Programming language and compiler design - Exercise solutions

Matthias Kohl

November 24, 2015

Disclaimer

There is no warranty on the correctness, completeness, optimality of the provided solutions.

1 Series 1

1.1 Exercise 1 (C language)

1. int 4 = @;
   (a) using characters not in language character set
   (b) starting variable names with numbers...
2. int y = 5 *
   (a) not separating statements with semicolons
   (b) not following the syntactic structure of statements
3. a = 5;
   (a) use of undefined identifier
   (b) calling function with wrong amount of arguments
   (c) typing error (casting pointers to int, too)
4. int a = 1 / 0;
   (a) dynamic typing errors (Java cast outside of hierarchy)
   (b) pointer without memory (or null pointer)

(c) arithmetic errors (division by 0)

(d) memory overflow

(e) I/O errors (file not found...)

1.2 Exercise 2

1. $S \to A \cdot b \cdot C; A \to a \cdot A \mid \epsilon; C \to c \cdot C \mid \epsilon$
2. $S \to b \mid a \cdot S \cdot c$
3. $S \to a \cdot b \cdot c \cdot C \mid a \cdot S \cdot C; C \to c \cdot C \mid c$
4. $S \to a \cdot S \cdot a \mid b \cdot S \cdot b \mid c \cdot S \cdot c \mid a \mid b \mid c \mid \epsilon$

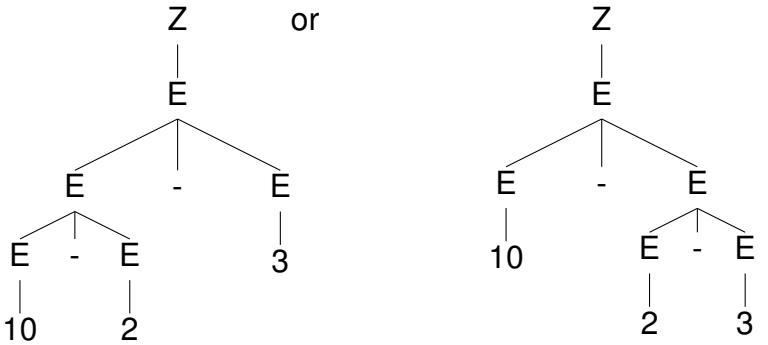
1.3 Exercise 3

1. $S \to AB \to aAB \to aaAB \to aaaAB \to aaaaAB \to aaaa\epsilon B \to aaaabB \to aaaabbB \to aaaabb\epsilon \to aaaabb$
2. $q_0 \to a \to q_0 \to a \to q_0 \to \epsilon \to q_1 \to b \to q_1 \to b \to q_1$ $[accepting\ state]$
3. $(q_0, aabb) \to (q_0, abb) \to (q_0, bb) \to (q_1, bb) \to (q_1, b) \to (q_1, \epsilon)$

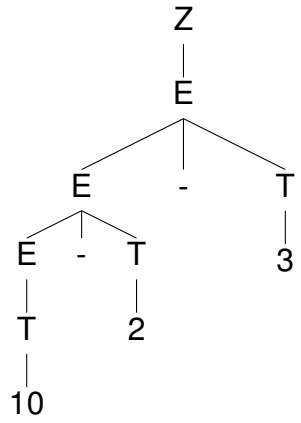
1.4 Exercise 4

1. $S \to aSb \to aaSbb \to aa\epsilon bb \to aabb$
2. $(q_0, aabb, Z) \to (q_1, abb, ZA) \to (q_1, bb, ZAA) \to (q_2, b, ZA) \to (q_2, \epsilon, Z) \to (q_3, \epsilon, \epsilon)$

1.5 Exercise 5

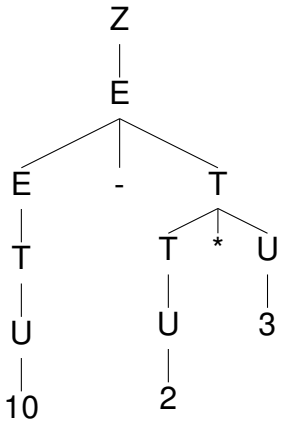
1. 

2.

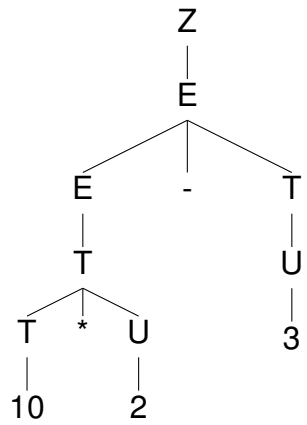


3. $Z \to E; E \to E - T \mid T; T \to T * U \mid U; U \to e$

10-2*3 :

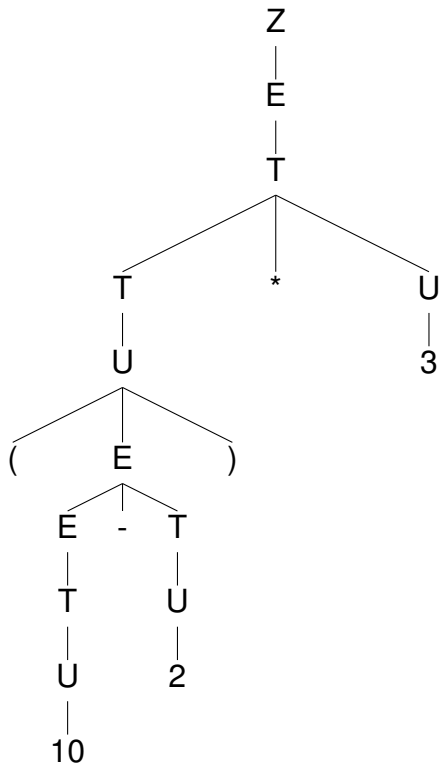


10*2-3 :

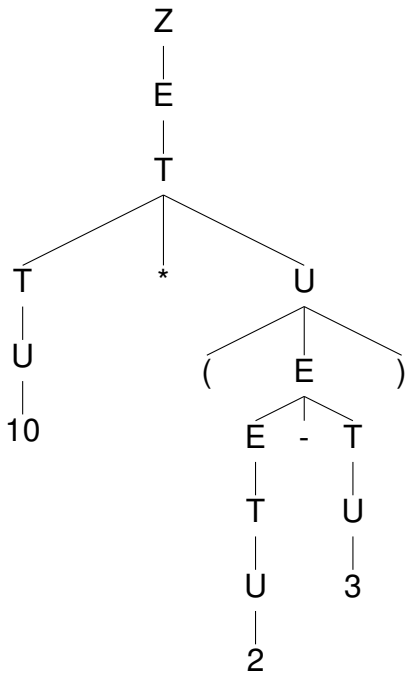


4. $Z \to E; E \to E - T \mid T; T \to T * U \mid U; U \to (E) \mid e$

(10-2)*3 :

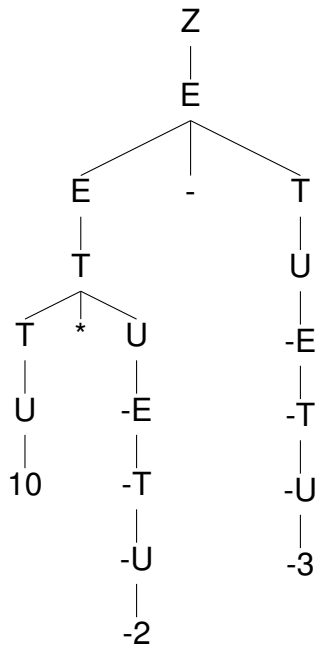


10*(2-3) :



5. $Z \to E; E \to E - T \mid T; T \to T * U \mid U; U \to (E) \mid -E \mid e$

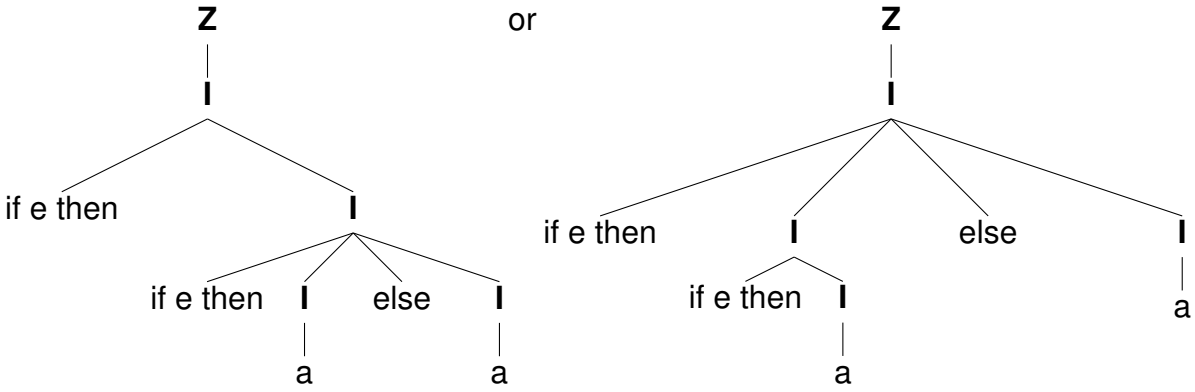
10* -2 - -3 :



1.6 Exercise 6

Exercise was not done in class !

1. *if e then if e then a else a*

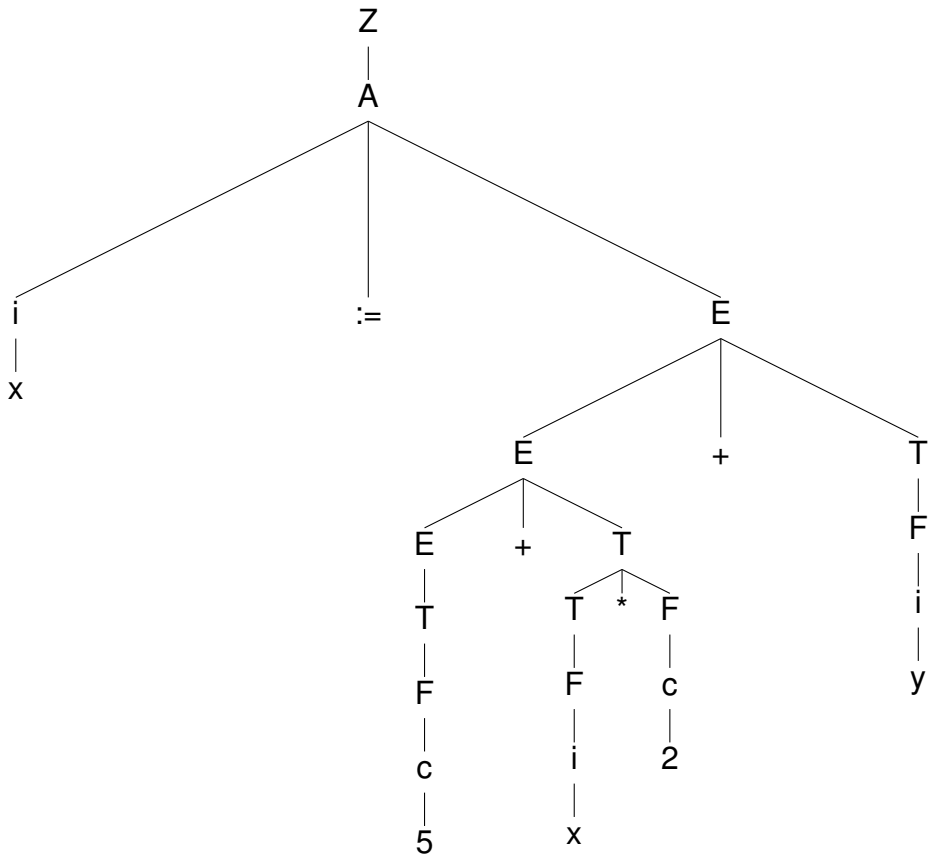


2. $Z \rightarrow I; I \rightarrow ifethenIelseI; I \rightarrow a; I \rightarrow skip;$

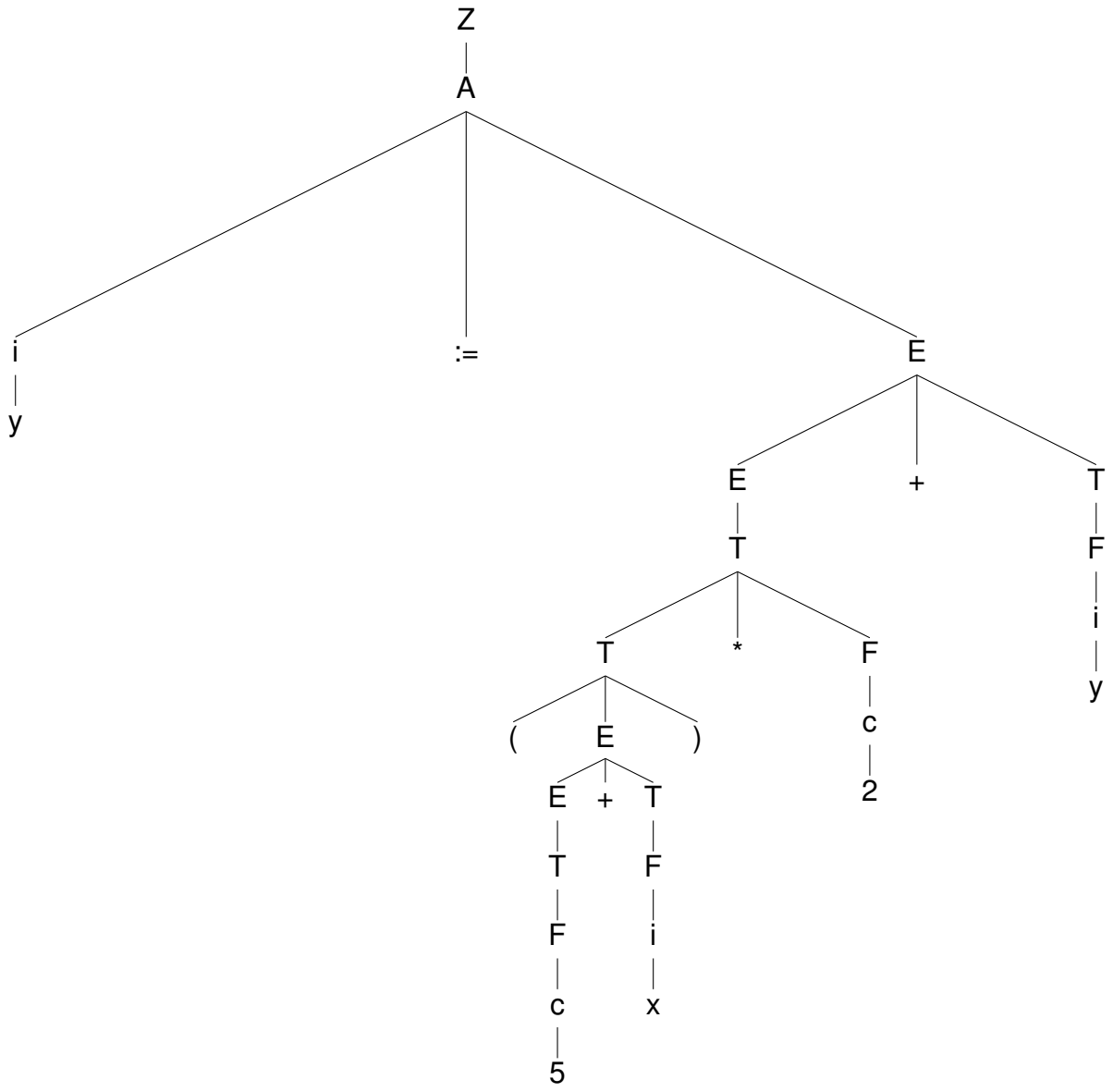
3. No

1.7 Exercise 7

1. (1)



