Amirkabir University of Technology
(Tehran Polytechnic)

Topic
Second Exercise of Programming Languages

Author
Amirmohammad Nazari

Professor
Mehran S. Fallah

# Answer to question 4.8 on page 85 in Concepts in Programming Languages

## Answer to part a

We can calculate the meaning via denotational semantics as following procedure.
$E[[x]](s_0) = 0$
$E[[x]](s_1) = 1$
$E[[x]](s_2) = 2$
$C[[x := 1; x := x + 1]](s_0) = C[[x := x + 1]](C[[x := 1]](s_0)) = C[[x := x+1]](s_1) = modify(s_1, x, E[[x+1]](s_1)) = modify(s_1, x, E[[x]](s_1)+1) = s_2$

## Answer to part b

In fact, states denote values of variables. In addition, variable x is only variable in the expression. Furthermore, value of x is 0 in state $s_0$ and value of x is 2 in state $s_2$. As a result, we can imply that
$C[[x := 1; x := x + 1]](s_0) = s_2$ and $C[[x := 2]]](s_0) = s_2$. Obviously, they are equivalent.

# Answer to question 4.9 on page 85 in Concepts in Programming Languages

## Answer to part a

We can calculate the meaning via denotational semantics as following procedure.
$C[[x := 0; y := 0; if\ x = y\ then\ z := 0\ else\ w := 1]](s_0) = C[[y := 0; if\ x = y\ thenz := 0\ else\ w := 1]](C[[x := 0]](s_0)) = C[[y := 0; if\ x = y\ then\ z := 0\ else\ w := 1]](modify(s_0, x, 0)) = C[[y := 0; if\ x = y\ then\ z := 0\ else\ w := 1]](s_1) = C[[if\ x = y\ then\ z := 0\ else\ w := 1]](C[[y := 0]](s_1)) = C[[if\ x = y\ then\ z := 0\ else\ w := 1]](modify(s_1, y, 0)) = C[[if\ x = y\ then\ z := 0\ else\ w := 1]](s_2) = if\ E[[x = y]](s_2)\ then\ C[[z := 0]](s_2)\ else\ C[[w := 1]](s_2) = C[[z := 0]](s_2) = modify(s_2, z, 0) = s_3$

### Answer to part b

$C[[if\ x = y\ then\ z := y\ else\ z := w]](s) = if\ E[[x = y]](s)\ then\ C[[z := y]](s)\ else\ C[[z := w]](s) = C[[z := y]](s) = modify(s, z, y) = s'$

## Answer to question 4.10 on page 85 in Concepts in Programming Languages

### Answer to part a

We can calculate the meaning of the expression as following procedure.

$\nu[[if\ false\ then\ 0\ else\ 1]]\eta_0 =$

$$\left\{ \begin{array}{cc} boolean & if\ \nu[[false]]\eta_0 = \nu[[0]]\eta_0 = \nu[[1]]\eta_0 = boolean \\ integer & if\ \nu[[false]]\eta_0 = boolean\ and\ \nu[[0]]\eta_0 = \nu[[1]]\eta_0 = integer \\ type\_error & otherwise \end{array} \right. =$$

$$\left\{ \begin{array}{cc} boolean & if\ boolean = integer = integer = boolean \\ integer & if\ boolean = boolean\ and\ integer = integer = integer \\ type\_error & otherwise \end{array} \right. =$$

$integer$

### Answer to part b

We can calculate the meaning of the expression as following procedure.

