Compiling an Interpreter

or

Fun with Algebra

the specification, use, and implementation of functional languages

CS6940, Fall 2000

Outline

- Programming with Functions
- Defining a Functional Language
- Type-Checking a Functional Program
- Implementing a Functional Language

Programming with Functions

- A program comprises function definitions and applications

$$\mathbf{f}(\mathbf{x}) \equiv (\mathbf{x} \bullet \mathbf{x}) + 10$$

$$\mathbf{f}(2) = 14$$

Programming with Functions

- A program comprises function definitions and applications

$$\mathbf{f}(\mathbf{x}) \equiv (\mathbf{x} \bullet \mathbf{x}) + 10$$

$$\mathbf{g}(\mathbf{y}) \equiv 3 \bullet \mathbf{y}$$

$$\mathbf{g}(\mathbf{f}(2)) = 42$$

Programming with Functions

- Functions consume and produce more than numbers

$$\textbf{mkpair}(\mathbf{x}, \mathbf{y}) \equiv \langle \mathbf{x}, \mathbf{y} \rangle$$

$$\textbf{mkpair}(1, 2) = \langle 1, 2 \rangle$$

Programming with Functions

- Functions consume and produce more than numbers

$$\textbf{mkpair}(\textbf{x}, \textbf{y}) \equiv \langle \textbf{x}, \textbf{y} \rangle$$

$$\textbf{mklist}(\textbf{x}, \textbf{y}) \equiv \textbf{mkpair}(\textbf{x}, \textbf{mkpair}(\textbf{y}, \text{empty}))$$

$$\textbf{mklist}(1, 2) = \langle 1, \langle 2, \text{empty} \rangle \rangle$$

Programming with Functions

- Functions consume and produce more than numbers

$$\mathbf{mkpair}(\mathbf{x}, \mathbf{y}) \equiv \langle \mathbf{x}, \mathbf{y} \rangle$$

$$\mathbf{mklist}(\mathbf{x}, \mathbf{y}) \equiv \mathbf{mkpair}(\mathbf{x}, \mathbf{mkpair}(\mathbf{y}, \mathrm{empty}))$$

$$\mathbf{fst}(\langle \mathbf{x}, \mathbf{y} \rangle) \equiv \mathbf{x}$$

$$\mathbf{fst}(\mathbf{mklist}(1, 2)) = 1$$

Programming with Functions

- Use functions to build complex data from simple constructs
- Implement branches with conditional functions

$$\mathbf{add}(\mathbf{n}, \mathbf{N}, \mathbf{pb}) \equiv \langle\langle \mathbf{n}, \mathbf{N}\rangle, \mathbf{pb}\rangle$$

$$\mathbf{lookup}(\mathbf{n}, \langle\langle \mathbf{n2}, \mathbf{N}\rangle, \mathbf{pb}\rangle) \equiv \begin{cases} \mathbf{n} = \mathbf{n2} & \mathbf{N} \\ \mathbf{n} \neq \mathbf{n2} & \mathbf{lookup}(\mathbf{n}, \mathbf{pb}) \end{cases}$$

$$\mathbf{lookup}(\text{"Jack"}, \mathbf{add}(\text{"Jack"}, \text{"x1212"}, \text{empty})) = \text{"x1212"}$$

Computation as Algebra

- Compute using algebraic equivalences

$$\mathbf{f}(\mathbf{x}) \equiv (\mathbf{x} \bullet \mathbf{x}) + 10$$

$$\mathbf{f}(2) \quad =$$

Computation as Algebra

- Compute using algebraic equivalences

$$\mathbf{f}(\mathbf{x}) \equiv (\mathbf{x} \bullet \mathbf{x}) + 10$$

$$\begin{aligned}\mathbf{f}(2) &= (2 \bullet 2) + 10 \\ &= 4 + 10 \\ &= 14\end{aligned}$$

Computation as Algebra

- Equivalence is pattern matching...

$$\mathbf{mkpair}(\mathbf{x}, \mathbf{y}) \equiv \langle \mathbf{x}, \mathbf{y} \rangle$$

$$\mathbf{mklist}(\mathbf{x}, \mathbf{y}) \equiv \mathbf{mkpair}(\mathbf{x}, \mathbf{mkpair}(\mathbf{y}, \text{empty}))$$

$$\mathbf{mklist}(1, 2) \quad =$$

Computation as Algebra

- Equivalence is pattern matching...

$$\mathbf{mkpair}(\mathbf{x}, \mathbf{y}) \equiv \langle \mathbf{x}, \mathbf{y} \rangle$$

$$\mathbf{mklist}(\mathbf{x}, \mathbf{y}) \equiv \mathbf{mkpair}(\mathbf{x}, \mathbf{mkpair}(\mathbf{y}, \text{empty}))$$

$$\begin{aligned} \mathbf{mklist}(1, 2) &= \mathbf{mkpair}(1, \mathbf{mkpair}(2, \text{empty})) \\ &= \langle 1, \mathbf{mkpair}(2, \text{empty}) \rangle \\ &= \langle 1, \langle 2, \text{empty} \rangle \rangle \end{aligned}$$

$$\begin{aligned} \textit{or} \quad &= \mathbf{mkpair}(1, \mathbf{mkpair}(2, \text{empty})) \\ &= \mathbf{mkpair}(1, \langle 2, \text{empty} \rangle) \\ &= \langle 1, \langle 2, \text{empty} \rangle \rangle \end{aligned}$$

Computation as Algebra

- ... and matching with conditionals

$$\textbf{add}(\textbf{n}, \textbf{N}, \textbf{pb}) \equiv \langle\langle \textbf{n}, \textbf{N}\rangle, \textbf{pb}\rangle$$

$$\textbf{lookup}(\textbf{n}, \langle\langle \textbf{n2}, \textbf{N}\rangle, \textbf{pb}\rangle) \equiv \begin{cases} \textbf{n} = \textbf{n2} & \textbf{N} \\ \textbf{n} \neq \textbf{n2} & \textbf{lookup}(\textbf{n}, \textbf{pb}) \end{cases}$$

---

lookup("Jack", **add**("Jack", "x1212", empty))

= **lookup**("Jack", ⟨⟨"Jack", "x1212"⟩, empty⟩)

= "x1212"

Computation as Algebra

- ... and matching with conditionals

$$\textbf{add}(\textbf{n}, \textbf{N}, \textbf{pb}) \equiv \langle\langle \textbf{n}, \textbf{N}\rangle, \textbf{pb}\rangle$$

$$\textbf{lookup}(\textbf{n}, \langle\langle \textbf{n2}, \textbf{N}\rangle, \textbf{pb}\rangle) \equiv \begin{cases} \textbf{n} = \textbf{n2} & \textbf{N} \\ \textbf{n} \neq \textbf{n2} & \textbf{lookup}(\textbf{n}, \textbf{pb}) \end{cases}$$

lookup("Jill", **add**("Jack", "x1212", empty))

= **lookup**("Jill", ⟨⟨"Jack", "x1212"⟩, empty⟩)

= **lookup**("Jill", empty)

stuck implies an error

Higher-Order Functions

- A ***higher-order function*** is one that consumes or produces functions

$$\mathbf{f}(\mathbf{x}) \equiv \mathbf{x} \bullet \mathbf{x}$$

$$\mathbf{twice}(\mathbf{g}, \mathbf{x}) \equiv \mathbf{g}(\mathbf{g}(\mathbf{x}))$$

$$\begin{aligned} \mathbf{twice}(\mathbf{f}, 2) &= \mathbf{f}(\mathbf{f}(2)) \\ &= \mathbf{f}(2 \bullet 2) \\ &= \mathbf{f}(4) \\ &= 4 \bullet 4 \\ &= 16 \end{aligned}$$

