Linguaggi formali

Un'introduzione "linguistica"
all'informatica

Prof. Ugo de'Liguoro

Comunicare con una macchina

I computer sono macchine simbolo del progresso tecnologico; per poterle usare però dobbiamo comunicare con loro.
Per questo si usano specifici linguaggi.

Una Babele di linguaggi

"Orsù, scendiamo laggiù e confondiamo la loro lingua, affinché l'uno non comprenda più il parlare dell'altro" GEN 11, 1-9

Pascal

HTML

SQL

UML

...

Linguaggi per rappresentare dati e programmi

L'Informatica studia l' elaborazione automatica delle **informazioni**

Informazioni (dati)

Algoritmi (programmi)

Linguaggi artificiali

Che cosa è un linguaggio?

Un linguaggio è un **sistema di segni** che combinati secondo regole comunicano un significato (asserzioni, descrizioni, comandi, ecc.).

Per comodità ci limiteremo a segni (dattilo) scritti, ma la teoria è del tutto generale

Vocabolario, stringhe e linguaggi

Vocabolario := collezione finita di simboli $\Sigma = \{a, b, \ldots\}$

Stringa su Σ := sequenza finita di simboli di Σ, anche ripetuti

Σ^* è l'insieme delle stringhe su Σ

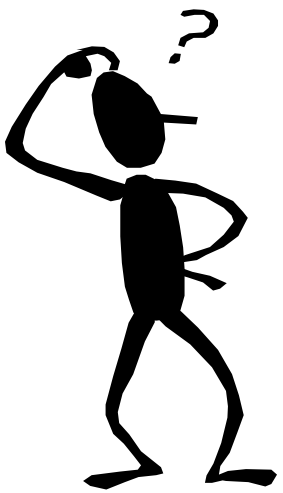
Linguaggio su Σ := un opportuno insieme di stringhe su Σ:

$L \subseteq \Sigma^*$.

Parole, frasi o interi testi sono stringhe sul vocabolario formato dalle lettere dell'alfabeto più altri simboli, quali l'interpunzione e gli spazi (anche quelli sono simboli, che richiedono nelle macchine un'opportuna codifica)

Grammatiche

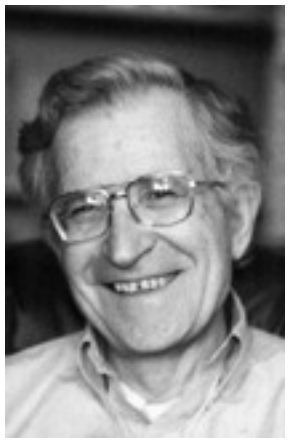
Una grammatica per il linguaggio L è un insieme di regole per distinguere ciò che appartiene ad L (stringhe ben formate) da ciò che non vi appartiene.



un accarezza il bambino cane

il bambino accarezza un cane

Grammatiche



Noam Chomsky

Una **grammatica generativa** per L è un 'insieme di regole che permettono di produrre tutte e sole le stringhe, o frasi ben formate, di L

F → SN SV

SN → N | A N

A → un | il

SV → V | V SN

V → accarezza

Grammatica **G**, cioè un insieme finito di regole di produzione

Grammatiche

F → SN SV

SN → N | A N

A → un | il

SV → V | V SN

V → accarezza

Grammatica **G**

F → SN SV
→ A N SV
→ il N SV
→ il bambino SV
→ il bambino V SN
→ il bambino accarezza SN
→ il bambino accarezza A N
→ il bambino accarezza un N
→ il bambino accarezza un cane

Produzione di una frase del linguaggio $\mathbf{L_G}$

Grammatiche

Σ = {il, un, bambino, cane, accarezza} sono i simboli terminali

Γ = {F, SN, SV, A, N, V} sono i simboli non terminali (F l'assioma)

G = {F → SN SV, … , A → il | un, … } insieme di produzioni:

Schema di una grammatica **indipendente dal contesto**

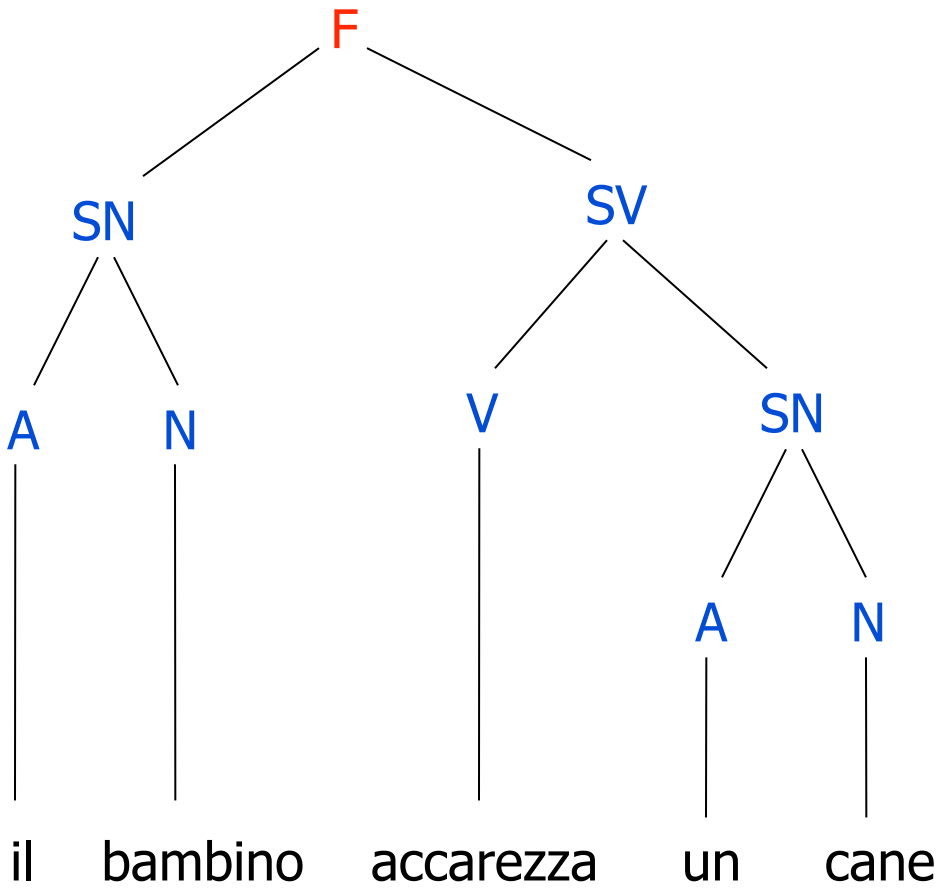
G insieme di produzioni della forma: $\Gamma \rightarrow (\Sigma \cup \Gamma)^*$

