CSC236 Assignment 2

February 28, 2015
c4xieyil, Yilin Xie
c3qiurui, Rui Qiu

1.a

Predicate $P(A)$: Let $T(A)$ be a logically equivalent representation of A (i.e. $T(A) \equiv A$),
and $T(A)$ has negation only on variables.
Let $N(A)$ be logically equivalent to $\neg A$ (i.e. $N(A) \equiv \neg A$), and $N(A)$ has negation only on variables.

We want to prove that $\forall A \in \mathcal{F}, P(A)$ holds using structure induction.

Proof by Structural Induction:

Base Case 1: $A \in \{ X_1, X_2, \dots \}$.
$T(A) = T(X_i) = X_i$ (by def of T), so $T(A) \equiv A$.
$N(A) = N(X_i) = \neg X_i$ (by def of N), so $N(A) \equiv \neg A$ and has negation only on variables.
so $P(A)$ holds for $A \in \{ X_1, X_2, \dots \}$.

Base Case 2: $A \in \{ \neg X_1, \neg X_2, \dots \}$.
$T(A) = T(\neg X_i) = N(X_i) = \neg X_i$
Therefore $T(A) \equiv A$ and only has negation on variable.
$N(A) = N(\neg X_i) = T(X_i) = X_i$
Therefore $N(A) \equiv \neg A$
so $P(A)$ holds when $A \in \{ \neg X_1, \neg X_2, \dots \}$

Inductive Step:
IH: $\forall A_1, B_1 \in \mathcal{F}$, assume $P(A_1)$ and $P(B_1)$ holds.
i.e. $T(A_1) \equiv A_1$ and $T(A_1)$ has negation only on variables.
$N(A_1) \equiv \neg A_1$ and $N(A_1)$ has negation only on variables.
$T(A_2) \equiv A_2$ and $T(A_2)$ has negation only on variables.
$N(A_2) \equiv \neg A_2$ and $N(A_2)$ has negation only on variables.

Case 1: $A = A_1$ or B_1, i.e. $P(A_1)$ or $P(B_1)$.
By IH, this is true obviously.

Case 2: $A = \neg A_1$
$T(A) = T(\neg A_1) = N(A_1) \equiv \neg A_1 = A$, so $T(A) \equiv A$.
$N(A) = N(\neg A_1) = T(A_1) \equiv \neg A_1 = A$, so $N(A) \equiv \neg A$.

By IH, $N(A_1)$ and $T(A_1)$ only has negation on variables,
since $T(A) = N(A_1), N(A) = T(A_1), T(A), N(A)$ also has negation only on variables.

Case 3: $A = A_1 \wedge B_1$
$T(A) = T(A_1 \wedge B_1) = T(A_1) \wedge T(B_1) \equiv A_1 \wedge B_1 = A$
$N(A) = N(A_1 \wedge B_1) = N(A_1) \vee N(B_1) \equiv \neg A_1 \vee \neg B_1 = \neg(A_1 \wedge B_1) = \neg A$
Therefore $T(A) \equiv A$ and $N(A) \equiv \neg A$,
and by IH, $T(A_1), T(B_1), N(A_1), N(B_1)$ has negation only on variables,
$T(A) = T(A_1) \wedge T(B_1), N(A) = N(A_1) \vee N(B_1)$
so $T(A), N(A)$ also has negation only on variables.

Case 4: $A = A_1 \vee B_1$
$T(A) = T(A_1 \vee B_1) = T(A_1) \vee T(B_1) \equiv A_1 \vee B_1 = A$, so $T(A) \equiv A$
$N(A) = N(A_1 \vee B_1) = N(A_1) \wedge N(B_1) \equiv \neg A_1 \wedge \neg B_1 = \neg(A_1 \vee B_1) = \neg A$, so $N(A) \equiv \neg A$

and by IH, $T(A_1), T(B_1), N(A_1), N(B_1)$ has negation only on variables,
$T(A) = T(A_1) \vee T(B_1), N(A) = N(A_1) \wedge N(B_1)$, so $T(A), N(A)$ also has negation on variables.

