Progetto PR2

Interprete Ocaml – miniPy

Stefan Daniel Motoc

Gennaio 2016

Esecuzione codice

Per eseguire il codice, aprire il terminale e digitare:

"*ocaml < proj.ml*"

Sintassi Astratta

La sintassi astratta è stata realizzata tenendo conto della correlazione tra i tipi utilizzati. Essendo il linguaggio *miniPy* usato per manipolare tuple di valori, queste ultime sono state considerate come liste di valori base, cioè come segue:

```
type bVal =
    | Int of int
    | Bool of bool
    | Tuple of bVal list
```

Per quanto riguarda il resto della sintassi astratta, sono di particolare rilevanza:

```
    | FunApp of ide * exp                    (*applicazione di funzione*)
```

dove: - ide = nome della funzione da applicare
- exp = parametro attuale su cui viene chiamata la funzione

```
    | For of ide * bVal * exp                (*for su tupla*)
```

dove: - ide = identificatore del tipo dei valori da considerare nella tupla
- bVal = tupla
- exp = espressione da applicare

Semantica Operazionale e Funzioni Ausiliarie

Per poter realizzare la semantica operazionale, mi sono servito di funzioni ausiliarie, tra cui:

```
(*trova i valori compresi tra gli indici i e j nella tupla*)
let slice tup i j =
    let sliceL lista a b =
        let lung = length lista in
            if ((a>=0)&&(b>=0)) then take (drop lista a) (b-a+1)
            else if((a<0)&&(b<0)) then (let newa = lung+b in
                                        let newb = lung+a in
                                       reverse(take (drop lista newa) (newb-newa+1))
                                        )
                                   else raise InvalidIndex
    in
    match tup with
        | Tuple(y)  -> sliceL y i j
        | _          -> raise InvalidTuple
;;
```

In questa funzione, quando gli indici i e j sono positivi, si restituisce una sezione della tupla, utilizzando lo stesso ordine in cui gli elementi si trovavano nella tupla iniziale. Quando invece gli indici sono entrambi negativi, viene restituita una sezione della tupla iniziale i cui valori sono letti in ordine inverso. In tutti gli altri casi, viene sollevata un'eccezione.

Per la sua realizzazione, ho utilizzato inoltre la funzione *take*, che restituisce i primi n elementi di una lista, e la funzione *drop*, che invece elimina i primi n elementi.

Una delle più importanti implementazioni della semantica operazionale è:

```
| For(i, tup, e) -> (match tup with
        | Tuple(y) -> (match i with
                        | ("int")  -> let t = filter isInteger y in
                                        Tuple( applyExp (Ide("int")) t e env fenv )
                        | ("bool") -> let t = filter isBoolean y in
                                        Tuple( applyExp (Ide("bool")) t e env fenv )
                        | _        -> raise WrongType
                        )
        | _       -> raise InvalidTuple
        );;
```

La *For(i, tup, e)* è stata intesa come un'espressione "*for each*" che permette, cioè, di applicare l'espressione *e* a tutti gli elementi della tupla *tup* che sono di tipo *i.* Tra i possibili valori base, ho scelto di rendere utilizzabile la *For* solo per gli elementi della tupla che hanno tipo *int* oppure *bool*, dato che la maggior parte delle espressioni del linguaggio miniPy sono utilizzabili in questi due casi. Per poterla usare, quindi, si deve specificare, al posto del comune iteratore, il tipo dei valori a cui applicare *e*, che permetterà di filtrare tra gli elementi della tupla quelli del tipo giusto.

Eccezioni

Le eccezioni utilizzate sono le seguenti:

- *exception InvalidVariable;;*
- *exception InvalidTuple;;*
- *exception InvalidIndex;;*
- *exception InvalidIdentifier;;*
- *exception WrongType;;*

Scope

Per realizzare un interprete con Scope Dinamico, si dovrebbe evitare di salvare l'ambiente durante la dichiarazione, facendo in modo che l'invocazione della funzione sia basata sull'ambiente in cui è stata invocata e non in quella in cui è stata dichiarata.