CS 520

Theory of Programming Language

03/17 – 03/31, 2021

1. Reminder.

① Simple imperative lang.

② Domain theory ⋯ Approx. computability machine.

- Poset. $(S, \sqsubseteq)$
- Predomain $(P, \sqsubseteq)$ ⋯ every chain $x_0 \sqsubseteq x_1 \cdots$ has the lub. $\bigsqcup_n x_n$.
- Domain $(D, \sqsubseteq, \perp)$.
- $f : (P, \sqsubseteq) \to (P', \sqsubseteq')$.

$\left[\begin{array}{ll} f & \text{monotone.} \\ f & \text{continuous.} \\ f & \text{strict.} \end{array}\right.$

- Least fixed-point thm.

$(D, \sqsubseteq, \bot)$ ... domain.

$f : D \to D$ .... continuous.

$\Rightarrow$ f has the least fixed point. $x \in D$.

$\bigsqcup_n f^n(\bot)$

1) $f(x) = x$

2) $\forall y \in D.\ f(y) = y \Rightarrow x \sqsubseteq y$.

2. Intuition about continuity.

$(P, \sqsubseteq)$, $(P', \sqsubseteq')$ ... Predomains.

$f : P \to P'$ ... monotone.

f is continuous intuitively iff. in order to produce a finite amount of information in the output, f uses only a finite amount of info. in the input.

abcd...... $\xrightarrow{f}$ 가나다라 ....

finite. inf.

$(\mathbb{Z}^{*,\omega}, \sqsubseteq)$ $\qquad$ $(2^{\mathbb{Z}}, \subseteq)$

$\{\langle -2,-3\rangle, \langle -3\rangle, \langle 1,2,3,\cdots\rangle, \cdots\}$ $\qquad$ $\{\{2\}, \{4,5\}, \cdots\}$.

$f : \mathbb{Z}^{*,\omega} \longrightarrow 2^{\mathbb{Z}}$ monotone.

(Note: $s' \sqsubseteq s$. So, $f(s') \sqsubseteq f(s)$)

ex. If f is continuous, then it satisfies the following property:

$\forall s \in \mathbb{Z}^{*,\omega}.\ \forall A \subseteq_{fin} f(s).\ \exists$ a finite prefix of s (s') s.t. $A \subseteq f(s')$.

Prove this statement.

Answer: Assm. f is cont. Pick $s \in \mathbb{Z}^{*,\omega}$ and $A \subseteq_{fin} f(s)$.

If s is finite, we can just set $s' = s$.

Otherwise, $s = \langle x_0, x_1, x_2, x_3, \ldots \rangle$ for $x_0, x_1, \ldots \in \mathbb{Z}$.

We create a chain $s_0 = \langle\rangle$

$s_1 = \langle x_0 \rangle$

$s_2 = \langle x_0, x_1 \rangle$

$\ldots$

Then, $\bigsqcup_n s_n = s$. Furthermore, s_n is finite.

Since f is cont., $f(s) = f(\bigsqcup_m s_n) = \bigcup_n f(s_n)$.

$A \subseteq_{fin} \bigcup_n f(s_n)$. and $f(s_n) \subseteq f(s_{n+1})$.

$\therefore$ $\exists m$. s.t. $\underbrace{A \subseteq \bigcup_{n=0}^{m} f(s_n) = f(s_m)}$

s_m is a finite prefix of s and $\subseteq$.

$\square$

3. Two constructions for predomains.

① Function Space: $(P, \sqsubseteq)$, $(P', \sqsubseteq')$.

the set of cont. func. from P to P'.

$(P'', \sqsubseteq'')$ where

✓ $P'' = [P \to_c P']$.

✓ $\sqsubseteq''$ ... pointwise order.

$f, g \in P'' = [P \to_c P']$.

order.

$f \sqsubseteq'' g$ iff $f(x) \sqsubseteq' g(x)$ for $x \in P$.

pointwise.

$\sqsubseteq$

Thm. $([P \to_c P'], \sqsubseteq'')$ is a predomain.

(refl. tran. anti-sy.)

(every chain has ~~a~~ the lub)

x.

$$\left[\begin{array}{l} (f_n)_{n \in \mathbb{N}} \ \ldots \text{ a chain in } [P \to_c P'] \\ \text{the lub of the chain is: } \underset{P}{x} \longmapsto \underbrace{\bigsqcup_n \big(\overset{P'}{f_n(x)}\big)}_{P'} \end{array} \right]$$

R.

discrete order.

