The following slides have been adapted from the material of the course

Principles for Software Composition

http://didawiki.di.unipi.it/doku.php/magistraleinformatica/psc/start

by

Roberto Bruni (University of Pisa, Italy)

http://www.di.unipi.it/~bruni/

Used with permission of the author

Lambda notation

Today

- Up to here: operational semantics of IMP expressions. An **interpreter** for running a command from an initial state.

- The denotational semantics is different, as it acts as a **compiler**: given a command, it returns a **lambda-expression**, that is, an abstract functional program.

- Lambda-notation is beneficial for two reasons:
  - For giving a **denotational semantics**
  - For defining higher-order **functional languages**

- Operational and denotational semantics should be consistent (correct and complete); we cover the basic form of this result.

Lambda notation

Key ingredients

anonymous functions

application

Lambda notation

Key ingredients

anonymous functions

$\lambda x.\ e$

application

Lambda notation

Key ingredients

anonymous functions

$\lambda x.\ e$ x serves as a formal parameter in e

application

Lambda notation

Key ingredients

anonymous functions

$\lambda x.\ e$ x serves as a formal parameter in e

denotes a function that waits for one value to be substituted for x and then evaluates e

application

Lambda notation

Key ingredients

anonymous functions

$\lambda x.\ e$ x serves as a formal parameter in e

denotes a function that waits for one value to be substituted for x and then evaluates e

application

$e_1\ e_2$

Lambda notation

Key ingredients

anonymous functions

$\lambda x.\ e$ x serves as a formal parameter in e

denotes a function that waits for one value to be substituted for x and then evaluates e

application

$e_1\ e_2$ e_2 is the argument passed to the function e_1

Lambda notation

Key ingredients

anonymous functions

$\lambda x.\ e$ x serves as a formal parameter in e

denotes a function that waits for one value to be substituted for x and then evaluates e

application

$e_1\ e_2$ e_2 is the argument passed to the function e_1

denotes the application of the function e_1 to e_2

Lambda notation

Key ingredients

anonymous functions

$\lambda x.\ e$ x serves as a formal parameter in e

denotes a function that waits for one value to be substituted for x and then evaluates e

application

$e_1\ e_2$ e_2 is the argument passed to the function e_1

denotes the application of the function e_1 to e_2

reduces the need of parentheses $e_1(e_2)$

Function definition

$$f(x) \triangleq x^2 - 2 \cdot x + 5$$

Function definition

$$f(x) \triangleq x^2 - 2 \cdot x + 5$$

$$f \triangleq \lambda x.\ (x^2 - 2 \cdot x + 5)$$

Function definition

$$f(x) \triangleq x^2 - 2 \cdot x + 5$$

$$f \triangleq \lambda x.\ (x^2 - 2 \cdot x + 5)$$

unnecessary parentheses added for clarity

Associative rules

$e_1\ e_2\ e_3$ is read $(e_1\ e_2)\ e_3$ application is left-associative

$\lambda x.\ \lambda y.\ \lambda z.\ e$ is read $\lambda x.\ (\lambda y.\ (\lambda z.\ e))$ abstraction is right-associative

Scoping

$\lambda x.\ e$

the scope of x is e

x not visible outside e

like a local variable

Alpha-conversion

$\lambda x.\ (x^2 - 2 \cdot x + 5)$

$\lambda y.\ (y^2 - 2 \cdot y + 5)$

names of formal parameters are inessential:
the two expressions denote the same function

Alpha-conversion

$\lambda x.\ (x^2 - 2 \cdot x + 5)$

$\lambda y.\ (y^2 - 2 \cdot y + 5)$

names of formal parameters are inessential: the two expressions denote the same function

$$\lambda x.\ e \equiv \lambda y.\ (e[{}^{y}/_{x}])$$

Alpha-conversion

$\lambda x.\ (x^2 - 2 \cdot x + 5)$

$\lambda y.\ (y^2 - 2 \cdot y + 5)$

names of formal parameters are inessential: the two expressions denote the same function

$$\lambda x.\ e \equiv \lambda y.\ (e[{}^{y}/_{x}])$$

capture-avoiding substitution

(to be formalised later)

Alpha-conversion

$\lambda x.\ (x^2 - 2 \cdot x + 5)$

$\lambda y.\ (y^2 - 2 \cdot y + 5)$

names of formal parameters are inessential: the two expressions denote the same function

$\lambda x.\ e \equiv \lambda y.\ (e[{}^{y}/_{x}])$ (under suitable conditions on e, y)

capture-avoiding substitution

(to be formalised later)

Application (beta rule)

$(\lambda x.\ e)\ e_0$ application of a function

Application (beta rule)

