□ Sample Language

Terms

$$\begin{aligned} e \;::=\;& x \mid \lambda x.e \mid e_1\, e_2 \mid \\ & (e_1, e_2) \mid \mathsf{fst}\, e \mid \mathsf{snd}\, e \mid () \mid \\ & \mathsf{fix}\, x.e \mid \\ & \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid \mathsf{tt} \mid \mathsf{ff} \mid \\ & \mathsf{lcase}\ e\ \mathsf{of}\ (e, e) \mid e : e \mid \mathsf{nil} \mid \\ & + \mid = \mid 0 \mid 1 \mid 2 \mid \ldots \end{aligned}$$

Canonical Forms: $c \;::=\; \lambda x.e \mid () \mid (e_1, e_2) \mid e_1 : e_2 \mid \mathsf{nil} \mid 0 \mid 1 \mid \ldots$

Values: $v \;::=\; \lambda x.e \mid () \mid (v_1, v_2) \mid v_1 : v_2 \mid \mathsf{nil} \mid 0 \mid 1 \mid \ldots$

□ Example Programs

$$
\begin{aligned}
\textsf{append} &\triangleq \textsf{fix}\, r.\lambda x.\lambda y. \\
&\qquad \textsf{lcase}\ x\ \textsf{of}\ (y,\ \lambda h.\lambda t.h : (r\,t\,y)) \\[1em]
\textsf{map} &\triangleq \textsf{fix}\, r.\lambda f.\lambda x. \\
&\qquad \textsf{lcase}\ x\ \textsf{of}\ (\textsf{nil},\ \lambda h.\lambda t.(f\,h) : (r\,f\,t)) \\[1em]
\textsf{filter} &\triangleq \textsf{fix}\, r.\lambda x.\lambda p. \\
&\qquad \textsf{lcase}\ x\ \textsf{of}\ (\textsf{nil},\ \lambda h.\lambda t.\textsf{if}\ p\,h\ \textsf{then}\ h : (r\,t\,p)\ \textsf{else}\ r\,t\,p) \\[1em]
\textsf{reduce} &\triangleq \textsf{fix}\, r.\lambda f.\lambda a.\lambda x. \\
&\qquad \textsf{lcase}\ x\ \textsf{of}\ (a,\ \lambda h.\lambda t.f\,h\,(r\,f\,a\,t))
\end{aligned}
$$

□ Alternative Formulations

$$\begin{aligned}
\mathsf{append} &\triangleq \lambda x.\lambda y.\mathsf{reduce}\,(\lambda h.\lambda t.h : t)\; y\; x \\
\mathsf{map} &\triangleq \lambda f.\lambda x.\mathsf{reduce}\,(\lambda h.\lambda t.(f\,h) : t)\;\mathsf{nil}\; x \\
\mathsf{filter} &\triangleq \lambda x.\lambda p.\mathsf{reduce}\,(\lambda h.\lambda t.\mathsf{if}\; p\,h\;\mathsf{then}\; h : t\;\mathsf{else}\; t)\;\mathsf{nil}\; x
\end{aligned}$$

$$\begin{aligned}
\mathsf{reduce}' \triangleq\; & \mathsf{fix}\, r.\lambda f.\lambda a.\lambda x. \\
& \quad \mathsf{lcase}\; x\;\mathsf{of}\;(a,\; \lambda h.\lambda t.r\; f\;(f\,h\,a)\; t)
\end{aligned}$$

