Assignment 2 question 1 sample solution

Let NNF (abbreviation for "Negation Normal Form") be the set of boolean formulas that have negations only applied to variables. $A \equiv B$ means A is logically equivalent to B.

Lemma: $\forall C_1, C_2, C_1', C_2' \in \mathcal{F}.\ C_1 \equiv C_1'$ and $C_2 \equiv C_2'$ implies

- $C_1 \wedge C_2 \equiv C_1' \wedge C_2'$
- $C_1 \vee C_2 \equiv C_1' \vee C_2'$
- $\neg C_1 \equiv \neg C_1'$

1.(a)

First we prove $\forall A \in \mathcal{F}.\ T(A), N(A) \in$ NNF. This is by structural induction on the recursive definition of $\mathcal{F}$.

1. Case $A = X_i$ for some $i \in \mathbb{N}$. $T(A)$ has no negation symbols, and $N(A)$ is a negation symbol applied to a variable, so both are in NNF
2. Case $A = A_1 \wedge A_2$, or $A = A_1 \vee A_2$, or $A = \neg A_1$. Note that for these formulas, neither N nor T add negation symbols to the results of their recursive calls. Since the IH gives us that $T(A_1)$, $T(A_2)$, $N(A_1)$, $N(A_2)$ are all in NNF, it follows that the right-hand sides of the equations defining T and F for non-variable inputs are all in NNF

Now we prove $\forall A \in \mathcal{F}.\ T(A) \equiv A$ and $F(A) \equiv \neg A$ by structural induction on the definition of $\mathcal{F}$.

1. Case $A = X_i$ for some $i \in \mathbb{N}$. $T(X_i) = X_i$, so $T(X_i) \equiv X_i$. $N(X_i) = \neg X_i$, so $N(X_i) \equiv \neg X_i$.
2. Case $A = A_1 * A_2$, where $*$ is $\wedge$ or $\vee$.
   - By the IH, $T(A_1) \equiv A_1$ and $T(A_2) \equiv A_2$. By the Lemma, $T(A_1) * T(A_2) \equiv A_1 * A_2$. Also by the IH, $N(A_1) \equiv \neg A_1$ and $N(A_2) \equiv \neg A_2$. Using those facts:
     - $N(A_1 \wedge A_2) = N(A_1) \vee N(A_2)$, and the Lemma implies $N(A_1) \vee N(A_2) \equiv \neg A_1 \vee \neg A_2 \equiv \neg(A_1 \wedge A_2)$. Similarly:
     - $N(A_1 \vee A_2) = N(A_1) \wedge N(A_2)$, and the Lemma implies $N(A_1) \wedge N(A_2) \equiv \neg A_1 \wedge \neg A_2 \equiv \neg(A_1 \vee A_2)$.
3. Case $A = \neg B$.
   $T(\neg B) = N(B)$ by definition, and $N(B) \equiv \neg B$ by the IH for B, so $T(\neg B) \equiv \neg B$.
   $N(\neg B) = T(B)$ by definition, and $T(B) \equiv B$ by the IH for B, so $N(\neg B) \equiv B$. Since $B \equiv \neg(\neg B)$, by transitivity of $\equiv$, have $N(\neg B) \equiv \neg(\neg B)$, which is what we needed to show.

1.(b)

```
def nnf(C):
  if type(C) == int:        # C is a variable
    return C
  elif len(C) == 2:         # C is a negated formula ¬D
    (_, D) = C
    if type(D) == int:      # D is a variable
      return C              # Since C is already in NNF
```

```
    elif len(D) == 2:       # C is a double negation ¬¬A for some A
      (_, A) = D
      return nnf(A)
    else:                   # len(D) == 3. D is ¬(A ∧ B) or ¬(A ∨ B) for some A,B
      (A, op, B) = D
      otherop = "and" if op == "or" else "or"
      return ( nnf(("not", A)), otherop, nnf(("not", B)) )
  else:                     # len(C) == 3. C is (A ∧ B) or (A ∨ B) for some A,B
    (A, op, B) = C
    return ( nnf(A), op, nnf(B) )
```

1.(c)

We adapt the proof from 1.(a).

