Aniello Murano

Automi e Pushdown

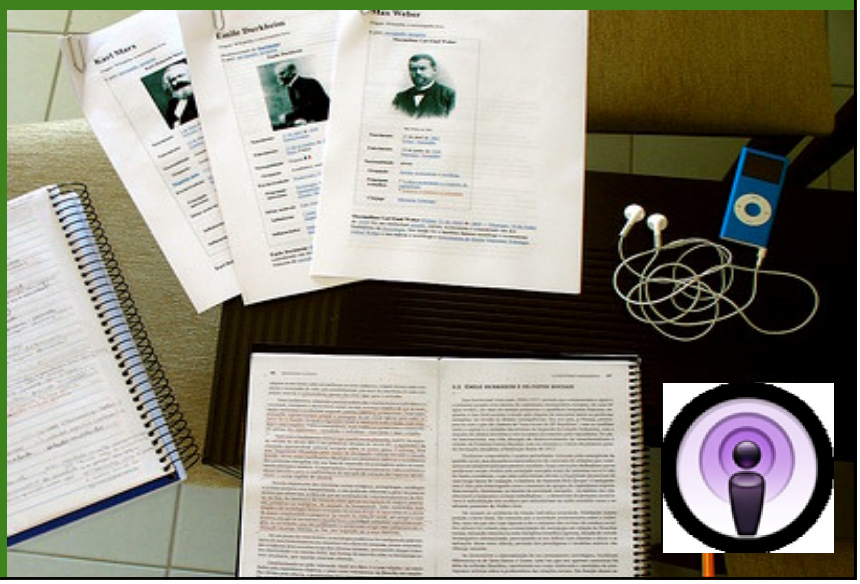
Lezione n.2
Parole chiave:
Automi e PDA

Corso di Laurea:
Informatica

Codice:

Email Docente:
murano@ na.infn.it

A.A. 2008-2009



Calcolabilità, complessità e macchine computazionali

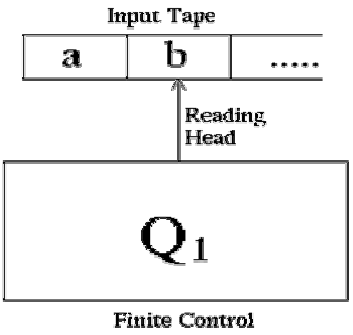
- Per presentare in modo formale i concetti basilari delle teorie della calcolabilità e della complessità, abbiamo innanzitutto bisogno di una definizione precisa di cosa è un computer.
- I computer di uso quotidiano, come un personal computer, sono troppo complessi per essere utilizzati come modelli
- Si preferisce invece utilizzare delle macchine computazionali "semplificate".
- Esistono diverse macchine che si possono usare, con diversa capacità computazionale
- Solitamente, all'aumentare della loro capacità computazionale, aumenta la difficoltà gestionale e valutativa di queste macchine.
- Tra le macchine computazionali più semplici, particolarmente adatte al nostro scopo, ci sono gli automi.
- Sebbene gli automi siano costituiti da una memoria finita, esistono svariati contesti reali in cui essi sono impegnati con successo.

Perchè studiare la teoria degli automi?

- La teoria degli automi e dei linguaggi formali è lo studio di dispositivi computazionali o "macchine" computazionali.
- Gli automi, originariamente, furono proposti per creare un modello matematico che riproducesse il funzionamento del cervello.
- La teoria degli automi è alla base della descrizione e della modellazione dei linguaggi di programmazione, della costruzione dei loro riconoscitori e traduttori, della realizzazione di strumenti di elaborazione testuale.
- Ecco alcuni esempi in cui gli automi finiti sono possono essere utilizzati come modelli per il software:
  - software per la progettazione e la verifica del comportamento dei circuiti digitali;
  - "analizzatore lessicale" di un compilatore;
  - software che eseguono una scansione di testi molto lunghi per trovare parole, frasi, ecc.;
  - software per verificare i protocolli di comunicazione o protocolli per lo scambio sicuro di informazioni.
- Riferimenti: Hopcroft, J.E., Ullman J.D., *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, Reading, Mass., 1979.

