Relazione Progetto OCaml PR 2

Venturi Ludovico
Docente: Francesca Levi

UNIPI, Gennaio 2020

Indice

1 Scelte progettuali

- tutti i valori associati alle chiavi in un dizionario hanno lo stesso tipo
- i tipi assumibili dai valori associati alle chiavi sono solamente interi e booleani, nello specifico: *Int(..), Bool(..)*
- non sono ammessi valori *Unbound* (corrispettivo di *null*)
- per semplicità di progettazione si assume che il primo valore nel dizionario, al momento della creazione, definisca il tipo dei valori del dizionario
- sono state introdotte le astrazioni funzionali multi argomento (funzioni con lista di parametri di dimensione variabile) per generalizzare dato che la *Fold* necessita di 2 argomenti

1.1 Sintassi Astratta

Estensione della sintassi astratta del linguaggio `type exp =`
`...| CreateDict of (ide * exp)list | Insert of ide * exp * exp | Delete of ide * exp | HasKey of ide * exp | Iterate of exp * exp | Fold of exp * exp | Filter of ide list * exp |...`
`...|FunCall of exp * exp list | FunArg of ide list * exp`

Estensione dei tipi esprimibili `type evT =`
`...|DictVal of (ide * evT)list |...`
`...| FunArgVal of ide list * exp * evT env`

1.2 Runtime Type Checker

Il *Runtime Type Checker* è stato implementato per la creazione del dizionario, dove si verifica che tutti i valori siano consistenti nel tipo (o tutti interi, o tutti booleani). Influisce anche sull'operazione di inserimento, mentre sulle operazioni *Delete, HasKey, Filter* non ha alcuna influenza.
Per quanto riguarda le applicazioni di funzioni sui valori, ovvero *Fold, Iterate*, si demanda il type checking (già implementato nel linguaggio didattico) alle funzioni chiamate (sfruttando la **lazy evaluation** dei parametri): se viene chiamata la Fold di una funzione che opera su booleani su di un dizionario contenente valori interi al momento della prima chiamata di funzione viene generato un **type error**.

2 Utilizzo

Aprire l'interprete top-level di OCaml da terminale digitando `ocaml`. Importare l'interprete del linguaggio qui discusso:

```
## use "venturi.ml";;
```

e da qui valutare le espressioni, entrando nel vivo del **REPL**(ReadEvalPrintLoop).

2.1 Esempi

Assumendo `let env0 : ide -> evT = emptyenv Unbound;;`

- `# let x = Dict([("Birman", Eint(3));("Mainecoon", Eint(13));("Siamese", Eint(17));("Foldex", Eint(21))]);;`
  val x : exp = Dict [("Birman", Eint 3); ("Mainecoon", Eint 13); ("Siamese", Eint 17); ("Foldex", Eint 21)]
- `# let y = eval (Insert("Korat", Eint(4), x)) env0;;`
  val y : evT = DictVal [("Korat", Int 4); ("Birman", Int 3); ("Mainecoon", Int 13); ("Siamese", Int 17); ("Foldex", Int 21)]