Fondamenti di Informatica per la Sicurezza a.a. 2003/04

◇ *Grammatiche* ◇

Stefano Ferrari

Università degli Studi di Milano

Dipartimento di Tecnologie dell'Informazione

Funzione calcolabile

Concetto intuitivo:

- esiste una sequenza univoca di istruzioni non ambigue
- eseguite da un operatore preciso e scrupoloso
- che in un numero finito di passi
- manipoli un dati in ingresso in un valore in uscita

algoritmo

Dualità funzione-algoritmo

Differenza tra una funzione computabile e l'algoritmo che la computa:

- una funzione f è una relazione tra un argomento x e un risultato $f(x)$
- un algoritmo è un metodo per calcolare una funzione

È la stessa relazione che lega un teorema e la sua dimostrazione

Teoria della computabilità

La Teoria della Computabilità studia le funzioni astrattamente calcolabili senza preoccuparsi dell'efficienza dell'algoritmo che le computa.

- nasce negli anni '30, con l'esigenza, sorta nell'ambito degli studi di logica,
  - di fornire un equivalente rigoroso del concetto intuitivo di algoritmo,
  - di indagare le possibilità ed i limiti dei metodi effettivi.

Cosa si può calcolare?

Non tutte le funzioni esprimibili matematicamente possono essere calcolate.

- funzioni su $\mathbb{N}^2$
- ogni programma P_n è identificata da un numero $n \in \mathbb{N}$ (numero finito di passi, un numero finito di istruzioni)
- definizione della funzione $h : \mathbb{N} \to \mathbb{N}$:
  1. input k
  2. trova P_k
  3. output $P_k(k) + 1$
- Può esistere j tale che P_j calcola h ($\forall x\ h(x) = P_j(x)$)?:
  - quanto vale $h(j)$?
  - $h(j) = P_j(j) + 1 = h(j) + 1$! Impossibile

Modelli di calcolo

Linguaggi insiemi di stringhe per descrivere (proprietà, algoritmi)

Grammatiche regole per descrivere linguaggi

Automi macchine ideali per il riconoscimento di linguaggi ed esecuzione di algoritmi

Linguaggio – definizioni

- un *alfabeto* è un insieme *finito* e non vuoto di simboli
  - $A = \{a, b, c\}$
  - $A = \{0, 1\}$
- una *stringa* (o *parola*) è una sequenza finita di simboli
  - se $A = \{a, b, c\}$, $aaabacab$ è una stringa su A
- la *lunghezza* di una stringa w, denotata con $|w|$, è il numero di simboli che la compongono
  - $|aaabacab| = 8$
- la stringa vuota ϵ è composta da zero simboli
  - $|\epsilon| = 0$
- $A^0 = \{\epsilon\}$, $A^1 = \{a, b, c\}$, $A^2 = \{aa, ab, ac \ldots\}$, $\ldots$, $A^* = \cup_i A^i$

Linguaggio – definizioni (2)

- sia $w = a_1 \cdots a_n$
  - $a_1 \cdots a_j$, con $j \in \{1, \ldots, n\}$ è un *prefisso* di w
  - $a_j \cdots a_n$, con $j \in \{1, \ldots, n\}$ è un *suffisso* di w
  - $a_i \cdots a_j$, con $i, j \in \{1, \ldots, n\}$ è una *sottostringa* di w
  - ϵ è prefisso, suffisso e sottostringa di w
- la *concatenazione* di due stringhe v e w è la striga vw
  - la concatenazione è un'operazione associativa
  - ϵ ne è l'identità
- un *linguaggio (formale)* L è un sottoinsieme di A^*
  - una *frase* di un linguaggio è una stringa appartenente al linguaggio stesso

Esempio

- il linguaggio delle frasi in italiano
  - alfabeto: $A = \{a, b, \ldots, z,$ <spazio>$\}$
  - stringhe su A: $cosa$, $il\,porto$, $ghsde\,afe\,li$
  - frasi: $la\,gatta\,corre$
- il linguaggio delle espressioni aritmetiche
  - alfabeto: $A = \{0, \ldots, 9, +, -, :, \times, (,)\}$
  - stringhe su A: 980, $34 + 61$, $: ()345+)$
  - frasi: $(25 + 12) : 3$

Grammatica formale

Definisce in modo rigoroso un linguaggio su A

- Tecnicamente è una quadrupla $\langle T, V, P, S \rangle$
  - $T \subset A^*$, insieme finito di simboli terminali
    - un simbolo, una stringa, o la stringa vuota di A
  - V, insieme finito di simboli non terminali
    - sono meta-simboli di appoggio
    - rappresentano categorie sintattiche
  - P, insieme di regole di riscrittura del tipo $\alpha \rightarrow \beta$, con α, β stringhe su $T \cup V$
  - $S \in V$, simbolo iniziale, detto *scopo*