$\mathbf{L_G} = \{\sigma \in \Sigma^* \mid F \rightarrow \ldots \rightarrow \sigma$ è una seq. di produzioni di **G**$\}$

L_G comprende anche frasi insensate come: "il cane accarezza il bambino". Essa è formale perché opera una distinzione sulla base della forma, non del contenuto (significato)

Alberi sintattici

L'albero sintattico astrae dall'ordine in cui le regole di G sono state applicate ed enfatizza la struttura grammaticale della frase prodotta

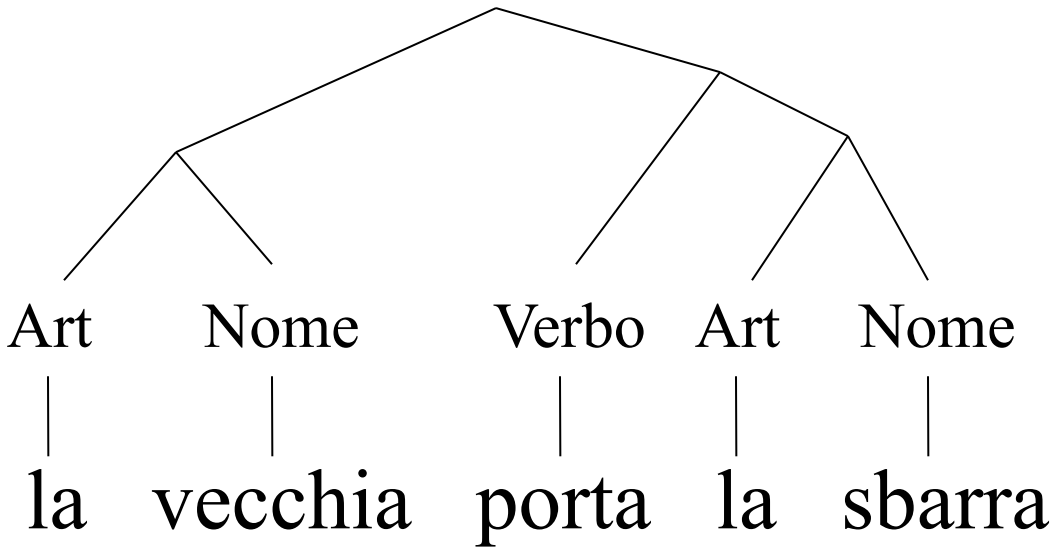


Ambiguità

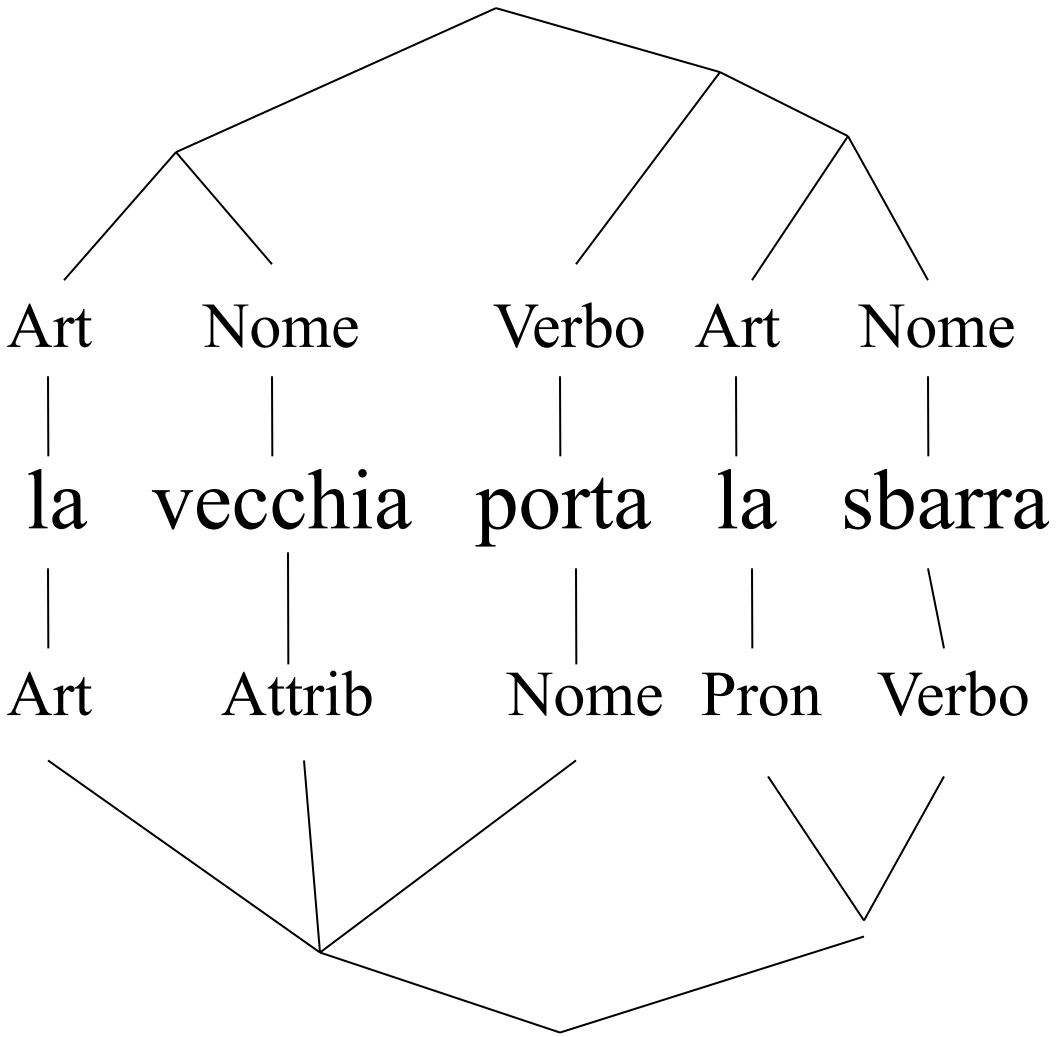
Ambiguità

la vecchia porta la sbarra

Ambiguità



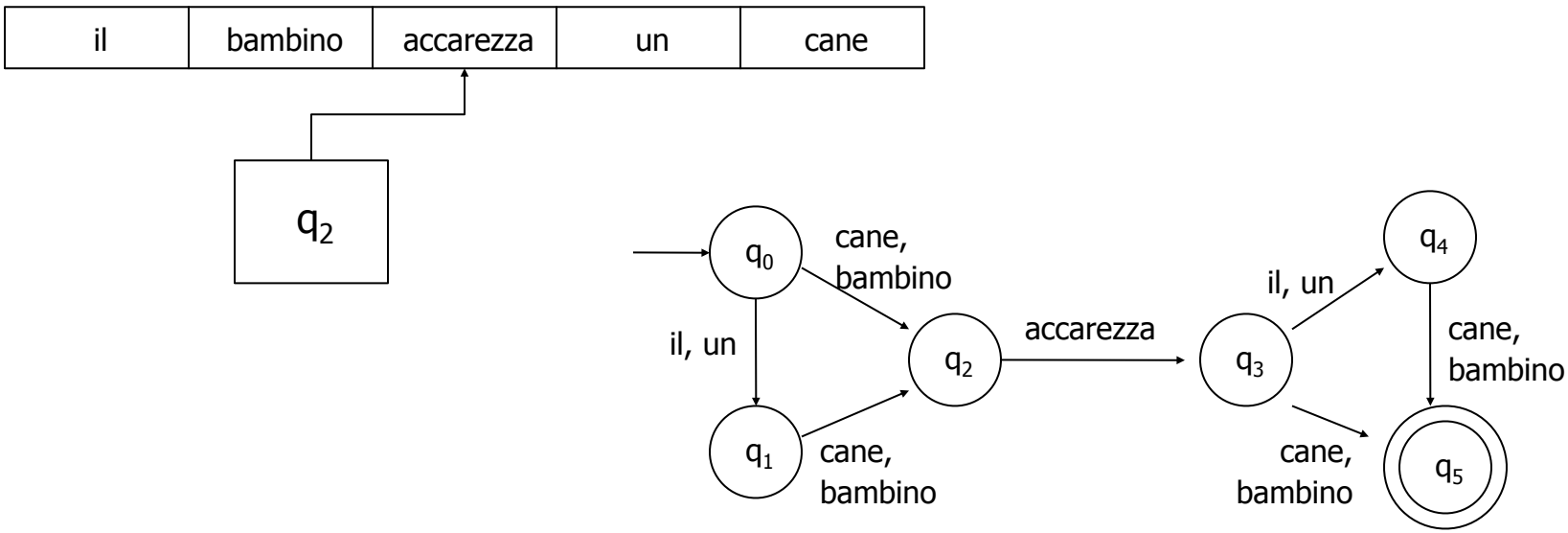
Ambiguità


Art
Nome
Verbo
Art
Nome
la
vecchia
porta
la
sbarra
Art
Attrib
Nome
Pron
Verbo

Automi riconoscitori

Il primo problema è decidere quando una stringa $\sigma \in \Sigma^*$ è un elemento di L_G, ossia è producibile mediante G.

Nel caso della nostra grammatica G basta un automa a stati finiti (FA):



Deterministic Finite State Automaton (DFA)

$$M = (Q, \Sigma, \delta, q_0, F)$$

Deterministic Finite State Automaton (DFA)

$$M = (Q, \Sigma, \delta, q_0, F)$$

Q: insieme finito di stati

