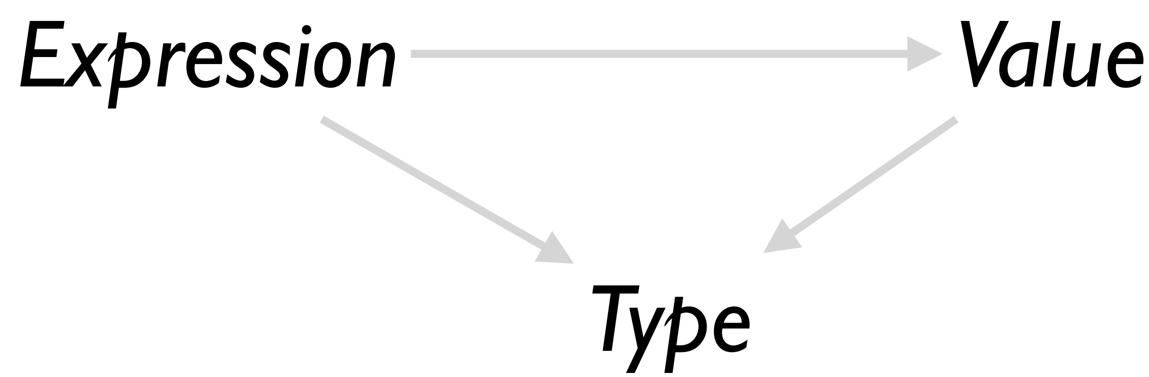
CS 162 Programming languages

Midterm Review

Yu Feng
Winter 2024

ML's holy grail



- Everything is an expression
- Everything has a value
- Everything has a type

Put it together: a “filter” function

If arg matches this pattern

then use this body expr

```
# let rec filter f l =
      match l with
         [] -> []
         | (h::t)-> if f h then h::(filter f t)
                    else (filter f t);;

val filter : ('a -> bool) -> 'a list -> 'a list = <fun>

# let list1 = [1;31;12;4;7;2;10];;
# filter is5lt list1 ;;
val it : int list = [31;12;7;10]
```

sum_leaf: tree -> int

```
type tree =
  Leaf of int
| Node of tree*tree
```

```
let rec sum_leaf t =
    match t with
    | Leaf n -> n
    | Node(t1,t2) -> (sum_leaf t1)
                    +(sum_leaf t2)
```

Syntax: what programs look like

$$
\begin{aligned}
e ::= \;& x \\
\mid\;& \lambda x.\, e \\
\mid\;& e_1\ e_2
\end{aligned}
$$

\x → e (Haskell)

fun x → e (OCaml)

lambda x. e (λ+)

- Programs are expressions e (also called λ-terms) of one of three kinds:
  - Variable x, y, z
  - Abstraction (i.e. nameless function definition)
    - $\lambda x.\, e$
    - x is the formal parameter, e is the function body
  - Application (i.e. function call)
    - $e_1\ e_2$
    - e_1 is the function, e_2 is the argument

Semantics: variable scope

The part of a program where a variable is visible

In the expression λx. e

- x is the newly introduced variable
- e is the scope of x
- any occurrence of x in λx. e is bound (by the binder λx)

x y

λx. x

λy. x y

λx. (λy. x)

(λx. λy. y) x

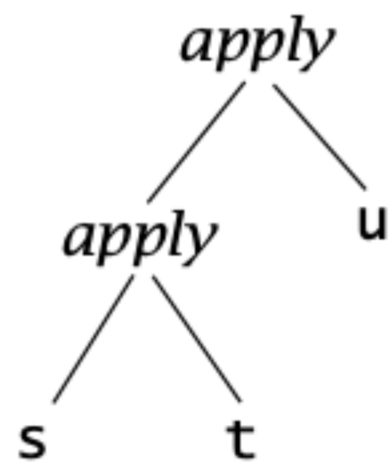
x is bounded

x is free

An occurrence of x in e is **free** if it's *not bound* by an enclosing abstraction

