OPLSS-2018-Foundations of Programing Languages

Monday, July 2, 2018 9:12 PM

Location: 175 Knight Law Center
Morning sessions begin at 9:00 AM and run until noon
Afternoon sessions begin at 2:00 PM and run until 5:00 PM

7/3/2018 9:03 AM
Topics:

1. Syntax and Scope.
2. Static and Dynamic Semantics
3. Type Safety
4. ƛ-Calculus

A simple language:

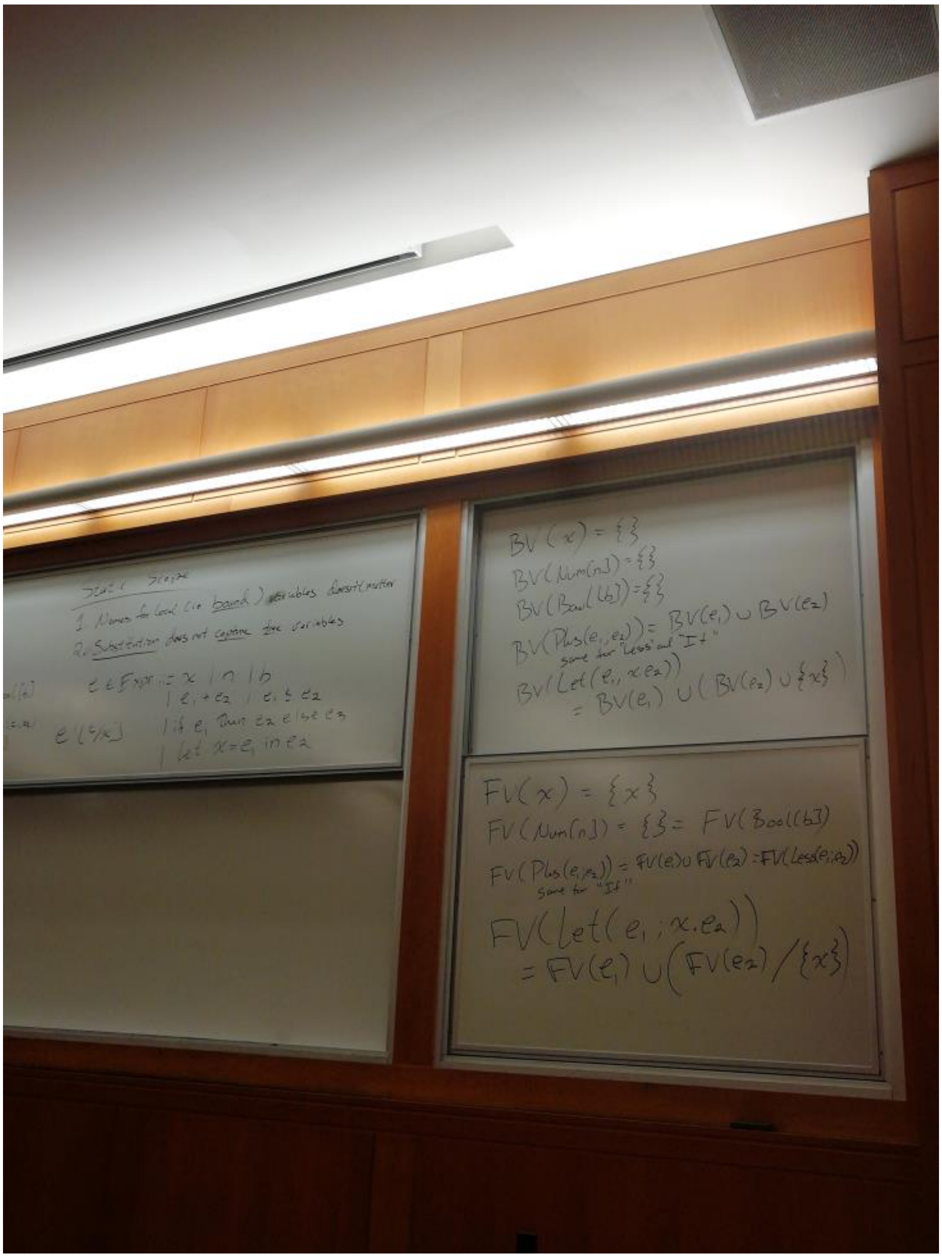
x, y, z ϵ Variables ::=..
N ϵ Num ::= 0|1|2|3|…
b ϵ Bool ::= True|False
e ϵ Expr ::= x | Num[n] | Bool [b]
|Plus (e_1, e_2) | Less (e_1, e_2)
|If (e_1; e_2; e_3)
|Let (e_1; x, e_2)

