DD2454 Semantics of Programming Languages
– Additional Exercises –

Dilian Gurov
Royal Institute of Technology – KTH
e–mail: dilian@csc.kth.se

1 Operational Semantics of IMP

1. Consider the extension of **IMP** with a command **read** X that waits for a number to be entered (say, from a keyboard), which is then assigned to location X.

   Use the small–step operational semantics of **IMP** extended in this way (see lecture notes) to derive the configuration space of the following program, starting from an arbitrary initial state σ and assuming that only 0's and 1's are entered (the program then maintains in location X the sum modulo 2 of all values entered so far).

   $X := 0;$
   while true do
   read Y;
   if $Y = 1$ **then** $X := 1 - X$ **else skip**

   Draw the configuration graph (as symmetrically as possible). Clearly identify the derivation of every transition in the graph. You may omit duplicating or very similar derivations.

2. Consider the small–step operational semantics of **IMP** (see handouts). Let us extend the boolean expressions of the language with the atomic (randomized) boolean expression **brand**, which evaluates to both **true** and **false**. So, we extend the semantics with the two new axioms $\langle \textbf{brand}, \sigma \rangle \to \textbf{true}$ and $\langle \textbf{brand}, \sigma \rangle \to \textbf{false}$. Now, consider the program:

   while true do
   if brand then $X := 1$
   else $X := 0$

   Derive the configuration space of the program, that is, execute the program under the small-step semantics, starting from a state σ such that $\sigma(X) = 0$. Draw the configuration graph.

   Solution: Applying the rules of the small–step operational semantics, and taking into account that $\sigma = \sigma[0/X]$ when $\sigma(X) = 0$, we can derive 10 transitions between 8 configurations, $\langle c, \sigma[0/X] \rangle$ being the initial configuration:

   $\langle c, \sigma[0/X] \rangle \to_S \langle \textbf{if brand then } X := 1 \textbf{ else } X := 0; c, \sigma[0/X] \rangle$
   $\langle \textbf{if brand then } X := 1 \textbf{ else } X := 0; c, \sigma[0/X] \rangle \to_S \langle X := 1; c, \sigma[0/X] \rangle$
   $\langle \textbf{if brand then } X := 1 \textbf{ else } X := 0; c, \sigma[0/X] \rangle \to_S \langle X := 0; c, \sigma[0/X] \rangle$
   $\langle X := 0; c, \sigma[0/X] \rangle \to_S \langle c, \sigma[0/X] \rangle$
   $\langle X := 1; c, \sigma[0/X] \rangle \to_S \langle c, \sigma[1/X] \rangle$
   $\langle c, \sigma[1/X] \rangle \to_S \langle \textbf{if brand then } X := 1 \textbf{ else } X := 0; c, \sigma[1/X] \rangle$
   $\langle \textbf{if brand then } X := 1 \textbf{ else } X := 0; c, \sigma[1/X] \rangle \to_S \langle X := 1; c, \sigma[1/X] \rangle$
   $\langle \textbf{if brand then } X := 1 \textbf{ else } X := 0; c, \sigma[1/X] \rangle \to_S \langle X := 0; c, \sigma[1/X] \rangle$
   $\langle X := 0; c, \sigma[1/X] \rangle \to_S \langle c, \sigma[0/X] \rangle$
   $\langle X := 1; c, \sigma[1/X] \rangle \to_S \langle c, \sigma[1/X] \rangle$

For instance, here is a derivation tree for the second transition:

$$\dfrac{\dfrac{\dfrac{-}{\langle \mathbf{brand}, \sigma\rangle \to \mathbf{true}}}{\langle \mathbf{if\ brand\ then}\ X := 1\ \mathbf{else}\ X := 0, \sigma[0/X]\rangle \to_S \langle X := 1, \sigma[0/X]\rangle}}{\langle \mathbf{if\ brand\ then}\ X := 1\ \mathbf{else}\ X := 0; c, \sigma[0/X]\rangle \to_S \langle X := 1; c, \sigma[0/X]\rangle}$$

The configuration graph is simply a graph with the 8 configurations as nodes, and the 10 transitions as arcs between the corresponding nodes.

3. Let us extend the simple imperative programming language **IMP** with *threads* by adding the statement **thread** c **end**. The intended behaviour of this statement is that it generates a new thread executing command c. Multiple threads are executed non-deterministically: At any point of an execution, any of the threads can become active (that is, be scheduled for execution).

   (a) Give an abstract machine semantics (see lecture notes) for **IMP** with threads. Configurations will now have multisets Γ of stacks of commands (one stack per thread), and transitions will have the shape $\langle \Gamma, \sigma\rangle \to_{AM} \langle \Gamma', \sigma'\rangle$. Configurations $\langle \emptyset, \sigma\rangle$ can be abbreviated as σ. If we take a formal-sum notation for multisets (where, for example, $2a+3b$ denotes the multiset $\{a, a, b, b, b\}$), we could give (for example) the following axiom for **skip**:

$$\langle (\mathbf{skip} \cdot \gamma) + \Gamma, \sigma\rangle \to_{AM} \langle \gamma + \Gamma, \sigma\rangle$$

   Solution: The rules are almost identical to the ones presented in class for **IMP** without threads, but, as in the case for **skip** given above, have an additional $+\Gamma$ component in the configurations. The only interesting rule is the (new) rule for **thread** c **end**:

$$\langle (\mathbf{thread}\ c\ \mathbf{end} \cdot \gamma) + \Gamma, \sigma\rangle \to_{AM} \langle c + \gamma + \Gamma, \sigma\rangle$$

   (b) Use your semantics to execute the program **thread** $X := 0$ **end**; $X := 1$ starting from an arbitrary initial state σ. Clearly identify the rules used in the derivation.

