Progetto Mini – Lisp 2021

Luca Sachetti – Leonardo Colosi – Odysseas Diamadis – Simone Bodi

DESCRIZIONE

```
1. type symbol = string
2.
3. datatype lisp = Unit of unit
4. | Int of int
5. | Str of string
6. | Var of symbol
7. | Sym of symbol
8. | none
9. | plus of lisp*lisp
10. | Plus of lisp
11. | car of lisp
12. | cdr of lisp
13. | letLisp of lisp*lisp
14. | lambda of lisp*lisp
15. | apply of lisp*lisp
16. | quote of lisp
17. | cons of lisp*lisp
18. | applyFun of lisp*lisp*lisp
```

Il linguaggio è costruito sul seguente datatype, abbiamo un costruttore per ogni tipo di costante (**Int**, **str**. **Etc**), un tipo di dato stringa per rappresentare le variabili e i simboli di funzione, inoltre per le liste abbiamo utilizzato il costruttore **cons** che si comporta come il **cons** del Lisp. Per valutare un termine è necessario passarlo alla funzione eval.

Le funzioni del linguaggio sono:

1. **Plus** consta di due tipologie di operazione:
    a. **Plus:** prende in input una sequenza di numeri, restituisce la loro somma.
        i. Esempio: Plus (cons((Int 1), cons((Int 2), cons((Int 1), none))));
    b. plus: prende in input due numeri e li somma.
        i. Esempio: plus((Int 1), (Int 2))**;**
2. **Car:** come lisp prende il primo elemento del cons.
    a. Esempio: car(cons((Int 2), cons((Int 1), none)));
3. **Cdr:** come lisp prende il primo elemento del cons:
    a. Esempio: cdr(cons((Int 2), cons((Int 1), none)));
4. **letLisp:** prende in input due parametri, il primo di tipo cons deve contenere una lista di coppie variabile – espressione, il secondo rappresenta il corpo del let.
    a. Esempio: val lam = letLisp (cons (cons (Var "z",cons (Int 9,none)),none),plus(Var "z", (Int 2)));
5. **lambda:** prende in input due parametri, una lista di variabili, e un'espressione che rappresenta il corpo della lambda.
    a. Esempio: lambda(cons((Var "x"),none), (Var "x"));
6. **apply:** prende in input due argomenti una lambda e un cons di valori che verranno assegnati alle variabili delle lambda (nello stesso ordine).
    a. Esempio: apply(lambda (cons (Var "x",none),Var "x"), cons((Int 2),none));
7. **quote**: come nel lisp, prende in input un'espressione e restituisce la lista degli elementi dell'espressione dove gli operatori sono mappati in variabili di tipo sym.

   a. Esempio: quote(plus(Int 10, Int 2))
8. **applyFun**: è una funzione di "utility" che l'utente non utilizza, ma viene utilizzata internamente dall'**apply** per tenere traccia dell'ambiente. Nello specifico accetta un terzo parametro cons che rappresenta l'ambiente, descritto come una lista di coppie variabile-espressione.

Sono presenti ulteriori funzioni accessorie come:

1. **pretty:** ritorna un'espressione scritta dal nostro linguaggio al lisp tradizionale.
   a. Esempio: pretty(plus((Int 0),(Int 2)));
   b. Stampa: (+ 0 2)
2. **printer:** serve per stampare i valori prodotti da un'espressione
   a. Esempio: printer (cons((Int 2), cons((Int 3),none)));
   b. Stampa (2 3)

All'interno del file sorgente sono presenti altre funzioni che vengono utilizzate internamente per il controllo del tipo dei termini passati alle funzioni.

SCOPING

Il nostro linguaggio utilizza una semantica Eager con scoping dinamico.

Esempio BRUTTO:

- val zlam = lambda((cons(Var "z", none), Var "y"));

- apply(lambda(cons((Var "x"), none), letLisp(cons(cons((Var "y"), cons((Int 5), none)), none),apply((Var "x"), cons((Int 1),none)))), cons(zlam,none));