Expressions

Definition (Expression)

An **expression** is a finite combination of symbols which may be reduced (rewritten) to obtain a simpler expression or a result.

Example

$$\underbrace{1+2}_{\text{expression}} \overset{\text{Rewrite}}{\rightsquigarrow} \underbrace{3}_{\text{also an expression}}$$

Expressions in Imperative and Functional Programming

Use of Expressions

- Primary building block of functional programs
- Imperative languages: some expressions and some statements/commands (not expressions)
- (Purely) Functional Languages: Everything is an expression

Example (C Expressions)

- `2`
- `1+2`
- `x ? 1 : 2`

Counterexample (C Statements)

- `if(x) { y=1; } else { y=2; }`
- `while(i) {i--;}`

Syntax and Semantics

Expression Syntax

- Is the expression a valid combination of symbols?

C Syntax

- **Valid**: `(1 + 2) * 3`
- **Invalid**: `(1 + 2 * 3`

Expression Semantics

- **Type-checking rules** (*static semantics*): Produce a type or fail with an error
- **Evaluation rules** (*dynamic semantics*): Produce a value, exception, or infinite loop

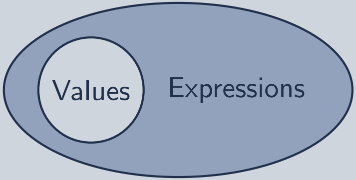
We will precisely define syntax and semantics over this course.

Values

Definition (Value)

A **value** is an expression that does not need further evaluation.

Illustration



Example

$$\underbrace{1+2}_{\text{expression}} \overset{\text{Rewrite}}{\rightsquigarrow} \underbrace{3}_{\text{value, expression}}$$

If Expressions

aka Conditionals

If Expression

if $\underbrace{e1}_{test}$ **then** $\underbrace{\text{e2}}_{\text{then clause}}$ **else** $\underbrace{\text{e3}}_{\text{else clause}}$

Evaluation

- Notation: Write α evaluates to β as $\alpha \rightsquigarrow \beta$
- When e1 $\rightsquigarrow$ true and e2 $\rightsquigarrow$ v,
  if e1 **then** e2 **else** e3 $\rightsquigarrow$ v
- When e1 $\rightsquigarrow$ false and e3 $\rightsquigarrow$ v,
  if e1 **then** e2 **else** e3 $\rightsquigarrow$ v

Type checking

- Notation: Write α has type τ as $\alpha{:}\tau$
- When
  - e1 : bool,
  - e2 : τ,
  - e3 : τ,

  (**if** e1 **then** e2 **else** e3) : τ

Type inference and annotation

Type Inference

- We can infer the of an **if** expression from the types of its constituent expressions.
- *type inference* is generally possible in functional languages and is performed by compilers
- If types cannot be inferred, compilation fails with a type error
- May add annotations to test/diagnose: Replace `e` with `e:t`

If Expression Type

$$(\textbf{if}\ e1\ \textbf{then}\ e2\ \textbf{else}\ e3) : \tau$$

where

- e1 : bool
- e2 : τ
- e3 : τ

Capability: "If it compiles, it (probably) works."

Definitions

Definition (definition)

A **definition** gives a name to a value.

Illustration

- Definitions are disjoint from expressions
- But syntactically, definitions contain expressions

Variables

Definition

- A **Variable** is a symbol representing an expression.
- **Bound** variables refer to some expression.
- **Free** variables DO NOT not refer to some expression (are unbound).
- **Immutable** variables CANNOT be changed (are constant).
- **Mutable** variables CAN be changed (reassigned).

Let Expression

Let Expression

$$\textbf{let}\ \underbrace{x}_{\text{identifier}} \leftarrow \underbrace{e_1}_{\text{binding exp.}}\ \textbf{in}\ \underbrace{e_2}_{\text{body exp.}}$$

