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 \( \text{???} \) 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

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

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

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

applying substitution, we can get:

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

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

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

We can eliminate the \( Y \) in the pre-condition

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

The pre-condition is implied by \( P \)

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

giving the forwards axiom.
(2) We can derive the backwards assignment axiom as

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

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

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

and dropping the conjunction, and applying the substitution gives

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

which simplifies to

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

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

\[
\begin{align*}
\{ x = 4 \} & \quad x := x + 1 & \{ x = 5 \} \\
\{ y = 3 \land x = 3 \} & \quad y := x + y & \{ y = 6 \land x = 3 \} \\
\{ \text{odd}(x) \} & \quad y := x + x + x & \{ \text{odd}(y) \land \text{odd}(x) \}
\end{align*}
\]

**Answers**

\[
\begin{align*}
& x = 4 \Rightarrow x + 1 = 5 \\
& \{ x + 1 = 5 \} x := x + 1 \{ x = 5 \} \quad \text{Assign} \quad x = 5 \Rightarrow x = 5 \quad \text{Conseq}
\end{align*}
\]

\[
\begin{align*}
& y = 3 \land x = 3 \Rightarrow x + y = 6 \land x = 3 \\
& \{ x + y = 6 \land x = 3 \} y := x + y \{ y = 6 \land x = 3 \} \quad \text{Assign} \quad \text{Conseq}
\end{align*}
\]

\[
\begin{align*}
& \text{odd}(x) \Rightarrow \text{odd}(x + x + x) \land \text{odd}(x) \\
& \{ \text{odd}(x + x + x) \land \text{odd}(x) \} y := x + x + x \{ \text{odd}(y) \land \text{odd}(x) \} \quad \text{Assign} \quad \text{Conseq}
\end{align*}
\]

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**

\[
\begin{align*}
& \{ x = 4 \} \\
& \{ x + 1 = 5 \} x := x + 1 \{ x = 5 \} \quad \text{Assign} \quad \text{Conseq}
\end{align*}
\]

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

\[
\begin{align*}
& \{ x = 4 \} \\
& \{ x + 1 = 5 \} x := x + 1 \{ x = 5 \} \quad \text{Consequence: } x = 4 \Rightarrow x + 1 = 5
\end{align*}
\]

\[
\begin{align*}
& \{ y = 3 \land x = 3 \} \\
& \{ x + y = 6 \land x = 3 \} y := x + y \{ y = 6 \land x = 3 \} \quad \text{Assign} \\
& \{ y = 6 \land x = 3 \} \quad \text{Consequence: } y = 3 \land x = 3 \Rightarrow x + y = 6 \land x = 3
\end{align*}
\]
Exercise 1.4. Show
\[
\frac{\{P\} C \{Q\}}{\exists X. P \ C \{\exists X. Q\}} \quad (1)
\]
is equi-expressive to
\[
\frac{\{P\} C \{Q\}}{\exists X. P \ C \{Q\}} \quad (2)
\]
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
\[
\frac{\{P\} C \{Q\}}{\exists X. P \ C \{Q\}} \quad (1) \quad \exists X. Q \Rightarrow Q \quad \text{Conseq}
\]

For the other direction. Note that it is always true that
\[Q \Rightarrow \exists X. Q\]
So,
\[
\frac{\{P\} C \{Q\}}{\exists X. P \ C \{\exists X. Q\}} \quad \text{Conseq}
\]
\[
\frac{\{P\} C \{Q\}}{\exists X. P \ C \{\exists X. Q\}} \quad (2)
\]

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

\[
\begin{align*}
\vdash & \{P\} C_1 \{R\} \\
\vdash & \{R\} C_2 \{Q\} \\
\sigma & \vdash P \\
C_1; C_2, \sigma & \to^* \text{skip}, \sigma'
\end{align*}
\]

Prove

\[
\sigma' \vdash Q
\]

We can prove that, if

\[
C_1; C_2, \sigma \to^* \text{skip}, \sigma'
\]

then, there exists \( \sigma'' \) such that

\[
\begin{align*}
C_1, \sigma & \to^* \text{skip}, \sigma'' \\
C_2, \sigma'' & \to^* \text{skip}, \sigma'
\end{align*}
\]

by induction on the operational semantics.

