PLCD Final Exam, December 2012
UJF, INPG, MoSIG 1

Wednesday 19 December 2012

Some comments, remarks, guidelines :
– The grading scale is indicative.
– Care of the submission will be taken into account.
– Each part has to be solved on a different sheet of paper.

1 Operational semantics : extended automata (10pt)

We are interested in a new intermediate form, which is the representation of a program as an extended automaton.

1.1 Transforming a program into an extended automaton

Definition An extended automaton is a 5-tuple $(Q, q_0, q_f, \mathcal{T}, V)$ where :
– Q is a finite-set of states, called control states, represented in this exercise by integers,
– q_0 is the initial state,
– q_f is the final state,
– $\mathcal{T} \subseteq Q \times (\texttt{BExp} \cup \texttt{Stm}_0) \times Q$ is a finite set of transitions,
– V is the set of local variables of the automaton.

Elements in $\texttt{Stm}_0$ (resp. in `BExp`) label transitions and are elementary statements (resp. Boolean expressions). These elements are defined by the following grammar where a designates an arithmetical expression (defined as in the course).

$$\begin{array}{rcl} \texttt{Stm}_0 & ::= & \texttt{skip} \mid x := a \\ b \in \texttt{BExp} & ::= & \texttt{true} \mid \texttt{false} \mid a = a \mid a < a \mid b \ \texttt{and} \ b \mid \ldots \end{array}$$

Translation function from `While` to extended automata. We define a function $\mathcal{F}$ which associates an extended automaton to a statement and a state (an integer). The integer, passed as a parameter to the function $\mathcal{F}$, is the initial state of the automaton returned by $\mathcal{F}$.

We add a syntactic category for programs with the following rule :

$$P ::= S$$

where S is a statement of the `While` language.

We extend the function $\mathcal{F}$ to programs : $\mathcal{F}(P) = \mathcal{F}(S, 0)$. Function $\mathcal{F}$ applied to statements is defined as follows (where *Var* is the function indicating the set of variables in an arithmetical or Boolean expression, defined as in the course) :

$$\begin{array}{lcl} \mathcal{F}(x := a, n) & = & (\{n, n+1\}, n, n+1, \{n \xrightarrow{x:=a} n+1\}, \{x\}) \\ \mathcal{F}(\texttt{skip}, n) & = & (\{n, n+1\}, n, n+1, \{n \xrightarrow{\texttt{skip}} n+1\}, \emptyset) \\ \mathcal{F}(S_1; S_2, n) & = & \text{Let } (Q_1, n, m, \mathcal{T}_1, V_1) = \mathcal{F}(S_1, n) \text{ in} \\ & & \quad \text{let } (Q_2, m, f, \mathcal{T}_2, V_2) = \mathcal{F}(S_2, m) \text{ in} \\ & & \qquad (Q_1 \cup Q_2, n, f, \mathcal{T}_1 \cup \mathcal{T}_2, V_1 \cup V_2) \end{array}$$

$$\begin{array}{l} \mathcal{F}(\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ fi}, n) = \\ \quad \text{Let } (Q_1, n+1, m, \mathcal{T}_1, V_1) = \mathcal{F}(S_1, n+1) \text{ in} \\ \qquad \text{let } (Q_2, m+1, f, \mathcal{T}_2, V_2) = \mathcal{F}(S_2, m+1) \text{ in} \\ \qquad\quad (Q_1 \cup Q_2 \cup \{n\}, n, f, \mathcal{T}_1 \cup \mathcal{T}_2 \cup \{n \xrightarrow{b} n+1, n \xrightarrow{\neg b} m+1, m \xrightarrow{\texttt{skip}} f\}, V_1 \cup V_2 \cup \mathit{Var}(b)) \end{array}$$

Exercice 1

Give the extended automaton for the following program :

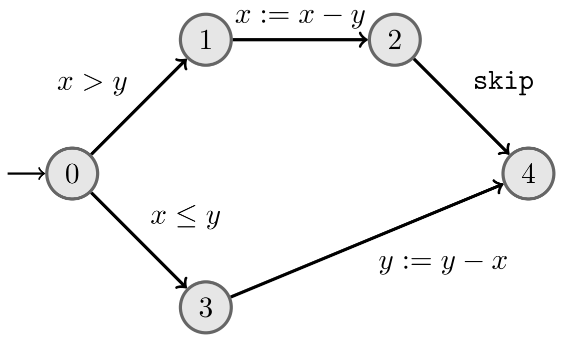
$$\texttt{if } x > y \texttt{ then } x := x - y \texttt{ else } y := y - x \texttt{ fi}$$

Solution de l'exercice 1

Since $\mathcal{F}(x := x-y, 1) = \big(\{1,2\}, 1, 2, \{(1, x := x-y, 2)\}\big)$ and $\mathcal{F}(y := y-x, 3) = \big(\{3,4\}, 3, 4, \{(3, y := y-x, 4)\}\big)$, we have :

$$\begin{aligned}
&\mathcal{F}\big(\texttt{if } x > y \texttt{ then } x := x - y \texttt{ else } y := y - x \texttt{ fi}\big)\\
&\quad = \mathcal{F}\big(\texttt{if } x > y \texttt{ then } x := x - y \texttt{ else } y := y - x \texttt{ fi}, 0\big)\\
&\quad = \big(\{0,1,2,3,4\}, 0, 4, \{(0, x > y, 1), (0, x \le y, 3), (1, x := x - y, 2), (3, y := y - x, 4), (2, \texttt{skip}, 4)\}\big)
\end{aligned}$$

