Outline

- **Programming with Functions**
- **Defining a Language**
- **Defining Type Rules**
- **Type Soundness**

Programming with Functions

- A program comprises function definitions and applications

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

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

Programming with Functions

- A program comprises function definitions and applications

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

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

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

Programming with Functions

- Functions consume and produce more than numbers

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

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

Programming with Functions

- Functions consume and produce more than numbers

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

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

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

Programming with Functions

- Functions consume and produce more than numbers

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

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

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

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

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

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

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

Computation as Algebra

- Compute using algebraic equivalences

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

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

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

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

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

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

$$\begin{aligned} &= \mathbf{lookup}(\text{"Jack"}, \langle \langle \text{"Jack"}, \text{"x1212"} \rangle, \text{empty} \rangle) \\ &= \text{"x1212"} \end{aligned}$$

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("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

$$\textbf{f}(\textbf{x}) \equiv \textbf{x} \times \textbf{x}$$

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

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

Higher-Order Functions

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

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

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

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

The Direction of Evaluation

$3 + 4 = ?$

The Direction of Evaluation

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

The Direction of Evaluation

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

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

Expressions and Values

- Many possible ***expressions***

$$8$$

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

fst

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

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

$$8$$

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 $\mathbf{a}(\mathbf{b}(\mathbf{...}))$ in place of **...**

Evaluation with Higher-Order Functions

- Problem: creating new function values

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

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

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

$$\begin{aligned} \mathbf{compose}(\mathbf{f}, \mathbf{g}) &\rightarrow \mathbf{...} \\ &\rightarrow \mathbf{h} \end{aligned}$$

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

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

Evaluation with Higher-Order Functions

- Definition with $\equiv$ merely creates a shorthand

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

- Apply functions through λ-application reduction

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

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

$$(\lambda\, \mathbf{x} \,.\, \mathbf{M})(\mathbf{v}) \rightarrow \mathbf{M}[\mathbf{v}/\mathbf{x}]$$

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

Evaluation with Higher-Order Functions

- Simple functions as values

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

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

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

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

Evaluation with Higher-Order Functions

- Simple functions as values

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

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

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

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

Evaluation with Higher-Order Functions

- Returning to the definition of **compose**

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

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

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

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

Abbreviations

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

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

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

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

Outline

- **Programming with Functions**
- ➡ **Defining a Language**
- **Defining Type Rules**
- **Type Soundness**

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

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

where parentheses can be put around any **M**

$\lceil 5 \rceil$ represents 5

Syntax: Expressions

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

where parentheses can be put around any **M**

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

Syntax: Expressions

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

where parentheses can be put around any **M**

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

Syntax: Expressions

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

where parentheses can be put around any **M**

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

Syntax: Values

$$\begin{array}{rcl} \mathbf{V} & = & \lceil \mathbf{n} \rceil \\ & | & \lambda \mathbf{x} . \mathbf{M} \end{array}$$

$\lceil 5 \rceil$	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

$$
\begin{array}{lll}
\lceil \mathbf{n}_1 \rceil - \lceil \mathbf{n}_2 \rceil & \rightarrow & \lceil \mathbf{n}_1 - \mathbf{n}_2 \rceil \\
\lceil \mathbf{n}_1 \rceil \times \lceil \mathbf{n}_2 \rceil & \rightarrow & \lceil \mathbf{n}_1 \times \mathbf{n}_2 \rceil \\
\texttt{if0}\ \lceil 0 \rceil\ \textbf{then}\ \mathbf{M}_1\ \textbf{else}\ \mathbf{M}_2 & \rightarrow & \mathbf{M}_1 \\
\texttt{if0}\ \lceil \mathbf{n} \rceil\ \textbf{then}\ \mathbf{M}_1\ \textbf{else}\ \mathbf{M}_2 & \rightarrow & \mathbf{M}_2 \\
 & & \text{if } \mathbf{n} \neq 0 \\
(\lambda\, \mathbf{x} \,.\, \mathbf{M})(\mathbf{V}) & \rightarrow & \mathbf{M}[\mathbf{V}/\mathbf{x}]
\end{array}
$$

