Linguaggi e compilatori

Corso di Laurea in Informatica

Mauro Leoncini

A.A. 2023/2024

Linguaggi e compilatori

1. Linguaggi formali
   - Alfabeti e linguaggi
   - Semplici riconoscitori in C++

Linguaggi e compilatori

Linguaggi e linguaggi formali

- Un linguaggio è un "sistema di parole e segni che le persone usano per comunicarsi pensieri e sentimenti" (Merriam-Webster)
- È questa, però, una nozione empirica, che può andare bene per i linguaggi *naturali* (italiano, inglese, cinese, ...)
- Essa risulta inadeguata per lo sviluppo di una teoria matematica e di algoritmi di manipolazione dei linguaggi che ci interessano particolarmente in Informatica
- NOTA: anche i linguaggi naturali, nonché la costruzione di sistemi software per la loro comprensione e sintesi, sono oggi oggetto di intensa ricerca in Informatica. Si tratta però di altro rispetto a ciò di cui ci occupiamo in questo insegnamento
- A noi interessano i *linguaggi formali*, ovvero linguaggi definiti mediante un qualche apparato formale di natura matematica.

Linguaggi formali

- Per un linguaggio definito formalmente, è (quasi sempre) possibile stabilire se una "frase" è corretta (*sintassi*)
- Soprattutto, è possibile associare un ben preciso significato alle frasi, senza possibili ambiguità (*semantica*)
- L'Informatica è totalmente permeata di linguaggi formali
- L'esigenza di non ambiguità, che appare evidente quando si pensa che gli algoritmi, espressi necessariamente in un qualche linguaggio, devono essere eseguiti da una macchina, esiste però anche in altre discipline
- Si pensi al linguaggio della Matematica

$$\forall \delta > 0, \exists \epsilon > 0 : |x - x_0| < \epsilon \Rightarrow |f(x) - f(x_0)| < \delta$$

o della Chimica o anche della musica.

Linguaggi formali in Informatica

- Linguaggi di programmazione: `C/C++`, `Java`, `Python`, ...
- Linguaggi di marcatura: `HTML`, `XML`, `LaTeX`
- Linguaggi di interrogazione: `SQL` (per DB relazionali), `SPARQL` (per dati rappresentati mediante il modello *RDF*), `GraphLog` (per *graph database*)
- Linguaggi di configurazione: `sendmail`, `apache`, `iptables` (nella directory `/etc` di un sistema Linux gli esempi si sprecano).
- Linguaggi per la descrizione (e visualizzazione) di strutture matematico/scientifiche: grafi, molecole, proteine, allineamenti di sequenze genomiche, ...

Andiamo per ordine: la nozione di alfabeto

- Un *alfabeto* è un insieme finito di simboli (detti anche *caratteri*)
- Esempi di alfabeti:
  - $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$
  - I set di caratteri ASCII e UNICODE;
  - $\mathcal{B} = \{\mathtt{0}, \mathtt{1}\}$, l'alfabeto binario;
  - $\mathcal{D} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, l'alfabeto del DNA.
- Utilizzeremo poi il simbolo Σ per indicare un generico alfabeto
- Come (forse) si può notare, i simboli di un alfabeto li scriviamo qui usando il font `typewriter`
- Per indicare un generico carattere utilizziamo invece le lettere iniziali dell'alfabeto latino (a, b e c)

Stringhe

- Una *stringa* su un dato alfabeto Σ è una sequenza di caratteri di Σ giustapposti
- Ad esempio 0110 è una stringa su $\mathcal{B}$ mentre TAATA è una stringa su $\mathcal{D}$
- Per indicare una stringa generica utilizzeremo sia le ultime lettere dell'alfabeto latino (w, x, y, z) sia le prime lettere dell'alfabeto greco (α, β, γ)
- Una stringa speciale è quella formata da zero caratteri, detta stringa vuota e indicata con ϵ
- Nonostante l'utilizzo di "font" diversi, spesso per evitare ambiguità racchiuderemo le stringhe fra coppie di apici (o doppi apici)
- In particolare, una stringa formata da un solo carattere sarà sempre racchiusa fra apici
- Per ragioni che saranno evidenti a breve, per esprimere succintamente il fatto che x è una stringa sull'alfabeto Σ si usa scrivere $x \in \Sigma^*$

Operazioni sulle stringhe

- Un'operazione fondamentale definita sulle stringhe è la *concatenazione*, ovvero la giustapposizione di due stringhe, una di seguito all'altra

$$x = \texttt{Linguaggi} \quad y = \texttt{formali}$$
$$xy = \texttt{Linguaggiformali}$$

- La concatenazione è un'operazione associativa ma chiaramente non commutativa
- La stringa vuota ϵ è l'*elemento neutro* dell'operazione di concatenazione:

$$\epsilon x = x\epsilon = x$$

- Se riguardiamo la concatenazione come il prodotto di stringhe possiamo conseguentemente definire la potenza di una stringa x

$$x^k = \begin{cases} \epsilon & \text{se } k = 0 \\ x^{k-1}x & \text{se } k > 0 \end{cases}$$