Facts.

① $(S, =)$ ..... predomain

$\sqsubseteq$

$$[S \to_c P'] = [S \to P'] . \quad \cdots\cdots$$

② P' is a domain (i.e., P' has the least element $\bot'$)

$\Rightarrow$. $[P \to_c P']$ is also a domain.

the least element $x \longmapsto \bot'$.

$\lambda x. \bot'$.

Thm. $(D, \sqsubseteq, \perp)$ ... domain.

$Y : [D \to_c D] \to D$

$Y(f)$ = the least fixed point of f = $\bigsqcup_n f^n(\perp)$

$\Rightarrow Y$ is continuous.

Proof. ex.

② Lifting: $(P, \sqsubseteq)$ --- predomain

$\hookrightarrow (P_\perp, \sqsubseteq')$ ---- output of this lifting construction.

$$P_\perp = P \cup \{\perp\}.$$

$\uparrow$ new element not in P.

$x, y \in P_\perp$ $\quad$ $x \sqsubseteq' y$ iff $x = \perp$ or $(x, y \in P$ and $x \sqsubseteq y)$.

<u>e.g.</u> $(\mathbb{Z}, =)$

$(\mathbb{Z}_\perp, \sqsubseteq)$

Lemma. $(P_\perp, \sqsubseteq')$ is a domain.

Why? 1) the least element is $\perp$ just added

2) the lub can be essentially computed using the lub in P.

(1) unit operation. $\iota_\uparrow : P \to_c P_\perp$, $\iota_\uparrow(a) = a$.

(unit).

(2) Kleisli extension. $f : P \to_c P'_\perp$ $(P', \sqsubseteq')$ ... predomain.

$f_{\perp\!\!\perp} : P_\perp \to_c P'_\perp$

$$f_{\perp\!\!\perp}(x) = \begin{cases} \perp' & \text{if } x = \perp \\ f(x) & \text{otherwise} \end{cases}$$

* $f : P \to_c P'_\perp \ , \ g : P' \to_c P''_\perp$

$$g \circ' f = g_{\perp\!\!\perp} \circ f.$$

normal fn compo.

behaves very much like compo.

$k : P'' \to P'''_\perp.$

- $\circ'$ is associative, $\to (k \circ' g) \circ' f = k \circ' (g \circ' f)$
- $\circ'$ has "identity element" $\underbrace{\iota_\uparrow}_{P'} \circ' f = f = f \circ' \underbrace{\iota_\uparrow}_{P}.$

↳ monad.

(3). Lifting $\quad f : P \to_c P'$,

$$f_\perp : P_\perp \to_c P'_\perp.$$

$$f_\perp(x) = \begin{cases} \perp & \text{if } x = \perp. \\ \lfloor f(x) \rfloor & \text{otherwise.} \end{cases}$$

1) ev. of all int. & bool. expressions terminates w/o errors.

4. Semantics of the simple imp. lang.

① Syntax-directed def. ... deno. semantics.

② Interpretation of types/non-terminals in our ab. grammar.

$\langle intexp \rangle$. $\quad [\![-]\!]_{intexp} : \langle intexp \rangle \to [\Sigma \to \mathbb{Z}]$ $\quad (\Sigma = [\langle var \rangle \to \mathbb{Z}])$

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

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

3) all vari. store only integers.

2) commands may go on forever.

(non-term. or error stop)

$[\![e]\!]\sigma$ (intexp) ... same as before.

$[\![b]\!]\sigma$ (boolexp) ... 〃

$$[\![\text{skip}]\!]\sigma = \sigma$$

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

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

$$\left(\begin{array}{l} [\![c_2]\!] : \Sigma \to \Sigma_\perp \\ [\![c_2]\!]_{\perp\!\!\perp} : \Sigma_\perp \to \Sigma_\perp \end{array} \right)$$

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

syntax

english

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

function space domain

least fixed point thm

$$Y_D : [D \to_c D] \to_c D$$

$$[\Sigma \to_c \Sigma_\perp].$$

1) where this def'n. comes from? why fixed point of F?

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

$\underset{\sim}{F}(f)(\sigma') = \text{if } (\llbracket b \rrbracket \sigma' = tt) \text{ then } f_\perp(\llbracket c \rrbracket \sigma') \text{ else } \sigma'$

input state

loop after one unrolling of while

2) why <u>least</u> fixed point?