We can graphically represent this extended automaton as follows :



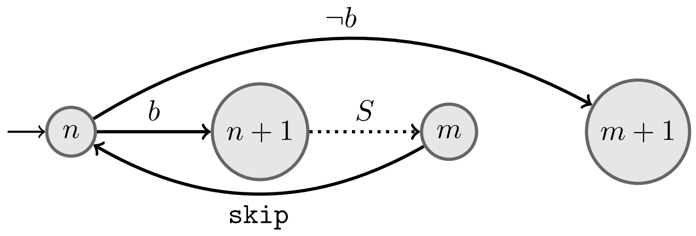
Exercice 2

Give the definition of $\mathcal{F}$ for the construct : `while` b `do` S `od`.

Solution de l'exercice 2

$$\begin{aligned}
\mathcal{F}(\texttt{while } b \texttt{ do } S \texttt{ od}, n) \quad = \quad & \text{let } (Q, n+1, m, \mathcal{T}) = \mathcal{F}(S, n+1) \text{ in}\\
& (Q \cup \{m+1\}, n, m+1, \mathcal{T} \cup \{n \xrightarrow{b} n+1, n \xrightarrow{\neg b} m+1, m \xrightarrow{\texttt{skip}} n\})
\end{aligned}$$

We can graphically represent this extended automaton as follows :



Exercice 3

Give the extended automaton for the following program :

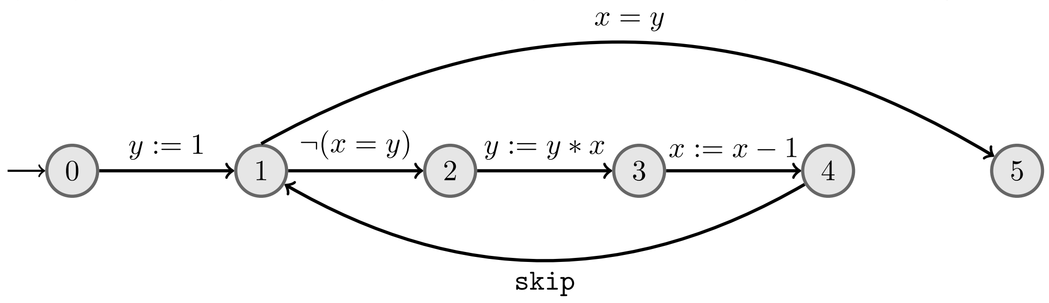
```
y :=1 ;
while ¬(x = y) do y := x * y; x := x − 1 od
```

Solution de l'exercice 3

Let P be the considered program. We have $\mathcal{F}(P) = \mathcal{F}(P, 0)$. Moreover, $\mathcal{F}(y := 1, 0) = (\{0,1\}, 0, 1, \{(0, y := 1, 1)\})$. Then :

$$\mathcal{F}(y := x * y; x := x - 1, 2) = \big(\{2,3,4\}, 3, 5, \{(2, y := y * x, 3), (3, x := x - 1, 4)\}\big)$$

It follows that $\mathcal{F}(\texttt{while}\ \neg(x = y)\ \texttt{do}\ y := x * y; x := x - 1\ \texttt{od}, 0)$
$= (\{0, 1, 2, 3, 4, 5\}, 1, 5, \{(1, \neg(x = y), 2), (1, x = y, 5), (4, \texttt{skip}, 1), (2, y := y * x, 3), (3, x := x - 1, 4)\}$. Finally : $\mathcal{F}(y := 1; \texttt{while}\ \neg(x = y)\ \texttt{do}\ y := x*y; x := x-1\ \texttt{od}, 0) = (\{0, 1, 2, 3, 4, 5\}, 0, \{(0, y := 1, 1), (1, \neg(x = y), 2), (1, x = y, 5), (4, \texttt{skip}, 1), (2, y := y * x, 3), (3, x := x - 1, 4)\}$.



1.2 Semantics

In this part, $(Q, q_0, q_f, \mathcal{T}, V)$ designates an extended automaton. As was the case for programs, $\sigma \in \mathbf{Mem}$, where **Mem** designates the memory, associating a value to a name.

Transition system associated to an automaton. A configuration is an element of $Q \times \mathbf{Mem}$. The set of configurations is noted **Conf**. The operational semantics of an extended automaton is defined by a transition system $(\mathbf{Conf}, \longrightarrow, (q_0, \sigma_0))$, where (q_0, σ_0) is the initial configuration corresponding to state σ_0. The transition relation is defined as follows :

$(q_i, \sigma_i) \longrightarrow (q_{i+1}, \sigma_{i+1})$ if one of the three following conditions holds :

- $q_i \xrightarrow{skip} q_{i+1}$ and $\sigma_{i+1} = \sigma_i$,
- $q_i \xrightarrow{x:=a} q_{i+1}$ and $\sigma_{i+1} = \sigma_i[x \mapsto \mathcal{A}[a]_{\sigma_i}]$,
- $q_i \xrightarrow{b} q_{i+1}$ and $\sigma_{i+1} = \sigma_i$ and $\mathcal{B}[b]_{\sigma_i} = tt$.

where $\mathcal{A} : \texttt{Aexp} \times \mathbf{Mem} \to \mathbb{Z}$ and $\mathcal{B} : \texttt{Bexp} \times \mathbf{Mem} \to \{tt, ff\}$ are the semantics functions defined as in the course.