$(\lambda x.\ e)\ e_0$ application of a function

$\equiv$

$e[{}^{e_0}/_{x}]$ evaluation via substitution

Application (beta rule)

$$(\lambda x.\ e)\ e_0$$

$$\equiv$$

$$e[{}^{e_0}/_{x}]$$

application of a function

evaluation via substitution

capture-avoiding substitution

Example

$\lambda x.\ (x^2 - 2 \cdot x + 5)$ a function

Example

$\lambda x.\ (x^2 - 2 \cdot x + 5)$ a function

$(\lambda x.\ (x^2 - 2 \cdot x + 5))\ 2$ its application

Example

$\lambda x.\ (x^2 - 2 \cdot x + 5)$ a function

$(\lambda x.\ (x^2 - 2 \cdot x + 5))\ 2$ its application

$\equiv$

$2^2 - 2 \cdot 2 + 5 = 5$ its evaluation

Example

$\lambda x.\ \lambda y.\ (x^2 - 2 \cdot y + 5)$ a function

Example

$\lambda x.\ \lambda y.\ (x^2 - 2 \cdot y + 5)$ a function

$(\lambda x.\ \lambda y.\ (x^2 - 2 \cdot y + 5))\ 2$ its application

Example

$\lambda x.\ \lambda y.\ (x^2 - 2 \cdot y + 5)$ a function

$(\lambda x.\ \lambda y.\ (x^2 - 2 \cdot y + 5))\ 2$ its application

$\equiv$

$\lambda y.\ (2^2 - 2 \cdot y + 5)$ its evaluation

Example

$\lambda x.\ \lambda y.\ (x^2 - 2 \cdot y + 5)$ a function

$(\lambda x.\ \lambda y.\ (x^2 - 2 \cdot y + 5))\ 2$ its application

$\equiv$

$\lambda y.\ (2^2 - 2 \cdot y + 5)$ its evaluation

it is still a function!

Example

$\lambda f.\ \lambda x.\ (x^2 + f\ 1)$ a function

Example

$\lambda f.\ \lambda x.\ (x^2 + f\ 1)$ a function

$(\lambda f.\ \lambda x.\ (x^2 + f\ 1))\ (\lambda y.\ (2 \cdot y))$ its application

(the argument is a function!)

Example

$\lambda f.\ \lambda x.\ (x^2 + f\ 1)$ a function

$(\lambda f.\ \lambda x.\ (x^2 + f\ 1))\ (\lambda y.\ (2 \cdot y))$ its application

$\equiv$ (the argument is a function!)

$\lambda x.\ (x^2 + (\lambda y.\ (2 \cdot y))\ 1)$ its evaluation

Example

$\lambda f.\ \lambda x.\ (x^2 + f\ 1)$ a function

$(\lambda f.\ \lambda x.\ (x^2 + f\ 1))\ (\lambda y.\ (2 \cdot y))$ its application

$\equiv$ (the argument is a function!)

$\lambda x.\ (x^2 + (\lambda y.\ (2 \cdot y))\ 1)$ its evaluation

higher-order: functions as arguments/results

Example

$\lambda f.\ \lambda x.\ (x^2 + f\ 1)$ a function

Example

$\lambda f.\ \lambda x.\ (x^2 + f\ 1)$ a function

$(\lambda f.\ \lambda x.\ (x^2 + f\ 1))\ (\lambda y.\ (2 \cdot y))\ \ 3$ its application

Example

$$\lambda f.\ \lambda x.\ (x^2 + f\ 1)$$ a function

$$(\lambda f.\ \lambda x.\ (x^2 + f\ 1))\ (\lambda y.\ (2 \cdot y))\ \ 3$$ its application

$$\equiv$$

$$\lambda x.\ (x^2 + (\lambda y.\ (2 \cdot y))\ 1)\qquad 3$$ its evaluation

Example

$$\lambda f.\ \lambda x.\ (x^2 + f\ 1)$$ a function

$$(\lambda f.\ \lambda x.\ (x^2 + f\ 1))\ (\lambda y.\ (2 \cdot y))\ \ 3$$ its application

$$\equiv$$

$$\lambda x.\ (x^2 + (\lambda y.\ (2 \cdot y))\ 1)\qquad 3$$ its evaluation
its application

$$\equiv$$

$$3^2 + (\lambda y.\ (2 \cdot y))\ 1$$ its evaluation

Example

$$\lambda f.\ \lambda x.\ (x^2 + f\ 1)$$ a function

$$(\lambda f.\ \lambda x.\ (x^2 + f\ 1))\ (\lambda y.\ (2 \cdot y))\ \ 3$$ its application

$$\equiv$$

$$\lambda x.\ (x^2 + (\lambda y.\ (2 \cdot y))\ 1)\qquad 3$$ its evaluation
its application

$$\equiv$$

$$3^2 + (\lambda y.\ (2 \cdot y))\ 1$$ its evaluation
its application

$$\equiv$$

$$3^2 + 2 \cdot 1 = 11$$ its evaluation

Conditional

$e \to e_1, e_2$ if e then e_1 else e_2

example $\min \triangleq \lambda x.\ \lambda y.\ x < y \to x, y$

Capture-avoiding substitutions

Free variables

$$\mathrm{fv}(n) \stackrel{\text{def}}{=} \varnothing$$

$$\mathrm{fv}(x) \stackrel{\text{def}}{=} \{x\}$$

$$\mathrm{fv}(t_0 \text{ op } t_1) \stackrel{\text{def}}{=} \mathrm{fv}(t_0) \cup \mathrm{fv}(t_1)$$

$$\mathrm{fv}(\mathbf{if}\ t\ \mathbf{then}\ t_0\ \mathbf{else}\ t_1) \stackrel{\text{def}}{=} \mathrm{fv}(t) \cup \mathrm{fv}(t_0) \cup \mathrm{fv}(t_1)$$