Descrizione

- Un automa è un dispositivo sequenziale creato per eseguire un particolare compito che, ad ogni istante, può trovarsi in un determinato "**stato**".
- Quando l'automa si trova in uno stato, può eseguire una transizione (in un altro stato) in base ad un input esterno, rappresentato da un simbolo del suo alfabeto
- Lo scopo dello stato è quello di ricordare la parte rilevante della **storia** del sistema. Dunque lo stato funge da **memoria.**
- Il modello di un automa consiste di un dispositivo di controllo, con un numero finito di stati, una testina di lettura e un nastro infinito diviso in celle.



Rappresentazione grafica di un automa

Descrizione(2)

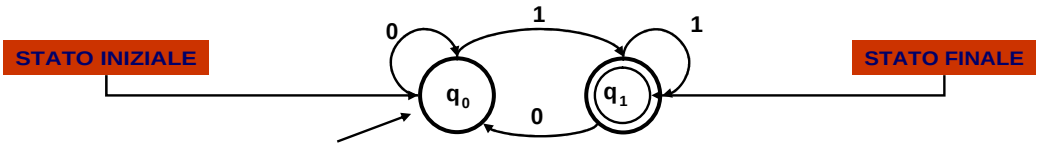
- Gli automi sono spesso utilizzati per descrivere linguaggi formali, e per questo sono chiamati accettori o riconoscitori di un linguaggio.
- L'evoluzione di un automa parte da un particolare stato detto **stato iniziale**.
- Un sottoinsieme privilegiato degli stati di un automa è detto insieme degli **stati finali** e corrispondono agli stati "accettanti".
- Una sequenza di simboli (detto anche stringa o parola) appartiene al linguaggio accettato da un automa se essa porta l'automa, in un certo numero di passi, dallo stato iniziale ad uno accettante.
- A diverse classi di automi corrispondono diverse classi di linguaggi, caratterizzate da diversi livelli di complessità.
- Analizziamo in dettaglio i vari tipi di automi

Definizione di un DFA

- Un "**deterministic finite automaton**" o "**DFA**" (automa a stati finiti determini-stico) è formalmente definito come una quintupla (Q,Σ,δ,q_0,F) dove
  - Q è un insieme finito di **stati**
  - Σ è un insieme finito di caratteri che costituiscono l'**alfabeto**
  - $\delta : Q \times \Sigma \rightarrow Q$ è la **funzione di transizione** che associa ad ogni coppia (stato, carattere) uno stato
  - $q_0 \in Q$ è lo **stato iniziale**
  - F sottoinsieme di Q è l'insieme degli **stati finali**

Run di accettazione di un DFA

- Un DFA A = (Q,Σ,δ,q_0,F) accetta una stringa $u = u_1\ u_2\ \dots\ u_n$ se (e solo se) c'è una sequenza di stati $r = r_1, r_2, \dots, r_n, r_{n+1}$ tale che:
  - $r_1 = q_0$
  - $r_{i+1} = \delta(r_i,u_i)$, per ogni i con $1 \le i \le n$
  - $r_{n+1} \in F$
- Il linguaggio **accettato** da un DFA è l'insieme di tutte le **parole** (stringhe) formate con i simboli di Σ per il quale l'automa partendo dallo stato iniziale raggiunge lo stato finale alla lettura dell'ultimo simbolo della parola.
- Un linguaggio è **regolare** se esiste un DFA che accetta tutte e sole le parole che esso contiene.
- **Esempio:** In figura è mostrato l'automa che accetta il linguaggio di tutte le parole composte di 0 e 1 e che terminano con 1.

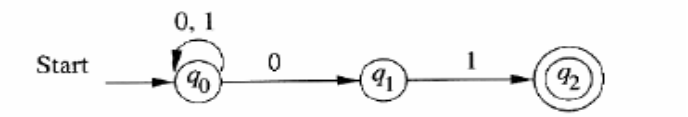


Esercizio

- Scrivere un DFA che accetti il linguaggio di tutte le parole composte di 0 e 1 e che contengono un numero pari di 1.
- Scrivere un DFA che accetti il linguaggio di tutte le parole composte di 0 e 1, il cui numero binario corrispondente è pari.
- Scrivere un DFA che accetti il linguaggio di tutte le parole composte di 0 e 1 e che terminano con "01".