Deterministic Finite State Automaton (DFA)

$$M = (Q, \Sigma, \delta, q_0, F)$$

Q: insieme finito di stati

Σ: alfabeto

Deterministic Finite State Automaton (DFA)

$$M = (Q, \Sigma, \delta, q_0, F)$$

Q: insieme finito di stati

Σ: alfabeto

δ: funzione di transizione

Deterministic Finite State Automaton (DFA)

$$M = (Q, \Sigma, \delta, q_0, F)$$

Q: insieme finito di stati

Σ: alfabeto

δ: funzione di transizione

funzione totale da $Q \times \Sigma$ a Q

Deterministic Finite State Automaton (DFA)

$$M = (Q, \Sigma, \delta, q_0, F)$$

Q: insieme finito di stati

Σ: alfabeto

δ: funzione di transizione

funzione totale da $Q \times \Sigma$ a Q

q_0: stato iniziale

Deterministic Finite State Automaton (DFA)

$$M = (Q, \Sigma, \delta, q_0, F)$$

Q: insieme finito di stati

Σ: alfabeto

δ: funzione di transizione

funzione totale da $Q \times \Sigma$ a Q

q_0: stato iniziale

F: insieme degli stati finali (accettanti)

Deterministic Finite State Automaton (DFA)

$$M = (Q, \Sigma, \delta, q_0, F)$$

Q: insieme finito di stati

Σ: alfabeto

δ: funzione di transizione

funzione totale da $Q \times \Sigma$ a Q

q_0: stato iniziale

F: insieme degli stati finali (accettanti)

