CDM Quiz

Name:_____ Student ID:_____

1 SAT or UNSAT, that is the question. (20')

1.1 DPLL (15')

Use DPLL algorithm to solve the following problem. You need to write detailed process. When using decide rule, please decide 1 is true at the first time and decide 2 is true at the second time.

$$\emptyset \parallel 1 \vee 3,\ \overline{2} \vee \overline{4},\ \overline{1} \vee 2 \vee \overline{4},\ \overline{1} \vee 3,\ \overline{2} \vee \overline{3} \vee 4,\ 1 \vee 4,\ \overline{1} \vee 2 \vee 4$$

Solution.

$$\begin{aligned}
\emptyset &\parallel 1 \vee 3,\ \overline{2} \vee \overline{4},\ \overline{1} \vee 2 \vee \overline{4},\ \overline{1} \vee 3,\ \overline{2} \vee \overline{3} \vee 4,\ 1 \vee 4,\ \overline{1} \vee 2 \vee 4 \Rightarrow (Decide)\\
1^d &\parallel 1 \vee 3,\ \overline{2} \vee \overline{4},\ \overline{1} \vee 2 \vee \overline{4},\ \overline{1} \vee 3,\ \overline{2} \vee \overline{3} \vee 4,\ 1 \vee 4,\ \overline{1} \vee 2 \vee 4 \Rightarrow (UnitPropagate)\\
1^d 3 &\parallel 1 \vee 3,\ \overline{2} \vee \overline{4},\ \overline{1} \vee 2 \vee \overline{4},\ \overline{1} \vee 3,\ \overline{2} \vee \overline{3} \vee 4,\ 1 \vee 4,\ \overline{1} \vee 2 \vee 4 \Rightarrow (Decide)\\
1^d 3 2^d &\parallel 1 \vee 3,\ \overline{2} \vee \overline{4},\ \overline{1} \vee 2 \vee \overline{4},\ \overline{1} \vee 3,\ \overline{2} \vee \overline{3} \vee 4,\ 1 \vee 4,\ \overline{1} \vee 2 \vee 4 \Rightarrow (UnitPropagate)\\
1^d 3 2^d \overline{4} &\parallel 1 \vee 3,\ \overline{2} \vee \overline{4},\ \overline{1} \vee 2 \vee \overline{4},\ \overline{1} \vee 3,\ \overline{2} \vee \overline{3} \vee 4,\ 1 \vee 4,\ \overline{1} \vee 2 \vee 4 \Rightarrow (BackTrack)\\
1^d 3 \overline{2} &\parallel 1 \vee 3,\ \overline{2} \vee \overline{4},\ \overline{1} \vee 2 \vee \overline{4},\ \overline{1} \vee 3,\ \overline{2} \vee \overline{3} \vee 4,\ 1 \vee 4,\ \overline{1} \vee 2 \vee 4 \Rightarrow (UnitPropagate)\\
1^d 3 \overline{2}\ \overline{4} &\parallel 1 \vee 3,\ \overline{2} \vee \overline{4},\ \overline{1} \vee 2 \vee \overline{4},\ \overline{1} \vee 3,\ \overline{2} \vee \overline{3} \vee 4,\ 1 \vee 4,\ \overline{1} \vee 2 \vee 4 \Rightarrow (BackTrack)\\
\overline{1} &\parallel 1 \vee 3,\ \overline{2} \vee \overline{4},\ \overline{1} \vee 2 \vee \overline{4},\ \overline{1} \vee 3,\ \overline{2} \vee \overline{3} \vee 4,\ 1 \vee 4,\ \overline{1} \vee 2 \vee 4 \Rightarrow (UnitPropagate)\\
\overline{1} 3 &\parallel 1 \vee 3,\ \overline{2} \vee \overline{4},\ \overline{1} \vee 2 \vee \overline{4},\ \overline{1} \vee 3,\ \overline{2} \vee \overline{3} \vee 4,\ 1 \vee 4,\ \overline{1} \vee 2 \vee 4 \Rightarrow (UnitPropagate)\\
\overline{1} 34 &\parallel 1 \vee 3,\ \overline{2} \vee \overline{4},\ \overline{1} \vee 2 \vee \overline{4},\ \overline{1} \vee 3,\ \overline{2} \vee \overline{3} \vee 4,\ 1 \vee 4,\ \overline{1} \vee 2 \vee 4 \Rightarrow (UnitPropagate)\\
\overline{1} 34 \overline{2} &\parallel 1 \vee 3,\ \overline{2} \vee \overline{4},\ \overline{1} \vee 2 \vee \overline{4},\ \overline{1} \vee 3,\ \overline{2} \vee \overline{3} \vee 4,\ 1 \vee 4,\ \overline{1} \vee 2 \vee 4
\end{aligned}$$

1.2 Convert Program to Propositional Logic (5')

We have learnt how to convert a program to a propositional logical formula. But programs with some kind of loops cannot be converted. Please describe the property of this kind of loops and give an example of such loop.

