CS 520

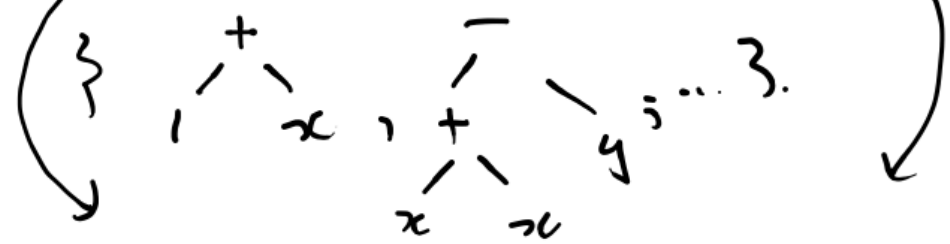
Theory of Programming Language

03/03 – 03/10, 2021

1. Reminder — integer expression

① <intexp> ::= 0 | 1 | ... | <var> | -<intexp> | <intexp> $\begin{matrix}+\\-\\\times\\\div\end{matrix}$ <intexp>.

② abstract grammar, abstract syntax, abstract phrase.



signature. initial algebra. elements of "

③ syntax-directed definition.

structural induction.

2. Syntax-directed definition

① definition on some operator/predicate on <intexp> specified by

② $FV : \langle intexp \rangle \to 2^{\langle var \rangle}$ ⤳ collection of all subsets of <var>.

$FV(c) = \phi$.

$FV(x) = \{x\}$

$FV(-e) = FV(e)$

$FV(e_1 + e_2) = FV(e_1) \cup FV(e_2)$

x

⋮ ....

ⓐ case analysis

ⓑ induction/recursion.

↳ wrt. immediate subexpressions.

exercise: Substitution. $\delta \in [\langle var \rangle \to \langle intexp \rangle]$ $\left(\begin{array}{l} \delta_0 = \; x \to x+1 \\ \delta_0(y) = y \end{array}\right)$

$\langle subst \rangle$ ... set of all substitutions.

$-/- \; : \; \langle intexp \rangle \times \langle subst \rangle \to \langle intexp \rangle$

$(x+y)/\delta_0 = (x+1)+y$

define $-/-$ in a syntax-directed way.

$\underline{c}/\delta = c$

$x/\delta = \delta(x)$

$(-e)/\delta = -(e/\delta) = \overline{1}$

$(e_1+e_2)/\delta = (e_1/\delta) + (e_2/\delta) =$ 

3. Denotational Semantics.

① Map syntactic objects (expressions, abstract phrases) to mathematical objects.

in a syntax-directed way. ..... define a meaning of each integer expression

compositional.

② Define a semantic domain ... key step in deno. semantics.

$[\![-]\!] : \langle \text{intexp} \rangle \to$ [ ] $\to [\Sigma \to \mathbb{Z}]$

$[\![x+y]\!] \in [\Sigma \to \mathbb{Z}]$ (state.)

$\Sigma = [\langle \text{var} \rangle \to \mathbb{Z}]$

$\sigma \in \Sigma \overset{\text{def.}}{=} [\langle \text{var} \rangle \to \mathbb{Z}]$.

(state.)

✓ $[\![c]\!] \in [\Sigma \to \mathbb{Z}] = \sigma \longmapsto c = \lambda\sigma.\, c$

✓ $[\![x]\!] \in [\Sigma \to \mathbb{Z}] = \sigma \longmapsto \sigma(x) = \lambda\sigma.\, \sigma(x)$

$[\![-e]\!] = \sigma \longmapsto -([\![e]\!]\sigma) = \lambda\sigma.\, -([\![e]\!]\sigma)$

(integer minus. (not tree operator).)

$[\![e_1 + e_2]\!] = \sigma \longmapsto ([\![e_1]\!]\sigma) + ([\![e_2]\!]\sigma) = \lambda\sigma.\, ([\![e_1]\!]\sigma) + ([\![e_2]\!]\sigma)$

(tree op. ; integer op. ; integer plus.)

1) unique algebra homo.

implicitly defines an algebra whose underlying set is $[\Sigma \to \mathbb{Z}]$.

2) fold_right.

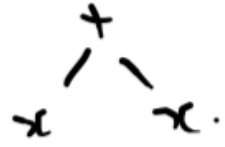
unique algebra homo.

exercise:

$[\![\Sigma \to \mathbb{Z}]\!]$

$\Downarrow$

$[\![2 * x]\!] \stackrel{?}{=} [\![x + x]\!] \in [\![\Sigma \to \mathbb{Z}]\!]$



$[\![2 * x]\!]\sigma \stackrel{?}{=} [\![x + x]\!]\sigma$ for all σ.

$\|$

$[\![2]\!]\sigma \times [\![x]\!]\sigma$

$\|$

$2 \times \sigma(x)$ $=$ $\sigma(x) + \sigma(x)$

$\|$

$[\![x]\!]\sigma + [\![x]\!]\sigma$

$\|$

$[\![x + x]\!]\sigma$

$=$ ↑ mathematical reasoning

4. Structural induction .... induction on integer expressions.
(like syntax-driven def.) where we can make ind. hypo on immediate subexpression.
1) case analysis. 2) induction hypo.