Come opera la macchina

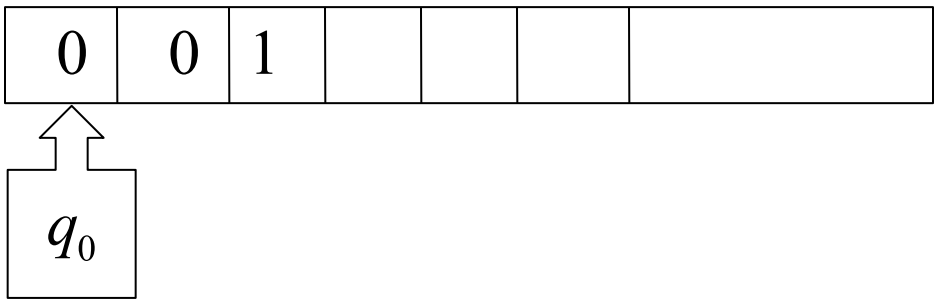
nastro input

controllo finito

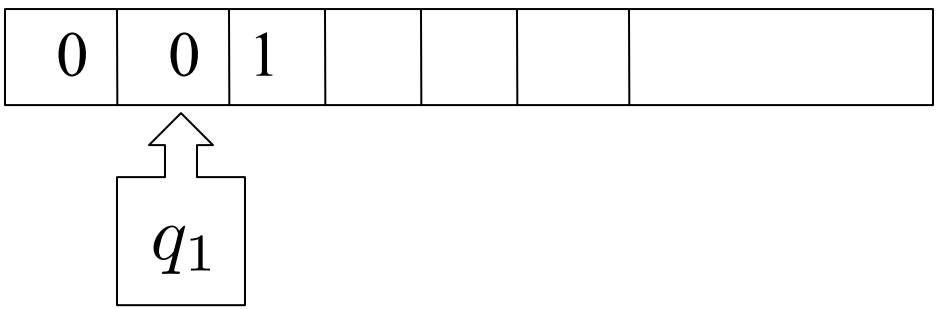
Come opera la macchina

Legge il carattere sul nastro nella posizione in cui si trova la testina; in funzione di questo carattere e dello stato cambia lo stato come stabilito dalla funzione di trnasizione

Come opera la macchina



$$\delta(q_0, 0) = q_1$$



Associazione di un linguaggio ad un DFA

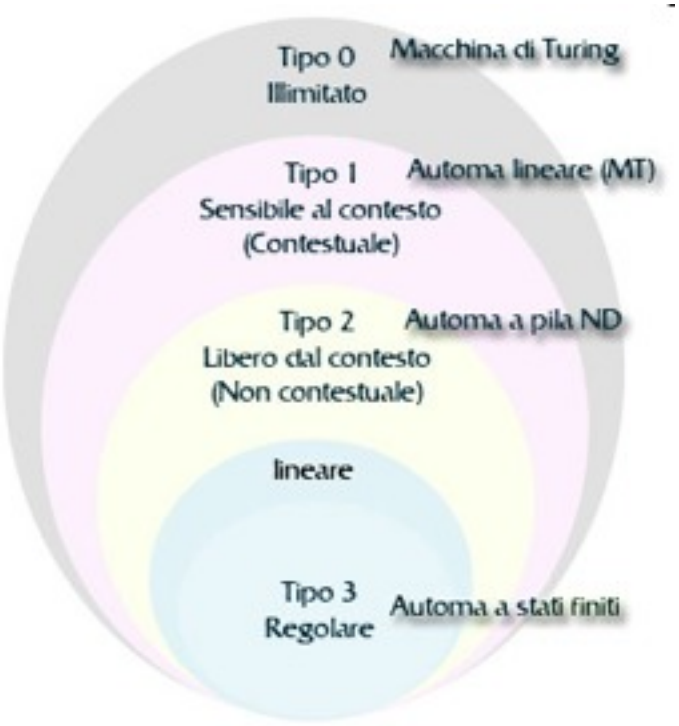
- **Configurazione:** (q, σ) dove $\sigma \in \Sigma^*$
- **Transizione:** $(q, a\sigma) \longrightarrow_M (q', \sigma)$ se $\delta(q, a) = q'$
- **Linguaggio:** $L_M = \{\sigma \in \Sigma^* \mid (q_0, \sigma) \longrightarrow_M^* (q, \tau) \text{ per qualche } q \in F\}$

Definizione. Una gramatica è **lineare** se non contestuale (i.e. con un solo simbolo a sin. $\rightarrow$) e a destra di ogni $\rightarrow$ vi è al massimo un non terminale.

Es. A $\rightarrow$ aBc è lineare; A $\rightarrow$ aBcC non è lineare

Teorema. Se G è lineare allora esiste un DFA M t.c. L_G = L_M ; viceversa se M' è un DFA allora esiste una grammatica lineare G' t.c. $L_{G'} = L_{M'}$.

Gerarchia di Chomsky

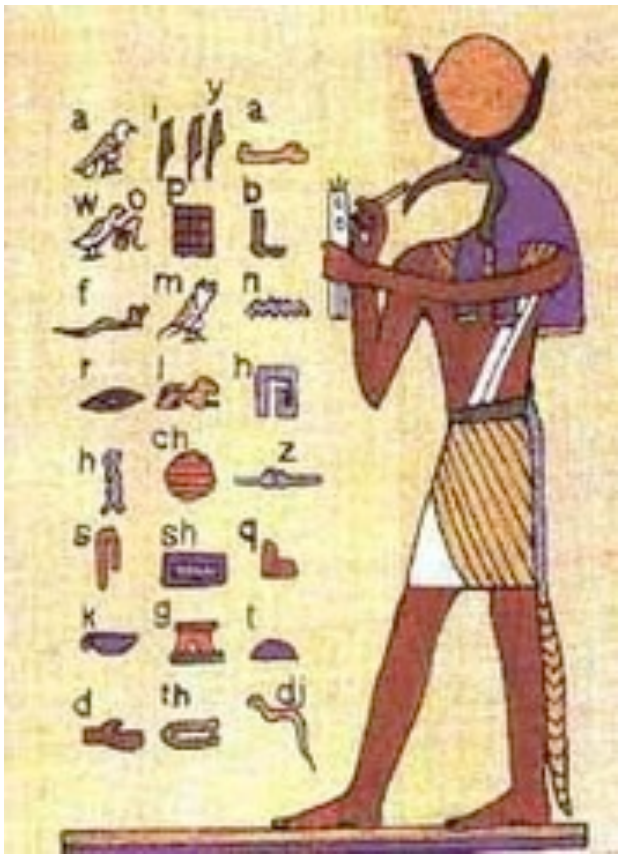


In generale un FA non basta.

Il diagramma mostra una gerarchia di grammatiche, via via più complesse, ed i relativi automi riconoscitori.

Il livello più alto (Tipo 0) è associato alle Macchine di Turing, un modello di calcolo universale.

Metalinguaggi: linguaggi che definiscono altri linguaggi



HTML versus XML

- HTML è un linguaggio di marcatori per:
  1. Definire il modo in cui il contenuto della pagina sarà visualizzato
  2. Indicare i link ad altri file (immagini, altre pagine HTML, ecc.)

Problema: non vi sono indicazioni circa il contenuto, ossia il significato dei dati!