Case 5: $A = \neg A_1 \vee B_1$ or $A_1 \vee \neg B_1$
$T(A) = T(\neg A_1 \vee B_1) = T(\neg A_1) \vee T(B_1) = N(A_1) \vee T(B_1) \equiv \neg A_1 \vee B_1$,
so $T(A) \equiv A$, and since $N(A_1), T(B_1)$ has negation only on variables (by IH).
$T(A) = N(A_1) \vee T(B_1)$ also has negation only on variables.

$N(A) = N(\neg A_1 \vee B_1) = N(\neg A_1) \wedge N(B_1) = T(A_1) \wedge N(B_1) \equiv A_1 \wedge \neg B_1 = \neg(\neg A_1 \vee B_1)$
so $N(A) \equiv \neg A$, and since $T(A_1), N(B_1)$ has negation only on variables (IH)
$N(A) = T(A_1) \wedge N(B_1)$ also has negation only on variables.
Since A_1, B_1 are arbitrary elements in $\mathcal{F}$, the proof for $A_1 \vee \neg B_1$ is the same as $A = \neg A_1 \vee B_1$, so $P(A)$ also holds when $A = A_1 \vee \neg B_1$

Case 6: $A = \neg A_1 \wedge B_1$ or $A_1 \wedge \neg B_1$
when $A = \neg A_1 \wedge B_1$:
$T(A) = T(\neg A_1 \wedge B_1) = T(\neg A_1) \wedge T(B_1) = N(A_1) \wedge T(B_1) \equiv \neg A_1 \wedge B_1$
so $T(A) \equiv A$, and since $N(A_1), T(B_1)$ has negation only on variables by IH,
$T(A) = N(A_1) \wedge T(B_1)$ should also have negation only on variables.
$N(A) = N(\neg A_1 \wedge B_1) = N(\neg A_1) \vee N(B_1) = T(A_1) \vee N(B_1) \equiv A_1 \vee \neg B_1 = \neg(\neg A_1 \wedge B_1)$
so $N(A) \equiv \neg A$, and since $T(A_1), N(B_1)$ has negation only on variables by IH, $N(A) = T(A_1) \vee N(B_1)$ should also have negation only on variables.

when $A = A_1 \wedge \neg B_1$:
Since A_1, B_1 are arbitrary elements in $\mathcal{F}$, proof for $A_1 \wedge \neg B_1$ is the same as when $A = \neg A_1 \wedge B_1$, so $P(A)$ also holds when $A = A_1 \wedge \neg B_1$.

Case 7: $A = \neg(A_1 \wedge B_1)$
$T(A) = T(\neg(A_1 \wedge B_1)) = N(A_1 \wedge B_1) = N(A_1) \vee N(B_1) \equiv \neg A_1 \vee \neg B_1$
so $T(A) \equiv A$, and since $N(A_1), N(B_1)$ has negation only on variables by IH, $T(A) = N(A_1) \vee N(B_1)$ should also have negation on variables.

$N(A) = N(\neg(A_1 \wedge B_1)) = T(A_1 \wedge B_1) = T(A_1) \wedge T(B_1) \equiv A_1 \wedge B_1$
since $A_1 \wedge B_1 = \neg A$, $N(A) \equiv \neg A$,
and $T(A_1), T(B_1)$ by IH has negation only on variables, $N(A) = T(A_1) \wedge T(B_1)$ should have negation on variables only.

Case 8: $A = \neg(A_1 \vee B_1)$
$T(A) = T(\neg(A_1 \vee B_1)) = N(A_1 \vee B_1) = N(A_1) \wedge N(B_1) \equiv \neg A_1 \wedge \neg B_1$
since $\neg A_1 \wedge \neg B_1 = \neg(A_1 \vee B_1)$, $T(A) \equiv A$.
By IH, $N(A_1), N(B_1)$ only has negation on variables.
$T(A) = N(A_1) \wedge N(B_1)$ should also have negation on variables only.

