LPO

11/10 Malchiodi Riccardo

Paradigmi di Programmazione

I paradigmi sono un modo per classificare i linguaggi di programmazione, in base ad alcuni stili e approcci, basati su un modello computazionale chiamato *emergente*.

I paradigmi principali sono:

1. **Imperativo**: basato sulle istruzioni e sugli stati di un programma, e che a sua volta può essere:
    - **Procedurale**: si basa sulle **funzioni** (o procedure), come ad esempio C;
    - **Orientato ad Oggetti**: si basa sulle **classi** (C#, Java, C++...)

2. **Dichiarativo**: si basa su un modello più astratto, che a sua volta può essere:
    - **Funzionale**: basato sulle funzioni matematiche e sulla loro applicazione, come ad esempio ML.
    - **Logico**: basato su regole logiche e query.

In un linguaggio imperativo un programma è composto da **espressioni** e **statements**.

F#

La sintassi F# riportata in formato **EBNF** (Extended Backus-Naur Form) descrive alcune regole grammaticali del linguaggio, in particolare le dichiarazioni di funzioni, le espressioni e i pattern. Questa sintassi riguarda principalmente la definizione di funzioni e l'uso di lambda expressions (funzioni anonime).

EBNF grammar

```
Dec ::= 'let' Pat '=' Exp | 'let' ID Pat+ '=' Exp
Exp ::= Exp Exp | 'fun' Pat+ '->' Exp | ...
Pat ::= ID // simplified patterns
```

La prima regola si occupa delle dichiarazioni con il costrutto let, che è utilizzato per definire valori o funzioni.
Regola per le dichiarazioni:
Dec ::= 'let' Pat '=' Exp | 'let' ID Pat+ '=' Exp
Questa regola significa che una dichiarazione (Dec) può avere due forme:

1. **let Pat = Exp**
   Definisce una semplice assegnazione dove un pattern (Pat) è assegnato a un'espressione (Exp).
   Esempio:
   let x = 5

2. **let ID Pat+ = Exp**
   Questa forma è usata per definire funzioni. L'identificatore (ID) rappresenta il nome della funzione, e uno o più pattern (Pat+) rappresentano i parametri. L'espressione (Exp) è il corpo della funzione.
   Esempio:
   let add x y = x + y

La regola per le espressioni (Exp) definisce come vengono composte le espressioni.

Exp ::= Exp Exp | 'fun' Pat+ '->' Exp | ...

Questa regola ha diverse varianti:

1. **Applicazione di funzione** (Exp Exp):
   Un'espressione può essere seguita da un'altra espressione, che significa applicare una funzione a un argomento.
   Esempio:
   add 3 5

2. **Funzioni anonime (lambda expressions)**:
   La sintassi fun Pat+ -> Exp permette di creare funzioni anonime, cioè funzioni senza nome.
   Esempio:
   fun x -> x + 1

La regola per i pattern (Pat) è semplificata qui come un identificatore (ID):

Pat ::= ID

Un pattern può semplicemente essere un i**dentificatore**, ad esempio il nome di una variabile o di un parametro di funzione.

Lambda Expressions (Funzioni anonime)

Le **lambda expressions**, o funzioni anonime, sono un costrutto per definire funzioni al volo senza dare loro un nome.

Esempio:

```
fun x -> x + 1
```

Questa è una funzione anonima che prende un parametro x e restituisce x + 1. La sintassi è: fun (parola chiave per una funzione anonima), seguita dal parametro x, e poi -> per separare i parametri dal corpo della funzione. Il corpo della funzione è x + 1.

Funzioni curried:

In F#, una funzione può avere più parametri, ma internamente una funzione con più parametri viene trattata come una serie di funzioni che restituiscono altre funzioni (questa tecnica si chiama *currying*).

Ad esempio:

```
fun x y -> x * y
```

Questo è in realtà una scorciatoia per scrivere:

```
fun x -> fun y -> x * y
```