CSEP590 - Model Checking and Software Verification
Summer 2003
Solution Set 1

1. Translating English into propositional formulas

Solution:
First, we define the atomic propositions:

f - "It is below freezing"
s - "It is snowing"
w - "It is too warm"

Then we have the propositional formulas:
a) $f \wedge s$
b) $f \wedge \neg s$
c) $s \vee f$
d) $(f \vee s) \wedge (f \rightarrow \neg s)$
e) $\neg s \leftrightarrow (f \vee w)$

2. Proving functional completeness

Solution:
a) We must show that propositional formulas involving logical connectives $\vee$, $\rightarrow$, and $\leftrightarrow$ are logically equivalent to propositional formulas involving only $\neg$ and $\wedge$. To this end, we have the following equivalences, where A and B are arbitrary propositional formulas:

i) $A \vee B \Leftrightarrow (\neg A \wedge \neg B)$ ~ by DeMorgan's
ii) $A \rightarrow B \Leftrightarrow (\neg A \vee B) \Leftrightarrow (A \wedge \neg B)$ ~ by i)
iii) $A \leftrightarrow B \Leftrightarrow (\neg A \vee B) \wedge (\neg B \vee A) \Leftrightarrow (A \wedge \neg B) \wedge (B \wedge \neg A)$ ~ by i)

Thus, by i, ii, iii, and the definition of a propositional formula, we conclude that $\{\neg, \wedge\}$ is a functionally complete set of logical connectives.

b) To show that $\{|\}$ is functionally complete, we can show that all propositional formulas involving only | are logically equivalent to a propositional formula involving only $\neg$ and $\wedge$. To this end, we have the following equivalences (which can be verified using truth tables), where A and B are again arbitrary propositional formulas:

i) $\neg A \Leftrightarrow A \mid A$
ii) $A \wedge B \Leftrightarrow \neg \neg (A \wedge B) \Leftrightarrow (A \mid B) \mid (A \mid B)$

Thus, $\{|\}$ is equivalent to $\{\neg, \wedge\}$, so by part a) and transitivity, we conclude that $\{|\}$ is functionally complete.

3. Natural deduction on propositions
Solution:

1	$p \to q$	premise
2	$r \to s$	premise
3	$p \vee r$	assumption
4	p	assumption
5	q	$\to$e 1,4
6	$q \vee s$	$\vee$i 5
7	r	assumption
8	s	$\to$e 2,7
9	$s \vee q$	$\vee$i 8
10	$q \vee s$	commutativity 9
11	$q \vee s$	$\vee$e 3, 4-6, 7-10
12	$(p \vee r) \to (q \vee s)$	$\to$i 3-11

4. Induction
Solution:

Claim: $\forall: n \in \mathbf{Z}^{\geq 0} \quad H_{2n} \geq 1 + n/2$

Proof:

$\forall: n \in \mathbf{Z}^{\geq 0}$, let P(n) represent the statement "$H_{2n} \geq 1 + n/2$"

Base Case: Prove P(0), that is, prove "$H_{20} \geq 1 + 0/2$"

We see that $H_{20} = H_1 = 1$ ~ by definition
We also see that $1 + 0/2 = 1$
Thus, we have $H_{20} = H_1 \geq 1 = 1 + 0/2$
Therefore P(0) is true.

Now let $k \in \mathbf{Z}^{\geq 0}$
Induction Hypothesis: Assume P(k) is true, that is, assume "$H_{2k} \geq 1 + k/2$"

Induction Step: Prove that P(k+1) must also be true, that is,
prove "$H_{2k+1} \geq 1 + (k+1)/2$"

By definition, we have
$H_{2k+1} = 1 + 1/2 + 1/3 + \ldots + 1/2^k + 1/(2^k + 1) + \ldots + 1/(2^{k+1})$
and
$H_{2k} = 1 + 1/2 + 1/3 + \ldots + 1/2^k$

Thus $H_{2k+1} = H_{2k} + 1/(2^k + 1) + \ldots + 1/(2^{k+1})$

But clearly $1/(2^k + r) \leq 1/(2^k + 2^k) = 1/(2^{k+1})$ for $1 \leq r \leq 2^k$
So $1/(2^k + 1) + \ldots + 1/(2^{k+1}) \leq 2^k * 1/(2^k + 2^k) = 1/2$,

and $H_{2k} \geq 1 + k/2$ by hypothesis, so we have:
$H_{2k+1} = H_{2k} + 1/(2^k + 1) + \ldots + 1/(2^{k+1}) \leq 1 + k/2 + 1/2$
$\leq 1 + (k+1)/2$

Thus if P(k) is true, then P(k+1) must also be true.

Induction Conclusion:
$\forall: n \in \mathbf{Z}^{\geq 0}$ P(n) holds, that is, $\forall: n \in \mathbf{Z}^{\geq 0}$ "$H_{2n} \geq 1 + n/2$" is true.