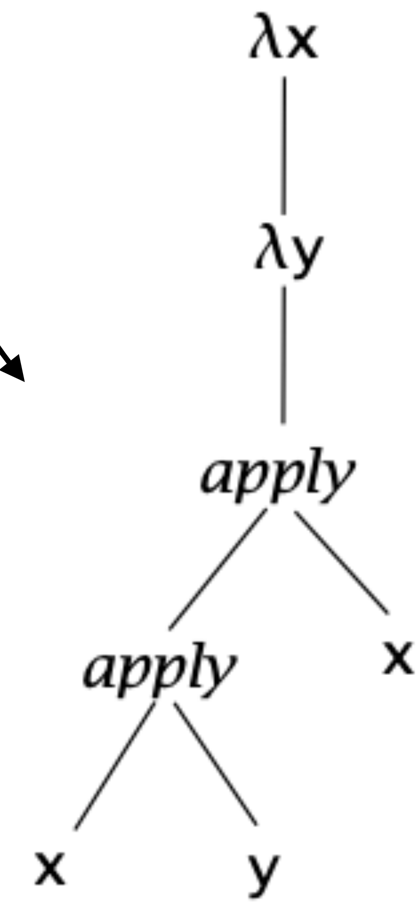
Precedence

$$s \;\; t \;\; u = (s \;\; t) \;\; u$$



Application associates to the **left**

$$\lambda x \,.\, \lambda y \,.\, x \;\; y \;\; x = \lambda x \,.\, (\lambda y \,.\, ((x \;\; y) \;\; x))$$



bodies of abstractions are as far to the **right** as possible

—*Types and programming languages*

Semantics: free variables

An variable x is free in e if there exists a free occurrence of x in e

We use "FV" to represent the set of all free variables in a term:

$$FV(x) = x$$
$$FV(\lambda x.\ e) = FV(e) \setminus x$$
$$FV(e_1\ e_2) = FV(e_1) \cup FV(e_2)$$

$$FV(x\ y) = \{x, y\}$$
$$FV(\lambda y.\ x\ y) = \{x\}$$
$$FV((\lambda x.\ \lambda y.\ y)\ x) = \{x\}$$

If e has no free variables it is said to be closed, or combinators

Semantics: β-reduction

$$(\lambda x \,.\, t_1)\ t_2 \rightarrow [x \mapsto t_2]t_1$$

β-reduction
(function call)

$[x \mapsto t_2]t_1$ means "t_1 with all ***free occurrences*** of x replaced with t_2"

$$
\begin{aligned}
x[x \mapsto e] &= e \\
y[x \mapsto e] &= y \quad \text{if } y \neq x \\
(\lambda x.c)[x \mapsto e] &= \lambda x.c \\
(\lambda y.c)[x \mapsto e] &= \lambda y.(c[x \mapsto e]) \quad \text{if } y \neq x \wedge y \notin \mathsf{FV}(e) \\
(c_1\, c_2)[x \mapsto e] &= (c_1[x \mapsto e])\, (c_2[x \mapsto e])
\end{aligned}
$$

Inference rules

$$\frac{\begin{array}{l}\text{Hypothesis 1}\\ \dots\\ \text{Hypothesis N}\end{array}}{\vdash \text{Conclusion}}$$

- This means“given hypothesis1,…N, the conclusion is provable”

$$\frac{\begin{array}{l}\text{Miterm 1 grade} >= 70\\ \dots\\ \text{Final grade} >= 140\end{array}}{\vdash \text{Final grade: A}}$$

Operational Semantics

- Operational semantics: define how program states are related to final values
- The ***big-step*** evaluation relation asserts that *we can prove for any expression of the form e that the meaning of this expression will evaluate to v*

$$e \Downarrow v$$

Operational Semantics in λ^+

$$\frac{}{i \Downarrow i}\ \textsc{Int} \qquad \frac{e_1 \Downarrow i_1 \qquad e_2 \Downarrow i_2}{e_1 \oplus e_2 \Downarrow i_1 \oplus i_2}\ \textsc{Arith}$$

$$\frac{e_1 \Downarrow i_1 \qquad e_2 \Downarrow i_2 \qquad i_1 \odot i_2 \text{ holds}}{e_1 \odot e_2 \Downarrow \mathsf{true}}\ \textsc{PredTrue}$$

$$\frac{e_1 \Downarrow i_1 \qquad e_2 \Downarrow i_2 \qquad i_1 \odot i_2 \text{ does not hold}}{e_1 \odot e_2 \Downarrow \mathsf{false}}\ \textsc{PredFalse}$$

$$\frac{e_1 \Downarrow \mathsf{true} \qquad e_2 \Downarrow v}{\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \Downarrow v}\ \textsc{IfTrue}$$

$$\frac{e_1 \Downarrow \mathsf{false} \qquad e_3 \Downarrow v}{\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \Downarrow v}\ \textsc{IfFalse}$$

$$\frac{}{\mathsf{lambda}\ x.\ e \Downarrow \mathsf{lambda}\ x.\ e}\ \textsc{Lambda}$$

$$\frac{e_1 \Downarrow \mathsf{lambda}\ x.\ e_1' \qquad e_2 \Downarrow v \qquad e_1'[x \mapsto v] \Downarrow v'}{(e_1\ e_2) \Downarrow v'}\ \textsc{App}$$

$$\frac{e_1 \Downarrow v_1 \qquad e_2[x \mapsto v_1] \Downarrow v_2}{\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \Downarrow v_2}\ \textsc{Let} \qquad \frac{e[f \mapsto \mathsf{fix}\ f\ \mathsf{is}\ e] \Downarrow v}{\mathsf{fix}\ f\ \mathsf{is}\ e \Downarrow v}\ \textsc{Fix}$$

