Capitolo 1

Argomenti introduttivi

1.1 Alfabeto

Definizione 1 *Un alfabeto è un insieme finito e non vuoto di simboli.*

Generalmente indichiamo gli alfabeti con la lettera greca Σ (*sigma* maiuscola).

1.1.1 Esempi di alfabeti

- $\Sigma = \{0, 1\}$, l'alfabeto binario.
- $\Sigma = \{a, b, \ldots, z\}$, l'alfabeto di tutte le lettere dell'alfabeto.
- $\Sigma = \{0, 1, \ldots, 9\}$, l'alfabeto delle cifre da 0 a 9.

1.2 Stringa

Definizione 2 *Una stringa è una sequenza finita di simboli scelti da un alfabeto.*

1.2.1 Esempi di stringhe

- Dato l'alfabeto $\Sigma = \{0, 1, \ldots, 9\}$, 1492, 0, 777 sono solo alcune delle infinite stringhe estraibili dall'alfabeto.

1.2.2 Lunghezza di una stringa

Una metodologia di classificazione delle stringhe è quella che usa come parametro di confronto la lunghezza di queste.

Definizione 3 *La lunghezza di una stringa è il numero di simboli che compongono la stringa.*

Generalmente si indica la lunghezza di una stringa w con $|w|$.

Esempio 1

- $|101| = 3$
- $|ciao| = 4$
- $|00| = 2$

1.2.3 La stringa vuota

Una stringa particolare, estraibile sempre da ogni alfabeto, è la stringa vuota. Essa ha lunghezza zero. E' indicata generalmente dalla lettera greca ϵ (*epsilon*) o dalla lettera greca λ (*lamda*). In questo testo sarà usata una o l'altra notazione, senza particolari preferenze.

Osservazione 1

$$|\lambda| = 0$$

1.2.4 Concatenazione tra stringhe

Prese due stringhe s_1 e s_2 denotiamo con $s_1 s_2$ la nuova stringa risultante dalla concatenazione di s_1 con s_2, cioò la stringa ottenuta appendendo in fondo a s_1 la stringa s_2.

Esempio 2 *Data* $s_1 = ci$ *e* $s_2 = ao$, $s_1 s_2 = ciao$

Osservazione 2

$$|s_1 s_2| = |s_2 s_1|$$

1.2.5 Potenze di un alfabeto

Definizione 4 *Dato un alfabeto* Σ *e un numero* n*, definiamo come* Σ^n *l'insieme delle stringhe estratte da* Σ *di lunghezza* n*.*

Esempio 3

- $\Sigma^0 = \{\epsilon\}$
- $\Sigma = \{a, b\}, n = 2 \rightarrow \Sigma^n = \{aa, ab, ba, bb\}$

Osservazione 3 Σ *è l'insieme di tutte le stringhe ottenute da un alfabeto. E' ovviamente un insieme infito, ottenuto dall'unione di tutti gli* Σ^n *con* n *che varia tra* 0 *e* ∞*.*

1.3 Linguaggio

Definizione 5 *Un insieme di stringhe L preso da un insieme Σ è definito un linguaggio.*

Generalmente le stringhe che appartengono ad L sono scelte secondo un criterio di appartenenza, o comunque non in maniera casuale.

1.3.1 Esempi di linguaggi

Esempio 4 *Dato $\Sigma = \{0, 1\}$ costruiamo il linguaggio tale per cui le stringhe sono composte da un numero uguale di 0 seguito da un numero uguale di 1. Le stringhe possono avere lunghezza uguale a 0.*

Inanzitutto scriviamo in maniera simbolica il linguaggio che vogliamo costruire.

$$L = \{w \in \Sigma | w = 0^n 1^n, n \geq 0\}$$

Ci accorgiamo inanzitutto che questo linguaggio è infinito, ci sono vincoli sulla lunghezza minima (se così vogliamo chiamarli, visto che la lunghezza minima è 0) ma non ce ne sono sulla lunghezza massima.

Possiamo quindi scrivere il nostro linguaggio in questo modo

$$L = \{\epsilon, 01, 0011, 000111, \ldots\}$$

Esempio 5 *Dato $\Sigma = \{a, b\}$ costruiamo il linguaggio delle stringhe palindrome in Σ con lunghezza maggiore di zero.*

In simboli:

$$L = \{w \in \Sigma |\ w \text{ è palindroma}, |w| > 0\}$$

Anche questo linguaggui è infinito.

$$L = \{a, b, ab, ba, aa, bb, aaa, aba, bbb, bab, ldots\}$$

1.4 Problema

Definizione 6 *Dato un linguaggio L ed una stringa w definiamo come problema la questione che porta a dire se w appartiene o no ad L.*

Dare una risposta ad un problema può essere un'operazione banale ma anche difficilissima.

Esempio 6 (Problema banale) *Dire se la stringa 'abba' appartiene al linguaggio delle stringhe palindrome ottenute da* $\Sigma = \{a, b\}$.

Basta un semplice algoritmo e anche il computer meno performante riesce a dare una risposta.