(2)



2. (1)

   1) LD R_0, x
   2) MULT R_0, R_0, 2
   3) ADD R_0, R_0, 5
   4) LD R_1, y
   5) ADD R_0, R_0, R_1
   6) ST R_0, x

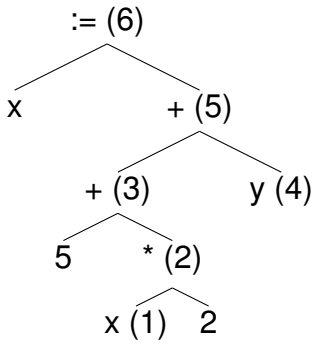
   (2)

   1) LD R_0, x

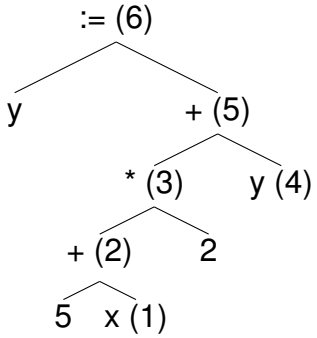
2) ADD R_0, R_0, 5
3) MULT R_0, R_0, 2
4) LD R_1, y
5) ADD R_1, R_0, R_1
6) ST R_1, y

3. See decorated simplified trees below, instruction numbers are in parenthesis

4. (1)



(2)



5. $S ::= x := e$
$e ::= n \mid x \mid e_1 + e_2 \mid e_1 * e_2$

1.8 Exercise 8

1. d

2. (Ada) F := for i in 1..n loop S end loop;
(C) for (I; C; Inc) S

2 Series 2

2.1 Exercise 15

1. $\Gamma = [x_1 \to Int, x_2 \to Int, x_3 \to Bool]$

$$\dfrac{\dfrac{\dfrac{\Gamma(x_1) = Int}{\Gamma \vdash x_1 : Int} \quad \Gamma \vdash 3 : Int}{\Gamma \vdash x_1 := 3;} \quad \dfrac{\dfrac{\Gamma(x_3) = Bool}{\Gamma \vdash x_3 : Bool}}{\Gamma \vdash not\ x_3 : Bool} \quad \dfrac{\dfrac{\dfrac{\Gamma(x_1) = Int}{\Gamma \vdash x_1 : Int} \quad \dfrac{\dfrac{\Gamma(x_2) = Int}{\Gamma \vdash x_2 : Int} \quad 1 : Int}{\Gamma \vdash x_2 + 1 : Int}}{\Gamma \vdash x_1 := x_2 + 1} \quad \dfrac{\dfrac{\Gamma(x_3) = Bool}{\Gamma \vdash x_3 : Bool} \quad \dfrac{\dfrac{\Gamma(x_3) = Bool}{\Gamma \vdash x_3 : Bool} \quad true : Bool}{\Gamma \vdash x_3\ and\ true : Bool}}{\Gamma \vdash x_3 := x_3\ and\ true}}{\Gamma \vdash x_1 := x_2 + 1;\ x_3 := x_3\ and\ true}}{\Gamma \vdash while\ not\ x_3\ do\ x_1 := x_2 + 1;\ x_3 := x_3\ and\ true\ od}}{\Gamma \vdash x_1 := 3; while\ not\ x_3\ do\ x_1 := x_2 + 1;\ x_3 := x_3\ and\ true\ od}$$

This expression is accepted by the type system.

2. $\Gamma = [x_1 \to Int, x_2 \to Int, x_3 \to Bool]$

$$\dfrac{\dfrac{\dots}{\Gamma \vdash x_1 := 3 * x_1 + 1} \quad \dfrac{\dfrac{\dfrac{\dfrac{\cancel{\Gamma(x_2) = Bool}}{\Gamma \vdash x_2 : Bool} \quad \dfrac{\Gamma(x_3) = Bool}{\Gamma \vdash x_3 : Bool}}{\Gamma \vdash x_2\ and \neg x_3 : Bool} \quad \dfrac{\dfrac{\Gamma(x_1) = Int}{\Gamma \vdash x_1 : Int} \quad \dfrac{\dfrac{\Gamma(x_2) = Int}{\Gamma \vdash x_2 : Int} \quad 1 : Int}{\Gamma \vdash x_2 + 1 : Int}}{\Gamma \vdash x_1 := x_2 + 1;} \quad \dfrac{\dfrac{\Gamma(x_1) = Int}{\Gamma \vdash x_1 : Int} \quad \dfrac{\Gamma(x_2) = Int}{\Gamma \vdash x_2 : Int}}{\Gamma \vdash x_1 := x_2;}}{\Gamma \vdash if\ x_2\ and \neg x_3\ then\ x_1 := x_2 + 1;\ else\ x_1 := x_2;\ fi}}{\Gamma \vdash x_1 := 3 * x_1 + 1; if\ x_2\ and \neg x_3\ then\ x_1 := x_2 + 1;\ else\ x_1 := x_2;\ fi}$$

This expression is rejected by the type system.

2.2 Exercise 16

1. $\Gamma_V = []$

$$\dfrac{\Gamma_V \vdash 3 : Int \quad \dfrac{\dfrac{\dfrac{\dfrac{\Gamma_V[x_1 \to Int](x_1) = Int}{\Gamma_V[x_1 \to Int] \vdash x_1 : Int}}{\Gamma_V[x_1 \to Int] \vdash 2 : Int} \quad \Gamma_V[x_1 \to Int] \vdash 2 * x_1 : Int}{\Gamma_V[x_1 \to Int] \vdash 2 * x_1 + 1 : Int} \quad \dfrac{\Gamma_V[x_1 \to Int][x_2 \to Int] \vdash true : Bool \quad \Gamma_V[x_1 \to Int][x_2 \to Int][x_3 \to Bool] \vdash \epsilon \mid \Gamma_I}{\Gamma_V[x_1 \to Int][x_2 \to Int] \vdash x_3 := true \mid \Gamma_I}}{\Gamma_V[x_1 \to Int] \vdash x_2 := 2 * x_1 + 1; x_3 := true \mid \Gamma_I}}{\Gamma_V \vdash x_1 := 3; x_2 := 2 * x_1 + 1; x_3 := true \mid \Gamma_I}$$

We can conclude that sequential evaluation gives: $\Gamma_I = [x_1 \to Int, x_2 \to Int, x_3 \to Bool]$

2. $\Gamma_V = []$

$$\dfrac{\Gamma_V \vdash 3 : Int \quad \dfrac{\dfrac{\Gamma_V \vdash 2 : Int \quad \dfrac{\cancel{\Gamma_V(x_1) = Int}}{\Gamma_V \vdash x_1 : Int}}{\Gamma_V \vdash 2 * x_1 : Int} \quad \Gamma_V \vdash 1 : Int}{\Gamma_V \vdash 2 * x_1 + 1 : Int}}{\Gamma_V \vdash x_1 := 3; x_2 := 2 * x_1 + 1; x_3 := true \mid \Gamma_I[x_1 \to Int]}$$

We can conclude that collateral evaluation rejects this expression.

2.3 Exercise 17

1. $e := true \mid false \mid n \mid x \mid e\ opa\ e \mid e\ oprel\ e \mid e\ opb\ e \mid e\ ?\ e : e$
2.

$$\frac{\Gamma \vdash e_1 : Bool \qquad \Gamma \vdash e_2 : t \qquad \Gamma \vdash e_3 : t}{\Gamma \vdash e_1\ ?\ e_2 : e_3 : t}$$

2.4 Exercise 19

1.

$$\frac{\Gamma \vdash e_1 \qquad \Gamma \vdash e_2 : Bool}{\Gamma \vdash repeat\ e_1\ until\ e_2}$$

2.
   -

$$\frac{\Gamma_V \vdash e_1 : Int \qquad \Gamma_V \vdash e_2 : Int \qquad \Gamma_V[x \to Int] \vdash e_3, x \notin dom(\Gamma_V)}{\Gamma_V \vdash for\ x\ from\ e_1\ to\ e_2\ do\ e_3}$$

     or

$$\frac{\Gamma_V \vdash e_1 : Int \qquad \Gamma_V \vdash e_2 : Int \qquad \Gamma_V[x \to Int] \vdash e_3, x \in dom(\Gamma_V) \Rightarrow \Gamma_V(x) = Int}{\Gamma_V \vdash for\ x\ from\ e_1\ to\ e_2\ do\ e_3}$$

   -

$$\frac{\Gamma_V \vdash x : Int\ \ \Gamma_V \vdash e_1 : Int\ \ \Gamma_V \vdash e_2 : Int \qquad \Gamma_V \vdash e_3}{\Gamma_V \vdash for\ x\ from\ e_1\ to\ e_2\ do\ e_3}$$

2.5 Exercise 20

1.

$$\frac{\Gamma_V[x \to t] \vdash D_V \mid \Gamma'_V, x \notin DV(D_V)}{\Gamma_V \vdash var\ x : t; D_V \mid \Gamma'_V[x \to t]}$$

2. Sequential:

$$\frac{\Gamma_V \vdash e : t' \qquad \Gamma_V[x \to t] \vdash D_V \mid \Gamma'_V, t = t', x \notin DV(D_V)}{\Gamma_V \vdash var\ x := e : t; D_V \mid \Gamma'_V[x \to t]}$$

Collateral:

$$\frac{\Gamma_V \vdash e : t' \qquad \Gamma_V \vdash D_V \mid \Gamma'_V, t = t', x \notin DV(D_V)}{\Gamma_V \vdash var\ x := e : t; D_V \mid \Gamma'_V[x \to t]}$$

2.6 Exercise 21

$D_{V_0} :=$ var x := 3
$D_{P_0} :=$ proc p is x := x + 1; proc q is call p
$S_{00} :=$ begin $D_{V_1} D_{P_1} S_1$ end
$S_0 := S_{00}$; call p
$D_{V_1} := \epsilon$
$D_{P_1} :=$ proc p is x := x + 5
$S_1 :=$ call q; call p

$$\dfrac{\dfrac{\Gamma_{V_0} \vdash 3 : Int \qquad \Gamma_{V_0}[x \mapsto Int] \vdash \epsilon \mid \Gamma_{V_1} \qquad x \notin DV(\epsilon)}{\Gamma_{V_0} \vdash D_V \mid \Gamma_{V_1}} \qquad \dfrac{\dfrac{\dots}{(\Gamma_{V_1}, \Gamma_{P_0}) \vdash \text{x := x + 1}} \qquad \dfrac{\dfrac{\Gamma_{P_1}(p) = proc}{(\Gamma_{V_1}, \Gamma_{P_1}) \vdash \text{call p}} \qquad (\Gamma_{V_1}, \Gamma_{P_2}) \vdash \epsilon \qquad q \notin DP(\epsilon)}{(\Gamma_{V_1}, \Gamma_{P_1}) \vdash \text{proc q is call p}} \qquad p \notin DP(\text{proc q is call p})}{(\Gamma_{V_1}, \Gamma_{P_0}) \vdash D_{P_0}} \qquad T_1}{(\Gamma_{V_0}, \Gamma_{P_0}) \vdash \text{begin } D_{V_0}; D_{P_0}; S_{00} \text{ end}} \qquad \dfrac{\Gamma_{P_0}(p) = proc}{(\Gamma_{V_1}, \Gamma'_{P_2}) \vdash \text{call p}}}{(\Gamma_{V_0}, \Gamma_{P_0}) \vdash \text{begin } D_{V_0}; D_{P_0}; S_0 \text{ end}}$$

T_1:

$$\dfrac{\Gamma_{V_1} \vdash D_{V_1} = \epsilon \mid \Gamma_{V_1} \qquad \dfrac{\dfrac{\Gamma_{V_1}(x) = Int \dots}{(\Gamma_{V_1}, \Gamma'_{P_2}) \vdash \text{x := x + 5}} \qquad (\Gamma_{V_1}, \Gamma_{P_3}) \vdash \epsilon \qquad p \notin DP(\epsilon)}{(\Gamma_{V_1}, \Gamma'_{P_2}) \vdash D_{P_1}} \qquad \dfrac{\dfrac{\Gamma'_{P_3}(q) = proc}{(\Gamma_{V_1}, \Gamma'_{P_3}) \vdash \text{call q}} \qquad \dfrac{\Gamma'_{P_3}(p) = proc}{(\Gamma_{V_1}, \Gamma'_{P_3}) \vdash \text{call p}}}{(\Gamma_{V_1}, \Gamma'_{P_3}) \vdash S_1}}{(\Gamma_{V_1}, \Gamma'_{P_2}) \vdash \text{begin } D_{V_1} D_{P_1} S_1 \text{ end}}$$

With:

$\Gamma_{V_0} = []$
$\Gamma_{P_0} = []$
$\Gamma_{V_1} = \Gamma_{V_0}[x \mapsto Int] = [x \mapsto Int]$
$\Gamma_{P_1} = \Gamma_{P_0}[p \mapsto proc] = [p \mapsto proc]$
$\Gamma_{P_2} = \Gamma_{P_1}[q \mapsto proc] = [p \mapsto proc, q \mapsto proc]$
$\Gamma'_{P_2} = upd(\Gamma_{P_0}, procpis \dots; procqis \dots) = upd(\Gamma_{P_0}[p \mapsto proc], procqis \dots) =$
$= upd(\Gamma_{P_0}[p \mapsto proc][q \mapsto proc], \epsilon) = [p \mapsto proc][q \mapsto proc]$
$\Gamma_{P_3} = \Gamma'_{P_2}[p \mapsto proc] = [p \mapsto proc, q \mapsto proc]$
$\Gamma'_{P_3} = upd(\Gamma'_{P_2}, procpis \dots) = upd(\Gamma'_{P_2}[p \mapsto proc], \epsilon) = [p \mapsto proc][q \mapsto proc]$

2.7 Exercise 22

1. $D_P1 = proc\ p1\ is\ call\ p2$
   $D_P2 = proc\ p2\ is\ call\ p1$
   $S = call\ p1;$
   Static binding:

$$\dfrac{\dfrac{[](p2) = proc}{([],[]) \vdash D_{P1}} \quad ([],\ldots) \vdash \ldots}{([],[]) \vdash D_{P1}; D_{P2}} \quad \dfrac{\ldots}{([],\Gamma'_V) \vdash S}}{[] \vdash \epsilon \mid [] \qquad ([],[]) \vdash begin\ \epsilon; D_{P1}; D_{P2}; S\ end}$$

   Dynamic binding:

$$\dfrac{\dfrac{\dfrac{\dfrac{\ldots \text{(loops infinitely)}}{([],\Gamma'_P) \vdash call\ p1}}{([],\Gamma'_P) \vdash call\ p2}}{([],\Gamma'_P) \vdash call\ p1}}{[] \vdash \epsilon \mid [] \qquad ([],\Gamma'_P) \vdash S}}{([],[]) \vdash begin\ \epsilon; D_{P1}; D_{P2}; S\ end}$$

   with $\Gamma'_P = [p1 \to call\ p2, p2 \to call\ p1]$

2. Static binding: Update Γ_P to Γ'_P in second axiom to add the information of future procedure declarations

$$\dfrac{\Gamma_V \vdash D_V \mid \Gamma'_V \qquad (\Gamma'_V, \Gamma'_P) \vdash D_P \qquad (\Gamma_V, \Gamma'_P) \vdash S}{(\Gamma_V, \Gamma_P) \vdash begin\ D_V; D_P; S\ end}$$

