Function Symbols

So far, all *FUNCTIONS* are ATOMS.

(Eg: `CAR`, `NULL`, `+`, . . .)

Now: λ-expressions:

```
(LAMBDA (X Y) (PLUS X (TIMES 2 Y)))
```

Special atom

a form

Formal parameter list: 2 elements $\leadsto$ 2 args.
Specifies how to get value from args

Comparison with λ-Expressions

- `(LAMBDA (X Y) (PLUS X (TIMES 2 Y)))`
  is a FUNCTION
  mapping $\mathcal{R} \times \mathcal{R}$ into $\mathcal{R}$!

- Analogy:

  function foo (x,y: real): real
  result $x + 2 \times y$
  end foo

- EXCEPT
  - no types
  - no name
  - full parentheses

Evaluating Forms with λ-Expressions

`(LAMBDA (X Y) (+ X (* 2 Y)))` of 4 and 15: Written

```
((LAMBDA (X Y) (+ X (* 2 Y))) 4 15)
```

1. Assign `X` to be 4, `Y` to be 15

2. Evaluate form: `(+ X (* 2 Y))`:

`EVAL[(+ X (* 2 Y))]`
$\leadsto$ `+` of `EVAL[X]` and `EVAL[(* 2 Y)]`
  `EVAL[X]` $\leadsto$ 4
  `EVAL[(* 2 Y)]`
    $\leadsto$ `*` of `EVAL[2]` and `EVAL[Y]`
    `EVAL[2]` $\leadsto$ 2
    `EVAL[Y]` $\leadsto$ 15
  $\leadsto$ 30
$\leadsto$ 34

λ-Expressions — General

- In general:
  list with 3 elements:

  `(LAMBDA (`$\mathtt{a}_1$ ... $\mathtt{a}_n$`)` $\langle$`form`$\rangle$`)`

  $n \geq 0$, $\mathtt{a}_i$ atoms

- Form

  `((LAMBDA (`$\mathtt{a}_1$ ... $\mathtt{a}_n$`)` $\langle$`form`$\rangle$`)` $\mathtt{v}_1$ ... $\mathtt{v}_n$`)`

  where each $\mathtt{v}_i$ is value.

- Evaluation:
  Evaluate $\langle$`form`$\rangle$ where each
  $\mathtt{a}_i$ variable has value $\mathtt{v}_i$.

Evaluating λ-Expression — II

- `(LAMBDA (X Y) (CONS (CAR X) Y))`
  takes 2 args:
  X: non-NIL list `(`s_1 ... s_n`)` $n > 0$
  Y: list `(`t_1 ... t_m`)` $m \geq 0$
  returns: `(`s_1 t_1 ... t_m`)`

- Evaluation: `(LAMBDA (X Y) (CONS (CAR X) Y))` of `(A B)` and `(C)`:
  - Assign `X` to be `(A B)`, `Y` to be `(C)`
  - Evaluate form: `(CONS (CAR X) Y)`:
    CONS of EVAL[(CAR X)] and EVAL[Y]
    EVAL[(CAR X)] ⇝ CAR of EVAL[X]
    ⇝ CAR of (A B) ⇝ A
    EVAL[Y] ⇝ (C)
    ⇝ CONS of A and (C)
    ⇝ (A C)

Examples of λ-Expressions I

- `(LAMBDA (X Y) (+ X (* 2 Y))`
  ... of 4 and 15 is
  ... of 1 and 3 is
  ... of () and 15 is

- `(LAMBDA (X Y) (CONS (CAR X) Y))`
  ... of (A B) and (C) is
  ... of (4 Q B) and () is
  ... of A and B is

- `(LAMBDA (X Y) Y)`
  ... of A and B is
  ... of (A 3) and (t B) is
  ... of (4 5 t) and (Fn A B) is

- `(LAMBDA (TEST) (ATOM TEST))`
  ... of A is
  ... of (A) is

Examples of λ-Expressions II

- (LAMBDA (P) (CAR P))
  ... of (A) is
  ... of (FOO A B) is
  ... of A is

- (LAMBDA (X) (CONS (CAR X) X))
  ... of (A B C) is
  ... of () is

- (LAMBDA (X) (CAR (CAR X)))
  ... of ((A) B C) is
  ... of ((t 9) Q) is

- (LAMBDA (X) (CAR (CDR X)))
  ... of ((A) (B) (C)) is
  ... of (CAR (CONS A B)) is

- See p17-18.

