16/11

lang	sì	no
REG	regex FA	pumping lemma
CFL	CFG PDA	pumping lemma

per es. se linguaggi

Data CFG, possiamo eliminare simboli e prod. mantenendo linguaggio

- eliminare $A \to \varepsilon$
- eliminare $A \to B$

} difficile interpretazione, ma efficiente per computer

- simboli inutili:

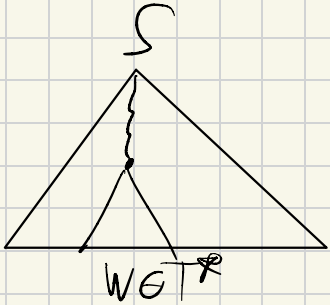
  dato $X \in V \cup T$:

  REACHABLE se $\exists S \Rightarrow^* \alpha X \beta$ / $\alpha, \beta \in (V \cup T)^*$

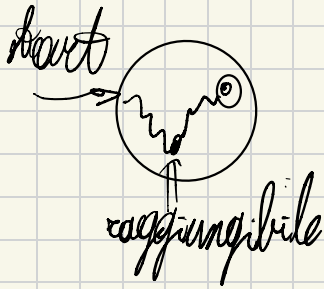
  GENERATING: $\exists X \Rightarrow^* w$, $w \in T^*$

  se entrambi USEFUL, else USELESS

es: tree



(simile a FA)



Es: $S \to AB \mid a$ → non generating

$A \to b$

⇓ eliminiamo $S \to AB$

$S \to a$

$A \to b$ → non reachable

eliminiamo ⟹ $S \to A$

Algoritmi

dati alg. per trovare gen. e reac.

dato G con $L(G) \neq \emptyset$:

- costruiamo G_1 togliendo non-generating da G e prod. in cui appaiono

- costruiamo G_2 eliminando da G_1 non-reachable e prod. con essi

Come trovare generatore $g(G)$

- base: $g(G) \leftarrow T$ (prima metto tutti terminali)
- induzione: se $(A \to X_1 \dots X_n) \in P$ e $X_i \in g(G)\ \forall i \in [1,n]$

$$g(G) \leftarrow g(G) \cup \{A\}$$

(da dx a sx: bottom-up)

Trovare reachable $r(G)$

- base: $r(G) \leftarrow \{S\}$ (simbolo iniziale)
- ind: se $(A \to X_1 \dots X_n) \in P$ e $A \in r(G)\ \forall i \in [1,n]$,

$$r(G) \leftarrow r(G) \cup \{X_i\}$$

NO dim. corretteza all'esame

15/11

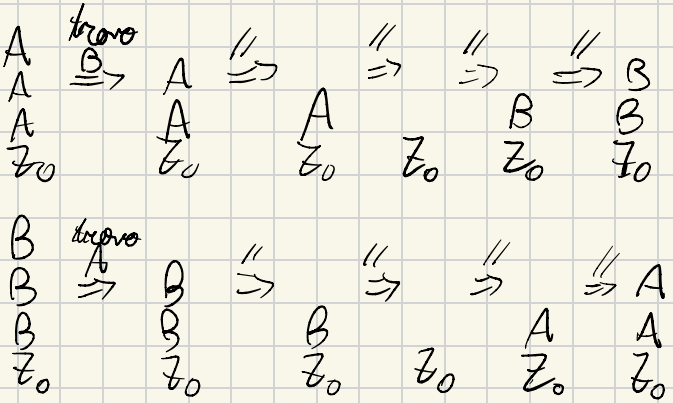
Es1:

$\Sigma = \{a, b\}$; $L = \{w \mid w \in \Sigma^*, \#_a(w) \neq \#_b(w)\}$

è CFL? ⇒ lo è ⇒ + facile creare PDA (final state)

$\Gamma = \{Z_0, A, B\}$

significato: trovato a/b



$Q = \{q_0, q_f\}$

δ (informale):

input: q_0, Z_0 or A, a ⇒ push A, sto in q_0

// : q_0, A, b ⇒ pop A, sto in q_0

q_0, Z_0 or B, b ⇒ push B, sto in q_0

q_0, B, a ⇒ pop B, sto in q_0

q_0, A or $B, \varepsilon \Rightarrow$ posto in q_f
(accettato per stato finale)

$q_0, Z_0 \Rightarrow$ niente

(Es:
$\Gamma = \{Z_0, X\}$, usare + stati, trovare stesso linguaggio)

Come eliminare ε-prod. $(A \to \varepsilon)$?
Se $\varepsilon \in L$ non possiamo mantenere linguaggio $\Rightarrow$
$\Rightarrow$ nuova CFG genererà $L \setminus \{\varepsilon\}$
Variabile NULLABLE se $A \overset{*}{\Rightarrow} \varepsilon$
Idea: se A nullable e c'è prod. $B \to CAD$,
divido regole:
- tolgo prod. con ε a dx
- costruisco 2 prod. alternative: $B \to CD \mid CAD$
$(A \to \varepsilon, A \not\to \varepsilon)$

se anche C, D nullable, tolgo ogni combinazione di A, C, D da prod. originale
(non produrre $B \to \varepsilon$)

Alg.: dato $G=(V,T,P,S)$, $n(G)$: insieme di var. nullable $\Rightarrow$
$\Rightarrow$ lo calcoliamo con ind.:

- base: $n(G) \leftarrow \{A \mid (A \to \varepsilon) \in P\}$ (tolgo ε-prod.)
- ind: $\exists (A \to B_1 B_2 \dots B_k) \in P$, $B_i \in n(G)\ \forall i \in [1,k]$,
  $n(G) \leftarrow n(G) \cup \{A\}$