Solution.
We don't know the upper bound of number of these loop iterations. Here is an example.

```
1 int func(int a)
2 {
3   int i;
4   while(i < a)
5     i = i + 2;
6   assert( i > a );
7   return i;
8 }
```

2 Working with SMT solver is like magic. (40')

Consider the following program. We treat f to be an uninterpreted function.

```
1 void func(int a, int b, int c)
2 {
3   c = 2;
4   a = f(c);
5   b = f(2);
6   assert(a != b);
7 }
```

2.1 Convert Program to Predicate Logical Formulas (4')

Here is a predicate logical formula related to the program. We treat f as an uninterpreted function in the formula.

$$(c = 2) \wedge (a = f(c)) \wedge (b = f(2)) \wedge (\neg(a = b))$$

The formula is unsatisfiable if and only if the assertion ( ).

(A) always fails
(B) always succeeds
(C) sometimes fails, sometimes succeeds
(D) I don't know

Solution. A

2.2 DPLL(T) (5')

Now we want to use Lazy SMT techniques to solve the formula. Lazy SMT techniques will firstly convert the formula to a propositional formula in CNF and then give it to DPLL. Please write the propositional formula in CNF and the answer of DPLL.

Solution.

$l1 \wedge l2 \wedge l3 \wedge \neg l4$.
The answer of DPLL is $l1 = T, l2 = T, l3 = T, l4 = F$

2.3 The Nelson-Oppen Method (13')

Now Lazy SMT techniques will use theory solvers to check the answer of DPLL. Please illustrate the details of the Nelson-Oppen method. This involves EUF solver and arithmetic solver.

You need to do purification, allocate formulas to proper solvers, give the shared constants, consider all possible equivalence relations and give the answer of the method. If you find that a formula can be allocated to EUF solver and arithmetic solver at the same time, please allocate it to EUF solver.

You don't need to give the inner process of EUF solver and arithmetic solver.

Solution.

Step1: purification.
$c = 2 \wedge a = f(c) \wedge b = f(d) \wedge (a \neq b) \wedge d = 2$

Step2: allocate formulas to proper solvers.

Arithmetic solver:$c = 2, d = 2$.
EUF solver:$a = f(c), b = f(d), a \neq b$

Step3: give the shared constants. c and d.

Step4: consider all possible equivalence relations

c=d	Arithmetic	EUF
T	T	F
F	F	T

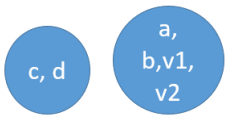
So the answer of the method is unsat.

2.4 EUF Solver (7')

In 2.3, the Nelson-Oppen method uses EUF solver for multiple times. Please draw circles to show the process of EUF solver and give its final answers. You need to explain the reasons of merging circles and its final answers.

Solution. v1=f(c), v2=f(d)
(1) When c=d,
According to the equations,

Because $(c = d) \rightarrow f(c) = f(d)$, then



But in the formula, $a \neq b$. So the formula is unsatisfiable.
(2) When $c \neq d$, According to the equations,

The formula is satisfiable.

2.5 Convert Loops to Formula (6')

Let's consider a more complex program. Please convert the following program to a predicate logical formula such that **the formula is unsatisfiable if and only if the assertion always succeeds**. There should be 2 or 3 steps. You don't need to remove quantifiers.

```
1 void func(int a, int b, int c)
2 {
3   c = 1;
4   for( ; c < 2; ++c)
5     a = a + b - c;
6   assert(a>3)
7 }
```

Solution.

Step1. unroll the loop.

```
1 c = 1;
2 a = a + b - c;
3 c = c + 1;
4 assert(a>3)
```

Step2. Because a, b and c have different values in different states. So we can introduce bound variables.

$$
\begin{aligned}
&(\forall a1)(\forall c1)(\\
&(c = 1) \wedge (a1 = a + b + c) \wedge (c1 = c + 1)\\
&\rightarrow\\
&(a1 > 3))
\end{aligned}
$$

Step3. Because it requires that the formula is unsatifiable if and only if the assertion always succeeds.

$$
\begin{aligned}
&(\exists a1)(\exists c1)(\\
&(c = 1) \wedge (a1 = a + b + c) \wedge (c1 = c + 1)\\
&\wedge\\
&\neg(a1 > 3))
\end{aligned}
$$

2.6 Trigger Matching (5')

When processing the following formula, what is the output of trigger matching algorithm? Please explain why trigger matching produces such output.

$$(\forall x)(f(x) = 3) \wedge (\forall x)(f(x) = 4)$$

Solution. Output is unknown. Because there are no ground terms.

