Exercise Sheet 1
Software Verficiation

Matthew Parkinson

April 18, 2010

1 Introduction

Exercise 1.1 (Slide 23). *(1) Provide a forwards assignment axioms: an assignment axiom which takes an arbitrary pre-condition and calculates a post-condition. That is, give a formula for ??? in terms of P, x and E in the following triple.*

$$\{P\}x := E\{???\}$$

(2) Show your answer can derive the backwards assignment axiom

$$\{P[x := E]\}x := E\{P\}$$

(3) Show your answer can be derived from the backwards assignment axiom.

Answers

(1) We can give the rule as

$$\{P\}x := E\{\exists Y.\ (\exists x.\ P \wedge Y = E) \wedge x = Y\}$$

(3) We can derive this new forwards assignment axiom as follows. Using the backwards assignment axiom we can get:

$$\{(\exists Y.(\exists x.\ P \wedge Y = E) \wedge x = Y)[x := E]\}x := E\{\exists Y.\ (\exists x.\ P \wedge Y = E) \wedge x = Y\}$$

applying substitution, we can get:

$$\{(\exists Y.(\exists x.\ P \wedge Y = E) \wedge E = Y)\}x := E\{\exists Y.\ (\exists x.\ P \wedge Y = E) \wedge x = Y\}$$

renaming x to a fresh variable and extending its scope, we get

$$\{(\exists Y.\exists x'.\ P[x := x'] \wedge Y = E[x := x'] \wedge E = Y)\}x := E\{\exists Y.\ (\exists x.\ P \wedge Y = E) \wedge x = Y\}$$

We can eliminate the Y in the pre-condition

$$\{\exists x'.\ P[x := x'] \wedge E = E[x := x']\}x := E\{\exists Y.\ (\exists x.\ P \wedge Y = E) \wedge x = Y\}$$

The pre-condition is implied by P

$$\{P\}x := E\{\exists Y.\ (\exists x.\ P \wedge Y = E) \wedge x = Y\}$$

giving the forwards axiom.

(2) We can derive the backwards assignment axiom as

$$\{P[x := E]\}x := E\{\exists Y.\ (\exists x.\ P[x := E] \wedge Y = E) \wedge x = Y\}$$

By the rule of consequence, we can modify the postcondition to

$$\{P[x := E]\}x := E\{\exists Y.\ (\exists x.\ P[x := Y] \wedge Y = E) \wedge x = Y\}$$

and dropping the conjunction, and applying the substitution gives

$$\{P[x := E]\}x := E\{\exists Y.\ P[x := Y] \wedge x = Y\}$$

which simplifies to

$$\{P[x := E]\}x := E\{P\}$$

Exercise 1.2. *Prove the following triples using the proof rules:*

$$\begin{array}{rcl} \{x = 4\} & x := x + 1 & \{x = 5\} \\ \{y = 3 \wedge x = 3\} & y := x + y & \{y = 6 \wedge x = 3\} \\ \{odd(x)\} & y := x + x + x & \{odd(y) \wedge odd(x)\} \end{array}$$

Answers

$$\dfrac{x = 4 \Rightarrow x + 1 = 5 \qquad \dfrac{}{\{x + 1 = 5\}\, x := x + 1\, \{x = 5\}}\text{Assign} \qquad x = 5 \Rightarrow x = 5}{\{x = 4\}\, x := x + 1\, \{x = 5\}}\text{Conseq}$$

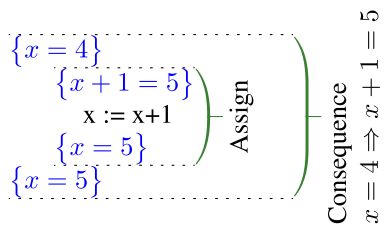
$$\dfrac{y = 3 \wedge x = 3 \Rightarrow x + y = 6 \wedge x = 3 \qquad \dfrac{}{\{x + y = 6 \wedge x = 3\}\, y := x + y\, \{y = 6 \wedge x = 3\}}\text{Assign}}{\{y = 3 \wedge x = 3\}\, y := x + y\, \{y = 6 \wedge x = 3\}}\text{Conseq}$$

$$\dfrac{odd(x) \Rightarrow odd(x + x + x) \wedge odd(x) \qquad \dfrac{}{\{odd(x + x + x) \wedge odd(x)\}\, y := x + x + x\, \{odd(y) \wedge odd(x)\}}\text{Assign}}{odd(x)\}\, y := x + x + x\, \{odd(y) \wedge odd(x)}\text{Conseq}$$

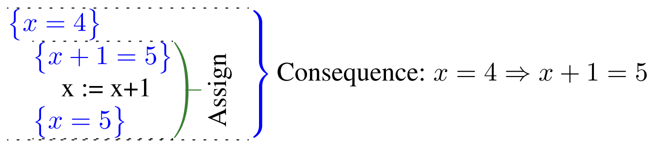
The previous style of proofs tends to be quite hard to read, so we can present them as proof outlines.

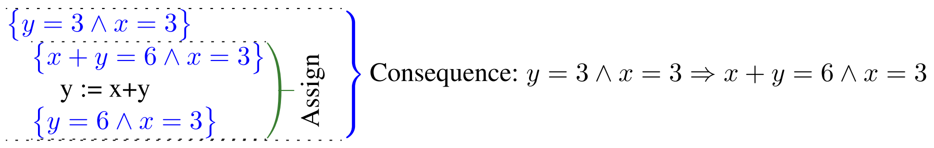
Exercise 1.3. *Do the same proofs as full proof outlines.*

Answers

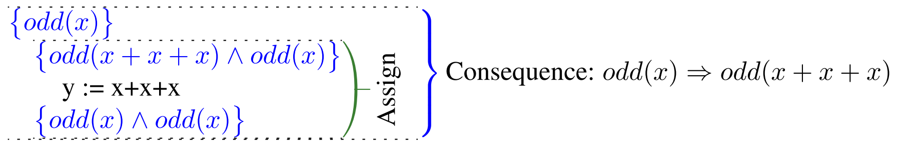


Note in your answers it is perfectly fine to drop duplicated annotations where clear, for example, the above could have been written:





