CS 520

Theory of Programming Language

03/17 – 03/31, 2021

1. Reminder

(predomains. , Contin. fns.
domains.) least fixed-point thm. (recursively defined domain)

① Simple imp. lang. — syntax, domain theory

② Denotational semantics.

$[\![-]\!]_{intexp} : \langle intexp \rangle \to [\Sigma \to_c \mathbb{Z}]$] similar to what we studied.

$[\![-]\!]_{boolexp} : \langle boolexp \rangle \to [\Sigma \to_c \mathbb{B}]$

$[\![-]\!]_{comm} : \langle comm \rangle \to [\Sigma \to_c \Sigma_\perp]$ — new.

①

$[\![skip]\!] \sigma = \sigma$

$[\![x := e]\!] \sigma = [\sigma \mid x : [\![e]\!] \sigma]$

$[\![c_1 ; c_2]\!] \sigma = [\![c_2]\!]_\perp ([\![c_1]\!] \sigma)$

$[\![\text{if } b \text{ then } c_1 \text{ else } c_2]\!] \sigma = \text{if } [\![b]\!] \sigma = tt \text{ then } [\![c_1]\!] \sigma \text{ else } [\![c_2]\!] \sigma.$

$[\![\text{while } b \text{ do } c]\!] \sigma = Y_{[\Sigma \to \Sigma_\perp]} (F)(\sigma)$

$F : [\Sigma \to \Sigma_\perp] \to_c [\Sigma \to \Sigma_\perp]$

$F(g)(\sigma') = \text{if } [\![b]\!] \sigma' = tt \text{ then } g_\perp ([\![c]\!] \sigma') \text{ else } \sigma'$

2. Justification of the semantics. , especially the one for while.

while b do c "=" if b then (c ; while b do c) else skip.

↑ intuitive eq. — c′

① " says that ⟦while ... ⟧ should be a fixed point of F

⟦ if b then (c ; c′) else skip ⟧ = F(⟦c′⟧).

①

RHS = F(⟦while b do c⟧) = ⟦while b do c⟧ = LHS

∴ ⟦while . . ⟧ should be a fixed point of F.

(2) Why least?

(1) ···· unless forced by loop unrolling, the meaning of while ... at b. should be ⊥.

(2) ⟦while true do skip⟧ = Y(F).

$F : [\Sigma \to_c \Sigma_\perp] \to_c [\Sigma \to_c \Sigma_\perp]$

$F(g) = g$

$F(g)(b) =$ if ⟦true⟧ b then g_μ(⟦skip⟧ b) else b.

$= g_\mu$(⟦skip⟧ b) $= g_\mu(b) = g(b)$

∴ every g is a fixed point of F.

least. $g_1(b) = \perp$.

non-least but fixed point $g_2(b) = b$.

$\llbracket \text{while } b \text{ do } c \rrbracket$?

$\downarrow$

F

(3) $w_0 = \text{while true do skip}.$

$w_1 = \text{if } b \text{ then } c ; w_0 \text{ else skip}.$

$w_2 = \text{if } b \text{ then } c ; w_1 \text{ else skip}.$

$w_n \approx \text{while } b \text{ do } c$ for all states σ for which the loop terminates at $< n$ iterations.

$[\Sigma \to_c \Sigma_\bot]$

$\Downarrow$

$\llbracket w_n \rrbracket = F^n(\bot).$

$\bot = (\lambda b.\, \bot)$

$\star \; Y(F) = \bigsqcup_n F^n(\bot) = \bigsqcup_n \llbracket w_n \rrbracket \;\text{" = "}\; \llbracket \text{while } b \text{ do } c \rrbracket$

show

ex. $[\![x:=e_1; y:=e_2]\!]$

$= [\![y:=e_2; x:=e_1]\!]$

if $x \notin FV(e_2)$,
$y \notin FV(e_1)$
$x \neq y$.

ans. $[\![x:=e_1; y:=e_2]\!]\rho$

$= [\![y:=e_2]\!]_\Pi([\![x:=e_1]\!]\rho)$

$= [\![y:=e_2]\!]_\Pi(\rho[\rho(e_1)/x])$

$= \rho[\rho(e_1)/x][\rho(e_2)/y]$

$= \rho[\rho(e_2)/y][\rho(e_1)/x]$

$[\![y:=e_2; x:=e_1]\!]\rho$

$= \rho[\rho(e_1)/x][\rho(e_2)/y]$

□

3. Extension of the lang.

(1) local variable decl.

<comm> ::= ..... | newvar <var> := <intexp> in <comm>.

$[\![c]\!] : \Sigma \to \Sigma_\perp$.

ex.