Esempio 7 (Problema non così banale) *Dire se il numero 2038074743 appartiene al linguaggio costruito da* $\Sigma = \{0, \ldots, 9\}$ *tale per cui il numero rappresentato da* $w \in L$ *è primo.*

1.5 Gerarchia di Chomsky

Anche se la definizione di linguaggio è abbastanza generica, esistono diverse categorie di linguaggi.

- Regolari, associati alla grammatiche di tipo 3, riconoscibili con gli automi a stati finiti.
- Context-free, di tipo 2, riconoscibili con gli automi a pila non deterministici.
- Context-sensitive, di tipo 1, riconoscibili con le macchine di Turing non deterministiche a nastro limitato.
- Ricorsivamente enumerabili, di tipo 0, riconoscibili con le macchine di Turing

Capitolo 2

Automi a stati finiti deterministici

2.1 Definizione informale

An automaton is supposed to run on some given sequence of inputs in discrete time steps. An automaton gets one input every time step that is picked up from a set of symbols or letters, which is called an alphabet. At any time, the symbols so far fed to the automaton as input, form a finite sequence of symbols, which finite sequences are called words. An automaton contains a finite set of states. At each instance in time of some run, the automaton is in one of its states. At each time step when the automaton reads a symbol, it jumps or transitions to another state that is decided by a function that takes the current state and symbol as parameters. This function is called the transition function. The automaton reads the symbols of the input word one after another and transitions from state to state according to the transition function, until the word is read completely. Once the input word has been read, the automaton is said to have stopped and the state at which automaton has stopped is called the final state. Depending on the final state, it's said that the automaton either accepts or rejects an input word. There is a subset of states of the automaton, which is defined as the set of accepting states. If the final state is an accepting state, then the automaton accepts the word. Otherwise, the word is rejected. The set of all the words accepted by an automaton is called the "language of that automaton". Any subset of the language of an automaton is a language recognized by that automaton.

In short, an automaton is a mathematical object that takes a word as input and decides either to accept it or reject it. Since all computational problems are reducible into the accept/reject question on words (all problem instances can be represented in a finite length of symbols), automata theory plays a crucial role in computational theory.

2.2 Definizione formale

Un automa a stati finiti deterministico (DFA) è costituito da:

- Un insieme finito di stati, Q.
- Un insieme finito di simboli di input, Σ.
- Una funzione di transizione δ, che ha come parametro uno stato ed un simbolo e ritorna uno stato.
- Uno stato di partenza, appartenente a Q
- Un insieme di stati accettanti, sottoinsieme di Q.

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

2.3 La funzione di transizione estesa

δ^* è detta funzione di transizione estesa. Essa prende in input uno stato e una stringa e ritorna uno stato.

E' definita per induzione come segue:

Definizione 7

- Caso base: $\delta^*(q, \lambda) = q$
- Passo induttivo: suppniamo $w = xa$, quindi:

$$\delta^*(q, w) = \delta(\delta^*(q, x), a)$$

2.4 Linguaggio dei DFA

Dato un DFA A, chiamiamo $L(A)$ il linguaggio accettato da A, cioè:

$$L(A) = \{w | \delta^*(q_0, w) \in F\}$$

Tutti i linguaggi accettabili da un DFA qualsiasi sono **linguaggi regolari**.

Capitolo 3

Automi a stati finiti non-deterministici

Un NFA è un automa a stati finiti non deterministico. Un NFA ha la caratteristicha che in un determinato istante della sua computazione può trovarsi in più di uno stato. Questa sua abilità è generalmente espressa con l'abilità dell'automa di provare ad "indovinare" qualcosa del suo input.

Un NFA è praticamente uguale ad un DFA, l'unica differenza sta nella funzione δ che non ritorna più un solo stato ma un insieme di stati.

3.1 δ^*

Anche in questo caso la δ^* è definita per induzione.

Definizione 8

- *Caso base:* $\delta^*(q, \lambda) = \{q\}$
- *Passo induttivo:* $w = xa$, $\delta^*(q, x) = \{p_1, p_2, \ldots, p_n\}$ *e*

$$\bigcup \delta(p_i, a) = \{r_1, r_2, \ldots, r_m\}$$

quindi $\delta^*(q, x) = \{r_1, r_2, \ldots, r_m\}$

3.2 Equivalenza tra DFA e NFA

Ogni linguaggio accettabile con una NFA è anche accettabile con un DFA. Un NFA è semplicemente una rappresentazione diversa per un DFA, fatta ad esempio per semplificare o rendere più chiaro il funzionamento.

Teorema 3.2.1 *Un linguaggio L accettato da un DFA se e solo se L è accettato anche da un NFA.*

E' possibile convertire un NFA in un DFA.

3.3 ϵ-NFA

Un ϵ-NFA è un NFA che permette di fare transizioni che non consumano input, cioè delle ϵ-transizioni. Questo vuol dire che l'automa può fare transizioni da solo, senza ricevere un input. Come per gli NFA, questa funzionalità non aumenta la classe di linguaggi riconoscibili tramite automi a stati finiti, rimane la stessa dei DFA.

3.3.1 δ di un ϵ-NFA

La funzione di transizione di un ϵ-NFA prende in input uno stato e un elemento di $\Sigma \cup \{\epsilon\}$ e ritorna un insieme di stati.