3 Hoare Logic: bridge between program and logic. (10')

3.1 Pre/Post-condition (3')

Write the pre/post-condition of the following Hoare triples to make them valid.

1. $\{\quad\}a := b; \{a > 4\}$
2. $\{c = 5\}while(c < 10)c := c + 2; \{\quad\}$

Solution.

1. $b = 5$
2. $c \geq 10$

3.2 Weakest Precondition (7')

Write $wlp(\quad a := b + 2;\ if(a > 2)\ then\ b := c + 1;\ else\ b := c - 1;,\quad b > 5 \wedge a < 10)$. You should give at least 3 steps.

Solution.

$$
\begin{aligned}
&wlp(\quad a := b + 2;\ if(a > 2)\ then\ b := c + 1;\ else\ b := c - 1;,\quad b > 5 \wedge a < 10)\\
=&wlp(a := b + 2, ((a > 2) \rightarrow wlp(b := c + 1, b > 5 \wedge a < 10)) \wedge (\neg(a > 2) \rightarrow wlp(b := c - 1, b > 5 \wedge a < 10)))\\
=&wlp(a := b + 2, ((a > 2) \rightarrow (c + 1 > 5 \wedge a < 10)) \wedge (\neg(a > 2) \rightarrow (c - 1 > 5 \wedge a < 10)))\\
=&((b + 2 > 2) \rightarrow (c + 1 > 5 \wedge b + 2 < 10)) \wedge (\neg(b + 2 > 2) \rightarrow (c - 1 > 5 \wedge b + 2 < 10))
\end{aligned}
$$

4 Turing Machine (30')

The non-context free language is defined as $L_{wcw} = \{wcw | w \in (0+1)^*\}$ over the input alphabet $\{0, 1, c\}$. For example, $010c010 \in L_{wcw}$,$c \in L_{wcw}$, but $001c100 \notin L_{wcw}$.

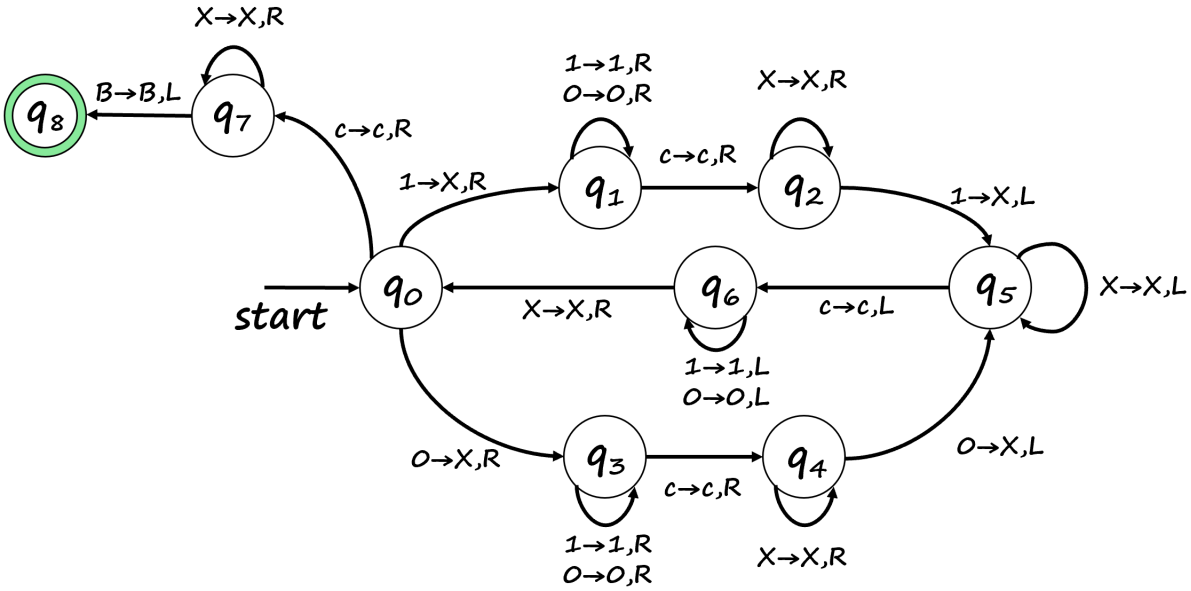
4.1 Turing Machine Basics (15')

Design a Turing machine M_{wcw} such that $L(M_{wcw}) = L_{wcw}$. You can only use the naive TM without using extra states, multitrack or multitape, etc.