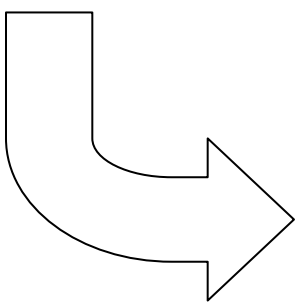
HTML definisce la forma grafica

```
<html>
<body>
Note:<br>
to: <i>Luca</i><br>
from: <i>Carlo</i><br>
title: <b>Appuntamento</b><br>
Ricordati la riunione di oggi
</body>
<html>
```

a capo

corsivo

neretto



Note:
to: *Luca*
from: *Carlo*
title: **Appuntamento**
Ricordati la riunione di oggi

XML definisce il contenuto

i tag sono semantici

```
<?xml version="1.0"?>
<note>
      <to>Luca</to>
      <from>Carlo</from>
      <title>Appuntamento</title>
      <message>Ricordati la riunione di oggi</message>
</note>
```

non appartengono ad un vocabolario predefinito

XML è un metalinguaggio

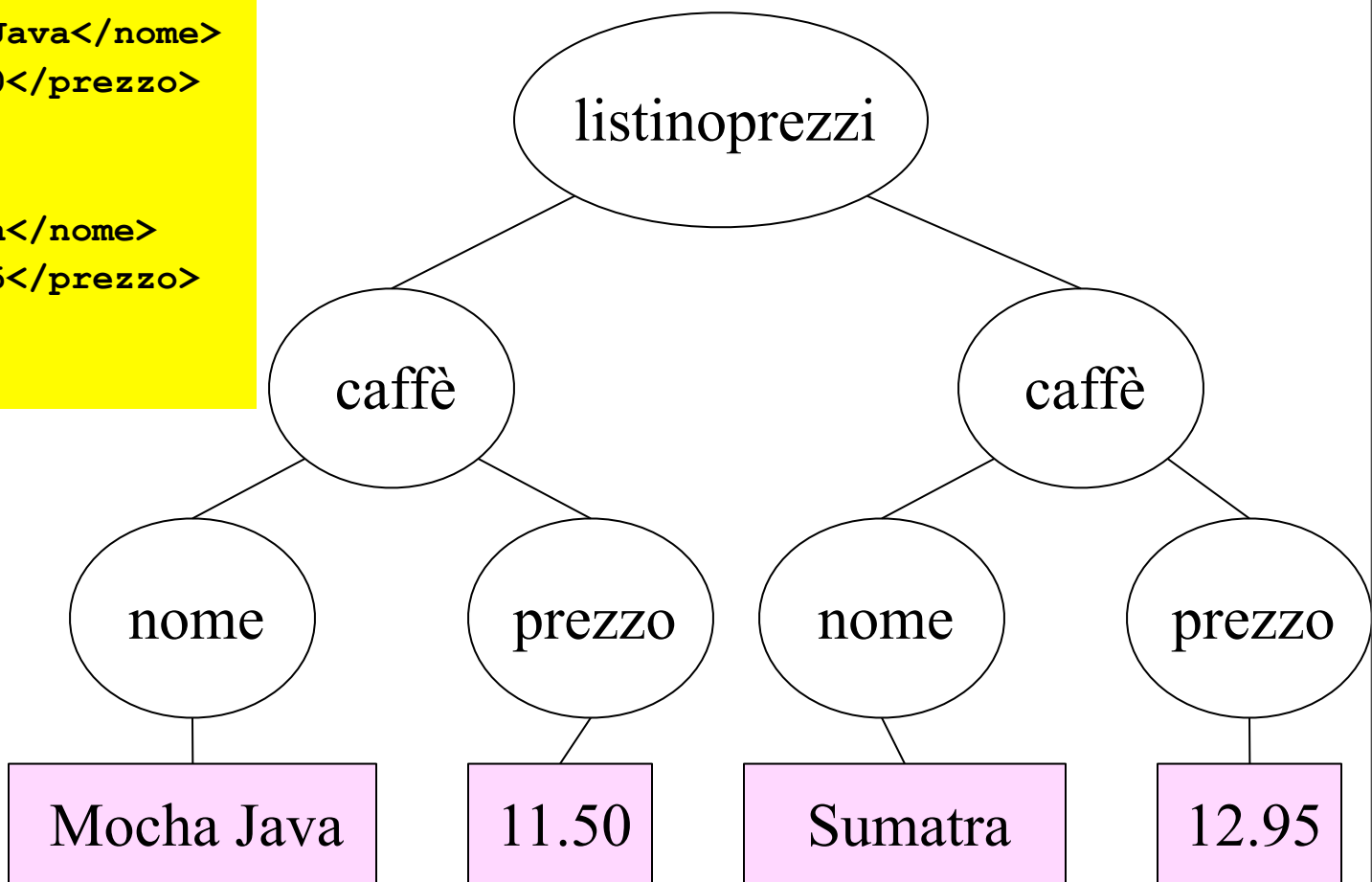
un metalinguaggio è un linguaggio con cui definire altri linguaggi

```
<listinoprezzi>
        <caffe>
                <nome>Mocha Java</nome>
                <prezzo>11.50</prezzo>
        </caffe>
        <caffe>
                <nome>Sumatra</nome>
                <prezzo>12.95</prezzo>
        </caffe>
</listinoprezzi>
```