Exercice 4

Give the semantics of transitions of the form : $q_i \xrightarrow{S} q_k$, in cases where S is the `skip` statement and S is an assignment $x := a$.

Solution de l'exercice 4

$$\frac{q_i \overset{\texttt{skip}}{\to} q_k}{(q_i, \sigma) {\to} (q_k, \sigma)} \qquad \frac{q_i \overset{x:=a}{\to} q_k}{(q_i, \sigma) {\to} (q_k, \sigma[x \mapsto \mathcal{A}[a]_\sigma])}$$

Exercice 5

Give the semantics of the transitions of the form : $q_i \xrightarrow{b} q_k$, where b is a Boolean expression.

Solution de l'exercice 5

$$\frac{q_i \overset{b}{\to} q_k \quad \mathcal{B}[b]_\sigma = tt}{(q_i, \sigma) {\to} (q_k, \sigma)}$$

1.3 Parallelism

In the remainder of this exercise, we suppose that we have a set $\mathcal{P} = \{A_1, \ldots, A_n\}$ of n automata executing in parallel. For each automaton A_i, we note Q_i the set of its control states, V_i its set of local variables and $\mathcal{T}_i$ its transition relation. The sets V_i are pairwise disjoints..

The execution of this set of automata is **asynchronous** : each automaton can execute freely. At the semantics level, the state of the program is described by a pair $(\mathbf{q}, \sigma)$ where :

- $\mathbf{q}$ is a vector of states $\mathbf{q} = (q_1, \ldots, q_n)$ with $\forall i \in [1, n] : q_i \in Q_i$, and
- $\sigma \in (\bigcup_{i \in [1,n]} V_i) \to \mathbb{Z}$ is the state for all variables.

We note $\mathbf{q}[i]$ the i-th coordinate of $\mathbf{q}$, that is, for $\mathbf{q} = (q_1, \ldots, q_n)$, $\mathbf{q}[i] = q_i$. The vector $\mathbf{q}$ where (only) the i-th coordinate is replaced by state q' is noted $\mathbf{q}[i/q']$.

The semantics is described by the unique general following rule :

$$\frac{\mathbf{q}[i] = q_i \quad q_i \overset{X}{\to} q'_i}{(\mathbf{q}, \sigma) \to (\mathbf{q}[i/q'_i], \sigma')}$$

in which $X \in \texttt{Stm}_0 \cup \texttt{Bexp}$, q_i is the control state of the i-th automaton, and the relation between σ and σ' depends on the statement executed by A_i.

Exercice 6

Instantiate the previous semantics rule when X is the `skip` statement.

Solution de l'exercice 6

$$\frac{\mathbf{q}[i] = q_i \quad q_i \overset{\texttt{skip}}{\to} q'_i}{(\mathbf{q}, \sigma) \to (\mathbf{q}[i/q'_i], \sigma)}$$

Exercice 7

Instantiate the previous general semantics rule when X is the statement $x := a$.

Solution de l'exercice 7

$$\frac{\mathbf{q}[i] = q_i \quad q_i \overset{x:=a}{\to} q'_i}{(\mathbf{q}, \sigma) \to (\mathbf{q}[i/q'_i], \sigma[x \mapsto \mathcal{A}[a]_\sigma)}$$

Exercice 8

Instantiate the previous general semantics rule when X is a Boolean expression b.

Solution de l'exercice 8

$$\frac{\mathbf{q}[i] = q_i \quad q_i \overset{b}{\to} q'_i \quad \mathcal{B}[b]_\sigma = tt}{(\mathbf{q}, \sigma) \to (\mathbf{q}[i/q'_i], \sigma)}$$

Exercice 9

Give a necessary and sufficient syntactic condition such that each automaton A_i only modifies its set of variables V_i.

Solution de l'exercice 9

The following proposition states that, for each automaton, for each transition modifying a variable x, this variable belongs to the set of local variables of this automaton :

$$\forall i \in [1,n] : \Big(\big(\exists q_1, q_2 \in \mathbb{N}, \exists x \in (\bigcup_{i \in [1,n]} V_i), \exists a \in \texttt{Aexp} : (q_1, x := a, q_2) \in T_i \big) \Rightarrow x \in V_i \Big)$$

Equivalently, we can express this condition as follows. First, we define a function $Def : \bigcup_{i\in[1,n]} T_i \to \bigcup_{i\in[1,n]} V_i$, similar to the function Def seen in the course, as follows :

- $\forall i \in [1,n], \forall q_1, q_2 \in Q_i : Def(q_1, b, q_2) = \emptyset$
- $\forall i \in [1,n], \forall q_1, q_2 \in Q_i : Def\big((q_1, x := a, q_2)\big) = \{x\}$.