   Solution: The execution can be presented schematically as:

$$\begin{array}{ll} & \langle \mathbf{thread}\ X := 0\ \mathbf{end}; X := 1, \sigma\rangle \\ \to_{AM} & \langle \mathbf{thread}\ X := 0\ \mathbf{end}\ \cdot X := 1, \sigma\rangle \\ \to_{AM} & \langle (X := 0) + (X := 1), \sigma\rangle \\ & \to_{AM} \langle X := 1, \sigma[0/X]\rangle \to_{AM} \sigma[1/X] \\ & \to_{AM} \langle X := 0, \sigma[1/X]\rangle \to_{AM} \sigma[0/X] \end{array}$$

   Notice the non-deterministic branching at the (third) configuration $\langle (X := 0) + (X := 1), \sigma\rangle$!

4. Consider the **IMP** program c:

$$\mathbf{while}\ \neg(X = Y)\ \mathbf{do}\ (X := X + 1; Y := Y - 1)$$

   Use the big-step operational semantics of **IMP** to prove that the program *terminates* for all (initial) states in $S \stackrel{def}{=} \{\sigma \in \Sigma \mid \exists k \geq 0.\, \sigma(Y) = \sigma(X) + 2k\}$.

   Solution: Define $\prec \subseteq S \times S$ as follows:

$$\sigma \prec \sigma' \stackrel{def}{\Leftrightarrow} \sigma(Y) - \sigma(X) < \sigma'(Y) - \sigma'(X)$$

   Since $\sigma(Y) - \sigma(X) \geq 0$ for all $\sigma \in \Sigma$, $\prec$ is well-founded. We use well-founded induction to prove $\forall \sigma \in S.\, \exists \sigma' \in \Sigma.\ \Vdash \langle c, \sigma\rangle \to \sigma'$.

   Let $\sigma \in S$, and let $\exists \sigma' \in \Sigma.\ \Vdash \langle c, \sigma''\rangle \to \sigma'$ hold for all $\sigma'' \in S$ such that $\sigma'' \prec \sigma$ (induction hypothesis).

   Case 1: $\sigma(X) = \sigma(Y)$. It is straightforward to produce a direct derivation of $\langle c, \sigma\rangle \to \sigma$, with last rule applied $\mathbf{while}_F$, and hence $\exists \sigma' \in \Sigma.\ \Vdash \langle c, \sigma\rangle \to \sigma'$.

   Case 2: $\sigma(X) \neq \sigma(Y)$. Again, we construct a derivation of $\langle c, \sigma\rangle \to \sigma'$, with last rule applied $\mathbf{while}_T$. The sub-derivations of the first two premises $\langle \neg(X = Y), \sigma\rangle \to \mathbf{true}$ and $\langle X := X + 1; Y := Y - 1, \sigma\rangle \to$

$\sigma[\sigma(X)+1/X, \sigma(Y)-1/Y]$ are easy to construct. The existence of a sub-derivation for the third subgoal $\langle c, \sigma[\sigma(X)+1/X, \sigma(Y)-1/Y]\rangle \to \sigma'$ is guaranteed (for some σ'!) by the induction hypothesis, since $\sigma[\sigma(X)+1/X, \sigma(Y)-1/Y] \in S$ and $\sigma[\sigma(X)+1/X, \sigma(Y)-1/Y] \prec \sigma$ whenever $\sigma \in S$. Therefore $\exists \sigma' \in \Sigma.\ \Vdash \langle c, \sigma\rangle \to \sigma'$.

5. Consider the following **IMP** program c:

   while $\neg(I \leq 0)$ **do**
   if $C \leq I$ **then**
   $I := I - C$
   else
   $I := I - 1$

   Prove termination of the program for all initial states in $S \stackrel{def}{=} \{\sigma \in \Sigma \mid \sigma(C) \geq 1\}$ by showing that for every state $\sigma \in S$ there is a state $\sigma' \in \Sigma$ such that $\Vdash \langle c, \sigma\rangle \to \sigma'$.
   Hint: use well–founded induction on S.

6. Consider the big-step operational semantics of **IMP**.

   (a) Show that $\Vdash \langle$**while** b **do** $c, \sigma\rangle \to \sigma'$ implies $\Vdash \langle b, \sigma'\rangle \to$ **false**. You will need to use a special kind of induction here, namely (mathematical) induction on the *depth* of the derivation trees for transitions $\langle$**while** b **do** $c, \sigma\rangle \to \sigma'$. (That is, you assume that $\Vdash \langle b, \sigma'\rangle \to$ **false** holds whenever a transition of the shape $\langle$**while** b **do** $c, \sigma\rangle \to \sigma'$ is derivable with a derivation tree of depth n, and you prove that then this also holds for $n+1$.)