$\{y = 3 \wedge x = 3\}$
$\{x + y = 6 \wedge x = 3\}$
y := x+y
$\{y = 6 \wedge x = 3\}$
Assign
Consequence: $y = 3 \wedge x = 3 \Rightarrow x + y = 6 \wedge x = 3$


$\{odd(x)\}$
$\{odd(x+x+x) \land odd(x)\}$
y := x+x+x
$\{odd(x) \land odd(x)\}$
Assign
Consequence: $odd(x) \Rightarrow odd(x+x+x)$

Exercise 1.4. *Show*

$$\frac{\{P\}\,C\,\{Q\}}{\{\exists X.P\}\,C\,\{\exists X.Q\}} \tag{1}$$

is equi-expressive to

$$\frac{\{P\}\,C\,\{Q\}}{\{\exists X.P\}\,C\,\{Q\}} \tag{2}$$

provided $X \notin FV(Q)$

Answers Show any proof with (2) can be expressed using (1) and the other structural rules.

Observe $X \notin FV(Q)$ means $\exists X.Q \Rightarrow Q$, as Q is independent of the value of X. So

$$\dfrac{\dfrac{\{P\}\,C\,\{Q\}}{\{\exists X.P\}\,C\,\{\exists X.Q\}}\,(1) \qquad \exists X.Q \Rightarrow Q}{\{\exists X.P\}\,C\,\{Q\}}\ \text{Conseq}$$

For the other direction. Note that it is always true that

$$Q \Rightarrow \exists X.Q$$

So,

$$\dfrac{\dfrac{\{P\}\,C\,\{Q\} \qquad Q \Rightarrow \exists X.Q}{\{P\}\,C\,\{\exists X.Q\}}\ \text{Conseq}}{\{\exists X.P\}\,C\,\{\exists X.Q\}}\,(2)$$

Exercise 1.5. *Prove the Hoare logic defined in the first section sound.*

Proof. By rule induction on a proof. We may assume the semantics holds of the premises, and must show it holds of the conclusion. For axioms, we must show it holds of the conclusion.

Skip Assume

$$\sigma \models P$$
$$skip, \sigma \rightarrow^* skip, \sigma'$$

Prove

$$\sigma' \models P$$

By inspecting operational semantics, we can see skip cannot reduce. Therefore, $\sigma = \sigma'$. Hence our obligation holds by assumption.

Assignment Assume

$$\sigma \models P[x := E]$$
$$x := E, \sigma \rightarrow^* skip, \sigma'$$

Prove

$$\sigma' \models P$$

By inspecting operational semantics, we can see an assignment can only reduce one way to skip, and then this cannot reduce. Therefore, $\sigma' = \sigma[x := v]$ where $[\![E]\!]_\sigma = v$.

Remember the lemma given in the notes,

$$\sigma \models P[x := E] \iff \sigma[x := [\![E]\!]_\sigma] \models P$$

This proves the case as required.

Sequencing Assume

$$\begin{array}{l} \models \{P\}C_1\{R\} \\ \models \{R\}C_2\{Q\} \\ \sigma \models P \\ C_1; C_2, \sigma \rightarrow^* skip, \sigma' \end{array}$$

Prove

$$\sigma' \models Q$$

We can prove that, if

$$C_1; C_2, \sigma \rightarrow^* skip, \sigma'$$

then, there exists σ'' such that

$$\begin{array}{l} C_1, \sigma \rightarrow^* skip, \sigma'' \\ C_2, \sigma'' \rightarrow^* skip, \sigma' \end{array}$$

by induction on the operational semantics.

Therefore, we can use our first inductive hypothesis: $\models \{P\}C_1\{R\}$, to prove $\sigma'' \models R$, and hence we can use the second inductive hypothesis to prove $\sigma' \models Q$ as requried.

If Assume

$$\begin{array}{l} \models \{P \wedge B\}C_1\{Q\} \\ \models \{P \wedge \neg B\}C_2\{Q\} \\ \sigma \models P \\ if\ B\ then\ C_1\ else\ C_2, \sigma \rightarrow^* skip, \sigma' \end{array}$$

Prove

$$\sigma' \models Q$$

Case split on $[\![B]\!]_\sigma = true$ or $[\![B]\!]_\sigma = false$:

First case, assume $[\![B]\!]_\sigma = true$. Therefore, it must reduce as

$$if\ B\ then\ C_1\ else\ C_2, \sigma \quad \rightarrow \quad C_1, \sigma \quad \rightarrow^* \quad skip, \sigma'$$

We know $\sigma \models P \wedge B$, therefore by inductive hypothesis, we know $\sigma' \models Q$.

Second case, similar. $\square$

2 Loops and control flow

Exercise 2.1. *Show why the loop invariant for the fast exponentiation is not strong enough. Find a proof of this program.*

Answers *The exit of the loop requires us to prove*

$$Y^Z = x * y^z \wedge z \leq 0 \Rightarrow x = Y^Z$$

but we actually need $z = 0$ to prove this.

We can strengthen the loop invariant to include $z \geq 0$, hence then the end of the loop would require us to prove

