CS 131 Programming Languages LAB

September 25, 2009

TA: Jerry Weng, jtweng@cs.ucla.edu, Boelter Hall 4428.

Homework 1

1. Random_chooser (1 hr)
2. Combine_chooser (1 hr)
3. random_sentence (7 hrs)
4. Test cases (1 hr)

Random Chooser

Pseudo-random number generator: for a certain seed, produces the same number.

Input: an integer and type A. Output: an integer and type A.

Let `dumb_chooser s n = (s mod n, (s * 7919+39) mod 65521)` where s = seed and n = maximum value.

```
# dumb_chooser 1 10 ;;
- int * int = (1,7958)
# dumb_chooser 7958 10 ;;
- int * int = (8,3760)
```

Combine Chooser

Suppose we have two random number generators, dumb_chooser and dumb_chooser_2. Perhaps the first chooser takes a string as the seed and the second chooser takes an integer list as the seed. Combining the two results and modding them is one way to get a good random value if either of the choosers is suspected as unfair. In this assignment, the operation to perform on the output A from dumb_chooser and output B from dumb_chooser_2 is `|A - B| mod 1000`.

```
# x = (“abc”, [1:2:3]) 10
= val: X type: Y = (7, (“def”, [4:5:6])) // (A, B)
# let x = combine_chooser dumb_chooser dumb_chooser
# combine_chooser x dumb_chooser
```

Don’t use any assignment operators or iterations in the code; work should be done through recursion.

OCaml Syntax

```
# let add x y = x + y
-: int -> (int -> int)
# let add3 = add 3
-: int -> int = fun
# add3 7
-: val it = 10

# let addxy x y = x + y
-: int -> int

# let x = random_add3 dumb_chooser
-: type ((A, n) -> (n, A)) -> ((A, n) -> (n, A))

# let makelist x = (x, [x:x])
-: val 'a -> 'a * 'a list
# makelist 3

type ('n, 't) symbol = | N of 'n | T of 't
```

```
(Expr, function
       | Expr -> [[;;];[;;]]
       | Lvalue -> [[;;;];[;;];[;]]
# T 3 ;;
-: ('a, int) symbol
# N "123" ;;
-: (string, 'a) symbol

(Expr, (function
       | Expr ->
              [      [T "(" ; N Expr ; N Expr];
                     [N Num]:
                     [T "++"]
              ]
       | Lvalue -> [[ ]]
```

awk_nonterminal: the type of all these things. each of them is referred to as non-terminal.

```
List nth [1:2:3:4:5] 2 1;
-: int = 3
List length [1:2:5]
-: int = 3
let awksub_test0 =
       random_sentence awksub_grammar dumb_chooser 10)
-: = ["++" ; "$" ; "4"]

let max (a,b) = if a > b then a else b
let max = function
       | (a,b) -> if a > b then a else b
let max m = match m with
       | (a,b) -> if a > b then a else b

let rec reverse list = match list with
       | [ ] -> [ ]
       | [a] -> [a]
       | head::tail -> (reverse tail)@[head]
       // 3   [4:5]
```