   Solution: In the base case $n = 0$ the result holds vacuously, since there are no derivations of depth 0 (that is, axioms) for transitions of the shape $\langle$**while** b **do** $c, \sigma\rangle \to \sigma'$. For the induction case, assume that for all $\sigma, \sigma' \in \Sigma$, $\Vdash \langle b, \sigma'\rangle \to$ **false** holds whenever $\langle$**while** b **do** $c, \sigma\rangle \to \sigma'$ is derivable with a derivation tree with depth n (induction hypothesis). Assume $\langle$**while** b **do** $c, \sigma\rangle \to \sigma'$ is derivable with a derivation tree with depth $n+1$. (We show that $\Vdash \langle b, \sigma'\rangle \to$ **false**.) We consider the two possible cases for the last rule applied in the derivation of $\langle$**while** b **do** $c, \sigma\rangle \to \sigma'$.

   Case 1: Last rule is **while**$_F$. Then $\Vdash \langle b, \sigma\rangle \to$ **false** and $\sigma' = \sigma$, and hence $\Vdash \langle b, \sigma'\rangle \to$ **false**.

   Case 2: Last rule is **while**$_T$. Then, for some σ'', $\langle$**while** b **do** $c, \sigma''\rangle \to \sigma'$ is derivable with a derivation of depth n. By the induction hypothesis, $\Vdash \langle b, \sigma'\rangle \to$ **false**.

   (b) Consider the transformation on **IMP** programs from program **while** b **do** (**while** b **do** c) to program **while** b **do** c. Show that the transformation is a semantics-preserving *optimization* by proving that the two programs are *equivalent*.

   Solution: The proof is standard, by transforming every derivation of $\langle$**while** b **do** (**while** b **do** c), $\sigma\rangle \to \sigma'$ to a derivation of $\langle$**while** b **do** $c, \sigma\rangle \to \sigma'$, and *vice versa*, by using (a).

2 Denotational Semantics of IMP

1. Consider the **IMP** program for computing powers:

$P := 1;$
while $\neg(Y = 0)$ **do**
$P := P \times X;$
$Y := Y - 1$

Use the fixed–point characterization of the denotational semantics of while–loops of **IMP** and the Fixed–Point Theorem to iteratively compute the denotation of the above program:

(a) Determine the transformer Γ for the while loop. Simplify it as much as possible.

(b) Use Γ to compute the first three (non–empty) approximants of the fixed–point computation.

(c) Guess the general shape of the i–th approximant.

(d) Use this to obtain the limit value (which is the denotation of the while loop).

(e) Compute the denotation of the whole program.

Solution:

(a) Let w stand for the while–loop in the program. Then $\mathcal{C}[\![w]\!] = \mathit{fix}\,\Gamma$, where Γ is defined by:

$$\begin{aligned}\Gamma(F) = &\ \{(\sigma,\sigma') \mid \mathcal{B}[\![\neg(Y=0)]\!]\,\sigma = \mathit{true} \wedge (\sigma,\sigma') \in F \circ \mathcal{C}[\![P := P \times X; Y := Y - 1]\!]\}\\ \cup &\ \{(\sigma,\sigma) \mid \mathcal{B}[\![\neg(Y=0)]\!]\,\sigma = \mathit{false}\}\\ = &\ \{(\sigma,\sigma') \mid \sigma(Y) \neq 0 \wedge (\sigma,\sigma') \in F \circ \mathcal{C}[\![P := P \times X; Y := Y - 1]\!]\}\\ \cup &\ \{(\sigma,\sigma) \mid \sigma(Y) = 0\}\end{aligned}$$

We can calculate $\mathcal{C}[\![P := P \times X; Y := Y - 1]\!]$ as follows:

$$\begin{aligned}&\mathcal{C}[\![P := P \times X; Y := Y - 1]\!]\\ &= \mathcal{C}[\![Y := Y - 1]\!] \circ \mathcal{C}[\![P := P \times X]\!]\\ &= \{(\sigma,\sigma[n/Y]) \mid n = \mathcal{A}[\![Y-1]\!]\,\sigma\} \circ \{(\sigma,\sigma[n/P]) \mid n = \mathcal{A}[\![P \times X]\!]\,\sigma\}\\ &= \{(\sigma,\sigma[\sigma(Y)-1/Y])\} \circ \{(\sigma,\sigma[\sigma(P) \times \sigma(X)/P])\}\\ &= \{(\sigma,\sigma[\sigma(Y)-1/Y,\sigma(P) \times \sigma(X)/P])\}\end{aligned}$$

So, we can finally present Γ simply as:

$$\begin{aligned}\Gamma(F) = &\ F \circ \{(\sigma,\sigma[\sigma(Y)-1/Y,\sigma(P) \times \sigma(X)/P]) \mid \sigma(Y) \neq 0\}\\ \cup &\ \{(\sigma,\sigma) \mid \sigma(Y) = 0\}\end{aligned}$$