Definizione di un NFA

- Un "**nondeterministic finite automaton**" o "**NFA**" (automa a stati finiti non-deterministico) differisce da un DFA per la definizione della funzione di transizione.
- Un NFA, stando in uno stato e leggendo un simbolo può andare in più stati.
- Il funzionamento di un NFA si può immaginare come se esistessero più copie dello stesso automa.
- Formalmente, un NFA è definito come per i DFA da una quintupla (Q,Σ,δ,q_0,F), con l'unica differenza che la δ è così definita:
  - $\delta : Q \times \Sigma \rightarrow 2^Q$ (con 2^Q denotiamo l'insieme di tutti i sottoinsiemi di Q)
- Nota: data una parola su Σ, un NFA può avere una, nessuna o più sequenze di stati associati dalla funzione di transizione.
- Una parola è accettata da un NFA se almeno una delle sequenze di stati associate dalla δ terminano in uno stato finale.
- **Esempio:** In figura è mostrato l'automa che accetta il linguaggio di tutte le parole composte di 0 e 1 e che terminano con "01".



Confronto tra run di un DFA e di un NFA

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

Quando A è un DFA	Quando A è un NFA
A ***accetta*** la stringa $u_1\ u_2\ \ldots\ u_n$ se c'è una sequenza di stati $r_1, r_2, \ldots, r_n, r_{n+1}$ tale che:	A ***accetta*** la stringa $u_1\ u_2\ \ldots\ u_n$ se c'è una sequenza di stati $r_1, r_2, \ldots, r_n, r_{n+1}$ tale che:
• $r_1 = s$	• $r_1 = s$
• $r_{i+1} = \delta(r_i,u_i)$, per ogni ***i*** con $\mathbf{1 \le i \le n}$	• $r_{i+1} \in \delta(r_i,u_i)$, per ogni ***i*** con $\mathbf{1 \le i \le n}$
• $r_{n+1} \in F$	• $r_{n+1} \in F$

Equivalenza tra DFA e NFA

- Due automi A e B sono detti equivalenti se accettano lo stesso linguaggio, ovvero L(A)=L(B)
- **Teorema: Per ogni NFA N esiste un DFA D equivalente.**
- Per provare questo teorema si usa la **subset construction:**
  - gli stati di D rappresentano tutti i possibili sottoinsiemi degli stati raggiungibili in N, sulla stessa parola.

- Sia N = (Q, Σ, δ, s, F) un NFA, la subset construction permette di costruire un DFA D = (Q′, Σ, δ′, s′, F′) nel modo seguente:
  - Q′ = P (Q)
  - $\delta'(R,a) = \{q \mid q=\delta(p,a) \text{ per ogni } p \in R\}$
  - s′ = {s}
  - F′= {R | R è un sottoinsieme di Q che contiene uno stato finale di N}

- **Osservazione:** La subset constuction è una traduzione esponenziale!

Esercizi

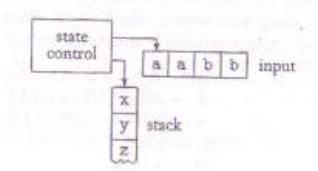
- Scrivere un NFA e un DFA per il linguaggio accettante tutte le parole terminanti per "001"
- Scrivere un NFA e un DFA per il linguaggio accettante tutte le parole che hanno un "1" nella terzultima posizione.
- Scrivere un DFA per il linguaggio D = {w | w contiene lo stesso numero di "01"e "10"}. Per esempio 010 è in D

Grammatiche, linguaggi regolari e automi

- Come si è detto precedentemente, gli NFA accettano (tutti e solo) i linguaggi regolari.
- Questa classe gode della proprietà di essere **chiusa** rispetto alle operazioni di unione, intersezione e complemento [Prova lasciata per esercizio agli studenti]
- Ogni linguaggio appartenente a questa classe può essere costruito tramite una **grammatica regolare**
- Ogni linguaggio appartenente a questa classe può essere univocamente determinato tramite una **espressione regolare**.
- Gli ultimi due concetti, sebbene siano marginali per questo corso, sono fondamentali per uno studio completo sulla teoria degli automi.
- Informazioni più dettagliate possono essere reperite dal libro di testo consigliato: Hopcroft, J.E., Ullman J.D., *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, Reading, Mass., 1979.

