Lec 17: Interpreter

CS220: Programming Principles

Sang Kil Cha
Interpreter (2)
Evaluation Step

Interpreter “evaluates” a list of ASTs to derive a value and update the context.
From an AST to a Value

Parse: Grammar $\rightarrow$ string$^1$ $\rightarrow$ AST

Evaluate: AST $\rightarrow$ Context $\rightarrow$ Context $\ast$ Value

★ Recall our AddSubLang can only return an int as a value.

$^1$Grammar was built-in to our parser.
Context in AddSubLang?

Is there a certain context to consider? In other words, can an expression in AddSubLang be interpreted in a different manner depending on a context?
Implementing an Interpreter

F# is perfect for implementing interpreters (and compilers). Why?

1. Representing an AST (a tree) is natural and easy.
2. Pattern matching on an AST can make interpreter concise.
3. Descendant of ML (Meta Language) 😊.
Interpreter for AddSubLang

```javascript
let evaluate = function
| Number n -> n // This is just a value
| Add (a, b) -> // ?
| Sub (a, b) -> // ?
```

How can we handle nested expressions?
Recursion!

``` Ocaml 
let rec evaluate = function 
| Number n -> n 
| Add (a, b) -> 
  let a' = evaluate a 
  let b' = evaluate b 
  a' + b' 
| Sub (a, b) -> // ? 
  let a' = evaluate a 
  let b' = evaluate b 
  a' - b' 
```
Interpreting a Sequence of Statements

We need to maintain a context to correctly handle multiple statements. For example, the following code has three statements to evaluate.

```plaintext
let x = 1
let y = 2
x + y
```

After evaluating the first two statements, we need to remember that the symbol `x` and `y` corresponds to a value 1 and 2, respectively.
let rec evalExpr ctxt = function
| Number n -> n
| ...

and rec evalStmt ctxt = function // mutually recursive
| Let (v, e) ->
  let e' = evalExpr ctxt e
cxtx' // update the cxtxt to have a mapping v to e'
| Expression e ->
  evalExpr ctxt e

let run program =
  let initialContext = // empty map (symbols to values).
  program
  |> List.fold evaluate initialContext
Conclusion
Further Readings

- The Wizard Book Ch. 4.1 (and the rest of Ch. 4).
Question?