(b) Then, the first three iterations in the fixed-point computation would give:

$$\begin{aligned}\Gamma^0(\emptyset) = &\ \emptyset\\ \Gamma^1(\emptyset) = &\ \Gamma(\Gamma^0(\emptyset)) = \Gamma(\emptyset) =\\ = &\ \{(\sigma,\sigma) \mid \sigma(Y) = 0\}\\ \Gamma^2(\emptyset) = &\ \Gamma(\Gamma^1(\emptyset)) =\\ = &\ \{(\sigma,\sigma[0/Y,\sigma(P) \times \sigma(X)/P]) \mid \sigma(Y) = 1\}\\ \cup &\ \{(\sigma,\sigma) \mid \sigma(Y) = 0\}\\ \Gamma^3(\emptyset) = &\ \Gamma(\Gamma^2(\emptyset)) =\\ = &\ \{(\sigma,\sigma[0/Y,\sigma(P) \times \sigma(X)^2/P]) \mid \sigma(Y) = 2\}\\ \cup &\ \{(\sigma,\sigma[0/Y,\sigma(P) \times \sigma(X)/P]) \mid \sigma(Y) = 1\}\\ \cup &\ \{(\sigma,\sigma) \mid \sigma(Y) = 0\}\end{aligned}$$

(c) In the general case we have:

$$\Gamma^i(\emptyset) = \left\{(\sigma,\sigma[0/Y,\sigma(P) \times \sigma(X)^{\sigma(Y)}/P]) \mid 0 \leq \sigma(Y) < i\right\}$$

(d) The fixed–point is the union/limit of all approximants:

$$\mathcal{C}[\![w]\!] = \mathit{fix}\,\Gamma = \left\{(\sigma,\sigma[0/Y,\sigma(P) \times \sigma(X)^{\sigma(Y)}/P]) \mid \sigma(Y) \geq 0\right\}$$

(e) Finally, for the denotational semantics of the whole program we obtain:

$$\mathcal{C}[\![p]\!] = \mathcal{C}[\![w]\!] \circ \mathcal{C}[\![P := 1]\!] = \left\{(\sigma,\sigma[0/Y,\sigma(X)^{\sigma(Y)}/P]) \mid \sigma(Y) \geq 0\right\}$$

2. Consider the following **IMP** program c for computing $sum(n) \stackrel{def}{=} \sum_{k=0}^{n} k$.

$Z := 0;$

$X := 1;$

while $X \leq Y$ **do**

$Z := Z + X;$

$X := X + 1$

Use the fixed–point characterization of the denotational semantics of while–loops of **IMP** and the Fixed–Point Theorem to iteratively compute the denotation of the above program. That is:

(a) Determine the transformer Γ for the while loop. Simplify it as much as possible.
Solution: After simplification, we obtain from the definition of $\Gamma_{b,c}$:

$$\begin{array}{rcl} \Gamma(F) & = & \{(\sigma,\sigma') \mid \sigma(X) \leq \sigma(Y) \wedge (\sigma[\sigma(Z)+\sigma(X)/Z, \sigma(X)+1/X], \sigma') \in F\} \\ & \cup & \{(\sigma,\sigma) \mid \sigma(X) > \sigma(Y)\} \end{array}$$

(b) Use Γ to compute the first three (non–empty) approximants of the fixed–point computation.

(c) Guess the general shape of the i–th approximant.
Solution: The i–th approximant can be presented as:

$$\begin{array}{rcl} \Gamma^i(\emptyset) & = & \{(\sigma, \sigma[\sigma(Z) + sum(\sigma(X), \sigma(Y))/Z, \sigma(Y) + 1/X]) \mid 0 \leq \sigma(Y) - \sigma(X) \leq i - 2\} \\ & \cup & \{(\sigma,\sigma) \mid \sigma(X) > \sigma(Y)\} \end{array}$$

where we use $sum(m,n)$ to denote $\sum_{k=m}^{n} k$.

(d) Use this to obtain the limit value (which is the denotation of the while loop).
Solution: The limit of the fixed-point construction is:

$$\begin{array}{rcl} \bigcup_{i\in\omega} \Gamma^i(\emptyset) & = & \{(\sigma, \sigma[\sigma(Z) + sum(\sigma(X), \sigma(Y))/Z, \sigma(Y) + 1/X]) \mid \sigma(X) \leq \sigma(Y)\} \\ & \cup & \{(\sigma,\sigma) \mid \sigma(X) > \sigma(Y)\} \end{array}$$

(e) Compute the denotation of the whole program.
Solution: For the whole program, we obtain:

$$\begin{array}{rcl} \mathcal{C}[\![c]\!] & = & \{(\sigma, \sigma[sum(1, \sigma(Y))/Z, \sigma(Y) + 1/X]) \mid \sigma(Y) \geq 1\} \\ & \cup & \{(\sigma, \sigma[0/Z, 1/X]) \mid \sigma(Y) < 1\} \end{array}$$

3. Use the fixed–point characterization of the denotational semantics of while–loops of **IMP** and the Fixed–Point Theorem to iteratively compute the denotation of the following **IMP** program *Med* for computing the mean value of two numbers (noting that it only terminates if the numbers are both even or both odd).

while $\neg(X = Y)$ **do**

if $X \leq Y$ **then** $X := X + 1; Y := Y - 1$

else $Y := Y + 1; X := X - 1$

Show the first few iterations of the fixed–point computation, then guess the general value of the i–th approximant, and finally the limit value.