Therefore, we can use our first inductive hypothesis: \( \vdash \{P\} C_1 \{R\} \), to prove \( \sigma'' \vdash R \), and hence we can use the second inductive hypothesis to prove \( \sigma' \vdash Q \) as required.

If  Assume

\[
\begin{align*}
\vdash & \{P \land B\} C_1 \{Q\} \\
\vdash & \{P \land \neg B\} C_2 \{Q\} \\
\sigma & \vdash P \\
\text{if } B \text{ then } C_1 \text{ else } C_2, \sigma & \to^* \text{skip}, \sigma'
\end{align*}
\]

Prove

\[
\sigma' \vdash Q
\]

Case split on \( \llbracket B \rrbracket_\sigma = \text{true} \) or \( \llbracket B \rrbracket_\sigma = \text{false} \):

First case, assume \( \llbracket B \rrbracket_\sigma = \text{true} \). Therefore, it must reduce as

\[
\text{if } B \text{ then } C_1 \text{ else } C_2, \sigma \quad \to \quad C_1, \sigma \quad \to^* \text{skip}, \sigma'
\]

We know \( \sigma \vdash P \land B \), therefore by inductive hypothesis, we know \( \sigma' \vdash Q \).

Second case, similar.
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

\[ Y^Z = x \cdot y^2 \land z \leq 0 \Rightarrow x = Y^Z \]

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

\[ Y^Z = x \cdot y^2 \land z \geq 0 \land z \leq 0 \Rightarrow x = Y^Z \]

which is valid.

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

\[ y = Y \land z = Z \land z \geq 0 \]

We can verify the loop as

\[
\begin{align*}
\{ Y^Z = x \cdot y^2 \land z \geq 0 \land z \geq 0 \} \\
\text{if } z \& l = 1 \text{ then} \\
\{ Y^Z = x \cdot y^2 \land z \geq 0 \land z \geq 0 \} & \quad \text{Assign} \\
\{ Y^Z = x \cdot y^2 \land z \geq 0 \land z \geq 0 \} & \quad \text{Consequence (A)} \\
\{ Y^Z = x \cdot y^2 \land z \geq 0 \land z \geq 0 \} & \quad \text{Sequencing} \\
\text{else} \\
\{ z \& l = 0 \land Y^Z = x \cdot y^2 \land z \geq 0 \} & \quad \text{Skip} \\
\{ Y^Z = x \cdot y^2 \land z \geq 0 \land z \geq 0 \} & \quad \text{Consequence (B)} \\
\{ Y^Z = x \cdot y^2 \land z \geq 0 \land z \geq 0 \} & \quad \text{Assign} \\
\{ z := z \& l \} & \\
\{ Y^Z = x \cdot (y \cdot y)^{z \gg 0} \land z \geq 0 \} & \quad \text{Assign} \\
\{ y := y \cdot y \} & \\
\{ Y^Z = x \cdot y^2 \land z \geq 0 \} & \\
\end{align*}
\]

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

\[
Y^Z = x \cdot y^2 \land z \geq 0 \land z \geq 0 \land z \& l = 1 \\
\Rightarrow Y^Z = x \cdot y^{2(z \gg 1) + 1} \land z \gg 1 \geq 0 \\
\Rightarrow Y^Z = x \cdot y \cdot (y \cdot y)^{z \gg 1} \land z \gg 1 \geq 0
\]

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

\[
z \& l = 0 \land Y^Z = x \cdot y^2 \land z \geq 0 \\
\Rightarrow Y^Z = x \cdot y^{2(z \gg 1)} \land z \gg 1 \geq 0 \\
\Rightarrow Y^Z = x \cdot (y \cdot y)^{z \gg 1} \land z \gg 1 \geq 0
\]
Exercise 2.2 (Prime Palindrome).