1. Specify the states, transitions, and the intuitive purpose of each state.(12')
2. Show how M_{wcw} accept $01c01$ using instaneous descriptions.(3')

Solution

1. The transition diagram:



q_0: Scaning the symbol of first w in wcw, it transfer to q_1 or q_3 depends on scanning 0 or 1, and modify the scanned symbol to X.

q_1: Having scanned a 1 of first w in wcw, try to find a c.

q_2: Having passed the middle c, then try to find a 1 of second w in wcw. If it find a 1, modify it to X and turn back.

q_3, q_4: like the behavior of q_1, q_2, but it handles 0 this time.

q_5: Turing back to find a c.

q_6: Having passed the middle c during turing back, then try to find an X which indicates the remaining part of first w in wcw.

q_7: Having scanned all the symbol of first w in wcw, then we need to check all the symbol of second w in wcw have beed modified to X.

q_8: Accept the input.

2. $q_001c01 \vdash_M Xq_31c01 \vdash_M X1q_3c01 \vdash_M X1cq_401 \vdash_M X1q_5cX1$
$\vdash_M Xq_61cX1 \vdash_M q_6X1cX1 \vdash_M Xq_01cX1 \vdash_M XXq_1cX1 \vdash_M XXcq_2X1$
$\vdash_M XXcXq_21 \vdash_M XXcq_5XX \vdash_M XXq_5cXX \vdash_M Xq_6XcXX$
$\vdash_M XXq_0cXX \vdash_M XXcq_7XX \vdash_M XXcXq_7X \vdash_M XXcXXq_7B$
$\vdash_M XXcXq_8X$

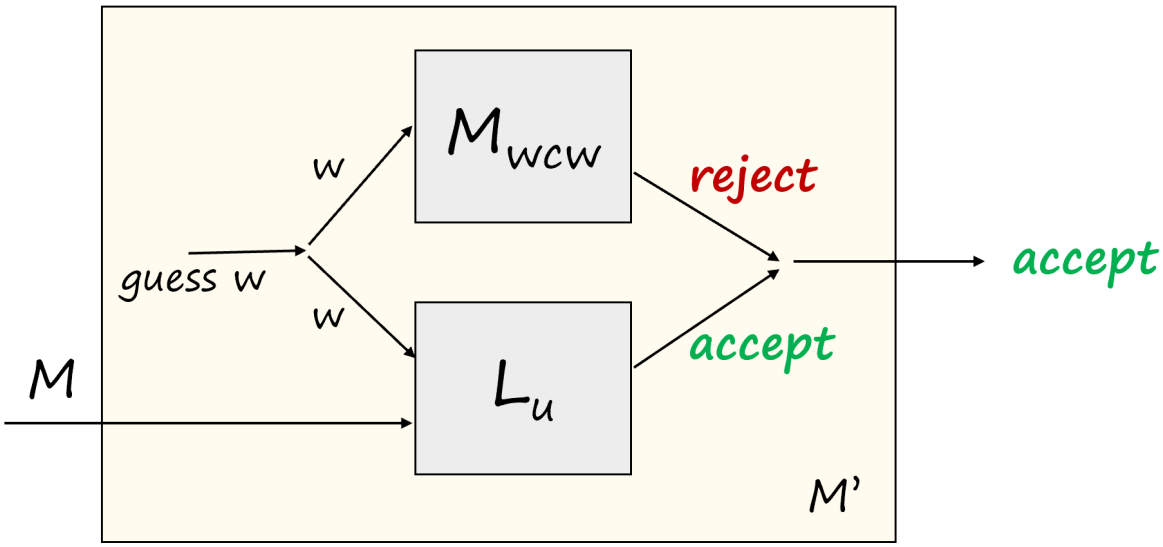
4.2 Undecidablity(15')

Go one step further, we want to know whether a Turing machine can only accept string in L_{wcw}. More formally, given a Turing machine M, we want to know wether $L(M) \subseteq L_{wcw}$. We define the partial non-context free language $L_{pncf} = \{M | L(M) \subseteq L_{wcw}\}$.**(You can not use Rice's Theorem in this problem.)**