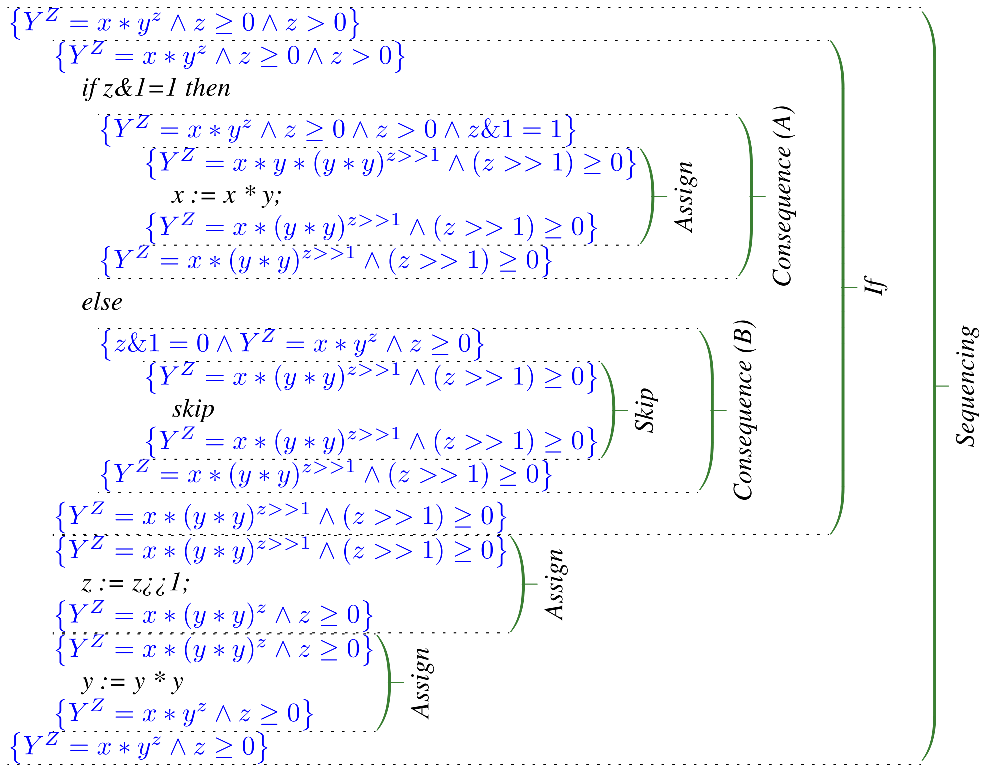
$$Y^Z = x * y^z \wedge z \geq 0 \wedge z \leq 0 \Rightarrow x = Y^Z$$

which is valid.

We must strengthen the pre-condtion of the code, as it is currently not sufficient to establish the loop invariant.

$$y = Y \wedge z = Z \wedge z \geq 0$$

We can verify the loop as



```
{$Y^Z = x * y^z \wedge z \geq 0 \wedge z > 0$}
    {$Y^Z = x * y^z \wedge z \geq 0 \wedge z > 0$}
      if z&1=1 then
        {$Y^Z = x * y^z \wedge z \geq 0 \wedge z > 0 \wedge z\&1 = 1$}
            {$Y^Z = x * y * (y * y)^{z>>1} \wedge (z >> 1) \geq 0$}
              x := x * y;
            {$Y^Z = x * (y * y)^{z>>1} \wedge (z >> 1) \geq 0$}
        {$Y^Z = x * (y * y)^{z>>1} \wedge (z >> 1) \geq 0$}
      else
        {$z\&1 = 0 \wedge Y^Z = x * y^z \wedge z \geq 0$}
            {$Y^Z = x * (y * y)^{z>>1} \wedge (z >> 1) \geq 0$}
              skip
            {$Y^Z = x * (y * y)^{z>>1} \wedge (z >> 1) \geq 0$}
        {$Y^Z = x * (y * y)^{z>>1} \wedge (z >> 1) \geq 0$}
    {$Y^Z = x * (y * y)^{z>>1} \wedge (z >> 1) \geq 0$}
    {$Y^Z = x * (y * y)^{z>>1} \wedge (z >> 1) \geq 0$}
      z := z¿¿1;
    {$Y^Z = x * (y * y)^z \wedge z \geq 0$}
    {$Y^Z = x * (y * y)^z \wedge z \geq 0$}
      y := y * y
    {$Y^Z = x * y^z \wedge z \geq 0$}
{$Y^Z = x * y^z \wedge z \geq 0$}
```

Two of the implications we have used are non-obvious. For the first, used in Consequence (A), we know if $z > 0$ and $z\&1 = 1$, then we know $z = 2(z >> 1) + 1$ and $z >> 1 \geq 0$.

$$\begin{aligned} & Y^Z = x * y^z \wedge z \geq 0 \wedge z > 0 \wedge z\&1 = 1 \\ \Rightarrow \quad & Y^Z = x * y^{2(z>>1)+1} \wedge z >> 1 \geq 0 \\ \Rightarrow \quad & Y^Z = x * y * (y * y)^{z>>1} \wedge z >> 1 \geq 0 \end{aligned}$$

For the second, used in consequence (A), we know if $z > 0$ and $z\&1 = 0$ then $z = 2(z >> 1)$ and $z >> 1 \geq 0$.

$$\begin{aligned} & z\&1 = 0 \wedge Y^Z = x * y^z \wedge z \geq 0 \\ \Rightarrow \quad & Y^Z = x * y^{2(z>>1)} \wedge z >> 1 \geq 0 \\ \Rightarrow \quad & Y^Z = x * (y * y)^{z>>1} \wedge (z >> 1) \geq 0 \end{aligned}$$

Exercise 2.2 (Prime Palindrome)**.**

Answer The proof proceeds as follows:

```
$\{true\}$
    $\{19 = 19\}$
        $\{19 = 19\}$
            i := 19                                                    Assign
        $\{i = 19\}$
    $\{i \geq 19 \wedge (\forall j. 20 \leq j \leq i \Rightarrow \neg prime(j) \neg palindrome(j))\}$    Consequence
    $\{i \geq 19 \wedge (\forall j. 20 \leq j \leq i \Rightarrow \neg prime(j) \neg palindrome(j))\}$
        while true do
            i := i + 1;
            if i > 100 then break
            if not prime(i) then continue
            if palindrome(i) then break                                While (A)        Seq
    $\left\{\begin{array}{l} i > 100 \Rightarrow \forall j. 20 \leq j \leq 100 \Rightarrow \neg prime(j) \vee \neg palindrome(j) \\ \wedge\ 20 \leq i \leq 100 \Rightarrow prime(i) \wedge palindrome(i) \end{array}\right\}$
$\left\{\begin{array}{l} i > 100 \Rightarrow \forall j. 20 \leq j \leq 100 \Rightarrow \neg prime(j) \vee \neg palindrome(j) \\ \wedge\ 20 \leq i \leq 100 \Rightarrow prime(i) \wedge palindrome(i) \end{array}\right\}$
```

To prove the while loop (A), we must show the loop invariant and the negation of the guard imply the break assertion. This is trivial as the guard is true, hence its negation is false, and thus implies anything.