Altre semplici nozioni sulle stringhe

- La scrittura $|x|$ denota la *lunghezza* della stringa x.
- x_i denota il carattere in posizione i, $i = 0, \ldots, |x| - 1$.
- Poiché la numerazione parte da zero, quando si parla dell'i-esimo carattere (primo, secondo, ecc) ci si riferisce al carattere di indice/posizione $i - 1$
- Sempre per lo stesso motivo, se $|x| = n$, l'ultimo carattere di x ha indice $n - 1$.
- Una *sottostringa* di x é una stringa formata dai caratteri $x_i, x_{i+1}, \ldots, x_j$, per opportuni valori di i e j, $0 \leq i \leq j < n$
- Casi particolari sono
  - $i = 0$, in ta caso la sottostringa è il *prefisso* di ordine $j + 1$
  - $j = n - 1$, in tal caso la sottostringa è il *suffisso* di ordine $n - i$
- La scrittura x^R indica la stringa ottenuta rovesciando i caratteri di x: $x = \texttt{Roma} \Rightarrow x^R = \texttt{amoR}$

Linguaggi

- Un *linguaggio* su un dato alfabeto Σ è un insieme di stringhe di caratteri (o simboli) di Σ
- La questione fondamentale è come caratterizzare le stringhe che fanno parte del linguaggio, e dunque il linguaggio stesso
- Se le stringhe che compongono il linguaggio sono in numero finito, una possibilità consiste nella loro *elencazione*.
- Ad esempio, il linguaggio

$$L_2 = \{00, 01, 10, 11\}$$

  definito su $\mathcal{B}$ è formato da 4 stringhe, indicate per esteso
- L'interesse maggiore però è per i linguaggi infiniti, dei quali vogliamo naturalmente dare una descrizione finita

Operazioni con i linguaggi

- Poiché i linguaggi sono insiemi, su di essi sono definite tutte le operazioni insiemistiche: *unione*, *intersezione*, *differenza*, ecc.
- Due linguaggi M ed N su uno stesso alfabeto Σ si possono poi *concatenare*:

$$L = MN = \{z \in \Sigma^* : \exists x \in M, \exists y \in N \text{ t.c. } z = xy\}$$

  In altre parole, L è costituito da tutte le stringhe che possono essere scritte come concatenazione di una stringa di M e di una stringa di N.
- L'*elemento neutro* per la concatenazione di linguaggi è il linguaggio costituito dalla sola stringa vuota $\{\epsilon\}$
- Esattamente come per le stringhe, possiamo definire la *potenza n-esima* anche per un linguaggio L:

$$\begin{aligned} L^0 &= \{\epsilon\} \\ L^n &= L^{n-1}L,\ n > 0. \end{aligned}$$

Operazioni con i linguaggi (continua)

- La *chiusura* (*riflessiva*) di L è il linguaggio

Unione di tutte le potenze

$$L^* = \bigcup_{n=0}^{\infty} L^n = L^0 \cup L^1 \cup L^2 \cup \ldots$$

- Ad esempio:

$$\begin{aligned} \mathcal{B}^* &= \bigcup_{n=0}^{\infty} \{0,1\}^n \\ &= \{0,1\}^0 \cup \{0,1\}^1 \cup \{0,1\}^2 \cup \ldots \\ &= \{\epsilon\} \cup \{0,1\} \cup \{00,01,10,11\} \ldots \end{aligned}$$

  e dunque $\mathcal{B}^*$ è l'insieme di *tutte le stringhe binarie*.
- In generale, Σ^* è l'insieme di tutte le stringhe su un alfabeto Σ.

Operazioni con i linguaggi (continua)

- La *chiusura* (non riflessiva) di L è definita come $L^+ = LL^*$, ovvero:

  Esclude la stringa vuota $$L^+ = \bigcup_{n=1}^{\infty} L^n = L^1 \cup L^2 \cup \ldots$$

- La *riflessione* di un linguaggio L su un alfabeto Σ è il linguaggio:

$$L^R = \{x \in \Sigma^* : \exists y \in L \,\text{t.c.}\, x = y^R\}.$$

- Il numero di stringhe di un linguaggio finito verrà indicato con la notazione $|L|$. Se L è infinito risulta $|L| = \mathbf{N}$, con ciò intendendo che L ha la stessa cardinalità dell'insieme dei numeri naturali.