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

Reductions

$$
\begin{array}{lll}
\lceil \mathbf{n}_1 \rceil - \lceil \mathbf{n}_2 \rceil & \rightarrow & \lceil \mathbf{n}_1 - \mathbf{n}_2 \rceil \\
\lceil \mathbf{n}_1 \rceil \times \lceil \mathbf{n}_2 \rceil & \rightarrow & \lceil \mathbf{n}_1 \times \mathbf{n}_2 \rceil \\
\texttt{if0}\ \lceil 0 \rceil\ \textbf{then}\ \mathbf{M}_1\ \textbf{else}\ \mathbf{M}_2 & \rightarrow & \mathbf{M}_1 \\
\texttt{if0}\ \lceil \mathbf{n} \rceil\ \textbf{then}\ \mathbf{M}_1\ \textbf{else}\ \mathbf{M}_2 & \rightarrow & \mathbf{M}_2 \\
 & & \text{if } \mathbf{n} \neq 0 \\
(\lambda\, \mathbf{x} \,.\, \mathbf{M})(\mathbf{V}) & \rightarrow & \mathbf{M}[\mathbf{V}/\mathbf{x}]
\end{array}
$$

$$\texttt{if0}\ \lceil 0 \rceil\ \textbf{then}\ \lceil 5 \rceil\ \textbf{else}\ (\lambda\, \mathbf{x} \,.\, \mathbf{x}) \rightarrow \lceil 5 \rceil$$

Reductions

$$
\begin{array}{lll}
\lceil \mathbf{n}_1 \rceil - \lceil \mathbf{n}_2 \rceil & \rightarrow & \lceil \mathbf{n}_1 - \mathbf{n}_2 \rceil \\
\lceil \mathbf{n}_1 \rceil \times \lceil \mathbf{n}_2 \rceil & \rightarrow & \lceil \mathbf{n}_1 \times \mathbf{n}_2 \rceil \\
\texttt{if0}\ \lceil 0 \rceil\ \textbf{then}\ \mathbf{M}_1\ \textbf{else}\ \mathbf{M}_2 & \rightarrow & \mathbf{M}_1 \\
\texttt{if0}\ \lceil \mathbf{n} \rceil\ \textbf{then}\ \mathbf{M}_1\ \textbf{else}\ \mathbf{M}_2 & \rightarrow & \mathbf{M}_2 \\
 & & \text{if } \mathbf{n} \neq 0 \\
(\lambda\, \mathbf{x} \,.\, \mathbf{M})(\mathbf{V}) & \rightarrow & \mathbf{M}[\mathbf{V}/\mathbf{x}]
\end{array}
$$

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

Reductions

$$
\begin{array}{lll}
\lceil \mathbf{n}_1 \rceil - \lceil \mathbf{n}_2 \rceil & \rightarrow & \lceil \mathbf{n}_1 - \mathbf{n}_2 \rceil \\
\lceil \mathbf{n}_1 \rceil \times \lceil \mathbf{n}_2 \rceil & \rightarrow & \lceil \mathbf{n}_1 \times \mathbf{n}_2 \rceil \\
\texttt{if0}\ \lceil 0 \rceil\ \textbf{then}\ \mathbf{M}_1\ \textbf{else}\ \mathbf{M}_2 & \rightarrow & \mathbf{M}_1 \\
\texttt{if0}\ \lceil \mathbf{n} \rceil\ \textbf{then}\ \mathbf{M}_1\ \textbf{else}\ \mathbf{M}_2 & \rightarrow & \mathbf{M}_2 \\
 & & \text{if } \mathbf{n} \neq 0 \\
(\lambda\, \mathbf{x} \,.\, \mathbf{M})(\mathbf{V}) & \rightarrow & \mathbf{M}[\mathbf{V}/\mathbf{x}]
\end{array}
$$

$$(\lambda\, \mathbf{x} \,.\, \mathbf{x} \times \lceil 10 \rceil)(\lceil 8 \rceil) \rightarrow \lceil 8 \rceil \times \lceil 10 \rceil$$

Reductions in Context

$$\begin{array}{lll} M_1 - M_2 & \to & M_1' - M_2 \\ & & \text{where } M_1 \to M_1' \\ V - M_2 & \to & V - M_2' \\ & & \text{where } M_2 \to M_2' \\ \\ M_1 \times M_2 & \to & M_1' \times M_2 \\ & & \ldots \end{array}$$

$$(\lceil 5 \rceil \times \lceil 2 \rceil) - (\lceil 3 \rceil \times \lceil 4 \rceil) \to \lceil 10 \rceil - (\lceil 3 \rceil \times \lceil 4 \rceil)$$

