An Interpreter for a Simple Functional Language (LET)

CS496

LET: A Simple Language – The Concrete Syntax

$$
\begin{array}{lcl}
\langle \mathit{Program} \rangle & ::= & \langle \mathit{Expression} \rangle \\
\\
\langle \mathit{Expression} \rangle & ::= & \langle \mathit{Number} \rangle \\
\langle \mathit{Expression} \rangle & ::= & \langle \mathit{Identifier} \rangle \\
\langle \mathit{Expression} \rangle & ::= & \langle \mathit{Expression} \rangle \texttt{ - } \langle \mathit{Expression} \rangle \\
\langle \mathit{Expression} \rangle & ::= & \text{zero? } (\langle \mathit{Expression} \rangle) \\
\langle \mathit{Expression} \rangle & ::= & \text{if } \langle \mathit{Expression} \rangle \text{ then } \langle \mathit{Expression} \rangle \text{ else } \langle \mathit{Expression} \rangle \\
\langle \mathit{Expression} \rangle & ::= & \text{let } \langle \mathit{Identifier} \rangle = \langle \mathit{Expression} \rangle \text{ in } \langle \mathit{Expression} \rangle \\
\langle \mathit{Expression} \rangle & ::= & (\langle \mathit{Expression} \rangle)
\end{array}
$$

Examples of Programs in LET

Examples of programs in concrete syntax:

- x

Non-examples

Examples of Programs in LET

Examples of programs in concrete syntax:

- x
- 55 - (x - 11)

Non-examples

Examples of Programs in LET

Examples of programs in concrete syntax:

- x
- 55 - (x - 11)
- zero? (55 - (x - 11))

Non-examples

Examples of Programs in LET

Examples of programs in concrete syntax:

- x
- 55 - (x - 11)
- zero? (55 - (x - 11))
- let y = 23 in if zero?(y) then 4 else 6

Non-examples

Examples of Programs in LET

Examples of programs in concrete syntax:

- x
- 55 - (x - 11)
- zero? (55 - (x - 11))
- let y = 23 in if zero?(y) then 4 else 6

Non-examples

- (zero? 55 - (x - 11))

Examples of Programs in LET

Examples of programs in concrete syntax:

- x
- 55 - (x - 11)
- zero? (55 - (x - 11))
- let y = 23 in if zero?(y) then 4 else 6

Non-examples

- (zero? 55 - (x - 11))
- zero 4

Examples of Programs in LET

Examples of programs in concrete syntax:

- x
- 55 - (x - 11)
- zero? (55 - (x - 11))
- let y = 23 in if zero?(y) then 4 else 6

Non-examples

- (zero? 55 - (x - 11))
- zero 4
- 1 + + 2

LET: Abstract Syntax

```
type expr =
  | Var of string
  | Int of int
  | Sub of expr*expr
  | Let of string*expr*expr
  | IsZero of expr
  | ITE of expr*expr*expr
```

Examples in Abstract Syntax

Let't revisit our earlier examples to translate them into the corresponding abstract syntax trees.

- Concrete syntax:

  55 -(x-11)

- Abstract syntax (type `prog`):

```
Sub (Int 55, Sub (Var "x", Int 11))
```

Examples in Abstract Syntax

- Concrete syntax:

  zero? (55 - (x - 11))

- Abstract syntax

```
IsZero (Sub (Int 55, Sub (Var "x", Int 11)))
```

- Exercise: write the abstract syntax tree for this LET expression:
  let y = 23 in if zero?(y) then 4 else 6

Syntax of LET

Specifying the Interpreter

Implementing the Interpreter

Interpreter for Expressions

For each language construction we present two steps:

1. Specification of the interpreter (presented in a blue box in math-like language)

   Specification here!

2. Implementation of the interpreter (presented as a standard box in OCaml)

```
Code here!
```

Recall from ARITH

$$\texttt{Sub(Int 2,Int 1)} \Downarrow 1$$

What about?

$$\texttt{Sub(Var "x",Int 1)} \Downarrow ???$$

In particular, how should we define the rule for lookup?

$$\frac{???}{\texttt{Var(id)} \Downarrow v} \; \mathit{EVar}$$

Environments

- Function whose domain is a finite set of variables and whose range is the expressed values.
- ρ ranges over environments.
- $\rho(var)$ looks up the value of variable *var*
- $[\,]$ denotes the empty environment.

