Bindex: Naming, Free Variables, and Environments

CS251 Programming Languages
Fall 2016, Lyn Turbak

Department of Computer Science
Wellesley College

A New Mini-Language: Bindex

Bindex adds variable names to Intex in two ways:

1. The arguments of Bindex programs are expressed via variable names rather than positionally. E.g.:

```
(bindex (a b) (/ (+ a b) 2))

(bindex (a b c x) (+ (* a (* x x)) (+ (* b x) c)))
```

2. Bindex has a local naming construct `(bind I_defn E_defn E_body)` that behaves like Racket's `(let {[I_defn E_defn]} E_body)`

```
(bindex (p q)
  (bind sum (+ p q)
    (/ sum 2)))

(bindex (a b)
  (bind a_sq (* a a)
    (bind b_sq (* b b)
      (bind numer (+ a_sq b_sq)
        (bind denom (- a_sq b_sq)
          (/ numer denom))))))
```

```
(bindex (x y)
  (+ (bind a (/ y x)
       (bind b (- a y)
         (* a b)))
     (bind c (bind d (+ x y)
               (* d y))
       (/ c x))))
```

Can use `bind` in any expression position

Bindex REPL Interpreter in action

REPL = Read/Eval/Print Loop. Our goal is to see how this all works.

```
- BindexEnvInterp.repl();

bindex> (+ (/ 6 3) ( * 5 8))
42

bindex> (bind a (+ 1 2) (bind b ( * a 5) (- a b)))
~12

bindex> (#args (num 5) (p 10) (q 8))

bindex> ( * (- q num) p)
30

bindex> (#run (bindex (x y) (+ ( * x x) ( * y y))) 3 4)
25

bindex> (#run (bindex (a b) (bind sum (+ a b) (/ sum 2))) 5 15)
10

bindex> (#quit)
Morituras te saluto!
val it = () : unit
```

Try it out:
`~wx/sml/bindex/BindexEnvInterp.sml`

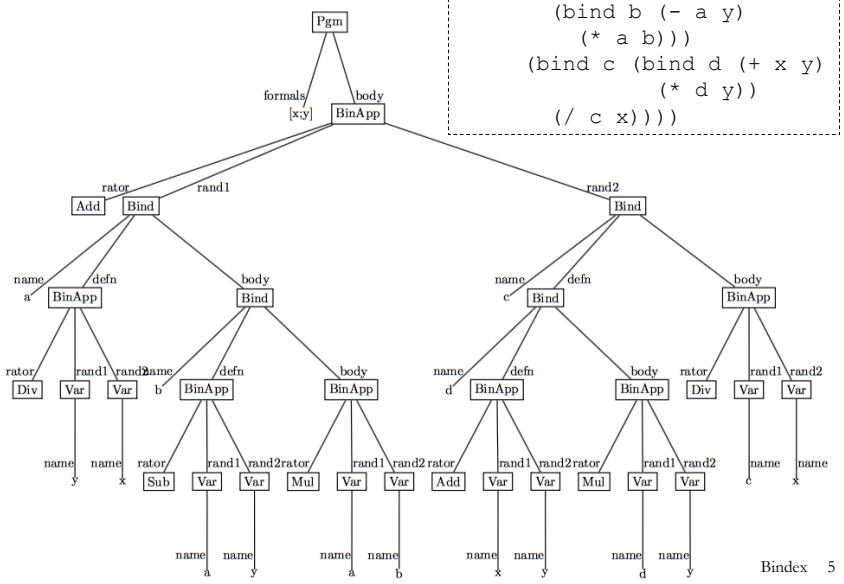
Bindex Abstract Syntax

```
type var = string (* introduce var as synonym for string *)

datatype pgm = Bindex of string list * exp (* param names, body *)

and exp = Int of int (* integer literal with value *)
        | Var of var (* variable reference *)
        | BinApp of binop * exp * exp
          (* binary application of rator to rand1 & rand2 *)
        | Bind of var * exp * exp
          (* bind name to value of defn in body *)

and binop = Add | Sub | Mul | Div | Rem (* binary arithmetic ops *)

val stringToExp : string -> exp
val stringToPgm : string -> pgm
val expToString : exp -> string
val pgmToString : pgm -> string
```

```
- Bindex.stringToPgm "(bindex (a b) (bind sum (+ a b) (/ sum 2)))"
val it =
  Bindex
    (["a","b"],
     Bind ("sum",BinApp (Add,Var "a",Var "b"),
                 BinApp (Div,Var "sum",Int 2))) : Bindex.pgm
```