$$\mathrm{fv}((t_0, t_1)) \stackrel{\text{def}}{=} \mathrm{fv}(t_0) \cup \mathrm{fv}(t_1)$$

$$\mathrm{fv}(\mathbf{fst}(t)) \stackrel{\text{def}}{=} \mathrm{fv}(t)$$

$$\mathrm{fv}(\mathbf{snd}(t)) \stackrel{\text{def}}{=} \mathrm{fv}(t)$$

$$\mathrm{fv}(\lambda x.\ t) \stackrel{\text{def}}{=} \mathrm{fv}(t) \backslash \{x\}$$

$$\mathrm{fv}((t_0\ t_1)) \stackrel{\text{def}}{=} \mathrm{fv}(t_0) \cup \mathrm{fv}(t_1)$$

$$\mathrm{fv}(\mathbf{rec}\ x.\ t) \stackrel{\text{def}}{=} \mathrm{fv}(t) \backslash \{x\}$$

Substitutions

$$n[^t/_x] = n$$

$$y[^t/_x] \stackrel{\text{def}}{=} \begin{cases} t & \text{if } y = x \\ y & \text{if } y \neq x \end{cases}$$

$$(t_0 \text{ op } t_1)[^t/_x] \stackrel{\text{def}}{=} t_0[^t/_x] \text{ op } t_1[^t/_x] \quad \text{with op} \in \{+, -, \times\}$$

$$(\textbf{if } t' \textbf{ then } t_0 \textbf{ else } t_1)[^t/_x] \stackrel{\text{def}}{=} \textbf{if } t'[^t/_x] \textbf{ then } t_0[^t/_x] \textbf{ else } t_1[^t/_x]$$

$$(t_0, t_1)[^t/_x] \stackrel{\text{def}}{=} (t_0[^t/_x], t_1[^t/_x])$$

$$\textbf{fst}(t')[^t/_x] \stackrel{\text{def}}{=} \textbf{fst}(t'[^t/_x])$$

$$\textbf{snd}(t')[^t/_x] \stackrel{\text{def}}{=} \textbf{snd}(t'[^t/_x])$$

$$(t_0\ t_1)[^t/_x] \stackrel{\text{def}}{=} (t_0[^t/_x]\ t_1[^t/_x])$$

$$(\lambda y.\ t')[^t/_x] \stackrel{\text{def}}{=} \lambda z.\ (\ t'[^z/_y][^t/_x]\ ) \quad \text{for } z \notin \text{fv}(\lambda y.\ t') \cup \text{fv}(t) \cup \{x\}$$

$$(\textbf{rec } y.\ t')[^t/_x] \stackrel{\text{def}}{=} \textbf{rec } z.\ (t'[^z/_y][^t/_x]) \quad \text{for } z \notin \text{fv}(\textbf{rec } y.\ t') \cup \text{fv}(t) \cup \{x\}$$

Denotational semantics

Arithmetic expressions

$$\mathsf{Aexp}, \prec \qquad\qquad a_i \prec a_0 \,\mathsf{op}\, a_1$$

$$\mathcal{A}[\![\cdot]\!] : \mathsf{Aexp} \to \mathbb{M} \to \mathbb{Z}$$

$$\begin{array}{rcl} \mathcal{A}[\![n]\!]\sigma & \triangleq & n \\ \mathcal{A}[\![x]\!]\sigma & \triangleq & \sigma(x) \\ \mathcal{A}[\![a_0 \,\mathsf{op}\, a_1]\!]\sigma & \triangleq & \mathcal{A}[\![a_0]\!]\sigma \; op \; \mathcal{A}[\![a_1]\!]\sigma \end{array}$$

Boolean expressions

$$\mathsf{Bexp}, \prec \qquad b_i \prec b_0 \,\mathsf{bop}\, b_1 \qquad b \prec \neg b$$

$$\mathcal{B}[\![\cdot]\!] : \mathsf{Bexp} \to \mathbb{M} \to \mathbb{Z}$$

$$\begin{aligned}
\mathcal{B}[\![v]\!]\sigma &\triangleq v \\
\mathcal{B}[\![a_0 \,\mathsf{cmp}\, a_1]\!]\sigma &\triangleq \mathcal{A}[\![a_0]\!]\sigma \; cmp \; \mathcal{A}[\![a_1]\!]\sigma \\
\mathcal{B}[\![\neg b]\!]\sigma &\triangleq \neg \mathcal{B}[\![b]\!]\sigma \\
\mathcal{B}[\![b_0 \,\mathsf{bop}\, b_1]\!]\sigma &\triangleq \mathcal{B}[\![b_0]\!]\sigma \; bop \; \mathcal{B}[\![b_1]\!]\sigma
\end{aligned}$$

Consistency of expressions

Consistency?