Oltre gli automi a stati finiti

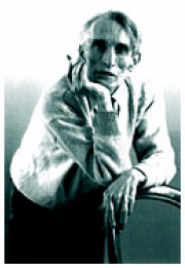
- Si consideri il linguaggio $L = \{a^n b^n \text{ con } n>1\}$
- Domanda: Questo linguaggio può essere accettato da un automa a stati finiti?
- Risposta: No! Gli automi a stati finiti posseggono una memoria finita e quindi non sono in grado di riconoscere linguaggi che, per la loro struttura, necessitano di ricordare una quantità di "informazioni" non limitate.
- Il linguaggio in oggetto può essere accettato da un **Pushdown Automaton** o **PDA**
- Un PDA è un automa a stati finiti con stack. Lo stack è un contenitore (anche infinito) che rispetta un accesso LIFO ed ha capienza infinita.
- Ad ogni transizione di un PDA è dunque possibile leggere il contenuto di una cella del nastro ed il simbolo in cima allo stack, passare in un nuovo stato e sostituire il simbolo letto sul top dello stack con una stringa.



Schema di un automa pushdown

Intermezzo

- I PDA sono stati introdotti da Oettinger nel 1961 e da Schutzenberger nel 1963
- Schutzenberger (1920-1996) è stato un ricercatore e un punto di riferimento per varie generazioni di ricercatori per i linguaggi formali e l'informatica teorica. Ha insegnato alla Facoltà di Scienze dell'Università di Parigi (VI e VII).



Definizione di PDA

- Un PDA nondeterministico è formalmente definito da una 6-pla $(Q,\Sigma,\Gamma,\delta,q_0,F)$ dove
  - Q è un insieme finito di **stati**
  - Σ è un insieme finito di caratteri che costituiscono l'**alfabeto del nastro**
  - Γ è un insieme finito di caratteri che costituiscono l'**alfabeto dello stack**
  - $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma^*)$ è la **funzione di transizione** che associa ad ogni tripla (stato, carattere letto sul nastro, carattere letto sul top dello stack) un insieme di possibili coppie (stato, stringa), dove la stringa sostituisce il top dello stack
  - $q_0 \in Q$ è lo **stato iniziale**
  - F sottoinsieme di Q è l'insieme degli **stati finali**

Passi di Calcolo

- Se $(q,B) \in \delta(p,a,A)$, con $a \in \Sigma_\varepsilon$, $A \in \Gamma_\varepsilon$ e $B \in \Gamma_\varepsilon$ allora è possibile che sia eseguito uno dei seguenti passi di calcolo,
  - (POP-PUSH) se $a \in \Sigma$, $A, B \in \Gamma$ con a in lettura sul nastro di input e A in cima alla pila passa nello stato q, rimpiazza A con B in cima alla pila e muove la testina di lettura del nastro di input di una cella verso destra.
  - (PUSH) se $a \in \Sigma$, $A = \varepsilon$, $B \in \Gamma$ allora, con a in lettura sul nastro di input e indipendentemente dal simbolo in cima alla pila, passa nello stato q, impila B in cima alla pila e muove la testina di lettura del nastro di input di una cella verso destra.
  - (POP) se $a \in \Sigma$, $A \in \Gamma$, $B = \varepsilon$ allora, con a in lettura sul nastro di input e A in cima alla pila, passa nello stato q, elimina A dalla cima della pila e muove la testina di lettura del nastro di input di una cella verso destra.
- Se $a = \varepsilon$, i tre tipi di mosse avvengono senza che la testina di lettura si sposti.