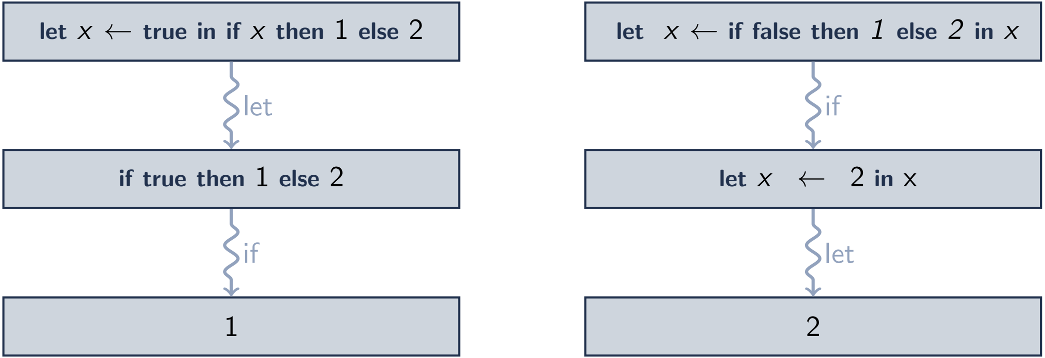
Evaluation

- Evaluate $e_1 \rightsquigarrow v_1$
- Substitute v_1 for x in e_2, yielding new expression e_2'
- Implementation: there is a memory location named x that contains/references v
- Evaluate: $e_2' \rightsquigarrow v_2$
- Result:
  $\textbf{let}\ x \leftarrow e_1\ \textbf{in}\ e_2 \quad \rightsquigarrow \quad v_2$

Type checking

- $e_1 : \tau_1$
- $x : \tau_1$
- $e_2 : \tau_2$
- $\textbf{let}\ x \leftarrow e_1\ \textbf{in}\ e_2 \quad : \quad \tau_2$

Example: Expression Evaluation



Exercise: Expression Evaluation

if **let** $x \leftarrow$ **true** **in** $\neg x$ **then** **false** **else** **true**

if **if** **true** **then** **false** **else** **true** **then** 1 **else** 2

Outline

Expressions

Functions

Lambda Expression

Anonymous Functions

Lambda Expression

$$\lambda \overbrace{x}^{\text{formal parameter}} . \underbrace{\alpha}_{\textit{body}}$$

- Create an unnamed function.
- The function is itself a value.
- Do not evaluate the body until applied (called)

Function Application

$$\overbrace{f}^{\text{function}} \underbrace{y}_{\text{actual parameter}}$$

- Bind actual parameter y to the formal parameter of f.
- Evaluate the body of f.

Example

$$\boxed{(\lambda a \,.\, 1 + a)\, 2} \overset{a = 2}{\rightsquigarrow} \boxed{1 + 2} \overset{+}{\rightsquigarrow} \boxed{3}$$

Contrasting Notations

Function Definition

Other notation	Lambda Calculus
$\textbf{function}(x)\ \{\textbf{return}\ 1 + x\}$	$\lambda x\,.\,1 + x$

Function Application

Other notation	Lambda Calculus
$f(2)$	$f\ 2$

Minimalist.

Functions are values: Binding

Example (Binding)

$$\textbf{let } f \leftarrow (\lambda a \,.\, 1 + a) \textbf{ in } f\ 1$$

$\rightsquigarrow$ let

$$(\lambda a \,.\, 1 + a)\, 2$$

$\rightsquigarrow$ a=2

$$1{+}2$$

$\rightsquigarrow$ +

$$3$$

Example (Passing)

$$(\lambda f \,.\, f\ 2)\,(\lambda a \,.\, 1 + a)$$

$\rightsquigarrow$ $f = \lambda a \,.\, 1{+}a$

$$(\lambda a \,.\, 1 + a)\, 2$$

$\rightsquigarrow$ a=2

$$1{+}2$$

$\rightsquigarrow$ +

$$3$$

Functions are values: Returning Functions

Example (Returning)

$$((\lambda b \,.\, (\lambda a \,.\, b + a))\, 1)\, 2$$

$\rightsquigarrow$ $b = 1$

$$(\lambda a \,.\, 1 + a)\, 2$$

$\rightsquigarrow$ a=2

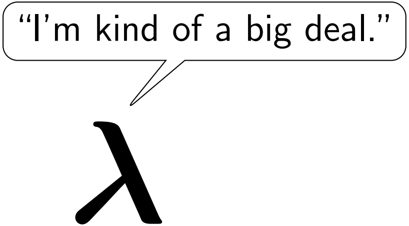
$$1+2$$

$\rightsquigarrow$ $+$

$$3$$

Lambda: Very Important!