Possiamo costruire $G_1=(V,T,P_1,S)$ da $G \Rightarrow P_1$ creato come:

- $(A \to \varepsilon)$ tolte da P_1
- data $p: (A \to \gamma_1 \dots \gamma_k) \in P$ $k \geq 1 \Rightarrow N=\{i_1,\dots,i_m\}$: insieme di INDICI di var. nullable
- $\forall N' \subseteq N$ (ogni subset), aggiungo a P_1 prod. ottenuta da p cancellando ogni X_i, $i \in N'$
- ecc.: se $m=k$, NON aggiungere $(A \to \varepsilon)$

Prod. UNARIE: $A \to B$, $A,B \in V$
(possiamo espandere rhs)

UNARY PAIR: $(A,B) \mid A \overset{*}{\Rightarrow} B$ con solo prod. unarie

Calcoliamo insieme di unary pairs $u(G)$

- base: $u(G) \leftarrow \{(A,A) \mid A \in V\}$
- ind.: se $(A,B) \in u(G)$ e $(B \to C) \in P$,

  $u(G) \leftarrow u(G) \cup \{(A,C)\}$

Costruiamo nuova CFG $G_i = (V, T, P_1, S) \Rightarrow P_i$ costruito così:

- calcolo $u(G)$
- $\forall (A,B) \in u(G)$ e $\forall (B \to \alpha) \in P$ che non sono unarie, aggiungo $(A \to \alpha)$ a P_1

Semplificazione CFG: conta ordine di elimin.:

- ε-prod.
- unary prod.
- simboli inutili

CFG in CHOMSKY NORMAL FORM (CNF) le prod. sono solo in 2 forme:

- $A \to BC$ con $A, B, C \in V$
- $A \to a$ con $A \in V$, $a \in T$

e no simboli inutili

Ogni CFL senza ε può essere preso con CNF grammar

Alg: dopo rimozioni, abbiamo:

- $A \to a$
- $A \to \alpha \mid \alpha \in (V \cup T)^*$, $|\alpha| \geq 2$

necessario che rhs con lung > 2 abbiano solo var. e siano composte in catene di prod. con solo 2 var. in rhs

$\forall$ prod. con rhs $\alpha \mid |\alpha| \geq 2$ e $\forall$ occorrenza in α di $a \in T$

- costruisco $A \to a$ (A nuova var.)
- uso A anziché a in α

GREIBACH NORMAL FORM (GNF):

ogni prod. come $A \to ad$, $a \in T$, $d \in V^*$

Proprietà:

- ogni CFL non vuota con stringhe $\neq \varepsilon$ ha solo 1 GNF grammar
- grammar in GNF genera stringhe di lunghezza n in n passi

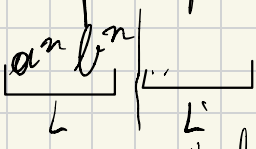
17/11

ES: $L = \{a^n b^n \mid n \geq 1\} \Rightarrow$ CFL (no REG)

$L' = \{a, b\}^+$ (NO regex) $\Rightarrow$ REG

$L \cdot L'$ REG?

come fare prima valutazione?



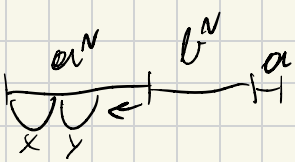
$\Rightarrow$ non posso contare n con FA

no condizione su questa

pumping lemma:

N costante

stringa

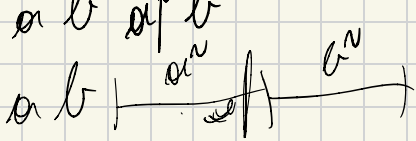


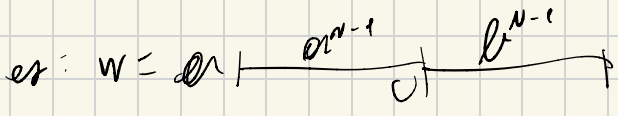
$y = a^m, m \geq 1, |xy| \leq N$

con $k \geq 2$, $a^{N+(k-1)m} b^N a \Rightarrow$ non in $L \cdot L'$

$L' \cdot L$ REG?

a b $a^N b^N$ $N \geq 2$





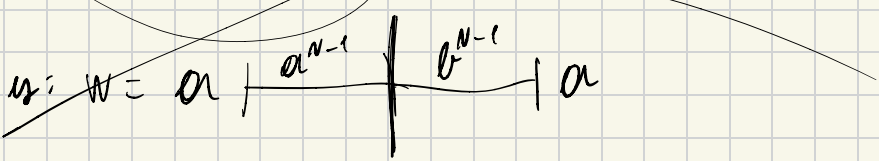
$K = 0 \Rightarrow a^{N-m} b^N$

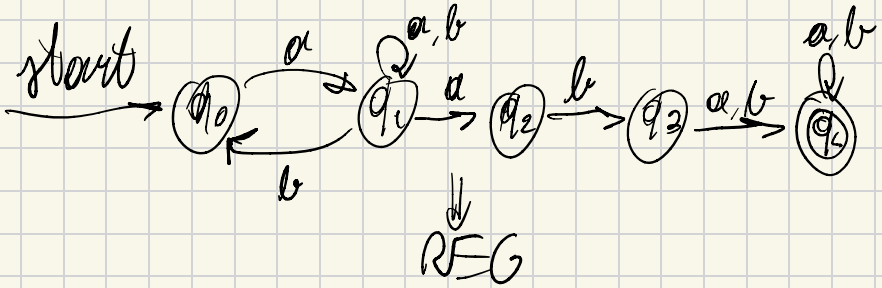
$L \cdot L \cdot L$ reg? $\Rightarrow$ reg