Answer The proof proceeds as follows:

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 : \( i > 100 \Rightarrow \forall j. 20 \leq j \leq 100 \Rightarrow \neg \text{prime}(j) \lor \neg \text{palindrome}(j) \)

\( \wedge 20 \leq i \leq 100 \Rightarrow \text{prime}(i) \wedge \text{palindrome}(i) \)

and continue : \( i \geq 19 \land (\forall j. 20 \leq j \leq i \Rightarrow \neg \text{prime}(j) \land \neg \text{palindrome}(j)) \)
Exercise 2.3. Give a rule for do C while B.

\[
\{ C \mid (B \land \neg C) \} \quad C \quad \{ B \}
\]

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
\[
\epsilon[\text{while } B \text{ do } C], \sigma \rightarrow \epsilon[C; \text{while } B \text{ do } C], \sigma \quad \text{where } [B]_\sigma = \text{true}
\]
\[
\epsilon[\text{while } B \text{ do } C], \sigma \rightarrow \epsilon[\text{skip}], \sigma \quad \text{where } [B]_\sigma = \text{false}
\]

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
\[
\sigma \models P \implies \forall m \leq n. C, \sigma \rightarrow^m \text{skip}, \sigma' \implies \sigma' \models P
\]

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
\[
\models \{P \land B\}C\{P\}
\]
\[
P \land \neg B \Rightarrow Q
\]
\[
\forall m < n. \models_m \{P\} \text{ while } B \text{ do } C\{Q\}
\]

and must prove
\[
\models_n \{P\} \text{ while } B \text{ do } C\{Q\}
\]

Hence, we can assume
\[
\sigma \models P
\]
\[
\text{while } B \text{ do } C, \sigma \rightarrow^m \text{skip}, \sigma'
\]
\[
m \leq n
\]

and must prove
\[
\sigma' \models Q
\]

There are two cases for the reduction of \(\text{while } B \text{ do } C, \sigma\) either \(B\) evaluates to true or false. First case, assume \(B\) is false in \(\sigma\), therefore
\[
\text{while } B \text{ do } C, \sigma \rightarrow \text{skip}, \sigma
\]

Hence, \(\sigma = \sigma'\). By our assumption \(P \land \neg B \Rightarrow Q\), we know that \(\sigma'\) satisfies \(Q\) as required. Second case, assume \(B\) evaluates to true in \(\sigma\). Therefore, we know
\[
\sigma \models P \land B
\]
\[
C, \sigma \rightarrow^i \text{skip}, \sigma''
\]

By assumption from premise, \(\models \{P \land B\}C\{P\}\), we know:
\[
\sigma'' \models P
\]

Hence,
\[
\text{while } B \text{ do } C, \sigma
\]
\[
\rightarrow C; \text{while } B \text{ do } C, \sigma
\]
\[
\rightarrow^i \text{skip}; \text{while } B \text{ do } C, \sigma''
\]
\[
\rightarrow \text{while } B \text{ do } C, \sigma''
\]
\[
\rightarrow^m-i-2 \text{ while } B \text{ do } C, \sigma'
\]

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

**Exercise 2.5.** Prove the soundness of while rule with break and continue, that is, prove
\[
(break : P, continue : Q \models \{Q \land B\}C\{Q\}) \land (Q \land \neg B \Rightarrow P) \land (\forall m < n. break : P', continue : Q' \models_m \{Q\} \text{ while } B \text{ do } C\{P\}) \implies (break : P', continue : Q' \models_n \{Q\} \text{ while } B \text{ do } C\{P\})
\]
Answer  
Assume 

\[
(break : P, continue : Q \models (Q \land B)C\{Q\})
\]

\[
(Q \land \neg B \Rightarrow P)
\]

\[
\sigma \models Q
\]

Prove \(\forall m \leq n\).