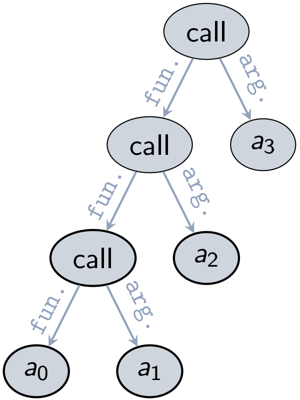
- Lambda can perform *any possible computation!*
- Trivial with Lambda:
  - N-ary functions
  - Variable binding (**let**)
  - Statements (;)
  - Named functions
  - OOP
- Also possible with Lambda:
  - Recursion
  - Conditionals (**if**)
  - Lists
  - Structs
  - Arithmetic ($+$, $-$, etc.)



Function Call Associativity

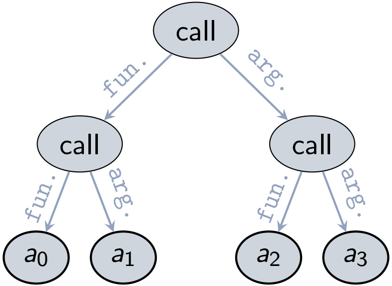
Evaluate Left to Right

$$(a_0 a_1 a_2 a_3) = ((((a_0 a_1) a_2) a_3)$$



Function Call Parenthesization

$(a_0\ a_1)\ (a_2\ a_3)$



Exercise: Lambda Evaluation

- $\left((\boldsymbol{\lambda} x \,.\, x)\ y \right)$

  $\leadsto$

- $\left((\boldsymbol{\lambda} x \,.\, x)\ (\boldsymbol{\lambda} y \,.\, y) \right)$

  $\leadsto$

- $\left((\boldsymbol{\lambda} x \,.\, (\boldsymbol{\lambda} y \,.\, xy))\ z \right)$

  $\leadsto$

- $\left((\boldsymbol{\lambda} x \,.\, x\, x)\ (\boldsymbol{\lambda} x \,.\, x\, x) \right)$

  $\leadsto$

Binary Functions

Reduction of Binary to Unary Functions

Binary: $\lambda \underbrace{x\ y}_{\text{formal params.}} . \underbrace{e}_{\text{body}}$ $\rightsquigarrow$ Unary: $\lambda x . \lambda y . e$

- Convert to two lambdas
- Outer lambda binds first param (x) and returns inner lambda
- Inner lambda binds second param (y) and evaluates

Example

$$(\lambda a\, b\, .\, a + b)\ 1\ 2$$

$\rightsquigarrow$

$$(\lambda a\, .\, \lambda b\, .\, a + b)\, 1\ 2$$

$\rightsquigarrow$ $a = 1$

$$(\lambda b\, .\, 1 + b)\, 2$$

$\rightsquigarrow$ $b = 2$

$$1 + 2$$

Each function call creates new bindings of its arguments.

MINES

N-ary Functions

N-ary functions

arity: $n+1$

$\lambda x_0\ x_1\ \ldots\ x_n \,.\, e$

$\rightsquigarrow$

arity: n

$\lambda x_0 \,.\, (\lambda x_1 \ldots x_n \,.\, \alpha)$

- Recursively convert to unary functions.

Example

$(\lambda a\, b\, c \,.\, a + b + c)\ 1\ 2\ 3$

$\rightsquigarrow$

$(\lambda a \,.\, \lambda b\, c \,.\, a + b + c)\, 1\ 2\ 3$

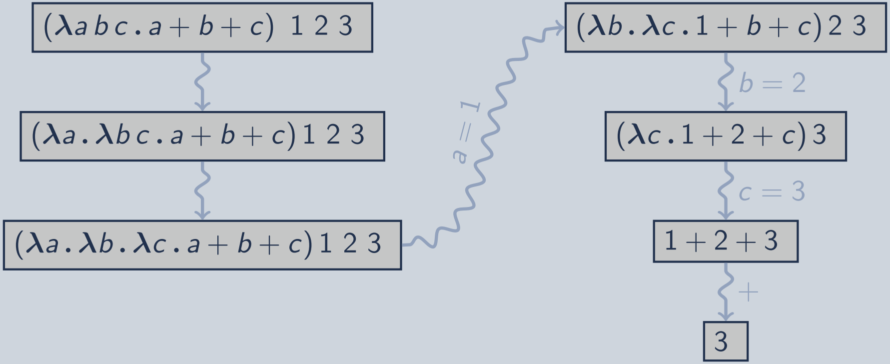
$\rightsquigarrow$

$(\lambda a \,.\, \lambda b \,.\, \lambda c \,.\, a + b + c)\, 1\ 2\ 3$

N-ary Functions

example, continued

Example



"Currying"