$(L \cdot L)\ L$

reg

A, B reg $\Rightarrow A \cdot B$ reg





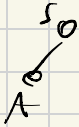
In ogni stringa abbastanza lunga di CFL possiamo trovare sottostringhe vicine rimovibili o iterabili producendo stringhe in linguaggio

Teorema sui parse trees:

G CNF grammar, T parse tree per $w \in L(G)$; se percorso + lungo in T ha n archi, allora $|w| \leq 2^{n-1}$

(CNF $\Rightarrow$ ogni nodo ha max. 2 figli)

Dim: induzione su n

- base: $n=1 \Rightarrow$ $S \Rightarrow a \Rightarrow |w| = 1 \leq 2^0 = 1$
- induzione: $n > 1$  $S \Rightarrow AB \overset{*}{\Rightarrow} w = uv$, $A \overset{*}{\Rightarrow} u$, $B \overset{*}{\Rightarrow} v$

nessun percorso in sottoalbero può essere + lungo di $n-1$

per ind., $|u| \leq 2^{n-2}$, $|v| \leq 2^{n-2}$

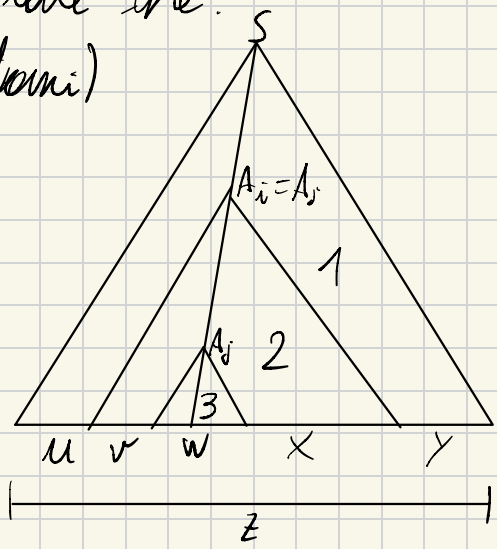
$|w| = |u| + |v| \leq 2^{n-2} \cdot 2 = 2^{n-1}$

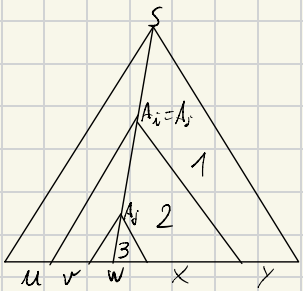
PUMPING LEMMA

$L \subseteq L$; $\exists n$ | se $z \in L$, $|z| \geq n$, possiamo fattorizzare $z = uvwxy$ tale che:

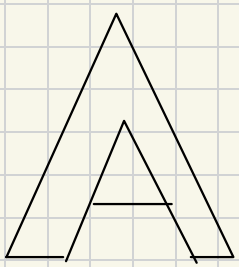
- $|vwx| \leq n$ (non molto lontani)
- $vx \neq \varepsilon$
- $uv^i wx^i y \in L \quad \forall i \geq 0$

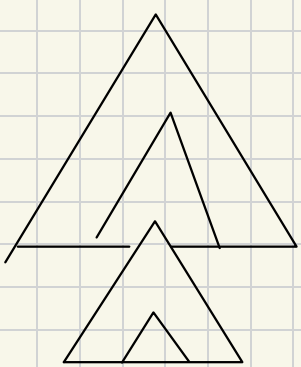
piccionaia su percorso + lungo $\Rightarrow$ deriverà var. che appare 2 volte





$\Rightarrow$





posso fare varie parse tree

Dim: G in CNF $\Rightarrow L(G) = L \setminus \{\varepsilon\}$

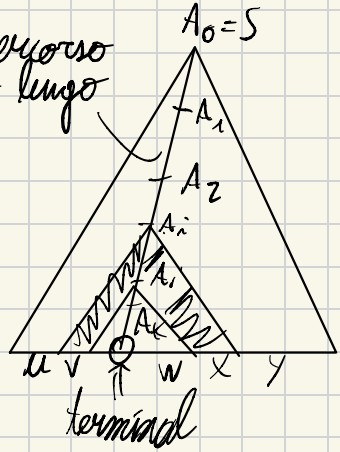
definiamo $m := |V|$, $n := 2^m$

sia $z \in L \mid |z| \geq n \Rightarrow$ usando teorema di prima, se parse tree in G per z ha percorso + lungo di lunghezza $\leq m$, allora $|z| \leq 2^{m-1} = n/2 \Rightarrow$

$\Rightarrow$ contraddice $|z| \geq n \Rightarrow$ deve avere percorso + lungo di lunghezza $> m$

sia $k+1$ lunghezza di percorso + lungo per $z \Rightarrow$

$\Rightarrow$ allora $k+1 > m$



abbiamo $k+1$ archi $\Rightarrow$

$\Rightarrow k+1$ variabili e 1 terminale

dato che $k+1 > m$, devono esistere $i < j \mid A_i = A_j$

(piccionaia)

$z = uvwxy \Rightarrow z_k = uv^k wx^k y \in L \; \forall k \geq 0$

se abbiamo + var. uguali, scegliamo coppia + in fondo $\Rightarrow$ in percorso da 1ª var. a fondo, alla 2ª vale. è uguale $\Rightarrow$ # archi $\geq m+1$

in albero + grande, percorso + lung $\leq m+1 \Rightarrow$

$\Rightarrow$ per teo. prima, allora abbiamo

$|vwx| \leq 2^{m+1-1} = 2^m = n$

$C \nsubseteq N \Rightarrow$ no ε-prod. $\Rightarrow$ deve essere $vx \neq \varepsilon$ (può essere che o v o x siano vuoti)