Linguaggio di marcatori adatto per il listino prezzi di una rivendita di caffè

La struttura di un file XML

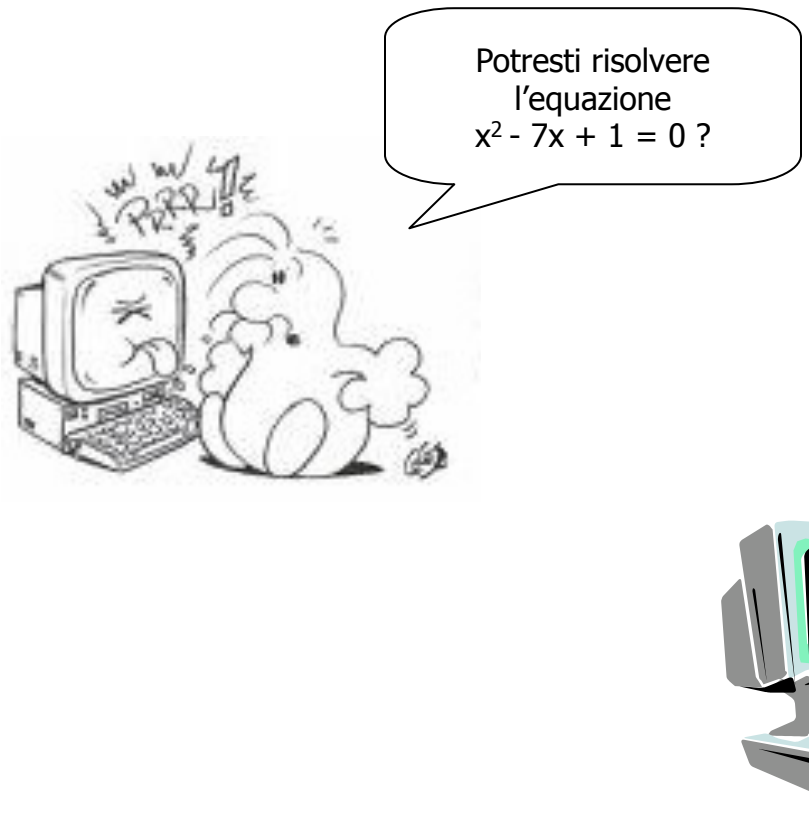
```
<listinoprezzi>
    <caffe>
        <nome>Mocha Java</nome>
        <prezzo>11.50</prezzo>
    </caffe>
    <caffe>
        <nome>Sumatra</nome>
        <prezzo>12.95</prezzo>
    </caffe>
</listinoprezzi>
```



Linguaggi che descrivono algoritmi

Linguaggi di programmazione

Per far sì che il calcolatore elabori i dati nel modo voluto occorre codificare l'algoritmo (= metodo di calcolo) nell'unico linguaggio che la macchina comprende: il codice macchina.



Un programma eseguibile deve essere "scritto" in linguaggio macchina (binario)

Linguaggi di programmazione

Per ovviare alla difficoltà di programmare in linguaggio macchina sono stati introdotti i linguaggi di programmazione, ad esempio: Java, C, C++, BASIC, Pascal, …

Proprietà:

- non ambigui (ogni frase ha un solo albero sintattico)
- concisi
- espressivi (se "all purpose" devono poter codificare qualunque algoritmo)
- alto livello (si basano su astrazioni per essere vicini al nostro modo di pensare)
- …

Codice in un linguaggio d'alto livello

```
1 var j, frame = -1, durata = 150, timeout_id = null;
2 var images = new Array(20);
3 function advance() {
4   for (j = 0; j < 19; j++) {
5     document.images[j].src = document.images[j+1].src;
6   }
7   if (frame == -1)                                    test
8     document.images[19].src = pics[randNum(8)].src;   caso "vero"
9   else
10    document.images[19].src = pics[frame].src;        caso "falso"
11  timeout_id = setTimeout("animate()", durata);
12 }
```

Figura 7.14 Un frammento di programma in un linguaggio di alto livello.

Backus-Naur Form (BNF)

E' il formalismo utilizzato per descrivere la sintassi dei linguaggi di programmazione, ed è un modo per definire grammatiche.

```
<statement> ::= <block>
        | "assert" <expression> [":" <expression>] ";"
        | "if" <par expression> <statement> ["else" <statement>]
        | "for" "(" <for control> ")" <statement>
        | "while" <par expression> <statement>
        | "do" <statement> "while" <par expression> ";"
        | "try" <block> <catches>
        | "try" <block> [<catches>] "finally" <block>
        | "switch" <par expression> "{" <switch block statement groups> "}"
        | "synchronized" <par expression> <block>
        | "return" [<expression>] ";"
        | "throw" <expression> ";"
        | "break" [<identifier>] ";"
        | "continue" [<identifier>] ";"
        | ";"
        | <statement expression> ";"
        | <identifier> ":" <statement>
```

La parsificazione

Fasi di costruzione di un parser:

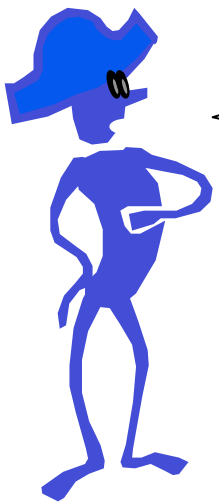
grammatica → automa → programma parser

Generatori di parser: sono programmi che, data una grammatica, generano automaticamente il parser:

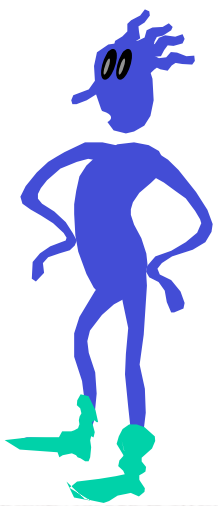
AnaGram, BYacc e BYACC/J, ...

Per un elenco (incompleto) vedi:
http://it.wikipedia.org/wiki/Generatore_di_parser

Qual è il "significato" delle espressioni un linguaggio di programmazione?



Una "frase" in Pascal o in Java è un programma, dunque il suo significato è il codice macchina generato dal compilatore



se ne può dare una definizione più intelleggibile?

La semantica operazionale

La semantica operazionale è un insieme di regole di produzione che definiscono quale sia la computazione di una "frase" di un linguaggio di programmazione.
Uno stato

$$s = \{L_1 \mapsto V_1, \ldots, L_k \mapsto V_k\}$$

è una mappa finita da locazioni in valori (i.e. una rappresentazione astratta della memoria), e con

$$s \uplus (L \mapsto V)$$

indichiamo lo stato s' in tutto simile ad s salvo che è definito su L (s potrebbe non esserlo), dove vale V.

La semantica operazionale

Allora la semantica di un'assegnazione $L := E$ produce l'effetto descritto nella regola:

$$\frac{\langle E, s\rangle \Rightarrow V}{\langle L := E\ ,\ s\rangle \longrightarrow (s \uplus (L \mapsto V))}$$

posto che V sia il valore dell'espressione E se valutata in s.