$$\frac{}{\mathsf{Nil} \Downarrow \mathsf{Nil}}\ \textsc{Nil} \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2}{e_1 :: e_2 \Downarrow v_1 :: v_2}\ \textsc{Cons}$$

$$\frac{e_1 \Downarrow \mathsf{Nil} \qquad e_2 \Downarrow v}{\mathsf{match}\ e_1\ \mathsf{with}\ \mathsf{Nil} \rightarrow e_2 \mid x :: y \rightarrow e_3\ \mathsf{end} \Downarrow v}\ \textsc{MatchNil}$$

$$\frac{e_1 \Downarrow v_1 :: v_2 \qquad e_3[x \mapsto v_1][y \mapsto v_2] \Downarrow v_3}{\mathsf{match}\ e_1\ \mathsf{with}\ \mathsf{Nil} \rightarrow e_2 \mid x :: y \rightarrow e_3\ \mathsf{end} \Downarrow v_3}\ \textsc{MatchCons}$$

Type sytem in λ+

$$\frac{}{\Gamma \vdash i : \mathsf{Int}}\ \text{T-INT} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Int} \quad \Gamma \vdash e_2 : \mathsf{Int} \quad \Box \in \{+, -, *\}}{\Gamma \vdash e_1 \Box e_2 : \mathsf{Int}}\ \text{T-ARITH}$$

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{Bool}}\ \text{T-TRUE} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{Bool}}\ \text{T-FALSE}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{Bool} \quad \Gamma \vdash e_2 : \mathsf{T}_1 \quad \Gamma \vdash e_3 : \mathsf{T}_2 \quad \mathsf{T}_1 = \mathsf{T}_2}{\Gamma \vdash \mathsf{if}\, e_1\, \mathsf{then}\, e_2\, \mathsf{else}\, e_3 : \mathsf{T}_1}\ \text{T-IF}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{Int} \quad \Gamma \vdash e_2 : \mathsf{Int} \quad \Box \in \{<, >, =\}}{\Gamma \vdash e_1 \Box e_2 : \mathsf{Bool}}\ \text{T-REL}$$

$$\frac{\Gamma(x) = \mathsf{T}}{\Gamma \vdash x : \mathsf{T}}\ \text{T-VAR} \qquad \frac{\Gamma \vdash e_1 : \mathsf{T}_1 \quad x : \mathsf{T}_1, \Gamma \vdash e_2 : \mathsf{T}_2}{\Gamma \vdash \mathsf{let}\, x = e_1\, \mathsf{in}\, e_2 : \mathsf{T}_2}\ \text{T-LET}$$

$$\frac{x : \mathsf{T}_1, \Gamma \vdash e : \mathsf{T}_2}{\Gamma \vdash (\mathsf{lambda}\, x : \mathsf{T}_1.\ e) : \mathsf{T}_1 \to \mathsf{T}_2}\ \text{T-LAMBDA}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{T}_1 \to \mathsf{T}_2 \quad \Gamma \vdash e_2 : \mathsf{T}_3 \quad \mathsf{T}_1 = \mathsf{T}_3}{\Gamma \vdash (e_1\ e_2) : \mathsf{T}_2}\ \text{T-APP}$$

$$\frac{f : \mathsf{T}, \Gamma \vdash e : \mathsf{T}}{\Gamma \vdash (\mathsf{fix}\, f : \mathsf{T}\ \mathsf{is}\ e) : \mathsf{T}}\ \text{T-FIX}$$

$$\frac{}{\Gamma \vdash \mathsf{Nil}[\mathsf{T}] : \mathsf{List}[\mathsf{T}]}\ \text{T-NIL}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{T} \quad \Gamma \vdash e_2 : \mathsf{List}[\mathsf{T}]}{\Gamma \vdash e_1 :: e_2 : \mathsf{List}[\mathsf{T}]}\ \text{T-CONS}$$

$$\frac{\Gamma \vdash e_1 : \mathsf{List}[\mathsf{T}_1] \quad \Gamma \vdash e_2 : \mathsf{T}_2 \quad x : \mathsf{T}_1, y : \mathsf{List}[\mathsf{T}_1], \Gamma \vdash e_3 : \mathsf{T}_3 \quad \mathsf{T}_2 = \mathsf{T}_3}{\Gamma \vdash \mathsf{match}\ e_1\ \mathsf{with}\ \mathsf{Nil} \to e_2 \mid x :: y \to e_3\ \mathsf{end} : \mathsf{T}_2}\ \text{T-MATCH}$$

$$\frac{\Gamma \vdash e : \mathsf{T}_2 \quad \mathsf{T}_1 = \mathsf{T}_2}{\Gamma \vdash (e @ \mathsf{T}_1) : \mathsf{T}_1}\ \text{T-ANNOT}$$

Big idea

- Big Idea: Just because we cannot prove something about the original program does not mean we cannot prove something about an *abstraction* of the program.