21/11

Es1: $L = \{0^i 1^i 2^i \mid i \geq 1\}$

$z = 0^n 1^n 2^n$

A fatt. vwx non può avere 0 e 2 $\Rightarrow$ quei pezzi sono distanti $n+1$ simboli:

- vwx non ha 2 $\Rightarrow uwy$ avrà 2^n e $0^i, 1^j$ con $i, j \leq n$
- vwy non ha 1 $\Rightarrow$ simile

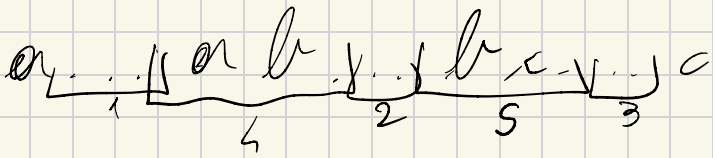
Per pumping, CFL non può avere parola incrociata (tipo $0^i 1^j 2^i$) $\Rightarrow$ se PDA volesse controllare, problemi con blocchi in stack

CFL non può copiare stringhe

Es2: $L = \{a^i b^j c^k \mid i, j \geq 0,\ k = \max\{i, j\}\}$

usiamo $z = a^n b^n c^n$

Vari casi per posizione vwx:



sottocasi:

a - v₀ x hanno lettere diverse

b - v₁ x hanno una lettera distinta ciascuno

caso 1: $k \geq 2 \Rightarrow \#_a$ aumenta e blocco c dovrebbe adattarsi

caso 2: come caso 1

caso 3: k qualunque $\Rightarrow \#_c \neq \max\{\#_a, \#_b\}$

casi 4/5:

- v o x hanno sia a che b: struttura violata
- // // sia b che c: // //
- 4b: $k = 2 \Rightarrow \#_a$ e $\#_b$ aumentano, $\#_c$ costante
- 5b:
  - $\square \times \neq \varepsilon \Rightarrow k = 0 \Rightarrow \#_c$ e $\#_b$ vendono ma $\#_a$ resta uguale e dà massimo
  - $\square \times = \varepsilon \Rightarrow k > 1 \Rightarrow \#_b$ aumenta e $> \#_a, \#_c$ costante

$s: \Sigma \to 2^{\Delta^*}$ (simbolo → linguaggio), Σ, Δ^* alfabeti

data $w = \underbrace{a_1 \dots a_n}_{\Sigma} \in \Sigma^*$: $s(w) = s(a_1) \dots s(a_n)$

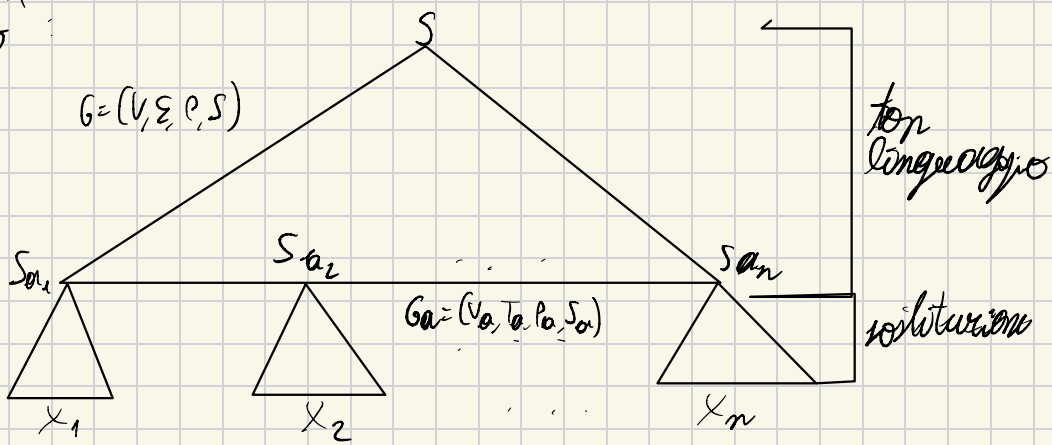
per $L \subseteq \Sigma^*$, $s(L) = \bigcup_{w \in L} s(w)$

s: SOSTITUZIONE

TEO: sia L CFL con Σ, s sostituzione su Σ tale che

$s(a)$ CFL $\forall a \in \Sigma \Rightarrow s(L)$ è CFL

Dim: G':



genero $S_{a_1}, \dots, S_{a_n}$ e genero $X_1 \dots X_n = w$

G: $V' = (\bigcup_{a \in \Sigma} V_a) \cup V$,

$T' = \bigcup_{a \in \Sigma} T_a$

$P' = (\bigcup_{a \in \Sigma} P_a) \cup P_2$

prendo P_2 ad ogni terminale in body metto simbolo S_a

$L(G) = r(L)$

$\supseteq$: $w \in r(L) \Rightarrow$ esiste $x \in L \mid x = a_1 \dots a_n \Rightarrow$

$\Rightarrow$ esistono $x_i \in r(a_i) \mid w = x_1 \dots x_n$

$\subseteq$: $w \in L(G) \Rightarrow w$ avrà stesso parse tree dolgo sottoalberi

(quercdove)

Esiste closure per CFL sotto:

1- unione

2- concatenazione

3- Kleene / positive closure

4- homomorphism

Dim: 1- dati $L_1, L_2 \Rightarrow$ CFL $L = \{1,2\}$, $r(1) = L_1$, $r(2) = L_2 \Rightarrow$

