Logic for Computer Science

Exercises, tutorial 8

Pascal Fontaine

December 6, 2021

1. Consider the following inference rule. Is it correct?

$$\frac{P(a), \forall x\,[P(x) \Rightarrow P(f(x))]}{\forall x P(x)}$$

2. What is the logical relation between the following formulas?

- $\forall x \exists y\,[P(x) \wedge Q(y)]$
- $\exists y \forall x\,[P(x) \wedge Q(y)]$

3. What is the logical relation between the following formulas?

- $\forall x \exists y\,[P(x) \Rightarrow Q(y)]$
- $\forall x P(x) \Rightarrow \exists y Q(y)$
- $\exists x P(x) \Rightarrow \exists y Q(y)$

4. Write in formal logic (you can use the equality predicate, if needed):

1. Every client has a manager
2. Every client has at most one manager
3. There is one and only one CEO
4. Every manager has a supervisor, except the CEO
5. A manager supervisor is a superior
6. The superior of a manager superior is also one of the manager superior

7. A manager of a client owns the right to execute an operation on behalf of the client

8. If a manager owns the right to execute an operation on behalf of a client, all his superiors owns the right to execute an operation on behalf of this client

9. The CEO is a superior of every manager

Then prove formally that the CEO owns the right to execute an operation on behalf of all clients.

Solution: Several translations are possible, we only give here one suitable way to formalize the above sentences. We use the following symbols

- Client(x): x is a client
- manages(y, x): y is a manager for x
- CEO(x): x is a CEO
- ceo: the CEO (axiomatized as the unique CEO)
- Allowed(y, x): y owns the right to execute an operation on behalf of a client x
- supervises(y, x): y is a supervisor of x
- superior(y, x): y is a superior of x

Then the above sentences are formalized as followed:

1. $\forall x \,.\, \text{Client}(x) \Rightarrow \exists y \,.\, \text{manages}(y, x)$
2. $\forall x \,.\, \text{Client}(x) \Rightarrow \forall y, z \,.\, (\text{manages}(y, x) \wedge \text{manages}(z, x)) \Rightarrow y = z$
3. $\forall x \,.\, \text{CEO}(x) \Leftrightarrow x = \text{ceo}$
4. $\forall x \,.\, \exists z \ \text{manages}(x, z) \Rightarrow [x = \text{ceo} \vee \exists y \,.\, \text{supervises}(y, x)]$
5. $\forall xy \,.\, \text{supervises}(y, x) \Rightarrow \text{superior}(y, x)$
6. $\forall xy \,.\, [\text{superior}(y, x) \wedge \text{superior}(z, y)] \Rightarrow \text{superior}(z, x)$
7. $\forall xy \,.\, [\text{Client}(x) \wedge \text{manages}(y, x)] \Rightarrow \text{Allowed}(y, x)$
8. $\forall xyz \,.\, [\text{Client}(x) \wedge \text{Allowed}(y, x) \wedge \text{superior}(z, y)] \Rightarrow \text{Allowed}(z, x)$

9. $\forall x \,.\, \text{superior}(\text{ceo}, x)$

We have to prove that $\forall x \,.\, \text{Client}(x) \Rightarrow \text{Allowed}(\text{ceo}, x)$ is a logical consequence of what precedes, or equivalently that $\neg\forall x \,.\, \text{Client}(x) \Rightarrow \text{Allowed}(\text{ceo}, x)$, together with the above formulas is unsatisfiable. After Skolemization, this gives

$$\text{Client}(s_1) \wedge \neg\text{Allowed}(\text{ceo}, s_1)$$

where s_1 is a fresh Skolem symbol. Consider the first formula, after Skolemization:

$$\forall x \,.\, \text{Client}(x) \Rightarrow \text{manages}(s_2(x), x)$$

and instantiate with s_1:

$$\text{Client}(s_1) \Rightarrow \text{manages}(s_2(s_1), s_1)$$

Instantiate formula 7 with s_1 and $s_2(s_1)$:

$$[\text{Client}(s_1) \wedge \text{manages}(s_2(s_1), s_1)] \Rightarrow \text{Allowed}(s_2(s_1), s_1))$$

Instantiate formula 8 with s_1, $s_2(s_1)$, and ceo:

$$[\text{Client}(s_1) \wedge \text{Allowed}(s_2(s_1), s_1) \wedge \text{superior}(\text{ceo}, s_2(s_1))] \Rightarrow \text{Allowed}(\text{ceo}, s_1)$$

and finally, together with the instance of formula 9

$$\text{superior}(\text{ceo}, s_2(s_1))$$

we obtain an unsatisfiable set of instances, which shows that indeed $\forall x \,.\, \text{Client}(x) \Rightarrow \text{Allowed}(\text{ceo}, x)$ is a consequence of formulas 1, 7, 8 and 9.
Btw, see `https://aws.amazon.com/security/provable-security/`.