Static Scope

1. Names for local (i.e. bound) variables doesn't matter
2. Substitution does not capture free variables

Substitution Definition:
x[e/x] = e
y[e/x] = y (x) (x!=y)
Num[n][e/x] = Num[n]
Bool[b][e/x] = Bool(b)
Plus(e_1, e_2) [e/x] = Plus (e_1[e/x], e_1[e/x])
Less()
If (e_1; e_2; e_3) [e/x]
Let (e_1; x, e_2) [e/x] =

Bind variables:
BV (x)
BV ()
Num([n]) = {}
BV ()



Alpha Equivalence

Properties of Substitution:

Properties of alpha Equivalence:

7/3/2018 10:49 AM

Jan Haffmann (CMU)
Book: Bob Harpper

Practical Foundations of Programing Languages (PFPL)
What is a program languages?

Today: programs are math objects
How do we define the program languages?

1. Static semantics: what are (valid) programs?

- Option: programs are all expressions
- Not ideal. Programs that don't make sense should be extended like 5+true
- Observation: expressions come in 2 types: numbers and bools, type systems
  - Ecample: (1+2)+8 is a valid expression Why? They are valid expression of type num
  - We need to use inference rules to prove the expression is the num type
- Need induction.
- Notation : we write
- Intermission: Inductive Definitions
  - Examples: trees
    i. Emp is a tree

- Judgment: t tree
- Inference Rules, for defining judgements include
- Type rules
- The type of an expression is the result type of the expression
- Rule Induction
- Inversion:

7/3/2018 2:03 PM
Note share: williamdemeo@gmail.com

2. Dynamic semantics: How to
   a. Operational : how to run a program
   b. Axiomatic: what can you prove about a program?
   c. Denotational: Describe programs as math

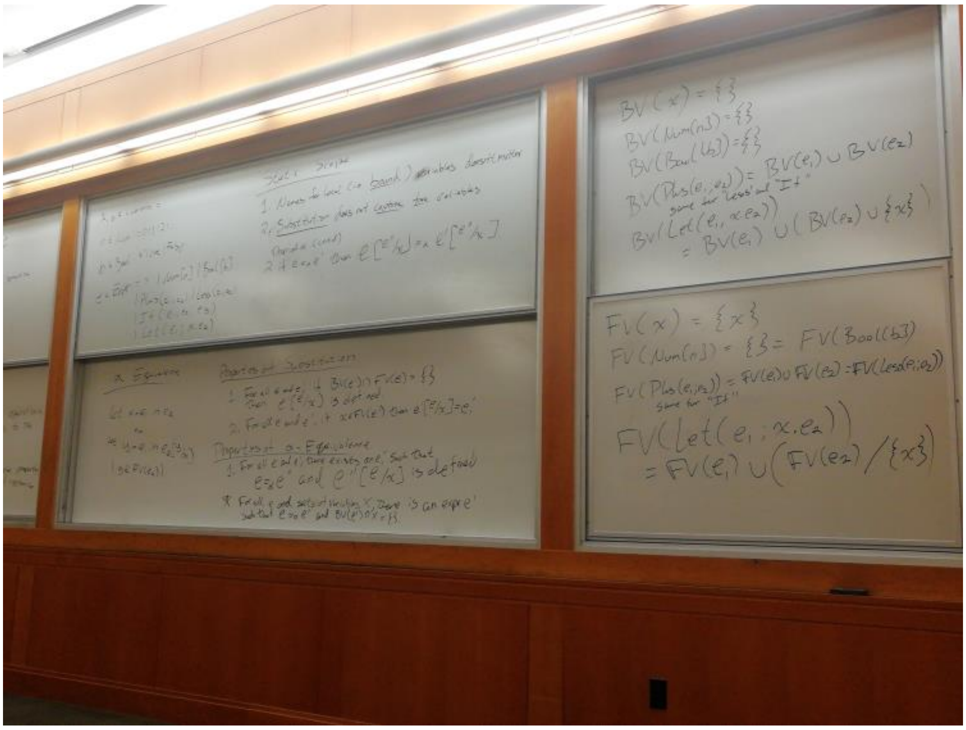
   Today: Structural operational semantics (small-step, semantics)
   Struct dynamics, transition system
   4 judgements: s state s initial s final s |-> s' (s can steps to s')