Higher-Order Functions

- A ***higher-order function*** is one that consumes or produces functions

$$\mathbf{fst}(\langle \mathbf{x}, \mathbf{y} \rangle) \equiv \mathbf{x}$$

$$\mathbf{twice}(\mathbf{g}, \mathbf{x}) \equiv \mathbf{g}(\mathbf{g}(\mathbf{x}))$$

$$\begin{aligned} \mathbf{twice}(\mathbf{fst}, \langle\langle 1, 2\rangle, 3\rangle) &= \mathbf{fst}(\mathbf{fst}(\langle\langle 1, 2\rangle, 3\rangle)) \\ &= \mathbf{fst}(\langle 1, 2\rangle) \\ &= 1 \end{aligned}$$

The Direction of Evaluation

$$3 + 4 = ?$$

The Direction of Evaluation

$$3 + 4 = 3 + (2 + 2)$$

The Direction of Evaluation

$$
\begin{aligned}
\mathbf{f}(2) &= -1 + \mathbf{f}(2) + 1 \\
&= -1 + \mathbf{f}(\mathbf{sqrt}(4)) + 1 \\
&= \ldots
\end{aligned}
$$

- For programming, we want an evaluation direction that produces ***values***

Expressions and Values

- Many possible ***expressions***

$$8$$

$$2 + 7 + \textbf{sqrt}(9)$$

$$\textbf{fst}$$

$$\langle 1, \textbf{fst}(\langle \text{empty}, \text{empty} \rangle) \rangle$$

- Certain expressions are designated as ***values***

$$8$$

$$\textbf{fst}$$

$$\langle 1, \text{empty} \rangle$$

Evaluation

- Define evaluation to ***reduce*** expressions to values

$$
\begin{aligned}
(2+7)+8 &\rightarrow 9+8 \\
&\rightarrow 17
\end{aligned}
$$

Evaluation with Higher-Order Functions

- Problem: creating new function values

$$\mathbf{f}(\mathbf{x}) \equiv \mathbf{x} + 1$$

$$\mathbf{g}(\mathbf{y}) \equiv \mathbf{y} + 2$$

$$\mathbf{compose}(\mathbf{a}, \mathbf{b}) \equiv \mathbf{...}$$

can't put **a**(**b**(**...**)) in place of **...**

Evaluation with Higher-Order Functions

- Problem: creating new function values

$$\mathbf{f}(\mathbf{x}) \equiv \mathbf{x} + 1$$

$$\mathbf{g}(\mathbf{y}) \equiv \mathbf{y} + 2$$

$$\mathbf{compose}(\mathbf{a}, \mathbf{b}) \equiv \mathbf{...}$$

$$\mathbf{compose}(\mathbf{f}, \mathbf{g}) \quad \rightarrow \quad \mathbf{...}$$

$$\rightarrow \quad \mathbf{h}$$

where

$$\mathbf{h}(\mathbf{z}) = \mathbf{f}(\mathbf{g}(\mathbf{z}))$$

Evaluation with Higher-Order Functions

- Redunction-friendly function notation:

Replace

$$\mathbf{f}(\mathbf{x}) \equiv \mathbf{x} + 1$$

with

$$\mathbf{f} \equiv (\lambda\, \mathbf{x}\, .\, \mathbf{x} + 1)$$

Evaluation with Higher-Order Functions

- Definition with $\equiv$ merely creates a shorthand

$$\mathbf{f} \equiv (\lambda\, \mathbf{x}\, .\, \mathbf{x} + 1)$$

- Apply functions through λ-application reduction

$$(\lambda\, \mathbf{x}\, .\, \mathbf{E})(\mathbf{v}) \rightarrow \mathbf{E} \text{ with } \mathbf{x} \text{ replaced by } \mathbf{v}$$

$$\begin{aligned} \mathbf{f}(10) &= (\lambda\, \mathbf{x}\, .\, \mathbf{x} + 1)(10) \\ &\rightarrow 10 + 1 \\ &\rightarrow 11 \end{aligned}$$

Evaluation with Higher-Order Functions

- Simple functions as values

$$\mathbf{mkadder} \equiv (\lambda\, \mathbf{m}\, .\, (\lambda\, \mathbf{n}\, .\, \mathbf{m} + \mathbf{n}))$$

$$\mathbf{add1} \equiv \mathbf{mkadder}(1)$$

$$\mathbf{add5} \equiv \mathbf{mkadder}(5)$$

$$\begin{aligned} \mathbf{add5} \quad &= \quad (\lambda\, \mathbf{m}\, .\, (\lambda\, \mathbf{n}\, .\, \mathbf{m} + \mathbf{n}))(5) \\ &\rightarrow \quad (\lambda\, \mathbf{n}\, .\, 5 + \mathbf{n}) \end{aligned}$$

Evaluation with Higher-Order Functions

- Simple functions as values

$$\mathbf{mkadder} \equiv (\lambda\, \mathbf{m}\,.\,(\lambda\, \mathbf{n}\,.\,\mathbf{m} + \mathbf{n}))$$

$$\mathbf{add1} \equiv \mathbf{mkadder}(1)$$

$$\mathbf{add5} \equiv \mathbf{mkadder}(5)$$

$$\begin{aligned} \mathbf{add5}(1) \quad &= \quad (\lambda\, \mathbf{m}\,.\,(\lambda\, \mathbf{n}\,.\,\mathbf{m} + \mathbf{n}))(5)(1) \\ &\to \quad (\lambda\, \mathbf{n}\,.\,5 + \mathbf{n})(1) \\ &\to \quad 5 + 1 \\ &\to \quad 6 \end{aligned}$$

Evaluation with Higher-Order Functions

- Returning to the definition of **compose**

$$\mathbf{f} \equiv (\lambda\, \mathbf{x}\ .\ \mathbf{x} + 1)$$

$$\mathbf{g} \equiv (\lambda\, \mathbf{y}\ .\ \mathbf{y} + 2)$$

$$\mathbf{compose} \equiv (\lambda\, (\mathbf{a}, \mathbf{b})\ .\ (\lambda\, \mathbf{z}\ .\ \mathbf{a}(\mathbf{b}(\mathbf{z}))))$$

$$\begin{aligned} \mathbf{compose}(\mathbf{f}, \mathbf{g}) \quad &= \quad (\lambda\, (\mathbf{a}, \mathbf{b})\ .\ (\lambda\, \mathbf{z}\ .\ \mathbf{a}(\mathbf{b}(\mathbf{z}))))(\mathbf{f},\mathbf{g}) \\ &\rightarrow \quad (\lambda\, \mathbf{z}\ .\ \mathbf{f}(\mathbf{g}(\mathbf{z}))) \end{aligned}$$

Outline

- Programming with Functions
- Defining a Functional Language
- Type-Checking a Functional Program
- Implementing a Functional Language

Definining a Functional Language

Steps to defining a language:

- Define the syntax for expressions
- Designate certain expressions as values
- Define the reduction rules on expressions

Syntax: Expressions

$$
\begin{array}{lcl}
\mathbf{M} & = & \ulcorner \mathbf{n} \urcorner \\
 & | & \mathbf{M} - \mathbf{M} \\
 & | & \mathbf{M} \bullet \mathbf{M} \\
 & | & \mathtt{\textbf{if0}}\ \mathbf{M}\ \mathtt{\textbf{then}}\ \mathbf{M}\ \mathtt{\textbf{else}}\ \mathbf{M} \\
 & | & \lambda\, \mathbf{x} \,.\, \mathbf{M} \\
 & | & \mathbf{M}\ \mathbf{M} \\
\mathbf{n} & = & \text{an integer} \\
\mathbf{x} & = & \text{a variable}
\end{array}
$$

$\ulcorner 5 \urcorner$ represents 5

Syntax: Expressions

$$
\begin{aligned}
\mathbf{M} \quad &= \quad \lceil \mathbf{n} \rceil \\
&\;|\quad \mathbf{M} - \mathbf{M} \\
&\;|\quad \mathbf{M} \bullet \mathbf{M} \\
&\;|\quad \texttt{if0}\ \mathbf{M}\ \texttt{then}\ \mathbf{M}\ \texttt{else}\ \mathbf{M} \\
&\;|\quad \lambda\, \mathbf{x}\,.\,\mathbf{M} \\
&\;|\quad \mathbf{M}\ \mathbf{M} \\
\mathbf{n} \quad &= \quad \text{an integer} \\
\mathbf{x} \quad &= \quad \text{a variable}
\end{aligned}
$$