5. k-SAT to 3-SAT reduction, and 2-SAT algorithm
a) k-SAT to 3-SAT reduction
Solution:
To show that k-SAT is reducible to 3-SAT, we must give a polynomial time ("polytime") algorithm for translating an instance of k-SAT into an instance of 3-SAT (a "reduction function"), and argue that this algorithm is correct.

Define the reduction function:
The main idea of the reduction function is that given a CNF clause with $k \neq 3$ literals, we can break down (or build up) the clause into 3-literal clauses by adding literals that contain new variables. The literals are added in such a way so that each literal can satisfy at most one clause. Specifically, our reduction function proceeds in three cases as follows:

Case 1: A clause with one literal (k = 1): x_1
Then we add new variables y_1 and y_2 to create 4 new clauses:

$$x_1 \vee y_1 \vee y_2, \quad x_1 \vee \neg y_1 \vee y_2, \quad x_1 \vee y_1 \vee \neg y_2, \quad x_1 \vee \neg y_1 \vee \neg y_2$$

Case 2: A clause with two literals (k = 2): $x_1 \vee x_2$
Then we add new variable y_1 to create 2 new clauses:

$$x_1 \vee x_2 \vee y_1, \quad x_1 \vee x_2 \vee \neg y_1$$

Case 3: A clause with k > 3 literals: $x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_k$
Then we recursively split the clause, adding new variables y_i as follows:

$$x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_{k-2} \vee y_1, \quad x_{k-1} \vee x_k \vee \neg y_1$$
$$x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_{k-3} \vee y_2, \quad \neg y_2 \vee x_{k-2} \vee y_1, \quad x_{k-1} \vee x_k \vee \neg y_1$$
$$x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_{k-4} \vee y_3, \quad \neg y_3 \vee x_{k-3} \vee y_2, \quad \neg y_2 \vee x_{k-2} \vee y_1, \quad x_{k-1} \vee x_k \vee \neg y_1$$
$$\ldots$$
$$x_1 \vee x_2 \vee y_{k-3}, \quad \ldots \quad \neg y_3 \vee x_{k-3} \vee y_2, \quad \neg y_2 \vee x_{k-2} \vee y_1, \quad x_{k-1} \vee x_k \vee \neg y_1$$

Argue that the reduction function is polytime:
Now, for a single clause C, we can see that cases 1 and 2 can be done in constant time, whereas case 3 takes only linear time in k, the number of literals in C. Thus, if our CNF formula contains m clauses, and s is the size (in literals) of the largest clause, this reduction function will run in time O(sm), so the reduction function is polytime.

Argue that the reduction function is correct:
Call our reduction function F, and our input formula ϕ, then we must show:

$$\phi \text{ is satisfiable } \Leftrightarrow \text{ } F(\phi) \text{ is satisfiable}$$

First, assume ϕ is satisfiable. Then since our reduction function adds new variable(s) at each step in such a way that at most one of the newly formed clauses can be satisfied by the literals containing the new variable(s), we know that there exists a truth assignment τ that satisfies $F(\phi)$ (in fact, τ is a simple extension of the truth assignment that satisfies ϕ). Thus, ϕ is satisfiable $\Rightarrow$ $F(\phi)$ is satisfiable.

Now assume that $F(\phi)$ is satisfiable. Then there exists a truth assignment τ that satisfies every clause in $F(\phi)$. Thus, for a given clause C in ϕ, the corresponding set of clauses in $F(\phi)$ are all satisfiable. But since the latter set of clauses were constructed so that the added literals containing a given variable satisfy at most one clause, we can see that there must be at least one literal from the original clause in ϕ that is satisfied, and hence the original clause must also be satisfiable.
Thus, $F(\phi)$ is satisfiable $\Rightarrow$ ϕ is satisfiable.

Therefore ϕ is satisfiable $\Leftrightarrow$ $F(\phi)$ is satisfiable, and our reduction function is correct.

b) Efficient algorithm for 2-SAT
Solution:
There are many ways to determine if a 2-CNF formula is satisfiable in polytime. One popular way is to construct and process a directed graph representing the formula as follows:

- Convert each 2-CNF clause into an implication
- For this, we can use the equivalence $(a \vee b) \Leftrightarrow (\neg a \rightarrow b) \Leftrightarrow (\neg b \rightarrow a)$
- Now, form one vertex for each literal (e.g. one for a, one for $\neg a$, …)
- The arcs of the graph are given by the two implications equivalent to each clause

This construction clearly takes linear time in the number of clauses.

Now, we search the graph for contradictions, that is, we perform a polytime DFS from each vertex-literal, searching for the negation of that literal. If we can reach some literal's negation from the literal itself, then we have a contradiction, and the 2-SAT formula is unsatisfiable. Otherwise, the formula is satisfiable.