1. Le liste devono essere omogenee: tutti gli elementi devono avere lo stesso tipo. Gli elementi nelle tuple possono avere tipi diversi.
    - Esempi:
      1,true è permesso e ha tipo int*bool;
      1::true::[] non è permesso.
2. Le liste dello stesso tipo possono avere lunghezze diverse. La dimensione delle tuple dello stesso tipo è fissa.
    - Esempi:
      1::[] e 3::7::[] sono liste di tipo int list ma con lunghezze diverse;
      1,2 e 1,2,3 sono tuple di tipi diversi:
      -tutte le tuple di tipo int*int hanno dimensione 2;
      -tutte le tuple di tipo int*int*int hanno dimensione 3.

Concatenazione di Liste

Sintassi

- **@** è un operatore infisso binario.
- **Espressione** ::= Espressione '@' Espressione

Regole di Associazioni e Precedenza

- @ è associativo a sinistra.
- @ ha una precedenza inferiore rispetto a ::.

Semantica Statica

Se **e1** e **e2** sono staticamente corretti con tipo **t list**, allora **e1 @ e2** è staticamente corretto con tipo **t list**.

Semantica Dinamica

La semantica è la stessa della concatenazione di stringhe. Esempi:

- **(1::[ ]) @ (2::3::[ ])** è uguale a **1::2::3::[ ]**.
- **(1::2::[ ]) @ (3::[ ])** è uguale a **1::2::3::[ ]**.

Differenza tra @ e ::

@ è un operatore mentre **::** è un costruttore.

Perché @ è un operatore e :: è un costruttore?

- **::** è usato per costruire valori di lista.
- **@** è usato per calcolare un'operazione sulle liste.

Più precisamente:

- **e1 :: e2 = e'** implica che **e1 = e'1** e **e2 = e'2**.
- **e1 @ e2 = e'** non implica necessariamente **e1 = e'1** e **e2 = e'2**.

Esempio:
(1::[]) @ [] = [] @ (1::[]) ma **1::[]** e **[]** non sono uguali.

Il costruttore di lista consente una **decomposizione non ambigua** dei valori di lista.

La complessità temporale asintotica di @ e :: è diversa:

- La complessità temporale di **el::ls** è **O(1)**.
- La complessità temporale di **ls1 @ ls2** è **O(n)**, con **n** la lunghezza di **ls1**.

Pattern delle Liste

Sintassi Estesa per i Pattern
Abbiamo visto che il costruttore di tuple , può essere usato nei pattern; anche i costruttori di lista possono essere utilizzati nei pattern. In generale, qualsiasi costruttore può essere utilizzato nei pattern.

```
Pat ::= ID | '(' Pat ')' | Pat (',' Pat)+ | [] | Pat '::'  Pat
```

Gli identificatori nei pattern devono essere distinti per motivi di efficienza.

- examples of valid patterns

```
x
()
[]
x::y
x::y::z::[]
x,y
x,y,z
```

- examples of invalid patterns

```
x::x
x::y::x::[]
x,x
x,y,y
```

Matching dei Pattern

meccanismo potente per la decomposizione dei valori.

- examples

```
let (x,y) = 1,2
```
x is 1 and y is 2

```
let hd::tl = 1::2::[]
```
hd is 1 and tl is 2::[]