Interated transition:
- States are expressions (well-type and closed, closed means has no free variables)
- All states are initial
- Values are final states

Values: e val
Transitions
Type safety: are very important
- You don't get stuck in the dynamics

7/3/2018 4:01 PM

Lamda calculus
- A really small language
- Russel's paradox, a funny paradox in set theory



OPLSS2018-day-01

Wednesday, July 4, 2018 11:47 PM

7/3/2018 OPLSS 2018

Properties of α-Equivalence

1. For all ~~e~~ and e', then exits an e'' such that $e =_\alpha e''$ and $e''[e'/x]$ is defined
2. if $e =_\alpha e'$ then $e[e''/x] =_\alpha e'[e''/x]$

for all e and sets of variables, there is an ~~example~~ expr e' such that $e =_\alpha e'$ and $BV(e') \cap X = \{\}$

Homework: ① Add some more operations, Minus + Equal to little language

② Prove some of these properties of substitution and rename them

Properties of Substitution

1. For all e and e' if $BV(e') \cap FV(e) = \{\}$ then $e'[e'/x]$ is defined
2. For all e and e', if $x \notin FV(e')$ then $e'[e/x] = e'$

Part 2: Jan Hoffmann (CMU)

Lang E

$$
\begin{aligned}
\text{exp } e ::= \; & x \\
& num[n] \qquad plus(e_1; e_2) \\
& bool[true] \qquad leq(e_1; e_2) \\
& bool[false] \\
& if(e_1; e_2; e_3) \\
& let(e_1; x.e_2)
\end{aligned}
$$

Type Notation: $\vdash (1+2)+8 : num$?

in general: $\vdash e : \tau$

Inductive Definition: Example: Trees 1) emp is a tree 2) if n is number, t_1 is a tree and t_2 is a tree then node (n, t_1, t_2) is a tree

Inference Rules: $\dfrac{J_1 \cdots J_n}{J}$ ← premises / ← conclusion

$\dfrac{}{\text{emp tree}}$ (1) $\dfrac{n\ \text{num} \quad t_1\ \text{tree} \quad t_2\ \text{tree}}{\text{node}(n, t_1, t_2)\ \text{tree}}$ (2)

Derivations: —

Type Rules

$\dfrac{}{\Gamma \vdash \text{num}[n] : \text{num}}$ $\dfrac{}{\Gamma \vdash \text{bool}[b] : \text{bool}}$ $\dfrac{}{\Gamma, x \vdash x :}$ $\dfrac{\Gamma \vdash e_1 : \tau \quad \Gamma, x : \tau_1 \vdash e_2 : \tau}{\Gamma \vdash \text{let}(e_1, x.e_2) : \tau}$

context

assume $x \notin \Gamma$ (α equiv)

Type judgment:

$\Gamma \vdash e : \tau$ ← type, ← expr

context that maps vars to types

$\dfrac{\Gamma \vdash e_1 : \text{num} \quad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash \text{plus}(e_1, e_2) : \text{num}}$ (plus)