3.3.2 ϵ-chiusura

Importantissimo per i eNFA è il concetto di ECLOSE di uno stato, è l'insieme di tutti gli stati raggiungibili con e-transizioni senza consumare input. E' definita per induzione.

Definizione 9

- Caso base: q è nell'ECLOSE(q).
- Passo induttivo: Se p è nell'ECLOSE(q) e c'è una ϵ-transizione dallo stato p allo stato r, quindi r è nell'ECLOSE(q).

Capitolo 4

Grammatiche regolari

Le grammatiche regolari sono di **tipo 3** sulla gerarchia di Chomsky.

Le grammatiche regolari sono tutte quelle accettabili con un DFA e rappresentabili con un espressione regolare.

4.1 Definizione formale

Definizione 10

Chiamiamo grammatica regolare una 4-upla $G = (V, T, P, S)$ dove:

- V è l'insieme delle variabili, esse verrano usate nelle regole di produzione, sono dette anche non-terminali.
- T è l'insieme dei terminali, simboli che formano le stringhe del linguaggio.
- P, è l'insieme delle regole di produzione, la rappresentazione ricorsiva della definizione del linguaggio.
- S. è il simbolo di partenza, una delle variabili di V.

Imponiamo su P dei vincoli:

- ϵ può solo comparire in $S \to \epsilon$
- Le produzioni sono tutte lineari a sinistra o a destra

4.2 Lineare a destra o sinistra

Definizione 11 *Una produzione è lineare a destra se è nel formato:*

* $A \to aB$

oppure

* $A \to c$.

Le variabili, se ci sono, stanno sempre a destra di tutti i terminali.

Definizione 12 *Una produzione è lineare a sinistra se è nel formato:*

* $A \to Ba$

oppure

* $A \to c$.

Le variabili, se ci sono, stanno sempre a sinistra di tutti i terminali.

Non sempre è possibile convertire da lineare a destra a lineare a sinistra, o viceversa.

4.3 Espressioni regolari

Definiamo induttivamente un'espressione regolare

Definizione 13

Base:

- ϵ e $\emptyset$ sono espressioni regolari
- Se a è un simbolo, allora a è un espressione regolare. $L(a) = \{a\}$
- Una variabile è una variabile e rappresenta un linguaggio.

Induzione:

- Se E ed F sono E.R. allora $E + F$ (unione) è espressione regolare.
$$L(E + F) = L(E) \cup L(F)$$
.
- Se E ed F sono E.R. allora EF (concatenazione) è espressione regolare.
$$L(EF) = L(E)L(F)$$
.
- Se E è E.R. allora E^* è espressione regolare.
$$L(E) = (L(E))^*$$
- SE E è E.R. allora (E) è espressione regolare.
$$L((E)) = L(E)$$

4.4 Pumping Lemma per linguaggi regolari

Supponiamo L un linguaggio regolare. Esiste quindi una costante n tale che se w è una stringa di L allora $|w| \geq n$. Possiamo quindi scrivere $w = xyz$ tale che:

- $y \neq \epsilon$
- $|xy| \leq n$
- Per ogni $k \geq 0$, xy^kz è in L

Capitolo 5

Automi a pila

5.1 Definizione informale

Gli automi a pila sono automi a stati finiti che possono interagire con uno stack mentre eseguono.

Lo stack svolge una funzione di memoria elementare da cui, in ogni mossa, si legge un simbolo (che viene usato insieme al simbolo di input nella δ) e sul quale viene scritto uno o più simboli in contemporanea al cambiamento di stato risultante dalla δ.

Un PDA riconosce tutti i linguaggi context-free.

5.2 Definizione formale

Un automa a pila P è una n-upla così composta:

- Q, un insieme finito di stati
- Σ, un insieme finito di simboli di input
- Γ, un insieme di **simboli di stack**, sono i simboli che possiamo pushare sullo stack.
- δ, la funzione di transizione. Prende come argomento tre parametri, lo stato corrente, il simbolo di input (oppure ϵ) e il *top()* dello stack. Ritorna un insieme di coppie (p, γ) dove p è il nuovo stato e γ è il simbolo o i simboli da pushare sullo stack.
- q_0, lo stato di inizio
- Z_0, il simbolo posto inizialmente sullo stack
- F, l'insieme degli stati di accettazione

5.3 Descrizioni istantanee

$$(q, aw, XB) \rightarrow (p, w, AB)$$

Definizione 14 *$\rightarrow^*$ è definita per induzione:*

- *$I \rightarrow^* I$ per ogni descrizione istantanea*
- *$I \rightarrow^* J$ se esiste K, $I \rightarrow K$ e $K \rightarrow^* J$.*

5.3.1 Inoltre

- Se una sequenza di ID è valida per un PDA, anche la computazione formata aggiungendo alla stringa di input di ogni ID un'altra stringa è valida
- Se una computazione è valida, aggiungendo in fondo allo stack di ogni ID dei simboli otteniamo una computazione valida
- Se una computazione è valida, e una finale dell'input non è consumata, rimuovendo questa parte da tutti gli input di ogni ID otteniamo una computazione valida

5.4 Accettazione

Un PDA può accettare sia per stato finale che per stack vuoto.

Un linguaggio ha un PDA che accetta per stato finale se e solo se ha un PDA che lo accetta per stack vuoto.

