- L'uso delle parole chiave **'rec'**, **'and'** supporta dichiarazioni mutuamente ricorsive.
- Le dichiarazioni ricorsive sono consentite solo per i tipi di funzione.

LISTE

F# provides built-in list values and types

Syntax for list values

```
Exp ::= ... | '['     ']' | Exp '::' Exp
```

Il costruttore [ ] definisce la lista vuota.
Il costruttore :: definisce le liste non vuote:

- h::t è la lista in cui h è il primo elemento della lista, chiamato anche testa (head), e t è il resto della lista, chiamato anche coda (tail).

Regole di associatività e precedenza per ::

- **:: è associativo a destra.**
  h1::h2::t è equivalente a h1::(h2::t).

- **:: ha una precedenza maggiore** rispetto al costruttore di tuple (virgola).
  Esempio: 1,2::[ ] significa 1,(2::[ ]).

- **L'espressione lambda:**
  fun x->x+1::[ ] significa fun x->(x+1::[ ]).

- **L'espressione condizionale**:
- if x<0 then -x::[ ] else x::[ ] significa if x<0 then (-x)::[ ] else (x::[ ]).
- Esempio: 1::[ ],2 significa (1::[ ]),2.

Sintassi per i tipi di lista

```
Type ::= ... | Type 'list'
```

list è un **costruttore di tipo unario postfix**: ha un argomento (unario) che specifica il tipo degli elementi della lista. Il costruttore viene dopo (postfix) l'argomento.

Semantica statica delle espressioni di lista

Regola per [ ]:

- [ ] ha tipo t list per qualsiasi tipo t poiché [ ] è vuota, i suoi elementi possono essere di qualsiasi tipo!

Regola per ::

- Se e1 è staticamente corretto con tipo t
- E e2 è staticamente corretto con tipo t list, allora e1::e2 è staticamente corretto con tipo t list.

Tipi polimorfici

è chiamato **tipo polimorfico** o **schema di tipo**. 'a è una **variabile di tipo**. 'a list significa semplicemente t list per tutti i tipi t.

Tipizzazioni riuscite

- 2 è staticamente corretto con tipo int.
- [ ] è staticamente corretto con tipo int list.
- Pertanto, 2::[ ] è staticamente corretto con tipo int list.
- 1 è staticamente corretto con tipo int.
- Pertanto, 1::2::[ ] è staticamente corretto con tipo int list.

Tipizzazioni fallite

Questo spiega perché :: è associativo a destra:

- 1 è staticamente corretto con tipo int.
- 2 è staticamente corretto con tipo int.
- Pertanto, 1::2 non è staticamente corretto.
- Pertanto, (1::2)::[ ] non è staticamente corretto.
- Questo è una conseguenza del vincolo che le liste devono essere omogenee:
    - true è staticamente corretto con tipo bool.
    - [ ] è staticamente corretto con tipo bool list.
    - Pertanto, true::[ ] è staticamente corretto con tipo bool list.
    - 1 è staticamente corretto con tipo int.
    - Pertanto, 1::true::[ ] non è staticamente corretto.

Abbreviazione sintattica

F# utilizza la seguente abbreviazione sintattica:

- [e1;e2;...;en] è una scorciatoia per e1::e2::...::en.

Esempio:

- [1;2;3] è uguale a 1::2::3::[].

Osservazione:

- [1;2;3] non è uguale a [1,2,3].
    - [1;2;3] ha tipo int list e contiene tre interi.
    - [1,2,3] ha tipo (int*int*int) list e contiene una tripletta di interi.

Liste vs. tuple

Differenze tra valori di tipo lista e tupla:

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.