The, we lift the function Def to a set of transitions T_i as follows $Def(T_i) = \bigcup_{t_i \in T_i} Def(t_i)$. Finally, the condition becomes : $\forall i \in [1,n], Def(T_i) \subseteq V_i$.

1.3.1 Introducing locking commands

In this part of the exercise, a program is a set $\mathcal{P} = \{A_1, \ldots, A_n\}$ of n automata executing in parallel augmented with a set of global variables V. We are now interested in a synchronization mechanism based on commands that lock the access to global variables.

A global configuration is now a 3-tuple $(\mathbf{q}, \sigma, \pi)$ where π is a set of pair (locked global variable, locking automaton) and $\mathbf{q}$ is as in the previous section. The state σ is extended to global variables. The set of elementary statements is now (where $x \in (\bigcup_{i\in[1,n]} V_i) \cup V$) :

$$\texttt{Stm}'_0 ::= \text{ skip } \mid x := a \mid \text{ lock(x) } \mid \text{ unlock(x)}$$

For a global variable x, the intuitive semantics is as follows :

- if an automaton A_i executes the statement $\texttt{lock}(x)$ then another automaton cannot read nor write variable x until A_i executes the statement $\texttt{unlock}(x)$.
- an automaton that executes the statement $\texttt{unlock}(x)$ frees the variable x.

Exercice 10

Give a necessary and sufficient syntactic condition such that each automaton A_i only modifies its set of variables V_i or the global variables in V.

Solution de l'exercice 10

Similarly to the previous question, the condition is :

$$\forall i \in [1,n], \forall t_i \in T_i : Def(t_i) \subseteq (V_i \cup V).$$

In the following, we suppose that this condition holds.

Exercice 11

Complete the following semantics rule for the elementary statement `skip`.

$$\frac{q_i \xrightarrow{\texttt{skip}} q'_i \quad \cdots}{(\mathbf{q}, \sigma, \pi) \longrightarrow \cdots}$$

Solution de l'exercice 11

$$\frac{q_i \xrightarrow{\texttt{skip}} q'_i \quad \mathbf{q}[i] = q_i}{(\mathbf{q}, \sigma, \pi) \longrightarrow (\mathbf{q}[i/q'_i], \sigma, \pi)}$$

Exercice 12

Complete the following semantics rule for the elementary statement of assignment.

$$\frac{q_i \xrightarrow{x:=a} q'_i \quad \cdots}{(\mathbf{q}, \sigma, \pi) \longrightarrow \cdots}$$

Solution de l'exercice 12

For $x \in V$, we define $\text{locking}(x, \pi) = \{A_j \in \mathcal{P} \mid \exists (x, A_j) \in \pi\}$, as the set of automata locking the variable x according to π.

We have two rules according to whether the assignment uses a global variable or not :

$$\frac{q_i \xrightarrow{x:=a} q'_i \quad \mathbf{q}[i] = q_i \quad \big(\{x\} \cup \mathit{Var}(a)\big) \cap V \neq \emptyset \quad \forall x' \in \{x\} \cup \mathit{Var}(a) : \text{locking}(x', \pi) \subseteq \{A_i\}}{(\mathbf{q}, \sigma, \pi) \longrightarrow (\mathbf{q}[i/q'_i], \sigma[x \mapsto \mathcal{A}[a]_\sigma], \pi)}$$

$$\frac{q_i \xrightarrow{x:=a} q'_i \quad \mathbf{q}[i] = q_i \quad \big(\{x\} \cup \mathit{Var}(a)\big) \cap V = \emptyset}{(\mathbf{q}, \sigma, \pi) \longrightarrow (\mathbf{q}[i/q'_i], \sigma[x \mapsto \mathcal{A}[a]_\sigma], \pi)}$$

We can merge these two rules, by observing that, when x is a local variable, then $\text{locking}(x, \pi) = \emptyset$. Hence :

$$\frac{q_i \xrightarrow{x:=a} q'_i \quad \mathbf{q}[i] = q_i \quad \forall x' \in \{x\} \cup \mathit{Var}(a) : \text{locking}(x', \pi) \subseteq \{A_i\}}{(\mathbf{q}, \sigma, \pi) \longrightarrow (\mathbf{q}[i/q'_i], \sigma[x \mapsto \mathcal{A}[a]_\sigma], \pi)}$$

Exercice 13

Complete the following semantics rule for Boolean expressions.

$$\frac{q_i \xrightarrow{b} q'_i \quad \cdots}{(\mathbf{q}, \sigma, \pi) \longrightarrow \ldots}$$

Solution de l'exercice 13

Similarly, we have two rules according to whether the Boolean expression uses a global variable or not :

$$\frac{q_i \xrightarrow{b} q'_i \quad \mathbf{q}[i] = q_i \quad \mathit{Var}(b) \cap V \neq \emptyset \quad \forall x \in \mathit{Var}(b) : \text{locking}(x, \pi) \subseteq \{P_i\} \quad \mathcal{B}[b]_\sigma = \mathit{tt}}{(\mathbf{q}, \sigma, \pi) \longrightarrow (\mathbf{q}[i/q'_i], \sigma, \pi)}$$

$$\frac{q_i \xrightarrow{b} q'_i \quad \mathbf{q}[i] = q_i \quad \mathit{Var}(b) \cap V = \emptyset \quad \mathcal{B}[b]_\sigma = \mathit{tt}}{(\mathbf{q}, \sigma, \pi) \longrightarrow (\mathbf{q}[i/q'_i], \sigma, \pi)}$$

