# Compilatori

. . . e quello che sono riuscito a carpire dalle sgangherate lezioni di leoncini

Matteo Lugli

## Linguaggi Formali

- Concetto di potenza n-esima di un linguaggio: $L^n = L^{n-1}L$, n>0
  Per esempio, $\{0,1\}^2 = \{00,01,10,11\}$
- $B^* = \bigcup_{n=0}^{\infty}\{0,1\}^n$, quindi l'insieme di tutte le stringhe binarie.
- $L_2 = \{x \in B^* : |x| = 2\} = 00,01,10,11$
- $L_R = \{x \in A^* : \exists y \in A^*, a \in A, x = yca\}$
  con A = $\{a,b,c\}$ . Quindi è l'insieme di tutte le stringhe (costituite da solo a,b,c che hanno come penultima lettera la c).
- **Caratterizzazione algoritmica** Un linguaggio può essere caratterizzato dall'insieme di algoritmi che data in input una stringa su un determinato alfabeto, rispondono sempre *True* o *False*. Ad esempio, il C++ è l'insieme delle stringhe (alfabeto ASCII) per cui il compilatore(algoritmo) non produce errore.
- **Caratterizzazione generativa** Si parla di caratterizzazione generativa quando si definiscono regole per generare solo *alcune* stringhe del linguaggio di interesse.

### Riconoscitori in C++

Esempio di riconoscitore di L2 implementato in C++:

```
#include <iostream>
using namespace std;

string L1[4] = {"00","01","10","11"};

int main(int argc, char **argv)
{
   if (argc>1) {
      for (int i=0;i<4;i++) {
         if (L1[i] == argv[1]) {
            cout << "Accept\n";
            return 0;
         }
      }
      cout << "Reject\n";
   } else {
      cout << "Missing the argument\n";
   }
   return 0;
}
```

Esempio di riconoscitore di $L_c$

```
#include <set>
set<char> S = set<char>(begin({'a','b','c'}),end({'a','b','c'}))
int main(int argc, char **argv) {
   if (argc>1) {
      string str(argv[1]);
      if (str.rbegin()[1] != 'c') {
         cout << "Reject\n";
         return 0;
      }
      set<char>::iterator c;
      string::iterator I = str.begin();
      for (;I!=str.end();I++) {
          c = S.find(*I);
          if (c == S.end()) {
             cout << "Reject\n";
             return 0;
          }
      }
      cout << "Accept\n";
   } else {
      cout << "Missing the argument\n";
   }
   return 0;
```

## Linguaggi Regolari

- **Linguaggio unitario**: linguaggio formato da un solo carattere, come {"$a$"}.
- **Linguaggio regolare**: un linguaggio si dice regolare se è esprimibile come combinazione di linguaggi unitari. Ma ogni parola è ottenuta dalla concatenazione di linguaggi unitari –> qualsiasi linguaggio finito è regolare.

### Espressioni regolari

Alcuni esempi e concetti fondamentali:

- $0 + ((1)^*10)$ –> $R_1 = \{0, 10, 110, 1110, ...\}$
- $(0 + 1)^*$ –> linguaggio che descrive tutte le stringhe binarie;
- $(1 + 01)^*(0 + 1 + 01)$ –> stringhe di lunghezza almeno 1 che non contengono zeri consecutivi.

Le espressioni regolari supportano tutte le operazioni insiemistiche, tranne alcune come la **negazione** o la **potenza finita**. Spesso quando si lavora con le espressioni regolari

si usando dei **metacaratteri**, ossia caratteri che non fanno parte dell'alfabeto per esprimere le espressioni stesse ("e.g. *escape sequence*).

Alcuni esercizi svolti in classe (le soluzioni sono state proposte e discusse in classe):

- Scrivere un'espressione regolare che descriva il seguente linguaggio: $\{a^n b^m c^k | m = 0 \Rightarrow k = 3\}$ Si parla essenzialmente di una sequenza di enne a, emme b, e kappa c. Nel caso in cui m sia 0, allora k deve essere uguale a 3. *Soluzione proposta:* $a^*(ccc|(b^*b)c^*)$
- Scrivere un'espressione regolare che descriva il linguaggio costruito sull'alfabeto $\{a, b\}$ che contiene stringhe contenenti *al più* due "a".
  *Soluzione proposta:* $(b^*a?b^*)\{0, 2\}$
- Scrivere un'espressione regolare che descriva il linguaggio costruito sull'alfabeto $\{a, b\}$ che contiene stringhe contenenti un numero dispari di b.
  *Soluzione proposta:* $(a^*ba^*(ba^*ba^*)^*)$