MicroML: Minimal Functional Programming Language, Expressions

< exp > ::= IDENT | BOOL | INT | REAL | STRING | () | [] | {< exp >}
| < exp > andalso < exp > | < exp > orelse < exp >
| let < dec > in < exp > end
| if < exp > then < exp > else < exp >
| fn IDENT → < exp >
| < exp > < infid > < exp > | < prefid > < exp >
| < exp > < exp >

Changes in New World View

- **Simplifications:**
  - Variables are write-once
  - No need to split the state into environment and memory
  - Static scope: use closures to give meaning of function bodies
  - Get functions as first class values just by adding syntax

- **Complications - Recursion:**
  - Didn’t have before with procedures and closures
  - Got “for free” (more or less) with objects and dynamic dispatch
  - Problem: Need something like a closure, but needs to “contain itself”.
  - Solution: Will record enough information to “simulate” this dynamically.

  - **Note:** From here on m + m’ means function m updated by m’ to be.

Expression Rules

- **Constants**
  - (t, m) ⇓ const_to_val(t)
  - (e, m) ⇓ v

- **Parentheses**
  - (f, m) ⇓ v

- **Non-Recursive Variables**
  - m(x) = v  ∀f, y, e, m’. v ≠ ⟨⟨f, y → e, m’⟩⟩
  - (x, m) ⇓ v

- **Recursive Identifiers**
  - m′ = ⟨⟨g, x → e, m’⟩⟩
  - f, m) ⇓ (x → e, m’ + {g → ⟨⟨g, x → e, m’⟩⟩})
Expression Rules

If Expression

\[(e_1, m) \Downarrow \text{true} \quad (e_2, m) \Downarrow v \quad (e_1, m) \Downarrow \text{false} \quad (e_3, m) \Downarrow v\]

if \(e_1\) then \(e_2\) else \(e_3\) \(\Downarrow v\)

Let Expression

\[(d, m) \Downarrow m' \quad (e, m + m') \Downarrow v\]

(\(\text{let} \ d \ \text{in} \ e \ \text{end}, m \Downarrow v\))

Functions

\[(\lambda x \rightarrow e, m) \Downarrow (x \rightarrow e, m)\]

Application

\[(e_1, m) \Downarrow \langle x \rightarrow e', m' \rangle \quad (e_2, m) \Downarrow v' \quad (e', m' + \{x \rightarrow v'\}) \Downarrow v\]

if \(e_1\) \(\Downarrow\) \(v\)

Monadic Operator Application

\[(e, m) \Downarrow v \quad \text{monOpApply}(\text{mon_op}, v) = v' \quad (\text{mon_op } e, m) \Downarrow v'\]

Binary Operator Application

\[(e_1, m) \Downarrow v_1 \quad (e_2, m) \Downarrow v_2 \quad \text{binOpApply}(\oplus, v_1, v_2) = v \quad (e_1 \oplus e_2, m) \Downarrow v\]

Declaration Rules

Val Declaration

\[(e, m) \Downarrow v \quad \text{val} \ x \ = \ e, m \Downarrow \{x \rightarrow v\}\]

Recursive Declarations

\[(\text{fun} \ f \ x \ = \ e, m) \Downarrow \{f \rightarrow (f, x, e, m)\}\]

Seq Declaration

\[(d_1, m) \Downarrow m' \quad (d_2, m + m') \Downarrow m'' \quad (d_1, d_2, m) \Downarrow m' + m''\]

Local Declaration

\[(d_1, m) \Downarrow m' \quad (d_2, m + m') \Downarrow m'' \quad (\text{local} \ d_1 \ \text{in} \ d_2 \ \text{end}, m) \Downarrow m''\]

Where Did the Types Go?

- In SIMP and OOPL, once we had declarations, they told us the types of our variables.
- In MicroML, no typed in declarations?
- Problem: Does that make it impossible to have a static type system then?
- Solution: Infer types from known types of constants and the way expressions are built.
- Start with a type system and type checking
- Use type variables.
- Algorithm W