We must also show that the body preserves the loop invariant in the context

$break : i > 100 \Rightarrow \forall j. 20 \leq j \leq 100 \Rightarrow \neg prime(j) \vee \neg palindrome(j)$

$\wedge\ 20 \leq i \leq 100 \Rightarrow prime(i) \wedge palindrome(i)$

and $continue : i \geq 19 \wedge (\forall j. 20 \leq j \leq i \Rightarrow \neg prime(j) \neg palindrome(j))$

```
$\{i \geq 19 \wedge (\forall j.20 \leq j \leq i \Rightarrow \neg prime(j) \neg palindrome(j))\}$
    $\{i \geq 19 \wedge (\forall j.20 \leq j \leq i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
    i := i + 1;                                                                   Assign +Conseq
    $\{i > 19 \wedge (\forall j.20 \leq j < i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
    $\{i > 19 \wedge (\forall j.20 \leq j < i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
    if i > 100 then
        $\{i > 100 \wedge (\forall j.20 \leq j \leq 100 \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
        break                                                                     Break +Conseq
        $\{false\}$
    else                                                                          If
        $\{i \leq 100 \wedge i > 19 \wedge (\forall j.20 \leq j < i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
        skip                                                                      Skip +Conseq
        $\{i \leq 100 \wedge i > 19 \wedge (\forall j.20 \leq j < i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
    $\{i \leq 100 \wedge i > 19 \wedge (\forall j.20 \leq j < i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
    $\{pre\}$
    if not prime(i) then
        $\{\neg prime(i) \wedge i \leq 100 \wedge i > 19 \wedge (\forall j.20 \leq j < i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
        continue                                                                  continue +conseq (B)
        $\{false\}$
    else                                                                          If
        $\{prime(i) \wedge i \leq 100 \wedge i > 19 \wedge (\forall j.20 \leq j < i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
        skip                                                                      skip +conseq
        $\{prime(i) \wedge i \leq 100 \wedge i > 19 \wedge (\forall j.20 \leq j < i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
    $\{prime(i) \wedge i \leq 100 \wedge i > 19 \wedge (\forall j.20 \leq j < i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$     Seq
    $\{prime(i) \wedge i \leq 100 \wedge i > 19 \wedge (\forall j.20 \leq j < i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
    if palindrome(i) then
        $\{palindrome(i) \wedge prime(i) \wedge i \leq 100 \wedge i > 19\}$
        break                                                                     Break +Conseq
        $\{false\}$
    else                                                                          If
        $\{\neg palindrome(i) \wedge i \leq 100 \wedge i > 19 \wedge (\forall j.20 \leq j < i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
        skip                                                                      Skip +Conseq
        $\{i \geq 19 \wedge (\forall j.20 \leq j \leq i \Rightarrow \neg prime(j) \vee \neg palindrome(j))\}$
    $\{i \geq 19 \wedge (\forall j.20 \leq j \leq i \Rightarrow \neg prime(j) \neg palindrome(j))\}$
$\{i \geq 19 \wedge (\forall j.20 \leq j \leq i \Rightarrow \neg prime(j) \neg palindrome(j))\}$
```

Exercise 2.3. *Give a rule for do C while B.*

$$\frac{\{P \vee (I \wedge B)\}\, C\, \{I\}}{\{P\}\, do\, C\ while\ B\, \{I \wedge \neg B\}}$$

Exercise 2.4. *The exercise to prove the soundness of the while rule with break and continue is perhaps too hard. When I set the example I forgot the proof required you to induct on the length of reduction. I have split it into two here. First, we will show the soundness of the while rule without break and continue, this will illustrate the induction on the length argument. The next exercise will extend this to deal with break and continue.*

Answer We can define the while evaluation as

$$\begin{array}{ll} \epsilon[while\ B\ do\ C], \sigma \rightarrow \epsilon[C; while\ B\ do\ C], \sigma & \text{where } [\![B]\!]_\sigma = true \\ \epsilon[while\ B\ do\ C], \sigma \rightarrow \epsilon[skip], \sigma & \text{where } [\![B]\!]_\sigma = false \end{array}$$

We define the n step reduction $\rightarrow^n$ as applying n reduction steps to a term.

We can then define semantics of an assertion for n steps, written $\models_n \{P\}C\{Q\}$, as

$$\begin{array}{l} \sigma \models P \Longrightarrow \\ \qquad \forall m \leq n. C, \sigma \rightarrow^m skip, \sigma' \Longrightarrow \sigma' \models P \end{array}$$

The original semantics from the notes that does not account for the number of steps is equivalent to $\forall n.\ \models_n \{P\}C\{Q\}$

We then prove the while rule sound by assuming

$$\begin{array}{l} \models \{P \wedge B\}\, C\, \{P\} \\ P \wedge \neg B \Rightarrow Q \\ \forall m < n.\ \models_m \{P\}\, while\ B\ do\ C\, \{Q\} \end{array}$$

and must prove

$$\models_n \{P\}\, while\ B\ do\ C\, \{Q\}$$

Hence, we can assume

$$\begin{array}{l} \sigma \models P \\ while\ B\ do\ C, \sigma \rightarrow^m skip, \sigma' \\ m \leq n \end{array}$$

and must prove

$$\sigma' \models Q$$

There are two cases for the reduction of $while\ B\ do\ C, \sigma$ either B evaluates to true or false. First case, assume B is false in σ, therefore

$$while\ B\ do\ C, \sigma \rightarrow skip, \sigma$$

Hence, $\sigma = \sigma'$. By our assumption $P \wedge \neg B \Rightarrow Q$, we know that σ' satisfies Q as required.

Second case, assume B evaluates to true in σ. Therefore, we know

$$\begin{array}{l} \sigma \models P \wedge B \\ C, \sigma \rightarrow^i skip, \sigma'' \end{array}$$

By assumption from premise, $\models \{P \wedge B\}\, C\, \{P\}$, we know:

$$\sigma'' \models P$$

Hence,

$$\begin{array}{l} while\ B\ do\ C, \sigma \\ \rightarrow C; while\ B\ do\ C, \sigma \\ \rightarrow^i skip; while\ B\ do\ C, \sigma'' \\ \rightarrow while\ B\ do\ C, \sigma'' \\ \rightarrow^{m-i-2} while\ B\ do\ C, \sigma' \end{array}$$

Therefore, we can use assumption about shorter lengths of reduction, $\forall m < n.\ \models_m \{P\}\, while\ B\ do\ C\, \{Q\}$, to show $\sigma' \models Q$.