$\langle a, \sigma \rangle \longrightarrow n$

$\mathcal{A}[\![a]\!]\sigma = n$

Consistency?

$$\langle a, \sigma \rangle \longrightarrow n \quad \stackrel{?}{\Leftrightarrow} \quad \mathcal{A}[\![a]\!]\sigma = n$$

Consistency?

$$\forall a, \sigma, n$$

$$\langle a, \sigma \rangle \longrightarrow n \quad \stackrel{?}{\Leftrightarrow} \quad \mathcal{A}[\![a]\!]\sigma = n$$

Consistency?

$$\forall a, \sigma, n$$

$$\langle a, \sigma \rangle \longrightarrow n \quad \stackrel{?}{\Leftrightarrow} \quad \mathcal{A}[\![a]\!]\sigma = n$$

$$P(a) \stackrel{\triangle}{=} \forall \sigma.\ \langle a, \sigma \rangle \longrightarrow \mathcal{A}[\![a]\!]\sigma$$

Consistency?

$$\forall a, \sigma, n$$

$$\langle a, \sigma \rangle \longrightarrow n \quad \stackrel{?}{\Leftrightarrow} \quad \mathcal{A}[\![a]\!]\sigma = n$$

$$P(a) \stackrel{\triangle}{=} \forall \sigma.\ \langle a, \sigma \rangle \longrightarrow \mathcal{A}[\![a]\!]\sigma \qquad \forall a \in \mathsf{Aexp}.\ P(a)\,?$$

Consistency?

$$\forall a, \sigma, n$$

$$\langle a, \sigma \rangle \longrightarrow n \quad \stackrel{?}{\Leftrightarrow} \quad \mathcal{A}[\![a]\!]\sigma = n$$

$$P(a) \stackrel{\triangle}{=} \forall \sigma.\ \langle a, \sigma \rangle \longrightarrow \mathcal{A}[\![a]\!]\sigma$$

$$\forall a \in \mathsf{Aexp}.\ P(a)\,?$$

structural induction!

Consistency?

$$\forall a, \sigma, n$$

$$\langle a, \sigma \rangle \longrightarrow n \quad \stackrel{?}{\Leftrightarrow} \quad \mathcal{A}[\![a]\!]\sigma = n$$

$$P(a) \stackrel{\triangle}{=} \forall \sigma.\ \langle a, \sigma \rangle \longrightarrow \mathcal{A}[\![a]\!]\sigma \qquad \forall a \in \mathsf{Aexp}.\ P(a)\,?$$

structural induction!

$$\forall x \in \mathsf{Ide}.\ P(x) \qquad\qquad \forall n \in \mathbb{Z}.\ P(n)$$

$$\frac{\forall a_0, a_1.\ P(a_0) \wedge P(a_1) \Rightarrow P(a_0\ \mathsf{op}\ a_1)}{\forall a.\ P(a)}$$

Base cases

$\forall x \in \mathsf{Ide}.\ P(x)$

Base cases

$\forall x \in \mathsf{Ide}.\ P(x)$

take $x \in \mathsf{Ide}$

Base cases

$\forall x \in \mathsf{Ide}.\ P(x)$

take $x \in \mathsf{Ide}$

we need to prove $P(x) \triangleq \forall \sigma.\ \langle x, \sigma \rangle \longrightarrow \mathcal{A}[\![x]\!]\sigma$

Base cases

$\forall x \in \mathsf{Ide}.\ P(x)$

take $x \in \mathsf{Ide}$

we need to prove $P(x) \triangleq \forall \sigma.\ \langle x, \sigma \rangle \longrightarrow \mathcal{A}[\![x]\!]\sigma = \sigma(x)$

Base cases

$\forall x \in \mathsf{Ide}.\ P(x)$

take $x \in \mathsf{Ide}$

we need to prove $P(x) \triangleq \forall \sigma.\ \langle x, \sigma \rangle \longrightarrow \mathcal{A}[\![x]\!]\sigma = \sigma(x)$

taken a generic σ we conclude by rule

Base cases

$\forall x \in \mathsf{Ide}.\ P(x)$ take $x \in \mathsf{Ide}$

we need to prove $P(x) \triangleq \forall \sigma.\ \langle x, \sigma \rangle \longrightarrow \mathcal{A}[\![x]\!]\sigma = \sigma(x)$

taken a generic σ we conclude by rule $\dfrac{}{\langle x, \sigma \rangle \longrightarrow \sigma(x)}$

Base cases

$\forall x \in \mathsf{Ide}.\ P(x)$ take $x \in \mathsf{Ide}$

we need to prove $P(x) \triangleq \forall \sigma.\ \langle x, \sigma \rangle \longrightarrow \mathcal{A}[\![x]\!]\sigma = \sigma(x)$

taken a generic σ we conclude by rule $\dfrac{}{\langle x, \sigma \rangle \longrightarrow \sigma(x)}$

$\forall n \in \mathbb{Z}.\ P(n)$

Base cases

$\forall x \in \mathsf{Ide}.\ P(x)$ take $x \in \mathsf{Ide}$

we need to prove $P(x) \triangleq \forall \sigma.\ \langle x, \sigma \rangle \longrightarrow \mathcal{A}[\![x]\!]\sigma = \sigma(x)$

taken a generic σ we conclude by rule $\dfrac{}{\langle x, \sigma \rangle \longrightarrow \sigma(x)}$

