COMP4233: Functional Programming

Lab 11: Language Interpretation

Dr. Zhiyuan Li

Computer Science and Technology
United International College

Outline

Example 1. Calculator

- Download the package `Lab_11_Template.zip` from iSpace and implement `ast.ml`, `parser.mly`, `lexer.mll`, and `main.ml` for the calculator as how it is introduced in Lec 11.
- Open Opam switch termal (cygwin), enter the project root directory, and execute `make test` to build and execute the test file at `\test\test.ml`.
- If your implementation is correct, all 8 testcases should be passes.

Example 2. Space & Line Break

- Define a new regular expression named `white`, which can be any number of empty spaces `' '` or line breaks `'\t'`.
- Modify the rule `read` to skip any string matching `white`.
- Hint: direct recursive call.

Example 3. Let & Identifier

- Let's create a naming system for our calculator. With this system, users can give names to expression.
- A naming system includes tokens
  - `LET` - binding a new name;
  - `EQ` - bingding operator `=`;
  - `ID` - the name of identifiers; and
  - `IN` - the scope of identifiers.
- To implement this, we need to
  - create a new `expr` AST node as a tripple - a string (the name of an identifier), an expression (the definition of the name), and another expression (the scope);
  - define the corresponding tokens in `parser.mly`;
  - modify `read` in `lexer.mll` to let the lexer recognize identifiers; and
  - implement the grammar $E \rightarrow \mathit{let}\ \mathit{id} = E$ as BNF in `parser.mly`.

Example 4. Substitution

- After upgrading lexer and parser, evaluator also needs an upgrade.
- Two models, **subtitution model** and **environmental model** are designed for this. We introduce subtitution model in this lab.
- For let expression `let <x> = <e1> in <e2>`,
  1. evaluate `<e1> ==> v`;
  2. **substitude** every `<x>` in `<e2>` by `<v>`;
  3. continuge evaluation.
- The notation for substitution is

```
let <x> = v in <e> --> <e>{v/<x>}
```

Example 4. Substitution

- Small-step substitution can be

```
1   (<e1>+<e2>){v/<x>} --> <e1>{v/<x>}+<e2>{v/<x>}
2   (<e1>*<e2>){v/<x>} --> <e1>{v/<x>}*<e2>{v/<x>}
3   <y>{v/<x>} --> <y>
4   <x>{v/<x>} --> v
```

- `let` expressions cannot be naïvely substituted.
- For example, `let x = 1 in (let x = 2 in x) + x`
- Correct subsitution is

```
1   ((let x = 2 in x) + x){1/x} -->*
2     (let x = 2 in x) + x{1/x}
```

- Wrong subsitution is

```
1   ((let x = 2 in x) + x){1/x} -->*
2     (let x = 2 in x{1/x}) + x{1/x}
```

Example 4. Substitution

- Thus, substitution for `let` is

```
1   (let <x> = <e>){v/<y>} -->
2     if <x> = <y> then
3       (let <x> = <e>)
4     else
5       (let <x> = <e>{v/<y>})
```

- Please implement the above evaluation in `main.ml`.

Exercises

- The exercise for Lab 11 & 12 is a language implementation.
- See "COMP4233_25S_PA.pdf" for details.

End of Lab 11