1. Show that $\overline{L_{pncf}}$, the complement of L_{pncf}, is recursive enumerable.(8')
2. Show that L_{pncf} is not recursive enumerable.(7')

Solution

1. $\overline{L_{pncf}} = \{M | \neg(L(M) \subseteq L_{wcw})\} = \{M | (\exists w)(w \in L(M) \wedge w \notin L_{wcw})\}$
   We can build a TM M' such that $L(M') = \overline{L_{pncf}}$ as follows:



   M' take a TM M as input, and it ceaselessly guesses a string w. It feeds $\langle M, w\rangle$ to L_u, the universal Turing machine, and also feeds w to M_{wcw} we've construted just now. When we find a w such that M_{wcw} rejects w but L_u accepts $\langle M, w\rangle$, then we accept M; otherwise we don't accept M.

   If $M \in \overline{L_{pncf}}$, then $(\exists w)(w \in L(M) \wedge w \notin L_{wcw})$, so we can finally guess a w such that M_{wcw} rejects w and L_u accepts $\langle M, w\rangle$, then M' accepts M, or $M \in L(M')$. So $M \in \overline{L_{pncf}} \rightarrow M \in L(M') \Leftrightarrow \overline{L_{pncf}} \subseteq L(M')$

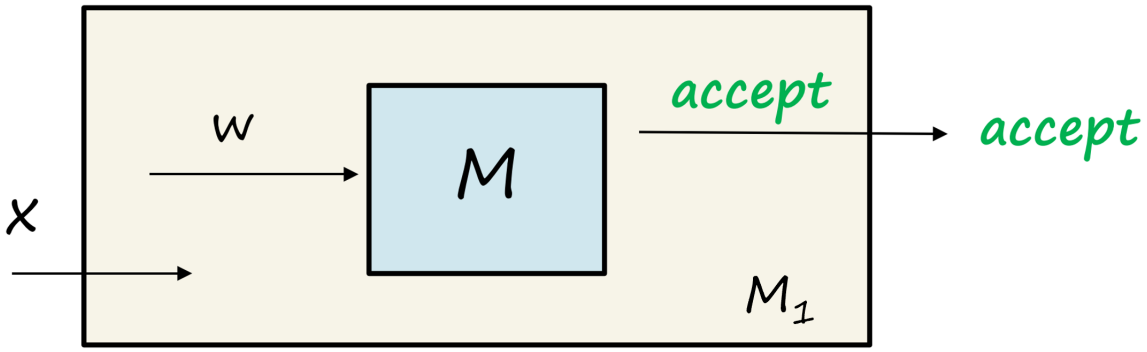
   If $M \in L(M')$, then we've gussed a w such that M_{wcw} rejects w and L_u accepts $\langle M, w\rangle$, which means $(\exists w)(w \in L(M) \wedge w \notin L_{wcw})$, so $M \in \overline{L_{pncf}}$. Then $M \in L(M') \rightarrow M \in \overline{L_{pncf}} \Leftrightarrow L(M') \subseteq \overline{L_{pncf}}$

   Then $\overline{L_{pncf}} \subseteq L(M') \wedge L(M') \subseteq \overline{L_{pncf}} \Rightarrow L(M') = L_{pncf}$

   So $\overline{L_{pncf}}$ is recursive enumerable since M' is a TM for it.

2. Firstly, we prove that $\overline{L_{pncf}} \notin R$ by reducing L_u to it.

   For TM string pair $\langle M, w\rangle$, we can construct a TM M_1 as follows:



M_1 firstly simulates M on w. If M accepts w, then M_1 accept its input; otherwise, M_1 does not accept its input.

If $\langle M, w\rangle \in L_u$, or M accepts w, then M_1 accepts all strings. But L_{wcw} does not contain all strings, so there exists w such that $w \in L(M_1) \wedge w \notin L_{wcw}$, or $M_1 \in \overline{L_{pncf}}$

If $\langle M, w\rangle \notin L_u$, or M doesn't accept w, then M_1 accepts no string, then $L(M_1) = \emptyset \subseteq L_{wcw}$. So $M' \notin \overline{L_{pncf}}$

We've reduced L_u to $\overline{L_{pncf}}$, then we have $\overline{L_{pncf}} \notin R$ since $L_u \notin R$.

Then according to the property of complement that:

$$L \in RE \wedge \overline{L} \in RE \Rightarrow L \in R \wedge \overline{L} \in R$$

We have $\overline{L_{pncf}} \in RE$ but $\overline{L_{pncf}} \notin R$, so $L_{pncf} \notin RE$.

Q.E.D