$\forall n \in \mathbb{Z}.\ P(n)$ take $n \in \mathbb{Z}$

Base cases

$\forall x \in \mathsf{Ide}.\ P(x)$ take $x \in \mathsf{Ide}$

we need to prove $P(x) \triangleq \forall \sigma.\ \langle x, \sigma \rangle \longrightarrow \mathcal{A}[\![x]\!]\sigma = \sigma(x)$

taken a generic σ we conclude by rule $\dfrac{}{\langle x, \sigma \rangle \longrightarrow \sigma(x)}$

$\forall n \in \mathbb{Z}.\ P(n)$ take $n \in \mathbb{Z}$

we need to prove $P(n) \triangleq \forall \sigma.\ \langle n, \sigma \rangle \longrightarrow \mathcal{A}[\![n]\!]\sigma$

Base cases

$\forall x \in \mathsf{Ide}.\ P(x)$ take $x \in \mathsf{Ide}$

we need to prove $P(x) \triangleq \forall \sigma.\ \langle x, \sigma \rangle \longrightarrow \mathcal{A}[\![x]\!]\sigma = \sigma(x)$

taken a generic σ we conclude by rule $\dfrac{}{\langle x, \sigma \rangle \longrightarrow \sigma(x)}$

$\forall n \in \mathbb{Z}.\ P(n)$ take $n \in \mathbb{Z}$

we need to prove $P(n) \triangleq \forall \sigma.\ \langle n, \sigma \rangle \longrightarrow \mathcal{A}[\![n]\!]\sigma = n$

Base cases

$\forall x \in \mathsf{Ide}.\ P(x)$ take $x \in \mathsf{Ide}$

we need to prove $P(x) \stackrel{\triangle}{=} \forall \sigma.\ \langle x, \sigma \rangle \longrightarrow \mathcal{A}[\![x]\!]\sigma = \sigma(x)$

taken a generic σ we conclude by rule $\dfrac{}{\langle x, \sigma \rangle \longrightarrow \sigma(x)}$

$\forall n \in \mathbb{Z}.\ P(n)$ take $n \in \mathbb{Z}$

we need to prove $P(n) \stackrel{\triangle}{=} \forall \sigma.\ \langle n, \sigma \rangle \longrightarrow \mathcal{A}[\![n]\!]\sigma = n$

taken a generic σ we conclude by rule

Base cases

$\forall x \in \mathsf{Ide}.\ P(x)$ take $x \in \mathsf{Ide}$

we need to prove $P(x) \triangleq \forall \sigma.\ \langle x, \sigma \rangle \longrightarrow \mathcal{A}[\![x]\!]\sigma = \sigma(x)$

taken a generic σ we conclude by rule $\dfrac{}{\langle x, \sigma \rangle \longrightarrow \sigma(x)}$

$\forall n \in \mathbb{Z}.\ P(n)$ take $n \in \mathbb{Z}$

we need to prove $P(n) \triangleq \forall \sigma.\ \langle n, \sigma \rangle \longrightarrow \mathcal{A}[\![n]\!]\sigma = n$

taken a generic σ we conclude by rule $\dfrac{}{\langle n, \sigma \rangle \longrightarrow n}$

Inductive case

$$\forall a_0, a_1.\ P(a_0) \wedge P(a_1) \Rightarrow P(a_0 \text{ op } a_1)$$

Inductive case

$\forall a_0, a_1.\ P(a_0) \wedge P(a_1) \Rightarrow P(a_0 \text{ op } a_1)$

Take generic a_0, a_1

Inductive case

$\forall a_0, a_1.\ P(a_0) \wedge P(a_1) \Rightarrow P(a_0 \text{ op } a_1)$ Take generic a_0, a_1

we assume $P(a_i) \triangleq \forall \sigma.\ \langle a_i, \sigma \rangle \longrightarrow \mathcal{A}[\![a_i]\!]\sigma$

Inductive case

$\forall a_0, a_1.\ P(a_0) \wedge P(a_1) \Rightarrow P(a_0 \text{ op } a_1)$ Take generic a_0, a_1

we assume $P(a_i) \triangleq \forall \sigma.\ \langle a_i, \sigma \rangle \longrightarrow \mathcal{A}[\![a_i]\!]\sigma$

we need to prove $P(a_0 \,\text{op}\, a_1) \triangleq \forall \sigma.\ \langle a_0 \,\text{op}\, a_1, \sigma \rangle \longrightarrow \mathcal{A}[\![a_0 \,\text{op}\, a_1]\!]\sigma$

Inductive case

$\forall a_0, a_1.\ P(a_0) \wedge P(a_1) \Rightarrow P(a_0 \text{ op } a_1)$ Take generic a_0, a_1

we assume $P(a_i) \triangleq \forall \sigma.\ \langle a_i, \sigma \rangle \longrightarrow \mathcal{A}[\![a_i]\!]\sigma$

we need to prove $P(a_0 \,\text{op}\, a_1) \triangleq \forall \sigma.\ \langle a_0 \,\text{op}\, a_1, \sigma \rangle \longrightarrow \mathcal{A}[\![a_0 \,\text{op}\, a_1]\!]\sigma$

$$= \mathcal{A}[\![a_0]\!]\sigma\ op\ \mathcal{A}[\![a_1]\!]\sigma$$

Inductive case

$\forall a_0, a_1.\ P(a_0) \wedge P(a_1) \Rightarrow P(a_0 \text{ op } a_1)$ Take generic a_0, a_1

we assume $P(a_i) \triangleq \forall \sigma.\ \langle a_i, \sigma \rangle \longrightarrow \mathcal{A}[\![a_i]\!]\sigma$

we need to prove $P(a_0 \,\text{op}\, a_1) \triangleq \forall \sigma.\ \langle a_0 \,\text{op}\, a_1, \sigma \rangle \longrightarrow \mathcal{A}[\![a_0 \,\text{op}\, a_1]\!]\sigma$