$N(A) = N(\neg(A_1 \vee B_1)) = T(A_1 \vee B_1) = T(A_1) \vee T(B_1) \equiv A_1 \vee B_1$
since $A_1 \vee B_1 = \neg(\neg(A_1 \vee B_1))$, $N(A) \equiv \neg A$
By IH, $T(A_1), T(B_1)$ only has negation on variables.
$N(A) = T(A_1) \vee T(B_1)$ should also have negation on variable only.

Case 9: $A = \neg A_1 \vee \neg B_1$
$T(A) = T(\neg A_1 \vee \neg B_1) = T(A_1) \vee T(\neg B_1) = N(A_1) \vee N(B_1) \equiv \neg A_1 \vee \neg B_1$
so $T(A) \equiv A$, and by IH $N(A_1), N(B_1)$ has negation only on variables,
$T(A) = N(A_1) \vee N(B_1)$ should have negation on variables too.

$N(A) = N(\neg A_1 \vee \neg B_1) = N(\neg A_1) \wedge N(\neg B_1) = T(A_1) \wedge T(B_1) \equiv A_1 \wedge B_1$
since $A_1 \wedge B_1 = \neg(\neg A_1 \vee \neg B_1)$, $N(A) \equiv \neg A$
and by IH, $T(A_1), T(B_1)$ only has negation on variables,
$N(A) = T(A_1) \wedge T(B_1)$ should also have negation on variables.

Case 10: $A = \neg A_1 \wedge \neg B_1$
$T(A) = T(\neg A_1 \wedge \neg B_1) = T(\neg A_1) \wedge T(\neg B_1) = N(A_1) \wedge N(B_1) \equiv \neg A_1 \wedge \neg B_1$
so $T(A) \equiv A$, and by IH, $N(A_1), N(B_1)$ has negation on variables only,
$T(A) = N(A_1) \wedge N(B_1)$ should also have negation on variables.

$N(A) = N(\neg A_1 \wedge \neg B_1) = N(\neg A_1) \vee N(\neg B_1) = T(A_1) \vee T(B_1) \equiv A_1 \vee B_1$
since $A_1 \vee B_1 = \neg(\neg A_1 \wedge \neg B_1), N(A) \equiv \neg A$
and by IH, $T(A_1), T(B_1)$ has negation only on variables, $N(A) = T(A_1) \vee T(B_1)$ should also have negation on variables only.

To **conclude**: $P(A)$ holds for any $A \in \mathcal{F}$. Therefore, $T(A)$ is logically equivalent to A and has negation only on variables.

1.b

Solution:

```
#PRE: A represents a boolean formula.
#POST: Returns a logically equivalent representation of A that has negation only on variables.

def T(A):
    if type(A) is int:
        return A
    elif type(A) is tuple and len(A) == 2 and A[0] == "not":
        if A[1] is int:
            return ('not', A[1])
        elif len(A[1]) == 2 and A[1][0] == "not":
            return T(A[1][1])
        elif len(A[1]) == 3:
            if A[1][1] == "and":
                return (T(('not', A[1][0])), 'or', T(('not', A[1][2])))
            else:
                return (T(('not', A[1][0])), 'and', T(('not', A[1][2])))
    elif type(A) is tuple and len(A) == 3:
        if A[1] == 'and':
            return (T(A[0]), 'and', T(A[2]))
        else:
            return (T(A[0]), 'or', T(A[2]))
```

1.c

Solution:

Let $S(A)$ be the size of input A, which is counted by how many `and`, `or`, `not` and boolean variables has appeared in A, each of them is counted as 1.

Let $P(k)$ be the **predicate**:
$P(k)$: if A is a boolean formula in $\mathcal{F}$ with $size(A) = k$, then $T(A)$ returns a logically equivalent representation and has negation only on variables.

We want to prove that $P(k)$ holds for all integers $k \geqslant 1$.