5. Consider the following algorithm (selection sort)

input : a possibly unsorted array a of n elements
output: a sorted array that contains the initial elements of a

```
1 for i ← 0 to n − 1 do
2 |   for j ← i + 1 to n do
3 |   |   if a[j] < a[i] then
4 |   |   |   Swap(a[i],a[j])
```

(notice that the body of the loop is executed last for its **to** value minus 1)

For simplicity, consider that indices and elements in the array are all natural numbers in $\mathbb{N}$, and that i and j are initially equal to 0

- Design invariants
- Prove that the loop invariants are inductive
- Prove that the array is sorted on termination

Solution:
The returned array is a permutation of the input

We will not prove that the elements of the returned array are in bijection with the input array: this would either require a higher-order quantifier, or a large theory. Taking the multiplicity of values in the array into account is fairly complicated in FOL, but it is relatively easy to prove in FOL that new values do not appear, and old ones do not disappear, proving that the two following invariants hold

- $I_1 = \forall x \exists y \,.\, a[x] = a_0[y]$
- $I_2 = \forall x \exists y \,.\, a_0[x] = a[y]$

with a_0 being the initial array. Then, the algorithm annotated with loop invariants is the following:

input : a possibly unsorted array a of n elements
output: a sorted array that contains the initial elements of a

```
1 for i ← 0 to n − 1 do
      I1 = ∀x∃y . a[x] = a0[y]
      I2 = ∀x∃y . a0[x] = a[y]
2     for j ← i + 1 to n do
          I1 = ∀x∃y . a[x] = a0[y]
          I2 = ∀x∃y . a0[x] = a[y]
3         if a[j] < a[i] then
4             Swap(a[i],a[j])
```

To prove that those invariants hold is a bit difficult to grasp since the induction is kind of evolved, due to the two loops.

Let's prove that initially, before the loops, the invariants are all verified. Initially, $a = a_0$ so they reduce to $\top$ for the outer loop. For the inner loops, since they will be required as loop invariants for the outer loop, they will necessary hold at the beginning of the inner loop.

Now it remains to prove that they are preserved by the body of the loops. The invariants I_1 and I_2 do not depend on anything but a. Only the inner loop can modify their truth value. They are preserved however if

$\{I_k\}$ `Swap(`$a[i]$,$a[j]$`)` $\{I_k\}$

holds, for $k = 1, 2$. The Hoare triple $\{I\}\ C\ \{I'\}$ holds if, from a state satisfying property I, code C runs without error and terminates in a state satisfying I'.

There are several ways to transform a Hoare triple into a first-order logic formula. One is by substitution, e.g., $\{I\}\ x \leftarrow E\ \{I'(x)\}$ becomes $I \Rightarrow I'(E)$. Another one is using primed variables and introduce equalities, e.g., $\{I\}\ x \leftarrow E\ \{I'(x)\}$ becomes $(I \wedge x' = E) \Rightarrow I'(x')$. In the current case, the second way is easier, and the considered Hoare triple becomes

$$\Big(I_k \wedge (\forall x\,.\, i = x \vee j = x \vee a'[x] = a[x]) \wedge a'[j] = a[i] \wedge a'[i] = a[j]\Big) \Rightarrow I'_k$$

where I'_k is I_k with a' replacing a, which is valid if and only if

$$I_k \wedge (\forall x\,.\, i = x \vee j = x \vee a'[x] = a[x]) \wedge a'[j] = a[i] \wedge a'[i] = a[j] \wedge \neg I'_k$$

is unsatisfiable. For I_1, after Skolemization, we get the set of formulas

$$\Big\{\forall x\,.\, a[x] = a_0[s_y(x)], \forall x\,.\, i = x \vee j = x \vee a'[x] = a[x],\\ a'[j] = a[i], a'[i] = a[j], \forall y\,.\, a'[s_x] \neq a_0[y]\Big\}.$$