3 Axiomatic Semantics of IMP

1. Consider the **IMP** program for computing powers:

   $P := 1;$
   while $\neg(Y = 0)$ **do**
   $P := P \times X;$
   $Y := Y - 1$

   (a) Specify this program in terms of a pre–condition and a post–condition.
   Solution: An obvious specification, which turns out to be provable, is:
   $Pre \equiv X = m \wedge Y = n$
   $Post \equiv P = m^n$

   (b) Use the axiomatic semantics of **IMP** to verify that the program meets its specification. Present the proof either as a derivation (that is, as a proof tree) or as a proof outline.
   Solution: The main difficulty is to pick an appropriate loop invariant. One good choice is $P \times X^Y = m^n$, which obviously implies the post–condition when $Y = 0$. Here is a proof outline for the above specification. The validity of the involved implications is obvious.

   $\{X = m \wedge Y = n\}$
   $\{1 \times X^Y = m^n\}$
   $P := 1;$
   $\{P \times X^Y = m^n\}$
   while $\neg(Y = 0)$ **do**
   $\{P \times X^Y = m^n \wedge \neg(Y = 0)\}$
   $\{(P \times X) \times X^{Y-1} = m^n\}$
   $P := P \times X;$
   $\{P \times X^{Y-1} = m^n\}$
   $Y := Y - 1$
   $\{P \times X^Y = m^n\}$
   $\{P \times X^Y = m^n \wedge \neg\neg(Y = 0)\}$
   $\{P = m^n\}$

2. Consider the following **IMP** program c for computing $sum(n) \stackrel{def}{=} \sum_{k=0}^{n} k$.

   $Z := 0;$
   $X := 1;$
   while $X \leq Y$ **do**
   $Z := Z + X;$
   $X := X + 1$

   Use the axiomatic semantics of **IMP** to verify that the program meets the specification

   $$\{Y = n \wedge Y \geq 0\}\, c\, \{Z = sum(n)\}$$

   (a) Present the proof (preferrably) as a proof tableau (that is, as a fully annotated program).
   Solution: The annotation is standard once one has chosen a suitable loop invariant. One such choice is $X \leq Y + 1 \wedge Y = n \wedge Z = sum(X - 1)$.

   (b) Identify and justify the resulting proof obligations.
   Solution: The standard annotation produces 3 proof obligations, which are easily discharged. The main property needed here is $sum(m) = sum(m - 1) + m$.

3. Consider the **IMP** program c for computing factorials:

$Y := 1;$
while $\neg(X = 1)$ **do**
$Y := Y \times X;$
$X := X - 1$

Use the axiomatic semantics of **IMP** (page 89) to verify that the program meets the specification $\{X = m \wedge X \geq 1\}c\{Y = m!\}$. Present the proof as a proof tableau (fully annotated program). Enumerate and justify the resulting proof obligations.

Solution: The main difficulty is finding an appropriate loop invariant. A natural loop invariant for the above program is the assertion $X! \times Y = m! \wedge X \geq 1$. Here is a proof tableau for the given specification:

$\{X = m \wedge X \geq 1\}$
$\{X! \times 1 = m! \wedge X \geq 1\}$
$Y := 1;$
$\{X! \times Y = m! \wedge X \geq 1\}$
while $\neg(X = 1)$ **do**
$\{X! \times Y = m! \wedge X \geq 1 \wedge \neg(X = 1)\}$
$\{(X-1)! \times Y \times X = m! \wedge X - 1 \geq 1\}$
$Y := Y \times X;$
$\{(X-1)! \times Y = m! \wedge X - 1 \geq 1\}$
$X := X - 1$
$\{X! \times Y = m! \wedge X \geq 1\}$
$\{X! \times Y = m! \wedge X \geq 1 \wedge \neg\neg(X = 1)\}$
$\{Y = m!\}$

The proof obligations resulting from this proof tableau are as follows:

(i) $\models X = m \wedge X \geq 1 \Rightarrow X! \times 1 = m! \wedge X \geq 1$
(ii) $\models X! \times Y = m! \wedge X \geq 1 \wedge \neg(X = 1) \Rightarrow (X-1)! \times Y \times X = m! \wedge X - 1 \geq 1$
(iii) $\models X! \times Y = m! \wedge X \geq 1 \wedge \neg\neg(X = 1) \Rightarrow Y = m!$

Their validity is obvious, apart maybe from (ii), the validity of which follows from the factorial property $a! = a \times (a-1)!$ which holds when $a - 1 \geq 1$. This explains why we need the conjunct $X \geq 1$ in the loop invariant.

4. Recall the sequence of Fibonacci numbers $0, 1, 1, 2, 3, 5, 8, 13, \ldots$ (add the last two to get the next). Let $Fib(i)$ denote the i-th member of the sequence. Consider the following **IMP** program c for computing Fibonacci numbers:

$X := 0;$
$Y := 1;$
while $3 \leq Z$ **do**
$T := Y;$
$Y := X + Y;$
$X := T;$
$Z := Z - 1$

Use the axiomatic semantics of **IMP** (page 89) to verify the above program, which has been specified as $\{Z \geq 2 \wedge Z = n\}\ c\ \{Y = Fib(n)\}$. Present the proof as a proof tableau (fully annotated program). Enumerate and justify the resulting proof obligations.