$= \mathcal{A}[\![a_0]\!]\sigma\ op\ \mathcal{A}[\![a_1]\!]\sigma$

take a generic σ

Inductive case

$\forall a_0, a_1.\ P(a_0) \wedge P(a_1) \Rightarrow P(a_0 \text{ op } a_1)$ Take generic a_0, a_1

we assume $P(a_i) \triangleq \forall \sigma.\ \langle a_i, \sigma \rangle \longrightarrow \mathcal{A}[\![a_i]\!]\sigma$

we need to prove $P(a_0 \,\text{op}\, a_1) \triangleq \forall \sigma.\ \langle a_0 \,\text{op}\, a_1, \sigma \rangle \longrightarrow \mathcal{A}[\![a_0 \,\text{op}\, a_1]\!]\sigma$

$= \mathcal{A}[\![a_0]\!]\sigma\ op\ \mathcal{A}[\![a_1]\!]\sigma$

take a generic σ

$\langle a_0 \,\text{op}\, a_1, \sigma \rangle \longrightarrow n$

Inductive case

$\forall a_0, a_1.\ P(a_0) \wedge P(a_1) \Rightarrow P(a_0 \text{ op } a_1)$ Take generic a_0, a_1

we assume $P(a_i) \triangleq \forall \sigma.\ \langle a_i, \sigma \rangle \longrightarrow \mathcal{A}[\![a_i]\!]\sigma$

we need to prove $P(a_0 \,\text{op}\, a_1) \triangleq \forall \sigma.\ \langle a_0 \,\text{op}\, a_1, \sigma \rangle \longrightarrow \mathcal{A}[\![a_0 \,\text{op}\, a_1]\!]\sigma$

$= \mathcal{A}[\![a_0]\!]\sigma\ op\ \mathcal{A}[\![a_1]\!]\sigma$

take a generic σ

$\langle a_0 \,\text{op}\, a_1, \sigma \rangle \longrightarrow n$

$\nwarrow_{n=n_0\ op\ n_1}\ \langle a_0, \sigma \rangle \longrightarrow n_0, \langle a_1, \sigma \rangle \longrightarrow n_1$

Inductive case

$\forall a_0, a_1.\ P(a_0) \wedge P(a_1) \Rightarrow P(a_0 \text{ op } a_1)$ Take generic a_0, a_1

we assume $P(a_i) \triangleq \forall \sigma.\ \langle a_i, \sigma \rangle \longrightarrow \mathcal{A}[\![a_i]\!]\sigma$

we need to prove $P(a_0 \,\text{op}\, a_1) \triangleq \forall \sigma.\ \langle a_0 \,\text{op}\, a_1, \sigma \rangle \longrightarrow \mathcal{A}[\![a_0 \,\text{op}\, a_1]\!]\sigma$

$= \mathcal{A}[\![a_0]\!]\sigma\ op\ \mathcal{A}[\![a_1]\!]\sigma$

take a generic σ

$\langle a_0 \,\text{op}\, a_1, \sigma \rangle \longrightarrow n$

$\nwarrow_{n=n_0\ op\ n_1}\ \langle a_0, \sigma \rangle \longrightarrow n_0, \langle a_1, \sigma \rangle \longrightarrow n_1$

by inductive hypotheses, $n_i = \mathcal{A}[\![a_i]\!]\sigma$

Inductive case

$\forall a_0, a_1.\ P(a_0) \wedge P(a_1) \Rightarrow P(a_0 \text{ op } a_1)$ Take generic a_0, a_1

we assume $P(a_i) \triangleq \forall \sigma.\ \langle a_i, \sigma \rangle \longrightarrow \mathcal{A}[\![a_i]\!]\sigma$

we need to prove $P(a_0 \,\text{op}\, a_1) \triangleq \forall \sigma.\ \langle a_0 \,\text{op}\, a_1, \sigma \rangle \longrightarrow \mathcal{A}[\![a_0 \,\text{op}\, a_1]\!]\sigma$

$= \mathcal{A}[\![a_0]\!]\sigma\ op\ \mathcal{A}[\![a_1]\!]\sigma$

take a generic σ

$\langle a_0 \,\text{op}\, a_1, \sigma \rangle \longrightarrow n$

$\nwarrow_{n=n_0\ op\ n_1}\ \langle a_0, \sigma \rangle \longrightarrow n_0, \langle a_1, \sigma \rangle \longrightarrow n_1$

by inductive hypotheses, $n_i = \mathcal{A}[\![a_i]\!]\sigma$

and thus $n = n_0\ op\ n_1 = \mathcal{A}[\![a_0]\!]\sigma\ op\ \mathcal{A}[\![a_1]\!]\sigma$

Denotational semantics of commands?