$\Rightarrow L_1 \cup L_2 = r(L)$

2- // // $\Rightarrow$ CFL $L = \{12\}$ // // $\Rightarrow$

$\Rightarrow L_1 L_2 = r(L)$

3- // $\Rightarrow$ CFL $L = \{1\}^{*/+}$ // $\Rightarrow L_1^{*/+} = s(L)$

4- dato L, $s(L) = \{h(a)\} \Rightarrow h(L) = r(L)$

Closure under string reverse

Theorem If L is a CFL, then so is L^R

Proof Assume L is generated by a CFG $G = (V, T, P, S)$. We build $G^R = (V, T, P^R, S)$, where

$$P^R = \{A \to \alpha^R \mid (A \to \alpha) \in P\}$$

inverting rhs

Using induction on derivation length in G and in G^R, we can show that $(L(G))^R = L(G^R)$ (omitted) □

CFL & intersection ⇒ CFL non chiuso sotto intersezione

$L_1 = \{0^n 1^n 2^i \mid n \geqslant 1,\ i \geqslant 1\}$ is a CFL, generated by the CFG

$$S \to AB$$
$$A \to 0A1 \mid 01$$
$$B \to 2B \mid 2$$

$L_2 = \{0^i 1^n 2^n \mid n \geqslant 1,\ i \geqslant 1\}$ is a CFL, generated by the CFG

$$S \to AB$$
$$A \to 0A \mid 0$$
$$B \to 1B2 \mid 12$$

$L_1 \cap L_2 = \{0^n 1^n 2^n \mid n \geqslant 1\}$ which is not a CFL

This was proved in a previous example

Intersection between CFL and regular language

Theorem Let L be some CFL and let R be some regular language. Then $L \cap R$ is a CFL

Proof Let L be accepted by the PDA

final state $\Rightarrow$ $P = (Q_P, \Sigma, \Gamma, \delta_P, q_P, Z_0, F_P)$

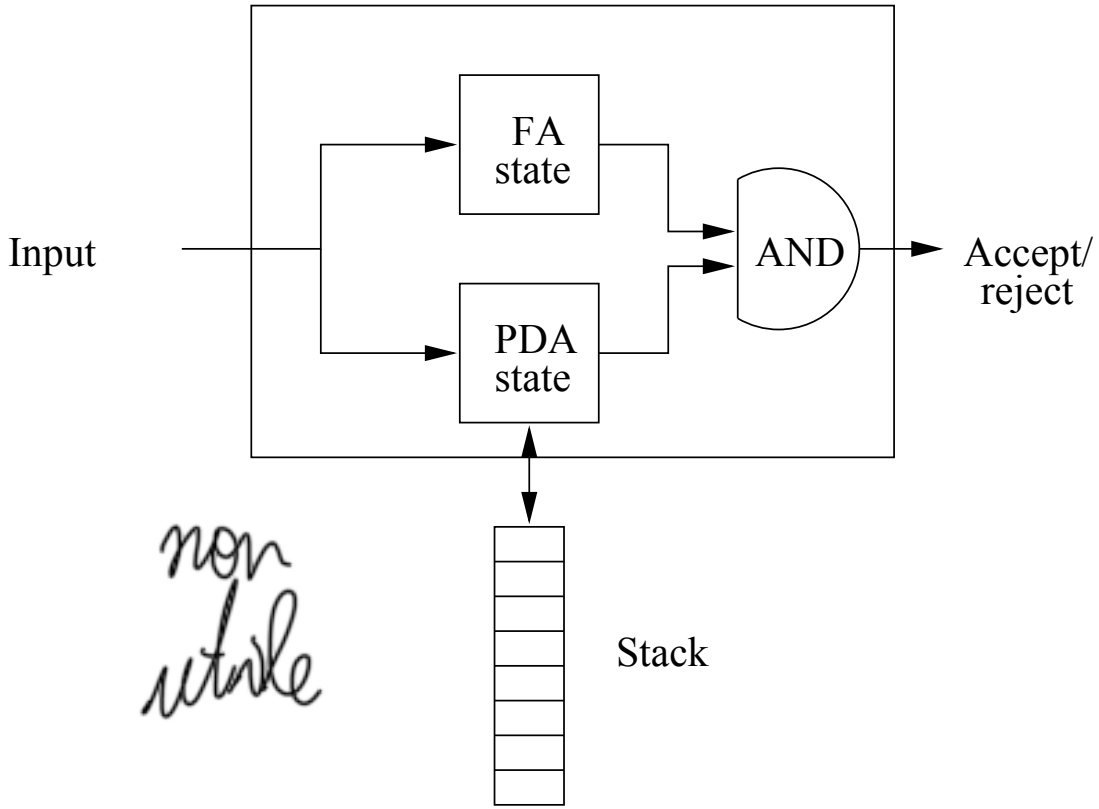
by final state, and let R be accepted by the DFA

$$A = (Q_A, \Sigma, \delta_A, q_A, F_A)$$

Intersection between CFL and regular language

by final state

We construct a PDA for $L \cap R$ based on the following idea



non determ

Intersection between CFL and regular language

We define

$$P' = (Q_P \times Q_A, \Sigma, \Gamma, \delta, (q_P, q_A), Z_0, F_P \times F_A)$$

where $(a \in \Sigma \cup \{\epsilon\})$

$\hat{\delta}$ perché può essere $a = \epsilon$

$$\delta((q,p), a, X) = \{((r,s), \gamma) \mid (r,\gamma) \in \delta_P(q,a,X), s = \hat{\delta}_A(p,a)\}$$

We can show (omitted) by induction on the number of steps in the computation $\overset{*}{\vdash}$ that

$$(q_P, w, Z_0) \overset{*}{\underset{P}{\vdash}} (q, \epsilon, \gamma)$$

se accetta per final state

if and only if

$$((q_P, q_A), w, Z_0) \overset{*}{\underset{P'}{\vdash}} ((q,p), \epsilon, \gamma), \quad p = \hat{\delta}(q_A, w)$$