Generalmente un PDA che accetta per stato finale è diverso da uno che accetta per stack vuoto.

E' possibile convertire un NPDA (che accetta per stack vuoto, N di null) in un LPDA (che accetta per stato finale, L di last), e viceversa.

5.4.1 Per stato finale

$$L(P) = \{w | (q_0, w, Z_0) \rightarrow^* (q, \epsilon, A)\}$$

con q appratenente all'insieme degli stati finaleìi del PDA.

Convesione in stack vuoto

Per prima cosa si pusha un simbolo X_0 sullo stack. Si crea uno stato aggiuntivo che svuota lo stack e a questo stato si collegano tutti gli stati finali con una e-transizione.

5.5 Per stack vuoto

$$N(P) = \{w|(q_0, w, Z_0) \rightarrow^* (q, \epsilon, \epsilon)\}$$

per un qualsiasi stato q.

Conversione in stato finale

Per prima cosa si aggiunge un simbolo X_0 in fondo allo stack. In seguito si aggiunge uno stato finale al quale si può accedere da ogni stato dell'automa con una transizione $\epsilon, X_0/\epsilon$

5.6 PDA deterministici

I PDA possono essere deterministici o non deterministici. Intuitivamente un PDA è deterministico quando non c'è possibilità di scegliere quale mossa svolgere in ogni determinata situazione.

In particolare se $\delta(q, a, X)$ produce come risultato più coppie (s, γ) allora l'automa è ovviamente non deterministico. Inoltre è non deterministico se mentre esiste un $\delta(q, a, X)$ esiste anche un $\delta(q, \epsilon, X)$.

5.6.1 Linguaggi del DPDA

Un DPDA accetta una classe di linguaggi che è tra i linguaggi regolari e i linguaggi CF.

Inanzitutto un DPDA riconosce per stato finale tutti i linguaggi regolari. Questo è semplice da capire perchè alla fine basta che il DPDA simula un automa a stati finiti deterministico, e quindi non fa nulla sullo stack.

Il DPDA può accettare per stack vuoto se e solo se il linguaggio che si sta cercando di accettare gode della proprietà del prefisso.

Se un linguaggio è accettato per stack vuoto da un DPDA allora L è generato da una grammatica context-free non ambigua.

Capitolo 6

Grammatiche Context Free

6.1 Introduzione

Le grammatiche context-free hanno alla base della proprio struttura il concetto della ricorsione. Per introdurre queste grammatiche possiamo partire da un esempio pratico.

Esempio 8 *Consideriamo il linguaggio delle stringhe palindrome, generate a partire dall'alfabeto* $\Sigma = \{0, 1\}$. *Come possiamo definirlo formalmente? Cioè quale criterio ci permette di dire se una stringa* w *è palindroma o no? In questo caso la definizione che torna più comoda è quella ricorsiva.*

- *Se* w *è* ϵ *oppure* 0 *oppure* 1 *allora è palindroma.*
- $0w0$, $1w1$ *è palindroma se e solo se* w *è palindroma.*

Se volessimo quindi rappresentare tutte le stringhe palindrome in $\{0, 1\}^*$ *potremmo scrivere le seguenti regole, che chiamiamo produzioni:*

- $P \to \epsilon$
- $P \to 0$
- $P \to 1$
- $P \to 0P0$
- $P \to 1p1$

Una grammatica context free è una notazione formale per esprimere questi linguaggi ricorsivi.

6.2 Definizione formale

Definizione 15 *Chiamiamo grammatica context-free una n-upla* $G = (V, T, P, S)$ *dove:*

- V *è l'insieme delle variabili, esse verrano usate nelle regole di produzione, sono dette anche non-terminali.*
- T *è l'insieme dei terminali, simboli che formano le stringhe del linguaggio.*
- P*, è l'insieme delle regole di produzione, la rappresentazione ricorsiva della definizione del linguaggio.*
- S*. è il simbolo di partenza, una delle variabili di* V*.*

6.3 Esempi

Esempio 9 *Prendiamo ancora il caso del linguaggio delle stringhe palindrome.*

- $V_{ariabili} = \{V\}$
- $T_{erminali} = \{0, 1\}$
- $P_{roduzioni} = \{\ldots\}$
- $S_{tart} = P$

$$G = (\{V\}, \{0, 1\}, P, V)$$
$$P = \{P \to \lambda, P \to 0, P \to 1, P \to 0P0, P \to 1P1\}$$

Definizione 16 *In* $P \to 101$ P *è detta testa della produzione mentre* 101 *è il corpo della produzione.*

6.4 Derivazioni

E' possibile applicare le regole di produzione di una grammatica per capire se una certa stringa appartiene ad un determinato linguaggio. Possiamo approcciare il problema in due modalità, differenti nell'esecuzione ma equivalenti nel risultato.

Il primo metodo di definizione del linguaggio, chiamato **inferenza ricorsiva** agisce *body to head*. Prendiamo le stringhe appartenenti al linguaggio di ogni variabile e le concateniamo, in ordine corretto, con ogni terminale che appare nella stringa del corpo e concludiamo che la stringa risultante appartiene al linguaggio della variabile di testa.