Specifica di linguaggi

- Se il linguaggio è infinito può essere comunque possibile descriverlo con quantità finita di informazione.
- Esempio: il linguaggio L_2 costituito da tutte le stringhe su $\mathcal{B}$ che terminano con il carattere 0
- Matematicamente:

Numeri binari che terminano per zero

$$L_0 = \{x \in \mathcal{B}^* | x = y0, y \in \mathcal{B}^*\}$$

- In questo caso e negli esempi sotto indicati si descrive formalmente una proprietà che caratterizza tutte e sole le stringhe del linguaggio:
  - $L_2 = \{x \in \mathcal{B}^* : |x| = 2\}$; Stringhe binarie di lunghezza 2
  - $L_3 = \{x \in \mathcal{B}^* : |x| \geq 3\}$; Stringhe binarie di lunghezza almeno 3
  - $L_{ss} = \{x \in \mathcal{B}^* : \exists k \geq 0 \,\text{t.c.}\, X = 01^k0\}$; Stringhe che incominciano e finisco con zero e hanno da 0 a n 1 in mezzo
  - $L_{pal} = \{x \in \mathcal{B}^* : x = x^R\}$.
  - $L_c = \{x \in A^* : \exists y \in A^*, a \in A, x = yca\}$ qualsiasi stringa con al penultimo c e alla fine a,b,c

Specifica riconoscitiva

- Riconosce una stringa

- Per "scopi informatici", molto più interessanti sono altri due modi di caratterizzare i linguaggi.
- Caratterizzazione *algoritmica* (o *riconoscitiva*).
- Ogni algoritmo di decisione, tale cioè che, data in input una stringa su una dato alfabeto, risponda sempre `yes` o `no` (oppure `True` o `False`) può essere utilizzato per definire un linguaggio.
- A algoritmo di decisione, Σ alfabeto generico:

$$\mathcal{L}_A = \{x \in \Sigma^* | A(x) = \texttt{True}\}$$

- Il linguaggio `C++` è l'insieme delle stringhe sull'alfabeto ASCII (programmi) per cui il compilatore `C++` (l'algoritmo) non produce errore.

Specifica generativa

● genera una stringa

- La seconda tecnica per descrivere un linguaggio che risulta fondamentale in ambito informatico è quella *generativa*.
- Con questa tecnica si danno "regole" mediante le quali è possibile generare tutte e sole le stringhe del linguaggio che si vuole specificare.
- I due formalismi più importanti in ambito informatico sono le *espressioni regolari* e le *grammatiche context-free*.
- Come vedremo, espressioni regolari e grammatiche sono gli strumenti fondamentali per definire il comportamento di lexer e parser.

Linguaggi e compilatori

Riconoscimento di semplici linguaggi

- Conveniamo (per il momento!) che i programmi riconoscitori stampino `Accept` oppure `Reject` a seconda che la stringa passata in input appartenga o meno al linguaggio.
- Scrivere un programma C++ che *riconosca il linguaggio* L_2 (questo vuol dire che riconosca tutte e sole le stringhe del linguaggio)
- Scrivere programmi C++ per riconoscere i linguaggi $L_{\mathtt{c}}$ e $L_{\mathtt{parity}}$, quest'ultimo formato dalla stringhe sull'alfabeto $\mathcal{B}$ che includono un numero pari di 1
- Nei programmi presentate (qui e nel resto del corso) per ragioni spazio a volte ometteremo di riportare la direttiva `#include <iostream>`

Riconoscitore C++ per L_2

```
#include <iostream>

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]) {
                std::cout << "Accept\n";
                return 0;
            }
        }
        std::cout << "Reject\n";
        return 0;
    }
    std::cout << "Missing the argument\n";
    return 1;
}
```

Riconoscitore alternativo: un primo esempio di uso di STL

```
#include <iostream>
#include <set>

std::set<std::string> L1 = {"00","01","10","11"};

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

Riconoscitore C++ per L_c

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

Riconoscitore C++ per L_{parity}

```
#include <cstring>

int main(int argc, char **argv) {
        if (argc>1) {
                int ones = 0;
                char* str = argv[1];
                for (long unsigned i=0; i<strlen(str); i++) {
                        if (str[i] != '0' && str[i] != '1') {
                                std::cout << "Reject\n";
                                return 0;
                        } else {
                                if (str[i] == '1')
                                ones++;
                        }
                }
                if ((ones&1)==0) std::cout << "Accept\n";
                else std::cout << "Reject\n";
                return 0;
        }
        std::cout << "Missing the argument\n";
        return 1;
}
```

Semplici esercizi

- Scrivere programmi C++ che riconoscano:
  - le stringhe sull'alfabeto $\mathcal{B}$ in cui ogni carattere `0` è seguito da un `1`;
  - le stringhe palindrome (su un qualsiasi alfabeto);
  - le stringhe sull'alfabeto $\mathcal{B}$ in cui almeno la metà dei caratteri è `1`;
  - le stringhe sull'alfabeto $\mathcal{A} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ della forma $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$, $n \geq 0$;
  - le stringhe β su un qualsiasi alfabeto Σ tali che $\beta = \alpha\alpha$, per una qualche altra stringa $\alpha \in \Sigma^*$