□ More Example Programs

$$
\begin{array}{rcl}
\mathsf{zip} & \triangleq & \mathsf{fix}\, r.\lambda x.\lambda y. \\
& & \quad \mathsf{lcase}\ x\ \mathsf{of}\ (\mathsf{nil},\ \lambda h_1.\lambda t_1. \\
& & \qquad \mathsf{lcase}\ y\ \mathsf{of}\ (\mathsf{nil},\ \lambda h_2.\lambda t_2.(h_1, h_2) : (r\, t_1\, t_2))) \\
& & \\
\mathsf{eqlist} & \triangleq & \mathsf{fix}\, r.\lambda x.\lambda y. \\
& & \quad \mathsf{lcase}\ x\ \mathsf{of}\ (\mathsf{lcase}\ y\ \mathsf{of}\ (\mathsf{tt},\ \lambda h.\lambda t.\mathsf{ff}),\ \lambda h_1.\lambda t_1. \\
& & \qquad \mathsf{lcase}\ y\ \mathsf{of}\ (\mathsf{ff},\ \lambda h_2.\lambda t_2. \\
& & \qquad\quad \mathsf{if}\ h_1 = h_2\ \mathsf{then}\ r\, t_1\, t_2\ \mathsf{else}\ \mathsf{ff})) \\
& & \\
\mathsf{search} & \triangleq & \mathsf{fix}\, r.\lambda x.\lambda p.\lambda f.\lambda i. \\
& & \quad \mathsf{lcase}\ x\ \mathsf{of}\ (i,\ \lambda h.\lambda t.\mathsf{if}\ p\, h\ \mathsf{then}\ f\, h\ \mathsf{else}\ r\, t\, p\, f\, i)
\end{array}
$$

□ Call-by-value Evaluation (Scheme, ML)

$$\frac{}{\lambda x.e \hookrightarrow \lambda x.e}\,\mathsf{ev\_lam}$$

$$\frac{e_1 \hookrightarrow \lambda x.e \quad e_2 \hookrightarrow v_1 \quad e[x := v_1] \hookrightarrow v_2}{e_1\, e_2 \hookrightarrow v_2}\,\mathsf{ev\_app}$$

$$\frac{e_1 \hookrightarrow v_1 \quad e_2 \hookrightarrow v_2}{(e_1, e_2) \hookrightarrow (v_1, v_2)}\,\mathsf{ev\_pair} \qquad \frac{}{() \hookrightarrow ()}\,\mathsf{ev\_unit}$$

$$\frac{e \hookrightarrow (v_1, v_2)}{\mathsf{fst}\, e \hookrightarrow v_1}\,\mathsf{ev\_fst} \qquad \frac{e \hookrightarrow (v_1, v_2)}{\mathsf{snd}\, e \hookrightarrow v_2}\,\mathsf{ev\_snd}$$

$$\frac{e[x := \mathsf{fix}\, x.e] \hookrightarrow v}{\mathsf{fix}\, x.e \hookrightarrow v}\,\mathsf{ev\_fix}$$

□ Call-by-value contd.

$$\frac{}{\mathsf{tt} \hookrightarrow \mathsf{tt}}\ \mathsf{ev\_tt} \qquad \frac{}{\mathsf{ff} \hookrightarrow \mathsf{ff}}\ \mathsf{ev\_ff}$$

$$\frac{e_1 \hookrightarrow \mathsf{tt} \quad e_2 \hookrightarrow v}{\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \hookrightarrow v}\ \mathsf{ev\_if\_tt} \qquad \frac{e_1 \hookrightarrow \mathsf{ff} \quad e_3 \hookrightarrow v}{\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \hookrightarrow v}\ \mathsf{ev\_if\_ff}$$

$$\frac{}{\mathsf{nil} \hookrightarrow \mathsf{nil}}\ \mathsf{ev\_nil} \qquad \frac{e_1 \hookrightarrow v_1 \quad e_2 \hookrightarrow v_2}{e_1 : e_2 \hookrightarrow v_1 : v_2}\ \mathsf{ev\_cons}$$

$$\frac{e_1 \hookrightarrow \mathsf{nil} \quad e_2 \hookrightarrow v}{\mathsf{lcase}\ e_1\ \mathsf{of}\ (e_2, e_3) \hookrightarrow v}\ \mathsf{ev\_lc\_nil} \qquad \frac{e_1 \hookrightarrow v_1 : v_2 \quad e_3\, v_1\, v_2 \hookrightarrow v}{\mathsf{lcase}\ e_1\ \mathsf{of}\ (e_2, e_3) \hookrightarrow v}\ \mathsf{ev\_lc\_cons}$$