$\dfrac{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \text{if}(e_1; e_2; e_3) : \tau}$

Type of the expression is the type of result type

Rules: $\dfrac{\Gamma \vdash e_1 : \text{num} \quad \Gamma \vdash e_2 : \text{num}}{\Gamma \vdash e_1 \le e_2 : \text{bool}}$

Example: $\cdot \vdash \text{let } x = 5 \text{ in } x \le 6 : \text{bool}$

$$\dfrac{\dfrac{}{\cdot \vdash 5 : \text{num}}(\text{num}) \quad \dfrac{x : \text{num} \vdash x : \text{num} \quad x : \text{num} \vdash 6 : \text{num}}{x : \text{num} \vdash x \le 6 : \text{bool}}}{\cdot \vdash \text{let } x = 5 \text{ in } x \le 6 : \text{bool}} (\text{let})$$

Lemma 1: For every expr e and every context Γ there is at most one type τ such that $\Gamma \vdash e : \tau$

Prove: Counter $x : \text{bool} \vdash x : \text{bool}$

$x : \text{num} \vdash x : \text{num}$

Rule Induction:

Ind for nats: to show $P(n)$, prove (1) $P(z)$. (2) If $P(n)$ then $P(S(n))$

Def of nats:

$\dfrac{}{z\ \text{Nat}}$ $\dfrac{n\ \text{nat}}{S(n)\ \text{nat}}$

QPLSS 2018

7/3/2018

Rule Indu: To show $P(a)$, we show for every rule $\frac{a_1 J \cdots a_n J \ b J'}{a J}$ that

$P(a_1)$ and ... and $P(a_n)$ implies $P(a)$

Reform the lemma:

show: If $\Gamma \vdash e : \tau_1$ and $\Gamma \vdash e : \tau_2$ then $\tau_1 = \tau_2$

By induction "var rule" then $e = x$, $\Gamma = \Gamma_1' x : \tau_1$

Inversion Example: Inversion for $plus(e_1; e_2)$

Lemma: If $\Gamma \vdash e_1 + e_2 : \tau$ then $\tau = num$

and $\Gamma \vdash e_1 : num$ and $\Gamma \vdash e_2 : num$

$P(\Gamma, e, \tau_1) =$ If $\Gamma \vdash e : \tau_1$ and $\Gamma \vdash e : \tau_2$ then $\tau_1 = \tau_2$

case (plus):

Then $e = plus(e_1; e_2)$ and $\Gamma \vdash e_1 : num$ and $\Gamma \vdash e_2 : num$ and $\tau_1 = num$

Since $\Gamma \vdash plus(e_1, e_2) : \tau_2$ by inversion

$\Gamma \vdash e_1 : num$, $\Gamma \vdash e_2 : num$, and $\tau_2 = num$

Thus $\tau_1 = \tau_2 = num$.

Case (var):

Then $e = x$ and $\Gamma = \Gamma_1' x : \tau_1$

Since $\Gamma \vdash x : \tau_2$ by inversion $\Gamma = \Gamma_1'' x : \tau_2$

But then $\Gamma_1' x : \tau_1 = \Gamma_1'' x : \tau_2$ and $\tau_1 = \tau_2$

Lemma 2 (substitution):

If $\Gamma, x : \tau \vdash e' : \tau'$ and $\Gamma \vdash e : \tau$ then $\Gamma \vdash [e/x]e' : \tau'$

example: $x : num \vdash \overbrace{x \leq 5}^{e'} : bool$

$\vdash \underbrace{6}_{e} : num$

$[e/x]e' = 6 \leq 5$

emma 3 (Weakening) If $\Gamma \vdash e:\tau$ and $x \notin \Gamma$ then $\Gamma, x:\tau' \vdash e:\tau$
(prove, rule induction)

Iterated transition:

$$\frac{}{s \mapsto^* s} \qquad \frac{s \mapsto s' \quad s' \mapsto^* s''}{s \mapsto^* s''}$$ (star means many steps)