Let size : $\mathcal{F} \to \mathcal{F}$ be the function that counts the number of logical connective occurrences (i.e. strings) that occur in the nested tuple (or size(i) = 0 for i $\in \mathbb{N}$). Other size functions that work well are the maximum depth of the formula, and the number of connectives plus the number of variables.

First we prove $\forall$n $\in \mathbb{N}$. $\forall$C $\in \mathcal{F}$. size(C) = n implies nnf(C) $\in$ NNF. This is by complete induction. Let n $\in \mathbb{N}$ be arbitrary. Assume the claim holds for all n' < n, i.e. $\forall$n' < n. $\forall$A $\in \mathcal{F}$. size(A) = n' implies nnf(A) $\in$ NNF. Let C be an arbitrary formula of $\mathcal{F}$ of size n.

1. Case n = 0. Then C is a variable i, and nnf(C) = i, which has no negation symbols, and thus is trivially in NNF.
2. Case n ≥ 1, and C is a tuple of the form (A, op, B) for op $\in$ {"and","or"}, where A and B are strictly smaller than C according to size; i.e. size(A), size(B) < n. Thus we may use the IH at the numbers size(A) and size(B). Thus nnf(A) and nnf(B) are in NNF. The rest is the same as in part (a): nnf on input C returns the formula (nnf(A), op, nnf(B)), which has no additional negation symbols other than those in nnf(A) and nnf(B). Thus nnf(C) is in NNF as well.
3. Case n ≥ 1, and C is a tuple of the form ("not", D).
   1. Subcase, D has the form ("not", A). The function returns nnf(A), which is in NNF by the IH.
   2. Subcase, D has the form (A, op, B). We use the IH on size(("not", A)) and size(("not, B)). This is justified because n = 1 + size(A) + size(B) + 1 (first 1 is for the ¬, second for op), and size(A) and size(B) are non-negative. Thus size(("not", A)) = 1 + size(A) < n and similarly for ("not", B). Finally, as with the earlier case, nnf does not add negations to the results of nnf(("not", A)) and nnf(("not", B)), so nnf(C) is in NNF.
   3. Subcase, D is a variable. Then C is already in NNF, and nnf(C) = C.

Now we prove $\forall$n $\in \mathbb{N}$. $\forall$C $\in \mathcal{F}$. size(C) = n implies nnf(C) ≡ C. This is by complete induction. Let n $\in \mathbb{N}$ be arbitrary. Assume the claim holds for all n' < n, i.e. $\forall$n' < n. $\forall$A $\in \mathcal{F}$. size(A) = n' implies nnf(A) ≡ A. Let C be an arbitrary formula of $\mathcal{F}$ of size n.

1. Case n = 0. Then C is a variable i, and nnf(i) = i. Every formula is logically equivalent to itself.
2. Case n = 1, and C is ("not", i) for some natural i. nnf(C) = C, so this case is trivial as well.
3. Case n ≥ 1, and C is ("not", ("not", A)), or (A, op, B), or ("not", (A, op, B)) for some op $\in$ {"and","or"}. Each case follows from the IH for size(A), size(B), size(A) + 1, size(B) + 1 (and the validity of this is argued in the previous proof about NNF), combined with one of the following facts about propositional logic:
   - A' ≡ A and B' ≡ B implies (A', op, B') ≡ (A, op, B)
   - X ≡ ("not",A) and Y ≡ ("not",B) implies (X, "or", Y) ≡ (("not", (A, "and", B))) and (X, "and", Y)

≡ (("not", (A, "or", B)))

- A' ≡ A implies A' ≡ ("not", ("not", A))

CSC 236 Winter 2015

Assignment 2

Solutions to Question 2

(1) Posted separately.