Exercise 2.5. *Prove the soundness of while rule with break and continue, that is, prove*

$$\begin{array}{c} (break : P, continue : Q \models \{Q \wedge B\}C\{Q\}) \quad \wedge \quad (Q \wedge \neg B \Rightarrow P) \\ \wedge \quad (\forall m < n. break : P', continue : Q' \models_m \{Q\}while\ B\ do\ C\{P\}) \\ \Longrightarrow (break : P', continue : Q' \models_n \{Q\}while\ B\ do\ C\{P\} \end{array}$$

Answer Assume

$$\begin{array}{l}(break : P, continue : Q \models \{Q \wedge B\}C\{Q\}) \\ (Q \wedge \neg B \Rightarrow P) \\ \sigma \models Q\end{array}$$

Prove $\forall m \leq n.$

1. $while\ B\ do\ C, [], \sigma \rightarrow^m \epsilon[break], [], \sigma' \Longrightarrow \sigma' \models P'$
2. $while\ B\ do\ C, [], \sigma \rightarrow^m \epsilon[continue], [], \sigma' \Longrightarrow \sigma' \models Q'$
3. $while\ B\ do\ C, [], \sigma \rightarrow^m skip, [], \sigma' \Longrightarrow \sigma' \models P$

We can prove that

$$while\ B\ do\ C, [], \sigma$$

cannot reduce to

$$\epsilon[break], [], \sigma'$$

This can be seen simply by inspecting the semantics. Similarly, the same is true for continue.

Hence, we can assume

$$while\ B\ do\ C, [], \sigma \rightarrow^m skip, [], \sigma'$$

and must prove

$$\sigma' \models P$$

From the semantics, we know that

$$while\ B\ do\ C, [], \sigma \rightarrow C; if\ \neg B\ then\ break\ else\ continue, while\ B\ do\ C, \sigma$$

Now consider the cases for how this reduces, either C reduces to $\epsilon[break]$, $\epsilon[continue]$, $skip$ or it doesn't terminate. By assumption it terminates, therefore consider the remaining three cases. Assume $C, [], \sigma \rightarrow^* \epsilon[break], [], \sigma''$, therefore

$$\begin{array}{l} while\ B\ do\ C, [], \sigma \\ \rightarrow \quad C; if\ \neg B\ then\ break\ else\ continue, while\ B\ do\ C, \sigma \\ \rightarrow \quad \epsilon[break]; if\ \neg B\ then\ break\ else\ continue, while\ B\ do\ C, \sigma'' \\ = \quad \epsilon'[break], while\ B\ do\ C, \sigma'' \\ \rightarrow skip, [], \sigma'' \end{array}$$

By assumption, with $(break : P, continue : Q \models \{Q \wedge B\}C\{Q\})$, we know $\sigma'' \models P$ as required.

Assume $C, [], \sigma \rightarrow^* \epsilon[continue], [], \sigma''$, therefore

$$\begin{array}{l} while\ B\ do\ C, [], \sigma \\ \rightarrow \quad C; if\ \neg B\ then\ break\ else\ continue, while\ B\ do\ C, \sigma \\ \rightarrow \quad \epsilon[continue]; if\ \neg B\ then\ break\ else\ continue, while\ B\ do\ C, \sigma'' \\ = \quad \epsilon'[continue], while\ B\ do\ C, \sigma'' \\ \rightarrow while\ B\ do\ C, [], \sigma'' \end{array}$$

By assumption, with $(break : P, continue : Q \models \{Q \wedge B\}C\{Q\})$, we know $\sigma'' \models Q$. Hence, we can use assumption about shorter length reductions to prove the total reduction satisfies the specification.

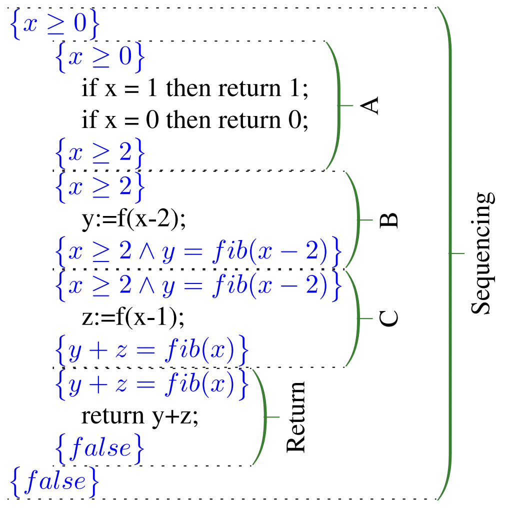
[Note: This proof lacks some detail about the reductions, but is sufficient to outline the soundness of the while rule]