$let\ x : int = e_1\ in\ (if\ e_2\ then\ e_1\ else\ x) =$

$$\left\{ \begin{array}{cc} \nu[[(if\ e_2\ then\ e_1\ else\ x)]](\eta[x \rightarrow integer]) & if\ \nu[[e_1]]\eta_0 = integer\ and\ int = int \\ \nu[[(if\ e_2\ then\ e_1\ else\ x)]](\eta[x \rightarrow boolean]) & if\ \nu[[e_1]]\eta_0 = boolean\ and\ int = boolean \\ type\_error & otherwise \end{array} \right. =$$

$$\left\{ \begin{array}{cc} \nu[[(if\ e_2\ then\ e_1\ else\ x)]](\eta[x \rightarrow integer]) & if\ integer = integer\ and\ int = int \\ \nu[[(if\ e_2\ then\ e_1\ else\ x)]](\eta[x \rightarrow boolean]) & if\ integer = boolean\ and\ int = boolean \\ type\_error & otherwise \end{array} \right. =$$

$\nu[[(if\ e_2\ then\ e_1\ else\ x)]](\eta[x \rightarrow integer]) =$
$\nu[[(if\ e_2\ then\ e_1\ else\ integer)]]\eta_1 =$

$$\left\{ \begin{array}{cc} boolean & if\ \nu[[e_2]]\eta_1 = \nu[[e_1]]\eta_1 = \nu[[integer]]\eta_1 = boolean \\ integer & if\ \nu[[e_2]]\eta_1 = boolean\ and\ \nu[[e_1]]\eta_1 = \nu[[integer]]\eta_1 = integer \\ type\_error & otherwise \end{array} \right. =$$

$$\{ \begin{array}{cc} boolean & if\ boolean = integer = integer = boolean \\ integer & if\ boolean = boolean\ and\ integer = integer = integer \\ type\_error & otherwise \end{array} = integer$$

## Answer to question 4.11 on page 87 in Concepts in Programming Languages

### Answer to part a

Obviously, it is possible that one process will have to wait for another to complete. For example , we start to evaluate g and $e_1$ and $e_2$ in parallel. $e_1$ has massive calculation so it can not calculate the value by the time g reach to $x = 0$ condition. As a result, g needs to wait until evaluation of $e_1$ terminates.

### Answer to part b

Firstly, if calculation is lazy then The program terminates normally without error since the program won't calculate the value of $e_2$. Secondly, if calculation is parallel then The program terminates with an error since the program calculates the value of $e_2$ in parallel. As a result, The $e_2$ calculation terminates with an error.

### Answer to part c

In fact, if we evaluate the expressions in parallel then program terminates with an error based on the question. I feel this way because we are going to evaluate g and $e_1$ and $e_2$ in parallel. As a result, evaluation of $e_2$ terminates with an error.

### Answer to part d

In fact, $e_1$ and $e_2$ have a shared variable. As a result, different orders of evaluation may give different results. Accordingly, if we want to evaluate in parallel then concurrency problems occur. I believe that parallelism is not possible here.

# Answer to question 4.12 on page 88 in Concepts in Programming Languages

## Answer to part a

Process 1 can be executed in parallel with nothing. Process 1 must wait for nothing.
Process 2 can be executed in parallel with process 3. Process 2 must wait for process 1.
Process 3 can be executed in parallel with nothing. Process 3 must wait for process 1.
Process 4 can be executed in parallel with nothing. Process 4 must wait for process 1, process 2, process 3.
Process 5 can be executed in parallel with nothing. Process 5 must wait for process 1, process 2, process 3, process 4.
Process 6 can be executed in parallel with nothing. Process 6 must wait for process 1, process 2, process 3, process 4, process 5.
Process 7 can be executed in parallel with nothing. Process 7 must wait for process 1, process 2, process 3, process 4, process 5.

## Answer to part b

Accordingly, you can evaluate g(x) and h(x) in parallel since they are independent processes completely. However, value of g(x) and h(x) evaluated in Process 2 and Process 3.

## Answer to part c

If the program does not satisfy the single assignment condition then parallelism is not possible. In fact, if the program does not satisfy the single assignment condition then processes have side effects on other processes.

## Answer to part d

I believe that it is impossible to recognize the programs which satisfy single assignment condition since we can reduce it to halting problem which is undecidable. However, if a program contains at most one assignment of value to variable for all variables then the program satisfies the single assignment condition.

## Answer to part e

Obviously, a single assignment language pass the declarative language test since single assignment does not violates declarative perspective in programming languages. In fact, programmers implement declarative programming languages on machines with a similar idea. Each variable may be assigned a value only once within the scope of the variable.