(2)

(a) **Termination**: see (c), which never mentions k_i nor l_i, so can be read for this code.
Invariant I'. For $i \in \mathbb{N}$, let $I'(i)$ be: $a_i = k_i m$ and $b_i = l_i n$.
Prove I' is true for all natural numbers, by Simple Induction.
$I\prime(0)$. From code: $a_0 = m = 1 \cdot m = k_0 m$ and $b_0 = n = 1 \cdot n = l_0 n$.
Inductive Step. Let $i \in \mathbb{N}$.
(IH) Assume $I'(i)$, and an $i+1$st iteration, in particular $a_i \neq b_i$.

- Case $a_i < b_i$.
  From code: $b_{i+1} = b_i$ and $l_{i+1} = l_i$, so they still have the required relationship.
  From code, (IH), algebra, and code: $a_{i+1} = a_i + m = k_i m + m = (k_i + 1) m = k_{i+1} m$.
- Case $b_i < a_i$ [just mirrors the previous case].
  From code: $a_{i+1} = a_i$ and $k_{i+1} = k_i$, so they still have the required relationship.
  From code, (IH), algebra, and code: $b_{i+1} = b_i + n = l_i n + n = (l_i + 1) n = l_{i+1} n$.

Post-Condition: At termination index t the loop condition says $a_t = b_t$.
Then by $I'(t)$: $b_t = l_t n$ and $b_t = a_t = k_t m$.

(b) See (c), which never mentions k_i nor l_i, and proves the new part of the post-condition.

(c) **Invariant** I. For $i \in \mathbb{N}$, let $I(i)$ be:

- a_i is a positive multiple of m and b_i is a positive multiple of n
- if c is a positive multiple of both m and n then $a_i \leq c$ and $b_i \leq c$

Prove I is true for all natural numbers, by Simple Induction.
$I(0)$. From code: $a_0 = m = 1 \cdot m$ and $b_0 = n = 1 \cdot n$, which are the smallest positive multiples of m and n, respectively.
Inductive Step. Let $i \in \mathbb{N}$.
(IH) Assume $I(i)$, and an $i+1$st iteration, in particular $a_i \neq b_i$.
Let c be a positive multiple of both m and n, i.e. $c = \alpha m = \beta n$ for positive natural numbers α and β.
From (IH): $a_i \leq c$ and $b_i \leq c$.

- Case $a_i < b_i$.
  From code: $b_{i+1} = b_i$, so b_{i+1} still has the required properties from the (IH).
  And from (IH): $a_i = km$ for a positive natural number k.
  So $km = a_i < b_i \leq c = \alpha m$.
  Dividing by positive number m: $k < \alpha$, which for integers means $k \leq \alpha - 1$.
  From code: $a_{i+1} = a_i + m = km + m = (k+1) m$, a positive multiple of m.
  And $(k+1) m \leq (\alpha - 1 + 1) m = c$, so a_{i+1} is at most c.
- Case $b_i < a_i$ [just mirrors the previous case].
  From code: $a_{i+1} = a_i$, so a_{i+1} still has the required properties from the (IH).
  And from (IH): $b_i = ln$ for a positive natural number l.
  So $ln = b_i < a_i \leq c = \beta n$.
  Dividing by positive number n: $l < \beta$, which for integers means $l \leq \beta - 1$.
  From code: $b_{i+1} = b_i + n = ln + n = (l+1) n$, a positive multiple of n.
  And $(l+1) n \leq (\beta - 1 + 1) n = c$, so b_{i+1} is at most c.

Variant: $mn - \min(a_i, b_i)$.
From the precondition: m and n are positive natural numbers.
From $I(i)$: a_i and b_i are multiples of m and n, so are integers, so the variant is always an integer.
An iteration increases the minimum of a_i and b_i by positive number m or n, so the variant decreases.
From $I(i)$: a_i and b_i are at most mn since that is a positive multiple of m and of n.
So the variant is non-negative, so always a natural number.
A proper variant shows the loop terminates.
Post-Condition: At termination index t, the loop condition says $a_t = b_t$.
By $I(t)$: if c is a positive multiple of both m and n then $b_t \leq c$, and $b_t = a_t$ is a positive multiple of both n and m.