PDA output accetta — DFA accetta

Intersection between CFL and regular language

(q, p) is an accepting state of P' if and only if

- q is an accepting state of P
- p is an accepting state of A

Therefore P' accepts w if and only if both P and A accept w, that is, $w \in L \cap R$ □

Other properties for CFLs

Theorem Let L, L_1, L_2 be CFLs and let R be a regular language. Then

- $L \setminus R$ is a CFL
- $\overline{L}$ may fall outside of CFLs
- $L_1 \setminus L_2$ may fall outside of CFLs

Proof

Operator $\setminus$ with REG : $\overline{R}$ is regular, $L \cap \overline{R}$ is CFL, and $L \cap \overline{R} = L \setminus R$

Other properties for CFLs

Complement operator : If $\overline{L}$ would always be a CFL, then we have that

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

would always be CFL, which is a contradiction

Operator $\setminus$ with CFL : Σ^* is a CFL. If $L_1 \setminus L_2$ would always be a CFL, then $\Sigma^* \setminus L = \overline{L}$ would always be a CFL, which is a contradiction $\square$

Test

Assert whether the following statements hold, and motivate your answer

- the intersection of a non-CFL L_1 and a CFL L_2 can be a non-CFL
- the intersection of a non-CFL and a finite language is always a CFL

(↳ TRUE: non-CFL, quindi infinito ⇒
⇒ inters. finita, quindi reg, quindi CFL

Computational properties for CFLs

We investigate the **computational complexity** for some of the transformations previously presented

We need these results to establish the efficiency of some decision problems which we will consider later

We denote with n the **length** of the entire representation of a PDA or a CFG (for more detailed results, we should instead distinguish between number of variables, number of stack symbols, etc.)

Computational properties for CFLs

The following conversions can be computed in time $\mathcal{O}(n)$

- conversion from PDA accepting by final state to PDA accepting by empty stack
- conversion from PDA accepting by empty stack to PDA accepting by final state
- conversion from CFG to PDA

Given a PDA of size n we can build an equivalent CFG in time (and space) $\mathcal{O}(n^3)$, using a **preliminary binarization** of the transitions of the autmaton

The construction of Chapter 6 (which we have not presented) requires exponential time

Conversion to CNF

We can compute in time $\mathcal{O}(n)$

- the set of reachable symbols $r(G)$
- the set of generating symbols $g(G)$
- the elimination of useless symbols from a CFG

Conversion to CNF

We can compute in time $\mathcal{O}(n)$ the set of nullable symbols $n(G)$

We can compute in time $\mathcal{O}(n)$ the elimination of ϵ-productions from a CFG, using a **preliminary binarization** of the grammar

in rhs, non-t di 2 simboli per prod.

We can compute in time $\mathcal{O}(n^2)$ the set of unary symbols $u(G)$ and the elimination of unary productions from a CFG

Conversion to CNF

We can compute in time $\mathcal{O}(n)$ the replacement of terminal symbols with variables (first transformation for CNF)

We can compute in time $\mathcal{O}(n)$ the reduction of production with right-hand side length larger than 2 (second transformation for CNF)

Given a CFG of size n, we can construct an equivalent CFG in CNF in time (and space) $\mathcal{O}(n^2)$

Emptiness test

Let G be some CFG with start symbol S. $L(G)$ is empty if and only if S is not generating

We can then test emptiness for $L(G)$ using the already mentioned algorithm for the computation of $g(G)$, running in time $\mathcal{O}(n)$

CFL membership

The **membership problem** for a CFL string is defined as follows

Given as input a string w, we want to decide whether $w \in L(G)$, where G is some **fixed** CFG

Note : G does not depend on w and is **not** considered part of the input for our problem. Therefore the length of G does not affect the running time of the problem

CFL membership

Assume G in CNF and $|w| = n$. Since the parse trees for w are binary, the number of internal nodes for each tree is $2n - 1$ (proof by induction)

We can therefore generate all the parse trees of G with $2n - 1$ nodes and test whether some tree yields w

There are more efficient algorithms that take advantage of **dynamic programming** techniques

CFL membership

CNF

Let $w = a_1 a_2 \cdots a_n$. We construct a triangular **parse table** where cell X_{ij} is set valued and contains all variables A such that

$$A \overset{*}{\underset{G}{\Rightarrow}} a_i a_{i+1} \cdots a_j$$

bottom up: prima lunghezza 1, poi lunghezza 2 combinando quelle di lunghezza 1

(<> possono generare sottostringa (i, j)

X_{15}				
X_{14}	X_{25}			
X_{13}	X_{24}	X_{35}		
X_{12}	X_{23}	X_{34}	X_{45}	
X_{11}	X_{22}	X_{33}	X_{44}	X_{55}
a_1	a_2	a_3	a_4	a_5

⇒ si guarda in regole CNF: BASE CASE

CFL membership

We **iteratively** construct the parse table, one row at a time and from bottom to top

First row is populated with the base case, while remaining rows are populated by the inductive case

Idea : $A \overset{*}{\underset{G}{\Rightarrow}} a_i a_{i+1} \cdots a_j$ if and only if

- for some production $A \to BC$ A divisa
- for some integer k with $i \leqslant k < j$ punto di divisione

we have $B \overset{*}{\underset{G}{\Rightarrow}} a_i a_{i+1} \cdots a_k$ and $C \overset{*}{\underset{G}{\Rightarrow}} a_{k+1} a_{k+2} \cdots a_j$

CFL membership

Base $X_{ii} \leftarrow \{A \mid (A \rightarrow a_i) \in P\}$ ⇒ 1ª riga

Induction We build X_{ij} for increasing values of $j - i \geqslant 1$

$X_{ij} \leftarrow X_{ij} \cup \{A\}$ if and only if there exist k, B, C such that