   Dynamic binding:

$$\dfrac{\Gamma_V \vdash D_V \mid \Gamma'_V \qquad (\Gamma'_V, \Gamma'_P) \vdash S}{(\Gamma_V, \Gamma_P) \vdash begin\ D_V; D_P; S\ end}$$

2.8 Exercise 23

1.

$$\frac{}{V \vdash n}$$

$$\frac{}{V \vdash skip}$$

$$\frac{x \in V}{V \vdash x}$$

$$\frac{V \vdash e_1 \qquad V \vdash e_2}{V \vdash e_1 \text{ opa/opb/oprel } e_2}$$

$$\frac{V \vdash e}{V \vdash \text{var } x := e \mid V \cup x}$$

$$\frac{V \vdash S_1 \mid V' \qquad V' \vdash S_2}{V \vdash S_1; S_2}$$

$$\frac{V \vdash b \qquad V \vdash S \mid V'}{V \vdash \text{while } b \text{ do } S \text{ od} \mid V'}$$

$$\frac{V \vdash b \qquad V \vdash S_1 \mid V_1 \qquad V \vdash S_2 \mid V_2}{V \vdash \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi} \mid V_1 \cap V_2}$$

2. (a)

$S_0 :=$ x:=1
$S_1 :=$ if x=0 then y:=x+1 else y:=x-1 fi

$$\dfrac{\dfrac{\{\} \vdash 1}{\{\} \vdash \text{x:=1} \mid \{x\}} \qquad \dfrac{\dfrac{x \in \{x\} \ldots}{\{x\} \vdash x = 0} \qquad \dfrac{x \in \{x\} \ldots}{\{x\} \vdash \text{y:=x+1} \mid \{x, y\}} \qquad \dfrac{x \in \{x\} \ldots}{\{x\} \vdash \text{y:=x-1} \mid \{x, y\}}}{\{x\} \vdash \text{if x=0 then y:=x+1 else y:=x-1 fi} \mid \{x, y\}}}{\{\} \vdash S_0; S_1}$$

(b)

$S_0 :=$ x:=1
$S_1 :=$ if x=0 then x:=x+1 else y:=x-1 fi

$$\dfrac{\dfrac{}{\{\} \vdash 1}\ \dfrac{\dfrac{x \in \{x\} \ldots}{\{x\} \vdash x = 0}\quad \dfrac{x \in \{x\} \ldots}{\{x\} \vdash \text{x:=x+1} \mid \{x\}}\quad \dfrac{x \in \{x\} \ldots}{\{x\} \vdash \text{y:=x-1} \mid \{x, y\}}}{\{x\} \vdash \text{if x=0 then x:=x+1 else y:=x-1 fi} \mid \{x\}}}{\{\} \vdash S_0; S_1}$$

(c)

$$\dfrac{\dfrac{}{\{\} \vdash 1}\ \dfrac{\dfrac{\dfrac{x \in \{x\}}{\{x\} \vdash x}\quad \dfrac{\cancel{y \in \{x\}}}{\{x\} \vdash y}}{\{x\} \vdash x + y}\quad \dfrac{\{x\} \vdash \text{y:=x+y} \mid \{x, y\}\qquad \{x\} \vdash \text{x:=x+1}}{\{x\} \vdash \text{y:=x+y; x:=x+1}}}{\{x\} \vdash \text{while x<=10 do y:=x+y; x:=x+1 od} \mid \ldots}}{\{\} \vdash \text{x:=1; while x<=10 do y:=x+y; x:=x+1 od} \mid \ldots}$$

3. The following program is correct at run-time but will be rejected by the type-system:
   if $true$ then $t := 2$ else $skip$ fi$; x := t + 2$
   This is rejected because t is not declared in both paths of the if-then-else statement but at run-time, only the first path would be executed where t is declared, so the program would be correct at run-time.

3 Series 3

3.1 Exercise 30

1. $FV(x + 1) = \{x\}$
2. $FV(3 * x + y) = \{x, y\}$

3.2 Exercise 31

1. $FV(n) = \emptyset; FV(x) = \{x\}; FV(a_1 + a_2) = FV(a_1) \cup FV(a_2); FV(a_1 - a_2) = FV(a_1) \cup FV(a_2); FV(a_1 * a_2) = FV(a_1) \cup FV(a_2)$
2. Proof by induction that
   $\forall a \in$ **Aexp**$, \forall \sigma, \sigma' \in$ **State**$, \forall x \in FV(a), \sigma(x) = \sigma'(x) \Rightarrow \mathcal{A}[a]\sigma = \mathcal{A}[a]\sigma'$

(a) Base steps:

- $a = n \rightarrow \mathcal{A}[a]\sigma = n = \mathcal{A}[a]\sigma'$ by definition of $\mathcal{A}$
- $a = x \rightarrow \mathcal{A}[a]\sigma = \sigma(x) = \sigma'(x) = \mathcal{A}[a]\sigma'$ by definition of $\mathcal{A}$ and assumption that $\forall x \in FV(a), \sigma(x) = \sigma'(x)$

(b) Induction step:
Let us consider $a = a_1 + a_2$.
$\mathcal{A}[a]\sigma = \mathcal{A}[a_1]\sigma +_I \mathcal{A}[a_2]\sigma$.
According to the definition of FV : $FV(a_1 + a_2) = FV(a_1) \cup FV(a_2)$.
Thus, $FV(a_1) \subseteq FV(a), FV(a_2) \subseteq FV(a)$.
From $\sigma(x) = \sigma'(x)$, we declare $\sigma(x) = \sigma'(x)$ for any $x \in FV(a_1)$ and $x \in FV(a_2)$.
With this, and with the induction hypothesis, we deduce that $\mathcal{A}[a_1]\sigma = \mathcal{A}[a_1]\sigma'$ and same goes for a_2.
Thus: $\mathcal{A}[a]\sigma = \mathcal{A}[a_1]\sigma +_I \mathcal{A}[a_2]\sigma = \mathcal{A}[a_1]\sigma' +_I \mathcal{A}[a_2]\sigma' = \mathcal{A}[a]\sigma'$
We can apply the same induction step to all other arithmetical operands.
Hence, by induction, the statement above holds.

3.3 Exercise 32

This is very similar to Exercise 31 (exercise was not finished entirely in class).

3.4 Exercise 33

$\sigma = [x \mapsto 1, y \mapsto 2, z \mapsto 3]$

1. $\sigma[x \mapsto y + z] = \sigma[x \mapsto \mathcal{A}[y + z]\sigma] = \sigma[x \mapsto 2 + 3] = [x \mapsto 5, y \mapsto 2, z \mapsto 3]$
2. $\sigma[x \mapsto 2*x+y, y \mapsto x] = \sigma[x \mapsto \mathcal{A}[2*x+y]\sigma, y \mapsto \mathcal{A}[x]\sigma] = \sigma[x \mapsto 2*1+2, y \mapsto 1] = [x \mapsto 4, y \mapsto 1, z \mapsto 3]$

3.5 Exercise 34

1. Let us consider $a' \in$ **Aexp**, $x \in$ **Var**
   - $n[a' \mid x] = n$
   - $y[a' \mid x] = \begin{cases} a' & \text{if } x = y \\ y & \text{otherwise} \end{cases}$
   - $(a_1 + a_2)[a' \mid x] = a_1[a' \mid x] + a_2[a' \mid x]$
2. Proof that $\forall a, a' \in$ **Aexp**, $\forall x \in$ **Var** $\cap FV(a), \forall \sigma \in$ **State**, $\mathcal{A}[a[a' \mid x]]\sigma = \mathcal{A}[a]\sigma[x \mapsto \mathcal{A}[a']\sigma]$
   Proof by induction:

(a) Base steps:

- $a = n \Rightarrow \mathcal{A}[n[a' \mid x]]\sigma = \mathcal{A}[n]\sigma = \mathcal{A}[n]\sigma[x \mapsto \mathcal{A}[a']\sigma]$
- $a = x \Rightarrow \mathcal{A}[x[a' \mid x]]\sigma = \mathcal{A}[a']\sigma = \mathcal{A}[x]\sigma[x \mapsto \mathcal{A}[a']\sigma]$

(b) Induction step:
Let us consider $a = a_1 + a_2$.

$$\begin{aligned}
\mathcal{A}[(a_1 + a_2)[a' \mid x]]\sigma &= \mathcal{A}[a_1[a' \mid x] + a_2[a' \mid x]]\sigma && \text{definition of substitution} \\
&= \mathcal{A}[a_1[a' \mid x]]\sigma + \mathcal{A}[a_2[a' \mid x]]\sigma && \text{definition of } \mathcal{A} \\
&= \mathcal{A}[a_1]\sigma[x \mapsto \mathcal{A}[a']\sigma + \\
&\mathcal{A}[a_2]\sigma[x \mapsto \mathcal{A}[a']\sigma && \text{by induction} \\
&= \mathcal{A}[a_1 + a_2]\sigma[x \mapsto \mathcal{A}[a']\sigma] && \text{definition of } \mathcal{A}
\end{aligned}$$

We can apply the same induction step to all other arithmetical operands. Hence, by induction, the statement above holds.

4 Series 4

4.1 Exercise 36

1. (a)
$$\dfrac{\dfrac{(S_{11}, \sigma) \to \sigma_0 \qquad (S_{12}, \sigma_0) \to \sigma_1}{(S_{11}; S_{12}, \sigma) \to \sigma_1} \qquad (S_2, \sigma_1) \to \sigma_2}{(S_1; S_2, \sigma) \to \sigma_2}$$

With:
$S_{11} ::= x := y \mid S_{12} ::= x := z \mid S_2 ::= y := z$
$\sigma_0 = \sigma[x \mapsto \mathcal{A}[y]\sigma] = [x \mapsto 7, y \mapsto 7]$
$\sigma_1 = \sigma_0[x \mapsto \mathcal{A}[z]\sigma_0] = [x \mapsto 0, y \mapsto 7]$
$\sigma_2 = \sigma_1[y \mapsto \mathcal{A}[z]\sigma_1] = [x \mapsto 0, y \mapsto 0]$

(b)
$$\dfrac{\dfrac{(S_{11}, \sigma) \to \sigma_0 \qquad (S_{12}, \sigma_0) \to \sigma_1}{(S_{11}; S_{12}, \sigma) \to \sigma_1} \qquad (S_2, \sigma_1) \to \sigma_2}{(S_1; S_2, \sigma) \to \sigma_2}$$

With:
$S_{11} ::= z := x \mid S_{12} ::= x := y \mid S_2 ::= y := z$
$\sigma_0 = \sigma[z \mapsto \mathcal{A}[x]\sigma] = [x \mapsto 5, y \mapsto 7, z \mapsto 5]$
$\sigma_1 = \sigma_0[x \mapsto \mathcal{A}[y]\sigma_0] = [x \mapsto 7, y \mapsto 7, z \mapsto 5]$
$\sigma_2 = \sigma_1[y \mapsto \mathcal{A}[z]\sigma_1] = [x \mapsto 7, y \mapsto 5, z \mapsto 5]$

2. (a)
$$\dfrac{(S_{11}, \sigma) \to \sigma_1}{(if\ S_1\ then\ S_{11}\ else\ S_{12}, \sigma) \to \sigma_1}$$

With:
$S_1 ::= x + y >= 3 \mid S_{11} ::= y := x \mid S_{12} ::= x := y$
$\mathcal{B}[x + y >= 3]\sigma = \mathcal{B}[\sigma[x] + \sigma[y] >= 3] = \mathcal{B}[5 + 7 \geq 3] = tt$
$\sigma_1 = \sigma[y \mapsto \mathcal{A}[x]\sigma] = [x \mapsto 5, y \mapsto 5]$

(b) $$\frac{(S_{11}, \sigma) \to \sigma_1}{(if\ S_1\ then\ S_{11}\ else\ S_{12}, \sigma) \to \sigma_1}$$

With:
$S_1 ::= y >= x \mid S_{11} ::= y := x \mid S_{12} ::= x := y$
$\mathcal{B}[y >= x]\sigma = \mathcal{B}[\sigma[y] >= \sigma[x]] = \mathcal{B}[7 \geq 5] = tt$
$\sigma_1 = \sigma[y \mapsto \mathcal{A}[x]\sigma] = [x \mapsto 5, y \mapsto 5]$

4.2 Exercise 39

1. Terminates in every state because the while loop iterates at most the number of times than the value of x. If $x < 1$ at the beginning, then the while loop does not loop at all. Proof by building the derivation tree.

2. Loops in every state because the condition of the while loop is always **true**. Proof by contradiction (assume that the statement terminates then proof by induction that the leaf of the derivation tree can still be augmented according to the rule of the while statement: contradiction).

3. There are states from which the execution terminates: For example $[x \mapsto 1]$ (the while loop does not loop at all).
   There are some states from which it doesn't terminate: for example $[x \mapsto 0]$ (the while loop decrements x infinitely and $\neg(x = 1)$ will always be **true**. Proof by contradiction (we know that $x < 1$ and assume that the statement terminates then proof by induction that the leaf of the derivation tree can still be augmented and that $x < 1$ is still true according to the rule of the while statement: contradiction).

4.3 Exercise 41

1. - $(n, \sigma) \to z = \mathcal{N}(n)$
   - $(x, \sigma) \to z = \sigma(x)$
   - $$\frac{(a_1, \sigma) \to v_1 \qquad (a_2, \sigma) \to v_2}{(a_1 + a_2, \sigma) \to v_1 + v_2}$$
   - $\mathcal{A}'[a]\sigma = \begin{cases} v & \text{if } (a, \sigma) \to v \\ undef & \text{otherwise} \end{cases}$

- $$\frac{(a_2, \sigma) \to v_2 \qquad v_2 \neq 0 \qquad (a_1, \sigma) \to v_1}{(a_1/a_2, \sigma) \to v_1/v_2}$$

2. Proof by induction that $\forall a \in$ **Aexp**, $\forall \sigma \in$ **State**, $\mathcal{A}[a]\sigma = \mathcal{A}'[a]\sigma$:
   (a) Base cases
   - $a = n \Rightarrow \mathcal{A}[n]\sigma = \mathcal{N}(n) = \mathcal{A}'[n]\sigma$
   - $a = x \Rightarrow \mathcal{A}[x]\sigma = \sigma(x) = \mathcal{A}'[n]\sigma$

   (b) Induction step: Use the semantics of $\mathcal{A}$ and $\mathcal{A}'$ to prove that $\mathcal{A}[a_1 + a_2]\sigma = \mathcal{A}'[a_1 + a_2]\sigma$.