The following set of instances of the above set of formulas is unsatisfiable:

$$\Big\{a[s_x] = a_0[s_y(s_x)], a[i] = a_0[s_y(i)], a[j] = a_0[s_y(j)],\\ i = s_x \vee j = s_x \vee a'[s_x] = a[s_x], a'[j] = a[i], a'[i] = a[j],\\ a'[s_x] \neq a_0[s_y(s_x)], a'[s_x] \neq a_0[s_y(i)], a'[s_x] \neq a_0[s_y(j)]\Big\}$$

To convince oneself of it, see that this reduces in the three cases $s_x = i$, $s_x = j$, $s_x \neq i \wedge s_x \neq j$ respectively to

- $\Big\{a[j] = a_0[s_y(j)], a'[i] = a[j], a'[i] \neq a_0[s_y(j)]\Big\}$
- $\Big\{a[i] = a_0[s_y(i)], a'[j] = a[i], a'[j] \neq a_0[s_y(i)]\Big\}$
- $\Big\{a[s_x] = a_0[s_y(s_x)], a'[s_x] = a[s_x], a'[s_x] \neq a_0[s_y(s_x)]\Big\}$

The case for I_2 is similar. So the body of the inner loop preserves the invariants, and they are thus loop invariants for the inner loop, and as a consequence, they are also invariants for the outer loop.

The returned array is sorted

input : a possibly unsorted array a of n elements
output: a sorted array that contains the initial elements of a

```
1 for i ← 0 to n − 1 do
      {∀x, y . x ≤ y ≤ i ⇒ a[x] ≤ a[y]}
        ∀x, y . x < i ≤ y < n ⇒ a[x] ≤ a[y]}
2     for j ← i + 1 to n do
          I3 = ∀x, y . x ≤ y ≤ i ⇒ a[x] ≤ a[y]
          I4 = ∀x, y . x < i ≤ y < n ⇒ a[x] ≤ a[y]
          I5 = ∀x . i < x < j ⇒ a[i] ≤ a[x]
3         if a[j] < a[i] then
4             Swap(a[i],a[j])
```

The property we would like to prove is that

$$P = \forall x, y \,.\, x \leq y < n \Rightarrow a[x] \leq a[y]$$

holds at the end of the algorithm. The following formulas are handy:

- $I_3 = \forall x, y \,.\, x \leq y \leq i \Rightarrow a[x] \leq a[y]$
- $I_4 = \forall x, y \,.\, x < i \leq y < n \Rightarrow a[x] \leq a[y]$
- $I_5 = \forall x \,.\, i < x < j \Rightarrow a[i] \leq a[x]$

The two first formulas are loop invariants for the outer loop and the inner loop. The last one is an invariant of the inner loop.

We will prove that these formulas hold initially, and that they are preserved in the loop. Further, we will show that they are, together, an inductive invariant: if they hold before the loop body, they hold after the loop body (including incrementation of the loop counter), without assuming any further knowledge about the algorithm or the properties satisfied by the array.

Initial conditions

Let's prove that initially, the invariants are all verified. For $k = 3, 4$, it means they have to hold before the outer loop, where we can assume $i = 0$. We want to prove that

$$i = 0 \Rightarrow I_k$$

is valid or equivalently

$$i = 0 \wedge \neg I_k$$

is unsatisfiable.

For I_3, we want to show the unsatisfiability of

$$i = 0 \wedge \neg \forall x, y \,.\, x \leq y \leq i \Rightarrow a[x] \leq a[y]$$

i.e., after Skolemization

$$i = 0 \wedge s_x \leq s_y \leq i \wedge a[s_x] \not\leq a[s_y]$$

Because of arithmetic, $s_x = 0 \wedge s_y = 0$ is a consequence of $i = 0 \wedge s_x \leq s_y \leq i$, and one can consider

$$s_x = 0 \wedge s_y = 0 \wedge i = 0 \wedge s_x \leq s_y \leq i \wedge a[s_x] \not\leq a[s_y]$$

or, due to equality

$$s_x = 0 \wedge s_y = 0 \wedge i = 0 \wedge s_x \leq s_y \leq i \wedge a[0] \not\leq a[0]$$

which is unsatisfiable due to arithmetics ($a[0] \not\leq a[0]$ reduces to $\bot$).