1. \(\text{while } B \text{ do } C, [] , \sigma \rightarrow^m e[break], [], \sigma' \implies \sigma' \models P'\)
2. \(\text{while } B \text{ do } C, [], \sigma \rightarrow^m e[continue], [], \sigma' \implies \sigma' \models Q'\)
3. \(\text{while } B \text{ do } C, [], \sigma \rightarrow^m \text{skip}, [], \sigma' \implies \sigma' \models P\)

We can prove that

\(\text{while } B \text{ do } C, [], \sigma\)

cannot reduce to

\(e[break], [], \sigma'\)

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

Hence, we can assume

\(\text{while } B \text{ do } C, [], \sigma \rightarrow^m \text{skip}, [], \sigma'\)

and must prove

\(\sigma' \models P\)

From the semantics, we know that

\(\text{while } B \text{ do } C, [], \sigma \rightarrow C; \text{if } \neg B \text{ then break else continue}, \text{while } B \text{ do } C, \sigma\)

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

\(\text{while } B \text{ do } C, [], \sigma \rightarrow C; \text{if } \neg B \text{ then break else continue}, \text{while } B \text{ do } C, \sigma\)

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

Assume \(C, [], \sigma \rightarrow^* e[continue], [], \sigma'', \therefore\)

\(\text{while } B \text{ do } C, [], \sigma \rightarrow C; \text{if } \neg B \text{ then break else continue}, \text{while } B \text{ do } C, \sigma\)

By assumption, with \((break : P, continue : Q \models (Q \land B)C\{Q\})\), 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
\{ \text{return} = \text{fib}(x) \}\text{return}

and must verify the body. We can proceed as follows

\[
\begin{array}{l}
\{ x \geq 0 \} \\
\{ x \geq 0 \} \\
\text{if } x = 1 \text{ then return } 1; \\
\text{if } x = 0 \text{ then return } 0; \\
\{ x \geq 2 \} \\
\{ x \geq 2 \} \\
y := \text{f}(x-2); \\
\{ x \geq 2 \land y = \text{fib}(x-2) \} \\
\{ x \geq 2 \land y = \text{fib}(x-2) \} \\
z := \text{f}(x-1); \\
\{ y+z = \text{fib}(x) \} \\
\{ y+z = \text{fib}(x) \} \\
\text{return } y+z;
\end{array}
\]

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

\[
\begin{array}{l}
\{ x \geq 0 \} \\
\{ x \geq 0 \} \\
\text{if } x = 1 \text{ then } \\
\{ x \geq 0 \land x = 1 \} \\
\{ 1 = \text{fib}(x) \} \\
\text{return } 1 \\
\{ false \} \\
\{ x \geq 0 \land x \neq 1 \} \\
\text{else} \\
\{ x \geq 0 \land x \neq 1 \} \\
\text{skip} \\
\{ x \geq 0 \land x \neq 1 \} \\
\{ x \geq 0 \land x \neq 1 \} \\
\text{if } x = 0 \text{ then } \\
\{ x \geq 0 \land x \neq 1 \land x = 0 \} \\
\{ 0 = \text{fib}(x) \} \\
\text{return } 0 \\
\{ false \} \\
\{ x \geq 2 \} \\
\text{else} \\
\{ x \geq 0 \land x \neq 1 \land x \neq 0 \} \\
\text{skip} \\
\{ x \geq 0 \land x \neq 1 \land x \neq 0 \} \\
\{ x \geq 0 \land x \neq 1 \land x \neq 0 \} \\
\text{if } x \geq 2 \text{ then } \\
\{ x \geq 0 \land x \neq 1 \land x \neq 0 \} \\
\text{else}
\end{array}
\]
The second, B, is

\[
\begin{array}{l}
\{x \geq 2\} \\
\{\exists X. x \geq 0 \land X = x - 2 \land X = x - 2 \land x \geq 2\} \\
\{X \geq 0 \land X = x - 2 \land X = x - 2 \land x \geq 2\} \\
\{x \geq 0 \land X = x - 2\}
\end{array}
\]

By condition on context we know free variables of

\[
\{y = \text{fib}(x)\}
\]

\[
\{y = \text{fib}(x)\}
\]

\[
\{y = \text{fib}(x)\}
\]

\[
\{x \geq 2 \land y = \text{fib}(x - 2)\}
\]