L'altro metodo di definizione del linguaggiom detto di **derivazione** agice **head to body**. Per prima cosa la variabile di partenza viene espansa in una delle sue produzioni, in seguito si prosegue ad espandere in modo ricorsivo ogni variabile del corpo ottenuto, finchè non si raggiunge una sola stringa di terminali.

Capitolo 7

Macchine di Turing

7.1 Problemi indecidibili

Ogni problema (e quindi ogni linguaggio associato al problema) può essere rappresentato in binario, usando soltanto le cifre 0 ed 1.

- I linguaggi sono tutti i possibili sottoinsiemi dell'insieme infinito $\{w|w = (0+1)*\}$ (quindi l'insieme potenza)
- La macchine di Turing sono N (numerabili ma infinite)
- $|MdT| = \mathbb{N}$ e $|Linguaggi| = 2^{\mathbb{N}}$
- $N < 2^N$

Quindi ci sono sicuramente problemi/linguaggi che non possono essere associati ad alcuna macchina di Turing, od ad alcun algoritmo.

7.1.1 Esempi di problemi indecidibili

Ambiguità di una grammatica

Non esiste una macchina di Turing (e quindi un algoritmo) che ci dice se una grammatica è ambigua

"Ciao mondo"

Non esiste algoritmo che dice se un programma stampa "Ciao mondo" come primo output.

Dimostriamo per assurdo che non esiste.

- Supponiamo di avere un programma H che prende in input un programma P ed un input I e ci dice, tramite "yes" or "no", se il programma P, con input I stampa "Ciao mondo".
- Costruiamo H_1, basato su H, che stampa $Ciao mondo" al posto di stampare "no". H_2 stampa quindi "Ciao mondo" se P con input I non stampa "Ciao mondo"
- Costruiamo H_2, basato su H_1, che prende come input solo P e si chiede cosa faccia P dato in input se stesso. H_2 stampa "si" se P stampa "Ciao mondo", oppure stampa "Ciao mondo" se P non stampa "Ciao mondo".
- Costruiamo H_p, basato su H_2, che prende come input il codice di se stesso. Questo programma non può esistere: se H_p (in input) stampasse "Ciao mondo", H_p dovrebbe stampare "yes" (che è assurdo visto che abbiamo detto che stampa "Ciao mondo"), mentre se H_p (in input) stampasse "yes", H_P dovrebbe stampare "Ciao mondo" (perchè H_p in input non ha stampato "Ciao mondo", che è assurdo).

Il **problema "Ciao mondo" è indecidibile**.

7.2 Macchina di Turing

Una macchina di Touring è formata da:

- un controllo finito, che può trovarsi in uno tra un insieme di stati possibili.
- un nastro, diviso in celle che possono contenere simboli, di lunghezza infinita
- un input, cioè una stringa di lunghezza finita di simboli scelti da un alfabeto di input, piazzato sul nastro
- una testina, un elemento che punta sempre ad una sola cella del nastro, serve a leggere/scrivere la cella.

La macchina di Touring può effettuare una mossa, una funzione dipendente dallo stato del controllo finito e dal simbolo letto dalla testina. In una mossa la macchina di Touring:

- Cambierà stato (e il nuovo stato potrebbe essere quello in cui già si trovava)
- Scriverà sulla cella corrente un nuovo simbolo
- Sposterà la testina di una posizione a sinistra o a destra (in alcune versioni potrebbe anche rimanere ferma, senza limitazioni).

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

- Q: insieme degli stati del controllo finito

- Σ: insieme dei simboli di input
- Γ: insieme dei simboli del nastro; Σ è sottoinsieme di Γ
- δ: la funzione di transizione (q, X) -> (p, Y, Dir)
- q_0: stato iniziale
- B: simbolo di blank
- F: insieme degli stati finali

Una macchina di Turing accetta una classe di linguaggi detti **ricorsivamente enumerabili**.

7.3 Estensioni delle MdT

E' possibile estendere le caratteristiche di una MdT.

7.3.1 MdT multinastro

Al posto che avere un solo nastro hanno più nastri dai quali possono leggere/scrivere contemporaneamente per capire l'azione da intraprendere.

Inizialmente tutti i nastri (diversi da quello principale che contiene l'input) sono vuoti (hanno quindi solo caratteri di blank).

Usare una MdT multinastro non allarga la classe di linguaggi accettabili dalle MdT. Infatti ogni MdT multinastro ha un equivalente MdT mononastro.

7.3.2 MdT non deterministiche

La funzione δ della macchina di Turing non ritorna una sola tripla ma un insieme di triple. Vengono quindi lanciate computazioni parallele.

Una MdT non deterministica ha un equivalente MdT deterministica (implementata per semplicità con una MdT multinastro).

7.4 Restrizioni sulle MdT

7.4.1 MdT con nastro semi-infinito

In questa MdT il nastro non è di lunghezza infinita da entrambe le parti ma è di lunghezza infinita solo verso destra.

Il trucco di questa MdT sta nello scrivere, nella stessa cella due tracce di dati diverse. Una per la cella X_n e una per la cella X_{-n}. In questo modo è possibile limitare lo spostamento a sinistra.

Viene inoltre applicata un' altra limitazione: non è mai possibile scrivere un *blank* sul nastro.

Una MdT semi-infinita accetta la stessa classe di linguaggi di una MdT infinita.