3. TODO (exercise was not finished in class)

4.4 Exercise 42

1. $\forall S_1, S_2, S_3 \in$ **Stm**:

$$\frac{(S_1, \sigma_0) \to \sigma_0' \qquad \dfrac{(S_2, \sigma_0') \to \sigma_0'' \qquad (S_3, \sigma_0'') \to \sigma_0'''}{(S_2; S_3), \sigma_0') \to \sigma_0'''}}{(S_1; (S_2; S_3), \sigma) \to \sigma_0'''}$$

$$\frac{\dfrac{(S_1, \sigma_1) \to \sigma_1' \qquad (S_2, \sigma_1') \to \sigma_1''}{(S_1; S_2), \sigma_1) \to \sigma_1''} \qquad (S_3, \sigma_1'') \to \sigma_1'''}{((S_1; S_2); S_3), \sigma) \to \sigma_1'''}$$

   Using the determinism of operational semantics, we can deduce that:
   $\sigma_0' = \sigma_1' \Rightarrow \sigma_0'' = \sigma_1'' \Rightarrow \sigma_0''' = \sigma_1'''$.
   Hence, $(S_1; (S_2; S_3))$ is semantically equivalent to $((S_1; S_2); S_3)$.

2. For $S_1 ::= x := x + 1; S_2 ::= x = 3; \sigma_0 = [x \mapsto 0]$:

$$\frac{(S_1, \sigma_0) \to \sigma_0' \qquad (S_2, \sigma_0') \to \sigma_0''}{(S_1; S_2)\sigma_0) \to \sigma_0''}$$

   With $\sigma_0 = [x \mapsto 0]$
   $\sigma_0' = [x \mapsto 1]$
   $\sigma_0'' = [x \mapsto 3]$

$$\frac{(S_2, \sigma_0) \to \sigma_1' \qquad (S_1, \sigma_1') \to \sigma_1''}{(S_2; S_1)\sigma_0) \to \sigma_1''}$$

   With $\sigma_0 = [x \mapsto 0]$
   $\sigma_1' = [x \mapsto 3]$
   $\sigma_1'' = [x \mapsto 4]$

$\sigma_0'' \neq \sigma_1'' \Rightarrow (S_1; S_2)$ is not semantically equivalent to $(S_2; S_1)$.

4.5 Exercise 43

1. All programs in this language always terminate.
2. Proof by induction that $\forall S \in \textbf{Stm}, \sigma \in \textbf{State}, S$ terminates:
   (a) Base steps:
   - $S ::= x := a$ terminates:
   $$\frac{}{(x := a, \sigma) \to \sigma[x \mapsto \mathcal{A}[a]\sigma]}$$
   - $S ::= skip$ terminates:
   $$\frac{}{(skip, \sigma) \to \sigma}$$

   (b) Induction on the semantics of S:
   - $S ::= (S_1; S_2)$ terminates if S_1 and S_2 terminate:
   $$\frac{(S_1, \sigma) \to \sigma' \qquad (S_2, \sigma') \to \sigma''}{(S_1; S_2, \sigma) \to \sigma''}$$
   - $S ::=$ if b then S_1 else S_2 terminates if S_1 and S_2 terminate:
   If $\mathcal{B}[b]\sigma = \textbf{tt}$:
   $$\frac{(S_1, \sigma) \to \sigma'}{(\text{if } b \text{ then } S_1 \text{ else } S_2, \sigma) \to \sigma'}$$
   If $\mathcal{B}[b]\sigma = \textbf{ff}$:
   $$\frac{(S_2, \sigma) \to \sigma'}{(\text{if } b \text{ then } S_1 \text{ else } S_2, \sigma) \to \sigma'}$$

   Hence, by induction, all programs in this language always terminate.

4.6 Exercise 45

1. If $\mathcal{B}[b]\sigma' = \textbf{tt}$:
   $$\frac{(S, \sigma) \to \sigma'}{(\text{repeat } S \text{ until } b, \sigma) \to \sigma'}$$
   If $\mathcal{B}[b]\sigma' = \textbf{ff}$:
   $$\frac{(S, \sigma) \to \sigma' \qquad (\text{repeat } S \text{ until } b, \sigma') \to \sigma''}{(\text{repeat } S \text{ until } b, \sigma) \to \sigma''}$$
2. We construct the proof-tree for the second statement (and we compare it to the proof-tree for the first statement above):

$$\frac{(S, \sigma_1) \to \sigma_1' \qquad T}{(S; \text{if } b \text{ then } skip \text{ else } (\text{repeat } S \text{ until } b), \sigma_1) \to \sigma_1''}$$

The part T has the following proof-tree:
If $\mathcal{B}[b]\sigma_1' =$ **tt**:

$$\frac{}{(skip, \sigma_1') \to \sigma_1'}$$

If $\mathcal{B}[b]\sigma_1' =$ **ff**:

$$\frac{\ldots}{(\text{repeat } S \text{ until } b, \sigma_1') \to \sigma_1''}$$

From the semantic determinism, we know that if $\sigma = \sigma_1$, then $\sigma' = \sigma_1'$. This also implies $\mathcal{B}[b]\sigma' = \mathcal{B}[b]\sigma_1'$.
At this point, there are two cases:

- $\mathcal{B}[b]\sigma' = \mathcal{B}[b]\sigma_1' =$ **tt**: then the final state in the first proof-tree is σ'. For the second proof-tree, if $\mathcal{B}[b]\sigma_1' =$ **tt**, then the final state is σ_1'. From what we noted above, the final states are equal.

- $\mathcal{B}[b]\sigma' = \mathcal{B}[b]\sigma_1' =$ **ff**: then the final state in the first proof-tree is σ''. For the second proof-tree, the final state is σ_1''. Because the same statement is executed in the same state, from semantic determinism, we can conclude that $\sigma'' = \sigma_1''$. Hence, the final states are equal.

3. We give a function $f :$ **Stm** $\times$ **State** $\to$ **Stm** that transforms any program containing the 'repeat' construct into a program containing only 'while' constructs:

   - $f(\text{repeat } S \text{ until } b, \sigma) = \begin{cases} S & \text{if } \mathcal{B}[b]\sigma = \textbf{tt} \\ S; \text{if } b \text{ then skip else } f(\text{repeat } S \text{ until } b) \text{ fi} & otherwise \end{cases}$
   - $f(S, \sigma) = S$

   This transformation is computable but only by simulating the program at runtime with the given initial state. The size of the resulting program is much bigger than the original program because we have to add the statement S and the 'if-then-else' construct as often as the loop is being executed at runtime.

4.7 Exercise 46

TODO (This exercise was not solved in class and might not be exactly correct as presented here !)

1. See below
2. There are a few ways to imagine keeping track of the fact that we are in a for-loop:
   (a) We assume that there is an undefined variable in every possible state σ and we have a function $undef$: **State** $\rightarrow$ **Var** giving us an undefined variable v in the given state σ before entering the for-loop.
   If $\sigma(v) = undef$:

$$\frac{(\text{for } x \text{ from } a_1 \text{ to } a_2 \text{ do } S, \sigma[x \mapsto \mathcal{A}[a_1]\sigma][v \mapsto 0]) \rightarrow \sigma'}{(\text{for } x \text{ from } a_1 \text{ to } a_2 \text{ do } S, \sigma) \rightarrow \sigma'}$$

   Else if $\sigma(x) > \mathcal{A}[a_2]\sigma$:

$$\frac{}{(\text{for } x \text{ from } a_1 \text{ to } a_2 \text{ do } S, \sigma) \rightarrow \sigma[v \mapsto undef]}$$

   Otherwise:

$$\frac{(S, \sigma) \rightarrow \sigma' \qquad (\text{for } x \text{ from } a_1 \text{ to } a_2 \text{ do } S, \sigma'[x \mapsto \sigma'(x) + 1]) \rightarrow \sigma''}{(\text{for } x \text{ from } a_1 \text{ to } a_2 \text{ do } S, \sigma) \rightarrow \sigma''}$$

   (b) We assume that we can use intermediate syntactic structures to define the recursion on:
   We assume that we have an intermediate syntactic structure 'for x to a_2 do S' defined as below.

$$\frac{(\text{for } x \text{ to } a_2 \text{ do } S, \sigma[x \mapsto \mathcal{A}[a_1]\sigma]) \rightarrow \sigma'}{(\text{for } x \text{ from } a_1 \text{ to } a_2 \text{ do } S, \sigma) \rightarrow \sigma'}$$

   If $\sigma(x) > \mathcal{A}[a_2]\sigma$:

$$\frac{}{(\text{for } x \text{ to } a_2 \text{ do } S, \sigma) \rightarrow \sigma}$$

   Otherwise:

$$\frac{(S, \sigma) \rightarrow \sigma' \qquad (\text{for } x \text{ to } a_2 \text{ do } S, \sigma'[x \mapsto \sigma'(x) + 1]) \rightarrow \sigma''}{(\text{for } x \text{ to } a_2 \text{ do } S, \sigma) \rightarrow \sigma''}$$

   (c) We assume the function $\mathcal{N}^{-1}$ as suggested in the exercise:
   If $\mathcal{A}[a_1]\sigma > \mathcal{A}[a_2]\sigma$:

$$\frac{}{(\text{for } x \text{ from } a_1 \text{ to } a_2 \text{ do } S, \sigma) \rightarrow \sigma[x \mapsto \mathcal{A}[a_1]\sigma]}$$

Otherwise:

$$\frac{(S, \sigma[x \mapsto]) \to \sigma' \qquad (\textsf{for } x \textsf{ from } \mathcal{N}^{-1}(\mathcal{A}[a_1]\sigma + 1) \textsf{ to } a_2 \textsf{ do } S, \sigma') \to \sigma''}{(\textsf{for } x \textsf{ from } a_1 \textsf{ to } a_2 \textsf{ do } S, \sigma) \to \sigma''}$$

3. (a) Using the first mechanism as described above:
$\sigma_0 := [x \mapsto 5]$
$S_0 ::= y := 1$
$S_1 ::= \textsf{for } z \textsf{ from } 1 \textsf{ to } x \textsf{ do } S_2$
$S_2 ::= S_{20}; S_{21}$
$S_{20} ::= y := y + x$
$S_{21} ::= x := x - 1$

$$\dfrac{\dfrac{(S_0, \sigma_0) \to \sigma_1 \qquad \dfrac{\dfrac{(S_2, \sigma_2) \to \sigma_3 \qquad \dfrac{\dfrac{(S_2, \sigma_4) \to \sigma_5 \qquad \dfrac{\dfrac{(S_2, \sigma_6) \to \sigma_7 \qquad \overline{(S_1, \sigma_8) \to \sigma_9}}{(S_1, \sigma_6) \to \sigma_9}}{(S_1, \sigma_4) \to \sigma_9}}{(S_1, \sigma_2) \to \sigma_9}}{(S_1, \sigma_1) \to \sigma_9}}{(S_0; S_1, \sigma_0) \to \sigma_9}$$

With:
$\sigma_1 = \sigma_0[y \mapsto 1] = [x \mapsto 5, y \mapsto 1]$
$\sigma_2 = \sigma_1[z \mapsto 1][v \mapsto 0] = [x \mapsto 5, y \mapsto 1, z \mapsto 1, v \mapsto 0]$
$\sigma_3 = \sigma_2[y \mapsto 1 + 5][x \mapsto 5 - 1] = [x \mapsto 4, y \mapsto 6, z \mapsto 1, v \mapsto 0]$
$\sigma_4 = \sigma_3[z \mapsto 1 + 1] = [x \mapsto 4, y \mapsto 6, z \mapsto 2, v \mapsto 0]$
$\sigma_5 = \sigma_4[y \mapsto 6 + 4][x \mapsto 4 - 1] = [x \mapsto 3, y \mapsto 10, z \mapsto 2, v \mapsto 0]$
$\sigma_6 = \sigma_5[z \mapsto 2 + 1] = [x \mapsto 3, y \mapsto 10, z \mapsto 3, v \mapsto 0]$
$\sigma_7 = \sigma_6[y \mapsto 10 + 3][x \mapsto 3 - 1] = [x \mapsto 2, y \mapsto 13, z \mapsto 3, v \mapsto 0]$
$\sigma_8 = \sigma_7[z \mapsto 3 + 1] = [x \mapsto 2, y \mapsto 13, z \mapsto 4, v \mapsto 0]$
$\sigma_9 = \sigma_8[v \mapsto undef] = [x \mapsto 2, y \mapsto 13, z \mapsto 4]$
The $undef$ function in σ_1 gives v as an undefined variable.
The for-loop continues with σ_2 because $\sigma_2(z) = 1 \leq \mathcal{A}[x]\sigma_2 = 5$.
The for-loop continues with σ_4 because $\sigma_4(z) = 2 \leq \mathcal{A}[x]\sigma_4 = 4$.
The for-loop continues with σ_6 because $\sigma_6(z) = 3 \leq \mathcal{A}[x]\sigma_6 = 3$.
The for-loop stops with σ_8 because $\sigma_8(z) = 4 > \mathcal{A}[x]\sigma_8 = 2$.

(b) Using the last mechanism as described above:
$\sigma_0 := [x \mapsto 5]$
$S_0 ::= y := 1$
$S_1 ::= \textsf{for } z \textsf{ from } 1 \textsf{ to } x \textsf{ do } S_2$
$S_2 ::= S_{20}; S_{21}$
$S_{20} ::= y := y + x$
$S_{21} ::= x := x - 1$

$$\dfrac{(S_0,\sigma_0)\to\sigma_1 \qquad \dfrac{(S_2,\sigma_2)\to\sigma_3 \qquad \dfrac{(S_2,\sigma_4)\to\sigma_5 \qquad \dfrac{(S_2,\sigma_6)\to\sigma_7 \qquad \overline{(S_{1_3},\sigma_7)\to\sigma_8}}{(S_{1_2},\sigma_5)\to\sigma_8}}{(S_{1_1},\sigma_3)\to\sigma_8}}{(S_1,\sigma_1)\to\sigma_8}}{(S_0;S_1,\sigma_0)\to\sigma_8}$$

With:
$\sigma_1 = \sigma_0[y \mapsto 1] = [x \mapsto 5, y \mapsto 1]$
$\sigma_2 = \sigma_1[z \mapsto 1] = [x \mapsto 5, y \mapsto 1, z \mapsto 1]$
$\sigma_3 = \sigma_2[y \mapsto 1+5][x \mapsto 5-1] = [x \mapsto 4, y \mapsto 6, z \mapsto 1]$
$S_{1_1} ::=$ for z from $\mathcal{N}^{-1}(\mathcal{A}[1]\sigma_1 + 1)$ to x do $S_2 =$ for z from 2 to x do S_2
$\sigma_4 = \sigma_3[z \mapsto 2] = [x \mapsto 4, y \mapsto 6, z \mapsto 2]$
$\sigma_5 = \sigma_4[y \mapsto 6+4][x \mapsto 4-1] = [x \mapsto 3, y \mapsto 10, z \mapsto 2]$
$S_{1_2} ::=$ for z from $\mathcal{N}^{-1}(\mathcal{A}[2]\sigma_3 + 1)$ to x do $S_2 =$ for z from 3 to x do S_2
$\sigma_6 = \sigma_5[z \mapsto 3] = [x \mapsto 3, y \mapsto 10, z \mapsto 3]$
$\sigma_7 = \sigma_6[y \mapsto 10+3][x \mapsto 3-1] = [x \mapsto 2, y \mapsto 13, z \mapsto 3]$
$S_{1_3} ::=$ for z from $\mathcal{N}^{-1}(\mathcal{A}[3]\sigma_5 + 1)$ to x do $S_2 =$ for z from 4 to x do S_2
$\sigma_8 = \sigma_7[z \mapsto 4] = [x \mapsto 2, y \mapsto 13, z \mapsto 4]$
With $S_1;\sigma_1$: $\mathcal{A}[1]\sigma_1 = 1 \leq \mathcal{A}[x]\sigma_1 = 5 \Rightarrow$ continue.
With $S_{1_1};\sigma_3$: $\mathcal{A}[2]\sigma_3 = 2 \leq \mathcal{A}[x]\sigma_3 = 4 \Rightarrow$ continue.
With $S_{1_2};\sigma_5$: $\mathcal{A}[3]\sigma_5 = 3 \leq \mathcal{A}[x]\sigma_5 = 3 \Rightarrow$ continue.
With $S_{1_3};\sigma_7$: $\mathcal{A}[4]\sigma_7 = 4 > \mathcal{A}[x]\sigma_7 = 2 \Rightarrow$ stop.