The third, C, is

\[
\begin{array}{l}
\{x \geq 2 \land y = \text{fib}(x - 2)\} \\
\{\exists X. x \geq 0 \land X = x - 1 \land X = x - 1 \land y = \text{fib}(x - 2)\} \\
\{X \geq 0 \land X = x - 1 \land X = x - 1 \land y = \text{fib}(x - 2)\} \\
\{X \geq 0 \land X = x - 1\}
\end{array}
\]

\[
\begin{array}{l}
z = \text{fib}(x - 1); \\
\{z = \text{fib}(x)\}
\end{array}
\]

\[
\begin{array}{l}
\{z = \text{fib}(x)\}
\end{array}
\]

\[
\begin{array}{l}
\{\exists X. x \geq 0 \land X = x - 1 \land y = \text{fib}(x - 2)\}
\end{array}
\]

\[
\begin{array}{l}
\{\exists X. y = \text{fib}(X) \land X = x - 2 \land x \geq 2\}
\end{array}
\]

\[
\begin{array}{l}
\{x \geq 2 \land y = \text{fib}(x - 2)\}
\end{array}
\]

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

\[
\begin{array}{l}
\Gamma, \{P\} f(X)\{Q\} \vdash \{P[X := E] \land y = Y\} y := f(E)(\exists y. Q[X := E] \land y = Y)(\text{return} := y)
\end{array}
\]

\[
\begin{array}{l}
\Gamma, \{P\} f(X)\{Q\} \vdash \{P[X := E] \land y = Y \land X = (E[y := Y])\} y := f(E)(\exists y. Q[X := E] \land y = Y)(\text{return} := y \land X = (E[y := Y]))
\end{array}
\]

\[
\begin{array}{l}
\Gamma, \{P\} f(X)\{Q\} \vdash \{P \land X = E \land X = (E[y := Y]) \land y = Y\} y := f(E)(\exists y. Q[X := E] \land y = Y)(\text{return} := y \land X = (E[y := Y]))
\end{array}
\]

\[
\begin{array}{l}
\Gamma, \{P\} f(X)\{Q\} \vdash \{P \land X = E\} y := f(E)(Q[\text{return} := y])
\end{array}
\]

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

\[
(\exists y. Q[X := E] \land y = Y)(\text{return} := y) \land X = (E[y := Y])
\]

\[
\Rightarrow (\exists y. Q[X := E][y := Y] \land y = Y)(\text{return} := y) \land X = (E[y := Y])
\]

\[
\Rightarrow Q[X := (E[y := Y])](\text{return} := y) \land X = (E[y := Y])
\]

Follows as \(Q\) doesn’t have \(y\) as a free variable.

\[
\exists Y. Q[X := (E[y := Y])](\text{return} := y) \land X = (E[y := Y])
\]

\[
\Rightarrow \exists Y. Q[\text{return} := y][X := (E[y := Y]) \land X = (E[y := Y])]
\]

\[
\Rightarrow \exists Y. Q[\text{return} := y] \land X = (E[y := Y])
\]

\[
\Rightarrow Q[\text{return} := y]
\]

\[
\text{We have made use of the following logical facts}
\]

\[
\text{true} \Rightarrow \exists X. X = E
\]

\[
X = E \land P \iff P[X := E] \land X = E
\]

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.

\[
\text{list}(x, vs) = x = \text{null} \land vs = \{\} \lor \\
\exists y. \text{@heap}[x] = y \land \text{list}(y, vs \setminus \{x, x + 1\}) \land \{x, x + 1\} \subseteq vs
\]

Assume return has specification

\[
\exists vs. \text{list}(\text{return}, vs) \land \forall i /\in vs. \text{@heap}[i] = \text{oldheap}[i] \land \forall i \in vs. \text{oldheap}[i] = \text{unalloc}
\]

and create list has specification

\[
\begin{cases}
\text{@heap} = \text{oldheap}
\end{cases}
\]

create_list()

\[
\begin{cases}
\exists vs. \text{list}(\text{return}, vs) \land \forall i /\in vs. \text{@heap}[i] = \text{oldheap}[i] \land \forall i \in vs. \text{oldheap}[i] = \text{unalloc}
\end{cases}
\]