Reductions in Context

$$\begin{array}{lll} M_1 - M_2 & \to & M_1' - M_2 \\ & & \text{where } M_1 \to M_1' \\ V - M_2 & \to & V - M_2' \\ & & \text{where } M_2 \to M_2' \\ \\ M_1 \times M_2 & \to & M_1' \times M_2 \\ & & \ldots \end{array}$$

$$\lceil 10 \rceil - (\lceil 3 \rceil \times \lceil 4 \rceil) \to \lceil 10 \rceil - \lceil 12 \rceil$$

Reductions in Context

$$\begin{array}{lll} \texttt{if0}\ M\ \texttt{then}\ M_1\ \texttt{else}\ M_2 & \to & \texttt{if0}\ M'\ \texttt{then}\ M_1\ \texttt{else}\ M_2 \\ & & \text{where } M \to M' \\ \\ M_1\ M_2 & \to & M_1'\ M_2 \\ & & \text{where } M_1 \to M_1' \\ V\ M_2 & \to & V\ M_2' \\ & & \text{where } M_2 \to M_2' \end{array}$$

$$(\lambda x . x)(\lceil 2 \rceil \times \lceil 2 \rceil) \to (\lambda x . x)(\lceil 4 \rceil)$$

Reductions in Context

$$\begin{array}{lll} \texttt{if0}\ M\ \texttt{then}\ M_1\ \texttt{else}\ M_2 & \to & \texttt{if0}\ M'\ \texttt{then}\ M_1\ \texttt{else}\ M_2 \\ & & \text{where } M \to M' \\ \\ M_1\ M_2 & \to & M_1'\ M_2 \\ & & \text{where } M_1 \to M_1' \\ V\ M_2 & \to & V\ M_2' \\ & & \text{where } M_2 \to M_2' \end{array}$$

$$((\lambda x . x)(\lambda y . y))(\lceil 2 \rceil \times \lceil 2 \rceil) \to (\lambda y . y)(\lceil 2 \rceil \times \lceil 2 \rceil)$$

Reductions in Context

A simpler way: define context

$$
\begin{array}{lcl}
\mathbf{E} & = & \mathbf{[\,]} \\
 & | & \mathbf{E} - \mathbf{M} \\
 & | & \mathbf{V} - \mathbf{E} \\
 & | & \mathbf{E} \times \mathbf{M} \\
 & | & \mathbf{V} \times \mathbf{E} \\
 & | & (\mathbf{E}\ \mathbf{M}) \\
 & | & (\mathbf{V}\ \mathbf{E}) \\
 & | & \texttt{if0}\ \mathbf{E}\ \texttt{then}\ \mathbf{M}\ \texttt{else}\ \mathbf{M}
\end{array}
$$

$$\mathbf{E[M]} \rightarrow \mathbf{E[M']} \quad \text{where } \mathbf{M} \rightarrow \mathbf{M'}$$

$\mathbf{E[M]}$ means $\mathbf{E}$ with $\mathbf{[\,]}$ replaced by $\mathbf{M}$

Reductions in Context

A simpler way: define context

$$
\begin{array}{lcl}
\mathbf{E} & = & \mathbf{[\,]} \\
 & | & \mathbf{E} - \mathbf{M} \\
 & | & \mathbf{V} - \mathbf{E} \\
 & | & \mathbf{E} \times \mathbf{M} \\
 & | & \mathbf{V} \times \mathbf{E} \\
 & | & (\mathbf{E}\ \mathbf{M}) \\
 & | & (\mathbf{V}\ \mathbf{E}) \\
 & | & \texttt{if0}\ \mathbf{E}\ \texttt{then}\ \mathbf{M}\ \texttt{else}\ \mathbf{M}
\end{array}
$$

$$\mathbf{E[M]} \rightarrow \mathbf{E[M']} \quad \text{where } \mathbf{M} \rightarrow \mathbf{M'}$$

$$
\begin{array}{lcl}
\mathbf{E} & = & \lceil 4 \rceil - (\mathbf{[\,]} \times (\lceil 2 \rceil + \lceil 1 \rceil)) \\
\mathbf{E[}(\lceil 4 \rceil - \lceil 5 \rceil)\mathbf{]} & = & \lceil 4 \rceil - ((\lceil 4 \rceil - \lceil 5 \rceil) \times (\lceil 2 \rceil + \lceil 1 \rceil))
\end{array}
$$

