Chapter 4

4.8 Denotational Semantics

a)

$S_0 = \{(x, 0)\}, S_1 = \{(x, 1)\}, S_2 = \{(x, 2)\}$
$C[[x:=1; x:=x+1]](S_0) = C[[x:=x+1]](C[[x:=1]](S_0)) = C[[x:=x+1]](S_1) = S_2$

b)

چون در تمامی stateها مقدار اولیه x صفر است، اگر مقدار آن را ۱ قرار دهیم و سپس یک واحد افزایش دهیم، مقدار نهایی آن ۲ خواهد بود.

4.9 Semantics of Initialize-Before-Use

a)

$$C[[x := 0; y := 0; if\ x = y\ then\ z := 0\ else\ w := 1\ ]](s_0)$$

$$= C[[y := 0; if\ x = y\ then\ z := 0\ else\ w := 1\ ]](C[[x := 0]\ (s_0))$$

$$= C[[y := 0; if\ x = y\ then\ z := 0\ else\ w := 1\ ]](s_1)$$

$$= C[[if\ x = y\ then\ z := 0\ else\ w := 1\ ]](C[[y := 0]\ (s_1))$$

$$= C[[if\ x = y\ then\ z := 0\ else\ w := 1\ ]](s_2)$$

$$= if\ E[[x = y]](s_2) = error\ or\ C[[z := 0]](s_2) = error\ or\ C[[w := 1]](s_2)$$

$$= error\ then\ error\ else\ C[[z := 0]](s2) \oplus\ C[[w := 1]](s_2)$$

$S_0 = \{(x, uninit), (y, uninit), (z, uninit), (w, uninit)\}$
$S_1 = \{(x, 0), (y, uninit), (z, uninit), (w, uninit)\}$
$S_2 = \{(x, 0), (y, 1), (z, uninit), (w, uninit)\}$

b)

C[[if x = y then z := y else z := w]](s) = if E[[x = y]](s) =
error or c[[z := 0]](s) = error or c[[w := 1]](s) =
error then error else c[[z := 0]](s) $\oplus$ c[[w: = 1]](s)

s = {(x, init) , (y, init) , (z, uninit) , (w, uninit)}

4.10 Semantics of Type Checking

a)

$V[[if\ false\ then\ 0\ else\ 1]](0)$

$type\ checking: V[[false]](0) = boolean\ and\ V[[0]](0) = V[[1]](0) = integer$

$\Rightarrow V[[if\ false\ then\ 0\ else\ 1]](0) = integer$

b)

$V[[let\ x: int = e_1\ in\ (if\ e_2\ then\ e_1\ else\ x)]]0$

$type\ checking: if\ V[[e_1]] = integer\ and\ V[[e_2]] = boolean$

$\Rightarrow V[[let\ x: int = e_1\ in\ (if\ e_2\ then\ e_1\ else\ x)]]0 = boolean$

c)

$$V[[let\ x = e1\ in\ e2\ ]] = \begin{cases} V[[e_2(\eta)[x \rightarrow \sigma] & if\ V[[e_1]] = V[[e_2]] \\ type\_error & otherwise \end{cases}$$

4.11 Lazy Evaluation and Parallelism

a)

بله؛ در صورتی که e_1 منتظر e_2 و یا e_2 منتظر e_1 باشد.

مانند توضیحات سوال، این حالت ممکن است پیش بیاید اگر یک عملیات به جای یک عدد به عنوان ورودی به تابع داده‌شود.

b)

در Lazy evaluation، اگر e_1 صفر باشد تابع g، ۱ برمی‌گرداند و e_2 ارزیابی نمی‌شود؛ اما در Parallel evaluation، چون e_2 هم ارزیابی می‌شود، خروجی نهایی error خواهدبود.

c)

از آن‌جایی که ارزیابی به‌صورت Parallel است و ارزیابی e_2 به error منتهی شده‌است، خروجی error خواهد بود.

برای ارزیابی تابع g، می‌توانیم ابتدا e_1 و e_2 را به‌صورت موازی و سپس تابع g را ارزیابی کنیم.

d)

اگر بخواهیم به‌صورت Parallel evaluation عمل کنیم، e_1 باید قبل از e_2 ارزیابی شود؛ برای این کار می‌توانیم قبل از ارزیابی توابع، به assignmentها رسیدگی کنیم؛ در غیر این صورت ممکن نخواهدبود.

4.12 Single-Assignment Languages

a)

فرایندهای دوم و سوم و فرایندهای ۶ و ۷ (از آنجا که فقط یکی از فرایندهای ۶ یا ۷ می‌توانند اجرا شوند، می‌توانیم آن‌ها را نیز موازی در نظر بگیریم) می‌توانند به‌صورت موازی اجرا شوند؛ اما بقیه فرایندها باید به ترتیب اجرا شوند.

b)

بله، می‌توان دو تابع g و h را به‌صورت موازی فراخوانی کرد.

اگر کامپایلر تمام مقادیری را که در یک scope هستند را نگه‌داری کند، می‌توان فراخوانی دو تابع g و h را از فرایندهای ۶ و ۷ حذف کرد؛ چرا که مقدار x تغییری نکرده‌است(single-assignment)؛ پس، خروجی دو تابع g و h نیز تغییری نمی‌کنند.

c)

اگر برنامه شرط single-assignment را رعایت نکند، نمی‌توان فراخوانی توابع g و h را هم‌زمان انجام داد؛ چرا که ممکن است مقدار x تغییر کند.

d)

خیر؛ این مسئله معادل مسئله‌ی halting است که Undecidable است.

اگر بتوان مقداردهی را به یک بار محدود کرد (چیزی شبیه 'static' در تعریف متغیرها)، می‌توان برنامه را

single-assignment در نظر گرفت.

e)

بله؛ از آن‌جا که هیچ side-effect operationی وجود ندارد و single-assignment هم هست، در هر scope مقداردهی‌ها یک بار انجام می‌شوند؛ پس یک زبان declarative است.