Goal

\[
\begin{cases}
\text{oldheap} = \text{@heap}
\end{cases}
\]

\[
\begin{cases}
x := \text{new}; \\
\text{if nondet()} \text{ then} \\
y := \text{create_list ();} \\
[x] := y; \\
\text{return } x
\end{cases}
\]

false

We can give a proof outline as follows
To prove (A),

\[
\begin{align*}
\{ \text{oldheap = @heap} \} \\
\text{x := new;} \\
\{ \text{@heap[x] = 0 \land @heap[x+1] = 0} \\
\forall y \in \{x, x + 1\}. \text{oldheap[y] = @unalloc} \\
\forall y \notin \{x, x + 1\}. \text{oldheap[y] = @heap[y]} \\
\{ \text{@heap[x] = 0 \land @heap[x+1] = 0} \} \\
\end{align*}
\]

A

\[
\begin{align*}
\text{if nondet()} \text{ then} \\
\{ \text{@heap[x] = 0 \land @heap[x+1] = 0} \\
\forall y \in \{x, x + 1\}. \text{oldheap[y] = @unalloc} \\
\forall y \notin \{x, x + 1\}. \text{oldheap[y] = @heap[y]} \\
y := \text{create list}(); \\
[x] := y; \\
\exists v. \text{list}(x, vs) \\
\forall y \in vs. \text{oldheap}[y] = @unalloc \\
\forall y \notin vs. \text{oldheap}[y] = @heap[y] \\
\{ \text{false} \} \\
\end{align*}
\]

B

\[
\begin{align*}
\text{else} \\
\{ \text{@heap[x] = 0 \land @heap[x+1] = 0} \\
\forall y \in \{x, x + 1\}. \text{oldheap[y] = @unalloc} \\
\forall y \notin \{x, x + 1\}. \text{oldheap[y] = @heap[y]} \\
\exists v. \text{list}(x, vs) \\
\forall y \in vs. \text{oldheap}[y] = @unalloc \\
\forall y \notin vs. \text{oldheap}[y] = @heap[y] \\
\{ \text{false} \} \\
\end{align*}
\]

\[
\begin{align*}
\text{return x} \\
\{ \text{false} \} \\
\end{align*}
\]
\[ \text{To prove (B),} \]

\[
\begin{align*}
&\text{If } \forall y. \text{oldheap}[y] = \text{@heap}[y], \\
&\text{then } \forall z. \text{@heap}[z] = \text{@unalloc} \land \text{oldheap}[z + 1] = \text{@unalloc}, \\
&\text{then } \text{oldheap}[z] = \text{@unalloc} \land \text{oldheap}[z + 1] = \text{@unalloc}, \\
&\text{then } \forall y. \text{oldheap}[y] = \text{@heap}[y]. \\
&\text{Therefore, } \forall z. \text{@heap}[z] = \text{@unalloc} \land \text{oldheap}[z + 1] = \text{@unalloc} \\
&\text{then } 0 = 0 \land 0 = 0 \\
&\text{Result: } \forall y \in \{z, z + 1\}, \text{oldheap}[y] = \text{@unalloc} \\
&\text{and } \forall y \notin \{z, z + 1\}, \text{oldheap}[y] = \text{@heap}[y]. \\
&\text{Moreover, } \forall y \in \{z, z + 1\}, \text{oldheap}[y] = \text{@unalloc} \\
&\text{and } \forall y \notin \{z, z + 1\}, \text{oldheap}[y] = \text{@heap}[z \leftarrow 0, z + 1 \leftarrow 0][y].
\end{align*}
\]

\begin{align*}
&\text{Call Rule} \\
&\text{Invariance + Var Elim} \\
&\text{Seq + Conseq} \\
&\text{Assign + Conseq}
\end{align*}

You should justify the four implications used in this proof.