La semantica operazionale

In modo analogo possiamo descrivere la composizione sequenziale:

$$\frac{\langle C_1, s\rangle \longrightarrow \langle C_1', s'\rangle}{\langle C_1; C_2\ , s\rangle \longrightarrow \langle C_1'; C_2\ , s'\rangle} \qquad \frac{\langle C_1, s\rangle \longrightarrow s'}{\langle C_1; C_2\ , s\rangle \longrightarrow \langle C_2, s'\rangle} \qquad \frac{}{\langle \mathbf{skip}, s\rangle \longrightarrow s}$$

la selezione

$$\frac{\langle B, s\rangle \Rightarrow \mathbf{true}}{\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', s\rangle \longrightarrow \langle C, s\rangle} \qquad \frac{\langle B, s\rangle \Rightarrow \mathbf{false}}{\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', s\rangle \longrightarrow \langle C', s\rangle}$$

l'iterazione

$$\frac{\langle B, s\rangle \Rightarrow \mathbf{true}}{\langle \mathbf{while}\ B\ \mathbf{do}\ C, s\rangle \longrightarrow \langle C; \mathbf{while}\ B\ \mathbf{do}\ C, s\rangle} \qquad \frac{\langle B, s\rangle \Rightarrow \mathbf{false}}{\langle \mathbf{while}\ B\ \mathbf{do}\ C, s\rangle \longrightarrow s}$$

La semantica operazionale

In modo analogo possiamo descrivere la composizione sequenziale:

$$\frac{\langle C_1, s\rangle \longrightarrow \langle C_1', s'\rangle}{\langle C_1; C_2\,, s\rangle \longrightarrow \langle C_1'; C_2\,, s'\rangle} \qquad \frac{\langle C_1, s\rangle \longrightarrow s'}{\langle C_1; C_2\,, s\rangle \longrightarrow \langle C_2, s'\rangle} \qquad \frac{}{\langle \mathbf{skip}, s\rangle \longrightarrow s}$$

la selezione

$$\frac{\langle B, s\rangle \Rightarrow \mathbf{true}}{\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', s\rangle \longrightarrow \langle C, s\rangle} \qquad \frac{\langle B, s\rangle \Rightarrow \mathbf{false}}{\langle \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C', s\rangle \longrightarrow \langle C', s\rangle}$$

l'iterazione

$$\frac{\langle B, s\rangle \Rightarrow \mathbf{true}}{\langle \mathbf{while}\ B\ \mathbf{do}\ C, s\rangle \longrightarrow \langle C; \mathbf{while}\ B\ \mathbf{do}\ C, s\rangle} \qquad \frac{\langle B, s\rangle \Rightarrow \mathbf{false}}{\langle \mathbf{while}\ B\ \mathbf{do}\ C, s\rangle \longrightarrow s}$$

Queste sono le tre strutture di controllo della programmazione strutturata

La semantica operazionale: a che serve?

Ci sono più possibili ragioni:

- descrivere il comportamento delle primitive di un nuovo linguaggio di programmazione, dando indicazioni univoche per implementare interpreti o compilatori

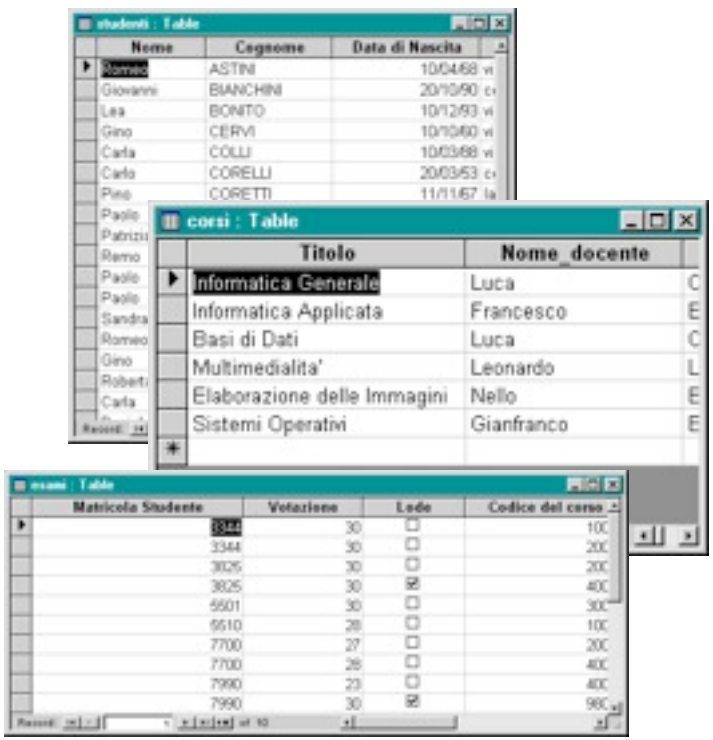
- risolvere ambiguità di costrutti di linguaggi già esistenti (spesso solo descritti in parole)

- disporre di una rappresentazione essenziale di come evolve un programma per potervi ragionare, stablilendone alcune proprietà

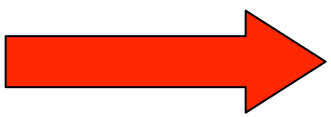
Linguaggi che interrogano

Interrogazioni, query, su un DB

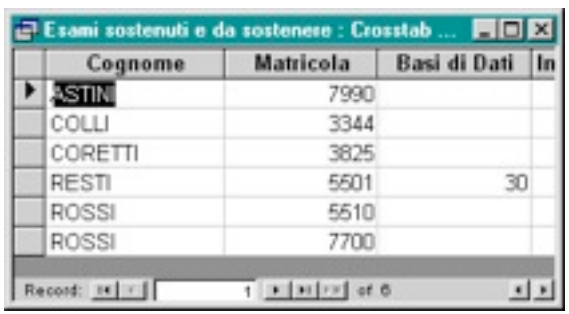
Tabelle



Query



Dynaset



Structured Query Language (SQL)

R

ColA	*ColB*
A	1
B	2
D	3
F	4
E	5

S

SColA	*SColB*
A	1
C	2
D	3
E	4

R JOIN $_{R.ColA = S.SColA}$ S

A	1	A	1
D	3	D	3
E	5	E	4

R JOIN $_{R.ColB = S.SColB}$ S

A	1	A	1
B	2	C	2
D	3	D	3
F	4	E	4

SQL rappresenta una query attraverso la combinazione di operazioni dell'algebra relazionale (operazioni su relazioni = tabelle).

Date le relazioni:

Guidatore(Nome, Cognome, NroPatente)

Automobile(Targa, Marca, Modello, NroPatente)

Domanda

Trovare i guidatori con le automobili associate mantenendo tutti i guidatori della tabella

SQL

```
select *
from Guidatore left join
        Automobileon
       (Guidatore.NroPatente =
        Automobile.NroPatente)
```

Storia di un assioma

Quando Georg Cantor "inventò" gli insiemi questi esistevano in un universo imprecisato di collezioni.