natural numbers
evaluate as expected

□ Call-by-value Example

$$\mathsf{app} \triangleq \lambda x.\lambda y.\mathsf{lcase}\ x\ \mathsf{of}\ (y,\ \lambda h.\lambda t.h : (\mathsf{append}\, t\, y))$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{app} \hookrightarrow \mathsf{app}}\,\mathsf{ev\_lam}}{\mathsf{append} \hookrightarrow \mathsf{app}}\,\mathsf{ev\_fix} \quad \dfrac{\dfrac{}{1 \hookrightarrow 1}\,\mathsf{ev\_1} \quad \dfrac{}{\mathsf{nil} \hookrightarrow \mathsf{nil}}\,\mathsf{ev\_nil}}{1 : \mathsf{nil} \hookrightarrow 1 : \mathsf{nil}}\,\mathsf{ev\_cons} \quad \mathcal{D}_1}{\mathsf{append}\,(1 : \mathsf{nil}) \hookrightarrow \lambda y.\mathsf{lcase}\ 1 : \mathsf{nil}\ \mathsf{of}\ (y,\ \lambda h.\lambda t.h : (\mathsf{append}\, t\, y))}\,\mathsf{ev\_app} \quad \mathcal{D}_2 \quad \mathcal{D}_3}{\mathsf{append}\,(1 : \mathsf{nil})\,(2 : 3 : \mathsf{nil}) \hookrightarrow 1 : 2 : 3 : \mathsf{nil}}\,\mathsf{ev\_app}$$

$\mathcal{D}_1 =$

$$\dfrac{}{\lambda y.\mathsf{lcase}\ 1 : \mathsf{nil}\ \mathsf{of}\ (y,\ \lambda h.\lambda t.h : (\mathsf{append}\, t\, y)) \hookrightarrow \lambda y.\mathsf{lcase}\ 1 : \mathsf{nil}\ \mathsf{of}\ (y,\ \lambda h.\lambda t.h : (\mathsf{append}\, t\, y))}\,\mathsf{ev\_lam}$$

$\mathcal{D}_2 =$

$$\dfrac{\dfrac{}{1 \hookrightarrow 1}\,\mathsf{ev\_1} \quad \dfrac{\dfrac{}{3 \hookrightarrow 3}\,\mathsf{ev\_3} \quad \dfrac{}{\mathsf{nil} \hookrightarrow \mathsf{nil}}\,\mathsf{ev\_nil}}{3 : \mathsf{nil} \hookrightarrow 3 : \mathsf{nil}}\,\mathsf{ev\_cons}}{2 : 3 : \mathsf{nil} \hookrightarrow 2 : 3 : \mathsf{nil}}\,\mathsf{ev\_cons}$$

□ Call-by-value Example contd.

$\mathcal{D}_3 =$

$$\dfrac{\dfrac{\dfrac{}{1 \hookrightarrow 1}\,\text{ev\_1} \quad \dfrac{}{\text{nil} \hookrightarrow \text{nil}}\,\text{ev\_nil}}{1 : nil \hookrightarrow 1 : \text{nil}}\,\text{ev\_cons} \qquad \dfrac{\color{purple}{\mathcal{D}_4} \quad \dfrac{}{\text{nil} \hookrightarrow \text{nil}}\,\text{ev\_nil} \quad \color{purple}{\mathcal{D}_5}}{(\lambda h.\lambda t.h : (\text{append}\, t\, (2:3:\text{nil})))\, 1\, \text{nil} \hookrightarrow 1:2:3:\text{nil}}\,\text{ev\_app}}{\text{lcase } 1:\text{nil of } (2:3:\text{nil}, \lambda h.\lambda t.h : (\text{append}\, t\, (2:3:\text{nil}))) \hookrightarrow 1:2:3:\text{nil}}\,\text{ev\_lc\_cons}$$