3 Functions

We will use the context

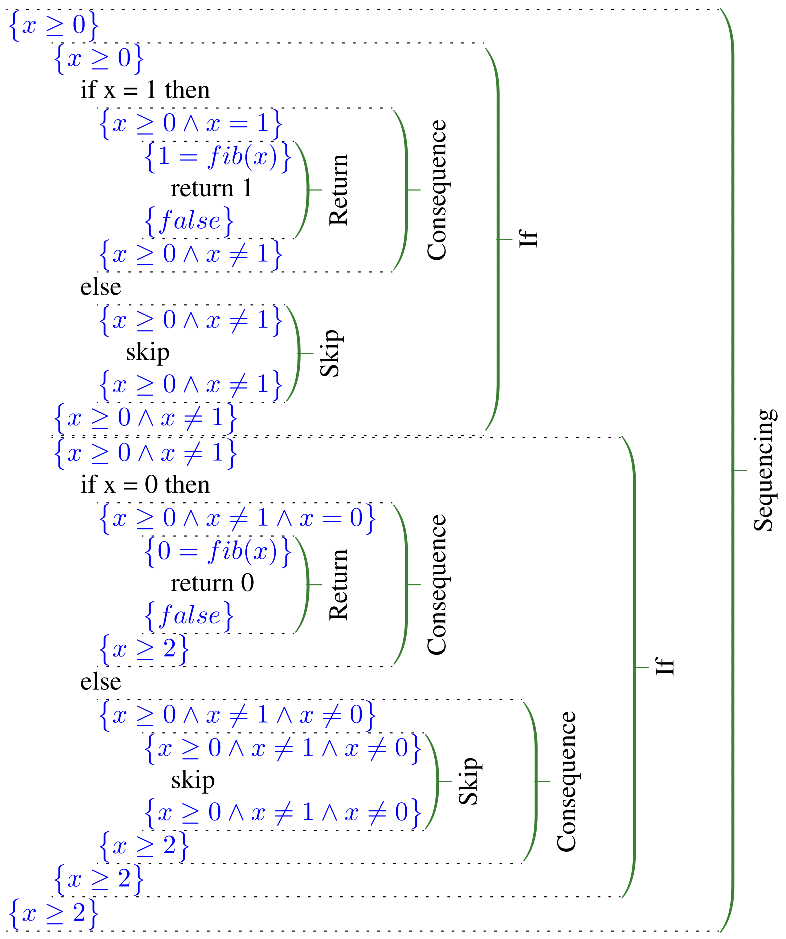
$$\{return = fib(x)\}return$$

and must verify the body. We can proceed as follows



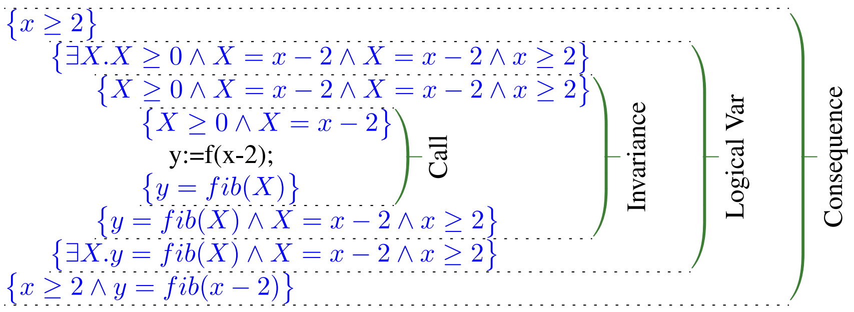
$\{x \geq 0\}$
$\{x \geq 0\}$
if x = 1 then return 1;
if x = 0 then return 0;
$\{x \geq 2\}$
$\{x \geq 2\}$
y:=f(x-2);
$\{x \geq 2 \wedge y = fib(x-2)\}$
$\{x \geq 2 \wedge y = fib(x-2)\}$
z:=f(x-1);
$\{y + z = fib(x)\}$
$\{y + z = fib(x)\}$
return y+z;
$\{false\}$
$\{false\}$

We are left with three proofs A, B and C. The first is



$\{x \geq 0\}$
$\{x \geq 0\}$
if x = 1 then
$\{x \geq 0 \wedge x = 1\}$
$\{1 = fib(x)\}$
return 1
$\{false\}$
$\{x \geq 0 \wedge x \neq 1\}$
else
$\{x \geq 0 \wedge x \neq 1\}$
skip
$\{x \geq 0 \wedge x \neq 1\}$
$\{x \geq 0 \wedge x \neq 1\}$
$\{x \geq 0 \wedge x \neq 1\}$
if x = 0 then
$\{x \geq 0 \wedge x \neq 1 \wedge x = 0\}$
$\{0 = fib(x)\}$
return 0
$\{false\}$
$\{x \geq 2\}$
else
$\{x \geq 0 \wedge x \neq 1 \wedge x \neq 0\}$
$\{x \geq 0 \wedge x \neq 1 \wedge x \neq 0\}$
skip
$\{x \geq 0 \wedge x \neq 1 \wedge x \neq 0\}$
$\{x \geq 2\}$
$\{x \geq 2\}$
$\{x \geq 2\}$

The second, B, is



$\{x \geq 2\}$

$\{\exists X.X \geq 0 \wedge X = x-2 \wedge X = x-2 \wedge x \geq 2\}$

$\{X \geq 0 \wedge X = x-2 \wedge X = x-2 \wedge x \geq 2\}$

$\{X \geq 0 \wedge X = x-2\}$

y:=f(x-2);

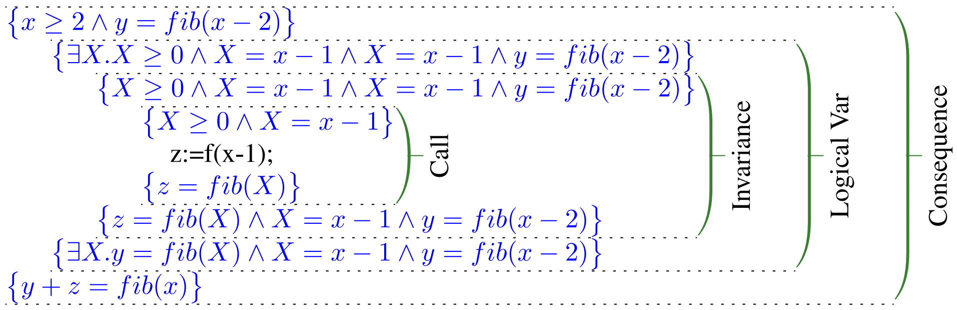
$\{y = fib(X)\}$

$\{y = fib(X) \wedge X = x-2 \wedge x \geq 2\}$

$\{\exists X.y = fib(X) \wedge X = x-2 \wedge x \geq 2\}$

$\{x \geq 2 \wedge y = fib(x-2)\}$

The third, C, is



$\{x \geq 2 \wedge y = fib(x-2)\}$

$\{\exists X.X \geq 0 \wedge X = x-1 \wedge X = x-1 \wedge y = fib(x-2)\}$

$\{X \geq 0 \wedge X = x-1 \wedge X = x-1 \wedge y = fib(x-2)\}$

$\{X \geq 0 \wedge X = x-1\}$

z:=f(x-1);