Values: e val $\qquad \frac{}{num[n]\ val} \qquad \frac{}{bool[b]\ val}$

Def: e is closed and well-typed: if $\cdot \vdash e:\tau$ for some τ

Transitions:

$$\frac{n = n_1 + n_2}{plus(num[n_1]; num[n_2]) \mapsto num[n]}$$

plus $\Rightarrow$ $\frac{e_1 \mapsto e_1'}{plus(e_1; e_2) \mapsto plus(e_1'; e_2)} \qquad \frac{e_2 \mapsto e_2'}{plus(num[n_1], e_2) \mapsto plus(num[n_1], e_2')}$

let $\Rightarrow$ $\frac{e_1 \mapsto e_1'}{let(e_1; x, e_2) \mapsto let(e_1'; x, e_2)} \qquad \frac{[e_1\ val]}{let(e_1'; x, e_2) \mapsto [e_1/x]e_2}$ call by name

if $\Rightarrow$ $\frac{e \mapsto e'}{if(e; e_1; e_2) \mapsto if(e'; e_1; e_2)} \qquad \frac{}{if(bool[True]; e_1; e_2) \mapsto e_1}$

$$\frac{}{if(bool[false]; e_1; e_2) \mapsto e_2}$$

call by value

Example: $let\ x = \underbrace{8+2}_{e_1}\ in\ \underbrace{(x+x)+2}_{e_2}$

$\mapsto let\ x=10\ in\ (x+x)+2$

$\mapsto (10+10)+2 \quad \mapsto 20+2 \quad \mapsto 22$

call by name

$let\ x = 8+2\ in\ (x+x)+2$

$\mapsto ((8+2)+(8+2))+2$

$\mapsto^* 22$

3
7/3/2018
OPLSS 2018

Lemma There is no expr e such that e val and $e \mapsto e'$ for some e

Lemma: If $e \mapsto e_1$ and $e \to e_2$ then $e_1 =_\alpha e_2$

Type Safety

theorem: $\underline{\text{write } e : \tau}$

1) progress If $\cdot \vdash e : \tau$ then either e val or there exists e' such that $e \mapsto e'$

2) preservation If $e : \tau$ and $e \mapsto e'$ then $e' : \tau$ (introduction on dynamics instead)

Proof:

Progress Rule ind on $e : \tau$

Rule (plus): Then $e = \text{plus}(e_1, e_2)$ $\tau = \text{num}$,

$e_1 : \text{num}$, and $e_2 : \text{num}$

IH: either e_1 val or there ex. e_1' such that $e_1 \mapsto e_1'$

either e_2 val or there ex. e_2' such that $e_2 \mapsto e_2'$

case e_1 val and e_2 val: Then by cell forms.

$e_1 = \text{num}[n_1]$ and $e_2 = \text{num}[n_2]$ for some n_2 and n_2

But then $e \mapsto \text{num}[n_1 + n_2]$

case e_1 val and $e_2 \mapsto e_2'$ then $e_1 = \text{num}[n]$ and

rule $\to e \mapsto \text{plus}(\text{num}[n], e_2')$

case $e_1 \mapsto e_1'$: Then $e \mapsto \text{plus}(e_1'; e_2)$ ✓ 3rd rule

exercises

Paul Downen λ - calculus

$e ::= x \mid e_1\ e_2 \mid \lambda x . e$

variable, result

using trees

$e ::= x \mid \text{app}(e_1 ; e_2) \mid \text{lam}(x.e)$

λ-calculus laws

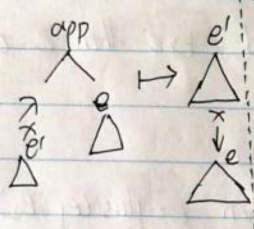
(α) $\lambda x.e =_\alpha \lambda y.e[y/x]$ $(y \notin FV(e))$

