Lab: Extending the GENERATORS language
CSC 312P, “Programming language implementation: the good parts”
Department of Computer Science · Grinnell College
November 9, 2015

Exercises

A. Create a subdirectory called GENERATORS-LAB in the directory you’re using for labs in this course and copy my implementation of the GENERATORS language into it. There are seven language-specific module in all, and you can pick them up from the /home/stone/courses/languages/code/GENERATORS directory.

- tokens.scm
- syntax-trees.scm
- expvals-and-environments.scm
- stores.scm
- scanner.scm
- parser.scm
- interpreter.scm

B. GENERATORS is an extension of the IMPLICIT-REFS language described in section 4.3 of the textbook, with the addition of six new kinds of expressions, adapted from Icon, with the following grammar:

```plaintext
<expression> ::= print ( <expression> )
| every <expression> do <expression>
| from <expression> to <expression>
| either <expression> {or <expression>}* end
| together <expression> {and <expression>}* end
| fail
```

A print-expression writes the value of its subexpression to standard output (as a side effect). The value of the print-expression is 1.

An every-expression generates the values of its first subexpression. After each successful generation, it evaluates its second subexpression. When the first subexpression fails, the entire every-expression fails. The second subexpression of an every-expression is bounded; it cannot be resumed after it fails (though it can be re-evaluated from scratch on the next iteration of the loop).

A from-expression generates the integers in the range from the value of its first subexpression up to and including the value of its second subexpression. It is an error if the value of either subexpression is not an integer.

An either-expression generates the values generated by all of its subexpressions, in order. When any of the subexpressions can produce no more values, the either-expression continues with the next subexpression. When there are no more subexpressions, it fails.

A together-expression generates the values that its last subexpression takes on for each combination of successful evaluations of all of its subexpressions. It fails when those combinations are exhausted.

A fail-expression always fails and never yields a value.

Write and a GENERATORS program that displays the sum of any pair of numbers in the range from 0 to 13 that differ by 3, 5, or 7.
C. A not-expression in Icon fails when its subexpression generates at least one value and succeeds when its subexpression fails.

Add a not-expression to GENERATORS. When it succeeds, it should generate the value 143 (once). The subexpression of a not-expression should be bounded. (In other words, it should not be resumed when the not-expression fails!)

D. A while-expression in Icon has the syntax

\[
\text{while } \langle \text{expression} \rangle \text{ do } \langle \text{expression} \rangle
\]

When a while-expression is evaluated, its first subexpression is evaluated first. If this subexpression succeeds, the second subexpression of the while-expression is evaluated, and then the first subexpression is evaluated again (from the beginning — re-evaluated, not resumed). If and when the first subexpression fails, the entire while-expression fails.

Add a while-expression to GENERATORS. Note that both subexpressions of a while-expression are bounded: Goal-directed evaluation can occur within either subexpression, but when either of them fails, the other is not resumed.