- $i \leqslant k < j$
- $(A \rightarrow BC) \in P$
- $B \in X_{ik}$ and $C \in X_{k+1,j}$

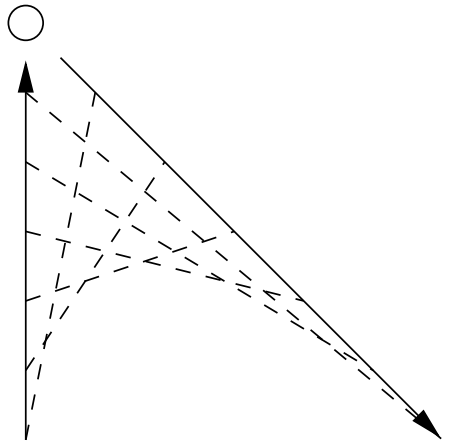
⇓

k: punto di taglio

CFL membership

In the inductive case, to populate X_{ij} we need to check at most n pairs of previously built cells of the parse table

$$(X_{ii}, X_{i+1,j}),\ (X_{i,i+1}, X_{i+2,j}),\ \ldots,\ (X_{i,j-1}, X_{jj})$$



The operation above is related to vector convolution

CFL membership

We assume we can compute each check $B \in X_{ik}$ in time $\mathcal{O}(1)$. Then each set X_{ij} can be populated in time $\mathcal{O}(n)$

We need to populate $\mathcal{O}(n^2)$ sets X_{ij}

We summarize all of the previous observations by means of the following statement

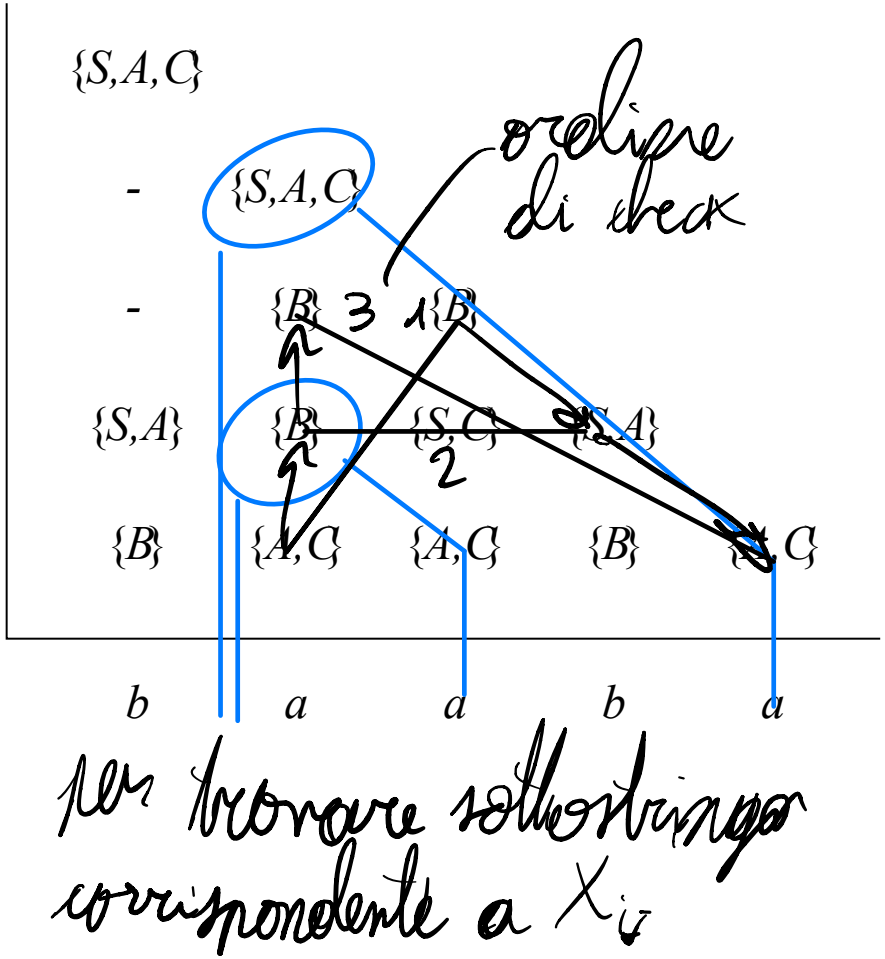
Theorem The algorithm for the construction of the parse table computes all of the sets X_{ij} in time $\mathcal{O}(n^3)$. We then have $w \in L(G)$ if and only if $S \in X_{1n}$

