1 Library functions

1.1 LIST

```
datatype 'a list = nil | :: of 'a * 'a list
exception Empty

val null : 'a list -> bool
val length : 'a list -> int
val @ : 'a list * 'a list -> 'a list
val hd : 'a list -> 'a
val tl : 'a list -> 'a list
val last : 'a list -> 'a
val getItem : 'a list -> ('a * 'a list) option
val nth : 'a list * int -> 'a
val take : 'a list * int -> 'a list
val drop : 'a list * int -> 'a list
val rev : 'a list -> 'a list
val concat : 'a list list -> 'a list
val revAppend : 'a list * 'a list -> 'a list
val app : ('a -> unit) -> 'a list -> unit
val map : ('a -> 'b) -> 'a list -> 'b list
val mapPartial : ('a -> 'b option) -> 'a list -> 'b list
val find : ('a -> bool) -> 'a list -> 'a option
val filter : ('a -> bool) -> 'a list -> 'a list
val partition : ('a -> bool)
-> 'a list -> 'a list * 'a list
val foldl : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
val foldr : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
val exists : ('a -> bool) -> 'a list -> bool
val all : ('a -> bool) -> 'a list -> bool
val tabulate : int * (int -> 'a) -> 'a list
    (* [f(0), f(1), ..., f(n-1)] *)
```

1.2 STRING

```
val size : string -> int
val substring : string * int * int -> string
val ^ : string * string -> string (* string @ *)
val concat : string list -> string
val concatWith : string -> string list -> string
val str : char -> string
val implode : char list -> string
val explode : string -> char list
val map : (char -> char) -> string -> string
val isPrefix    : string -> string -> bool
val isSubstring : string -> string -> bool
val isSuffix    : string -> string -> bool
```

1.3 CHAR

```
val minChar : char
val maxChar : char
val maxOrd : int

val ord : char -> int
val chr : int -> char
val succ : char -> char
val pred : char -> char

val contains : string -> char -> bool
val notContains : string -> char -> bool

val isAscii : char -> bool
val toLower : char -> char
val toUpper : char -> char
val isAlpha : char -> bool
val isAlphaNum : char -> bool
val isCntrl : char -> bool
val isDigit : char -> bool
val isGraph : char -> bool
val isHexDigit : char -> bool
val isLower : char -> bool
val isPrint : char -> bool
val isSpace : char -> bool
val isPunct : char -> bool
val isUpper : char -> bool
```

2 Datatypes

```
'a option = SOME of 'a | NONE
'a MyList = Cons of 'a * 'a MyList | Nil
'a Stream = SCons of 'a * 'a Stream
'a rlist = RCons of 'a * (('a rlist) ref) | RNil
'a Church = Church of ('a -> 'a) -> 'a -> 'a
```

3 Examples

```
fun make_account (opening_balance: int) =
let
  val balance = ref opening_balance
in
  fn (trans: transactions) => case trans
    of Withdraw(a) =>
       ((balance := !balance-a); !balance)
     | Deposit(a) =>
       ((balance := !balance+a); !balance)
     | Check_balance => (!balance)
end

datatype 'a tree = Empty
                 | Node of 'a tree * 'a * 'a tree
exception FoundSoFar of int list

fun findAll'(p, Empty) =
    raise FoundSoFar []
  | findAll'(p, Node(L, d, R)) =
    findAll'(p, L)
    handle FoundSoFar ll => (findAll'(p,R)
    handle FoundSoFar lr =>
      if (p d)
        then raise FoundSoFar (ll@[d]@lr)
        else raise FoundSoFar (ll@lr))

fun findAll(p, T) =
    (findAll'(p, T)
     handle FoundSoFar l => l)

fun findAllCont' p Empty cont = cont []
  | findAllCont' p (Node(L,d,R)) cont =
    findAllCont' p L (fn ll =>
      findAllCont' p R (fn lr =>
        if p(d)
          then cont (ll@[d]@lr)
          else cont (ll@lr)))

(* Notice the similarities with the exceptions *)
```

```
fun findAllCont p T = findAllCont' p T (fn l => l)
```

4 Environment diagrams

```
let val x = 5
  val r = ref(10+x)
  val x = 1
  fun foo x = let val x = 10 in !r + x end
  val _ = (r := x+1)
in
  foo(x + !r)
end
```

5 Language design

Definition 1. *Successful evaluation of an expression produces a value. This is called* value soundness*.*

Definition 2. *The evaluation result of an expression is unique. This is called* determinancy*.*

Definition 3. *Denote by* $[e'/x]e$ *the result of replacing all free occurrences of* x *in* e *with* e'*.*

Definition 4. *We write* $e \downarrow v$ *to say that* e *evaluates to* v*.*

$$\frac{e_1 \downarrow v_1, [v_1/x]e_2 \downarrow v}{\text{let } x = e_1 \text{ in } e_2 \downarrow v}$$

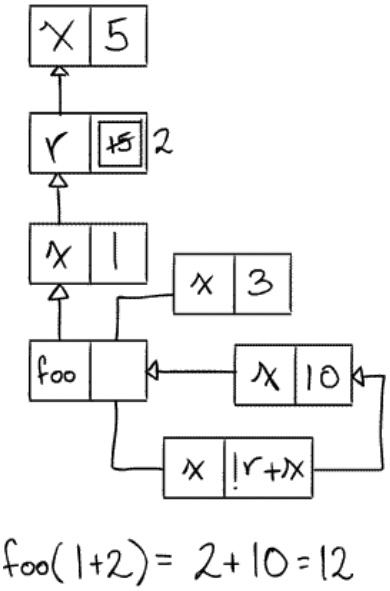

Figure 1: A horrible function's environment diagram.