Example:

$$[i := 1, v := 5, x := 10]$$

Evaluation Judgements for LET

$$\mathsf{e}, \rho \Downarrow r$$

- e is an expression in LET
- ρ is an environment
- r is a result:

$$\begin{aligned} \mathbb{R} &:= \mathbb{EV} \cup \{\mathit{error}\} \\ \mathbb{EV} &:= \mathbb{Z} \cup \mathbb{B} \\ \mathbb{B} &:= \{\mathit{true}, \mathit{false}\} \end{aligned}$$

Specifying the Behavior of the Interpreter for Expressions

$$\frac{}{\texttt{Int(n)}, \rho \Downarrow n} \; \textit{EInt}$$

Specifying the Behavior of the Interpreter for Expressions

$$\frac{\rho(\texttt{id}) = v}{\texttt{Var(id)}, \rho \Downarrow v} \; \mathit{EVar}$$

$$\frac{\texttt{id} \notin \mathit{Dom}(\rho)}{\texttt{Var(id)}, \rho \Downarrow \mathit{error}} \; \mathit{EVarErr}$$

Specifying the Behavior of the Interpreter for Expressions

$$\frac{\texttt{e1}\Downarrow m \quad \texttt{e2},\rho\Downarrow n \quad p = m/n}{\texttt{Div(e1,e2)},\rho\Downarrow p}\ \mathit{EDiv}$$

$$\frac{\texttt{e1},\rho\Downarrow m \quad \texttt{e2},\rho\Downarrow 0}{\texttt{Div(e1,e2)},\rho\Downarrow \mathit{error}}\ \mathit{EDivErr}$$

- Recall that we will not be presenting the error propagation rules; they are left implicit
- Also, we omit the rules for `Sub` since they are similar

Specifying the Behavior of the Interpreter for Expressions

$$\frac{\texttt{e}, \rho \Downarrow v \quad v = 0}{\texttt{IsZero(e)}, \rho \Downarrow \mathit{true}} \; \mathit{EIZTrue} \qquad \frac{\texttt{e}, \rho \Downarrow v \quad v \neq 0}{\texttt{IsZero(e)}, \rho \Downarrow \mathit{false}} \; \mathit{EIZFalse}$$

$$\frac{\texttt{e}, \rho \Downarrow v \quad v \notin \mathbb{Z}}{\texttt{IsZero(e)}, \rho \Downarrow \mathit{error}} \; \mathit{EIZErr}$$

Specifying the Behavior of the Interpreter for Expressions

$$\frac{\texttt{e1}, \rho \Downarrow \mathit{true} \qquad \texttt{e2}, \rho \Downarrow v}{\texttt{ITE(e1,e2,e3)}, \rho \Downarrow v} \; \mathit{EITETrue}$$

$$\frac{\texttt{e1}, \rho \Downarrow \mathit{false} \qquad \texttt{e3}, \rho \Downarrow v}{\texttt{ITE(e1,e2,e3)}, \rho \Downarrow v} \; \mathit{EITEFalse}$$

$$\frac{\texttt{e1}, \rho \Downarrow v \qquad v \notin \mathbb{B}}{\texttt{ITE(e1,e2,e3)}, \rho \Downarrow \mathit{error}} \; \mathit{EITEErr}$$

Specifying the Behavior of the Interpreter for Expressions

$$\frac{\texttt{e1}, \rho \Downarrow w \quad \texttt{e2}, \rho \oplus \{\texttt{id} := w\} \Downarrow v}{\texttt{Let(id,e1,e2)}, \rho \Downarrow v} \; \mathit{ELet}$$

$\rho \oplus \{\texttt{id} := w\}$ denotes extension of ρ with new association $\texttt{id} := w$

Summary (1/2)

$$\frac{}{\texttt{Int(n)}, \rho \Downarrow n} \ \textit{EInt}$$

$$\frac{\rho(\texttt{id}) = v}{\texttt{Var(id)}, \rho \Downarrow v} \ \textit{EVar} \qquad \frac{\texttt{id} \notin \textit{Dom}(\rho)}{\texttt{Var(id)}, \rho \Downarrow \textit{error}} \ \textit{EVarErr}$$

$$\frac{\texttt{e1} \Downarrow m \quad \texttt{e2}, \rho \Downarrow n \quad p = m/n}{\texttt{Div(e1,e2)}, \rho \Downarrow p} \ \textit{EDiv} \qquad \frac{\texttt{e1}, \rho \Downarrow m \quad \texttt{e2}, \rho \Downarrow 0}{\texttt{Div(e1,e2)}, \rho \Downarrow \textit{error}} \ \textit{EDivErr}$$

