Chapter 13

Untyped semantics of ASL programs

In this section, we give a semantic treatment of ASL programs. We will use *dynamic typechecking*, i.e. we will test the type correctness of programs during their interpretation.

13.1 Semantic values

We need a type for ASL semantic values (representing results of computations). A semantic value will be either an integer, or a Caml functional value from ASL values to ASL values.

```
#type semval = Constval of int
#            | Funval of (semval -> semval);;
Type semval defined.
```

We now define two exceptions. The first one will be used when we encounter an ill-typed program and will represent run-time type errors. The other one is helpful for debugging: it will be raised when our interpreter (semantic function) goes into an illegal situation.

The following two exceptions will be raised in case of run-time ASL type error, and in case of bug of our semantic treatment:

```
#exception Illtyped;;
Exception Illtyped defined.

#exception SemantBug of string;;
Exception SemantBug defined.
```

We must give a semantic value to our basic functions (equality and arithmetic operations). The next function transforms a Caml function into an ASL value.

```
#let init_semantics caml_fun =
#    Funval
#      (function Constval n ->
#         Funval(function Constval m -> Constval(caml_fun n m)
#                       | _ -> raise Illtyped)
#              | _ -> raise Illtyped);;
init_semantics : (int -> int -> int) -> semval = <fun>
```

Now, associate a Caml Light function to each ASL predefined function:

```
#let caml_function = function
#    "+" -> prefix +
#  | "-" -> prefix -
#  | "*" -> prefix *
#  | "/" -> prefix /
#  | "=" -> (fun n m -> if n=m then 1 else 0)
#  | s -> raise (SemantBug "Unknown primitive");;
caml_function : string -> int -> int -> int = <fun>
```

In the same way as, for parsing, we needed a global environment from which the binding depth of identifiers was computed, we need a semantic environment from which the interpreter will fetch the value represented by identifiers. The global semantic environment will be a reference on the list of predefined ASL values.

```
#let init_sem =  map (fun x -> init_semantics(caml_function x))
#                    init_env;;
init_sem : semval list =
 [Funval <fun>; Funval <fun>; Funval <fun>; Funval <fun>; Funval <fun>]

#let global_sem = ref init_sem;;
global_sem : semval list ref =
 ref [Funval <fun>; Funval <fun>; Funval <fun>; Funval <fun>; Funval <fun>]
```

13.2 Semantic functions

The semantic function is the interpreter itself. There is one for expressions and one for declarations. The one for expressions computes the value of an ASL expression from an environment `rho`. The environment will contain the values of globally defined ASL values or of temporary ASL values. It is organized as a list, and the numbers representing variable occurrences will be used as indices into the environment.

```
#let rec nth n = function
#     []  -> raise (Failure "nth")
#  | x::l -> if n=1 then x else nth (n-1) l;;
nth : int -> 'a list -> 'a = <fun>

#let rec semant rho =
#  let rec sem = function
#      Const n -> Constval n
#    | Var(n) -> nth n rho
#    | Cond(e1,e2,e3) ->
#        (match sem e1 with Constval 0 -> sem e3
#                         | Constval n -> sem e2
#                         | _ -> raise Illtyped)
#    | Abs(_,e') -> Funval(fun x -> semant (x::rho) e')
```

```
#    | App(e1,e2) -> (match sem e1
#                        with Funval(f) -> f (sem e2)
#                           | _ -> raise Illtyped)
#  in sem
#;;
semant : semval list -> asl -> semval = <fun>
```

The main function must be able to treat an ASL declaration, evaluate it, and update the global environments (`global_env` and `global_sem`).

```
#let semantics = function Decl(s,e) ->
#    let result = semant !global_sem e
#    in global_env := s::!global_env;
#       global_sem := result::!global_sem;
#       print_string "ASL Value of ";
#       print_string s;
#       print_string " is ";
#       (match result with
#         Constval n -> print_int n
#       | Funval f -> print_string "<fun>");
#       print_newline();;
semantics : top_asl -> unit = <fun>
```

13.3 Examples

```
#semantics (parse_top "let f be \\x. + x 1;");;
ASL Value of f is <fun>
- : unit = ()

#semantics (parse_top "let i be \\x. x;");;
ASL Value of i is <fun>
- : unit = ()

#semantics (parse_top "let x be i (f 2);");;
ASL Value of x is 3
- : unit = ()

#semantics (parse_top "let y be if x then (\\x.x) else 2 fi 0;");;
ASL Value of y is 0
- : unit = ()
```