5 Series 5

5.1 Exercise 47

Exercise was not done in class !

$\sigma_0 := [x \mapsto 0]$ this does not make sense, should be: $\hat{\rho_0} := [x \mapsto l_0]; \sigma_0 := [l_0 \mapsto 0]$
$D_{V_0} ::=$ var $y := 1$
$S_1 ::= (S_{10}; S_{11}; S_{12})$
$S_{10} ::= x := 1$
$S_{11} ::=$ begin $D_{V_1}; S_{111}$ end
$D_{V_1} ::=$ var $x := 2$
$S_{111} ::= y := x + 1$
$S_{12} ::= x := y + x$

Block rule:

$$\frac{\dfrac{(\epsilon, \hat{\rho}_0, \rho_1, \sigma_1) \to (\rho_1, \sigma_1)}{(D_{V_0}, \hat{\rho}_0, \emptyset, \sigma_0) \to (\rho_1, \sigma_1)} \qquad (S_1, \hat{\rho}_0 \oplus \rho_1, \sigma_1) \to \sigma_5}{(\textsf{begin } D_{V_0} S_1 \textsf{ end}, \hat{\rho}_0, \sigma_0) \to \sigma_5}$$

Sequential composition:

$$\frac{(S_{10}, \hat{\rho}_0 \oplus \rho_1, \sigma_1) \to \sigma_2 \qquad \dfrac{(S_{11}, \hat{\rho}_0 \oplus \rho_1, \sigma_2) \to \sigma_4 \qquad (S_{12}, \hat{\rho}_0 \oplus \rho_1, \sigma_4) \to \sigma_5}{(S_{11}; S_{12}, \hat{\rho}_0 \oplus \rho_1, \sigma_2) \to \sigma_5}}{(S_1, \hat{\rho}_0 \oplus \rho_1, \sigma_1) \to \sigma_5}$$

Inner block:

$$\frac{(D_{V_1}, \hat{\rho}_0 \oplus \rho_1, \emptyset, \sigma_2) \to (\rho_2, \sigma_3) \qquad (S_{111}, \hat{\rho}_0 \oplus \rho_1 \oplus \rho_2, \sigma_3) \to \sigma_4}{(\textsf{begin } D_{V_1}; S_{111} \textsf{ end}, \hat{\rho}_0 \oplus \rho_1, \sigma_2) \to \sigma_4}$$

With:

$\rho_1 = \emptyset[y \mapsto l_1] = [y \mapsto l_1]$
$\sigma_1 = \sigma_0[l_1 \mapsto 1] = [l_0 \mapsto 0, l_1 \mapsto 1]$
$\sigma_2 = \sigma_1[l_0 \mapsto 1] = [l_0 \mapsto 1, l_1 \mapsto 1]$
$\rho_2 = \emptyset[x \mapsto l_2] = [x \mapsto l_2]$
$\sigma_3 = \sigma_2[l_2 \mapsto 2] = [l_0 \mapsto 1, l_1 \mapsto 1, l_2 \mapsto 2]$
$\sigma_4 = \sigma_3[l_1 \mapsto 2 + 1] = [l_0 \mapsto 1, l_1 \mapsto 3, l_2 \mapsto 2]$
$\sigma_5 = \sigma_4[l_0 \mapsto 3 + 1] = [l_0 \mapsto 4, l_1 \mapsto 3, l_2 \mapsto 2]$

6 Series 9

6.1 Exercise 86

1. Def : **Stm** $\cup$ **Aexp** $\cup$ **Bexp** $\to 2^{\textbf{Var}}$; Use : **Stm** $\cup$ **Aexp** $\cup$ **Bexp** $\to 2^{\textbf{Var}}$
2. $Def(a) = \emptyset$ $\quad \forall a \in$ **Aexp**
   $Def(b) = \emptyset$ $\quad \forall b \in$ **Bexp**
   $Def(x := a) = \{x\}$
   $Def(\textsf{skip}) = \emptyset$
   $Def(S_1; S_2) = Def(S_1) \cup Def(S_2)$
   $Def(\textsf{if } b \textsf{ then } S_1 \textsf{ else } S_2) = Def(S_1) \cap Def(S_2)$
   $Def(\textsf{while } b \textsf{ do } S_1) = \emptyset$

$Use(n) = \emptyset$
$Use(x) = \{x\}$
$Use(a_1 + a_2) = Use(a_1) \cup Use(a_2)$
$Use(\text{true}) = \emptyset$
$Use(\text{false}) = \emptyset$
$Use(a_1 = a_2) = Use(a_1) \cup Use(a_2)$
$Use(a_1 \leq a_2) = Use(a_1) \cup Use(a_2)$
$Use(\neg b) = Use(b)$
$Use(b_1 \wedge b_2) = Use(b_1) \cup Use(b_2)$
$Use(x := a) = Use(a)$
$Use(\text{skip}) = \emptyset$
$Use(S_1; S_2) = Use(S_1) \cup Use(S_2)$
$Use(\text{if } b \text{ then } S_1 \text{ else } S_2) = Use(b) \cup Use(S_1) \cup Use(S_2)$
$Use(\text{while } b \text{ do } S_1) = Use(b) \cup Use(S_1)$

6.2 Exercise 87

1.

$Def(\text{if } a < b \text{ then } c := d - y \text{ else } y := e - x) = Def(c := d - y) \cap Def(y := e - x)$
$= \{c\} \cap \{y\} = \emptyset$

$Use(\text{if } a < b \text{ then } c := d - y \text{ else } y := e - x) = Use(c := d - y) \cup Def(y := e - x)$
$= Use(a < b) \cup Use(d - y) \cup Use(e - x)$
$= Use(a) \cup Use(b) \cup Use(d) \cup Use(y) \cup Use(e) \cup Use(x)$
$= \{a\} \cup \{b\} \cup \{d\} \cup \{y\} \cup \{e\} \cup \{x\}$
$= \{a, b, d, e, x, y\}$

6.3 Exercise 88

Natural operational semantics for the parrallel operator $S_1 \parallel S_2$: Non-determinism between the following two statements

$$\frac{(S_1, \sigma) \to \sigma_1 \qquad (S_2, \sigma_1) \to \sigma_2}{(S_1 \parallel S_2, \sigma) \to \sigma_2}$$

and

$$\frac{(S_2, \sigma) \to \sigma_1 \qquad (S_1, \sigma_1) \to \sigma_2}{(S_1 \parallel S_2, \sigma) \to \sigma_2}$$

6.4 Exercise 89

1. $$\frac{(x := x + 1, \sigma_0) \to \sigma_1 \qquad (y := y + 1, \sigma_1) \to \sigma_2}{(x := x + 1 \parallel y := y + 1, \sigma_0) \to \sigma_2}$$

With $\sigma_1 = \sigma_0[x \mapsto 1 + 1] = [x \mapsto 2, y \mapsto 1]$ and $\sigma_2 = \sigma_1[y \mapsto 1 + 1] = [x \mapsto 2, y \mapsto 2]$.
or

$$\frac{(y := y + 1, \sigma) \to \sigma_1 \qquad (x := x + 1, \sigma_1) \to \sigma_2}{(x := x + 1 \parallel y := y + 1, \sigma) \to \sigma_2}$$

With $\sigma_1 = \sigma_0[y \mapsto 1 + 1] = [x \mapsto 1, y \mapsto 2]$ and $\sigma_2 = \sigma_1[x \mapsto 1 + 1] = [x \mapsto 2, y \mapsto 2]$.

2. $$\frac{(x := y + 1, \sigma_0) \to \sigma_1 \qquad (y := x + 2, \sigma_1) \to \sigma_2}{(x := y + 1 \parallel y := x + 1, \sigma_0) \to \sigma_2}$$

With $\sigma_1 = \sigma_0[x \mapsto 1 + 1] = [x \mapsto 2, y \mapsto 1]$ and $\sigma_2 = \sigma_1[y \mapsto 2 + 2] = [x \mapsto 2, y \mapsto 4]$.
or

$$\frac{(y := x + 2, \sigma) \to \sigma_1 \qquad (x := y + 1, \sigma_1) \to \sigma_2}{(x := y + 1 \parallel y := x + 2, \sigma) \to \sigma_2}$$

With $\sigma_1 = \sigma_0[y \mapsto 1 + 2] = [x \mapsto 1, y \mapsto 3]$ and $\sigma_2 = \sigma_1[x \mapsto 3 + 1] = [x \mapsto 4, y \mapsto 3]$.

6.5 Exercise 90

If $Def(S_1) \cap Def(S_2) = \emptyset \wedge Use(S_1) \cap Def(S_2) = \emptyset \wedge Use(S_2) \cap Def(S_1) = \emptyset$, then we can be sure that the obtained state σ_2 from executing $S_1 \parallel S_2$ is independent of the order of execution of S_1 and S_2.

6.6 Exercise 91

If $Def(S_1) \cap Def(S_2) = \emptyset \wedge Use(S_1) \cap Def(S_2) = \emptyset \wedge Use(S_2) \cap Def(S_1) = \emptyset$:

$$\frac{(S_1, \sigma) \to \sigma_1 \qquad (S_2, \sigma) \to \sigma_2}{(S_1 \parallel S_2, \sigma) \to \sigma_3}$$

With $\sigma_3(x) = \begin{cases} \sigma_1(x) & if x \in Def(S_1) \\ \sigma_2(x) & if x \in Def(S_2) \\ \sigma(x) & otherwise \end{cases}$.

6.7 Exercise 92

1. $$\frac{(x := x + 1, \sigma_0) \to \sigma_1 \qquad (y := y + 1, \sigma_0) \to \sigma_2}{(x := x + 1 \parallel y := y + 1, \sigma_0) \to \sigma_3}$$

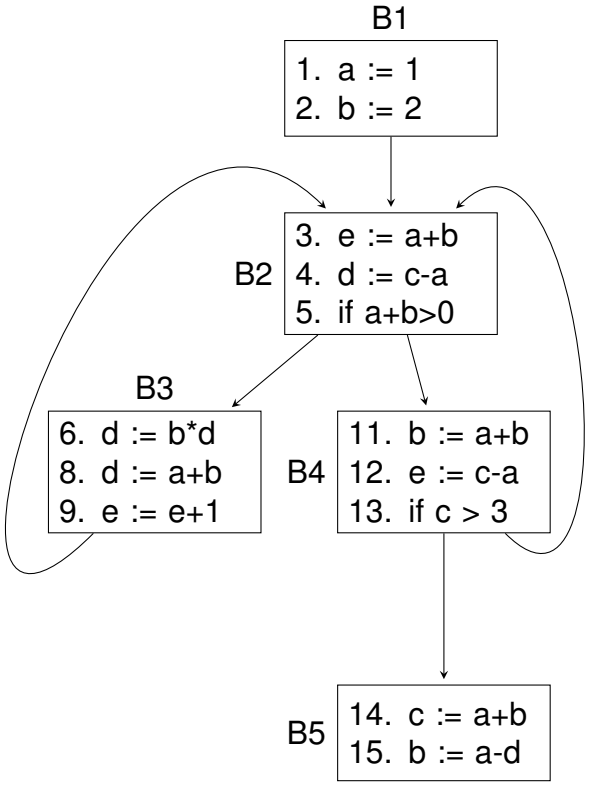
   With:

$$\sigma_1 = \sigma_0[x \mapsto 1 + 1] = [x \mapsto 2, y \mapsto 1]$$
$$\sigma_2 = \sigma_0[y \mapsto 1 + 1] = [x \mapsto 1, y \mapsto 2]$$
$$\sigma_3 = \begin{cases} \sigma_1(x) & if x \in Def(S_1) = \{x\} \\ \sigma_2(x) & if x \in Def(S_2) = \{y\} \Rightarrow \sigma_3 = [x \mapsto 2, y \mapsto 2] \\ \sigma_0(x) & otherwise \end{cases}$$

2. Cannot construct the derivation tree because $Use(x := y + 1) \cap Def(y := x + 2) = \{y\} \neq \emptyset$.

7 Series 10

7.1 Exercise 96



7.2 Exercise 97

1. $In(B) = \begin{cases} \emptyset & \text{if } B = B1 \text{ in first step} \\ \Sigma & \text{if } B \neq B1 \text{ in first step} \\ \bigcap_{B' \in pred(B)} Out(B') & \text{with } Out(B') \text{ from last step} \end{cases}$

   $Out(B) = (In(B) \setminus Kill(B)) \cup Gen(B)$

   $\Sigma = \{\overbrace{a+b}^{e_1}, \overbrace{c-a}^{e_2}, \overbrace{b*d}^{e_3}, \overbrace{e+1}^{e_4}, \overbrace{a-d}^{e_5}\}$

2.

B	$Pred(B)$	$Gen(B)$	$Kill(B)$	$In(B)$	$Out(B)$	$In(B)$	$Out(B)$
B1	$\emptyset$	$\emptyset$	e_1, e_2, e_3, e_5	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
B2	B1, B3, B4	e_1, e_2	e_3, e_4, e_5	Σ	e_1, e_2	$\emptyset$	e_1, e_2
B3	B2	e_1	e_3, e_4, e_5	Σ	e_1, e_2	e_1, e_2	e_1, e_2
B4	B2	e_2	e_1, e_3, e_4	Σ	e_2, e_5	e_1, e_2	e_2
B5	B4	e_5	e_1, e_2, e_3	Σ	e_4, e_5	e_2, e_5	e_5

B	$In(B)$	$Out(B)$
B1	$\emptyset$	$\emptyset$
B2	$\emptyset$	e_1, e_2
B3	e_1, e_2	e_1, e_2
B4	e_1, e_2	e_2
B5	e_2	e_5

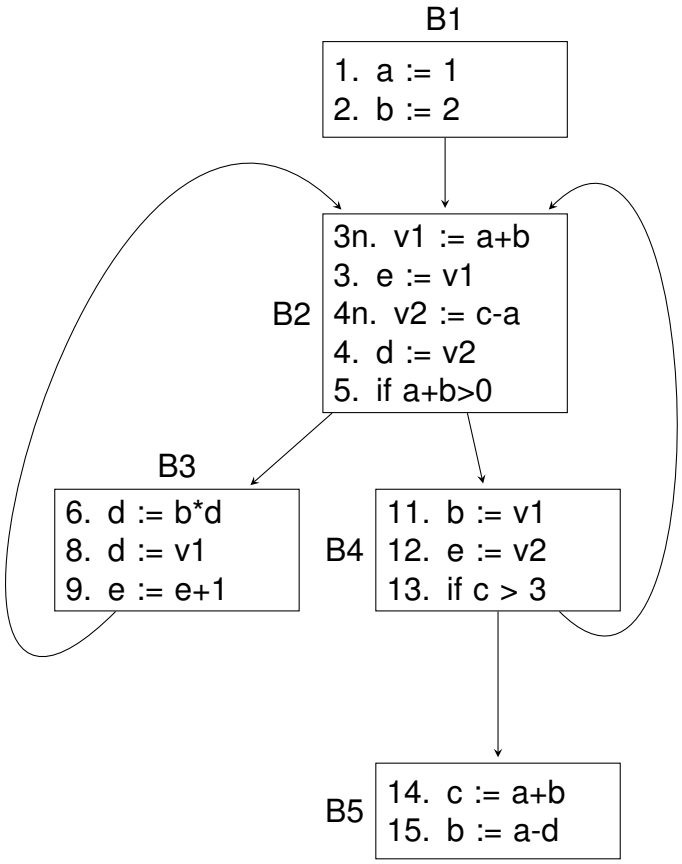
   From this table, we can see the fix point for $In(B)$ as follows:
   $In(B1) = \emptyset$
   $In(B2) = \emptyset$
   $In(B3) = \{e_1, e_2\}$
   $In(B4) = \{e_1, e_2\}$
   $In(B5) = \{e_2\}$

3. - Traverse all blocks of the CFG starting from the block with no predecessor and going in a breadth-first way.
   - Check if any of the available expressions from the fix-point are used before their variables are assigned in that block. If this is the case, the expression can be replaced by a new variable.
   - This variable is assigned by backtracking to the last use of the expression, then replacing the expression by the new variable both when it is first computed and in the block. If the last use of the expression has already been replaced by a new variable, then this variable can simply be used to replace the expression again in the block.