7.4.2 PDA Multistack

Un PDA multistack (cioè un PDA che ha più stack su cui fa push/pop in ogni transizione) permette di simulare una macchina di Turing: infatti un linguaggio accettatto da una MdT è anche accettato da una PDA a due stack.

In uno stack viene messo quello che è a sinistra della testina, in un altro quello che è a destra.

- In input al PDA passo wZ, gli stack sono vuoti
- wZ viene copiato nel primo stack, smettendo di copiare quando c'è Z.
- Viene pushato tutto dal primo al secondo stato, in questo modo avrò la prima lettera di w in cima al secondo stato.
- Il PDA inizia a simulare: la "testina" legge il simbolo in cima al secondo stack e agisce in base a quello. Se si sposta a DX lo sposterà sul primo stack, se si sposta a sx, lo riscriverà sul secondo, insieme ad un simbolo di Blank. E così prosegue l'esecuzione.

Capitolo 8

Indecidibilità

8.1 Problemi

- Decidibili: la macchina di Turing si ferma e mi dice se accetta o no.Questi linguaggi sono ricorsivi. Esiste quindi un algoritmo.
- Semi-decidibli: la macchian di Turing si ferma se accetta ma non è detto che si fermi in caso di non accettazione
- Indecidibili: la macchina di Turing non può rispondere.

8.2 Ricorsivamente Enumerabile

Un linguaggio è ricorsivamente enumerabile se $L = L(M)$ per una qualche macchina di Turing M, cioè se L è accettato da qualche macchina di Turing.

8.3 MdT come stringhe binarie

Ogni stringa binaria può essere vista come una macchina di Turing, se la stringa non è ben formata allora è una macchina di Turing che non fa nula.

8.3.1 Corrispondenza tra stringa binaria e numero

Se w è una stringa binaria allora $1w$ è un intero.

Ad esempio $1w$ è la $1w_{10} - esima$ stringa.

Esempi

- $e \to 1_b - esima$
- $0 \to 10_b - esima$
- $1 \to 11_b - esima$

Addirittura, con questo criterio, le stringhe sono ordinate prima per lunghezza e poi per ordine lessicografico.

8.3.2 Codici per le TM

Visto che possiamo (per ora supponiamo) rappresentare una TM come un numero binario e possiamo associare a questo numero binario un numero naturale, possiamo parlare di MdT $1i_{bin}esima$ quando stiamo parlando della macchina di Turing rappresentata dalla stringa binaria i.

Supponiamo la regola di transizione:

$$\delta(q_i, X_j) = (q_k, X_l, D_m)$$

Codifichiamo questa regola con la stringa binaria: 0^i 1 0^j 1 0^k 1 0^l 1 0^m Non ci possono essere occorrenze di due o più 1 consecutivi.

Il codice per l'intera Turing machine è fatto in questo modo:

$$C_1 11 C_2 11 ... C_{n-1} 11 C_n$$

Per la stessa MdT possono esistere più codici, in quanto l'ordine delle transizioni può essere scambiato a piacimento.

Una coppia (TM, w) può essere rappresentata come $TM111w$

8.4 Linguaggio diagonale

E' fatto da tutte le stringhe w_i tali che la MdT rappresentata da w_i non accetta in input w_i stesso.

$$\{w_i | w_1 \notin L(M_i)\}$$

8.4.1 Perchè si chiama diagonale?

Mettiamo in una tabella j sulle colonne e i sulle righe. Mettiamo 1 se la macchina di Turing M_i accetta la stringa w_j.

La riga i-esima diventa quindi il vettore caratteristico della MdT M_i, ci indica tutte le stringhe membro di $L(M_i)$.

La riga diagonale ci dice se M_i accetta w_i. Per costruire L_d, dobbiamo quindi fare il complemento della diagonale.

Il problema pero' è che questo linguaggio non può stare in alcuna riga, il complemento della diagonale infatti è anch'esso un vettore caratteristico ed è per forza diverso per una colonna rispetto ad ogni riga della tabella.

Non esiste quindi macchina di Turing che accetta L_d (più forte del dire che tale linguaggio è indecidibile).

8.4.2 Ld non è ricorsivamente enumerabile

Teorema: L_d non è un RE. Non esiste macchina di Turing che accetta L_d.

Supponiamo che L_d sia $L(M)$ di una MdT M. M sarebbe nella lista di macchine di Turing costruite sopra. Quindi per qualche i avremmo che $M = Mi$.

Chiediamoci se w_i sta in L_d.

- Se w_i sta in L_d quindi M_i accetta w_i. Ma per definizione di Ld, w_i non è in Ld, perchè L_d contiene i w_j che non sono accettati da M_j.
- Se w_i non sta in Ld, M_i non accetta w_i, Quindi per definizione di L_d, w_i si trova in L_d.

E' un assurdo!

8.5 Linguaggi ricorsivi

Chiamiamo L un linguaggio ricorsivo se $L = L(M)$ per qualche macchina di Turing M tale che:

- Se w sta in L, quindi M accetta
- Se w non sta in L, quindi M si ferma, prima o poi.

Ricorsivi $\subset$ RE $\subset$ Non-RE

8.6 Complementi di linguaggi

Se L è ricorsivo anche $\not L$ è ricorsivo.