$\{z = fib(X)\}$

$\{z = fib(X) \wedge X = x-1 \wedge y = fib(x-2)\}$

$\{\exists X.y = fib(X) \wedge X = x-1 \wedge y = fib(x-2)\}$

$\{y + z = fib(x)\}$

Exercise 3.1. *Prove the two functions call rules are derivable from each other:*

$$\begin{array}{cl}
\Gamma, \{P\}f(X)\{Q\} \vdash \{P[X := E] \wedge y = Y\}y := f(E)\{(\exists y.Q[X := E] \wedge y = Y)[return := y]\} & \\
\hline
\Gamma, \{P\}f(X)\{Q\} \vdash \{P[X := E] \wedge y = Y \wedge X = (E[y := Y])\}y := f(E)\{(\exists y.Q[X := E] \wedge y = Y)[return := y] \wedge X = (E[y := Y])\} & \textit{Invariance} \\
\hline
\Gamma, \{P\}f(X)\{Q\} \vdash \{P \wedge X = E \wedge X = (E[y := Y]) \wedge y = Y\}y := f(E)\{Q[X := (E[y := Y])][return := y] \wedge X = (E[y := Y])\} & \textit{Conseq} \\
\hline
\Gamma, \{P\}f(X)\{Q\} \vdash \{\exists Y.\ P \wedge X = E \wedge X = (E[y := Y]) \wedge y = Y\}y := f(E)\{\exists Y.Q[X := (E[y := Y])][return := y] \wedge X = (E[y := Y])\} & \textit{VarElim} \\
\hline
\Gamma, \{P\}f(X)\{Q\} \vdash \{P \wedge X = E\}y := f(E)\{Q[return := y]\} & \textit{Conseq}
\end{array}$$

By condition on context we know free variables of P are X and of Q are $X, return$.

$$\begin{array}{l}
(\exists y.Q[X := E] \wedge y = Y)[return := y] \wedge X = (E[y := Y]) \\
\Rightarrow (\exists y.Q[X := E][y := Y] \wedge y = Y)[return := y] \wedge X = (E[y := Y]) \\
\Rightarrow Q[X := (E[y := Y])][return := y] \wedge X = (E[y := Y])
\end{array}$$

Follows as Q doesn't have y as a free variable.

$$\begin{array}{l}
\exists Y.Q[X := (E[y := Y])][return := y] \wedge X = (E[y := Y]) \\
\Rightarrow \exists Y.Q[return := y][X := (E[y := Y])] \wedge X = (E[y := Y]) \\
\Rightarrow \exists Y.Q[return := y] \wedge X = (E[y := Y]) \\
\Rightarrow \exists Y.Q[return := y] \\
\Rightarrow Q[return := y]
\end{array}$$

We have made use of the following logical facts

$$\begin{array}{l}
true \Rightarrow \exists X.\ X = E \\
X = E \wedge P \Leftrightarrow P[X := E] \wedge X = E
\end{array}$$

4 Heap - Encoding as an array

Exercise 4.1. *Prove createlist body meets its specification*

The definition of list in the notes is incorrect for this. It should have included two locations in the footprint for the list.

$$list(x, vs) = \begin{array}{l} x = null \wedge vs = \{\} \vee \\ \exists y.\ @heap[x] = y \wedge list(y, vs \setminus \{x, x+1\}) \wedge \{x, x+1\} \subseteq vs \end{array}$$

Assume return has specification

$$\exists vs.list(return, vs) \wedge \forall i \notin vs.@heap[i] = oldheap[i] \wedge \forall i \in vs.oldheap[i] = @unalloc$$

and create list has specification

$$\left.\begin{array}{l} \{@heap = oldheap\} \\ \quad \textit{create\_list()} \\ \{\exists vs.list(return, vs) \wedge \forall i \notin vs.@heap[i] = oldheap[i] \wedge \forall i \in vs.oldheap[i] = @unalloc\} \end{array}\right\} \textit{Assumption}$$

Prove

$$\left.\begin{array}{l} \{oldheap = @heap\} \\ \quad \textit{x := new;} \\ \quad \textit{if nondet() then} \\ \quad\quad \textit{y := create\_list ();} \\ \quad\quad \textit{[x] := y;} \\ \quad \textit{return x} \\ \{false\} \end{array}\right\} \textit{Goal}$$

We can give a proof outline as follows

$\{oldheap = @heap\}$

$\{oldheap = @heap\}$

x := new;

$$\left\{\begin{array}{l} @heap[x] = 0 \land @heap[x+1] = 0 \\ \land \forall y \in \{x, x+1\}.oldheap[y] = @unalloc \\ \land \forall y \notin \{x, x+1\}.oldheap[y] = @heap[y] \end{array}\right\}$$

A

$$\left\{\begin{array}{l} @heap[x] = 0 \land @heap[x+1] = 0 \\ \land \forall y \in \{x, x+1\}.oldheap[y] = @unalloc \\ \land \forall y \notin \{x, x+1\}.oldheap[y] = @heap[y] \end{array}\right\}$$

if nondet() then

$$\left\{\begin{array}{l} @heap[x] = 0 \land @heap[x+1] = 0 \\ \land \forall y \in \{x, x+1\}.oldheap[y] = @unalloc \\ \land \forall y \notin \{x, x+1\}.oldheap[y] = @heap[y] \end{array}\right\}$$

y := create_list ();

[x] := y;

$$\left\{\begin{array}{l} \exists vs.\ list(x, vs) \\ \land \forall y \in vs.oldheap[y] = @unallocc \\ \land \forall y \notin vs.oldheap[y] = @heap[y] \end{array}\right\}$$

B

else

$$\left\{\begin{array}{l} @heap[x] = 0 \land @heap[x+1] = 0 \\ \land \forall y \in \{x, x+1\}.oldheap[y] = @unalloc \\ \land \forall y \notin \{x, x+1\}.oldheap[y] = @heap[y] \end{array}\right\}$$

skip

$$\left\{\begin{array}{l} \exists vs.\ list(x, vs) \\ \land \forall y \in vs.oldheap[y] = @unalloc \\ \land \forall y \notin vs.oldheap[y] = @heap[y] \end{array}\right\}$$