Generazione del linguaggio

- Si parte dallo scopo S
- Si applica una regola di produzione $\alpha \rightarrow \beta$
  - si cerca la presenza della sotto-sequenza α
  - la si sostituisce con β
  - α e β sono chiamate *forme di frase*
- Si procede finché non si ottiene una stringa con soli simboli terminali
- tale stringa è una frase del linguaggio
- tutte (e solo) le frasi del linguaggio si ottengono con questo procedimento

Esempio

- grammatica: $\langle T, V, P, S \rangle$
  - $A = \{0, 1\}$
  - $T = A^*$
  - $V = \{X\}$
  - $P = \{X \to 0, X \to 1X, X \to 0X\}$
  - $S = X$
- linguaggio: $\{0, 10, \ldots, 0110, \ldots\}$
  - $S = X \to 0$
  - $S = X \to 1X \to 10$
  - $S = X \to 0X \to 01X \to 011X \to 0110$

Grammatiche BNF

La notazione BNF (Backus-Naur Form) è più conveniente per rappresentare grammatiche

- le regole di produzione sono della forma $\alpha ::= \beta$
- i meta-simboli in V sono della forma $\langle nome \rangle$
- il meta-simbolo speciale | (pipe) è usato per l'alternativa
  - $\alpha ::= \beta_1,\ \alpha ::= \beta_2,\ \ldots,\ \alpha ::= \beta_n$
  - $\alpha ::= \beta_1 | \beta_2 | \ldots | \beta_n$

Nota su http://www.faqs.org/rfcs/rfc2234.html si può trovare la definizione di Augmented BNF (ABFN)

Esempio

- grammatica: $\langle T, V, P, S \rangle$
  - $A = \{il, gatto, topo, sasso, mangia, beve\}$
  - $T = A$
  - $V = \{\langle frase \rangle, \langle soggetto \rangle, \langle articolo \rangle, \langle nome \rangle, \langle verbo \rangle, \langle compl\_ogg \rangle\}$
  - $P = \{$

    $\langle frase \rangle ::= \langle soggetto \rangle \langle verbo \rangle \langle compl\_ogg \rangle$

    $\langle soggetto \rangle ::= \langle articolo \rangle \langle nome \rangle$

    $\langle articolo \rangle ::= il$

    $\langle nome \rangle ::= gatto | topo | sasso$

    $\langle verbo \rangle ::= mangia | beve$

    $\langle compl\_ogg \rangle ::= \langle articolo \rangle \langle nome \rangle \}$
  - $S = \langle frase \rangle$
- esempio: $il\ gatto\ mangia\ il\ sasso$

Classificazione di Chomsky

È possibile caratterizzare le grammatiche sulla base del tipo delle loro produzioni

- Tipo 3 (grammatiche regolari)
- Tipo 2 (context-free)
- Tipo 1 (context-sensitive)
- Tipo 0 nessuna restrizione

Grammatiche di tipo 3

Grammatiche regolari

- lineari a destra ($F ::= \alpha, F ::= \alpha G$), con $F, G \in V, \alpha \in T^*$
- lineari a sinistra ($F ::= \alpha, F ::= G\alpha$), con $F, G \in V, \alpha \in T^*$
- Esempi:
  - $F ::= aG|a,\ G ::= bG|b$
  - $F ::= Fa|a$
- Il linguaggio generato si dice di tipo 3 (o linguaggio regolare)

Grammatiche di tipo 2

Grammatiche context-free (libere dal contesto)

- $F ::= \alpha$, con $F \in V$ e $\alpha \in (T \cup V)^*$
- Esempi:
  - $F ::= aG|a,\ G ::= Gb|b$
  - $F ::= aG|b,\ G ::= Fc$
  - $F ::= aFc|b$
- Si dice che il linguaggio è di tipo 2
  - tipicamente, i linguaggi di programmazione appartengono a questo tipo

Grammatiche di tipo 1 e 0

- tipo1: Grammatiche context-sensitive (dipendenti dal contesto)
  - $\alpha F \beta ::= \alpha \gamma \beta$, con $\alpha,\ \beta,\ \gamma \in (T \cup V)^*$, $\gamma \neq \epsilon$, e $F \in V$
  - $S ::= \epsilon$
  - Esempio: $aFb ::= abb$
- Grammatiche di tipo 0, nessuna restrizione

Gerarchia di grammatiche e linguaggi

- ogni grammatica di tipo n è anche una grammatica di tipo $n-1$
- un linguaggio è di tipo n se è generato da una grammatica di tipo n, ma non da una di tipo $n+1$

Esempi:

- linguaggio di tipo 3: $\{a^m b^n \mid m, n \geq 0\}$
- linguaggio di tipo 2: $\{a^n b^n \mid n \geq 0\}$
- linguaggio di tipo 1: $\{a^n b^n c^n \mid n \geq 0\}$
- linguaggio di tipo 0: $\{a^n \mid n$ è un numero primo$\}$