Relazione Progetto OCaml PR 2

Venturi Ludovico
Docente: Francesca Levi

UNIPI, Dicembre 2019

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
- la `Fold` è stata interpretata come funzione a 2 argomenti, dove il primo è l'*accumulatore* che prende il valore di default del tipo del dizionario; la `Fold` è applicabile sia ad interi che a booleani
- 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 |.D..`
`...|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 una sorta di **lazy evaluation** per i parametri attuali della funzione): 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).
Vari testcase sono riportati nel file `testcase.ml`.

2.1 Esempi

In `venturi.ml` vengono creati 4 ambienti:

```
emptyenv Unbound;;(*in env0*)
..  "intDict" -> [("Birman", Eint(3));("Mainecoon", Eint(13));("Siamese",
Eint(17));("Foldex", Eint(21))](*in env1*)
..  "boolDict" -> [("Birman", Ebool(true));("Mainecoon", Ebool(false));("Siamese",
Ebool(false));("Foldex", Ebool(false))](*in env2*)
..  "emptyDict" -> [] (*in env3*)
```

Sempre nello stesso file vengono riportate le prove **per ogni** operazione del dizionario, incluse tutte le eccezioni da esse generabili.
Alcuni esempi:

- **Insert**
  - `eval (Insert("Ragdoll", Eint(111), Den "intDict")) env3;;`
    - : evT = DictVal [("Birman", Int 3); ("Mainecoon", Int 13); ("Siamese", Int 17); ("Foldex", Int 21); ("Ragdoll", Int 111)]
  - `eval (Insert("Siamese", Eint(111), Den "intDict")) env3;;`
    Exception: InvalidArgumentException "key already present".
- **Fold**
  - `let env5 = bind env3 "foldFun" (eval (FunArg(["x"; "y"], Prod(Sum(Den "x", Den "y"), Eint 3) )) env3 );;`
    `eval (Fold(Den "foldFun", Den "intDict")) env5;;`
    - : evT = Int 810
  - `eval (Let("magazzino", CreateDict( [ ("mele", Eint(430)); ("banane", Eint(312)); ("arance", Eint(525)); ("pere", Eint(217))] ), Let("ff", FunArg(["x"; "y"], Sum(Sum(Den "x", Den "y"), Eint(1))), Fold(Den "ff", Den "magazzino")))) env3;;`
    - : evT = Int 1488
  - `eval (Fold( FunArg(["a"; "b"], Or(Not(Den "a"), Den "b")), Den "intDict" )) env3;;`
    Exception: Failure "Type error".