Reductions

$$
\begin{array}{lcl}
\lceil \mathbf{n}_1 \rceil - \lceil \mathbf{n}_2 \rceil & \rightarrow & \lceil \mathbf{n}_1 - \mathbf{n}_2 \rceil \\
\lceil \mathbf{n}_1 \rceil \times \lceil \mathbf{n}_2 \rceil & \rightarrow & \lceil \mathbf{n}_1 \times \mathbf{n}_2 \rceil \\
\texttt{if0}\ \lceil 0 \rceil\ \texttt{then}\ \mathbf{M}_1\ \texttt{else}\ \mathbf{M}_2 & \rightarrow & \mathbf{M}_1 \\
\texttt{if0}\ \lceil \mathbf{n} \rceil\ \texttt{then}\ \mathbf{M}_1\ \texttt{else}\ \mathbf{M}_2 & \rightarrow & \mathbf{M}_2 \\
 & & \text{if } \mathbf{n} \neq 0 \\
(\lambda \mathbf{x} . \mathbf{M})(\mathbf{V}) & \rightarrow & \mathbf{M[V/x]} \\
\mathbf{E[M]} & \rightarrow & \mathbf{E[M']} \\
 & & \text{where } \mathbf{M} \rightarrow \mathbf{M'}
\end{array}
$$

Is this language deterministic?

Deterministic Reduction

Theorem: For any **M**, at most one reduction rule applies.

Proof: By induction on the structure of **M**.

... requires a lemma ...

Lemma: There exists at most one **E** and $\mathbf{M}_0$ such that $\mathbf{E[M_0]} = \mathbf{M}$ where $\mathbf{M}_0$ is reducible by one of the first five reduction rules.

Proof: By induction on the structure of **M**.

Induction on Expressions

$$
\begin{array}{rcl}
\mathbf{M} & = & \lceil \mathbf{n} \rceil \\
& | & \mathbf{x} \\
& | & \mathbf{M} - \mathbf{M} \\
& | & \mathbf{M} \times \mathbf{M} \\
& | & \texttt{if0}\ \mathbf{M}\ \texttt{then}\ \mathbf{M}\ \texttt{else}\ \mathbf{M} \\
& | & \lambda \mathbf{x} . \mathbf{M} \\
& | & \mathbf{M}\ \mathbf{M}
\end{array}
$$

base case   inductive case

Base Cases

$$
\begin{array}{rcl}
\mathbf{E} & = & [\,] \mid \mathbf{E} - \mathbf{M} \mid \mathbf{V} - \mathbf{E} \mid \mathbf{E} \times \mathbf{M} \mid \mathbf{V} \times \mathbf{E} \\
& | & (\mathbf{E}\ \mathbf{M}) \mid (\mathbf{V}\ \mathbf{E}) \mid \texttt{if0}\ \mathbf{E}\ \texttt{then}\ \mathbf{M}\ \texttt{else}\ \mathbf{M}
\end{array}
$$

- Assume $\mathbf{M} = \lceil \mathbf{n} \rceil$
  - The only way to match the grammar for $\mathbf{E}$ is $\mathbf{E} = [\,]$ and $\mathbf{M}_0 = \lceil \mathbf{n} \rceil$. But that $\mathbf{M}_0$ is not reducible, so there are no matches.
- Assume $\mathbf{M} = \mathbf{x}$
  - The only way to match the grammar for $\mathbf{E}$ is $\mathbf{E} = [\,]$ and $\mathbf{M}_0 = \mathbf{x}$...

Inductive Cases

$$
\begin{array}{rcl}
\mathbf{E} & = & [\,] \mid \mathbf{E} - \mathbf{M} \mid \mathbf{V} - \mathbf{E} \mid \mathbf{E} \times \mathbf{M} \mid \mathbf{V} \times \mathbf{E} \\
& | & (\mathbf{E}\ \mathbf{M}) \mid (\mathbf{V}\ \mathbf{E}) \mid \texttt{if0}\ \mathbf{E}\ \texttt{then}\ \mathbf{M}\ \texttt{else}\ \mathbf{M}
\end{array}
$$

- Assume $\mathbf{M} = \mathbf{M}_1 - \mathbf{M}_2$
  - Assume $\mathbf{M}_1 \neq \mathbf{V}_1$. The only match is $\mathbf{E} = \mathbf{E}_1 - \mathbf{M}_2$. By induction, there is a unique $\mathbf{E}_1[\mathbf{M}_0'] = \mathbf{M}_1$, and $\mathbf{M}_0' = \mathbf{M}_0$.
  - Assume $\mathbf{M}_1 = \mathbf{V}_1$. This matches $\mathbf{E} = \mathbf{E}_1 - \mathbf{M}_2$, but $\mathbf{E}_1$ would have to be $[\,]$ and $\mathbf{M}_0$ would have to be $\mathbf{V}_1$, which is not reducible. So $\mathbf{E} = \mathbf{V}_1 - \mathbf{E}_2$. By induction, there is a unique $\mathbf{E}_2[\mathbf{M}_0'] = \mathbf{M}_2$, and $\mathbf{M}_0' = \mathbf{M}_0$.