Se L e il suo complemento sono RE, L è ricorsio (e quindi anche $\not L$)

8.7 Linguaggio universale

%L_u$ è il linguaggio universale. L'insieme delle stringhe binarie che rappresentano una coppia (M, w) dove M è una TM e w una stringa di $(0+1)*$ in modo tale che w sta in $L(M)$.

L_u è l'insieme delle stringhe rappresentati una TM e un input accettato dalla TM.

C'è annche una TMU, la macchina di Turing universale tale che $L_u = L(TMU)$.

Per descrivere TMU si puo' usare una macchina multinastro a 4 nastri.

- Nel primo nastro si tengono le transizioni di M, seguite dalla stringa w nel formato $M111w$.
- Il secondo nastro viene usato per tenere il nastro simulato di M, usando lo stesso formato di M: X_i viene rappresentato come 0^i e i vari X sono separati da un 1.
- Il terzo nastro sarà usato per tenere lo stato di M, con q_i rappresentato da 0^i
- Il quarto nastro viene usato per calcoli.

8.7.1 Indecidibilità di L_u

Teorema 8.7.1 *Lu è RE ma non ricorsivo.*

E' facile vedere che L_u è RE: visto che la MTU deve simulare una MdT, la simulazione potrebbe non fermarsi mai, perchè la MdT che viene simulata non termina la sua esecuzione.

Più complesso è dire invece che RE è non ricorsivo.

Si suppone, per assurdo, che L_u sia ricorsivo. Quindi $\not L_u$, il suo complemento, è ricorsivo (per un teorema enunciato precedentemente). Ma, se avessimo una MdT che potesse accettare $\not L_u$ ne potremmo costruire una che acceta L_d. Ma visto che sappiamo che L_d è non RE, abbiamo una contraddizione, L_u non può essere ricorsivo.

Seguendo l'assurdo, come si può fare una MdT che accetta L_d?

Supponiamo l'esistenza di una MdT M che accetta $\not L_u$. Modifichiamo M in modo da accettare L_d, e costruiamo quindi M'.

- Dato in input w, M' lo cambia in $w111w$
- M' simula M sul nuovo input. Se w è w_i, M' determina se M_i accetta w_i. Quindi M accetta $\not{L}_u$ se e solo se M_i non accetta w_i, cioè se w_i non è in L_d.
- Quindi M' accetta w se e solo se w è in L_d. Visto che sappiamo che M' non può esistere, allora L_u è non ricorsivo.

8.8 Riduzione

Il concetto di riduzione: se abbiamo un algoritmo che converte istanze di un problema P1 in istanze di P2 che hanno la stessa risposta, diciamo che P1 riduce a P2.

P2 è quindi difficile tanto quanto P1. Quindi se P1 non è ricorsivo, neanche P2 può esserlo. Se p1 è non RE, p2 non può essere RE.

8.9 Esempio

Prendiamo due linguaggi: Le e Lne. Se w è una stringa binaria quindi rappresenta una macchina di Turing Mi. Se il linguaggio L(Mi)=0 w sta in Le. Le è il linguaggio di tutte le TM che non accettano input. Lne è il complemento.

Lne è il linguaggio "più semplice". E' RE ma non ricorsivo. Le non è RE.

Lne è RE Lne è non ricorsivo. Le è non RE

8.10 Proprietà

Una proprieàt di un linguaggio RE è un insieme di linguaggi RE. La proprietà di essere CF è l'insieme di tutti i linguaggi CF.

Banale: vuota o tutti i linguaggi RE.

Teorema di rice: Ogni proprietà non banale del linguaggio RE è indecidibile.

Capitolo 9

Altra Teoria

9.1 Con quale criterio riempiamo la tabella del linguaggio diagonale?

La tabella, di dimensioni infinite è costruita mettendo sulle colonne numeri 1..n che indichiamo con j, mentre sulle righe numeri 1..m che indichiamo con i.

Riempiamo la tabella mettendo 1 se la cella indicata dalla MdT Mi (quindi rappresentata dal valore binario della cella i-esima) accetta in input la stringa Wj

---

9.2 Come si definisce LD? E' RE?

LD è il linguaggio diagonale. LD è l'insieme delle stringhe Wi tali che Wi non si trova in L(Mi). Cioè l'insieme delle stringhe Wi non accettate dalla macchian di Touring Mi. Meglio ancora è l'insieme delle stringhe W tali che la MdT M (il cui codice è w) non accetta w come input.

LD non è RE. Non esiste MdT che lo accetta.

Dimostazione: Supponiamo che LD sia accettato da una MdT M. Visto che LD è costruito su {0,1}, M sta nella lista di MdT che abbiamo costruito (con la tabella). Quindi c'è almeno un codice alla riga i per cui M=Mi. Ora chiediamo se Wi è in LD.

- Se wi fosse in LD, Mi accetterebbe Wi. Però, per definizione di LD, Wi non è in LD, perchè LD contiene solo i Wj tali che Mj non accetta Wj.
- Se Wi non fosse in LD, Mi non accetterebbe Wi, quindi per definizione di LD, Wi è in LD.