Configurazione

- Come nel caso degli NFA, il concetto di configurazione è utile per descrivere la situazione in cui si trova la macchina complessivamente:
  - lo stato,
  - l'input ancora da esaminare e
  - il contenuto della pila.
- Quindi una configurazione è un elemento di $Q \times \Sigma^* \times \Gamma^*$.
- La configurazione iniziale è (q_0,x,ε)

Accettazione

- Un PDA M = (Q,Σ,Γ,δ,q_0,F) accetta una stringa w= w_1w_2 ... w_n, con w_i ϵ Σ_ε, se esistono una sequenza di stati, r_0, r_1,....,r_n di Q e una stringa s_0, s_1, . . . , s_m di Γ* che soddisfino le seguenti tre condizioni:
  1. $r_0 = q_0$ e $s_0 = \varepsilon$.
  2. Per i = 0, . . . ,m - 1, si ha che (r_{i+1} , b) ϵ δ(r_i , w_{i+1} , a), dove s_i = a·t e s_{i+1} = b·t, per qualche a,b in Γ_ε e t in Γ*.
  3. r_n appartiene a F.

Esempio

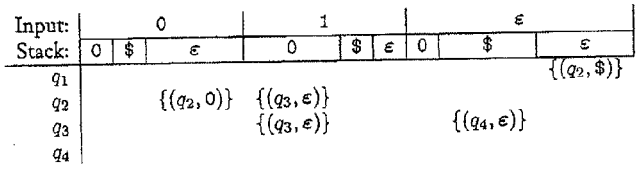
- Un esempio di PDA che accetta il linguaggio L={0^n1^n|n≥0} è il seguente:

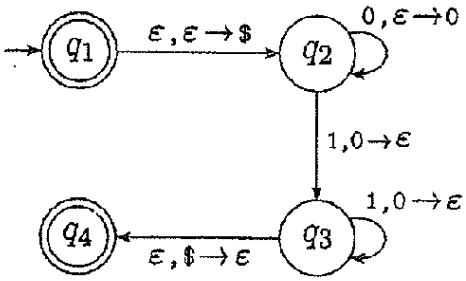
$Q = \{q_1, q_2, q_3, q_4\}$,

$\Sigma = \{0,1\}$,

$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, and





Proprietà dei PDA

- I PDA sono schiusi rispetto all'unione, ma **non** rispetto al complemento e all'intersezione.
- I PDA sono invece chiusi rispetto all'intersezione e al complemento con linguaggi regolari, cioè l'intersezione (o il complemento) di un linguaggio accettato da un PDA con un linguaggio regolare è sempre accettato da un PDA.
- Anche per i PDA esiste la possibilità di usare delle grammatiche in grado di generare tutti i soli i linguaggi da essi accettati: le **grammatiche context-free**

Esercizi

- Definire un PDA che accetti il linguaggio $\{a^n b^n$, con $n>1\}$
- Definire un PDA che accetti il linguaggio: $\{a^i b^j c^k \mid i,j,k > 0$ e $i = j$ oppure $i = k\}$.
- Definire un PDA che accetti il linguaggio: $\{ww^R \mid w \in \{0,1\}^*\}$. "R sta per reverse" (linguaggio delle palindrome)

PDA deterministici

- Un PDA $A = (Q,\Sigma,\Gamma,\delta,q_0,F)$ è deterministico (DPDA) se
  - per ogni $q \in Q$, $a \in \Sigma_\varepsilon$ e $B \in \Gamma$, abbiamo che $card(\delta(q,a,B)) \leq 1$,
  - per ogni $q \in Q$ e $B \in \Gamma$, se $\delta(q,\varepsilon,B) \neq \emptyset$, allora $\delta(q,a,B) = \emptyset$ per ogni $a \in \Sigma$
- Dunque, se c'è una ε-mossa eseguibile in un certo stato e con un certo simbolo in cima alla pila allora quella è l'unica mossa eseguibile in quello stato e con quel simbolo in cima alla pila.
- A differenza del caso dei DFA, esistono linguaggi accettati da un PDA che non possono essere accettati da un DPDA (per esempio il linguaggio delle palindrome)
- Quindi L(PDA), la classe dei linguaggi accettati dai PDA, contiene strettamente L(DPDA), la classe dei linguaggi accettati dai PDA deterministici,

Oltre i PDA

- Per quanto un PDA possa essere più potente in un NFA, esistono linguaggi che non possono essere riconosciuti da un PDA. Per esempio, il linguaggio $\{a^n b^n c^n$, con $n>1\}$ necessita di una **macchina di Turing** per essere accettato.
- Le macchine di Turing saranno oggetto delle prossime lezioni.