Similarly to the previous question, we can merge these two rules as follows :

$$\frac{q_i \xrightarrow{b} q'_i \quad \mathbf{q}[i] = q_i \quad \forall x \in \mathit{Var}(b) : \text{locking}(x, \pi) \subseteq \{P_i\} \quad \mathcal{B}[b]_\sigma = \mathit{tt}}{(\mathbf{q}, \sigma, \pi) \longrightarrow (\mathbf{q}[i/q'_i], \sigma, \pi)}$$

Exercice 14

Complete the following semantics rule for the elementary statement `lock`(x).

$$\frac{q_i \xrightarrow{\texttt{lock}(x)} q'_i \quad \cdots}{(\mathbf{q}, \sigma, \pi) \longrightarrow \cdots}$$

Solution de l'exercice 14

We define locked$(\pi) = \{x \in V \mid \exists(x, A_j) \in \pi\}$, as the set of variables that are locked according to π.

$$\frac{q_i \xrightarrow{\texttt{lock}(x)} q_i' \quad \mathbf{q}[i] = q_i \quad x \notin \text{locked}(\pi)}{(\mathbf{q}, \sigma, \pi) \longrightarrow (\mathbf{q}[i/q_i'], \sigma, \pi \cup \{(x, A_i)\})}$$

Exercice 15

Complete the following semantics rule for the elementary statement `unlock`(x).

$$\frac{q_i \xrightarrow{\texttt{unlock}(x)} q_i' \quad \cdots}{(\mathbf{q}, \sigma, \pi) \longrightarrow \cdots}$$

Solution de l'exercice 15

$$\frac{q_i \xrightarrow{\texttt{lock}(x)} q_i' \quad \mathbf{q}[i] = q_i \quad (x, A_i) \in \pi}{(\mathbf{q}, \sigma, \pi) \longrightarrow (\mathbf{q}[i/q_i'], \sigma, \pi \setminus \{(x, A_i)\})}$$

Exercice 16

Give a sufficient condition such that, given a system with two automata, there is no deadlock.

Solution de l'exercice 16

- No read nor write or global variables.
- At least one process does not read nor write global variables.
- No `lock,unlock` statement.

1.3.2 Introducing statements for emission and reception

In this second part, we suppose that the automata communicate through (non-blocking) emissions and (possibly blocking) reception over a channel.

A program is a set of communicating automata $\mathcal{P} = \{A_1, \ldots, A_n\}$ augmented with a channel C. A channel is a set of triples (i, j, v) where i and j designate automata and v is the value transmitted from i to j.

Now a configuration is of the form $(\mathbf{q}, \sigma, \pi)$ where

- π is the content of the channel, that is the set of 3-tuples (i, j, v) where i (resp. j) is the emitting (resp. receiving) automaton and v the transmitted value.

$\mathbf{q}$ and σ are as before. The set of elementary statements is now defined as :

$$\texttt{Stm}_0 ::= \text{ skip } \mid x := a \mid \text{ send(i,j,a) } \mid \text{receive(i,j,x)}$$

where :

- i is the emitting automaton,
- j is the receiving automaton,
- a is an arithmetical expression which value is emitted,
- x is a variable that will contain the received value.

Exercice 17

Give the semantics rule for the elementary statement `send`.

Solution de l'exercice 17

$$\frac{\mathbf{q}[i] = q_i \quad q_i \xrightarrow{\texttt{send}(i,j,a)} q_i' \quad \mathcal{A}[a]_\sigma = v}{(\mathbf{q}, \sigma, \pi) \rightarrow (\mathbf{q}[i/q_i'], \sigma, \pi \cup \{i, j, v)\})}$$

Exercice 18

Give the semantics rule for the elementary statement `receive`.

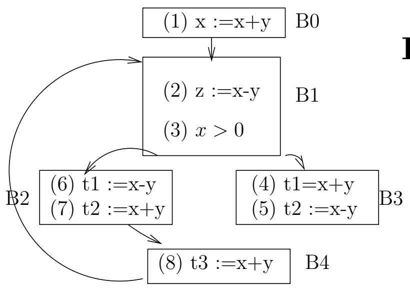
Solution de l'exercice 18

Observe that the order of sending messages is lost. Moreover, we does not know what value is picked up in case there are several messages coming from a process destinated to the current process.

$$\frac{\mathbf{q}[i] = q_i \quad q_i \xrightarrow{\texttt{receive}(j,i,x)} q_i' \quad \exists(j,i,v) \in \pi}{(\mathbf{q}, \sigma, \pi) \to (\mathbf{q}[i/q_i'], \sigma[x \mapsto v], \pi \setminus \{i,j,v)\})}$$

2 Optimization (5pt)

We consider the control-flow graph below :



Exercice 19 Available Expressions

1. Compute the sets $Gen(b)$ et $Kill(b)$ for each basic block b.
2. Compute the sets $In(b)$ et $Out(b)$ for each basic block b.
3. Determine and suppress redundant computations.