Bindex AST example

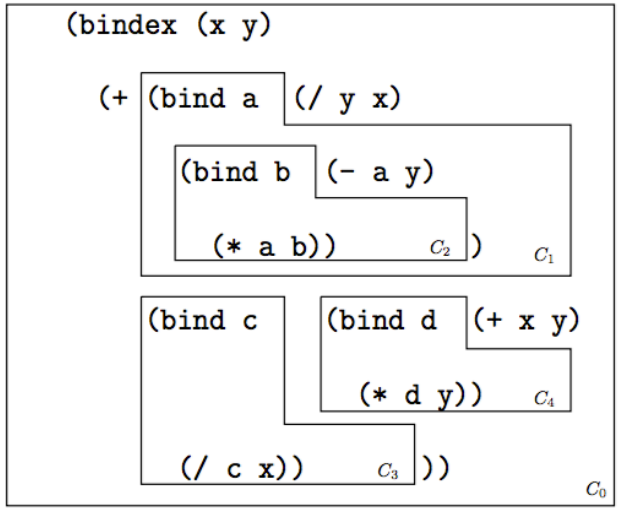
```
(bindex (x y)
  (+ (bind a (/ y x)
       (bind b (- a y)
         (* a b)))
     (bind c (bind d (+ x y)
               (* d y))
       (/ c x))))
```



Calculating Free Variables in Bindex

Bindex Phrase P	Free Variables: FV(P)
L (integer literal)	$\{\}$
I (variable reference)	$\{I\}$
$(O_{rator}\ E_{rand1}\ E_{rand2})$	$FV(E_{rand1}) \cup FV(E_{rand2})$
(bind I E_{defn} E_{body})	$FV(E_{defn}) \cup (FV(E_{body}) - \{I\})$
(bindex (I_1 ... I_n) E_{body})	$FV(E_{body}) - \{I_1 \ldots I_n\}$

Bindex Lexical Contours and Free Variables



String sets

```
signature STRING_SET =
sig
  type t (* The type of a string set *)
  val empty : t
  val singleton : string -> t
  val isEmpty : t -> bool
  val size : t -> int
  val member : string -> t -> bool
  val insert : string -> t -> t
  val delete : string -> t -> t
  val union : t -> t -> t
  val intersection : t -> t -> t
  val difference : t -> t -> t
  val fromList : string list -> t
  val toList : t -> string list
  val toPred : t -> (string -> bool)
  val toString : t -> string
end

structure StringSetList :> STRING_SET = struct
  (* See ~wx/sml/utils/StringSet.sml for details *)
end
```

Bindex: Code for handling free variables

```
structure S = StringSetList

(* val freeVarsPgm : pgm -> S.t *)
(* Returns the free variables of a program *)
fun freeVarsPgm (Bindex(fmls,body)) =
  S.difference (freeVarsExp body) (S.fromList fmls)

(* val freeVarsExp : exp -> S.t *)
(* Returns the free variables of an expression *)
and freeVarsExp (Int i) = S.empty
  | freeVarsExp (Var name) = S.singleton name
  | freeVarsExp (BinApp(_,rand1,rand2)) =
    S.union (freeVarsExp rand1) (freeVarsExp rand2)
  | freeVarsExp (Bind(name,defn,body)) =
    S.union (freeVarsExp defn)
            (S.difference (freeVarsExp body) (S.singleton name))

(* val freeVarsExps : exp list -> S.t *)
(* Returns the free variables of a list of expressions *)
and freeVarsExps exps =
  foldr (fn (s1,s2) => S.union s1 s2) S.empty (map freeVarsExp exps)

(* val varCheck : pgm -> bool *)
and varCheck pgm = S.isEmpty (freeVarsPgm pgm)
```