$\lceil 5 \rceil - \lceil 3 \rceil$ represents the subtraction of 3 from 5

Syntax: Expressions

$$
\begin{array}{lcl}
\mathbf{M} & = & \lceil \mathbf{n} \rceil \\
 & | & \mathbf{M} - \mathbf{M} \\
 & | & \mathbf{M} \bullet \mathbf{M} \\
 & | & \texttt{if0}\ \mathbf{M}\ \texttt{then}\ \mathbf{M}\ \texttt{else}\ \mathbf{M} \\
 & | & \lambda\, \mathbf{x} . \mathbf{M} \\
 & | & \mathbf{M}\ \mathbf{M} \\
\mathbf{n} & = & \text{an integer} \\
\mathbf{x} & = & \text{a variable}
\end{array}
$$

$\lambda\, \mathbf{x} . \mathbf{x}$ represents the identity function

Syntax: Expressions

$$
\begin{array}{lcl}
\mathbf{M} & = & \lceil \mathbf{n} \rceil \\
 & | & \mathbf{M} - \mathbf{M} \\
 & | & \mathbf{M} \bullet \mathbf{M} \\
 & | & \mathtt{if0}\ \mathbf{M}\ \mathtt{then}\ \mathbf{M}\ \mathtt{else}\ \mathbf{M} \\
 & | & \lambda\, \mathbf{x} \,.\, \mathbf{M} \\
 & | & \mathbf{M}\ \mathbf{M} \\
\mathbf{n} & = & \text{an integer} \\
\mathbf{x} & = & \text{a variable}
\end{array}
$$

$(\lambda\, \mathbf{x} \,.\, \mathbf{x})(\lceil 5 \rceil)$ represents applying the identity function to 5

Syntax: Values

$$\mathbf{V} \;=\; \lceil \mathbf{n} \rceil \;\mid\; \lambda\, \mathbf{x}\,.\,\mathbf{M}$$

$\lceil 5 \rceil$	a value
$\lambda\, \mathbf{x}\,.\,\mathbf{x}$	a value
$\lceil 5 \rceil - \lceil 3 \rceil$	**not** a value
$(\lambda\, \mathbf{x}\,.\,\mathbf{x})(\lceil 5 \rceil)$	**not** a value
$\lambda\, \mathbf{y}\,.\,((\lambda\, \mathbf{x}.\,\mathbf{x})(\mathbf{y}))$	a value

Reductions

$$\lceil \mathbf{n}_1 \rceil - \lceil \mathbf{n}_2 \rceil \quad \rightarrow \quad \lceil \mathbf{n}_1 - \mathbf{n}_2 \rceil$$

$$\lceil \mathbf{n}_1 \rceil \bullet \lceil \mathbf{n}_2 \rceil \quad \rightarrow \quad \lceil \mathbf{n}_1 \bullet \mathbf{n}_2 \rceil$$

$$\texttt{if0} \lceil 0 \rceil\ \textbf{then}\ \mathbf{M}_1\ \textbf{else}\ \mathbf{M}_2 \quad \rightarrow \quad \mathbf{M}_1$$

$$\texttt{if0} \lceil \mathbf{n} \rceil\ \textbf{then}\ \mathbf{M}_1\ \textbf{else}\ \mathbf{M}_2 \quad \rightarrow \quad \mathbf{M}_2$$

if $\mathbf{n} \neq 0$

$$(\lambda\, \mathbf{x} \,.\, \mathbf{M})(\mathbf{V}) \quad \rightarrow \quad \mathbf{M}$$

with **V** in place of **x**

$$\lceil 5 \rceil - \lceil 3 \rceil \rightarrow \lceil 2 \rceil$$

Reductions

$$\ulcorner \mathbf{n}_1 \urcorner - \ulcorner \mathbf{n}_2 \urcorner \quad \to \quad \ulcorner \mathbf{n}_1 - \mathbf{n}_2 \urcorner$$

$$\ulcorner \mathbf{n}_1 \urcorner \bullet \ulcorner \mathbf{n}_2 \urcorner \quad \to \quad \ulcorner \mathbf{n}_1 \bullet \mathbf{n}_2 \urcorner$$

$$\texttt{if0}\ \ulcorner 0 \urcorner\ \texttt{then}\ \mathbf{M}_1\ \texttt{else}\ \mathbf{M}_2 \quad \to \quad \mathbf{M}_1$$

$$\texttt{if0}\ \ulcorner \mathbf{n} \urcorner\ \texttt{then}\ \mathbf{M}_1\ \texttt{else}\ \mathbf{M}_2 \quad \to \quad \mathbf{M}_2$$

if $\mathbf{n} \neq 0$

$$(\lambda\, \mathbf{x}\, .\, \mathbf{M})(\mathbf{V}) \quad \to \quad \mathbf{M}$$

with **V** in place of **x**

$$\texttt{if0}\ \ulcorner 0 \urcorner\ \texttt{then}\ \ulcorner 5 \urcorner\ \texttt{else}\ (\lambda\, \mathbf{x}\, .\, \mathbf{x}) \to \ulcorner 5 \urcorner$$

Reductions

$$\lceil \mathbf{n}_1 \rceil - \lceil \mathbf{n}_2 \rceil \quad \rightarrow \quad \lceil \mathbf{n}_1 - \mathbf{n}_2 \rceil$$

$$\lceil \mathbf{n}_1 \rceil \bullet \lceil \mathbf{n}_2 \rceil \quad \rightarrow \quad \lceil \mathbf{n}_1 \bullet \mathbf{n}_2 \rceil$$

$$\texttt{if0} \lceil 0 \rceil\, \texttt{then}\ \mathbf{M}_1\ \texttt{else}\ \mathbf{M}_2 \quad \rightarrow \quad \mathbf{M}_1$$

$$\texttt{if0} \lceil \mathbf{n} \rceil\, \texttt{then}\ \mathbf{M}_1\ \texttt{else}\ \mathbf{M}_2 \quad \rightarrow \quad \mathbf{M}_2$$

if $\mathbf{n} \neq 0$

$$(\lambda\, \mathbf{x} \,.\, \mathbf{M})(\mathbf{V}) \quad \rightarrow \quad \mathbf{M}$$

with **V** in place of **x**

$$\texttt{if0} \lceil 1 \rceil\, \texttt{then} \lceil 5 \rceil\, \texttt{else}\ (\lambda\, \mathbf{x} \,.\, \mathbf{x}) \rightarrow (\lambda\, \mathbf{x} \,.\, \mathbf{x})$$

Reductions

$$\ulcorner \mathbf{n}_1 \urcorner - \ulcorner \mathbf{n}_2 \urcorner \quad \rightarrow \quad \ulcorner \mathbf{n}_1 - \mathbf{n}_2 \urcorner$$

$$\ulcorner \mathbf{n}_1 \urcorner \bullet \ulcorner \mathbf{n}_2 \urcorner \quad \rightarrow \quad \ulcorner \mathbf{n}_1 \bullet \mathbf{n}_2 \urcorner$$

$$\mathtt{if0}\ \ulcorner 0 \urcorner\ \mathtt{then}\ \mathbf{M}_1\ \mathtt{else}\ \mathbf{M}_2 \quad \rightarrow \quad \mathbf{M}_1$$

$$\mathtt{if0}\ \ulcorner \mathbf{n} \urcorner\ \mathtt{then}\ \mathbf{M}_1\ \mathtt{else}\ \mathbf{M}_2 \quad \rightarrow \quad \mathbf{M}_2$$

if $\mathbf{n} \neq 0$

$$(\lambda\, \mathbf{x}\, .\, \mathbf{M})(\mathbf{V}) \quad \rightarrow \quad \mathbf{M}$$

with **V** in place of **x**

$$(\lambda\, \mathbf{x}\, .\, \mathbf{x} \bullet \ulcorner 10 \urcorner)(\ulcorner 8 \urcorner) \rightarrow \ulcorner 8 \urcorner \bullet \ulcorner 10 \urcorner$$