Blocks	Gen	Kill
B0	∅	e1 e2
B1	e2 e3	∅
B2	e1 e2	∅
B3	e1 e2	∅
B4	e1	∅

Table 1 – Gen et Kill

Blocks	I		II		III	
	In	Out	In	Out	In	Out
B0	∅	∅	∅	∅	∅	∅
B1	e1 e2 e3	e1 e2 e3	∅	e2 e3	∅	e2 e3
B2	e1 e2 e3	e1 e2 e3	e1 e2 e3	e1 e2 e3	e2 e3	e1 e2 e3
B3	e1 e2 e3	e1 e2 e3	e1 e2 e3	e1 e2 e3	e2 e3	e1 e2 e3
B4	e1 e2 e3	e1 e2 e3	e1 e2 e3	e1 e2 e3	e1 e2 e3	e1 e2 e3

Table 2 – In and Out

Solution de l'exercice 19

Let $e1 = x+y, e2 = x-y, e3 = x > 0$ be the expressions to optimize. We have, $Out[Entry] = \emptyset$ and $Out[B_i] = \{e1, e2, e3\}, i = 0, \ldots, 4$.

1. The sets $Gen(b)$ et $Kill(b)$ are given in Table 1.

2. Dataflow equations, for each block, are as follows :
   $In[B0] = Out[Entry]$
   $In[B1] = Out[B0] \cap Out[B4]$
   $In[B2] = Out[B1]$
   $In[B3] = Out[B1]$
   $In[B4] = Out[B2]$
   $Out[B_i] = In[B_i] \setminus Kill[B_i] \cup Gen[B_i], i = 0, ..., 4.$

   Results are shown in Table 2.
3. At the entry of B0 and B1, no expression is available, so no change.
   At the entry of B2 and B3, e2 is available. So, in B1, we add `u2 := x-y`, `z:=u2`. Moreover, we add, in B2, `t1:=u2` and in B3, `t2:=u2`.
   At the entry of B4, e1 and e2 are available. Consequently, in B2, we add `u1 := x+y`, `t2:=u1`. Moreover, in B4, we add `t3:=u1`. The resulting CFG is shown in Figure 1.

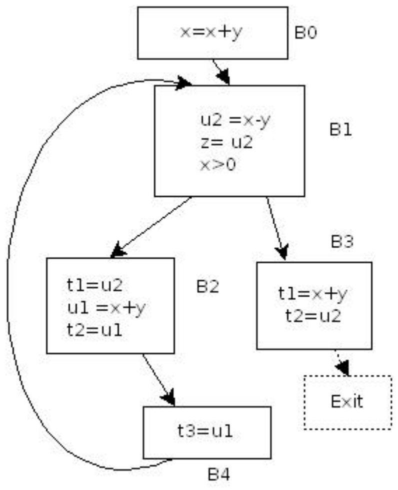


FIGURE 1 – New CFG for the Ex 19-3

Exercice 20 Actives Variables

We consider the control-flow graph obtained at the end of the previous exercise.

1. Compute the sets $Gen(b)$ et $Kill(b)$ for each basic block b.
2. Compute the sets $In(b)$ et $Out(b)$ for each basic block b.
3. Suppress useless statements which are assignments $x := e$ such that x is inactive at the end of a block and not used in the block following the statement.

Blocs	Gen	Kill
B4	u1	t3
B3	x y u2	t1 t2
B2	x y u2	t1 t2 u1
B1	x y	z u2
B0	x y	x

TABLE 3 – Gen and Kill

Blocs	I		II		III	
	Out	In	Out	In	Out	In
B4	ϕ	u1	x y	x y u1	x y	x y u1
B3	ϕ	x y u2	ϕ	x y u2	ϕ	x y u2
B2	ϕ	x y u2	u1	x y u2	x y u1	x y u2
B1	ϕ	x y	x y u2	x y	x y u2	x y
B1	ϕ	x y	x y	x y	x y	x y

TABLE 4 – In and Out

Solution de l'exercice 20

1. The sets $Gen(b)$ and $Kill(b)$ are shown in Table **??**.
2. Below are the dataflow equations for each block :
   $Out[B4] = In[B1]$
   $Out[B3] = In[Exit]$
   $Out[B2] = In[B4]$
   $Out[B1] = In[B2] \cup In[B3]$
   $Out[B0] = In[B1]$
   $In[B_i] = Out[B_i] \setminus Kill[B_i] \cup Gen[B_i]$ Results are shown in Table **??**.
3. At the exit of B4, t3 is not active, thus the assignment `t3:=u1` can be removed (so can be entire block).
   At the exit of B3, no variable is active, and then the entire block can be removed.
   At the exit of B2, t1 and t2 are not active, and then the assignements `t1:=u2` and `t2:=u1` can be removed.
   At the exit of B1, z is not active and then the assignement `z:=u2` can be suppressed.
   At the exit of B0, x and y are active, thus no assignement can be removed.