$\mathcal{D}_4 =$

$$\dfrac{\dfrac{}{\lambda h.\lambda t.h : (\text{append}\, t\, (2:3:\text{nil})) \hookrightarrow \lambda h.\lambda t.h : (\text{append}\, t\, (2:3:\text{nil}))}\,\text{ev\_lam} \quad \dfrac{}{1 \hookrightarrow 1}\,\text{ev\_1} \quad \color{purple}{\mathcal{D}_6}}{(\lambda h.\lambda t.h : (\text{append}\, t\, (2:3:\text{nil})))\, 1 \hookrightarrow \lambda t.1 : (\text{append}\, t\, (2:3:\text{nil}))}\,\text{ev\_app}$$

$\mathcal{D}_6 =$

$$\dfrac{}{\lambda t.1 : (\text{append}\, t\, (2:3:\text{nil})) \hookrightarrow \lambda t.1 : (\text{append}\, t\, (2:3:\text{nil}))}\,\text{ev\_lam}$$

$\mathcal{D}_5 =$

$$\dfrac{\dfrac{}{1 \hookrightarrow 1}\,\text{ev\_1} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\text{app} \hookrightarrow \text{app}}\,\text{ev\_lam}}{\text{append} \hookrightarrow \text{app}}\,\text{ev\_fix} \quad \dfrac{}{\text{nil} \hookrightarrow \text{nil}}\,\text{ev\_nil} \quad \color{purple}{\mathcal{D}_7}}{\text{append}\,\text{nil} \hookrightarrow \lambda y.\text{lcase nil of } (y, \lambda h.\lambda t.h : (\text{append}\, t\, y))}\,\text{ev\_app} \quad \color{purple}{\mathcal{D}_8} \quad \color{purple}{\mathcal{D}_9}}{\text{append}\,\text{nil}\,(2:3:\text{nil}) \hookrightarrow 2:3:\text{nil}}\,\text{ev\_a}}{1 : (\text{append}\,\text{nil}\,(2:3:\text{nil})) \hookrightarrow 1:2:3:\text{nil}}\,\text{ev\_cons}$$

□ Call-by-value Example contd.

$\mathcal{D}_7 =$

$$\dfrac{}{\lambda y.\textsf{lcase nil of } (y,\, \lambda h.\lambda t.h\,{:}\,(\textsf{append}\, t\, y)) \hookrightarrow \lambda y.\textsf{lcase nil of } (y,\, \lambda h.\lambda t.h\,{:}\,(\textsf{append}\, t\, y))}\ \textsf{ev\_lam}$$

$\mathcal{D}_8 =$

$$\dfrac{\dfrac{}{2 \hookrightarrow 2}\ \textsf{ev\_2} \qquad \dfrac{\dfrac{}{3 \hookrightarrow 3}\ \textsf{ev\_3} \qquad \dfrac{}{\textsf{nil} \hookrightarrow \textsf{nil}}\ \textsf{ev\_nil}}{3\,{:}\,\textsf{nil} \hookrightarrow 3\,{:}\,\textsf{nil}}\ \textsf{ev\_cons}}{2\,{:}\,3\,{:}\,\textsf{nil} \hookrightarrow 2\,{:}\,3\,{:}\,\textsf{nil}}\ \textsf{ev\_cons}$$

$\mathcal{D}_9 =$

$$\dfrac{\dfrac{}{\textsf{nil} \hookrightarrow \textsf{nil}}\ \textsf{ev\_nil} \qquad \dfrac{\dfrac{}{2 \hookrightarrow 2}\ \textsf{ev\_2} \qquad \dfrac{\dfrac{}{3 \hookrightarrow 3}\ \textsf{ev\_3} \qquad \dfrac{}{\textsf{nil} \hookrightarrow \textsf{nil}}\ \textsf{ev\_nil}}{3\,{:}\,\textsf{nil} \hookrightarrow 3\,{:}\,\textsf{nil}}\ \textsf{ev\_cons}}{2\,{:}\,3\,{:}\,\textsf{nil} \hookrightarrow 2\,{:}\,3\,{:}\,\textsf{nil}}\ \textsf{ev\_cons}}{\textsf{lcase nil of } (2\,{:}\,3\,{:}\,\textsf{nil},\, \lambda h.\lambda t.h\,{:}\,(\textsf{append}\, t\, 2\,{:}\,3\,{:}\,\textsf{nil})) \hookrightarrow 2\,{:}\,3\,{:}\,\textsf{nil}}\ \textsf{ev\_lc\_nil}$$

□ Eager Evaluation

Evaluates each argument once.