Reductions in Context

$$\mathbf{M}_1 - \mathbf{M}_2 \;\rightarrow\; \mathbf{M}_1' - \mathbf{M}_2$$
$$\text{where } \mathbf{M}_1 \rightarrow \mathbf{M}_1'$$
$$\mathbf{V} - \mathbf{M}_2 \;\rightarrow\; \mathbf{V} - \mathbf{M}_2'$$
$$\text{where } \mathbf{M}_2 \rightarrow \mathbf{M}_2'$$

$$\mathbf{M}_1 \bullet \mathbf{M}_2 \;\rightarrow\; \mathbf{M}_1' \bullet \mathbf{M}_2$$

...

$$(\lceil 5 \rceil \bullet \lceil 2 \rceil) - (\lceil 3 \rceil \bullet \lceil 4 \rceil) \rightarrow \lceil 10 \rceil - (\lceil 3 \rceil \bullet \lceil 4 \rceil)$$

Reductions in Context

$$\mathbf{M}_1 - \mathbf{M}_2 \quad \longrightarrow \quad \mathbf{M}_1' - \mathbf{M}_2$$
$$\text{where } \mathbf{M}_1 \longrightarrow \mathbf{M}_1'$$
$$\mathbf{V} - \mathbf{M}_2 \quad \longrightarrow \quad \mathbf{V} - \mathbf{M}_2'$$
$$\text{where } \mathbf{M}_2 \longrightarrow \mathbf{M}_2'$$

$$\mathbf{M}_1 \bullet \mathbf{M}_2 \quad \longrightarrow \quad \mathbf{M}_1' \bullet \mathbf{M}_2$$

...

$$\lceil 10 \rceil - (\lceil 3 \rceil \bullet \lceil 4 \rceil) \longrightarrow \lceil 10 \rceil - \lceil 12 \rceil$$

Reductions in Context

$$\texttt{if0}\ \mathbf{M}\ \texttt{then}\ \mathbf{M}_1\ \texttt{else}\ \mathbf{M}_2 \ \rightarrow\ \texttt{if0}\ \mathbf{M}'\ \texttt{then}\ \mathbf{M}_1\ \texttt{else}\ \mathbf{M}_2$$

where $\mathbf{M} \rightarrow \mathbf{M}'$

$$\mathbf{M}_1\ \mathbf{M}_2 \ \rightarrow\ \mathbf{M}_1'\ \mathbf{M}_2$$

where $\mathbf{M}_1 \rightarrow \mathbf{M}_1'$

$$\mathbf{V}\ \mathbf{M}_2 \ \rightarrow\ \mathbf{V}\ \mathbf{M}_2'$$

where $\mathbf{M}_2 \rightarrow \mathbf{M}_2'$

$$(\lambda\,\mathbf{x}\,.\,\mathbf{x})(\lceil 2 \rceil \bullet \lceil 2 \rceil) \rightarrow (\lambda\,\mathbf{x}\,.\,\mathbf{x})(\lceil 4 \rceil)$$

Reductions in Context

$\texttt{if0}\ \mathbf{M}\ \texttt{then}\ \mathbf{M}_1\ \texttt{else}\ \mathbf{M}_2$	$\rightarrow$	$\texttt{if0}\ \mathbf{M}'\ \texttt{then}\ \mathbf{M}_1\ \texttt{else}\ \mathbf{M}_2$ where $\mathbf{M} \rightarrow \mathbf{M}'$
$\mathbf{M}_1\ \mathbf{M}_2$	$\rightarrow$	$\mathbf{M}_1'\ \mathbf{M}_2$ where $\mathbf{M}_1 \rightarrow \mathbf{M}_1'$
$\mathbf{V}\ \mathbf{M}_2$	$\rightarrow$	$\mathbf{V}\ \mathbf{M}_2'$ where $\mathbf{M}_2 \rightarrow \mathbf{M}_2'$

$$((\lambda\, \mathbf{x}\,.\,\mathbf{x})(\lambda\, \mathbf{y}\,.\,\mathbf{y}))(\lceil 2 \rceil \bullet \lceil 2 \rceil) \rightarrow (\lambda\, \mathbf{y}\,.\,\mathbf{y})(\lceil 2 \rceil \bullet \lceil 2 \rceil)$$

Extended Example: Factorial

$$\mathbf{fac} \equiv \lambda \mathtt{n.if0\ n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else\ n} \bullet \mathbf{fac}(\mathbf{n} - \lceil 1 \rceil)$$

Illegal: **fac** isn't merely a shorthand because it mentions itself

$$\mathbf{mkfac} \equiv \lambda\, \mathtt{f.}\, \lambda\, \mathtt{n.if0\ n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else\ n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$
$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\, \mathtt{f}.\, \lambda\, \mathtt{n}.\, \mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$

$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$\mathbf{fac}(\lceil 0 \rceil)$$
$$=$$
$$(\mathbf{mkfac}(\mathbf{mkfac}))(\lceil 0 \rceil)$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\,\mathtt{f}\,.\,\lambda\,\mathtt{n}\,.\,\mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$
$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$(\mathbf{mkfac}(\mathbf{mkfac}))(\lceil 0 \rceil)$$
$$\rightarrow$$
$$(\lambda\,\mathtt{n}\,.\,\mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\mathbf{n} - \lceil 1 \rceil))(\lceil 0 \rceil)$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\, \mathbf{f}.\, \lambda\, \mathbf{n}.\, \mathtt{if0}\ \mathbf{n}\ \mathtt{then}\ \lceil 1 \rceil\ \mathtt{else}\ \mathbf{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$

$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$(\lambda\, \mathbf{n}.\, \mathtt{if0}\ \mathbf{n}\ \mathtt{then}\ \lceil 1 \rceil\ \mathtt{else}\ \mathbf{n} \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\mathbf{n} - \lceil 1 \rceil))(\lceil 0 \rceil)$$

$$\rightarrow$$

$$\mathtt{if0}\,\lceil 0 \rceil\ \mathtt{then}\ \lceil 1 \rceil\ \mathtt{else}\ \lceil 0 \rceil \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\lceil 0 \rceil - \lceil 1 \rceil)$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\, \mathtt{f}\,.\, \lambda\, \mathtt{n}\,.\, \mathtt{if0}\ \mathtt{n}$$
$$\qquad \mathtt{then}\ \lceil 1 \rceil$$
$$\qquad \mathtt{else}\ \mathbf{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$

$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$\mathtt{if0}\lceil 0 \rceil$$
$$\ \mathtt{then}\ \lceil 1 \rceil$$
$$\ \mathtt{else}\ \lceil 0 \rceil \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\lceil 0 \rceil - \lceil 1 \rceil)$$
$$\rightarrow$$
$$\lceil 1 \rceil$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\, \mathtt{f}\,.\, \lambda\, \mathtt{n}\,.\, \mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \ulcorner 1 \urcorner$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \ulcorner 1 \urcorner)$$
$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$\mathbf{fac}(\ulcorner 2 \urcorner)$$
$$=$$
$$(\mathbf{mkfac}(\mathbf{mkfac}))(\ulcorner 2 \urcorner)$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\,\mathtt{f}.\,\lambda\,\mathtt{n}.\,\mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$

$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$(\mathbf{mkfac}(\mathbf{mkfac}))(\lceil 2 \rceil)$$
$$\rightarrow$$
$$(\lambda\,\mathtt{n}.\,\mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\mathbf{n} - \lceil 1 \rceil))(\lceil 2 \rceil)$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\,\mathtt{f}.\,\lambda\,\mathtt{n}.\,\mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$