$[\![\text{newvar } x := e \text{ in } c]\!]\sigma = (\lambda\sigma'.\ [\sigma' | x : \sigma(x)])\,([\![c]\!]([\sigma | x : [\![e]\!]\sigma]))$

- $\sigma(x)$: restore the value of global x.
- $[\![e]\!]\sigma$: value of global x

(a) FV(c) defined in a syn. directed way.

$$FV(c_1 ; c_2) = FV(c_1) \cup FV(c_2)$$

$$FV(x := e) = FV(e) \cup \{x\}.$$

$$FV(\text{newvar } x := e \text{ in } c) = (FV(c) \setminus \{x\}) \cup FV(e).$$

(b) FA(c) syntax directed.

$$FA(c_1 ; c_2) = FA(c_1) \cup FA(c_2)$$

$$FA(x := e) = \{x\}$$

$$FV(\text{newvar } x := e \text{ in } c) = FA(c) \setminus \{x\}.$$

✱

Lemma. [Correspondence].

1) c ... command.
σ, σ' ... states in Σ. s.t. $\sigma(\omega) = \sigma'(\omega)$ for $\omega \in FV(c)$. ✱

$\sigma \sim_{FV(c)} \sigma'$

$\Rightarrow$ either $[\![c]\!]\sigma = [\![c]\!]\sigma' = \perp$

or $[\![c]\!]\sigma, [\![c]\!]\sigma' \in \Sigma$ and $([\![c]\!]\sigma)(\omega) = ([\![c]\!]\sigma')(\omega)$ for all $\omega \in FV(c)$.

2) c ... command, σ ... state. s.t. $[\![c]\!]\sigma \neq \perp$.

$\Rightarrow$ for all $\omega \notin FA(c)$, $\sigma(\omega) = ([\![c]\!]\sigma)(\omega)$.

(2) Syntactic Sugar. ... define a lang. const. as a macro.

$\underline{\text{twice}}(c) \overset{\text{def.}}{=} c ; c.$

$\overset{s}{\overset{\vee}{x}+2}$

$(\text{for } \overset{\vee}{x} := e_1 \text{ to } e_2 \text{ do } c.) \overset{\text{def.}}{=} \text{newvar } y := e_2 \text{ in}$
$\text{newvar } x := e_1 \text{ in}$
$\text{while } (x \leq y) \text{ do } (c ; x := x+1).$

(y is fresh) $FV(c) \not\ni x.$

$\text{for } x := e_1 \text{ to } e_2 \text{ do } c \overset{\text{def.}}{=} x := e_1 ; \text{while } (x \leq e_2) \text{ do}$
$(c ; x := x+1)$

↑ global var.

4. Quality of semantics. How can we measure it formally?

(1) Soundness, full abstraction.

c "=" c'

↳ intuitive notion of equality.

$[\![c]\!] = [\![c']\!]$ $\Longrightarrow$ (sound.) c "=" c'

$\Longleftrightarrow$ (fully abstract)

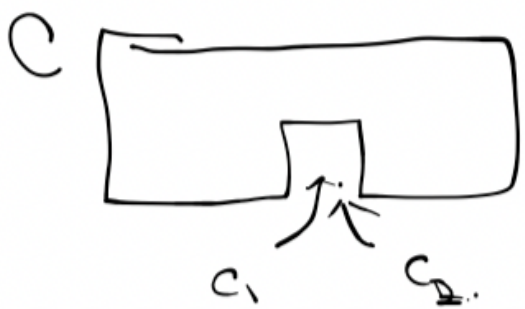
(2) observational equivalence.

ⓐ a set of observable phrases with a hole. $\mathcal{C}$. ← observable contexts.

ⓑ a set of observations. $\mathcal{O} : \langle comm \rangle \to \mathbb{B}$.

$$c =_{obs} c' \quad \text{iff} \quad \forall C \in \mathcal{C}.\ \forall O \in \mathcal{O}.\ O(C[c_1]) = O(C[c_2])$$

$C[c_1]$ ← use case of c_1.



C.

(c) example.

$$\mathcal{C}_{c_1,c_2} = \{ \; \boxed{\begin{array}{l} \text{newvar } x_1 := k_1 \text{ in} \\ \quad\vdots \\ \text{newvar } x_n := k_n \text{ in} \\ *\,[\;\;] \\ \text{if } x_i = k \text{ then skip} \\ \text{else while true do skip} \end{array}} \;\Big|\; \begin{array}{l} k, k_1, \cdots, k_n \in \mathbb{Z}. \\ \{x_1, \cdots x_n\} = FV(c_1) \\ \qquad \cup FV(c_2) \\ i \in \{1, \cdots, n\} \end{array} \; \}.$$

$$\mathcal{O} = \{ \; \lambda c. \text{ if } \llbracket c \rrbracket \sigma_0 = \perp \text{ then } tt \text{ else } ff \; \}.$$

where $\sigma_0 = \lambda x.0$

* $\underline{x} := y+z \;;$ if $x > w$ then $\underline{z} := z+1$ else skip.

FV ... x, y, z, w.

FA ... z, x