5. Consider the axiomatic semantics of **IMP**. Show that for all commands $c \in Com$ and all assertions $A \in Assn$,

$$\Vdash \{A\}\, c\, \{\mathbf{true}\}$$

Solution: We use structural induction on commands c to show $\forall c \in Com. \forall A \in Assn.\ \Vdash \{A\}\, c\, \{\mathbf{true}\}$. Here we only show the most interesting case $c \equiv \mathbf{while}\ b\ \mathbf{do}\ c'$. Assume $\forall A \in Assn.\ \Vdash \{A\}\, c'\, \{\mathbf{true}\}$ (induction hypothesis). Let $A \in Assn$. Consider the (incomplete) derivation:

$$\dfrac{\dfrac{\surd}{\models A \Rightarrow \mathbf{true}} \qquad \dfrac{\{\mathbf{true} \wedge b\}\, c'\, \{\mathbf{true}\}}{\{\mathbf{true}\}\ \mathbf{while}\ b\ \mathbf{do}\ c'\ \{\mathbf{true} \wedge \neg b\}} \qquad \dfrac{\surd}{\models \mathbf{true} \wedge \neg b \Rightarrow \mathbf{true}}}{\{A\}\ \mathbf{while}\ b\ \mathbf{do}\ c'\ \{\mathbf{true}\}}$$

The two resulting proof obligations hold trivially, while the sub-goal $\{\mathbf{true} \wedge b\}\, c'\, \{\mathbf{true}\}$ is derivable due to the induction hypothesis. Hence $\forall A \in Assn.\ \Vdash \{A\}\ \mathbf{while}\ b\ \mathbf{do}\ c'\ \{\mathbf{true}\}$.

6. Consider the axiomatic semantics of **IMP**. Show that for all commands $c \in Com$:

$$\Vdash \{\mathbf{true}\}\, c\, \{\mathbf{true}\}$$

Hint: use structural induction on c.

4 More Execrcises on IMP

1. Consider the transformation on **IMP** programs, from a program $c \equiv$ **if** b **then** $c_1; c_2$ **else** $c_1; c_3$ to program $c' \equiv c_1;$ **if** b **then** c_2 **else** c_3. Assume that execution of c_1 does not affect the evaluation of b.

   (a) Show that the transformation is a semantics preserving optimization, by proving $c \sim c'$ for the operational semantics of **IMP** (page 20).

   Solution: We can formalize the assumption that c_1 does not affect the evaluation of b as

   $$\Vdash \langle c_1, \sigma\rangle \to \sigma' \;\Rightarrow\; (\Vdash \langle b, \sigma\rangle \to \textbf{true} \Leftrightarrow \Vdash \langle b, \sigma'\rangle \to \textbf{true}) \qquad (\star)$$

   for all σ, σ'. We have to show that

   $$\Vdash \langle c, \sigma\rangle \to \sigma' \Leftrightarrow \Vdash \langle c', \sigma\rangle \to \sigma'$$

   for all σ, σ'. We show direction ($\Rightarrow$) only; the other direction is similar.
   Assume $\Vdash \langle c, \sigma\rangle \to \sigma'$. Then, there must be a derivation of the shape:

   $$\dfrac{\dfrac{(A)}{\langle b, \sigma\rangle \to \textbf{true}} \qquad \dfrac{\dfrac{(B)}{\langle c_1, \sigma\rangle \to \sigma''} \qquad \dfrac{(C)}{\langle c_2, \sigma''\rangle \to \sigma'}}{\langle c_1; c_2, \sigma\rangle \to \sigma'}}{\langle \textbf{if } b \textbf{ then } c_1; c_2 \textbf{ else } c_1; c_3, \sigma\rangle \to \sigma'}$$

   for some state σ'' and subderivations A, B and C, or alternatively, a derivation of the shape:

   $$\dfrac{\dfrac{(D)}{\langle b, \sigma\rangle \to \textbf{false}} \qquad \dfrac{\dfrac{(E)}{\langle c_1, \sigma\rangle \to \sigma'''} \qquad \dfrac{(F)}{\langle c_3, \sigma'''\rangle \to \sigma'}}{\langle c_1; c_3, \sigma\rangle \to \sigma'}}{\langle \textbf{if } b \textbf{ then } c_1; c_2 \textbf{ else } c_1; c_3, \sigma\rangle \to \sigma'}$$

   for some state σ''' and subderivations D, E and F. In the first case, we can use the subderivations A, B, C and the assumption $(\star)$ to construct the derivation:

   $$\dfrac{\dfrac{(B)}{\langle c_1, \sigma\rangle \to \sigma''} \qquad \dfrac{\dfrac{(A), (\star)}{\langle b, \sigma''\rangle \to \textbf{true}} \qquad \dfrac{(C)}{\langle c_2, \sigma''\rangle \to \sigma'}}{\langle \textbf{if } b \textbf{ then } c_2 \textbf{ else } c_3, \sigma''\rangle \to \sigma'}}{\langle c_1;\ \textbf{if } b \textbf{ then } c_2 \textbf{ else } c_3, \sigma\rangle \to \sigma'}$$

   showing $\Vdash \langle c', \sigma\rangle \to \sigma'$, and similarly for the second case.