$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$(\lambda\,\mathtt{n}.\,\mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\mathbf{n} - \lceil 1 \rceil))(\lceil 2 \rceil)$$

$$\rightarrow$$

$$\mathtt{if0}\,\lceil 2 \rceil$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \lceil 2 \rceil \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\lceil 2 \rceil - \lceil 1 \rceil)$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\, \mathtt{f.}\, \lambda\, \mathtt{n.}\ \mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \ulcorner 1 \urcorner$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \ulcorner 1 \urcorner)$$

$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$\mathtt{if0} \ulcorner 2 \urcorner$$
$$\mathtt{then}\ \ulcorner 1 \urcorner$$
$$\mathtt{else}\ \ulcorner 2 \urcorner \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\ulcorner 2 \urcorner - \ulcorner 1 \urcorner)$$

$$\rightarrow$$

$$\ulcorner 2 \urcorner \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\ulcorner 2 \urcorner - \ulcorner 1 \urcorner)$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\, \mathtt{f}\,.\, \lambda\, \mathtt{n}\,.\, \mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$

$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$\lceil 2 \rceil \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\lceil 2 \rceil - \lceil 1 \rceil)$$

$$\rightarrow$$

$$\lceil 2 \rceil \bullet (\lambda\, \mathtt{n}\,.\, \mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\mathbf{n} - \lceil 1 \rceil))(\lceil 2 \rceil - \lceil 1 \rceil)$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\, \mathtt{f}.\, \lambda\, \mathtt{n}.\, \mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$
$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$\lceil 2 \rceil \bullet (\lambda\, \mathtt{n}.\, \mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\mathbf{n} - \lceil 1 \rceil))(\lceil 2 \rceil - \lceil 1 \rceil)$$
$$\to$$
$$\lceil 2 \rceil \bullet (\lambda\, \mathtt{n}.\, \mathtt{if0}\ \mathtt{n}$$
$$\mathtt{then}\ \lceil 1 \rceil$$
$$\mathtt{else}\ \mathtt{n} \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\mathbf{n} - \lceil 1 \rceil))(\lceil 1 \rceil)$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\, \mathbf{f}\,.\, \lambda\, \mathbf{n}\,.\, \mathbf{if0}\ \mathbf{n}$$
$$\mathbf{then}\ \lceil 1 \rceil$$
$$\mathbf{else}\ \mathbf{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$
$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$\lceil 2 \rceil \bullet (\lambda\, \mathbf{n}\,.\, \mathbf{if0}\ \mathbf{n}$$
$$\mathbf{then}\ \lceil 1 \rceil$$
$$\mathbf{else}\ \mathbf{n} \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\mathbf{n} - \lceil 1 \rceil))(\lceil 1 \rceil)$$
$$\rightarrow$$
$$\lceil 2 \rceil \bullet \mathbf{if0} \lceil 1 \rceil$$
$$\mathbf{then}\ \lceil 1 \rceil$$
$$\mathbf{else}\ \lceil 1 \rceil \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\lceil 1 \rceil - \lceil 1 \rceil)$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\,\mathtt{f}.\,\lambda\,\mathtt{n}.\,\mathtt{if0}\ \mathtt{n}$$
$$\qquad\qquad \mathtt{then}\ \lceil 1 \rceil$$
$$\qquad\qquad \mathtt{else}\ \mathtt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$

$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$\lceil 2 \rceil \bullet \mathtt{if0}\,\lceil 1 \rceil$$
$$\qquad \mathtt{then}\ \lceil 1 \rceil$$
$$\qquad \mathtt{else}\ \lceil 1 \rceil \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\lceil 1 \rceil - \lceil 1 \rceil)$$

$$\rightarrow$$

$$\lceil 2 \rceil \bullet (\lceil 1 \rceil \bullet (\mathbf{mkfac}(\mathbf{mkfac}))(\lceil 1 \rceil - \lceil 1 \rceil))$$

Extended Example: Factorial

$$\textbf{mkfac} \equiv \lambda\, \mathbf{f}.\, \lambda\, \mathbf{n}.\, \texttt{if0}\ \mathbf{n}$$
$$\texttt{then}\ \lceil 1 \rceil$$
$$\texttt{else}\ \mathbf{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$

$$\textbf{fac} \equiv \textbf{mkfac}(\textbf{mkfac})$$

$$\lceil 2 \rceil \bullet (\lceil 1 \rceil \bullet (\textbf{mkfac}(\textbf{mkfac}))(\lceil 1 \rceil - \lceil 1 \rceil))$$

$$\rightarrow$$

$$\lceil 2 \rceil \bullet (\lceil 1 \rceil \bullet (\lambda\, \mathbf{n}.\, \texttt{if0}\ \mathbf{n}$$
$$\texttt{then}\ \lceil 1 \rceil$$
$$\texttt{else}\ \ldots\ )(\lceil 1 \rceil - \lceil 1 \rceil))$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\, \mathtt{f}\,.\, \lambda\, \mathtt{n}\,.\, \mathtt{if0}\ \mathtt{n}\ \mathtt{then}\ \ulcorner 1 \urcorner\ \mathtt{else}\ \mathtt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \ulcorner 1 \urcorner)$$

$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$\ulcorner 2 \urcorner \bullet (\ulcorner 1 \urcorner \bullet (\lambda\, \mathtt{n}\,.\, \mathtt{if0}\ \mathtt{n}\ \mathtt{then}\ \ulcorner 1 \urcorner\ \mathtt{else}\ \ldots\ )(\ulcorner 1 \urcorner - \ulcorner 1 \urcorner))$$

$$\rightarrow$$

$$\ulcorner 2 \urcorner \bullet (\ulcorner 1 \urcorner \bullet (\lambda\, \mathtt{n}\,.\, \mathtt{if0}\ \mathtt{n}\ \mathtt{then}\ \ulcorner 1 \urcorner\ \mathtt{else}\ \ldots\ )(\ulcorner 0 \urcorner))$$

Extended Example: Factorial

$$\textbf{mkfac} \equiv \lambda\, \mathbf{f}\, .\, \lambda\, \mathbf{n}\, .\, \mathtt{if0}\ \mathbf{n}$$
$$\quad \mathtt{then}\ \lceil 1 \rceil$$
$$\quad \mathtt{else}\ \mathbf{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$
$$\textbf{fac} \equiv \textbf{mkfac}(\textbf{mkfac})$$

$$\lceil 2 \rceil \bullet (\lceil 1 \rceil \bullet (\lambda\, \mathbf{n}\, .\, \mathtt{if0}\ \mathbf{n}$$
$$\quad \mathtt{then}\ \lceil 1 \rceil$$
$$\quad \mathtt{else}\ \ldots\ )(\lceil 0 \rceil))$$
$$\rightarrow$$
$$\lceil 2 \rceil \bullet (\lceil 1 \rceil \bullet \mathtt{if0}\, \lceil 0 \rceil$$
$$\quad \mathtt{then}\ \lceil 1 \rceil$$
$$\quad \mathtt{else}\ \ldots\ )$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\, \mathtt{f}\,.\,\lambda\, \mathtt{n}\,.\,\mathtt{if0}\ \mathtt{n}$$
$$\quad \mathtt{then}\ \lceil 1 \rceil$$
$$\quad \mathtt{else}\ \mathtt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$

$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$\lceil 2 \rceil \bullet (\lceil 1 \rceil \bullet \mathtt{if0}\,\lceil 0 \rceil$$
$$\quad \mathtt{then}\ \lceil 1 \rceil$$
$$\quad \mathtt{else}\ \ldots\ )$$
$$\rightarrow$$
$$\lceil 2 \rceil \bullet (\lceil 1 \rceil \bullet \lceil 1 \rceil)$$