G. Cantor

G. Frege

Gottlob Frege, inventore del calcolo dei predicati, asserì che ogni predicato ha un insieme come estensione.

$$\text{Se } X = \{x \mid x \notin x\} \text{ allora } X \in X \Leftrightarrow X \notin X.$$

B. Rassell

La comprensione (di Zermelo) è una query

L'assioma di comprensione di Zermelo sceglie in un insieme dato tutti gli elementi che soddisfano un predicato P:

$$A = \{x \in U \mid P(x)\}$$

L'assioma stabilisce che A esiste se esiste U (A è incluso in U) e che A è l'estensione di P in U. Possiamo pensare:

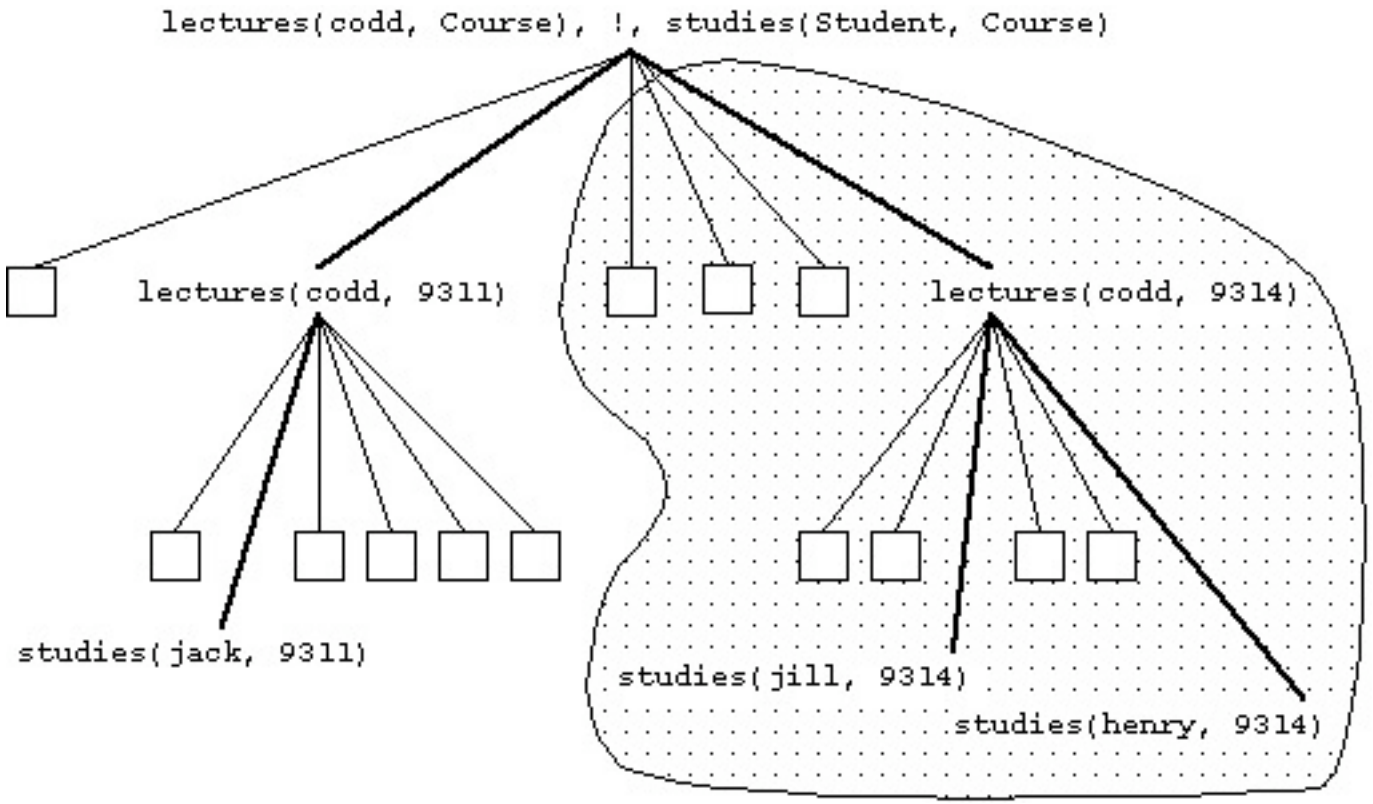
- U è il DB
- P è la query
- A è il risultato della query

Query come clausole di Horn: il PROLOG

```
man(david).
man(john).
woman(suzie).
woman(eliza).
parent(david, john).
parent(john, eliza).
parent(suzie, eliza).

father(X,Y) :- parent(X,Y), man(X).
mother(X,Y) :- parent(X,Y), woman(X).

?- mother(suzie, elsia).
Yes

?-father(david, X).
X = john
Yes
```

R. Kowalsy

Query come clausole di Horn: il PROLOG



I linguaggi della logica

La formalizzazione del ragionamento, da Aristotele a Bool, da Frege a Russell, al nostro Peano ...

Aristotele

G. Bool

G. Frege

B. Rassell

G. Peano

ha prodotto una famiglia di formalismi per esprimere asserzioni, dimostrazioni e teorie.

Le formule e le derivazioni della logica

Nella logica esistono due entità sintattiche: le formule:

$$\exists x A(f(x), y) \wedge \neg \forall y B(f(y), y)$$

e le derivazioni:

$$\dfrac{\dfrac{\dfrac{[A \wedge B]}{B} \qquad \dfrac{\dfrac{[A \wedge B]}{A} \qquad [A \to (B \to C)]}{B \to C}}{C}}{\dfrac{A \wedge B \to C}{(A \to (B \to C)) \to (A \wedge B \to C)}}$$

Qualche conclusione

I linguaggi formali sono uno strumento utile per:
- definire e organizzare dati
- descrivere algoritmi
- interrogare basi di dati

ma anche:
- costruire modelli astratti della computazione (e.g. Macchine di Turing)
- studiare l'efficienza degli algoritmi e la difficoltà di risolvere problemi di calcolo
- formalizzare asserzioni e ragionamenti
- verificare (semi) automaticamente programmi e protocolli
- ...