Sometimes call-by-value is too eager:

$$\begin{aligned}\mathsf{search} \;&\triangleq\; \mathsf{fix}\, f.\lambda x.\lambda p.\lambda g.\lambda i.\\ &\qquad \mathsf{lcase}\ x\ \mathsf{of}\ (i,\ \lambda h.\lambda t.\mathsf{if}\ p\,h\ \mathsf{then}\ g\,h\ \mathsf{else}\ f\,t\,p\,g\,i)\end{aligned}$$

failure expression, i, is always evaluated in recursive call.
This can be fixed by "thunking":

$$\begin{aligned}\mathsf{search} \;&\triangleq\; \mathsf{fix}\, f.\lambda x.\lambda p.\lambda g.\lambda i.\\ &\qquad \mathsf{lcase}\ x\ \mathsf{of}\ (i\,(),\ \lambda h.\lambda t.\mathsf{if}\ p\,h\ \mathsf{then}\ g\,h\ \mathsf{else}\ f\,t\,p\,g\,i)\end{aligned}$$

i must now be a function which takes a dummy, (), argument.

Eager evaluation can also cause "unnecessary" non-termination.

□ Call-by-name Evaluation

$$\frac{}{\lambda\, x.e \ \hookrightarrow\ \lambda x.e}\ \textsf{ev\_lam}$$

$$\frac{e_1 \ \hookrightarrow\ \lambda x.e \qquad e[x := e_2] \ \hookrightarrow\ v}{e_1\, e_2 \ \hookrightarrow\ v}\ \textsf{ev\_app}$$

$$\frac{}{(e_1, e_2) \ \hookrightarrow\ (e_1, e_2)}\ \textsf{ev\_pair} \qquad \frac{}{() \ \hookrightarrow\ ()}\ \textsf{ev\_unit}$$

$$\frac{e \ \hookrightarrow\ (v_1, v_2)}{\textsf{fst}\, e \ \hookrightarrow\ v_1}\ \textsf{ev\_fst} \qquad \frac{e \ \hookrightarrow\ (v_1, v_2)}{\textsf{snd}\, e \ \hookrightarrow\ v_2}\ \textsf{ev\_snd}$$

$$\frac{e[x := \textsf{fix}\, x.e] \ \hookrightarrow\ v}{\textsf{fix}\, x.e \ \hookrightarrow\ v}\ \textsf{ev\_fix}$$

□ Call-by-name contd.

$$\frac{}{\mathsf{tt} \hookrightarrow \mathsf{tt}}\,\mathsf{ev\_tt} \qquad \frac{}{\mathsf{ff} \hookrightarrow \mathsf{ff}}\,\mathsf{ev\_ff}$$

$$\frac{e_1 \hookrightarrow \mathsf{tt} \quad e_2 \hookrightarrow v}{\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \hookrightarrow v}\,\mathsf{ev\_if\_tt} \qquad \frac{e_1 \hookrightarrow \mathsf{ff} \quad e_3 \hookrightarrow v}{\mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \hookrightarrow v}\,\mathsf{ev\_if\_ff}$$

$$\frac{}{\mathsf{nil} \hookrightarrow \mathsf{nil}}\,\mathsf{ev\_nil} \qquad \frac{}{e_1 : e_2 \hookrightarrow e_1 : e_2}\,\mathsf{ev\_cons}$$

$$\frac{e_1 \hookrightarrow \mathsf{nil} \quad e_2 \hookrightarrow v}{\mathsf{lcase}\ e_1\ \mathsf{of}\ (e_2, e_3) \hookrightarrow v}\,\mathsf{ev\_lc\_nil} \qquad \frac{e_1 \hookrightarrow v_1 : v_2 \quad e_3\, v_1\, v_2 \hookrightarrow v}{\mathsf{lcase}\ e_1\ \mathsf{of}\ (e_2, e_3) \hookrightarrow v}\,\mathsf{ev\_lc\_cons}$$

natural numbers
evaluate as expected

□ Lazy Evaluation

Call-by-name only evaluates an expression when needed.

$$\mathsf{eqlist}\,(1:2:\cdots:100:\mathsf{nil})\,(0:1:\cdots:100:\mathsf{nil})$$

is much faster in call-by-name than call-by-value.