Skip+Conseq

$$\left\{\begin{array}{l} \exists vs.\ list(x, vs) \\ \land \forall y \in vs.oldheap[y] = @unalloc \\ \land \forall y \notin vs.oldheap[y] = @heap[y] \end{array}\right\}$$

If

return x

$\{false\}$

Sequence

To prove (A),

$\{oldheap = @heap\}$

$$\left\{\begin{array}{l} \forall z.@heap[z] = @unalloc \land @heap[z+1] = @unalloc \\ \Rightarrow @heap\{z \leftarrow 0, z+1 \leftarrow 0\}[z] = 0 \land @heap\{z \leftarrow 0, z+1 \leftarrow 0\}[z+1] = 0 \\ \land \forall y \in \{z, z+1\}.oldheap[y] = @unalloc \\ \land \forall y \notin \{z, z+1\}.oldheap[y] = @heap\{z \leftarrow 0, z+1 \leftarrow 0\}[y] \end{array}\right\}$$

x := new;

$$\left\{\begin{array}{l} @heap[x] = 0 \land @heap[x+1] = 0 \\ \land \forall y \in \{x\}.oldheap[y] = @unalloc \land oldheap[y+1] = @unalloc \\ \land \forall y \notin \{x\}.oldheap[y] = @heap[y] \land oldheap[y+1] = @heap[y+1] \end{array}\right\}$$

New

$$\left\{\begin{array}{l} @heap[x] = 0 \land @heap[x+1] = 0 \\ \land \forall y \in \{x\}.oldheap[y] = @unalloc \land oldheap[y+1] = @unalloc \\ \land \forall y \notin \{x\}.oldheap[y] = @heap[y] \land oldheap[y+1] = @heap[y+1] \end{array}\right\}$$

Conseq

$$
\begin{array}{l}
oldheap = @heap \\
\Rightarrow\quad \forall y. oldheap[y] = @heap[y] \\
\Rightarrow\quad \forall z. @heap[z] = @unalloc \wedge @heap[z+1] = @unalloc \\
\qquad \Rightarrow oldheap[z] = @unalloc \wedge oldheap[z+1] = @unalloc \\
\qquad\qquad \wedge \forall y. oldheap[y] = @heap[y] \\
\Rightarrow\quad \forall z. @heap[z] = @unalloc \wedge @heap[z+1] = @unalloc \\
\qquad \Rightarrow 0 = 0 \wedge 0 = 0 \\
\qquad\qquad \wedge \forall y \in \{z, z+1\}. oldheap[y] = @unalloc \\
\qquad\qquad \wedge \forall y \notin \{z, z+1\}. oldheap[y] = @heap[y] \\
\Rightarrow\quad \forall z. @heap[z] = @unalloc \wedge @heap[z+1] = @unalloc \\
\qquad \Rightarrow @heap\{z \leftarrow 0, z+1 \leftarrow 0\}[z] = 0 \wedge @heap\{z \leftarrow 0, z+1 \leftarrow 0\}[z+1] = 0 \\
\qquad\qquad \wedge \forall y \in \{z, z+1\}. oldheap[y] = @unalloc \\
\qquad\qquad \wedge \forall y \notin \{z, z+1\}. oldheap[y] = @heap\{z \leftarrow 0, z+1 \leftarrow 0\}[y]
\end{array}
$$

To prove (B),

$$
\left\{\begin{array}{l}
@heap[x] = 0 \wedge @heap[x+1] = 0 \\
\wedge \forall y \in \{x, x+1\}. oldheap[y] = @unalloc \\
\wedge \forall y \notin \{x, x+1\}. oldheap[y] = @heap[y]
\end{array}\right\}
$$

$$
\left\{\begin{array}{l}
\exists oh2.\ oh2[x] = 0 \wedge oh2[x+1] = 0 \\
\wedge \forall y \in \{x, x+1\}. oldheap[y] = @unalloc \\
\wedge \forall y \notin \{x, x+1\}. oldheap[y] = oh2[y] \wedge \\
oh2 = @heap
\end{array}\right\}
$$

$$
\{oh2 = @heap\}
$$

y := create_list ();

$$
\left\{\begin{array}{l}
\exists vs.\ list(x, vs) \\
\wedge \forall y \in vs. oldheap[y] = @unalloc \\
\wedge \forall y \notin vs. oldheap[y] = @heap[y]
\end{array}\right\}
$$

Call Rule

$$
\left\{\begin{array}{l}
\exists oh2.\ oh2[x] = 0 \wedge oh2[x+1] = 0 \\
\wedge \forall y \in \{x, x+1\}. oldheap[y] = @unalloc \\
\wedge \forall y \notin \{x, x+1\}. oldheap[y] = oh2[y] \wedge \\
\exists vs.\ list(y, vs) \\
\wedge \forall w \in vs. oh2[w] = @unalloc \\
\wedge \forall w \notin vs. oh2[w] = @heap[w]
\end{array}\right\}
$$

Invariance+VarElim

$$
\left\{\begin{array}{l}
\exists vs.\ @heap[x] = 0 \wedge vs \cap \{x, x+1\} = \{\} \wedge list(y, vs) \\
\wedge \forall w \in vs \cup \{x, x+1\}. oldheap[w] = @unalloc \\
\wedge \forall w \notin vs \cup \{x, x+1\}. oldheap[w] = @heap[w]
\end{array}\right\}
$$

[x] := y;

$$
\left\{\begin{array}{l}
\exists vs.\ @heap[x] = y \wedge vs \cap \{x, x+1\} = \{\} \wedge list(y, vs) \\
\wedge \forall y \in vs \cup \{x, x+1\}. oldheap[y] = @unallocc \\
\wedge \forall y \notin vs \cup \{x, x+1\}. oldheap[y] = @heap[y]
\end{array}\right\}
$$

Assign + Conseq

$$
\left\{\begin{array}{l}
\exists vs.\ list(x, vs) \\
\wedge \forall y \in vs. oldheap[y] = @unallocc \\
\wedge \forall y \notin vs. oldheap[y] = @heap[y]
\end{array}\right\}
$$

Conseq + Seq + Conseq

You should justify the four implications used in this proof.