Binary Curry

$$\texttt{curry} \equiv \boldsymbol{\lambda} f\, a\,.\, (\boldsymbol{\lambda} b\,.\, f\, a\, b)$$

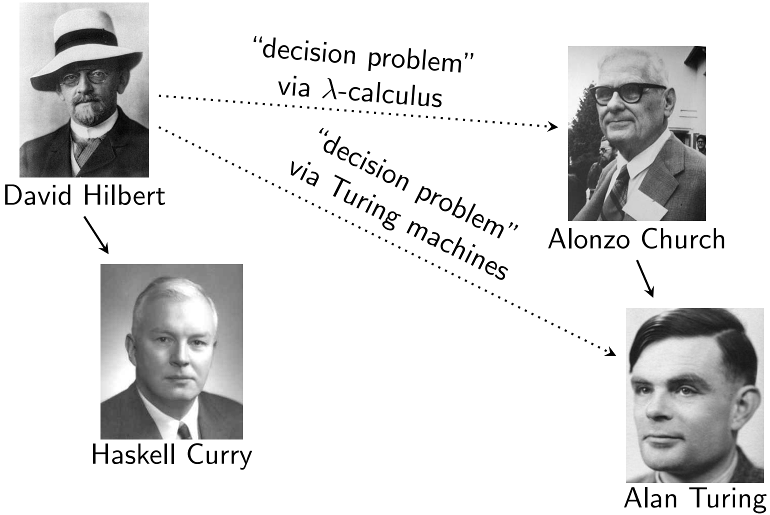
Function curry(f, a)

```
function g(b) is
    f(a,b)
g
```

Haskell B. Curry

Historical Interlude 0: Mathematics of Lambda

Curry, Hilbert, Church, and Turing



Let to Lambda

Reduction of Let to Lambda

Let

$$\textbf{let}\ \overbrace{x}^{\text{identifier}} \leftarrow \overbrace{e_1}^{\text{binding exp.}}\ \textbf{in}\ \overbrace{e_2}^{\text{body exp.}} \rightsquigarrow$$

Lambda

$$(\lambda\ \overbrace{x}^{\text{identifier}}\ .\ \overbrace{e_2}^{\text{body exp.}})\ \overbrace{e_1}^{\text{binding exp.}}$$

- Identifier x becomes lambda parameter
- Body exp. e_2 becomes the lambda body
- Binding exp. e_1 becomes the call argument

Example

$$\boxed{\textbf{let}\ a \leftarrow 2\ \textbf{in}\ 1 + a} \overset{\text{let} \rightsquigarrow \lambda}{\rightsquigarrow} \boxed{(\lambda a \,.\, 1 + a)\, 2} \overset{a = 2}{\rightsquigarrow} \boxed{1 + 2} \overset{+}{\rightsquigarrow} \boxed{3}$$

Sequential Definition to Let

Reduction of Sequential Definition to Let

Sequential Def.

$$\textbf{def}\ \overbrace{x}^{\text{identifier}} \leftarrow \overbrace{e}^{\text{binding exp.}}\ ;\ \underbrace{\beta}_{\text{body}}$$

$\rightsquigarrow$

Let

$$\textbf{let}\ \overbrace{x}^{\text{identifier}} \leftarrow \overbrace{e}^{\text{binding exp.}}\ \textbf{in}\ \overbrace{\beta}^{\text{body}}$$

Example

$$\boxed{\textbf{def}\ a \leftarrow 2;\ 1 + a} \rightsquigarrow \boxed{\textbf{let}\ a \leftarrow 2\ \textbf{in}\ 1 + a} \rightsquigarrow \boxed{(\lambda a \,.\, 1 + a)\, 2} \rightsquigarrow \boxed{1 + 2} \rightsquigarrow \boxed{3}$$

Local Functions to Let and Lambda

Reduction of Local Functions to Let and Lambda

Local Function

$$\textbf{flet}\ \overbrace{f}^{\text{name}}\left(\overbrace{x}^{\text{formal param.}}\right) \rightarrow \overbrace{e_1}^{\text{fun. body}}\ \textbf{in}\ \overbrace{e_2}^{\text{flet body}}$$