Avoids non-termination from unused arguments, but very inefficient:

$$(\lambda x.x + x + x + x)\,\mathsf{long\_computation}$$

computes long_computation four times.

Actual implementations (e.g. Haskell) use call-by-need, or lazy, evaluation which waits to evaluate until necessary, but caches the value a variable is bound to so that further uses of the variable do not have to recompute its value.

□ Infinite Structures

Laziness allows for directly handling infinite structures:

$$\begin{aligned}
\mathsf{zeros} &\triangleq \mathsf{fix}\, l.\, 0 : l \\
\mathsf{nats} &\triangleq \mathsf{fix}\, l.\, 0 : (\mathsf{map}\,(\lambda x. x + 1)\, l) \\
\mathsf{enum} &\triangleq \lambda l. \mathsf{zip}\ \mathsf{nats}\ l
\end{aligned}$$

Have to be careful when applying functions to inifinite lists.

- append, map, zip behave as expected on infinite lists.
- reduce′ always diverges on an infinite lists.
- filter, reduce, search, eqlist might diverge on an infinite list.

□ Some Haskell Syntax

fix is implicit in all Haskell definitions.

```
let zeros = 0:zeros
let nats = 0:(map (\x -> x + 1) nats)
let enum = \l -> zip nats l
```

Some alternatives for nats:

```
let nats = 0:(map (+ 1) nats)
let nats = [0 ..]
```

Notes:

```
(+ 1) ≡ (\x -> x + 1)
[0 .. 5] ≡ 0:1:2:3:4:5:[] ≡ [0, 1, 2, 3, 4, 5]
```

□ More Syntax

Haskell has pattern matching. Note that this forces evaluation of argument to the shape of pattern:

```
let pair_ones = (1,1):pair_ones
(\(x,y) -> ()) pair_ones!!1000000
```

Takes a while to evaluate even though argument isn't used.

Some useful Haskell built-in functions:

`[1,2,3,4,5]!!3` $\equiv$ `4`. (Note `!!` is an infix function.)

`take 3 [1,2,3,4,5]` $\equiv$ `[1,2,3]`

□ Lazy Evaluation

Consider expression:

```
let sums = (1,1):(map (\(i,n) -> (i+1, n+i+1)) sums)
```

Consider an explicitly delayed version:

```
let ones = 1:ones
let sums' =
    (1,1):(map (\(i,n) -> (i+(ones!!1000000), n+i+1)) sums')
```

First evaluation of `sums'!!1000` takes a while, but subsequent calls need not reevaluate the list.

□ Homework 2

1. [20 points]

   (a) What does:

   $$\mathsf{append}\,(1\!:\!\mathsf{nil})\,(2\!:\!3\!:\!\mathsf{nil})$$

   evaluate to using call-by-name?

   (b) Write a complete call-by-name evaluation derivation for:

   $$\mathsf{append}\,(1\!:\!\mathsf{nil})\,(2\!:\!3\!:\!\mathsf{nil}) \hookrightarrow v$$

   where v is your answer to part a.

2. [20 points]
   Write an expression for an inifinite list representing the graph of the factorial function:

   $$e = (1,1) : (2,2) : (3,6) : (4,24) : (5,120) : (6,720) : (7,5040) : \ldots$$

3. [30 points]
   Write an expression for an inifinite list representing the graph of the Fibonacci function:

   $$e = (1,1) : (2,1) : (3,2) : (4,3) : (5,5) : (6,8) : (7,13) : \ldots$$