Inductive Cases

$$
\begin{array}{rcl}
\mathbf{E} & = & [\,] \mid \mathbf{E} - \mathbf{M} \mid \mathbf{V} - \mathbf{E} \mid \mathbf{E} \times \mathbf{M} \mid \mathbf{V} \times \mathbf{E} \\
& | & (\mathbf{E}\ \mathbf{M}) \mid (\mathbf{V}\ \mathbf{E}) \mid \texttt{if0}\ \mathbf{E}\ \texttt{then}\ \mathbf{M}\ \texttt{else}\ \mathbf{M}
\end{array}
$$

- Assume $\mathbf{M} = \mathbf{M}_1 \times \mathbf{M}_2$.
  - Analogous to the subtraction case.

Inductive Cases

$$\begin{array}{rcl} \mathbf{E} & = & \mathtt{[\,]} \mid \mathbf{E} - \mathbf{M} \mid \mathbf{V} - \mathbf{E} \mid \mathbf{E} \times \mathbf{M} \mid \mathbf{V} \times \mathbf{E} \\ & \mid & (\mathbf{E}\ \mathbf{M}) \mid (\mathbf{V}\ \mathbf{E}) \mid \mathtt{if0}\ \mathbf{E}\ \mathtt{then}\ \mathbf{M}\ \mathtt{else}\ \mathbf{M} \end{array}$$

- Assume $\mathbf{M} = \mathbf{M}_1\ \mathbf{M}_2$.
  - Analogous to the subtraction case.

Inductive Cases

$$\begin{array}{rcl} \mathbf{E} & = & \mathtt{[\,]} \mid \mathbf{E} - \mathbf{M} \mid \mathbf{V} - \mathbf{E} \mid \mathbf{E} \times \mathbf{M} \mid \mathbf{V} \times \mathbf{E} \\ & \mid & (\mathbf{E}\ \mathbf{M}) \mid (\mathbf{V}\ \mathbf{E}) \mid \mathtt{if0}\ \mathbf{E}\ \mathtt{then}\ \mathbf{M}\ \mathtt{else}\ \mathbf{M} \end{array}$$

- Assume $\mathbf{M} = \lambda\,\mathbf{x}\,.\,\mathbf{M}_1$.
  - Analogous to the number case.

Inductive Cases

$$\begin{array}{rcl} \mathbf{E} & = & \mathtt{[\,]} \mid \mathbf{E} - \mathbf{M} \mid \mathbf{V} - \mathbf{E} \mid \mathbf{E} \times \mathbf{M} \mid \mathbf{V} \times \mathbf{E} \\ & \mid & (\mathbf{E}\ \mathbf{M}) \mid (\mathbf{V}\ \mathbf{E}) \mid \mathtt{if0}\ \mathbf{E}\ \mathtt{then}\ \mathbf{M}\ \mathtt{else}\ \mathbf{M} \end{array}$$

- Assume $\mathbf{M} = \mathtt{if0}\ \mathbf{M}_1\ \mathtt{then}\ \mathbf{M}_2\ \mathtt{else}\ \mathbf{M}_3$. The only match is $\mathbf{E} = \mathtt{if0}\ \mathbf{E}_1\ \mathtt{then}\ \mathbf{M}_2\ \mathtt{else}\ \mathbf{M}_3$.
  - Assume $\mathbf{M}_1 = \mathbf{V}_1$. Then $\mathbf{E}_1 = \mathtt{[\,]}$ and there is no non-value $\mathbf{M}_0$.
  - Assume $\mathbf{M}_1 \neq \mathbf{V}_1$. By induction, there is a unique $\mathbf{E}_1\mathtt{[}\mathbf{M}_0'\mathtt{]} = \mathbf{M}_1$, and $\mathbf{M}_0' = \mathbf{M}_0$.

Inductive Cases

$$\begin{array}{rcl} \mathbf{E} & = & \mathtt{[\,]} \mid \mathbf{E} - \mathbf{M} \mid \mathbf{V} - \mathbf{E} \mid \mathbf{E} \times \mathbf{M} \mid \mathbf{V} \times \mathbf{E} \\ & \mid & (\mathbf{E}\ \mathbf{M}) \mid (\mathbf{V}\ \mathbf{E}) \mid \mathtt{if0}\ \mathbf{E}\ \mathtt{then}\ \mathbf{M}\ \mathtt{else}\ \mathbf{M} \end{array}$$

Since we have covered every possible shape of **M**, the lemma is proved.