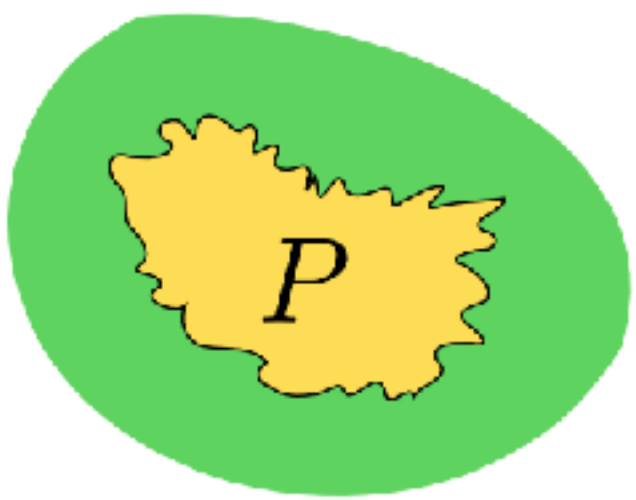
- Strategy: In addition to the operational semantics, we will also define *abstract semantics* that will overapproximate the states a program is in.

- Example: In λ^+, the operational semantics compute a concrete integer or list, while our abstract semantics only compute the if the result is of kind integer or list.

Soundness

- Specifically, we only care about abstract semantics that are sound

- Soundness means that for any program: If we evaluate it under *concrete* semantics (operational semantics) and our *abstract* semantics, the abstract value obtained overapproximates the concrete value.



$0=0 \Downarrow True \qquad 1 \Downarrow 1$

$\overline{}$ if

if $n=0$ then -- $\Downarrow$

$\lambda n.$ if $n=0$ --

else $1 * (\text{fix } r \text{ is } e)\, 0$

$\overline{}$ Fix $\qquad \overline{}$ int

fix r is $e \Downarrow \qquad 0 \Downarrow 0$

$\overline{}$ APP

$1=0 \Downarrow false \qquad 1 * (\text{fix } r \text{ is } e)\, 0 \Downarrow 1$

$\overline{}$ if

if $n=0$ then -- -- $\Downarrow$ 1

$[n \to 1]$

$\lambda n.$ if $n=0$ then 1

else $1 * (\text{fix } r \text{ is } e)\, (n-1)$

$\overline{}$ Fix $\qquad \overline{}$ int

fix r is λn -- -- -- $\Downarrow \qquad 1 \Downarrow 1$

$\overline{}$ APP

(fix r is $\lambda n.$

if $n=0$ then 1

else $1 * r(n-1)$) $1 \Downarrow 1$

$$\frac{\Gamma \vdash e_1 : \text{Int} \quad \Gamma \vdash e_2 : \text{Int} \quad \square \in \{+, -, ..\}}{\Gamma \vdash e_1 \square e_2 : \text{Int}}$$

N : negative number

Z : zero

P : positive number

$$\frac{\Gamma \vdash e_1 : N \quad \Gamma \vdash e_2 : P}{\Gamma \vdash e_1 + e_2 : N}$$

$$\frac{\Gamma \vdash e_1 : N \quad \Gamma \vdash e_2 : P}{\Gamma \vdash e_1 - e_2 : \text{Int}}$$

$$
\dfrac{
  \dfrac{\Gamma(r) = \mathrm{Int} \to \mathrm{Bool}}{r : \mathrm{Int} \to \mathrm{Bool}}
  \qquad
  \dfrac{m : \mathrm{Int} \qquad \dfrac{}{1 : \mathrm{Int}}\,\mathrm{Int}}{m : \mathrm{Int} \vdash m - 1 : \mathrm{Int}}\,\mathrm{Arith}
  \qquad
  \mathrm{Int} = \mathrm{Int}
}{
  \begin{array}{l} r : \mathrm{Int} \to \mathrm{Bool} \\ m : \mathrm{Int} \ \vdash \ r\ (m-1) : \mathrm{Bool} \end{array}
}\,\mathrm{APP}
$$

$$
\dfrac{\begin{array}{l} r : \mathrm{Int} \to \mathrm{Bool} \\ m : \mathrm{Int} \ \vdash \ r\ (m-1) : \mathrm{Bool} \end{array}}{r : \mathrm{Int} \to \mathrm{Bool} \vdash \lambda m : \mathrm{Int}.\ r\ (m-1) : \mathrm{Int} \to \mathrm{Bool}}\,\lambda
$$