   (b) Give an alternative proof by showing $\mathcal{C}[\![c]\!] = \mathcal{C}[\![c']\!]$ in the denotational semantics of **IMP** (page 60).

2. Show that the transformation on **IMP** programs, from program

   $$c_1 \equiv X := 3;\ \textbf{if } 0 \le X \textbf{ then } c \textbf{ else } c'$$

   to program

   $$c_2 \equiv X := 3;\ c$$

   is semantically preserving, and thus an optimization. Give two alternative proofs based:

   (a) on the big-step operational semantics of **IMP** (page 20), and

   (b) on its denotational semantics.

3. Let us extend the simple imperative language **IMP** with a "**repeat** c **until** b" command, having the intuitive meaning of repeatedly executing c and then testing b, and exiting the loop as soon as b is true.

   (a) Complete the operational semantics of the extended language by providing transition rules for the new command. Prove the equivalence (**repeat** c **until** b) $\sim$ (c; **while** $\neg b$ **do** c) under this semantics. Hint: Induction on derivations is an option here.

(b) Complete the denotational semantics of the extended language by defining a suitable transformation on partial functions $\Gamma'(\varphi)$ so that $\mathcal{C}[\![\textbf{repeat } c \textbf{ until } b]\!]$ can be defined as the least fixed point of Γ'. Prove the equality $\mathcal{C}[\![\textbf{repeat } c \textbf{ until } b]\!] = \mathcal{C}[\![c; \textbf{while } \neg b \textbf{ do } c]\!]$ under this semantics. Hint: You probably have to use some form of fixed point induction here.

(c) Complete the axiomatic semantics of the extended language by providing a Hoare rule for the new command. Give an intuitive justification for the rule.

5 Operational Semantics of REC

1. Consider the operational semantics of call–by–value evaluation of closed terms written in **REC**.

   (a) Evaluate the closed term $pow(2,1)$ under the declaration:

   $$pow(x,y) = \textbf{if } y \textbf{ then } 1 \textbf{ else } (x \times pow(x, y-1)).$$

   (b) Suggest a small–step operational semantics for **REC** by giving a set of rules for deriving one–step transitions of the shape $t \xrightarrow{1}{}^{d}_{va} t'$, enforcing a left–to–right call–by–value evaluation strategy. Assume that all function variables are of arity 2.
   Hint: numbers should be treated as final terms in such a semantics, and should therefore not be evaluated further.

   Solution: We have no rules for $t \equiv n$ since numbers are final values and are not evaluated further, and no rules for $t \equiv x$ since we are are considering closed terms only. A good choice of rules capturing the idea of single–step computation could be:

   $$\textsc{Op1}\ \frac{t_1 \xrightarrow{1}{}^{d}_{va} t_1'}{t_1 \textbf{ op } t_2 \xrightarrow{1}{}^{d}_{va} t_1' \textbf{ op } t_2} \qquad \textsc{Op2}\ \frac{t_2 \xrightarrow{1}{}^{d}_{va} t_2'}{n \textbf{ op } t_2 \xrightarrow{1}{}^{d}_{va} n \textbf{ op } t_2'}$$

   $$\textsc{Op3}\ \frac{\cdot}{n_1 \textbf{ op } n_2 \xrightarrow{1}{}^{d}_{va} n_1 \text{ op } n_2} \qquad \textsc{If1}\ \frac{t_0 \xrightarrow{1}{}^{d}_{va} t_0'}{\textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2 \xrightarrow{1}{}^{d}_{va} \textbf{if } t_0' \textbf{ then } t_1 \textbf{ else } t_2}$$

   $$\textsc{If2}\ \frac{\cdot}{\textbf{if } 0 \textbf{ then } t_1 \textbf{ else } t_2 \xrightarrow{1}{}^{d}_{va} t_1} \qquad \textsc{If3}\ \frac{n \not\equiv 0}{\textbf{if } n \textbf{ then } t_1 \textbf{ else } t_2 \xrightarrow{1}{}^{d}_{va} t_2}$$

   $$\textsc{Fun1}\ \frac{t_1 \xrightarrow{1}{}^{d}_{va} t_1'}{f_i(t_1,t_2) \xrightarrow{1}{}^{d}_{va} f_i(t_1',t_2)} \qquad \textsc{Fun2}\ \frac{t_2 \xrightarrow{1}{}^{d}_{va} t_2'}{f_i(n,t_2) \xrightarrow{1}{}^{d}_{va} f_i(n,t_2')}$$

   $$\textsc{Fun3}\ \frac{\cdot}{f_i(n_1,n_2) \xrightarrow{1}{}^{d}_{va} d_i[n_1/x_1, n_2/x_2]}$$

   (c) Evaluate the closed term $pow(2,1)$ under your small-step semantics, and show the term sequence $t \xrightarrow{1}{}^{d}_{va} t' \xrightarrow{1}{}^{d}_{va} t'' \xrightarrow{1}{}^{d}_{va} \ldots \xrightarrow{1}{}^{d}_{va} n$ of the evaluation.