Lemma [Coincidence]

σ, σ' .... states. $(\sigma, \sigma' \in \Sigma = [\langle var \rangle \to \mathbb{Z}])$

e .... aexp.

If ① $\sigma(x) = \sigma'(x)$ for all $x \in FV(e)$, then $[\![e]\!]\sigma = [\![e]\!]\sigma'$

$[\![e]\!]\sigma \in [\Sigma \to \mathbb{Z}]$

$[\![e]\!]\sigma \in \mathbb{Z}$

Proof By structural induction. Let σ, σ' be states s.t. → ①

- $e \equiv c$:

$$[\![e]\!]\sigma \overset{?}{=} [\![e]\!]\sigma'$$
$$\| \qquad \| \qquad \| \qquad \|$$
$$c \quad = \quad c$$

- $e \equiv x$:

$$[\![x]\!]\sigma \overset{?}{=} [\![x]\!]\sigma'$$
$$\| \qquad \| \qquad \| \qquad \|$$
$$\sigma(x) \quad = \quad \sigma'(x) \quad , \; x \in FV(x).$$

- $e \equiv -e_1$:

$$[\![-e_1]\!]\sigma = -([\![e_1]\!]\sigma)$$
$$\overset{\text{induction hypo.}}{=} -([\![e_1]\!]\sigma') \quad (\because FV(e) = FV(e_1))$$
$$= [\![-e_1]\!]\sigma'$$

- $e \equiv e_1 + e_2$:

$$[\![e_1 + e_2]\!]\sigma \overset{?}{=} [\![e_1 + e_2]\!]\sigma'$$
$$\| \qquad \| \qquad \| \qquad \|$$
$$[\![e_1]\!]\sigma + [\![e_2]\!]\sigma \quad = \quad [\![e_1]\!]\sigma' + [\![e_2]\!]\sigma'$$

$FV(e) \supseteq FV(e_1)$, $FV(e) \supseteq FV(e_2)$ → ①

$\sigma(x) = \sigma'(x)$ for all $x \in FV(e_1)$ (" " $FV(e_2)$)

induction hypo.

$[\![e_1]\!]\sigma = [\![e_1]\!]\sigma'$, $[\![e_2]\!]\sigma = [\![e_2]\!]\sigma'$.

×
—
⋮

exercise.

Lemma [Substitution] .... correspondence bt. (syntactic) subst. & semantic subst.

δ ... subst. $\langle var \rangle \to \langle int\,exp \rangle$.

σ, σ' ... states.

e ... int. exp

$\sigma'(x) = [\![\delta(x)]\!]\sigma$. for all $x \in FV(e)$. ..... inf. notation. $\sigma' = \sigma / \delta$.

$\Rightarrow$ $[\![e/\delta]\!]\sigma = [\![e]\!]\sigma'$.

$[\![e/\delta]\!]\sigma = [\![e]\!](\sigma/\delta)$.

Prove this Lemma by structural induction.

Proof By structural ind. on e. σ, σ', v, v' are as given in the lemma.

- $e \equiv c$: $\llbracket e \rrbracket \sigma = \llbracket c \rrbracket \sigma = v_c = c$. $\checkmark$
  $\llbracket e \rrbracket \sigma' = \llbracket c \rrbracket \sigma' = c$.

- $e \equiv x$: $\llbracket x \rrbracket \sigma = \sigma(x) = v(x) = \llbracket x \rrbracket v$. $\checkmark$ (2)

- $e \equiv e_1 + e_2$: $\llbracket (e_1 + e_2) \rrbracket \sigma = \llbracket (e_1) \rrbracket \sigma + \llbracket (e_2) \rrbracket \sigma$
  $= \llbracket e_1 \rrbracket \sigma + \llbracket e_2 \rrbracket \sigma$
  $= \llbracket e_1 \rrbracket v + \llbracket e_2 \rrbracket v = \llbracket e_1 + e_2 \rrbracket v$

ind. hypo.

σ, v (2) " for e.
σ, σ' " for e_1
v, v' " for e_2.
$\therefore$ ind. hypo.

$\square$

Cor.

$$[\![e/x_1 \to e_1, \ldots, x_n \to e_n]\!]\, b = [\![e]\!]\,[\,b \mid x_1 : [\![e_1]\!]\, b \mid \cdots \mid x_n : [\![e_n]\!]\, b\,]$$

($[\![e_1]\!]\, b$: c_1, $[\![e_n]\!]\, b$: c_n)

substitution.

$$\left[\begin{array}{l} x_i \to e_i \\ y \neq x_1, \ldots, x_n \to y \end{array}\right]$$

$[b \mid x : v] \in [\langle \text{var} \rangle \to \mathbb{Z}]$

$[b \mid x : v](x) = v$.

$[b \mid x : v](y) = b(y)$.

$x_1 \to c_1$

$\vdots$

$x_n \to c_n$.

$x \neq x_1, \ldots, x_n \to b(x)$

* where does this struct. ind. come from?

A: initialite of $\langle \text{intexp} \rangle$.

↘ P... property.

an algebra where the underlying set $\{ e \in \langle \text{intexp} \rangle \mid e \text{ satisfies } P \}$

$c_i = \langle \text{intexp} \rangle$