Recursive definitions

$$\mathcal{C}[\![\cdot]\!] : \mathsf{Com} \to \mathbb{M} \to \mathbb{M}$$

Recursive definitions

for divergence

$$\mathcal{C}[\![\cdot]\!] : \mathsf{Com} \to \mathbb{M} \to \mathbb{M} \cup \{\bot\}$$

Recursive definitions

for divergence

$$\mathcal{C}[\![\cdot]\!] : \mathsf{Com} \to \mathbb{M} \to \mathbb{M} \cup \{\bot\}$$

$$\begin{aligned}
\mathcal{C}[\![\mathbf{skip}]\!]\sigma &\stackrel{\triangle}{=} \\
\mathcal{C}[\![x := a]\!]\sigma &\stackrel{\triangle}{=} \\
\mathcal{C}[\![c_0; c_1]\!]\sigma &\stackrel{\triangle}{=} \\
\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]\sigma &\stackrel{\triangle}{=} \\
\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma &\stackrel{\triangle}{=}
\end{aligned}$$

Recursive definitions

for divergence

$$\mathcal{C}[\![\cdot]\!] : \mathsf{Com} \to \mathbb{M} \to \mathbb{M} \cup \{\bot\}$$

$$\begin{aligned}
\mathcal{C}[\![\mathbf{skip}]\!]\sigma &\stackrel{\triangle}{=} \sigma \\
\mathcal{C}[\![x := a]\!]\sigma &\stackrel{\triangle}{=} \\
\mathcal{C}[\![c_0; c_1]\!]\sigma &\stackrel{\triangle}{=} \\
\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]\sigma &\stackrel{\triangle}{=} \\
\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma &\stackrel{\triangle}{=}
\end{aligned}$$

Recursive definitions

for divergence

$$\mathcal{C}[\![\cdot]\!] : \mathsf{Com} \to \mathbb{M} \to \mathbb{M} \cup \{\bot\}$$

$$\begin{aligned}
\mathcal{C}[\![\mathbf{skip}]\!]\sigma &\stackrel{\triangle}{=} \sigma \\
\mathcal{C}[\![x := a]\!]\sigma &\stackrel{\triangle}{=} \sigma[\mathcal{A}[\![a]\!]\sigma / x] \\
\mathcal{C}[\![c_0; c_1]\!]\sigma &\stackrel{\triangle}{=}
\end{aligned}$$

$$\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]\sigma \stackrel{\triangle}{=}$$

$$\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma \stackrel{\triangle}{=}$$

Recursive definitions

for divergence

$$\mathcal{C}[\![\cdot]\!] : \mathsf{Com} \to \mathbb{M} \to \mathbb{M} \cup \{\bot\}$$

$$\begin{array}{rcl}
\mathcal{C}[\![\mathbf{skip}]\!]\sigma & \stackrel{\triangle}{=} & \sigma \\
\mathcal{C}[\![x := a]\!]\sigma & \stackrel{\triangle}{=} & \sigma[\mathcal{A}[\![a]\!]\sigma / x] \\
\mathcal{C}[\![c_0; c_1]\!]\sigma & \stackrel{\triangle}{=} & \mathcal{C}[\![c_1]\!](\mathcal{C}[\![c_0]\!]\sigma)
\end{array}$$

$$\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]\sigma \stackrel{\triangle}{=}$$

$$\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma \stackrel{\triangle}{=}$$

Recursive definitions

for divergence

$$\mathcal{C}[\![\cdot]\!] : \mathsf{Com} \to \mathbb{M} \to \mathbb{M} \cup \{\bot\}$$

$$\begin{aligned}
\mathcal{C}[\![\mathbf{skip}]\!]\sigma &\triangleq \sigma \\
\mathcal{C}[\![x := a]\!]\sigma &\triangleq \sigma[\mathcal{A}[\![a]\!]\sigma / x] \\
\mathcal{C}[\![c_0; c_1]\!]\sigma &\triangleq \mathcal{C}[\![c_1]\!](\mathcal{C}[\![c_0]\!]\sigma)
\end{aligned}$$

almost…

$$\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]\sigma \triangleq$$

$$\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma \triangleq$$

Recursive definitions

for divergence

$$\mathcal{C}[\![\cdot]\!] : \mathsf{Com} \to \mathbb{M} \to \mathbb{M} \cup \{\bot\}$$

$$\begin{array}{rcl}
\mathcal{C}[\![\mathbf{skip}]\!]\sigma & \triangleq & \sigma \\
\mathcal{C}[\![x := a]\!]\sigma & \triangleq & \sigma[\mathcal{A}[\![a]\!]\sigma / x] \\
\mathcal{C}[\![c_0; c_1]\!]\sigma & \triangleq & \mathcal{C}[\![c_1]\!](\mathcal{C}[\![c_0]\!]\sigma)
\end{array}$$

almost…

$$\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]\sigma \triangleq \begin{cases} \mathcal{C}[\![c_0]\!]\sigma & \text{if } \mathcal{B}[\![b]\!]\sigma \\ \mathcal{C}[\![c_1]\!]\sigma & \text{otherwise} \end{cases}$$