   Here, for B1 and B2, we don't have any available expressions in the fix-point. For B3: e_1 is used and a and b are not assigned before the use of e_1. So we can replace this occurence of e_1 by a new variable v_1, which is assigned before statement 3. We also need to update the use of e_1 in statement 3.

In B3, e_2 is not used.
For B4: e_1 is used before a and b are assigned, so this can be replaced by the same variable v_1 than above.
e_2 is used before c and a are assigned so we can replace e_2 by a new variable v_2, which is assigned before statement 4. The uses in statement 4. and 12. are then replaced by that variable.
For B5: e_2 is not used.

This gives the following new CFG:



7.3 Exercise 98

7.3.1 Left CFG

1. $In(B) = \begin{cases} \emptyset & \text{if } B = B1 \text{ in first step} \\ \Sigma & \text{if } B \neq B1 \text{ in first step} \\ \bigcap_{B' \in pred(B)} Out(B') & \text{with } Out(B') \text{ from last step} \end{cases}$
   $Out(B) = (In(B) \setminus Kill(B)) \cup Gen(B)$
   $\Sigma = \{e_1, e_2, e_3, e_5, e_6, e_7\}$

2.

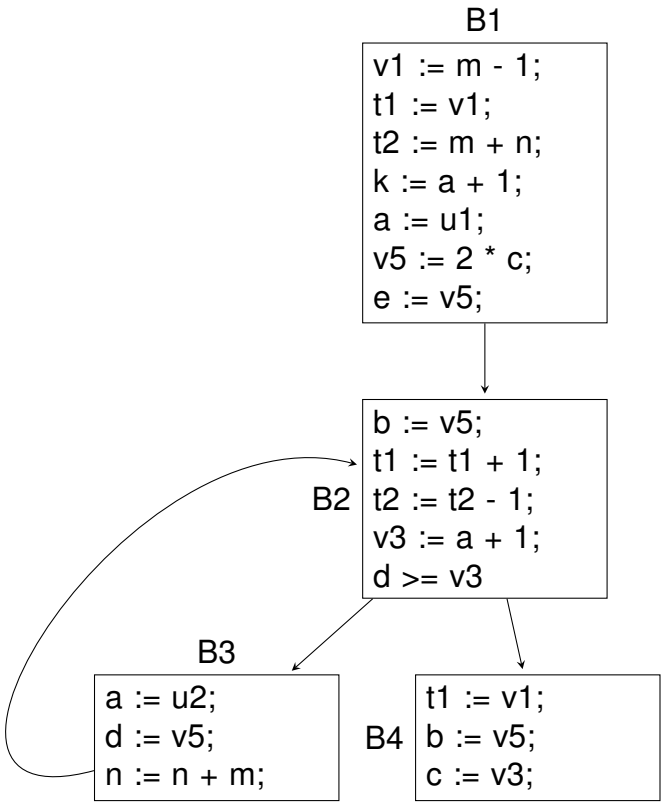
B	$Pred(B)$	$Gen(B)$	$Kill(B)$	$In(B)$	$Out(B)$	$In(B)$	$Out(B)$
B1	$\emptyset$	e_1, e_2, e_5	e_3, e_6, e_7	$\emptyset$	e_1, e_2, e_5	$\emptyset$	e_1, e_2, e_5
B2	B1, B3	e_3, e_5	e_6, e_7	Σ	e_1, e_2, e_3, e_5	e_1, e_5	e_1, e_3, e_5
B3	B2	e_5	e_2, e_3	Σ	e_1, e_5, e_6, e_7	e_1, e_2, e_3, e_5	e_1, e_5
B4	B2	e_1, e_3	e_5, e_6	Σ	e_1, e_2, e_3, e_7	e_1, e_2, e_3, e_5	e_1, e_2, e_3

B	$In(B)$	$Out(B)$	$In(B)$	$Out(B)$
B1	$\emptyset$	e_1, e_2, e_5	$\emptyset$	
B2	e_1, e_5	e_1, e_3, e_5	e_1, e_5	
B3	e_1, e_3, e_5	e_1, e_5	e_1, e_3, e_5	
B4	e_1, e_3, e_5	e_1, e_3	e_1, e_3, e_5	

From this table, we can see the fix point for $In(B)$ as follows:
$In(B1) = \emptyset$
$In(B2) = \{e_1, e_5\}$
$In(B3) = \{e_1, e_3, e_5\}$
$In(B4) = \{e_1, e_3, e_5\}$

3. Using the algorithm to replace available expressions, we get:
For B1: No expressions can be replaced because none are available from the fix-point.
For B2: e_5 can be replaced by a new variable v_5 (because c is not assigned before use of e_5).
For B3: e_5 can be replaced by the variable v_5 (because c is not assigned before use of e_5 and last use of e_5 was replaced by v_5).
For B4: e_1 can be replaced by a new variable v_1 (because m is not assigned before use of e_1).
For B4: e_3 can be replaced by a new variable v_3 (because a is not assigned before use of e_3).
For B4: e_5 can be replaced by v_5 (because c is not assigned before use of e_5 and last use of e_5 was replaced by v_5).

This gives the following new CFG:



7.3.2 Right CFG

This part of the exercise was not done in class !

1. $In(B) = \begin{cases} \emptyset & \text{if } B = B1 \text{ in first step} \\ \Sigma & \text{if } B \neq B1 \text{ in first step} \\ \bigcap_{B' \in pred(B)} Out(B') & \text{with } Out(B') \text{ from last step} \end{cases}$

   $Out(B) = (In(B) \setminus Kill(B)) \cup Gen(B)$

   $\Sigma = \{\overbrace{m-1}^{e_1}, \overbrace{4*n}^{e_2}, \overbrace{a[t1]}^{e_3}, \overbrace{i+1}^{e_4}, \overbrace{4*i}^{e_5}, \overbrace{a[t2]}^{e_6}, \overbrace{j-1}^{e_7}, \overbrace{a[t4]}^{e_8}\}$

2.

B	$Pred(B)$	$Gen(B)$	$Kill(B)$	$In(B)$	$Out(B)$	$In(B)$	$Out(B)$
B1	$\emptyset$	e_1, e_2, e_3	e_3, e_4, e_5, e_7	$\emptyset$	e_1, e_2, e_3	$\emptyset$	e_1, e_2, e_3
B2	B1, B3, B4	e_3, e_5, e_6	e_4, e_5, e_6	Σ	$\Sigma \setminus \{e_4\}$	e_1, e_2, e_3	e_1, e_2, e_3, e_5, e_6
B3	B2	e_1, e_7	e_7	Σ	Σ	$\Sigma \setminus \{e_4\}$	$\Sigma \setminus \{e_4\}$
B4	B3	e_1, e_3, e_5	$\emptyset$	Σ	Σ	Σ	Σ
B5	B3	e_5, e_8	$\emptyset$	Σ	Σ	Σ	Σ

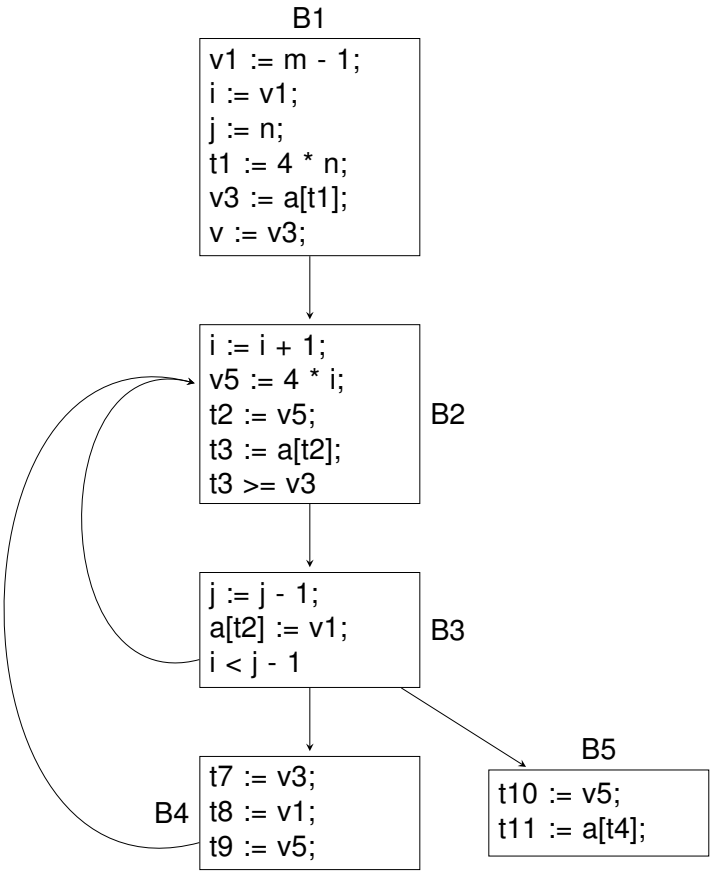
B	$In(B)$	$Out(B)$	$In(B)$	$Out(B)$	$In(B)$
B1	$\emptyset$	e_1, e_2, e_3	$\emptyset$	e_1, e_2, e_3	$\emptyset$
B2	e_1, e_2, e_3	e_1, e_2, e_3, e_5, e_6	e_1, e_2, e_3	e_1, e_2, e_3, e_5, e_6	e_1, e_2, e_3
B3	e_1, e_2, e_3, e_5, e_6	$\Sigma \setminus \{e_4, e_8\}$	e_1, e_2, e_3, e_5, e_6	$\Sigma \setminus \{e_4, e_8\}$	e_1, e_2, e_3, e_5, e_6
B4	$\Sigma \setminus \{e_4\}$	$\Sigma \setminus \{e_4\}$	$\Sigma \setminus \{e_4, e_8\}$	$\Sigma \setminus \{e_4, e_8\}$	$\Sigma \setminus \{e_4, e_8\}$
B5	$\Sigma \setminus \{e_4\}$	$\Sigma \setminus \{e_4\}$	$\Sigma \setminus \{e_4, e_8\}$	$\Sigma \setminus \{e_4\}$	$\Sigma \setminus \{e_4, e_8\}$

From this table, we can see the fix point for $In(B)$ as follows:
$In(B1) = \emptyset$
$In(B2) = \{e_1, e_2, e_3\}$
$In(B3) = \{e_1, e_2, e_3, e_5, e_6\}$
$In(B4) = \{e_1, e_2, e_3, e_5, e_6, e_7\}$
$In(B5) = \{e_1, e_2, e_3, e_5, e_6, e_7\}$

3. Using the algorithm to replace available expressions, we get:
   For B1: No expressions can be replaced.
   For B2: e_3 can be replaced by a new variable v_3 (because t1 is not assigned before use of e_3).
   For B3: e_1 can be replaced by a new variable v_1 (because m is not assigned before use of e_1).
   For B4: e_1 can be replaced by the variable v_1 (because m is not assigned before use of e_1 and last use of e_1 was replaced by v_1).
   For B4: e_3 can be replaced by the variable v_3 (because t1 is not assigned before use of e_3 and last use of e_3 was replaced by v_3).
   For B4: e_5 can be replaced by a new variable v_5 (because i is not assigned before use of e_5).
   For B5: e_5 can be replaced by the variable v_5 (because i is not assigned before use of e_5 and last use of e_5 was replaced by v_5).

   We note that e_6=a[t2] is available in B3, which could be used in the assignment 'a[t2] = m - 1' (where a[t2] first has to be calculated before making the assignment) but we did not see how to replace array locations in the lecture. The basic idea would be to replace a[t2] by l6 and then use this location in the assignment instead of a[t2]. I did not add this here.

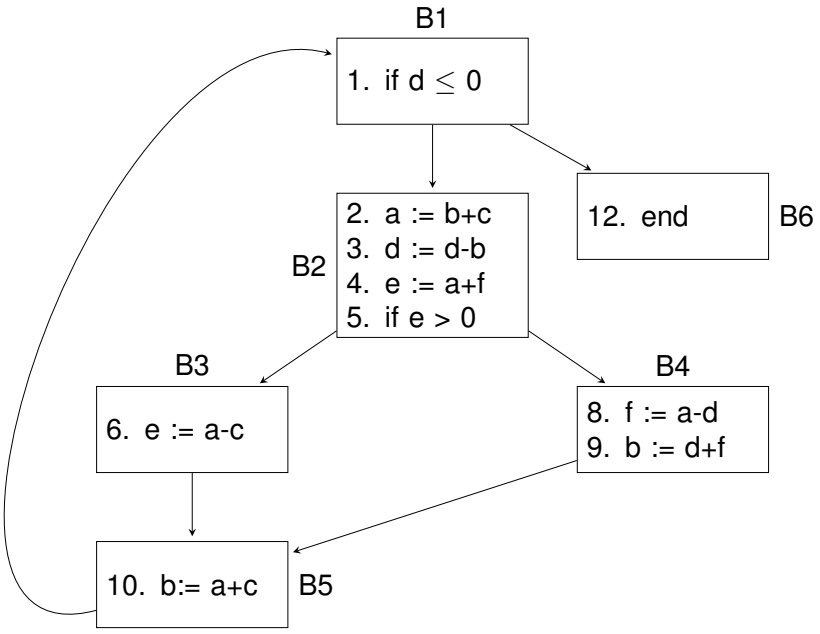
   This gives the following new CFG:



7.4 Exercise 100

1.
   1. if d $\leq$ 0 goto 12
   2. a := b+c
   3. d := d-b
   4. e := a+f
   5. if e > 0 goto 8
   6. e := a-c
   7. goto 10
   8. f := a-d
   9. b := d+f
   10. b:= a+c
   11. goto 1
   12. end

2. From the 3-address code above, we generate the following CFG:



3. $\Sigma = \{a, b, c, d, e, f\}$
   $In(B) = (Out(B) \setminus Kill(B)) \cup Gen(B)$
   $Out(B) = \begin{cases} \emptyset & \text{in first step} \\ \bigcup_{B' \in Succ(B)} In(B') & \text{with } In(B') \text{ from last step} \end{cases}$

4.