Extended Example: Factorial

$$\mathbf{mkfac} \equiv \lambda\, \texttt{f}\,.\,\lambda\, \texttt{n}\,.\, \texttt{if0}\ \texttt{n}$$
$$\texttt{then}\ \lceil 1 \rceil$$
$$\texttt{else}\ \texttt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$

$$\mathbf{fac} \equiv \mathbf{mkfac}(\mathbf{mkfac})$$

$$\lceil 2 \rceil \bullet (\lceil 1 \rceil \bullet \lceil 1 \rceil)$$
$$\rightarrow$$
$$\lceil 2 \rceil \bullet \lceil 1 \rceil$$

Extended Example: Factorial

$$\textbf{mkfac} \equiv \lambda\, \texttt{f}\,.\, \lambda\, \texttt{n}\,.\, \texttt{if0 n}$$
$$\texttt{then}\ \lceil 1 \rceil$$
$$\texttt{else}\ \texttt{n} \bullet (\mathbf{f}(\mathbf{f}))(\mathbf{n} - \lceil 1 \rceil)$$
$$\textbf{fac} \equiv \textbf{mkfac}(\textbf{mkfac})$$

$$\lceil 2 \rceil \bullet \lceil 1 \rceil$$
$$\rightarrow$$
$$\lceil 2 \rceil$$

Outline

- Programming with Functions
- Defining a Functional Language
- Type-Checking a Functional Program
- Implementing a Functional Language

Well-Formedness

Quiz: What is the value of the following expression?

$$\lambda\lceil 10 \rceil$$

Answer: Trick question. It’s not an expression.

Safety and Types

- The following is a syntactically well-formed expression

$$(\lambda\, \mathbf{x}\, .\, \mathbf{x}) - \lceil 10 \rceil$$

- It is not a value...
- ... but no reduction rule applies; the expression is ***stuck***
- The language is ***unsafe***

Safety and Types

One way to safety:

$$(\lambda\, \mathbf{x}\, .\, \mathbf{x}) - \lceil 10 \rceil \rightarrow \text{error}_{\text{minus}}$$

Safety

- For any expression, a ***safe language*** produces a well-defined result (possibly an error) or reduces forever

Safety and Types

Another way to safety:

Reject $(\lambda\, \mathbf{x}\, .\, \mathbf{x}) - \lceil 10 \rceil$ as an expression

Types

- A ***type system*** defines a restriction on well-formedness, in addition the syntax rules
- A typed, well-formed expression never gets stuck, and *never signals certain errors*, such as $\text{error}_{\text{minus}}$

Type Rules

$$\lceil 5 \rceil : \texttt{int}$$

$$\lceil 6 \rceil - \lceil 1 \rceil : \texttt{int}$$

$$(\lambda\, \texttt{x}\,.\,\texttt{x})(\lceil 8 \rceil) : \texttt{int}$$

$$(\lambda\, \texttt{x}\,.\,\texttt{x}) - \lceil 10 \rceil : \textit{no type}$$

$$\texttt{if0}\, \lceil 0 \rceil\, \texttt{then}\, \lceil 1 \rceil\, \texttt{else}\, (\lambda\, \texttt{x}\,.\,\texttt{x}) : \textit{no type}$$

Type Rules

- arithmetic expressions produce integers

$$\lceil \mathbf{n} \rceil : \texttt{int}$$

$$\frac{\mathbf{M}_1 : \texttt{int} \qquad \mathbf{M}_2 : \texttt{int}}{\mathbf{M}_1 - \mathbf{M}_2 : \texttt{int}}$$

$$\frac{\lceil 5 \rceil : \texttt{int} \qquad \dfrac{\lceil 3 \rceil : \texttt{int} \qquad \lceil 1 \rceil : \texttt{int}}{\lceil 3 \rceil - \lceil 1 \rceil : \texttt{int}}}{\lceil 5 \rceil - (\lceil 3 \rceil - \lceil 1 \rceil) : \texttt{int}}$$

Type Rules

- **if0**: assume both branches have the same type

$$\frac{M : \mathtt{int} \qquad M_1 : T \qquad M_2 : T}{\mathtt{if0}\ M\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 : T}$$

$$\frac{\lceil 0 \rceil : \mathtt{int} \qquad \dfrac{\lceil 2 \rceil : \mathtt{int} \qquad \lceil 3 \rceil : \mathtt{int}}{\lceil 2 \rceil + \lceil 3 \rceil : \mathtt{int}} \qquad \lceil 1 \rceil : \mathtt{int}}{\mathtt{if0}\ \lceil 0 \rceil\ \mathtt{then}\ (\lceil 2 \rceil + \lceil 3 \rceil)\ \mathtt{else}\ \lceil 1 \rceil : \mathtt{int}}$$

Type Rules

- What about variables?

x

shouldn't have a type

λ **x.x**

x needs a type, used towards the expression type

- Accumulate variable context in an environment, Γ

$$\Gamma \vdash \mathbf{x} : \mathbf{T} \qquad \text{if } \Gamma(\mathbf{x}) = \mathbf{T}$$

- - - - - - - - - - - - - - - - -

$$\{\mathbf{x}{=}\mathbf{int}\} \vdash \mathbf{x} : \mathbf{int}$$

Type Rules

- Fix up old rules

$$\Gamma \vdash \lceil \mathbf{n} \rceil : \mathbf{int}$$

$$\frac{\Gamma \vdash \mathbf{M}_1 : \mathbf{int} \qquad \Gamma \vdash \mathbf{M}_2 : \mathbf{int}}{\Gamma \vdash \mathbf{M}_1 - \mathbf{M}_2 : \mathbf{int}}$$

$$\frac{\Gamma \vdash \mathbf{M} : \mathbf{int} \qquad \Gamma \vdash \mathbf{M}_1 : \mathbf{T} \qquad \Gamma \vdash \mathbf{M}_2 : \mathbf{T}}{\Gamma \vdash \mathbf{if0}\ \mathbf{M}\ \mathbf{then}\ \mathbf{M}_1\ \mathbf{else}\ \mathbf{M}_2 : \mathbf{T}}$$

$$\frac{\{\mathbf{x}=\mathbf{int}\} \vdash \lceil 9 \rceil : \mathbf{int} \qquad \{\mathbf{x}=\mathbf{int}\} \vdash \mathbf{x} : \mathbf{int}}{\{\mathbf{x}=\mathbf{int}\} \vdash \lceil 9 \rceil - \mathbf{x} : \mathbf{int}}$$

Type Rules

- Function type: $\mathbf{T}_1 \to \mathbf{T}_2$

$$\frac{\Gamma\{\mathbf{x}{=}\mathbf{T}'\} \vdash \mathbf{M} : \mathbf{T}}{\Gamma \vdash (\lambda\, \mathbf{x} \,.\, \mathbf{M}) : \mathbf{T}' \to \mathbf{T}}$$

$$\frac{\Gamma \vdash \mathbf{M}_1 : \mathbf{T}' \to \mathbf{T} \qquad \Gamma \vdash \mathbf{M}_2 : \mathbf{T}'}{\Gamma \vdash (\mathbf{M}_1\ \mathbf{M}_2) : \mathbf{T}}$$

$$\frac{\dfrac{\{\mathbf{x}{=}\mathtt{int}\} \vdash \mathbf{x} : \mathtt{int}}{\{\} \vdash (\lambda\, \mathbf{x} \,.\, \mathbf{x}) : \mathtt{int} \to \mathtt{int}} \qquad \lceil 5 \rceil : \mathtt{int}}{\{\} \vdash (\lambda\, \mathbf{x} \,.\, \mathbf{x})(\lceil 5 \rceil) : \mathtt{int}}$$

