Miniproject 3 – Static Scoping and Recursion

EECS 662 - Programming Languages

Spring 2013

The objective of this project is to add static scoping and recursion to CFWAE. you will add a `rec` construct to CFWAE creating the new language CFWAER. You will also add closures and a value construct to implement static scoping.

Exercise 1 *In this exercise you will modify your CFWAE interpreter to use static scoping. You will add a new type for values that includes closures as well as numbers and modify evaluation of* `fun` *and* `app` *to use the new closure construct.*

1. *Define a datatype to represent values returned by interpreting CFWAE expressions. This datatype should minimally include closures and numeric values.*
2. *Write a function,* `(parse-cfwae expr)` *that takes a CFWAE expression and generates a CFWAE ASE structure that represents it.*
3. *Write a function,* `(interp-cfwae expr env)` *that interprets* `expr` *using the deferred substitution list provided in* `env`*. This interpreter needs to return a value rather than a CFWAE expression and should implement static scoping using closures.*
4. *Write a function,* `(eval-cfwae expr)` *that combines your parser and interpreter for CFWAE into a single evaluation function.*

CFWAE ::=	*number* | *id* |
	`{` `+` *CFWAE CFWAE* `}` |
	`{` `-` *CFWAE CFWAE* `}` |
	`{` $*$ *CFWAE CFWAE* `}` |
	`{` `/` *CFWAE CFWAE* `}` |
	`{` `fun` `{` *id* `}` *CFWAE* `}` |
	`{` *CFWAE CFWAE* `}` |
	`{` `if0` *CFWAE CFWAE CFWAE* `}` |
	`{` `with` `{` *id CFWAE* `}` *CFWAE* `}` |

Table 1: Grammar for CFWAE where *id* is an identifier and *number* is an integer.

Exercise 2 *In this exercise you will modify your interpreter from the previous problem to create a parser for the CFWAER language shown in table 2. CFWAER is identical to CFWAE with a* `rec` *construct added for recursive definitions. Note that the* `rec` *syntax is identical to* `with` *with only a keyword change.*

You will also need to modify your `DfrdSub` *data type and associated search functions to include the recursive* `aRecSub` *declaration. This is a trivial change requiring only the addition of the constructor to* `Env` *as described in your text.*

1. *Define a datatype representing the abstract syntax for CFWAER including the new syntax for recursive functions. Your new type will be almost identical to the CFWAE type.*
2. *Write a function,* `(parse-cfwaer expr)`*, that accepts the concrete syntax of CFWAER and generates a CFWAER data structure representing it. If your parser fails, it should throw an error using the* `error` *function discussed in class.*

CFWAER ::=	*number* | *id* |
	`{` `+` *CFWAER CFWAER* `}` |
	`{` `-` *CFWAER CFWAER* `}` |
	`{` $*$ *CFWAER CFWAER* `}` |
	`{` `/` *CFWAER CFWAER* `}` |
	`{` `fun` `{` *id* `}` *CFWAER* `}` |
	`{` *CFWAER CFWAER* `}` |
	`{` `if0` *CFWAER CFWAER CFWAER* `}`
	`{` `with` `{` *id CFWAER* `}` *CFWAER* `}` |
	`{` `rec` `{` *id CFWAER* `}` *CFWAER* `}` |

Table 2: Grammar for CFWAER where *id* is an identifier and *number* is an integer.

3. *Write a function, `(interp-cfwaer expr env)`, that accepts a CFWAER AST structure and a deferred substitution list and produces a CFWAER value. The interpreter should use static scoping and implement recursion in the deferred substitution list.*

4. *Write a function, `(eval-cfwaer expr)` that combines your parser, elaborator, type checker and interpreter into a single operation that parses, elaborates and interprets a CFWAER expression.*

Notes

This project looks long. It's really not. Most of the changes aside from the recursive function processing are trivial. Don't be intimidated and just do things step-by-step. Add static scoping in Exercise 1 first, then worry about adding recursion. There are numerous examples in the text and we will cover all the code in class.

THIS MINIPROJECT IS LOOSELY BASED ON Shriram Krishnamurthy's homework assignments associated with *Programming Languages: Application and Interpretation*. Any errors in the writeup are exclusively mine.