Visto che Wi non può essere e non essere in LD allo stesso tempo, concludiamo che M non può esistere, LD non è ricorsivamente enumerabile.

9.3 Linguaggi Ricorsivi e Ricorsivamente Enumerabili

(Uso la definizione data in classe, non quella del libro)

I linguaggi Ricorsivi sono tutti quelli per cui esiste una MdT M tale che L=L(M). La macchian di Touring dice sempre se una stringa appartiene o no a quel linguaggio. Esiste quindi un algoritmo "deterministico". Esempi: tutti quelli dipendenti dal contesto (aˆn bˆn cˆn)

Per i linguaggi ricorsivamente enumerabili esiste una MdT M tale che M si ferma sempre se una stringa w appartine al linguaggio, mentre non è detto che si ferma se non appartiene. Esempi: Halting Problem, All regular, context-free, context-sensitive and recursive languages.

I linguaggio NON-RE sono quelli per cui non esiste una MdT.

9.4 Si consideri il linguaggio universale Lu

1) Si dia la sua definizione

Il linguaggio universale Lu è il linguaggio formato dalle stringhe binarie Mi111wj dove Mi è la codifica di una macchina di touring e wj è una stringa accettata dalla macchina di touring. Oppure ancora è il linguaggio formato dall'insieme di coppie (M,w) dove M è una macchina di touring con input binario e w è una stringa binaria tale che W è in L(M).

2) A quale classi di linguaggi appartiene? Come si dimostra tale appartenenza?

Lu è RE ma non ricorsivo. Lu è indecidibile. Questo perchè riducendo Lu ad un problema P e si puo' dimostrare che non c'è algoritmo per risolvere P.

Teorema: Lu è RE ma non ricorsivo. "Sappiamo da sopra che LU è RE" (perchè possiamo costruire la TM, usando una TM multinastro) Supponiamo che Lu sia ricorsivo. Anche -Lu (il suo complemento) è ricorsivo. Quindi se abbiamo una MdT

M che accetta -Lu possiamo anche costruire una macchina di Touring che accetta Ld. Visto che sappiamo che Ld è non-RE (e quindi non esiste MdT per Ld) abbiamo una contraddizione.

3) In che modo è in relazione con il problema della terminazione dei programmi? Spesso si sente dire che l'halting problem è simile a Lu. Infatti la macchina originale di Alan Turing accettava per "halting" e non per "stato finale". Possiamo quindi definire H(M) per la MdT M l'insieme di input w tali che M si ferma per w, indipendentemente se M accetta w o no. Quindi l'halting problem diventa un insieme di coppie (M,w) tali che w sta in H(M). Come si voleva dimostrare.

9.5 Perchè si è sicuri che esista un linguaggio non RE? Dare un esempio di tale linguaggio e dimostrare che non è RE.

Perchè si è sicuri?????? (BOH?)

Prendiamo due linguaggi: Le ed Lne. Entrambi consistono di stringhe binarie. Se w è una stringa binaria rappresentante una MdT, w appartine a Le se la MdT non accetta stringhe, altrimenti a Lne se MdT accetta stringhe. Lne è il linguaggio più semplice da dimostrae ed è RE ma non ricorsivo. Le è non RE.

Per prima cosa si dimostra che Lne è ricorsivamente enumerabile. Per fare questo basta mostrare una TM (non-deterministica) che accetta Lne.

- M prende in input una MdT codificata da Mi.
- Usando le sue capacità non-deterministiche, M indovina l'input di w per cui Mi potrebbe accettare
- M testa se Mi accetta w.
- Se Mi accetta w, alloara M accetta Mi.

Proviamo ora che Lne non è ricorsivo. Per fare questo riduciamo Lu a Lne. (Procedimento lunghissimo)

Detto ciò possiamo sapere lo stato di Le. Se Le fosse RE sia Lne che Le sarebero ricorsivi. Visto che Lne è non ricorsivo alloa concludiamo che Le è non RE.

Capitolo 10

Laboratorio

10.0.1 Input Analizzatore Lessicale

Sequenza di stringhe di simboli unicode che vengono trasformati in token

10.0.2 Input Analizzatore sintattico

Sequenza di token lessicalmente corretti

10.0.3 Dfferenza tra grammatica lessicale e grammatica sintattica

L'analisi lessicale raggruppa e controlla i simboli in token che verranno analizzati sintatticamente dal parser che controlla la loro composizione alla base della sintassi del linguaggio.

10.0.4 Compilatore

- Line Recontructor - prepara l'input per il lexer
- Lexer + Parser - riconosce Token e crea la struttura sintattica del codice
- Semantic Checker - crea il parse tree
- Code Generator - analisi ed ottimizzazione del parse tree

10.0.5 Parse Tree

E' l'output di un parser e rappresenta la struttura sintattica di un programma

10.0.6 Parse Stack

Contiene i vari token letti dall'inputi attraverso l'operazione di shift.

10.0.7 Bottom-up parser

Si parte dalla stringa prodotta e si costruisce il parse tree dalle fogli verso i vari padri fino alla root (start symbol).

10.0.8 Left-right parser

Legge la stringa da sinistra a destra E cerca nell'input elaborato le sottostringhe che corrispondono al corpo di una produzione partedo da destra e applica quella produzione al contrario.