Handling State

$$\begin{array}{lcl}
\mathbf{M} & = & \mathbf{...} \\
 & | & \mathbf{newref\ M} \\
 & | & \mathbf{defref\ M} \\
 & | & \mathbf{setref\ M} = \mathbf{M}
\end{array}$$

$$\begin{array}{lcl}
\mathbf{E} & = & \mathbf{...} \\
 & | & \mathbf{newref\ E} \\
 & | & \mathbf{deref\ E} \\
 & | & \mathbf{setref\ E} = \mathbf{M} \\
 & | & \mathbf{setref\ V} = \mathbf{E}
\end{array}$$

Handling State

Possible reduction rules:

$$\begin{array}{lcl}
\mathbf{V} & = & \mathbf{...\ |\ newref\ V} \\
\\
\mathbf{deref}\ (\mathbf{newref\ V}) & \rightarrow & \mathbf{V} \\
\mathbf{setref}\ (\mathbf{newref\ V_1}) = \mathbf{V_2} & \rightarrow & \mathbf{newref\ V_2}
\end{array}$$

Example:

$$\begin{array}{ll}
\multicolumn{2}{l}{(\lambda\ \mathbf{r\ .\ deref\ r})(\mathbf{setref}\ (\mathbf{newref} \lceil 5 \rceil) = \lceil 12 \rceil)} \\
= & (\lambda\ \mathbf{r\ .\ deref\ r})(\mathbf{newref} \lceil 12 \rceil) \\
= & \mathbf{deref}\ (\mathbf{newref} \lceil 12 \rceil) \\
= & \lceil 12 \rceil
\end{array}$$

Handling State

Possible reduction rules:

$$\begin{array}{lcl}
\mathbf{V} & = & \mathbf{...\ |\ newref\ V} \\
\\
\mathbf{deref}\ (\mathbf{newref\ V}) & \rightarrow & \mathbf{V} \\
\mathbf{setref}\ (\mathbf{newref\ V_1}) = \mathbf{V_2} & \rightarrow & \mathbf{newref\ V_2}
\end{array}$$

Problem:

$$\begin{array}{ll}
\multicolumn{2}{l}{(\lambda\ \mathbf{r}\ .\ (\lambda\ \mathbf{d\ .\ deref\ r})(\mathbf{setref\ r} = \lceil 10 \rceil))(\mathbf{newref} \lceil 5 \rceil)} \\
= & (\lambda\ \mathbf{d\ .\ deref}\ (\mathbf{newref}\ 5))(\mathbf{setref}\ (\mathbf{newref} \lceil 5 \rceil) = \lceil 10 \rceil) \\
= & (\lambda\ \mathbf{d\ .\ deref}\ (\mathbf{newref}\ 5))(\mathbf{newref} \lceil 10 \rceil) \\
= & \mathbf{deref}\ (\mathbf{newref} \lceil 5 \rceil) \\
= & \lceil 5 \rceil
\end{array}$$

Handling State

Correct reduction requires a ***store***

$$\begin{array}{lcl}
\sigma & = & \text{a store address} \\
\mathbf{S} & = & \text{a mapping from } \sigma \text{ to } \mathbf{V} \\
\mathbf{V} & = & \mathbf{...}\ |\ \sigma
\end{array}$$

$$\begin{array}{lcl}
\langle \mathbf{S}, \lceil \mathbf{n_1} \rceil - \lceil \mathbf{n_2} \rceil \rangle & \rightarrow & \langle \mathbf{S}, \lceil \mathbf{n_1 - n_2} \rceil \rangle \\
\mathbf{...} & & \\
\langle \mathbf{S},\ \mathbf{newref\ V} \rangle & \rightarrow & \langle \mathbf{S}[\sigma = \mathbf{V}], \sigma \rangle \\
 & & \text{where } \sigma \text{ is not in } \mathbf{S} \\
\langle \mathbf{S}[\sigma = \mathbf{V}],\ \mathbf{deref}\ \sigma \rangle & \rightarrow & \langle \mathbf{S}[\sigma = \mathbf{V}], \mathbf{V} \rangle \\
\langle \mathbf{S}[\sigma = \mathbf{V_1}],\ \mathbf{setref}\ \sigma = \mathbf{V_2} & \rightarrow & \langle \mathbf{S}[\sigma = \mathbf{V_2}], \sigma \rangle
\end{array}$$

Handling State

$\langle \mathbf{S}, \lceil \mathbf{n}_1 \rceil - \lceil \mathbf{n}_2 \rceil \rangle \rightarrow \langle \mathbf{S}, \lceil \mathbf{n}_1 - \mathbf{n}_2 \rceil \rangle$

...