$$\frac{\texttt{e}, \rho \Downarrow v \quad v = 0}{\texttt{IsZero(e)}, \rho \Downarrow \textit{true}} \ \textit{EIZTrue} \qquad \frac{\texttt{e}, \rho \Downarrow v \quad v \neq 0}{\texttt{IsZero(e)}, \rho \Downarrow \textit{false}} \ \textit{EIZFalse}$$

$$\frac{\texttt{e}, \rho \Downarrow v \quad v \notin \mathbb{Z}}{\texttt{IsZero(e)}, \rho \Downarrow \textit{error}} \ \textit{EIZErr}$$

Summary (2/2)

$$\frac{\texttt{e1}, \rho \Downarrow \mathit{true} \quad \texttt{e2}, \rho \Downarrow v}{\texttt{ITE(e1,e2,e3)}, \rho \Downarrow v} \; \mathit{EITETrue} \qquad \frac{\texttt{e1}, \rho \Downarrow \mathit{false} \quad \texttt{e3}, \rho \Downarrow v}{\texttt{ITE(e1,e2,e3)}, \rho \Downarrow v} \; \mathit{EITEFalse}$$

$$\frac{\texttt{e1}, \rho \Downarrow v \quad v \notin \mathbb{B}}{\texttt{ITE(e1,e2,e3)}, \rho \Downarrow \mathit{error}} \; \mathit{EITEErr}$$

$$\frac{\texttt{e1}, \rho \Downarrow w \quad \texttt{e2}, \rho \oplus \{\texttt{id} := w\} \Downarrow v}{\texttt{Let(id,e1,e2)}, \rho \Downarrow v} \; \mathit{ELet}$$

Example

Show that the following judgement is derivable:

$\texttt{Let("x",Int 2,ITE(IsZero(Var "x"),Int 1,Var "x"))} \Downarrow 2$

Syntax of LET

Specifying the Interpreter

Implementing the Interpreter

Interpreter for Expressions

```
eval_expr: expr -> ???
```

- What should the return type of the interpreter be?
  - Since ARITH is a subset of LET, evaluation may produce an error due to division by zero:

```
eval_expr: expr ->??? result
```

- New! We can write programs such zero?(4). Thus

```
eval_expr: expr -> exp_val result
```

where `exp_val` ("expressed values") means boolean or integer

```
type exp_val =
  | NumVal of int
  | BoolVal of bool
```

Interpreter for Expressions – The Need for Environments

- Now that we know the type of the interpreter for expressions

```
eval_expr: expr -> exp_val result
```

  we must move on to defining the interpreter itself
- Before doing so, however, one final observation
- The value of `5-1` should clearly be `Ok (NumVal 4)`
- That of `if zero?(4-4) then 2 else 1` should clearly be `Ok (NumVal 2)`
- What should the value of `x-2` be?

Interpreter for Expressions – The Need for Environments

- What should the value of `x-2` be?
- We need the value of `x` to be able to answer
- Hence we need environments
- The final type of the interpreter is therefore

```
eval_expr: expr -> env -> exp_val result
```

Environments (OCaml)

```
type env =
  | EmptyEnv
  | ExtendedEnv of string*exp_val*env
```

- Two constructors
  - `EmptyEnv`: constructs an empty environment
  - `ExtendedEnv`: extends a previous environment with a new association pair

Example: $[i := 1, v := 5, x := 10]$ in OCaml

```
ExtendEnv("i",NumVal 1,
 ExtendEnv("v",NumVal 5,
  ExtendEnv("x",NumVal 10,
    EmptyEnv)))
```

Implementing Environments

```
let empty_env : unit -> env = fun () ->
  EmptyEnv

let extend_env : env -> string -> exp_val -> env = fun env
  ExtendEnv(id,v,env)

let rec apply_env : env -> string -> exp_val result = fun e
  match env with
  | EmptyEnv ->
    error @@ id^" not found!"
  | ExtendEnv(v,ev,tail) ->
    if id=v
    then return ev
    else apply_env tail id
```

Implementing the Interpreter

$$\frac{}{\texttt{Int(n)} \Downarrow n}\ \textit{EInt}$$

```
let rec eval_expr : expr -> env -> exp_val result = fun e en
  match e with
  | Int n -> return (NumVal n)
```