$\rightsquigarrow$

Let / Lambda

$$\textbf{let}\ f \leftarrow \lambda x \,.\, e_1\ \textbf{in}\ e_2$$

Example

$\textbf{flet}\ f\,(a) \rightarrow 1 + a\ \textbf{in}\ f\,2$ $\rightsquigarrow$ $\textbf{let}\ f \leftarrow \lambda a \,.\, 1 + a\ \textbf{in}\ f\,2$ $\rightsquigarrow$ $(\lambda a \,.\, 1 + a)\,2$ $\overset{\cdots}{\rightsquigarrow}$

Global Functions to Sequential Assignment

Reduction Global Functions to Sequential Assignment

Local Function

$$\textbf{defun}\ \overbrace{f}^{\text{name}} \left(\overbrace{x}^{\text{formal param.}}\right) \rightarrow \overbrace{e}^{\text{body exp.}} ;$$

$$\underbrace{\beta}_{\text{more expressions}}$$

$\leadsto$

Sequential Assignment

$$\textbf{def}\ f \leftarrow \boldsymbol{\lambda} x \,.\, e;$$

$$\underbrace{\beta}_{\text{more expressions}}$$

Example

$\textbf{defun}\ f\,(a) \rightarrow 1 + a;$
$f\,2$

$\leadsto$

$\textbf{def}\ f \leftarrow \boldsymbol{\lambda} a \,.\, 1 + a;$
$f\,2$

$\leadsto$

$\textbf{let}\ f \leftarrow \boldsymbol{\lambda} a \,.\, 1 + a\ \textbf{in}\ f\,2$

$\leadsto \ldots$

Example: Reduction to Lambda

- $\left(\textbf{let}\ \overbrace{x}^{\text{name}} \leftarrow \overbrace{2}^{\text{binding exp}}\ \textbf{in}\ \overbrace{1+x}^{\text{body}}\right) \overset{\texttt{let}}{\leadsto} \left(\left(\boldsymbol{\lambda}\ \overbrace{x}^{\text{name}}\ .\overbrace{1+x}^{\text{body}}\right) \overbrace{2}^{\text{binding exp}}\right) \overset{x=2}{\leadsto} 3$
- $\textbf{let}\ x \leftarrow 1\ \textbf{in}\ (\textbf{let}\ y \leftarrow 2\ \textbf{in}\ x+y)$

  $\overset{\texttt{let}\ x}{\leadsto} \boldsymbol{\lambda} x\,.\,(\textbf{let}\ y \leftarrow 2\ \textbf{in}\ x+y)\,1 \overset{\texttt{let}\ y}{\leadsto} \boldsymbol{\lambda} x\,.\,((\boldsymbol{\lambda} y\,.\,x+y)\,2)\,1$

  $\overset{x=1}{\leadsto} (\boldsymbol{\lambda} y\,.\,1+y)\,2 \overset{y=1}{\leadsto} 1+2 \overset{+}{\leadsto} 3$
- $\textbf{flet}\ f\,(x) \rightarrow \textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ 0\ \textbf{in}\ f\ \textbf{true}$

  $\overset{\text{flet}}{\leadsto} \textbf{let}\ f \leftarrow \boldsymbol{\lambda} x\,.\,\textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ 0\ \textbf{in}\ f\ \textbf{true}$

  $\overset{\text{let}}{\leadsto} (\boldsymbol{\lambda} f\,.\,f\ \textbf{true})\,(\boldsymbol{\lambda} x\,.\,\textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ 0)$

  $\overset{f=\boldsymbol{\lambda} x\,.\,\ldots}{\leadsto} (\boldsymbol{\lambda} x\,.\,\textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ 0)\,\textbf{true}$

  $\overset{x=\textbf{true}}{\leadsto} \textbf{if}\ \textbf{true}\ \textbf{then}\ 1\ \textbf{else}\ 0 \overset{\text{if}}{\leadsto} 1$

Exercise: Reduction to Lambda

- $\textbf{let}\ x \leftarrow 1\ \textbf{in}\ \boldsymbol{\lambda} a \,.\, x + a$
- $(\textbf{let}\ x \leftarrow 1\ \textbf{in}\ \boldsymbol{\lambda} a \,.\, x + a)\, 2$
- $\textbf{flet}\ f\ (a\ b) \rightarrow a + b\ \textbf{in}\ f\ 1\ 2$
- $\textbf{flet}\ f\ (n) \rightarrow (\textbf{if}\ n = 0\ \textbf{then}\ 1\ \textbf{else}\ n * f(n-1))\ \textbf{in}\ f\ 2$