$\langle \mathbf{S}, \textbf{newref } \mathbf{V} \rangle \rightarrow \langle \mathbf{S}[\sigma = \mathbf{V}], \sigma \rangle$ where σ is not in **S**

$\langle \mathbf{S}[\sigma = \mathbf{V}], \textbf{deref } \sigma \rangle \rightarrow \langle \mathbf{S}[\sigma = \mathbf{V}], \mathbf{V} \rangle$

$\langle \mathbf{S}[\sigma = \mathbf{V}_1], \textbf{setref } \sigma = \mathbf{V}_2 \rightarrow \langle \mathbf{S}[\sigma = \mathbf{V}_2], \sigma \rangle$

$$
\begin{aligned}
&\langle \{\}, (\lambda\, \mathbf{r} \,.\, (\lambda\, \mathbf{d} \,.\, \textbf{deref } \mathbf{r})(\textbf{setref } \mathbf{r} = \lceil 10 \rceil))(\textbf{newref} \lceil 5 \rceil) \rangle \\
&= \langle \{\sigma = \lceil 5 \rceil\}, (\lambda\, \mathbf{r} \,.\, (\lambda\, \mathbf{d} \,.\, \textbf{deref } \mathbf{r})(\textbf{setref } \mathbf{r} = \lceil 10 \rceil))(\sigma) \rangle \\
&= \langle \{\sigma = \lceil 5 \rceil\}, (\lambda\, \mathbf{d} \,.\, \textbf{deref } \sigma)(\textbf{setref } \sigma = \lceil 10 \rceil) \rangle \\
&= \langle \{\sigma = \lceil 10 \rceil\}, (\lambda\, \mathbf{d} \,.\, \textbf{deref } \sigma)(\sigma) \rangle \\
&= \langle \{\sigma = \lceil 10 \rceil\}, \textbf{deref } \sigma \rangle \\
&= \langle \{\sigma = \lceil 10 \rceil\}, \lceil 10 \rceil \rangle
\end{aligned}
$$

Handling State

After changing the language, we have to go back and fix the proofs (in principle).

Outline

- **Programming with Functions**
- **Defining a Language**
- ➡ **Defining Type Rules**
- **Type Soundness**

Type Rules

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

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

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

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

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

Type Rules

- arithmetic expressions produce integers

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

$$\frac{M_1 : \texttt{int} \qquad M_2 : \texttt{int}}{M_1 - M_2 : \texttt{int}}$$

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

Type Rules

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

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

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

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

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

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

Type Rules

- Fix up old rules

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

$$\frac{\Gamma \vdash M_1 : \texttt{int} \qquad \Gamma \vdash M_2 : \texttt{int}}{\Gamma \vdash M_1 - M_2 : \texttt{int}}$$

$$\frac{\Gamma \vdash M : \texttt{int} \qquad \Gamma \vdash M_1 : T \qquad \Gamma \vdash M_2 : T}{\Gamma \vdash \texttt{if0}\ M\ \texttt{then}\ M_1\ \texttt{else}\ M_2 : T}$$

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

Type Rules

- Function type: $\mathbf{T_1} \rightarrow \mathbf{T_2}$

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

$$\frac{\Gamma \vdash \mathbf{M_1} : \mathbf{T'} \rightarrow \mathbf{T} \qquad \Gamma \vdash \mathbf{M_2} : \mathbf{T'}}{\Gamma \vdash (\mathbf{M_1}\;\mathbf{M_2}) : \mathbf{T}}$$