Implementing the Interpreter

- We must lookup the value of variables in the environment

$$\frac{\rho(\texttt{id}) = v}{\texttt{Var(id)}, \rho \Downarrow v}\ \mathit{EVar}$$

```
  | Var(id) -> apply_env en id
```

Implementing the Interpreter

$$\frac{\texttt{e1}\Downarrow m \quad \texttt{e2}\Downarrow n \quad p = m - n}{\texttt{Sub(e1,e2)}\Downarrow p}\ \mathit{ESub}$$

$$\frac{\texttt{e1}\Downarrow \mathit{error}}{\texttt{Sub(e1,e2)}\Downarrow \mathit{error}}\ \mathit{ESubErr1} \qquad \frac{\texttt{e1}\Downarrow m \quad \texttt{e2}\Downarrow \mathit{error}}{\texttt{Sub(e1,e2)}\Downarrow \mathit{error}}\ \mathit{ESubErr2}$$

```
  | Sub(e1,e2) ->
    eval_expr e1 en >>= fun n1 ->
    eval_expr e2 en >>= fun n2 ->
    return @@ NumVal (n1-n2)
```

Implementing the Interpreter

$$\frac{\mathtt{e}, \rho \Downarrow v \quad v = 0}{\mathtt{IsZero(e)}, \rho \Downarrow \mathit{true}}\ \mathit{EIZTrue} \qquad \frac{\mathtt{e}, \rho \Downarrow v \quad v \neq 0}{\mathtt{IsZero(e)}, \rho \Downarrow \mathit{false}}\ \mathit{EIZFalse}$$

$$\frac{\mathtt{e}, \rho \Downarrow v \quad v \notin \mathbb{Z}}{\mathtt{IsZero(e)}, \rho \Downarrow \mathit{error}}\ \mathit{EIZErr}$$

```
| IsZero(e) ->
  eval_expr e en >>=
  int_of_numVal >>= fun n ->
  return @@ BoolVal (n = 0)
```

Implementing the Interpreter

$$\frac{\texttt{e1}, \rho \Downarrow \mathit{true} \qquad \texttt{e2}, \rho \Downarrow v}{\texttt{ITE(e1,e2,e3)}, \rho \Downarrow v}\ \mathit{EITETrue}$$

$$\frac{\texttt{e1}, \rho \Downarrow \mathit{false} \qquad \texttt{e3}, \rho \Downarrow v}{\texttt{ITE(e1,e2,e3)}, \rho \Downarrow v}\ \mathit{EITEFalse}$$

$$\frac{\texttt{e1}, \rho \Downarrow v \qquad v \notin \mathbb{B}}{\texttt{ITE(e1,e2,e3)}, \rho \Downarrow \mathit{error}}\ \mathit{EITEErr}$$

```
| ITE(e1,e2,e3) ->
  eval_expr e1 en >>=
  bool_of_boolVal >>= fun b ->
  if b
  then eval_expr e2 en
  else eval_expr e3 en
```

`let`-expressions

$$\frac{\texttt{e1}, \rho \Downarrow w \quad \texttt{e2}, \rho \oplus \{\texttt{id} := w\} \Downarrow v}{\texttt{Let(id,e1,e2)}, \rho \Downarrow v}\ \mathit{ELet}$$

```
| Let(v,def,body) ->
  eval_expr def en >>= fun ev ->
  eval_expr body (extend_env en v ev)
```

- Important: static scoping is implemented by extending environments

Summary on LET

- Introduced syntax
- We have specified interpreter
- Implemented the interpreter
- We coin our current implementation, the preliminary version

Next

- We revisit our code for the interpreter, and restructure it further
- This will produce the final version

Further restructuring: avoid passing environment around

```
  | Sub(e1,e2) ->
    eval_expr e1 en >>= fun n1 ->
    eval_expr e2 en >>= fun n2 ->
    return @@ NumVal (n1-n2)
  | ITE(e1,e2,e3) ->
    eval_expr e1 en >>=
    bool_of_boolVal >>= fun b ->
    if b
    then eval_expr e2 en
    else eval_expr e3 en
```

The Interpreter for LET

- Code available in Canvas `Modules/Interpreters`
- Directory `let-lang` (preliminary)
- Compile from root dir with `dune utop` or `dune utop src`
- Type, for eg., `interp "let x=2 in x+3";;` in `utop`