$$\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma \triangleq$$

Recursive definitions

for divergence

$$\mathcal{C}[\![\cdot]\!] : \mathsf{Com} \to \mathbb{M} \to \mathbb{M} \cup \{\bot\}$$

$$\begin{aligned}
\mathcal{C}[\![\mathbf{skip}]\!]\sigma &\triangleq \sigma \\
\mathcal{C}[\![x := a]\!]\sigma &\triangleq \sigma[\mathcal{A}[\![a]\!]\sigma / x] \\
\mathcal{C}[\![c_0; c_1]\!]\sigma &\triangleq \mathcal{C}[\![c_1]\!](\mathcal{C}[\![c_0]\!]\sigma)
\end{aligned}$$

almost…

$$\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]\sigma \triangleq \begin{cases} \mathcal{C}[\![c_0]\!]\sigma & \text{if } \mathcal{B}[\![b]\!]\sigma \\ \mathcal{C}[\![c_1]\!]\sigma & \text{otherwise} \end{cases}$$

$$\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma \triangleq \begin{cases} \sigma & \text{if } \neg\mathcal{B}[\![b]\!]\sigma \\ \mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!](\mathcal{C}[\![c]\!]\sigma) & \text{otherwise} \end{cases}$$

Recursive definitions

for divergence

$$\mathcal{C}[\![\cdot]\!] : \mathsf{Com} \to \mathbb{M} \to \mathbb{M} \cup \{\bot\}$$

$$\begin{aligned}
\mathcal{C}[\![\mathbf{skip}]\!]\sigma &\triangleq \sigma \\
\mathcal{C}[\![x := a]\!]\sigma &\triangleq \sigma[\mathcal{A}[\![a]\!]\sigma / x] \\
\mathcal{C}[\![c_0; c_1]\!]\sigma &\triangleq \mathcal{C}[\![c_1]\!](\mathcal{C}[\![c_0]\!]\sigma)
\end{aligned}$$

almost…

$$\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]\sigma \triangleq \begin{cases} \mathcal{C}[\![c_0]\!]\sigma & \text{if } \mathcal{B}[\![b]\!]\sigma \\ \mathcal{C}[\![c_1]\!]\sigma & \text{otherwise} \end{cases}$$

$$\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma \triangleq \begin{cases} \sigma & \text{if } \neg\mathcal{B}[\![b]\!]\sigma \\ \mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!](\mathcal{C}[\![c]\!]\sigma) & \text{otherwise} \end{cases}$$

almost…

Recursive definitions

for divergence

$$\mathcal{C}[\![\cdot]\!] : \mathsf{Com} \to \mathbb{M} \to \mathbb{M} \cup \{\bot\}$$

$$\begin{array}{rcl}
\mathcal{C}[\![\mathbf{skip}]\!]\sigma & \stackrel{\triangle}{=} & \sigma \\
\mathcal{C}[\![x := a]\!]\sigma & \stackrel{\triangle}{=} & \sigma[\mathcal{A}[\![a]\!]\sigma / x] \\
\mathcal{C}[\![c_0; c_1]\!]\sigma & \stackrel{\triangle}{=} & \mathcal{C}[\![c_1]\!](\mathcal{C}[\![c_0]\!]\sigma)
\end{array}$$

almost…

$$\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]\sigma \stackrel{\triangle}{=} \begin{cases} \mathcal{C}[\![c_0]\!]\sigma & \text{if } \mathcal{B}[\![b]\!]\sigma \\ \mathcal{C}[\![c_1]\!]\sigma & \text{otherwise} \end{cases}$$

$$\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma \stackrel{\triangle}{=} \begin{cases} \sigma & \text{if } \neg\mathcal{B}[\![b]\!]\sigma \\ \mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!](\mathcal{C}[\![c]\!]\sigma) & \text{otherwise} \end{cases}$$

almost…

not well-founded recursion!

Recursive definitions

for divergence

$$\mathcal{C}[\![\cdot]\!] : \mathsf{Com} \to \mathbb{M} \to \mathbb{M} \cup \{\bot\}$$

$$\begin{aligned}
\mathcal{C}[\![\mathbf{skip}]\!]\sigma &\triangleq \sigma \\
\mathcal{C}[\![x := a]\!]\sigma &\triangleq \sigma[\mathcal{A}[\![a]\!]\sigma / x] \\
\mathcal{C}[\![c_0; c_1]\!]\sigma &\triangleq \mathcal{C}[\![c_1]\!](\mathcal{C}[\![c_0]\!]\sigma)
\end{aligned}$$

almost…

$$\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]\sigma \triangleq \begin{cases} \mathcal{C}[\![c_0]\!]\sigma & \text{if } \mathcal{B}[\![b]\!]\sigma \\ \mathcal{C}[\![c_1]\!]\sigma & \text{otherwise} \end{cases}$$

$$\mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!]\sigma \triangleq \begin{cases} \sigma & \text{if } \neg\mathcal{B}[\![b]\!]\sigma \\ \mathcal{C}[\![\mathbf{while}\ b\ \mathbf{do}\ c]\!](\mathcal{C}[\![c]\!]\sigma) & \text{otherwise} \end{cases}$$

almost…

not well-founded recursion!

how do we know one solution exists? how do we know it is unique?