For I_4, we want to show the unsatisfiability of

$$i = 0 \wedge \neg \forall x, y \,.\, x < i \leq y < n \Rightarrow a[x] \leq a[y]$$

or after Skolemization

$$i = 0 \wedge s_x < i \leq s_y < n \wedge a[s_x] \not\leq a[s_y]$$

which is unsatisfiable (in arithmetic) since $s_x < i$ reduces to $\bot$ when $i = 0$. There is indeed no natural number strictly smaller than 0.

Now, for the inner loop, we want to show that I_3 to I_5 hold at the beginning of the loop. For I_3 and I_4, it is trivial since (we will show later that they are loop invariants for the outer loop and as such) they hold at the beginning of each loop, and they do not depend on j, so we essentially have to prove that $I_k \Rightarrow I_k$ is valid, for $k = 3, \ldots 4$.

For I_5, we want to show that I_5 holds at the beginning of the inner loop, that is, it is a consequence of I_3 and I_4, and the fact that $j = i + 1$. In fact, I_3 and I_4 are not mandatory since

$$j = i + 1 \Rightarrow \forall x \,.\, i < x < j \Rightarrow a[i] \leq a[x]$$

is valid. Indeed

$$j = i + 1 \wedge \neg \forall x \,.\, i < x < j \Rightarrow a[i] \leq a[x]$$

is unsatisfiable: after Skolemization, it becomes

$$j = i + 1 \wedge i < s_x < j \wedge a[i] \not\leq a[x]$$

and, since there is no natural between i and $i+1$, so $j = i+1 \wedge i < s_x < j$ is unsatisfiable, due to trivial arithmetic reasoning (that could come from, e.g., reasoning on Presburger axioms).

Inductive invariant

The more complicated part of the proof comes from showing that the invariants are preserved by the loop body. They cannot be considered separately (taken separately, they are not inductive). We do not prove in details all invariants, but focus on I_5.

Basically, we want to prove that I_5 is preserved by the body of the inner loop (including incrementing j)

$\{\forall x \,.\, i < x < j \Rightarrow a[i] \leq a[x]\}$
1 **if** $a[j] < a[i]$ **then**
2 $\quad$ `Swap(`$a[i]$`,`$a[j]$`)`
3 $j \leftarrow j + 1$
$\{\forall x \,.\, i < x < j \Rightarrow a[i] \leq a[x]\}$

This can be translated into the following formula (the presentation as a sequent stands for the conjunction of the formulas over the line implies the formula under the line)

$$\frac{\begin{array}{l}\forall x \,.\, i < x < j \Rightarrow a[i] \leq a[x] \\ a[j] < a[i] \Rightarrow (a'[j] = a[i] \wedge a'[i] = a[j]) \\ a[j] \not< a[i] \Rightarrow (a'[i] = a[i] \wedge a'[j] = a[j]) \\ \forall x \,.\, x = i \vee x = j \vee a'[x] = a[x] \\ j' = j + 1\end{array}}{\forall x \,.\, i < x < j' \Rightarrow a'[i] \leq a'[x]}$$

which is valid if and only if the set containing the following formulas is unsatisfiable:

$$\begin{array}{l}\forall x \,.\, i < x < j \Rightarrow a[i] \leq a[x] \\ a[j] < a[i] \Rightarrow (a'[j] = a[i] \wedge a'[i] = a[j]) \\ a[j] \not< a[i] \Rightarrow (a'[i] = a[i] \wedge a'[j] = a[j]) \\ \forall x \,.\, x = i \vee x = j \vee a'[x] = a[x] \\ j' = j + 1 \\ \neg\forall x \,.\, i < x < j' \Rightarrow a'[i] \leq a'[x]\end{array}$$

that is, after Skolemization

$$\begin{array}{l}
\forall x \,.\, i < x < j \Rightarrow a[i] \leq a[x] \\
a[j] < a[i] \Rightarrow (a'[j] = a[i] \wedge a'[i] = a[j]) \\
a[j] \not< a[i] \Rightarrow (a'[i] = a[i] \wedge a'[j] = a[j]) \\
\forall x \,.\, x = i \vee x = j \vee a'[x] = a[x] \\
j' = j + 1 \\
i < s_x < j' \\
a'[i] \not\leq a'[s_x]
\end{array}$$

For the case of $s_x < j$, the first formula together with the fourth guarantees there is no model. For $s_x = j$, the second and the third formulas imply $a'[i] \leq a'[s_x]$. Therefore the formula is unsatisfiable.