Summary: Function, Form

Functions take arguments and return values.

"function + arguments ⇝ returned value"
`CAR` of `(A B C)` is `A`

- Primitive (atomic)
  (*E.g.*, `CONS`, `CAR`, `CDR`, `ATOM`, ...)

- λ-expression: `(LAMBDA (`a_1 ... a_n`)` ⟨form⟩`)`
  (*E.g.*, `(LAMBDA (A B) B)`, ...)

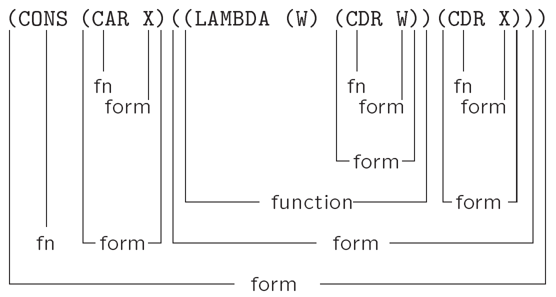
Forms are expressions to evaluate (possibly with variables).

"form + values for variables ⇝ value"
`(CAR X)` `[X = (A B C)]` `A`

- Atom [constant or variable]
  (*E.g.*, `t`, `5`, `X`, ...)

- Function application: `(fn` f_1 ... f_n`)`
  (*E.g.*, `(CAR Z)`, `(CONS 5 (CAR X))`, ...)

Example of FORM and FUNCTION



Notation + Examples

```
(LAMBDA (X Y) (CONS X (CDR (CDR Y)))
```
of A and (B C D) is (A D)

```
(LAMBDA (X Y) (CONS X (QUOTE (CDR Y)))
```
of A and (B C D) is (A CDR Y)

...

→ <u>((LAMBDA (X Y) (CONS X (CDR (CDR Y))) 'A '(B C D))</u>
(A D)
→ <u>((LAMBDA (X Y) (CONS X (QUOTE (CDR Y))) 'A '(B C D))</u>
(A CDR Y)
→ <u>((LAMBDA (X) (CAR X)) (CONS (QUOTE A) NIL))</u>
A
→ <u>((LAMBDA (X) (CAR X)) '(CONS (QUOTE A) NIL))</u>
CONS
→ <u>((LAMBDA (X) (CAR X)) '(LAMBDA (X) (CAR X)))</u>
LAMBDA
→ <u>((LAMBDA (X) (CAR X)) (LAMBDA (X) (CAR X)))</u>
undefined --- ''LAMBDA'' is not function

...

The LET Operator

(LET ((v_1 e_1) ... (v_n e_n)) ⟨form⟩)

is an ABBREVIATION for

((LAMBDA (v_1 ... v_n) ⟨form⟩)
e_1 ... e_n)

→ <u>(LET ((X 'A) (Y '(B C))) (CONS X Y))</u>
(A B C)
→ <u>((LAMBDA (X Y) (CONS X Y)) 'A '(B C))</u>
(A B C)
→ <u>(LET ((X 'A) (Y '(B C D))) (CONS X (CDR (CDR Y))))</u>
(A D)
→ <u>(LET ((X (CONS (QUOTE A) NIL))) (CAR X))</u>
A

Program ≡ Data

CONS is $\left\{ \begin{array}{l} \text{a function of two args} \\ \text{an atom} \end{array} \right\}$

CONS of A and (foo bar B) is
(A foo bar B)

CAR of (A foo bar B) is
A

CONS of CAR and (foo bar CONS) is
(CAR foo bar CONS)

CAR of (CONS foo bar CONS) is
CONS