Paul Downen

[2018/06/03] (1)

(Lecture 1)

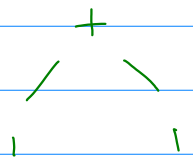
1. Syntax & scope (Downen)
2. Static & Dynamic Semantics of a prog. lang. (Hoffman)

LUNCH

3. Type Safety
4. λ-calculus

Example $1+1=2$ semantically, but not syntactically.

Syntactically,



$\neq 2$

- A language with numbers & booleans & variables

$x, y, z \in Variables ::= \ldots$

$n \in Num ::= 0 \mid 1 \mid 2 \mid \ldots$

$b \in Bool ::= True \mid False$

$e \in Expr ::= x \mid Num[n] \mid Bool[b] \mid Plus(e_1, e_2)$
$\mid Less(e_1, e_2) \mid If(e_1, e_2, e_3) \mid Let(e_1, x.e_2)$

$x.e_2$ means "bind free occurrences of x in e_2"
i.e. e_2 might have a free occurrence of x in it, but $x.e_2$ binds the x

Q: Isn't $\{\wedge, \vee\}$ more basic/primitive than if-then & let?

Paul Downen

(lecture 1)

[2018/06/03]

Static Scope

local (bound)

1. names of variables don't matter (i.e. α-equivalence)
2. substitution does not "capture" free variables.

EX: Let $x = 5$ in
Let $y = x$ in
Let $x = 10$ in y (static binding) (good)

Let $x = 5$
in let $x = 10$ in x (Dynamic binding) (no good because names & variables matter)

Substitution

$e'[e/x]$ want e' but whenever free x occurs in e', replace it with e.

Definition of substitution

Base cases
- $x[e/x] = e$
- $y[e/x] = y$
- $Num[n][e/x] = Num[n]$
- $Bool[b][e/x] = Bool[b]$

Inductive cases
- $Plus(e_1; e_2)[e/x] = Plus(e_1[e/x]; e_2[e/x])$
- $Less(e_1; e_2) = Less(e_1[e/x]; e_2[e/x])$
- $If(e_1, e_2, e_3)[e/x] = If(e_1[e/x]; e_2[e/x]; e_3[e/x])$

→ Let →

Substitution for Let

Ex: $(\text{Let } x = 1 \text{ in } x)[5/x] = \text{let } x = 1 \text{ in } x$

Case: $\text{Let}(e_1; x.e_2)[e/x] = \text{Let}(e_1[e/x]; x.e_2)$

↑ plug in for any free x's in e_1. ↑ don't plug in here because x is bound.

Case: $\text{Let}(e_1; y.e_2)[e/x] = \text{Let}(e_1[e/x]; y.e_2[e/x])$

$x \neq y$ & $y \notin FV(e)$

But these don't cover all the cases (To be cont)

Bound & Free Variables

$BV(x) = \emptyset$

$BV(\text{Num}[n]) = \emptyset$

$BV(\text{Bool}[b]) = \emptyset$

$BV(\text{Plus}(e_1; e_2)) = BV(e_1) \cup BV(e_2)$

Similar for If and Less.

$BV(\text{let}(e_1; x.e_2)) = BV(e_1) \cup BV(e_2) \cup \{x\}$.

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

$FV(\text{Num}[n]) = \emptyset = FV(\text{Bool}[b])$

$FV(\text{Plus}(e_1; e_2)) = FV(e_1) \cup FV(e_2) = FV(\text{less}(e_1; e_2))$

$FV(\text{Let}(e_1; x.e_2)) = FV(e_1) \cup (FV(e_2) \setminus \{x\})$

Paul Downen

[2018/06/03] (Lect 1)

α-equivalence

Syntactically, `let x=1 in x` is not equal to `let y=1 in y`

But `let x=1 in x` $=_\alpha$ `let y=1 in y`

let $x = e_1$ in e_2 $=_\alpha$ let $y = e_1$ in $e_2[y/x]$ $(y \notin FV(e_2))$

Properties of Substitution

1. For all e and e': if $BV(e') \cap FV(e) = \emptyset$ then $e'[e/x]$ is defined.

2. For all e, e': if $x \notin FV(e')$, then $e'[e/x] = e'$.

Properties of α-equivalence

1. For all e, e', $\exists e''$. $(e =_\alpha e''$ and $e''[e'/x]$ is defined)
(so, with α-equivalence, substitution becomes a total fn.)

2. If $e =_\alpha e'$, then $e[e''/x] =_\alpha e'[e''/x]$.

Homework

1. Add more operations to this language.
(e.g. minus, equal)

2. Prove some of the properties listed above
(by structural induction)