CS 520
Theory of Programming Language

05/26 – 06/02, 2021

1. Reminder.

<tag> ::= 0 | 1 | 2 | ···

① <exp> ::= <var> | λ<var>.<exp> | <exp> <exp>

| true | false | if <exp> then <exp> else <exp> | <exp> ∧ <exp> | <exp> > <exp>

∨
⇒
⋮

| ... -1 | 0 | 1 ... | <exp> + <exp>

| <<exp>, ··· , <exp>> | <exp>.<tag>

| @ <tag> <exp> | sumcase <exp> of (<exp>, ··· , <exp>)

| letrec <var> ≡ λ<var>.<exp> in <exp>

<cfm> ::= <intcfm> | <boolcfm> | <funcfm> | <tuplecfm> | <altcfm>

<intcfm> ::= ... -2 | -1 | 0 | 1 | 2 ···   <boolcfm> ::= true | false

<funcfm> ::= λ<var>.<exp>   <tuplecfm> ::= <<cfm>, ··· , <cfm>> ✓

<altcfm> ::= @ <tag> <cfm> ✓

2. Denotational Semantics.

① Reminder of the deno. Semantics of the core lambda cal.

values, cfns.

comp.

$$D = (V)_\perp \qquad V \simeq [V \to_c V_\perp] = [V \to_c D].$$

general exprs.

$$[\![-]\!] \in [\langle exp \rangle \to E \to D], \quad E = [\langle var \rangle \to V]$$

V_*

V_*

V_*

✓

② Spaces for computations and values ...... domain

$V_* \stackrel{\text{def}}{=} (V + \{\text{error}, \text{typeerror}\})_\perp$ ...... predomains

$V \underset{\psi}{\overset{\phi}{\rightleftarrows}} V_{int} + V_{bool} + V_{fun} + V_{tuple} + V_{alt}$

(ψ, ϕ ... isomorphisms)

$V_{int} \stackrel{\text{def}}{=} \mathbb{Z} = \{\ldots, -1, 0, 1, 2, \cdots\}$. $V_{bool} \stackrel{\text{def}}{=} \{tt, ff\} = \mathbb{B}$.

$V_{fun} \stackrel{\text{def}}{=} [V \to_c V_*]$. $V_{tuple} \stackrel{\text{def}}{=} \bigcup_{n \geq 0} V^n$ $V_{alt} \stackrel{\text{def}}{=} \mathbb{N} \times V$

$\underbrace{V \times V \times \cdots \times V}_{n} \to \langle v_1, \cdots, v_n \rangle$

$⁂ \; [\![-]\!] \in [\langle exp \rangle \to E \to V_*]$. , $E = [\langle var \rangle \to V]$.

$(-)_* = T$

$(V')_* = (V' + \{..\})_\perp$

③ Side remark ... General principle (semantics based on monad).

V , $T(V)$, $V \to_c T(V)$, $E = [\langle var \rangle \to V]$

space for values   space for computations. $[\![-]\!] \in [\langle exp \rangle \to E \to T(V)]$.

strong monad ... sequencing, unit.

④ A few auxiliary fns :

$\bar{\iota}_{norm} : V \to_c V_*$, $\bar{\iota}_{norm}(v) \stackrel{def}{=} \langle 0, v \rangle$. $err \stackrel{def.}{=} \langle 1, error \rangle$. $tyerr \stackrel{def}{=} \langle 1, typeerror \rangle$

$\bar{\iota}_{int} : V_{int} \to V$, $\bar{\iota}_{int}(n) = \psi(\langle 0, n \rangle)$ $\quad$ $\bar{\iota}_{bool} : V_{bool} \to V$, $\bar{\iota}_{bool}(b) = \psi(\langle 1, b \rangle)$.

$\bar{\iota}_{fun} : V_{fun} \to V$, $\bar{\iota}_{fun}(f) = \psi(\langle 2, f \rangle)$. $\quad$ $\bar{\iota}_{tuple} : V_{tuple} \to V$, $\bar{\iota}_{tuple}(t) = \psi(\langle 3, t \rangle)$

$\bar{\iota}_{alt} : V_{alt} \to V$, $\bar{\iota}_{alt}(a) = \psi(\langle 4, a \rangle)$.

easy case analysis

$f : V \to V_*$ ... given.

$f_* : V_* \to V_*$, $f_*(d) = \begin{cases} f(v) & \text{if } d = \bar{\iota}_{norm}(v). \\ d. & \text{otherwise.} \end{cases}$

($f_{\perp\!\!\perp}$)

$g : V_\theta \to V_*$ ... given , $\quad \theta \in \{int, bool, fun, tuple, alt\}$.

$g_{\theta *} : V_* \to V_*$, $g_{\theta *}(d) = \begin{cases} g(v) & \text{if } d = \bar{\iota}_{norm}(\bar{\iota}_\theta(v)) \\ tyerr & \text{if } d = \bar{\iota}_{norm}(\bar{\iota}_{\theta'}(v)),\ \theta' \neq \theta. \\ d. & \text{otherwise. } (d \neq \bar{\iota}_{norm}(v) \text{ for any } v) \end{cases}$

runtime type-checking & type-casting.

(5) $[\![-]\!] \in [\langle exp \rangle \to E \to V_*]$, $\eta \in E = [\langle var \rangle \to \underset{\sim}{V}]$.

$$[\![x]\!]\eta = \iota_{norm}(\eta(x)) \quad (= \langle 0, \eta(x) \rangle)$$

$$[\![\lambda x.e]\!]\eta = (\iota_{norm} \circ \iota_{fun})(\lambda v \in V.\ [\![e]\!][\eta \mid x : v])$$

$$[\![e_1\ e_2]\!]\eta = (\lambda f_1 \in V_{fun}.\ (\lambda v_2 \in V.\ f_1(v_2))_* ([\![e_2]\!]\eta))_{fun*} ([\![e_1]\!]\eta)$$

$i/_*$ $\Downarrow$

$$[\![\langle e_1, e_2, e_3 \rangle]\!]\eta = (\lambda v_1 \in V.\ (\lambda v_2 \in V.\ (\lambda v_3 \in V_3 (\underset{\iota_{norm}}{\circ}\ \iota_{tuple}(\langle v_1, v_2, v_3 \rangle))([\![e_3]\!]\eta))_* ([\![e_2]\!]\eta))_* ([\![e_1]\!]\eta)$$

$$[\![e.k]\!]\eta = \left(\lambda t \in V_{tuple}.\ \begin{array}{ll} \text{if } k \in dom(t) & \\ \text{then} & \iota_{norm}(t.k) \\ \text{else} & tyerr. \end{array}\right)_{tuple*} ([\![e]\!]\eta)$$

(b) How to interpret " letrec $x \equiv \lambda y. e$ in e' " ?

$$[\![\text{letrec } \underbrace{x \equiv \lambda y. \boxed{e}}_{} \text{ in } e']\!]\eta = [\![e']\!][\eta \mid x : \underline{v}]$$

$\boxed{e}$: $\ldots x \ldots$ → canonical form.

How to define v?

(1) recursively defined ... fixed point thm. → domains, conti. fns.

(2). Has to be a value. → V .... not a domain.

a function value .... $V_{fun} = [V \to_c V_*]$ .... domain.

$F : V_{fun} \to V_{fun}$. $\quad F(f)(w) = [\![e]\!][\eta \mid y : w \mid x : \iota_{fun}(f)]$

$v = \iota_{fun}(Y_{V_{fun}}(F))$.