Esercizi su implementazione di interpreti

1

Si estenda il linguaggio MiniCaml con il costrutto condizionale

```
on G1 ; E1 | G2 ; E2
```

dove `G1` e `G2` sono espressioni booleane dette guardie. La valutazione del costrutto `on` avviene nel modo seguente:

- si valutano le guardie;
- se nessuna della due guardie viene valutata a true si solleva una eccezione;
- se una sola delle espressione viene valutata a true, si valuta l'espressione associata;
- se entrambe l'espressioni vengono valutate a true, viene valutata una qualunque delle due espressioni associate (nota: a scelta dello studente).

Si diano le regole di semantica operazionale che descrivono il comportamento di questo nuovo costrutto e si mostri come deve essere modificato l'interprete del linguaggio.

2

Si estenda il linguaggio MiniCaml con una nuova nozione di blocco della forma

```
use x = E1  or E2  in E  when G
```

dove `E1`, `E2` ed `E` sono espressioni generiche mentre `G` è una espressione a valori booleani detta guardia. La valutazione del blocco `use` avviene nel modo seguente

- si valuta la guardia;
- se la guardia viene valutata a true si prosegue con la valutazione di `E` nell'ambiente esteso con il legame tra `x` e il valore di `E1`;
- se la guardia viene valutata a false si prosegue con la valutazione di `E` nell'ambiente esteso con il legame tra `x` e il valore di `E2`.

Si mostri come deve essere modificato l'interprete del linguaggio.

3

Una funzione (unaria e non ricorsiva) a dominio finito è una funzione che è definita solo per un numero finito di elementi. Ad esempio si consideri la seguente funzione con una sintassi nello stile di OCaml

```
fun y -> 50 + y for y in [0; 1; 2; 3; 4];;
```

La funzione `sum` è definita solamente per valori del parametro attuale che appartengono all'insieme $\{0, 1, 2, 3, 4\}$, insieme che è calcolato al momento della definizione della funzione stessa.

Si diano le regole della semantica operazionale della definizione e applicazione di questo nuova astrazione funzionale e si estenda l'implementazione dell'interprete di MiniCaml in modo da includere la possibilità di definire e applicare funzioni a dominio finito.

4

Si consideri il nucleo di un semplice linguaggio di programmazione funzionale, la cui sintassi è descritta di seguito

Posizione $p ::= 0, 1, \ldots, 100$

Identificatori $I ::= \ldots$

Espressioni $e ::= I \mid p \mid moveLeft\ \ e \mid moveRight\ \ e \mid let\ \ I = e_1\ \ in\ \ e_2$

Intuitivamente, una *posizione* è un tipo di dato che individua una posizione su una retta finita (da 0 a 100). L'espressione "moveLeft" decrementa di uno il valore della posizione passata come argomento . Similmente "moveRight" incrementa di uno il valore della posizione passata come argomento Il decremento della posizione 0 produce 0 mentre l'incremento della posizione 100 produce come risultato 100.

Si definisca l'interprete del linguaggio utilizzando OCaml come linguaggio di implementazione.

5

Si estenda il linguaggio MiniCaml con il costrutto `DynFun of ide * exp` per la dichiarazione di funzioni non ricorsive che utilizzano regole di scoping *dinamico* nella risoluzione dei riferimenti non locali.

- Si mostri come deve essere modificato l'interprete del linguaggio
- Si discutano le eventuali altre modifiche necessarie per poter gestire funzioni ricorsive.

6

Si estenda il linguaggio MiniCaml introducendo il tipo di dato `IntSet` che permette di dichiarare insiemi di interi. In aggiunta, il linguaggio è esteso con le operazioni primitive `insert myset elem`, `remove myset elem` che permettono di operare su insiemi di interi. Si mostri come deve essere modificato l'interprete del linguaggio

7

Si estenda il linguaggio MiniCaml introducendo il tipo di dato `IntStack` che permette di dichiarare una pila di interi di lunghezza massima prefissata. In aggiunta, il linguaggio è esteso con le operazioni tipiche della pila quali `push, pop, top` e `isEmpty`. Si mostri come deve essere modificato l'interprete del linguaggio.