Environments

```
signature ENV = sig
  type 'a env
  val empty: 'a env
  val bind : string -> 'a -> 'a env -> 'a env
  val bindAll : string list -> 'a list -> 'a env -> 'a env
  val make : string list -> 'a list -> 'a env
  val lookup : string -> 'a env -> 'a option
  val map: ('a -> 'a) -> 'a env -> 'a env
  val remove : string -> 'a env -> 'a env
  val removeAll : string list -> 'a env -> 'a env
  val merge : 'a env -> 'a env -> 'a enve
end

structure Env :> ENV = struct
  (* See ~wx/sml/utils/Env.sml for details *)
end
```


Bindex Interpreter

```
open Bindex
exception EvalError of string

(* val run : Bindex.pgm -> int list -> int *)
fun run (Bindex(fmls,body)) ints =
  let val flen = length fmls
  val ilen = length ints
  in if flen = ilen then
       eval body (Env.make fmls ints)
     else
       raise (EvalError ("Program expected " ^ (Int.toString flen)
                         ^ " arguments but got " ^ (Int.toString ilen)))
  end

(* val eval : Bindex.exp -> int Env.env -> int *)
and eval (Int i) env = i
  | eval (Var name) env =
      (case Env.lookup name env of
         SOME(i) => i
       | NONE => raise (EvalError("Unbound variable: " ^ name)))
  | eval (BinApp(rator,rand1,rand2)) env =
      (binopToFun rator)(eval rand1 env, eval rand2 env)
  | eval (Bind(name,defn,body)) env =
      eval body (Env.bind name (eval defn env) env)

(* val binopToFun : Bindex.binop -> (int * int) -> int *)
(* This is unchanged from the Intex interpreter *)
```


Extending Bindex: Sigmex = Bindex + `sigma`

(sigma I_{var} E_{lo} E_{hi} E_{body}**)**
Assume that I_{var} is a variable name, E_{lo} and E_{hi} are expressions denoting integers that are not in the scope of I_{var}, and E_{body} is an expression that is in the scope of var. Returns the sum of E_{body} evaluated at all values of the index variable I_{var} ranging from the integer value of E_{lo} up to the integer value of E_{hi}, inclusive. This sum would be expressed in traditional mathematical summation notation as:

$$\sum_{I_{var}=E_{lo}}^{E_{hi}} E_{body}$$

If the value of E_{lo} is greater than that of E_{hi}, the sum is 0.

Sigmex: `sigma` examples

Mathematical Notation	BINDEX Notation	Value
$\sum_{i=3}^{7} i$	`(sigma i 3 7 i)`	$3+4+5+6+7=25$
$\sum_{j=1+2}^{2*3} j^2$	`(sigma j (+ 1 2) (* 2 3) (* j j))`	$3^2+4^2+5^2+6^2=86$
$\sum_{j=5}^{1} j^2$	`(sigma j 5 1 (* j j))`	0
$\sum_{i=2}^{5}\sum_{j=i}^{4} i \cdot j$	`(sigma i 2 5 (sigma j i 4 (* i j)))`	$2\cdot 2+2\cdot 3+2\cdot 4+3\cdot 3+3\cdot 4+4\cdot 4=55$
$\sum_{i=\sum_{k=1}^{3} k^2}^{\sum_{j=1}^{5} j} i$	`(sigma i (sigma k 1 3 (* k k)) (sigma j 1 5 j) i)`	$\sum_{i=(1^2+2^2+3^2)}^{1+2+3+4+5} = \sum_{i=14}^{15} = 14+15 = 29$