Property

When $i = n - 1$, from I_3 we know that all but the last element of the array is sorted, and from I_4, we know that the last element is greater than all the previous ones. The array must thus be sorted.

- $I_3 = \forall x, y \,.\, x \leq y \leq i \Rightarrow a[x] \leq a[y]$
- $I_4 = \forall x, y \,.\, x < i \leq y < n \Rightarrow a[x] \leq a[y]$

```
method Sort(a: array<int>) returns (b :array<int>)
   requires 0 ≤a.Length
   ensures 0 ≤a.Length
   modifies a
   ensures ∀x, y • 0 ≤x ≤y < a.Length ⟹a[x] ≤a[y]
{
   if a.Length ≤1 {
     return a;
     }
   for i := 0 to a.Length − 1
     invariant 0 ≤i < a.Length
     invariant ∀ x, y • 0 ≤x ≤y ≤i ⟹a[x] ≤a[y]
     invariant ∀ x, y • 0 ≤x < i ≤y < a.Length ⟹a[x] ≤a[y]
   {
      for j := i + 1 to a.Length
        invariant i < j ≤a.Length
        invariant ∀ x, y • 0 ≤x ≤y ≤i ⟹a[x] ≤a[y]
        invariant ∀ x, y • 0 ≤x < i ≤y < a.Length ⟹a[x] ≤a[y]
        invariant ∀ x • i < x < j ⟹a[i] ≤a[x]
      {
```

```
        if a[j] < a[i] {
            a[j], a[i] := a[i], a[j];
        }
      }
    }
    return a;
}
```

6. Consider the following algorithm in Dafny.

```
method binary_search(a: array<int>, v: int) returns (b : int)
{
    var L : int;
    var R : int;
    L := 0;
    R := a.Length − 1;
    while L ≤R
      {
        var m : int;
        m :=L + (R − L) / 2;
        if a[m] < v { L := m + 1; }
        else if a[m] > v { R :=m − 1; }
        else { return m; }
      }
    return −1;
}
```

Add **requires**, **modifies**, **ensures** and **invariant** annotations to prove correctness of the algorithm. As a helper, here is the code for the selection sort algorithm, together with enough invariant annotations to prove that the array is sorted on termination:

```
method Sort(a: array<int>) returns (b :array<int>)
    requires 0 ≤a.Length
    ensures 0 ≤a.Length
    modifies a
    ensures ∀x, y • 0 ≤x ≤y < a.Length ⟹a[x] ≤a[y]
{
    if a.Length ≤1 {
      return a;
      }
    for i := 0 to a.Length − 1
      invariant 0 ≤i < a.Length
```

```
    invariant ∀ x, y • 0 ≤x ≤y ≤i ⟹a[x] ≤a[y]
    invariant ∀ x, y • 0 ≤x < i ≤y < a.Length ⟹a[x] ≤a[y]
  {
    for j := i + 1 to a.Length
      invariant i < j ≤a.Length
      invariant ∀ x, y • 0 ≤x ≤y ≤i ⟹a[x] ≤a[y]
      invariant ∀ x, y • 0 ≤x < i ≤y < a.Length ⟹a[x] ≤a[y]
      invariant ∀ x • i < x < j ⟹a[i] ≤a[x]
    {
      if a[j] < a[i] {
          a[j], a[i] := a[i], a[j];
      }
    }
  }
  return a;
}
```

Solution:

```
method binary_search(a: array<int>, v: int) returns (b : int)
  requires 0 < a.Length
  requires ∀ x, y • 0 ≤x ≤y < a.Length ⟹a[x] ≤a[y]
  ensures b =−1 ∨0 ≤b < a.Length
  ensures b =−1 ⟹∀ x • 0 ≤x < a.Length ⟹a[x] ≠v
  ensures b ≥0 ⟹ a[b] =v
{
  var L : int;
  var R : int;
  L := 0;
  R := a.Length − 1;
  while L ≤R
    invariant 0 ≤L ≤a.Length
    invariant −1 ≤R < a.Length
    invariant L ≤R + 1
    invariant ∀ x • 0 ≤x < L ⟹a[x] < v
    invariant ∀ x • R < x < a.Length ⟹v < a[x]
    {
      var m : int;
      m :=L + (R − L) / 2;
      if a[m] < v { L := m + 1; }
      else if a[m] > v { R :=m − 1; }
      else { return m; }
```

```
        }
    return −1;
}
```