   (d) Let $t \xrightarrow{*}{}^{d}_{va} n$ denote the fact that there is a term sequence $t \xrightarrow{1}{}^{d}_{va} t' \xrightarrow{1}{}^{d}_{va} t'' \xrightarrow{1}{}^{d}_{va} \ldots \xrightarrow{1}{}^{d}_{va} n$ of zero or more one–step transitions (that is, that t evaluates to n in the reflexive transitive closure of your small–step semantics, as in the example above). Then $t \xrightarrow{*}{}^{d}_{va} n$ is derivable if all transitions $t \xrightarrow{1}{}^{d}_{va} t'$, $t' \xrightarrow{1}{}^{d}_{va} t''$, $\ldots$, are derivable. Prove that for every t and n, if $t \rightarrow^{d}_{va} n$ is derivable in the large–step semantics, then $t \xrightarrow{*}{}^{d}_{va} n$ is derivable in your small–step semantics.
   Hint: you could use induction on derivations.
   Solution: While induction on derivations might be easier to understand here, it is rule induction which yields the shortest proof, and this is the technique which we shall illustrate.

   Let's denote by $P(t,n)$ the property that $t \xrightarrow{*}{}^{d}_{va} n$ is derivable in the small–step semantics. Then, what we need to prove is that $P(t,n)$ holds of all derivable tuples $t \rightarrow^{d}_{va} n$ in the large–step semantics. The principle of rule induction allows us to make this inference if we can prove that all tuples $t \rightarrow^{d}_{va} n$ which are axioms in the large–step semantics satisfy $P(t,n)$, and that all other rules preserve $P(t,n)$, that is, if the assumption tuples satisfy $P(t,n)$ then also the conclusion tuple does so. So, we proceed by investigating each large–step rule separately.

   (num) This rule is an axiom, so we have to show that $n \xrightarrow{*}{}^{d}_{va} n$ is derivable in the small–step semantics. But this follows immediately from reflexivity of $\xrightarrow{*}{}^{d}_{va}$.

   (op) This rule has two premises, so we assume derivability of $t_1 \xrightarrow{*}{}^{d}_{va} n_1$ and $t_2 \xrightarrow{*}{}^{d}_{va} n_2$, and we have to show derivability of $t_1 \textbf{ op } t_2 \xrightarrow{*}{}^{d}_{va} n_1 \text{ op } n_2$. Derivability of $t_1 \xrightarrow{*}{}^{d}_{va} n_1$ gives us, by means of rule Op1, derivability of $t_1 \textbf{ op } t_2 \xrightarrow{*}{}^{d}_{va} n_1 \textbf{ op } t_2$. On the other hand, derivability of $t_2 \xrightarrow{*}{}^{d}_{va} n_2$ gives us, by means of rule Op2, derivability of $n_1 \textbf{ op } t_2 \xrightarrow{*}{}^{d}_{va} n_1 \textbf{ op } n_2$. And

finally, axiom rule OP3 gives derivability of n_1 **op** $n_2 \overset{1}{\longrightarrow}^{d}_{va} n_1$ op n_2. Transitivity of $\overset{*}{\longrightarrow}^{d}_{va}$ allows us to combine these there results to infer derivability of t_1 **op** $t_2 \overset{*}{\longrightarrow}^{d}_{va} n_1$ op n_2.

(condt) Assume $t_0 \overset{*}{\longrightarrow}^{d}_{va} 0$ and $t_1 \overset{*}{\longrightarrow}^{d}_{va} n_1$ are derivable. The first assumption gives us, by means of rule IF1, derivability of **if** t_0 **then** t_1 **else** $t_2 \overset{*}{\longrightarrow}^{d}_{va}$ **if** 0 **then** t_1 **else** t_2. Next, rule IF2 guarantees derivability of **if** 0 **then** t_1 **else** $t_2 \overset{1}{\longrightarrow}^{d}_{va} t_1$. Derivability of **if** t_0 **then** t_1 **else** $t_2 \overset{1}{\longrightarrow}^{d}_{va} n_1$ follows then by transitivity of $\overset{*}{\longrightarrow}^{d}_{va}$ from these two results and from the second assumption.

(condf) Similar to the previous case.

(fn) Very similar to case *(op)* when all function variables are of arity 2.

This concludes the proof.

2. Consider the following program for computing the mean value of two numbers:

$$med(x, y) = \textbf{if } x - y \textbf{ then } x \textbf{ else } med(x + 1, y - 1)$$

Evaluation of closed terms $med(m, n)$ terminates for all $m, n \in N$ such that $m \leq n$ and $n - m$ is even, both in call–by–value and call–by–name order of execution.

(a) Formalize these termination statements by referring to the (big–step) operational semantics of **REC**.

(b) Prove the statements by using well–founded induction.

6 Denotational Semantics of REC

1. Use the denotational semantics of **REC** for call–by–value (pages 144–147), and the Fixed–Point Theorem (page 121) to iteratively compute the denotation of the following declaration.

$$pos(x) = \textbf{if } x \textbf{ then } 0 \textbf{ else } dec(x, 0 - x)$$

$$dec(x, y) = \textbf{if } x \textbf{ then } 0 \textbf{ else } (\textbf{if } y \textbf{ then } 1 \textbf{ else } dec(x - 1, y - 1))$$

Show the first few iterations of the fixed–point computation, the general i-th approximant, and the fixed point. Then evaluate the term $pos(3)$.