Type Rules

- One more example (abbreviate **`int`** with **`i`**)

$$
\frac{
\dfrac{
\dfrac{\{\mathbf{f}{=}\mathbf{i}\to\mathbf{i}\} \vdash \mathbf{f} : \mathbf{i}\to\mathbf{i} \qquad \{\mathbf{f}{=}\mathbf{i}\to\mathbf{i}\} \vdash 5 : \mathbf{i}}
{\{\mathbf{f}{=}\mathbf{i}\to\mathbf{i}\} \vdash \mathbf{f}\lceil 5 \rceil : \mathbf{i}}
}{
\{\} \vdash (\lambda\, \mathbf{f}\,.\, \mathbf{f}\lceil 5 \rceil) : (\mathbf{i}\to\mathbf{i})\to\mathbf{i}
}
\qquad
\dfrac{
\dfrac{\{\mathbf{y}{=}\mathbf{i}\} \vdash \mathbf{y} : \mathbf{i} \qquad \{\mathbf{y}{=}\mathbf{i}\} \vdash \lceil 1 \rceil : \mathbf{i}}
{\{\mathbf{y}{=}\mathbf{i}\} \vdash \mathbf{y} - \lceil 1 \rceil : \mathbf{i}}
}{
\{\} \vdash (\lambda\, \mathbf{y}\,.\, \mathbf{y} - \lceil 1 \rceil) : \mathbf{i}\to\mathbf{i}
}
}{
\{\} \vdash (\lambda\, \mathbf{f}\,.\, \mathbf{f}\lceil 5 \rceil)(\lambda\, \mathbf{y}\,.\, \mathbf{y} - \lceil 1 \rceil) : \mathbf{i}
}
$$

Outline

- Programming with Functions
- Defining a Functional Language
- Type-Checking a Functional Program
- ➡ Implementing a Functional Language

Implementing a Functional Language

- So far, the language is defined in terms of rewriting rules
- But real machines do not provide a "rewrite" opcode
- Implement an interpreter to run on a realistic machine
    - Use ML notation to describe the interpreter
    - Start with a ***meta-circular*** interpreter, then convert to machine code — in 10 easy steps!

Abstract Syntax

- Ignore parsing

```
type xpr = Value of xval
           | Minus of xpr * xpr
           | Times of xpr * xpr
           | Lam of xvar * xpr
           | Var of xvar
           | App of xpr * xpr
           | IfZero of xpr * xpr * xpr
type xval = Num of int
            | Fun of (xval → xval)
```

λ **x . (x** $- \lceil 5 \rceil$) $\overset{\text{parse}}{\Longrightarrow}$ **Lam("x",Minus(Var("x"),Value(Num(5))))**

Step 1

- Instead of rewriting the source syntax step-by-step, use ML's recurson to evaluate sub-expressions.

$$eval \lceil 2 \rceil - \lceil 1 \rceil \quad = \quad eval \lceil 2 \rceil \ \text{-} \ eval \lceil 1 \rceil$$

Step 1

```
let rec eval = function
   Value(v) → v
 | Minus(m1,m2) → let Num(n1) = eval(m1)
                      and Num(n2) = eval(m2)
                       in Num(n1 - n2)
 | Times(m1,m2) → let Num(n1) = eval(m1)
                      and Num(n2) = eval(m2)
                       in Num(n1 * n2)
 | Lam(var,m) → Fun(fun v → eval(replace (var, v) m))
 | App(m1,m2) → let Fun(f) = eval(m1)
                  in f(eval(m2))
 | IfZero(m1,m2,m3) → let Num(n) = eval(m1)
                       in eval(if (n=0)
                                then m2
                                else m3)
```

Step 2

- Use an environment for function bodies instead of replacement

Old way:

$$(\lambda\, \mathbf{x}\, .\, \mathbf{x} - \lceil 1 \rceil)(\lceil 10 \rceil) \rightarrow \lceil 10 \rceil - \lceil 1 \rceil$$

New way:

$$\{\mathbf{y}{=}7\}\ (\lambda\, \mathbf{x}\, .\, \mathbf{x} - \lceil 1 \rceil)(\lceil 10 \rceil) \rightarrow \{\mathbf{y}{=}7, \mathbf{x}{=}10\}\ \mathbf{x} - \lceil 1 \rceil$$

Step 2

```
let rec eval = function
    (Const(v), e) → Num(v)
  | (Minus(m1,m2), e) → let Num(n1) = eval(m1, e)
                         and Num(n2) = eval(m2, e)
                          in Num(n1 - n2)
  | (Times(m1,m2), e) → let Num(n1) = eval(m1, e)
                         and Num(n2) = eval(m2, e)
                          in Num(n1 * n2)
  | (Lam(var,m), e) → Fun(fun v →
                            eval(m, Extend(var,v,e)))
  | (App(m1,m2), e) → let Fun(f) = eval(m1, e)
                       in f(eval(m2, e))
  | (IfZero(m1,m2,m3), e) → let Num(n) = eval(m1, e)
                                in eval((if (n==0)
                                            then m2
                                            else m3),
                                        e)
  | (Var(var), e) → lookup(var, e)
```

Step 3

- Pre-compute variable locations in the environment
- Introduce a "bytecode" compiler for pre-computing

$$\lambda\, \mathbf{x}\,.\,(\lambda\, \mathbf{y}\,.\,(\mathbf{x} \bullet \mathbf{y}))$$

$$\overset{\text{compile}}{\Longrightarrow}$$

$$\lambda\;\;.\,(\lambda\;\;.\,(@2 \bullet @1))$$

Step 3

```
let rec comp = function
    (Const(v), e) → CConst(v)
  | (Minus(m1,m2), e) → CMinus(comp(m1, e),comp(m2, e))
  | (Times(m1,m2), e) → CTimes(comp(m1, e),comp(m2, e))
  | (Lam(var,m), e) → CLam(comp(m, CExtend(var,e)))
  | (App(m1,m2), e) → CApp(comp(m1, e),comp(m2, e))
  | (IfZero(m1,m2,m3), e) → CIfZero(comp(m1, e),
                                     comp(m2, e),
                                     comp(m3, e))
  | (Var(var), e) → CVar(offset(var, e))
```

Step 3

```
let rec eval = function
    (CConst(v), e) → Num(v)
  | (CMinus(m1,m2), e) → let Num(n1) = eval(m1, e)
                         and Num(n2) = eval(m2, e)
                          in Num(n1 - n2)
  | (CTimes(m1,m2), e) → let Num(n1) = eval(m1, e)
                         and Num(n2) = eval(m2, e)
                          in Num(n1 * n2)
  | (CLam(m), e) → Fun(fun v → eval(m, Extend(v,e)))
  | (CApp(m1,m2), e) → let Fun(f) = eval(m1, e)
                       in f(eval(m2, e))
  | (CIfZero(m1,m2,m3), e) → let Num(n) = eval(m1, e)
                             in eval((if (n=0)
                                         then m2
                                         else m3),
                                      e)
  | (CVar(n), e) → lookup(n, e)
```

Step 4

- Stop relying on ML functions to implement our functions
- Instead, define a function as an expression-envrionment pair:

```
type xval = Num of int
          | Fun of cxpr * xenv
```

Step 4

```
let rec eval = function
    (CConst(v), e) → Num(v)
  | (CMinus(m1,m2), e) → let Num(n1) = eval(m1, e)
                         and Num(n2) = eval(m2, e)
                          in Num(n1 - n2)
  | (CTimes(m1,m2), e) → let Num(n1) = eval(m1, e)
                         and Num(n2) = eval(m2, e)
                          in Num(n1 * n2)
  | (CLam(m), e) → Fun(m, e)
  | (CApp(m1,m2), e)→ let Fun(fm, fe) = eval(m1, e)
                      in eval(fm, Extend(eval(m2, e), fe))
  | (CIfZero(m1,m2,m3), e) → let Num(n) = eval(m1, e)
                             in eval((if (n=0)
                                       then m2
                                       else m3),
                                    e)
  | (CVar(n), e) → lookup(n, e)
```

