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 $\texttt{BExp}$) 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):

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

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

Exercise 1

Give the extended automaton for the following program:

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

Exercise 2

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

Exercise 3

Give the extended automaton for the following program:
y :=1 ;
$\texttt{while}\ \neg(x = y)\ \texttt{do}\ y := x * y; x := x - 1\ \texttt{od}$

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 $\mathcal{B} : \texttt{Bexp} \times \mathbf{Mem} \to \{tt, \mathit{ff}\}$ are the semantics functions defined as in the course.

Exercise 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 $x := a$.

Exercise 5

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

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:

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

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.

Exercise 6

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

Exercise 7

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

Exercise 8

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

Exercise 9

Give a necessary and sufficient syntactic condition such that each automaton A_i only modifies its set of variables 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 `unlock`(x) frees the variable x.

Exercise 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.
In the following, we suppose that this condition holds.

Exercise 11

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

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

Exercise 12

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

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

Exercise 13

Complete the following semantics rule for Boolean expressions.

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

Exercise 14

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

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

Exercise 15

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

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

Exercise 16

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

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.

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.

Exercise 17

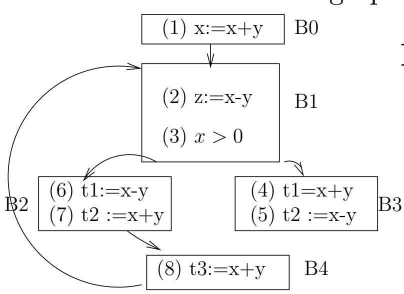
Give the semantics rule for the elementary statement `send`.

Exercise 18

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

2 Optimization (5pt)

We consider the control-flow graph below:



Exercise 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.

Exercise 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.

3 Hoare Logic (4pt)

Exercise 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
```

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 */
```

Exercise 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)`.