Proof by complete induction:

Base Case: when $k = 1$, then $size(A) = 1$, which only contains one variable. From our codes in 1(b), $T(A)$ returns A, which is logically equivalent to A, and has negation only on variable.

Inductive Step:
Let i be an arbitrary integer such that $i \geqslant 1$
Assume $P(i)$ holds for all i where $1 \leqslant i < k$, we must prove that $P(k)$ holds as well.

Case 1: $len(A)$ is 2 and $A[0]$ is “not”.
Let $A =$ (“not”, B) where B is a boolean formula.

Subcase 1: $B \in \{X_1, X_2 \dots\}$
Then $A =$ (“not”, B) = (“not” X_i)
According to code in 1(b), $T(A)$ also returns (“not”, X_i)
Therefore $T(A) \equiv A$ and has negation only on variable.

Subcase 2: $len(B)$ is 2 and $B[0]]$ is “not”.
Let $B =$ (“not” , C) where C is a boolean formula.
According to code in 1(b), $T(A)$ returns $T(C)$.
Since $size(C) < size(A)$, by IH, we have $T(C) \equiv C$ and has negation only on variables so $T(A) \equiv C$.
and $A =$ (“not”, (“not”, C)) = C
Therefore $T(A) \equiv A$ and has negation only on variables.

Subcase 3: $len(B)$ is 3 and $B[1]$ is “and”.
Let $B = (C$, “and” , $D)$ where C, D are boolean formula.
According to code in 1(b), $T(A)$ returns (T((“not” , C)), “or”, T((“not”, B))
since $size$(“not” , $C) < size(A)$
By IH, T((“not”, C)) $\equiv$ (“not”, C) and has negation only on variables.
since $size$(“not”, $D) < size(A)$
By IH, T((“not”, D)) $\equiv$ (“not”, D) and has negation only on variables.
Therefore $T(A) =$ ((“not” , C), “or”, (“not” , D))
and $A =$ (“not”, (C, “and”, D)) = ((“not”C), “or”, (“not” D)).
Therefore $T(A) \equiv A$ has negation only on variables.

Subcase 4: $len(B)$ is 3 and $B[1]$ is “or”.
Let $B = (C$, “or”, $D)$ where C, D are boolean formula.
According to code in 1(b), $T(A)$ returns (T((“not” C)), “and”, T((“not” D)))
since $size$(“not”, $C) < size(A)$
By IH, T((“not”, C)) $\equiv$ (“not”, C) and has negation only on variables.
since $size$(“not”, $D) < size(A)$
By IH, T((“not”, D)) $\equiv$ (“not”, D) and has negation only on variables.

Case 2: $len(A)$ is 3 .
Subcase 1: $A[1]$ is “or”.

Let $A = (B,$ “or”, $C)$, where B and C are boolean formula, $size(B), size(C) < size(A) = k$
By IH, $T(B) \equiv B$ and has negation only on variables.
$T(C) \equiv C$ and has negation only on variables.

According to code in 1(b), $T(A)$ returns $(T(B,$ “or”, $T(C))$
since $T(B) \equiv B$ and $T(C) \equiv C$ and has negation only on variable.
$T(A) = (B,$ “or”, $C)$
$T(A) \equiv A$ and has negation only on variables.

Subcase 2: $A[1]$ is “and”.
Let $A = (B,$ “and”, $C)$, where B and C are boolean formula since $size(B), size(C) < size(A) = k$
By IH, $T(B) \equiv B$ and has negation only on variables.
$T(C) \equiv C$ and has negation only on variables.

According to code in 1(b), $T(A)$ returns $(T(B),$ “and”, $T(C))$
since $T(B) \equiv B$ and $T(C) \equiv C$ and has negation only on variables.
so $T(A) = (B,$ “and”, $C)$.
Therefore $T(A) \equiv A$ and has negation only on variables.

To **conclude**, if A represents a boolean formula then $T(A)$ returns a logically equivalent representation that has negation only on variables.