Step 5

- Stop relying on ML recursion
- Instead, package work-to-do in a ***continuation***

$$\textit{eval}\ \lceil 3 \rceil - \lceil 2 \rceil\ \texttt{then}\ \textit{kont}$$

$$\longrightarrow$$

$$\textit{eval}\ \lceil 3 \rceil\ \texttt{then}\ ? - \lceil 2 \rceil\ \texttt{then}\ \textit{kont}$$

$$\longrightarrow$$

$$\textit{eval}\ \lceil 2 \rceil\ \texttt{then}\ 3 - ?\ \texttt{then}\ \textit{kont}$$

$$\longrightarrow$$

$$\textit{kont}\ \texttt{with}\ 1$$

Step 5

```
type kont = Done
          | KSubArg of cxpr * xenv * kont
          | KMultArg of cxpr * xenv * kont
          | KSub of xval * kont
          | KMult of xval * kont
          | KAppArg of cxpr * xenv * kont
          | KApp of xval * kont
          | KIfZero of cxpr * cxpr * xenv * kont
```

Step 5

```
let rec eval = function
    (CConst(v), e, k) → kontinue(Num(v), k)
  | (CMinus(m1,m2), e, k) → eval(m1, e, KSubArg(m2,e,k))
  | (CTimes(m1,m2), e, k) → eval(m1, e, KMultArg(m2,e,k))
  | (CLam(m), e, k) → kontinue(Fun(m,e), k)
  | (CApp(m1,m2), e, k) → eval(m1, e, KAppArg(m2,e,k))
  | (CIfZero(m1,m2,m3), e, k) →
                                 eval(m1, e, KIfZero(m2,m3,e,k))
  | (CVar(n), e, k) → kontinue(lookup(n, e), k)
```

Step 5

```
let rec kontinue = function
    (v, KSubArg(m,e,k)) → eval(m, e, KSub(v,k))
  | (v, KMultArg(m,e,k)) → eval(m, e, KMult(v,k))
  | (Num(n2), KSub(Num(n1),k)) → kontinue(Num(n1-n2), k)
  | (Num(n2), KMult(Num(n1),k)) → kontinue(Num(n1*n2), k)
  | (v, KAppArg(m,e,k)) → eval(m, e, KApp(v,k))
  | (v, KApp(Fun(m,e),k)) → eval(m, Extend(v,e), k)
  | (Num(n), KIfZero(m2,m3,e,k)) → eval((if (n=0)
                                           then m2
                                           else m3),
                                        e, k)
  | (v, Done) → v
```

Step 6

- Stop relying on ML’s argument passing
- Instead, use a fixed set of registers for arguments

Step 6

```
let rec eval = function unit →
 match (!mReg, !eReg, !kReg) with
   (CConst(v), e, k) → vReg := Num(v); kontinue()
 | (CMinus(m1,m2), e, k) → mReg := m1;
            kReg := KSubArg(m2,e,k); eval()
 | (CTimes(m1,m2), e, k) → mReg := m1;
            kReg := KMultArg(m2,e,k); eval()
 | (CLam(m), e, k) → vReg := Fun(m,e); kontinue()
 | (CApp(m1,m2), e, k) → mReg := m1;
            kReg := KAppArg(m2,e,k); eval()
 | (CIfZero(m1,m2,m3), e, k) → mReg := m1;
            kReg := KIfZero(m2,m3,e,k); eval()
 | (CVar(n), e, k) → vReg := lookup(n, e); kontinue()
```

Step 6

```
let rec kontinue = function unit →
 match (!vReg, !kReg) with
  (v, KSubArg(m,e,k)) → mReg := m; eReg := e;
        kReg := KSub(v, k); eval()
 | (v, KMultArg(m,e,k)) → mReg := m;
        eReg := e; kReg := KMult(v,k); eval()
 | (Num(n2), KSub(Num(n1),k)) → vReg := Num(n1 - n2);
        kReg := k; kontinue()
 | (Num(n2), KMult(Num(n1),k)) → vReg := Num(n1 * n2);
       kReg := k; kontinue()
 | (v, KAppArg(m,e,k)) → mReg := m; eReg := e;
       kReg := KApp(v,k); eval()
 | (v, KApp(Fun(m,e),k)) → mReg := m;
       eReg := Extend(v,e); kReg := k; eval()
 | (Num(n), KIfZero(m2,m3,e,k)) →
       mReg := (if (n=0) then m2 else m3);
       eReg := e; kReg := k; eval()
 | (v, Done) → v
```

Step 7

- Stop using ML’s fancy datatypes
- Instead, assume only number and cons cells

Step 7

```
let rec comp = function
    (Const(v), e) → Cons(Int(1), Int(v))
  | (Minus(m1,m2), e) → Cons(Int(2),
                          Cons(comp(m1, e), comp(m2, e)))
  | (Times(m1,m2), e) → Cons(Int(3),
                          Cons(comp(m1, e), comp(m2, e)))
  | (Lam(var,m), e) → Cons(Int(4),
                       comp(m, CExtend(var, e)))
  | (App(m1,m2), e) → Cons(Int(5),
                       Cons(comp(m1, e), comp(m2, e)))
  | (IfZero(m1,m2,m3), e) →
      Cons(Int(6), Cons(comp(m1, e), Cons(comp(m2, e),
                                          comp(m3, e))))
  | (Var(var), e) → Cons(Int(7), Int(offset(var, e)))
```

Step 7

```
let rec eval = function unit →
 let e = !eReg and k = !kReg
 in match (!mReg) with
    Cons(Int(1), v) → vReg := v;
        kontinue()
  | Cons(Int(2), Cons(m1, m2)) → mReg := m1;
        kReg := Cons(Int(1), Cons(m2, Cons(e, k)));
        eval()
  | Cons(Int(3), Cons(m1, m2)) → mReg := m1;
        kReg := Cons(Int(2), Cons(m2, Cons(e, k)));
        eval()
  | ...
```

Step 7

```
let rec kontinue = function unit →
 match (!vReg, !kReg) with
   (v, Cons(Int(1), Cons(m, Cons(e, k)))) →
      mReg := m;
      eReg := e;
      kReg := Cons(Int(3), Cons(v, k));
      eval()
 | (v, Cons(Int(2), Cons(m, Cons(e, k)))) →
      mReg := m;
      eReg := e;
      kReg := Cons(Int(4), Cons(v, k));
      eval()
 | ...
```

Step 8

- Stop using cons cells
- Instead, we have a flat, numerically addressed memory containing only numbers

Step 8

```
let rec
comp = function
    (Const(v), e) → malloc(1, v)
  | (Minus(m1,m2), e) →
      malloc(2, malloc(comp(m1, e), comp(m2, e)))
  | (Times(m1,m2), e) →
      malloc(3, malloc(comp(m1, e), comp(m2, e)))
  | (Lam(var,m), e) →
      malloc(4, comp(m, CExtend(var, e)))
  | ...
```

Step 8

```
let rec eval = function unit →
 let e = !eReg and k = !kReg and p = !mReg
 in match (read p) with
      1 → vReg := read(p+1);
          kontinue()
    | 2 → mReg := read(read(p+1));
          kReg := malloc(1,
                    malloc(read(read(p+1)+1),
                    malloc(e, k)));
          eval()
    | 3 → ...
    | 4 → vReg := malloc(read(p+1), e);
          kontinue()
    | ...
```

Step 8

```
let rec kontinue = function unit →
 let p = !kReg and v = !vReg
  in match (read p) with
     1 → mReg := read(read(p+1));
         eReg := read(read(read(p+1)+1));
         kReg := malloc(3, malloc(v,
                   read(read(read(p+1)+1)+1)));
         eval()
   | 2 → mReg := read(read(p+1));
         eReg := read(read(read(p+1)+1));
         kReg := malloc(4, malloc(v,
                   read(read(read(p+1)+1)+1)));
         eval()
   | ...
```

Step 9

- Implement a garbage collector

(code provided on the course page)

Step 10

- Convert *eval* and *kontinue* to assembly

(not provided)

Conclusion

- Functional programming is programming with algebra
- A language definition comprises
  - a grammar
  - a set of reduction rules
  - an optional set of typing rules
- Implementation can be described as a transformation from meta-circular (obvious) to machine code (complex)