B	$Succ(B)$	$Gen(B)$	$Kill(B)$	$Out(B)$	$In(B)$	$Out(B)$	$In(B)$
B1	B2, B6	d	$\emptyset$	$\emptyset$	d	b,c,d,f	b,c,d,f
B2	B3, B4	b,c,d,f	a,d,e	$\emptyset$	b,c,d,f	a,c,d	b,c,d,f
B3	B5	a,c	e	$\emptyset$	a,c	a,c	a,c
B4	B5	a,d	b,f	$\emptyset$	a,d	a,c	a,c,d
B5	B1	a,c	b	$\emptyset$	a,c	d	a,c,d
B6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

B	$Out(B)$	$In(B)$	$Out(B)$	$In(B)$	$Out(B)$	$In(B)$	$Out(B)$
B1	b,c,d,f	b,c,d,f	b,c,d,f	b,c,d,f	b,c,d,f	b,c,d,f	b,c,d,f
B2	a,c,d	b,c,d,f	a,c,d	b,c,d,f	a,c,d,f	b,c,d,f	a,c,d,f
B3	a,c,d	a,c,d	a,c,d,f	a,c,d,f	a,c,d,f	a,c,d,f	a,c,d,f
B4	a,c,d	a,c,d	a,c,d,f	a,c,d	a,c,d,f	a,c,d	a,c,d,f
B5	b,c,d,f	a,c,d,f	b,c,d,f	a,c,d,f	b,c,d,f	a,c,d,f	b,c,d,f
B6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

From this table, we can see the fix point for $Out(B)$ as follows:
$Out(B1) = \{b, c, d, f\}$
$Out(B2) = \{a, c, d, f\}$
$Out(B3) = \{a, c, d, f\}$
$Out(B4) = \{a, c, d, f\}$

$Out(B5) = \{b, c, d, f\}$
$Out(B6) = \emptyset$

5. The algorithm goes as follows:

   - Start with the end of a block and its corresponding out-set of active variables $Out(B)$.
   - Go through each statement of the block from last to first (and keep track of the set of active variables for each statement).
   - For each statement, if an assigned variable is not active, suppress the statement. Then, remove the assigned variables and afterwards add all used variables to the set of active variables.

   For B1 and B6, nothing is assigned so nothing can be suppressed.

   For B2:

Statement S	$Out(S)$	$Out_{new}(S)$	Remove?
5. if e > 0	$Out(B2)$	a,c,d,e,f	No because nothing is assigned
4. e := a+f	a,c,d,e,f	a,c,d,f	No because e is active
3. d := d-b	a,c,d,f	a,b,c,d,f	No because d is active
2. a := b+c	a,b,c,d,f	b,c,d,f	No because a is active

   For B3:

Statement S	$Out(S)$	$Out_{new}(S)$	Remove?
6. e := a-c	$Out(B3)$	a,c,d,f	Yes (e is not active after this)

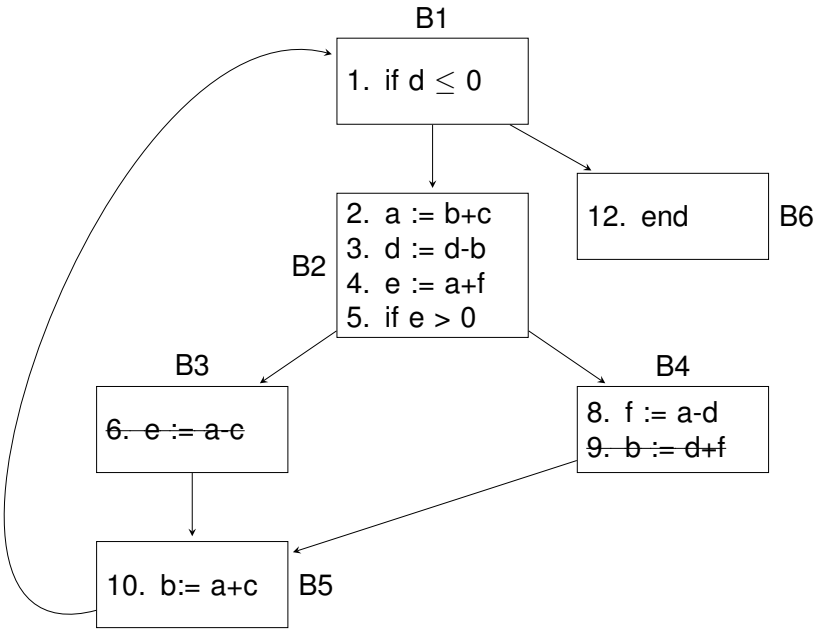
   For B4:

Statement S	$Out(S)$	$Out_{new}(S)$	Remove?
9. b := d+f	$Out(B4)$	a,c,d,f	Yes (b is not active after this)
8. f := a-d	a,c,d,f	a,c,d	No because f is active

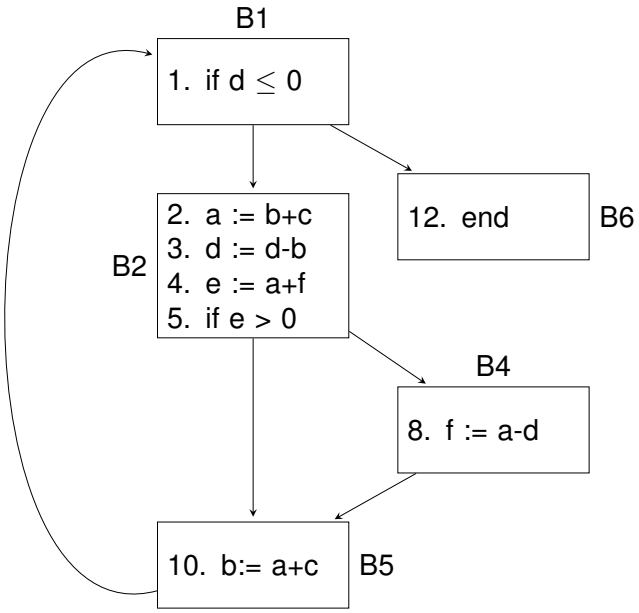
   For B5:

Statement S	$Out(S)$	$Out_{new}(S)$	Remove?
10. b := a+c	$Out(B5)$	a,c,d,f	No, because b is active

   Suppressing the statements updates the CFG as follows:



This modifies the CFG into the following new CFG:



7.5 Exercise 101

7.5.1 Left CFG

1. See Exercise 100 above

2.

B	$Succ(B)$	$Gen(B)$	$Kill(B)$	$Out(B)$	$In(B)$	$Out(B)$	$In(B)$
B1	B2	u1, m, n	a, j, i	$\emptyset$	u1, m, n	j, i	u1, m, n
B2	B3, B4	j, i	j, i	$\emptyset$	j, i	u2, u3	u2, u3, j, i
B3	B2	u2	a	$\emptyset$	u2	j, i	j, i, u2
B4	B2	u3	i	$\emptyset$	u3	j, i	j, u3

B	$Out(B)$	$In(B)$	$Out(B)$
B1	u2, u3, j, i	u1, u2, u3, m, n	u2, u3, j, i
B2	j, i, u2, u3	u2, u3, j, i	u2, u3, j, i
B3	u2, u3, j, i	u2, u3, j, i	u2, u3, j, i
B4	u2, u3, j, i	u2, u3, j	u2, u3, j, i

From this table, we can see the fix point for $Out(B)$ as follows:
$Out(B1) = \{u2, u3, j, i\}$
$Out(B2) = \{u2, u3, j, i\}$
$Out(B3) = \{u2, u3, j, i\}$
$Out(B4) = \{u2, u3, j, i\}$

3. For B2:

Statement S	$Out(S)$	$Out_{new}(S)$	Remove?
j := j-1	$Out(B2)$	$Out(B2)$	No because j is active
i := i+1	$Out(B2)$	$Out(B2)$	No because i is active

For B3:

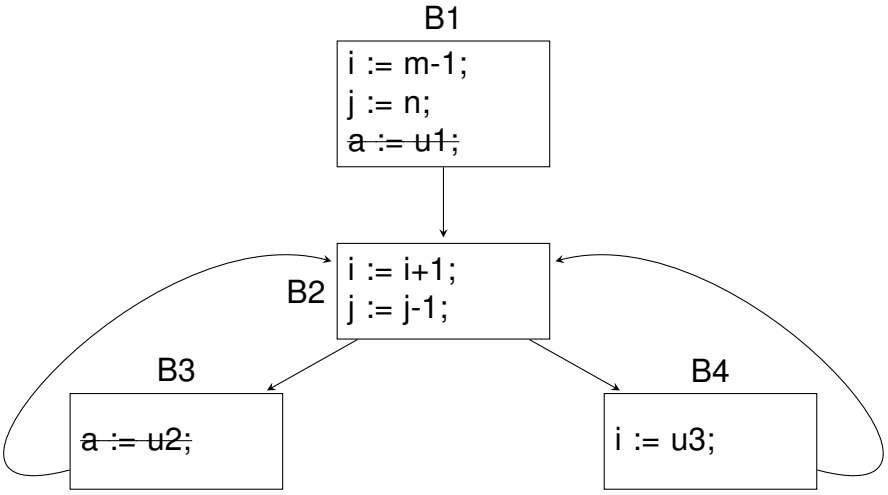
Statement S	$Out(S)$	$Out_{new}(S)$	Remove?
a := u2	$Out(B3)$	$Out(B3)$	Yes (a is not active after this)

For B4:

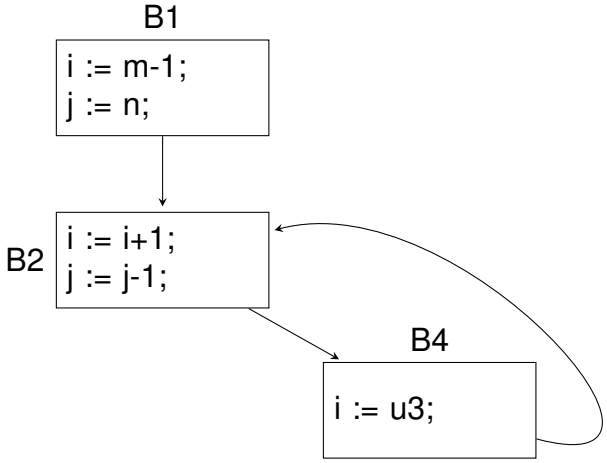
Statement S	$Out(S)$	$Out_{new}(S)$	Remove?
i := u3	$Out(B4)$	u2, u3, j	No, because i is active

For B1:

Statement S	$Out(S)$	$Out_{new}(S)$	Remove?
a := u1	$Out(B1)$	u1, u2, u3, j, i	Yes (a is not active after this)
j := n	u1, u2, u3, j, i	u1, u2, u3, i, n	No, because j is active
i := m-1	u1, u2, u3, i, n	u1, u2, u3, n, m	No, because i is active



This modifies the CFG into the following new CFG:



7.5.2 Middle CFG

1. See Exercise 100 above

2.

B	$Succ(B)$	$Gen(B)$	$Kill(B)$	$Out(B)$	$In(B)$	$Out(B)$	$In(B)$
B1	B2	$\emptyset$	a	$\emptyset$	$\emptyset$	c, a	c
B2	B3	c, a	a, c, b	$\emptyset$	c, a	a	c, a
B3	B2, B4	a	$\emptyset$	$\emptyset$	a	c, a, b	c, a, b
B4	$\emptyset$	b	$\emptyset$	$\emptyset$	b	$\emptyset$	b

B	$Out(B)$	$In(B)$	$Out(B)$
B1	c, a	c	c, a
B2	c, a, b	c, a	c, a, b
B3	c, a, b	c, a, b	c, a, b
B4	$\emptyset$	b	$\emptyset$

From this table, we can see the fix point for $Out(B)$ as follows:
$Out(B1) = \{c, a\}$
$Out(B2) = \{c, a, b\}$
$Out(B3) = \{c, a, b\}$
$Out(B4) = \emptyset$

3. For B3 and B4, nothing is assigned so nothing can be suppressed.
   For B2:

Statement S	$Out(S)$	$Out_{new}(S)$	Remove?
a := b*2	$Out(B2)$	c, b	No because a is active
c := c+b	c, b	c, b	No because c is active
b := a+1	c, b	c, a	No because b is active

   For B1:

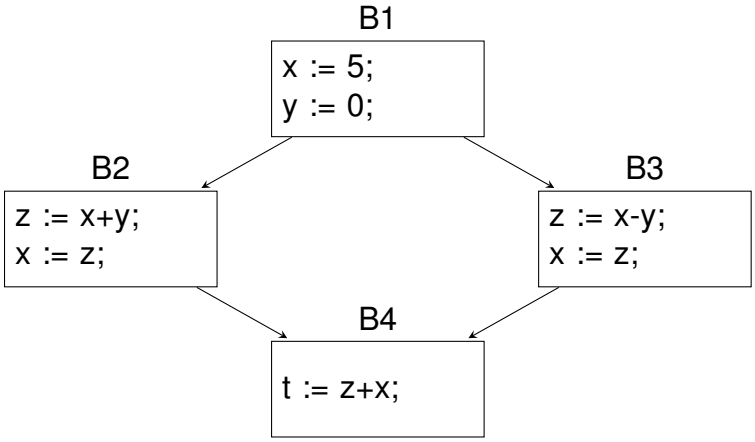
Statement S	$Out(S)$	$Out_{new}(S)$	Remove?
a := 0	$Out(B1)$	c	No, because a is active

   Thus, the CFG remains exactly the same, nothing is suppressed.

7.6 Exercise 102

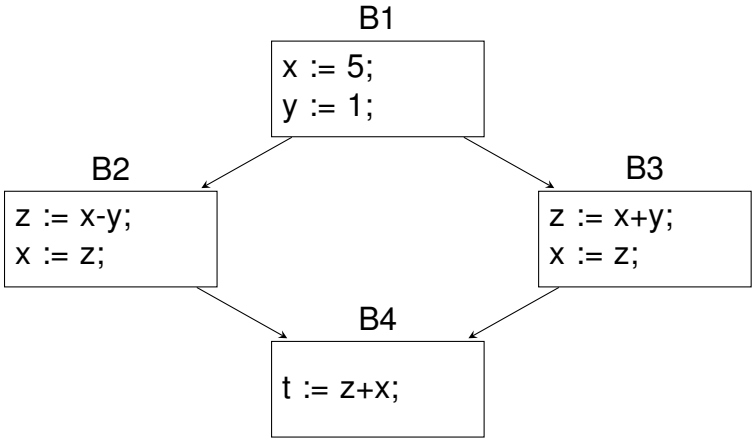
7.6.1 Left CFG

Var $= \{x, y, z, t\}$
There is no cycle, so the $In(B)$ and $Out(B)$ sets can be computed directly:
$In(B1) = \lambda x.\bot$
$Out(B1) = [x \mapsto 5, y \mapsto 0, z \mapsto \bot, t \mapsto \bot]$
$In(B2) = Out(B1)$
$Out(B2) = [z \mapsto 5, x \mapsto 5, y \mapsto 0, t \mapsto \bot]$
$In(B3) = Out(B1)$
$Out(B3) = [z \mapsto 5, x \mapsto 5, y \mapsto 0]$
$In(B4) = Out(B2) \sqcup Out(B3) = [z \mapsto 5, x \mapsto 5, y \mapsto 0, t \mapsto \bot]$
$Out(B4) = [z \mapsto 5, x \mapsto 5, y \mapsto 0, t \mapsto 10]$



7.6.2 Right CFG

Var $= \{x, y, z, t\}$
There is no cycle, so the $In(B)$ and $Out(B)$ sets can be computed directly:
$In(B1) = \lambda x.\bot$
$Out(B1) = [x \mapsto 5, y \mapsto 1, z \mapsto \bot, t \mapsto \bot]$
$In(B2) = Out(B1)$
$Out(B2) = [z \mapsto 4, x \mapsto 4, y \mapsto 1, t \mapsto \bot]$
$In(B3) = Out(B1)$
$Out(B3) = [z \mapsto 6, x \mapsto 6, y \mapsto 1]$
$In(B4) = Out(B2) \sqcup Out(B3) = [z \mapsto \top, x \mapsto \top, y \mapsto 1, t \mapsto \bot]$
$Out(B4) = [z \mapsto \top, x \mapsto \top, y \mapsto 1, t \mapsto \top]$



7.7 Exercise 103

7.7.1 Left CFG

Var $= \{x, y, i, z, t\}$

$$In(B) = \begin{cases} \lambda x.\bot & \text{in the first step} \\ \bigsqcup_{B' \in Pred(B)} Out(B') & \text{otherwise} \end{cases}$$

$Out(B) = F_B(In(B))$ where F_B is the transfer function of constant propagation.

