Programming Languages and Techniques (CIS120e)

Lecture 9
Sep 27, 2010

An Abstract Stack Machine
Announcements

• Homework 3 is available on the web.
  – Due Wednesday at 11:59:59pm
  – Practice with BSTs, generic functions, abstract datatypes

• Midterm 1 will be in class on Friday, October 15th.
Thought Exercise

let x = 3
let f (y:int) : int = x + y
let g (z:int) : int = f(z+1) * x
let a = g x * g x

What value does the variable a get bound to?

How did you know that?
Abstract Machines

• The job of a programming language is to provide some abstraction of the underlying hardware

• We can think of this abstraction as a thing in its own right — an abstract machine

• There are lots of ways of visualizing abstract machines. We’ll look at one today: A stack machine

• This model...
  • is a good way of understanding how recursive functions work
  • gives an accurate picture of how OCaml data structures are shared internally (which helps predict how fast programs will run), and
  • will extend smoothly to include imperative features (assignment, pointer manipulation) and objects
Stack Machine

• Three “spaces”
  – workspace
    • contains the expression we are currently working on simplifying
  – stack
    • temporary storage for remembering bindings and partially simplified expressions
  – heap
    • storage area for large data structures

• Initial state:
  – workspace contains whole program
  – stack and heap empty
Example

```
let x = 3
let f (y:int) : int = x + y
let g (z:int) : int = f(z+1) * x
let a = g x * g x
```
Example

For simplicity, let’s change the top-level declarations into a single expression with several local bindings.

(Note that it is legal to locally declare functions.)

```
let x = 3 in
let f (y:int) : int = x + y in
let g (z:int) : int = f(z+1) * x in
g x * g x
```
let f(n:int) : int = 
  if n = 0 then 1 else n * (f (n-1)) in
f 4
Example

type 'a list =
  | Nil
  | Cons of 'a * ('a list)

let rec length (l: 'a list) : int =
  begin match l with
  | Nil -> 0
  | Cons(_, rest) -> 1 + length rest
  end in

let mylist = Cons(1, Cons(2, Cons(3, Nil))) in

length mylist
let append (l1: 'a list) (l2: 'a list) : 'a list =
begin match l1 with
| Nil -> l2
| Cons(h, t) -> Cons(h, append t l2)
end in

let a = Cons(1, Nil) in
let b = Cons(2, Cons(3, Nil)) in

append a b
type 'a tree =
  | Empty
  | Node of ('a tree) * 'a * ('a tree)

let rec insert (t:'a tree) (n:'a) : 'a tree =
begin match t with
  | Empty -> Node(Empty,n,Empty)
  | Node(lt,x,rt) ->
      if x = n then t
      else if n < x then Node(insert lt n, x, rt)
      else Node(lt, x, insert rt n)
end in

let a = Node(Node(Empty, 2, Empty),
  4,
  Node(Empty, 6, Empty)) in
insert a 3