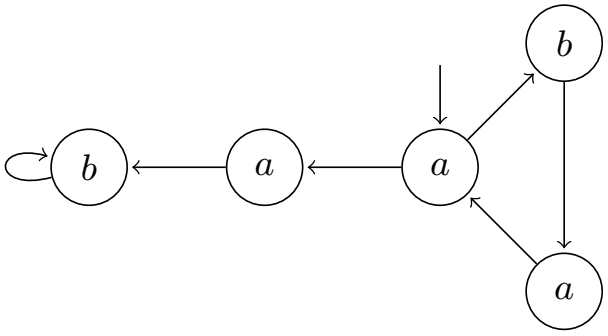
Solution: The following certificate reflects the steps taken by the solver in this trace: first it learns $\neg c$ by resolving C_4 and C_5, then $\neg a$ by resolving C_2, C_6, and $\neg c$, and finally conflicts on C_3.

$$\begin{array}{lllll} \textit{Step 1}: & \textbf{Assume} & C_4 & \\ \textit{Step 2}: & \textbf{Assume} & C_5 & \\ \textit{Step 3}: & \textbf{Resolve} & \neg c & [21] \\ \textit{Step 4}: & \textbf{Assume} & C_6 & \\ \textit{Step 5}: & \textbf{Assume} & C_2 & \\ \textit{Step 6}: & \textbf{Resolve} & \neg a & [543] \\ \textit{Step 7}: & \textbf{Assume} & C_3 & \\ \textit{Step 8}: & \textbf{Resolve} & \bot & [7643] \end{array}$$

The fact that $\neg c$ appears before $\neg a$ in this certificate is not essential; however, refutations that do not contain derivations for these clauses do not correspond to this trace.

7 Temporal Logic (35 points)

20 **Task 1** Consider the computation structure given below. Determine whether it satisfies the following CTL formulas. If it does satisfy the formula provide a brief justification, if it does not satisfy the formula provide a counterexample path to demonstrate the inconsistency.



[10] $\mathbf{AF}\,(a \wedge \mathbf{AX}a)$

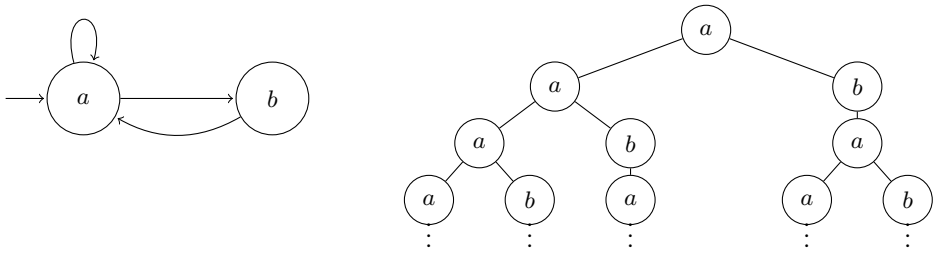
Solution: Does not satisfy. The initial state does not satisfy $a \wedge \mathbf{AX}\,a$ because there is a path to a b successor. The state to the left of the initial state does not saitsfy $a \wedge \mathbf{AX}\,a$ because the next successor is b. So the counterexample path is $a\,a\,b\,b\,\ldots$.

[10] $\mathbf{AG}\,\mathbf{A}[a\,\mathbf{U}\,b]$

Solution: Does satisfy. For every state, either b is true or a is true. Moreover, all paths starting at the initial state must eventually satisfy b, either in the immediate successor or the next after that. So $a\,\mathbf{U}\,b$ must be true.

15 **Task 2** While it may seem natural to think of the linear traces generated by a Kripke structure as the canonical way to understand their corresponding computation, the possible branches that the computation takes are not evident when considering only traces. CTL formulas model the computation embodied in a Kripke structure in terms of trees obtained by "unrolling" each path through the structure.

For example, the tree on the right corresponds to the structure on the left.



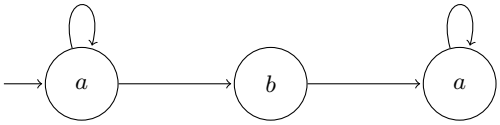
In this problem, you will provide an example that illustrates the distinction between CTL's tree-based modeling of computation, and the linear behavior exhibited by its traces.

Draw a Kripke structure K with the following properties.

- All paths end with an infinite sequence of states satisfying a: for any path $w_0, w_1, \ldots$ in K, there exists an i such that $w_j \models a$ for all $j \geq i$.
- K does not satisfy **AF AG** a.

You may find it helpful to first draw a computation tree that satisfies these properties, but this is not necessary for credit.

Solution:



Clearly, this solution is not unique. Any acceptable structure will need to have a self-transition on a state with a, and from that state there will need to be a transition to a state without a. It is essential that no path can indefinitely transition between a and not a, as this would violate the first bullet of the question.