2.a

Proof: When proving the correctness of a program, it is equal to prove whenever the precondition holds before the execution of program, then the program terminates and when it does, the postcondition holds. Or we say **total correctness = partial correctness + termination**.

To prove the **partial correctness**, which is the following:

> Suppose $m, n > 0$ and $m, n \in \mathbb{N}$. If $g(m, n)$ terminates then when it does, it returns tuple (k, l, b) where $k, l, b \in \mathbb{N}$.

We need to use the following loop invariant:

$I(i)$: After i^{th} iteration, denote the a, b, k, l to be a_i, b_i, k_i, l_i. And if $k_i = 1 + p, l_i = 1 + q, k_i + l_i - 2 = p + q = i$, where p is the number of iterations going through if-loop $a < b$ branch, q is the number of iterations going through if-loop $else$ branch, then $a_i = (1 + p)m, b_i = (1 + q)n$.

Proof of $I(i)$ by simple induction:

Base Case: $I(0) : i = 0 = p + q \implies p = 0, q = 0$, then $k_0 = 1, l_0 = 1$, so $a_0 = k_0 m, b_0 = l_0 n$.

Inductive Step:
IH: Suppose I(i) holds for any natural number t.
We are going to prove it holds for $t + 1$, i.e. $I(t + 1)$.
Since $I(t)$ holds, then $t = p + q$, so $t + 1 = (p + 1) + q$ or $p + (1 + q)$

So we can divide this into two cases, which are the branches of the if-loop.

Case 1: $k_{t+1} = 2 + p, l_{t+1} = 1 + q$, then $a_{t+1} = (2 + p)m = k_{t+1}m, b_{t+1} = (1 + q)n = l_{t+1}n$, therefore $I(t + 1)$ holds.

Case 2: $k_{t+1} = 1 + p, l_{t+1} = 2 + q$, then $a_{t+1} = (1 + p)m = k_{t+1}m, b_{t+1} = (2 + q)n = l_{t+1}n$, therefore $I(t + 1)$ holds as well.

If PRE is true, and suppose the loop terminates after i^{th} iteration, and we proved $I(i)$ holds for any natural number i, then:

$$k_i m = a_i = b_i = l_i n$$

This is exactly what the POST asks for. Hence, the partial correctness is proved.

To prove the **termination**, which is the following:

> Suppose $m, n > 0$ and $m, n \in \mathbb{N}$, the program will terminate.

It is equivalent to say the while-loop inside the code will terminate, because the other part of the code will terminate obviously.

We construct a variable $lcm(m, n)$ which is the **least common multiple of** m, n in maths, then for value $lcm(m, n) - a_i$, we find it is decreasing with each iteration of the while loop.

Specifically, $lcm(m, n) - a_i = lcm(m, n) - k_i m = \frac{mn}{gcd(m,n)} - k_i m = m(\frac{n}{gcd(m,n)} - k_i)$ where $gcd(m, n)$ is the **greatest common divisor of** m, n.

Note that $m, n, gcd(m, n)$ are fixed value in the equation above, and k_i is monotonically increasing as the loop iterates. Hence the value of that expression is decreasing all the time.

Hence the loop WILL terminate at some point.

So far, we can combine the **partial correctness** and **termination**, then $g(m, n)$ is correct.

■

2.b

Proof: The proof of **termination** part is identical to the one in part (a), because it is only affected by PRE.

The proof of partial correctness is where two two differ.

> Suppose $m, n > 0, m, n \in \mathbb{N}$. If $g(m, n)$ terminates and when it does, it returns a tuple of natural numbers (k, l, b) such that $k * m = l * n = b$, and if c is a common multiple of m and n then $b \leqslant c$.

Suppose PRE and $I(i)$ are true, and the loop terminates after i iterations. Then $a_i = b_i = k_i m = l_i n$. Now suppose again c is another multiple of m and n with $c < b_i$, then for some $j < i$, $a_j = b_j$, i.e., the loop terminates at j^{th} iteration which is before i^{th} iteration. This is a contradiction.