B	$Pred(B)$	$In(B)$	$Out(B)$	$In(B)$	$Out(B)$
B1	$\emptyset$	$\lambda x.\bot$	$x \mapsto 4,$ $y \mapsto 0,$ $i \mapsto 0$	$\lambda x.\bot$	$x \mapsto 4,$ $y \mapsto 0,$ $i \mapsto 0$
B2	B1, B2	$\lambda x.\bot$	$i \mapsto 1$	$x \mapsto 4,$ $y \mapsto 0,$ $i \mapsto \top$	$z \mapsto 4,$ $t \mapsto 4,$ $i \mapsto \top,$ $x \mapsto 4,$ $y \mapsto 0$

B	$In(B)$	$Out(B)$	$In(B)$
B1	$\lambda x.\bot$	$x \mapsto 4, y \mapsto 0, i \mapsto 0$	$\lambda x.\bot$
B2	$x \mapsto 4,\ y \mapsto 0,$ $i \mapsto \top, z \mapsto 4, t \mapsto 4$	$x \mapsto 4,\ y \mapsto 0,$ $i \mapsto \top, z \mapsto 4, t \mapsto 4$	$x \mapsto 4,\ y \mapsto 0,$ $i \mapsto \top, z \mapsto 4, t \mapsto 4$

7.7.2 Right CFG

8 Series 11

8.1 Exercise 104

1.

Procedure	Shifting
main	$x \to 4$ $y \to 8$ $z \to 12$

2. ADD SP, SP, #-4
   ST FP, [SP]
   ADD FP, SP, #0
   ADD SP, SP, #-12

   ADD R_1, R_0, #5
   ST R_1, [FP - 4]
   ADD R_1, R_0, #1
   ST R_1, [FP - 8]
   LD R_1, [FP - 4]
   LD R_2, [FP - 8]

ADD R_3, R_1, R_2
ST R_3, [FP - 12]

ADD SP, FP, #0
LD FP, [SP]
ADD SP, SP, #4

8.2 Exercise 105

prologue(8)
lb:
LD R_1, [FP - 4]
LD R_2, [FP - 8]
CMP R_1, R_2
BEQ ltrue
BA lfalse
lfalse:
LD R_1, [FP - 4]
LD R_2, [FP - 8]
CMP R_1, R_2
BGT ltrue2
BA lfalse2
ltrue2:
LD R_1, [FP - 4]
LD R_2, [FP - 8]
SUB R_3, R_1, R_2
ST R_3, [FP - 4]
BA lnext
lfalse2:
LD R_1, [FP - 8]
LD R_2, [FP - 4]
SUB R_3, R_1, R_2
ST R_3, [FP - 8]
lnext:
BA lb
ltrue:
epilogue

8.3 Exercise 106

1.

GCStm(for (var = lbound; var < ubound; step) {S(var)})	= Let	lb=newLabel(), ltrue = newLabel(), lfalse = newLabel()
	in	GCStm(var := lbound)|| lb:|| GCBExp(var < ubound, ltrue, lfalse)|| ltrue:|| GCStm(S(var))|| GCStm(var := var + step)|| BA lb|| lfalse:

GCStm(for (var = lbound; var < ubound; step) {S(var)})	= Let	(Cstep,istep)=GCAexp(step) (Clbound, ilbound) = GCA-exp(lbound) (Cubound,iubound) = GCA-exp(ubound) i = AllocRegister()

	in	Clbound||
		ADD R_i, $R_{ilbound}$||
		lfor:||
		Cubound||
		CMP R_i, $R_{iubound}$||
		BGT lend||
		GCStm(S(var), i)||
		Cstep||
		Add R_i, R_i, R_{istep}||
		BA lfor||
		lend:

2.

prologue(44)
ADD R_1, R_0, #0
ST R_1, [FP - 40]
lb:
LD R_1, [FP - 40]
ADD R_2, R_0, #10
CMP R_1, R_2
BLE ltrue
BA lfalse
ltrue:
ADD R_1, R_0, #0
LD R_2, [FP - 40]

ADD R_3, R_2, FP
ST R_1, [R_3]
LD R_1, [FP - 40]
ADD R_2, R_1, #1
ST R_2, [FP - 40]
BA lb
lfalse:
epilogue

3. prologue(408)
TODO (similar to 2)
epilogue
RET

8.4 Exercise 107

For a (possibly better) solution of this exercise, see the lecture slides.

1.

GCStm(repeat S until b)	= Let	lb=newLabel(),
		ltrue = newLabel()
	in	lb:||
		GCStm(S)||
		GCBExp(b, ltrue, lb)||
		ltrue:

2.

GCBExp(b1 xor b2,ltrue,lfalse)	= Let	l2false=newLabel(),
		l2true = newLabel()
	in	GCBExp(b1, l2false, l2true)||
		l2false:||
		GCBExp(b2, lfalse, ltrue)||
		l2true:||
		GCBExp(b2, ltrue, lfalse)

3.

GCAexp(b ? e1 : e2)	= Let	ltrue=newLabel(),
		lfalse = newLabel()
	in	GCBExp(b, ltrue, lfalse)||
		ltrue:||
		GCAExp(e1)||
		lfalse:||
		GCAExp(e2)

8.5 Exercise 108

prologue(8)

ADD R_1, R_0, #1
ST R_1, [FP - 4]
▷ SL(p) = DL(main) / push static link of main on stack
push(FP)
call main.p
ADD SP, SP, #4
epilogue

main.p:
prologue(0)
▷ get static link to previous environment
LD R_1, [FP + 8]
LD $R_2, [R_1 - 4]$
ST $R_2, [R_1 - 8]$
epilogue
RET

8.6 Exercise 109

In the original code from the exercise sheet, after the first 'end' statement, another 'end' statement is missing which should terminate procedure p at this point.

main:
prologue(12)
ADD R_1, R_0, #5
ST R_1, [FP - 4]
▷ SL(p) = DL(main) / push static link of main on stack
push(FP)
call main.p
ADD SP, SP, #4
epilogue

main.p:
prologue(8)
▷ x := 4
ADD R_1, R_0, #4
ST R_1, [FP - 4]
▷ y := 5
ADD R_1, R_0, #5
▷ get static link to previous environment
LD R_2, [FP + 8]

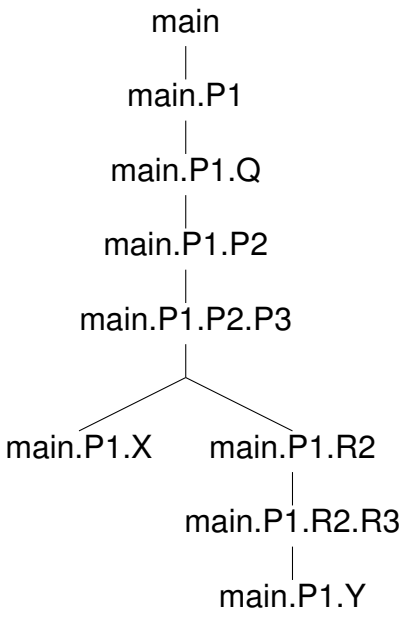
ST R_1, [$R_2 - 8$]
▷ z := x + y
LD R_1, [FP - 4]
LD R_3, [$R_2 - 8$] ▷ optimized: load y again from address R_2
ADD R_4, R_1, R_3
LD R_5, [FP + 8]
ST R_4, [$R_5 - 12$]
epilogue
RET

8.7 Exercise 112

1. We assume that return addresses are stored between the static link to the caller and the first parameter. Thus, last parameter is at +12 if procedure does not return anything / +16 if procedure returns something to leave space for return value. For a procedure parameter, first the address of the procedure is pushed then its static link, thus the address always has a higher offset than the SL.

Procedure	Label	Type	Offset
main	x1	int	-4
	P1	(void) $\rightarrow$ void	X
main.P1	x2	int	-4
	y2	int	-8
	z2	int	-12
	Y	(int) $\rightarrow$ int	X
	R2	(int, (int) $\rightarrow$ int) $\rightarrow$ void	X
	P2	(int, (int) $\rightarrow$ int) $\rightarrow$ void	X
	X	(int) $\rightarrow$ int	X
	Q	(void) $\rightarrow$ void	X
main.P1.Y	x	int	16 (param)
main.P1.R2	y3	int	-4
	b3	int	20 (param)
	p	(int) $\rightarrow$ int	12, 16 (proc param)
	R3	((int) $\rightarrow$ int) $\rightarrow$ void	X
main.P1.R2.R3	p	(int) $\rightarrow$ int	12, 16 (proc param)
main.P1.P2	x3	int	-4
	a	int	20 (param)
	p	(int) $\rightarrow$ int	12, 16 (proc param)
	P3	((int) $\rightarrow$ int) $\rightarrow$ void	X
main.P1.P2.P3	p	(int) $\rightarrow$ int	12, 16 (proc param)
main.P1.X	x	int	16 (param)
main.P1.Q	z3	int	-4

2.



3. main:
prologue(4)
▷ P1()
push(FP) ▷ push static link
call main.P1
ADD SP, SP, #4 ▷ remove static link
epilogue
RET

main.P1:
prologue(12)
▷ x1:=11
ADD R1, R0, #11
LD R2, [FP + 8] ▷ load static link
ST R1, [R2 - 4]
▷ Q()
push(FP) ▷ push static link
call main.P1.Q
ADD SP, SP, #4 ▷ remove static link
epilogue
RET

main.P1.Q:
prologue(4)

```
▷ y2 = 2
LD R1, [FP + 8]
ADD R2, R0, #2
ST R2, [R1 - 8]
▷ z2 = 3
LD R1, [FP + 8]
ADD R2, R0, #3
ST R2, [R1 - 12]
▷ z3 = x2 + y2
LD R1, [FP + 8]
LD R2, [R1 - 4]
LD R3, [R1 - 8]
ADD R3, R3, R2
ST R3, [FP - 4]
▷ P2(z3, X)
LD R1, [FP - 4]
push(R1) ▷ push 1st parameter for P2
SET R2, main.P1.X
push(R2) ▷ push @ of X as parameter for P2
LD R1, [FP + 8]
push(R1) ▷ push static link of X as param for P2
push(R1) ▷ push static link of P2 (same as static link of X)
call main.P1.P2
ADD SP, SP, #16 ▷ remove SL, 3 parameters
epilogue
RET


main.P1.P2:
prologue(4)
▷ x2 = 1+a
LD R1, [FP + 20]
ADD R1, R1, #1
LD R2, [FP + 8]
ST R1, [R2 - 4]
▷ P3(p)
LD R1, [FP + 16]
push(R1) ▷ push @ of function p as parameter of P3
LD R2, [FP + 12]
push(R2) ▷ push SL of p as parameter of P3
push(FP) ▷ push static link
call main.P1.P2.P3
ADD SP, SP, #12 ▷ remove SL, 2 parameters
```

```
epilogue
RET


main.P1.P2.P3:
prologue(0)
▷ x3 = p(x1)
LD R1, [FP + 8]
LD R1, [R1 + 8]
LD R1, [R1 + 8]
LD R2, [R1 - 4]
push(R2) ▷ parameter for p
LD R2, [FP + 12]
push(R2) ▷ SL of p
LD R1, [FP + 16]
call R1
LD R2, [SP + 4] ▷ get return
LD R1, [FP + 8]
ST R2, [R1 - 4]
ADD SP, SP, #12 ▷ remove SL, parameters
▷ R2(x3,Y)
push(R2) ▷ contains value of x3
SET R1, main.P1.Y
push(R1) ▷ push @ of Y as parameter for R2
LD R1, [FP + 8]
LD R1, [R1 + 8]
push(R1) ▷ push static link of Y as parameter
push(R1) ▷ push static link of R2 (same level as SL of Y)
call main.P1.R2
ADD SP, SP, #16 ▷ remove SL, 3 parameters
epilogue
RET


main.P1.R2:
prologue(4)
▷ R3(p)
LD R1, [FP + 16]
push(R1) ▷ push @ of p as parameter for R3
LD R2, [FP + 12]
push(R2) ▷ push SL of p as parameter for R3
push(FP) ▷ push static link
```

```
call main.P1.R2.R3
ADD SP, SP, #12 ▷ remove SL, 2 parameters
epilogue
RET


main.P1.R2.R3:
prologue(0)
▷ y2 = 2
ADD R1, R0, #2
LD R2, [FP + 8]
LD R2, [R2 + 8]
ST R1, [R2 - 8]
▷ x2 = y2+x1+b3
LD R2, [FP + 8]
LD R2, [R2 + 8]
LD R2, [R2 + 8]
LD R3, [R2 - 4]
ADD R1, R1, R3
LD R2, [FP + 8]
LD R3, [R2 + 20]
ADD R1, R1, R3
LD R2, [FP + 8]
LD R2, [R2 + 8]
ST R1, [R2 - 4]
▷ y3 = 1+x2
ADD R1, R1, #1
LD R2, [FP + 8]
ST R1, [R2 - 4]
▷ b3 = p(x1)
LD R2, [FP + 8]
LD R2, [R2 + 8]
LD R2, [R2 + 8]
LD R3, [R2 - 4]
push(R3) ▷ push parameter for p
LD R4, [FP + 12]
push(R4) ▷ push static link of p
push(R0) ▷ create return addr
LD R5, [FP + 16] ▷ get @ of p
call R5
LD R4, [SP + 4] ▷ get return and store in b3
LD R2, [FP + 8]
ST R4, [R2 + 20]
```

```
ADD SP, SP, #12 ▷ remove return addr, SL, 1 parameter
epilogue
RET


main.P1.Y:
prologue(0)
▷ x+1
LD R1, [FP + 16]
ADD R1, R1, #1
▷ call printf
push(R2) ▷ push 1st parameter for printf, assumes it is in R2
push(R1) ▷ push 2nd parameter for printf
push(FP) ▷ push static link
call printf
ADD SP, SP, #12 ▷ remove static link, 2 parameters
▷ return x+1
ST R1, [FP + 12] ▷ store value of x+1 in return address
epilogue
RET
```