3 Hoare Logic (4pt)

Exercice 21

Prove that the following Hoare triple is valid : $\{n \geq 1\} \quad S \quad \{p = m * n\}$ where S is :

```
p := 0 ;
c := 1 ;
while c <= n do
  p := p + m ;
  c := c + 1 ;
od
```

Solution de l'exercice 21

The invariant is $I \equiv p = (c-1) * m \wedge c \leq n + 1$.

$$\dfrac{\dfrac{\dfrac{\{I\} \quad p := p + m \quad \{p = c * m \wedge c \leq n\} \quad \{p = c * m \wedge c \leq n\} \quad c := c + 1 \quad \{I\}}{\{c \leq n \wedge I\} \quad S_1 \quad \{I\}}}{\{I\} \quad \textsf{while } c \leq n \ \textsf{ do } S_1 \textsf{ od} \quad \{I \wedge n < c\}}}{\{I\} \quad \textsf{while } c \leq n \ \textsf{ do } S_1 \textsf{ od} \quad \{p = c * m\}}$$

$$\dfrac{\dfrac{\{0=0\}\quad p:=0\quad \{p=0\}\quad \{p=m\}\quad c:=1\quad \{I\}}{\{0=0\}\quad S_0\quad \{I\}}}{\{n\geq 1\}\quad S_0\quad \{I\}}$$

Finally, we get :

$$\dfrac{\{n\geq 1\}\quad S_0\quad \{I\}\quad \{I\}\quad \textsf{while } c\leq n \textsf{ do } S_1 \textsf{ od}\quad \{p=c*m\}}{\{n\geq 1\}\quad S\quad \{p=m*n\}}$$

4 Code Generation (4.5 points)

```
main() {
  int x1 ;
  int y1 ;
  int f1(int v) {
    int y ;
    int Q2 (int x) {
      int z ;
      z = 3;
      x = y+x+z+x1;
      return(x);
    } /*  end Q2 */
    int f2() {
      y = Q2 (5);
      return y+v;
    } /*  end f2 */
    y = y1;
    f2();
    return (y+1);
  } /* end f1 */
  x1 = 11;
  y1 = 21;
  x1 = 42 + f1(4);
}/* end main */
```

Exercice 22

We consider the above program.

1. Draw the stack during the execution of `Q2`.
2. In procedure `main`, give the sequence of instructions corresponding to `y1=21;`.
3. In procedure `main`, give the sequence of instructions corresponding to `x1=42+f1(4);`.
4. In function `f1`, give the sequence of instructions corresponding to `y = y1;`.
5. In function `f1`, give the sequence of instructions corresponding to `return (y+1);`.
6. In function `f2`, give the sequence of instructions corresponding to `y= Q2 (5)`.
7. In procedure `Q2`, give the sequence of instructions corresponding to `x = y+x+z+x1`.
8. In procedure `Q2`, give the sequence of instructions corresponding to `return (x)`.

9. At the end of program's execution, give the value associated to `x1`.
10. Suppose that procedure `Q2` was defined with a parameter passed by reference (that is `int Q2 (int *x)`. In procedure `f2`, if we add the statement `y = Q2 (&y1)`, give the sequence of instructions corresponding to `y = Q2 (&y1)`.

Solution de l'exercice 22

As discussed in the class, there are two ways to return the value from a function. 1) we can use a set of reserved registers e.g. RV, RV1 OR 2) we can reserve place on the stack. This place must be reserved by the caller, after pushing the parameters and the callee must move the return value at that place on the stack. In such a case, from callee, the parameters are accessed at [FP-16], [FP-20] and so on. In the case, the callee returns a function pointer, caller has to make space for two return values : for function and for SL. The parameters are accessed at [FP-20], [FP - 24] and so on. In the following solutions, both of the above methods are given (as and when is necessary).

1. The stack is in Figure 2.

2.
   ```
   SUB R0 R0 R0
   ADD R1 R0 21
   ST R1 [FP-8]
   ```
3. **with RV :**
   ```
   ADD R1 R0 4
   PUSH(R1)
   PUSH(FP)
   CALL f1
   ADD SP SP 8
   ADD R1 RV R0
   ADD R2 R0 42
   ADD R2 R2 R1
   ST R2 [FP-4]
   ```
   without RV :
   ```
   ADD R1,R0,4
   PUSH(R1)
   ADD SP,SP,-4 !space for return
   PUSH(FP)
   CALL main.f1
   ADD SP,SP,4
   LD R1, [SP] !read the return
   ADD R1,R1,42
   ST R1, [FP-4]
   ```