Hence $c \geqslant b_i$. Therefore the partial correctness is proved.

Therefore, $g(m, n)$ is correct.

■

2.c

Proof:

To prove **partial correctness**:

> Suppose m, n are positive natural numbers, if the loop in code terminates then b is a multiple of both m and n, and if natural number c is a positive multiple of both m and n then $b \leqslant c$.

Suppose the loop terminates after i iterations. Denote the variables a, b after i iterations to be a_i, b_i. We also create two helper variables p_i and q_i to denote the numbers of iterations going through two different if-branches, and generally we have the relation $p_i + q_i = i$.

Loop invariant $I(i)$: After i^{th} iteration, $p_i + q_i = i$, then $a_i = (1 + p_i)m, b_i = (1 + q_i)n$.

Base Case: $I(0)$ holds, then $p_0 + q_0 = 0 \implies p_0 = q_0 = 0, a_0 = 1m = m, b_0 = 1n = n$.

Inductive Step:
IH: Suppose $I(i)$ holds for any natural number t.
Need to show $I(t + 1)$ holds as well.
By $I(t), p_t + q_t = t, a_t = (1 + p_t)m, b_t = (1 + q_t)n$.
Then for $I(t + 1)$:

Case 1: when the extra iteration adds 1 to p_t, i.e. $p_{t+1} = p_t + 1$, but at the same time, $q_{t+1} = q_t$. Then
$p_{t+1} + q_{t+1} = p_t + 1 + q_t = t + 1$,
$a_{t+1} = (2 + p_t)m = (1 + p_{t+1})m, b_{t+1} = (1 + q_t)n = (1 + q_{t+1})n$, so $I(t + 1)$ holds.

Case 2: when the extra iteration adds 1 to q_t, i.e. $q_{t+1} = q_t + 1$, but at the same time, $p_{t+1} = p_t$. Then
$p_{t+1} + q_{t+1} = p_t + q_t + 1 = t + 1$,
$a_{t+1} = (1 + p_t)m = (1 + p_{t+1})m, b_{t+1} = (2 + q_t)n = (1 + q_{t+1})n$, so $I(t + 1)$ holds.

So $I(i)$ is proved for any natural i.

If PRE is true and $I(i)$ holds for any natural number i. When the loop terminates, then $a_i = b_i = (1 + p_i)m = (1 + q_i)n$, therefore the returned number b_i is a multiple of both m and n. And we can set up a contradiction to prove such b is the smallest multiple for both numbers.

Suppose there exists another multiple of both m and n, say it is $c < b$, then the while loop terminates when both m and n hit the value c with iteration number $j < i$. This is impossible because we know that the loop terminates after i iterations and if it is already terminated at j^{th} iteration, it cannot run again and terminate again at i^{th} iteration. So we have a contradiction. Hence such b is smaller than any other multiple of m and n.

Therefore the **partial correctness** is proved.

To prove **termination**:

> Suppose $m, n > 0$ are two positive natural numbers, the program will terminate.

It is equivalent to say the while-loop inside the code will terminate, because the other part of the code will terminate obviously.

We construct a variable $lcm(m, n)$ which is the **least common multiple of** m, n in maths, then for value $lcm(m, n) - a_i$, we find it is decreasing with each iteration of the while loop.

Specifically, $lcm(m, n) - a_i = lcm(m, n) - (1 + p)m = \frac{mn}{gcd(m,n)} - (1 + p)m = m(\frac{n}{gcd(m,n)} - (p + 1))$ where $gcd(m, n)$ is the **greatest common divisor of** m, n.

Note that $m, n, gcd(m, n)$ are fixed value in the equation above, and p is monotonically increasing as the loop iterates. Hence the value of that expression is decreasing all the time.

Hence the loop WILL terminate at some point.

So far, we can combine the **partial correctness** and **termination**, then $h(m, n)$ is correct.

■