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

Type Rules

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

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

Type Rules

$$\frac{\Gamma \vdash \mathbf{M} : \mathbf{T}}{\Gamma \vdash \mathbf{newref}\ \mathbf{M} : \mathbf{ref}\ \mathbf{T}} \qquad \frac{\Gamma \vdash \mathbf{M} : \mathbf{ref}\ \mathbf{T}}{\Gamma \vdash \mathbf{deref}\ \mathbf{M} : \mathbf{T}}$$

$$\frac{\Gamma \vdash \mathbf{M_1} : \mathbf{ref}\ \mathbf{T} \qquad \Gamma \vdash \mathbf{M_2} : \mathbf{T}}{\Gamma \vdash \mathbf{setref}\ \mathbf{M_1} = \mathbf{M_2} : \mathbf{ref}\ \mathbf{T}}$$

$$\frac{\dfrac{\dfrac{\{\} \vdash \lceil 5 \rceil : \mathtt{int}}{\{\} \vdash \mathbf{newref}\lceil 5 \rceil : \mathbf{ref}\ \mathtt{int}} \qquad \{\} \vdash \lceil 7 \rceil : \mathtt{int}}{\{\} \vdash \mathbf{setref}\ (\mathbf{newref}\lceil 5 \rceil) = \lceil 7 \rceil : \mathbf{ref}\ \mathtt{int}}}{\{\} \vdash \mathbf{deref}\ (\mathbf{setref}\ (\mathbf{newref}\lceil 5 \rceil) = \lceil 7 \rceil) : \mathtt{int}}$$

Outline

- **Programming with Functions**
- **Defining a Language**
- **Defining Type Rules**
- ➡ **Type Soundness**

Soundness

Theorem: If $\{\} \vdash \mathbf{M} : \mathbf{T}$ then either

- There exists $\mathbf{S}'$ and $\mathbf{V}$ such that $\langle \{\}, \mathbf{M} \rangle \rightarrow \mathbf{...} \rightarrow \langle \mathbf{S}', \mathbf{V} \rangle$
- For all $\mathbf{S}'$ and $\mathbf{M}'$, if $\langle \{\}, \mathbf{M} \rangle \rightarrow \mathbf{...} \rightarrow \langle \mathbf{S}', \mathbf{M}' \rangle$ then there exists $\mathbf{S}''$ and $\mathbf{M}''$ such that $\langle \mathbf{S}', \mathbf{M}' \rangle \rightarrow \langle \mathbf{S}'', \mathbf{M}'' \rangle$

In other words, an evaluation never gets stuck.

The proof relies on two lemmas: a **preservation lemma** and a **progress lemma**.

Soundness: Preservation

Lemma (Preservation): If

- $\langle \mathbf{S}, \mathbf{M} \rangle \rightarrow \langle \mathbf{S}', \mathbf{M}' \rangle$ and
- $||\mathbf{S}|| \vdash \mathbf{M} : \mathbf{T}$,

then

- $||\mathbf{S}'|| \vdash \mathbf{M}' : \mathbf{T}$

where $||\mathbf{S}||(\sigma) = \mathbf{T}$ if $\mathbf{S}(\sigma) = \mathbf{V}$ and $\{\} \vdash \mathbf{V} : \mathbf{T}$.

Proof: By induction on **M**.

Soundness: Progress

Lemma (Progress): If

- **M** is not a **V** and
- and $||\mathbf{S}|| \vdash \mathbf{M} : \mathbf{T}$,

then

- there exist $\mathbf{M}'$ and $\mathbf{S}'$ such that $\langle \mathbf{S}, \mathbf{M} \rangle \rightarrow \langle \mathbf{S}', \mathbf{M}' \rangle$.

Proof: By induction on **M**.

Soundness Proof Sketch

Lemma: If $||\mathbf{S}|| \vdash \mathbf{M} : \mathbf{T}$ then either

- There exists $\mathbf{S}'$ and $\mathbf{V}$ such that $\langle \mathbf{S}, \mathbf{M} \rangle \rightarrow \mathbf{...} \rightarrow \langle \mathbf{S}', \mathbf{V} \rangle$
- For all $\mathbf{S}'$ and $\mathbf{M}'$, if $\langle \mathbf{S}, \mathbf{M} \rangle \rightarrow \mathbf{...} \rightarrow \langle \mathbf{S}', \mathbf{M}' \rangle$ then there exists $\mathbf{S}''$ and $\mathbf{M}''$ such that $\langle \mathbf{S}', \mathbf{M}' \rangle \rightarrow \langle \mathbf{S}'', \mathbf{M}'' \rangle$

Proof sketch:

- The Progress Lemma says that we can take a step if we're not yet to a value.
- The Preservation Lemma says that the step preserves the type, so we'll be able to take another step.

Conclusion

- Programming languages are formally defined using algebra
- A language definition comprises
  - a grammar
  - a set of reduction rules
  - an optional set of typing rules
- Soundness ensures that the type rules and reduction rules are consistent