(β) $(\lambda x.e)e' =_\beta e[e'/x]$

(η) $\lambda x.(ex) =_\eta e$ $(x \notin FV(e))$

Call-by-Name

$(\lambda x.e)e' \mapsto e[e'/x]$



$$\frac{e_1 \mapsto e_1'}{e_1 e_2 \mapsto e_1' e_2}$$

$$\frac{(\lambda x.\lambda y.x)\,1 \mapsto \lambda y.1}{((\lambda x.\lambda y.x)\,1)\,2 \mapsto (\lambda y.1)\,2}$$

$\downarrow$

1

$E \in EvalCxt ::= \square \mid E\,e$

$\square[e] = e$

$(E\,e')[e] = (E[e])\,e'$

Call-by-Value

$(\lambda x.e)V \mapsto e[V/x]$

$V \in Value ::= x \mid \lambda x.e$

$E \in EvalCxt ::= \square \mid E\,e \mid V\,E$

$(V\,E)[e] = V\,(E[e])$

$$\frac{e \mapsto e'}{E[e] \mapsto E[e']}$$

$$\frac{}{x \text{ val}} \qquad \frac{}{\lambda x.e \text{ val}}$$

$$\frac{e' \text{ val}}{(\lambda x.e)e' \mapsto e[e'/x]}$$

$$\frac{e_1 \mapsto e_1'}{e_1 e_2 \mapsto e_1' e_2} \qquad \frac{e_1 \text{ val} \quad e_2 \mapsto e_2'}{e_1 e_2 \mapsto e_1 e_2'}$$

Example: Encoding Game :)

if e then e_1 else e_2 $\qquad (e\,e_1)\,e_2$

True $= \lambda x.\ \lambda y.\ x$

False $= \lambda x.\ \lambda y.\ y$

if True then e_1 else e_2 $\overset{*}{\mapsto} e_1$

if False then e_1 else e_2 $\overset{*}{\mapsto} e_2$

4
7/3/2018

Homework:

1. Encode natural numbers (≥ 0) in the lambda calculus

Example 2 $e \in e' = e' e$ $\quad e_1 \cup e_2 = \lambda x.$ or $(e_1 x)(e_2 x)$

$e_1 \cap e_2 = \lambda x.$ and $(e_1 x)(e_2 x)$

Russels Paradox (a fun

$R = \{ e \mid e \notin e \}$ $\quad R = \lambda x.$ not $(x\ x)$

$R \in R$? $\quad R\ R \mapsto (\text{not } (x\ x))[R/x] = \text{not } (R\ R)$

$\mapsto \text{not } (\text{not } (R\ R)) \mapsto \cdots$

$\mapsto \text{not } (\text{not } (\text{not } (\ldots)))$

$\Omega = (\lambda x.\ x x)(\lambda x.\ x x)$

$\Omega \mapsto \Omega$

$Y f = (\lambda x_1.\ f(x x))(\lambda x.\ f(x x))$

$Y f \mapsto f(Y f)$ $\rightarrow$ use it to implement recursion

times $= \lambda x.\lambda y$ if $x = 0$ then 0

else $y +$ (times $(x-1)$ y)

timesish $= \lambda$next. $\lambda x.\lambda y.$ if $x == 0$ then 0

else $y +$ (next $(x-1)$ y)

times $= Y$ timesish (homework, convince yourself it's times :)

$\tau \in \text{Type} ::= \tau_1 \to \tau_2 \mid \alpha$

$$\frac{}{\Gamma, x{:}\tau \vdash x : \tau} \qquad \frac{\Gamma \vdash e : \tau' \to \tau \quad \Gamma \vdash e' : \tau'}{\Gamma \vdash e\ e' : \tau} \rightarrow E$$

e' $\downarrow$ input type

$$\frac{\Gamma, x{:}\tau \vdash e : \tau'}{\Gamma \vdash \lambda x.\, e : \tau \to \tau'} \rightarrow I$$

τ' (type of functions) $\downarrow$ output type

(type systems

Theorem: Termination

If $\Gamma \vdash e : \tau$ then there is an e such that $e \mapsto^{*} e' \not\mapsto$