4.
   ```
   LD R1 [FP+8]
   LD R2 [R1-8]
   ST R2 [FP-4]
   ```

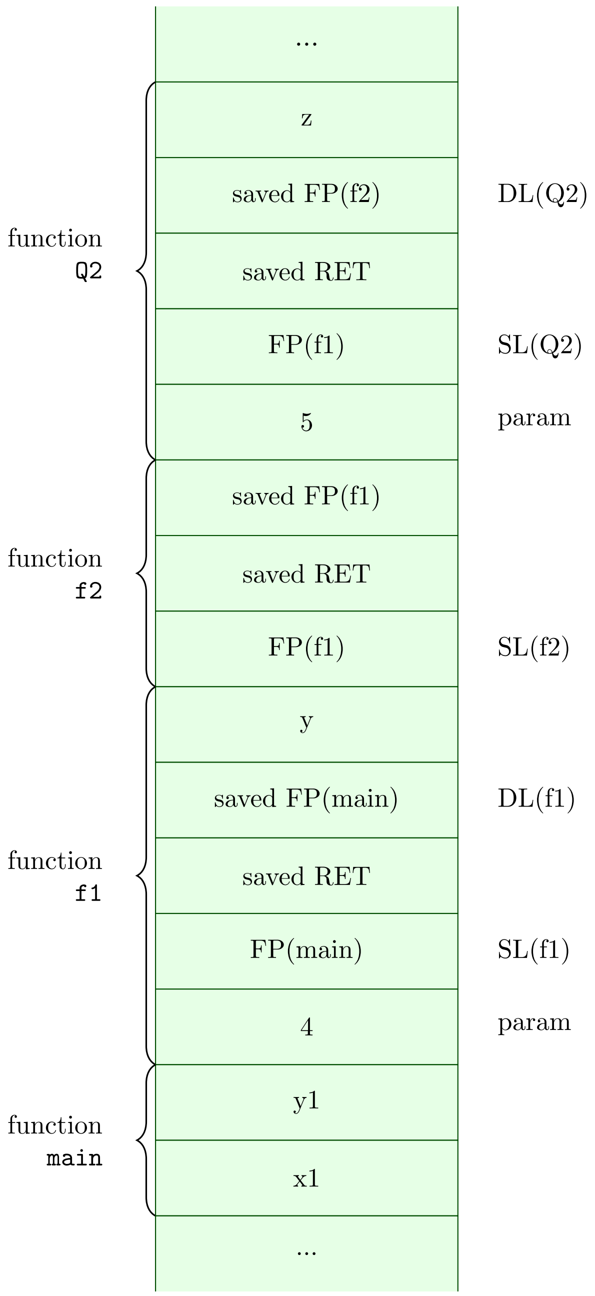


FIGURE 2 – Stack during the execution of Q2

5. **with RV :**
```
LD R1 [FP-4]
ADD R1 R1 1
ADD RV R1 0
Epilogue
RET
```

   without RV :
```
LD R1,[FP-4]
ADD R1,R1,1
ST R1,[FP+12] !placing return on the stack
```

6. **with RV :**
```
ADD R1 R0 5
PUSH(R1)
LD R2 [FP+8]
PUSH(R2)
CALL Q2
ADD SP SP 8
ADD R1 RV R0
LD R2 [FP+8]
ST R1 [R2-4]
```

   without RV :
```
ADD R1,R0,5
PUSH(R1)
ADD SP, SP,-4
LD R2,[FP+8]
PUSH(R2)
CALL main.f1.Q2
ADD SP,SP,4
LD R1,[SP]
ST R1,[R2-4]
ADD SP,SP,4
```

7. **with RV :**
```
LD R1 [FP+12] !R1:=x
LD R2 [FP+8]
LD R3 [R2-4] !R3 :=y
LD R4 [FP-4] !R4 := z
LD R5 [R2+8]
LD R6 [R5-4] !R6 := x1
ADD R1 R1 R3 !R1 :=x+y
ADD R1 R1 R4 !R1 := x+y+z
ADD R1 R1 R6 !R1 := x+y+z+x1
ST R1 [FP+12] !x=x+y+z+x1
```

without RV :

```
LD R1,[FP+16] !R1 := x (offset changed to 16)
LD R2,[FP+8]
LD R3,[R2-4] !R3 := y
LD R4, [FP-4] !R4 := z
LD R5, [R2+8]
LD R6, [R5-4] !R6 := x1
ADD R1,R3,R1 !R1 := x+y
ADD R1,R4,R1 !R1 := x+y+z
ADD R1,R6,R1 !R1 := x+y+z+x1
ST R1,[FP+16] !x=x+y+z+x1
```

8. **with RV :**

```
LD R1 [FP+12]
ADD RV R1 R0
Epilogue
RET
```

without RV :

```
LD R1,[FP+16]
ST R1,[FP+12]
```

9. We have x1 =11, y1=21 and we have to compute x1 = 42 + f1(4).
Computation of f1(4) :
    y = 21, f2() ;
    Computation of f2() :
        y = Q2(5)
        computation of Q2(5) :
            x= 21+5+3+11 = 40
        y=40
    f1(4) return y+1 $\rightarrow$ 41
therefore, x1 = 42 + f1(4) $\rightarrow$ x1 = 42+41 = 83

10. **with RV :**

```
LD R1 [FP+8]
LD R2 [R1+8]
```

```
ADD R3 R2 -8 !@y1
PUSH(R3)
PUSH(R1) !sl(Q2)
CALL Q2
ADD SP SP 8
ADD R1 RV 0
LD R2 [FP+8]
ST R1 [R2-4]
```

without RV :

```
LD R1,[FP+8]
LD R2,[R1+8]
ADD R3,R2,-8
PUSH(R3)
ADD SP, SP,-4
PUSH(R1)
CALL main.f1.Q2
ADD SP,SP,4
LD R4,[SP]
ST R4,[R1-8]
ADD SP,SP,4
```