Example

Let G be a CFG with productions

$$S \to AB \mid BC$$
$$A \to BA \mid a$$
$$B \to CC \mid b$$
$$C \to AB \mid a$$

and let $w = baaba$



24/11

Automata, Languages and Computation

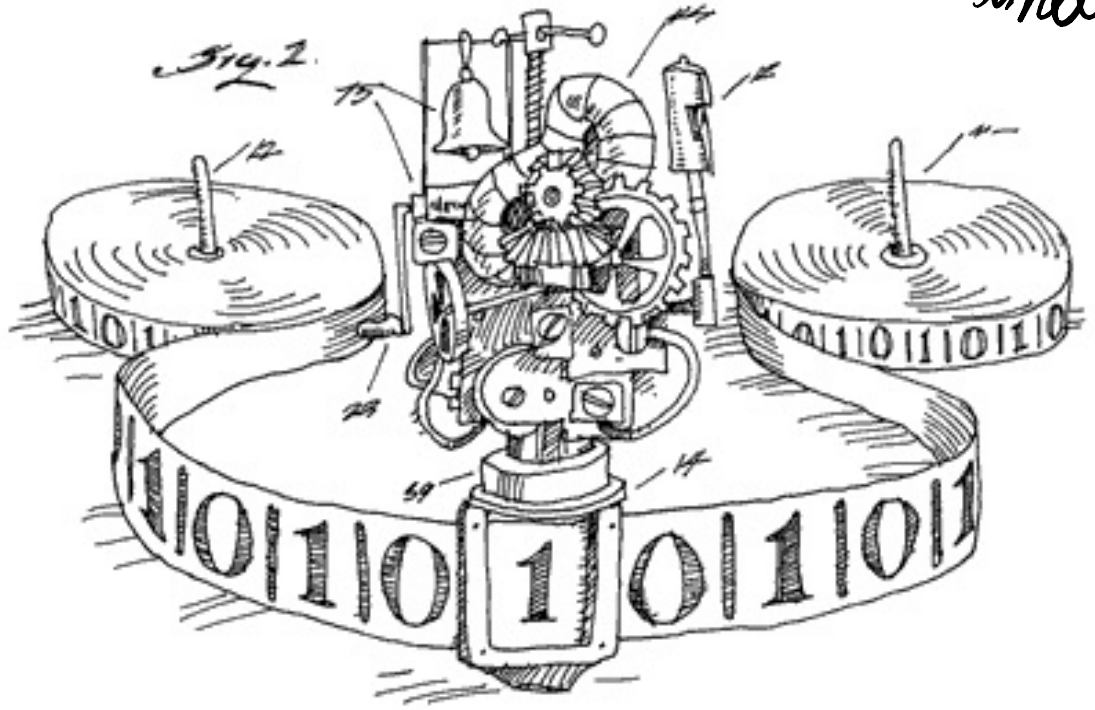
Chapter 8 : Turing Machines

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Lecture based on material originally developed by :
Gösta Grahne, Concordia University

Turing machines

memoria: posso accedere dove voglio, andare avanti e indietro



1. Turing machine (TM) : formal model of computer algorithms that allows the mathematical study of computability
2. Programming techniques for TM : techniques to facilitate the writing of programs for TM
3. TM Extensions : machines that are more complex than TM but with the same computational capacity
4. TM with restrictions : automata that are simpler than TM but with the same computational capacity

Turing machine

In order to mathematically study undecidabilty we need a **simple** formalism to represent programs (Python is **not** suitable)

Historically used formalisms:

- predicate calculus (Gödel, 1931) ⇒ logica per creare programmi
- partial recursive functions (Kleene, 1936) } simili a functional programming
- lambda calculus (Church, 1936)
- Turing machine (Turing, 1936)

determinare cosa si può calcolare

Turing machine

recognition device

A Turing machine is a finite state automaton with the addition of a **memory tape** with

- sequential access
- unlimited capacity in both tape directions

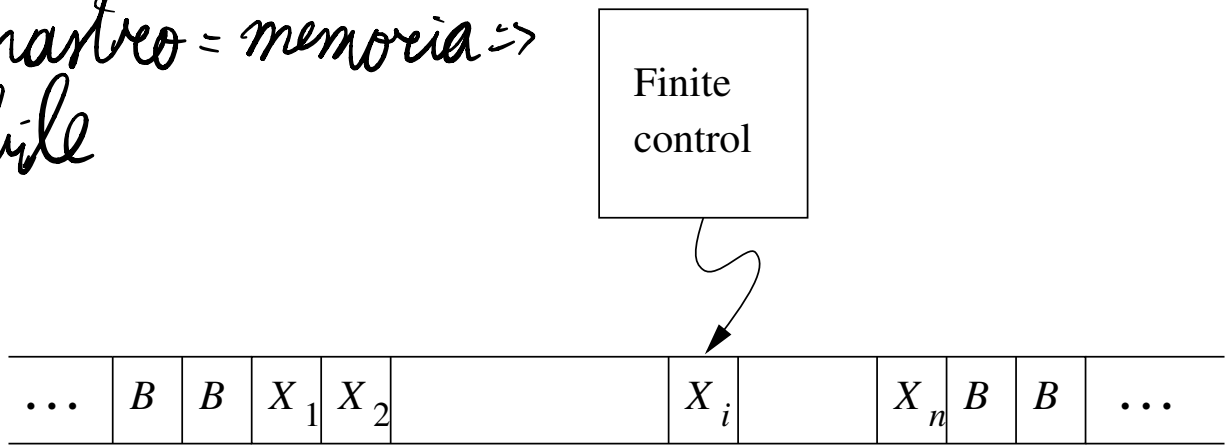
Differently from the PDA model, input string is initially placed into the auxiliary memory

The Turing machine model allows the study of computability properties such as **undecidabilty** and **intractability**

irrisolvibile risolvibile non efficientemente

Turing machine

stringa = nastro = memoria =>
=> riscrivibile



Informally, a Turing machine performs a move according to its state and the symbol which is read by the tape head

In a single move, a Turing machine

- changes its state
- writes a new symbol in the cell read by the tape head
- moves the tape head to the cell to the right or to the left

Turing machine

A **Turing machine**, MT for short, is a 7-tuple

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

where

- Q is a finite set of **states**
- Σ is a finite set of **input symbols**
- Γ is a finite set of **tape symbols**, with $\Sigma \subseteq \Gamma$
- δ is a **transition function** from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$
- q_0 is the **initial state**
- $B \in \Gamma$ is the **blank symbol**, with $B \notin \Sigma$
- $F \subseteq Q$ is the set of **final states**

"vuoto" ⇒ non c'è inizio o fine per tape

Note that the automaton is deterministic, and it has no 'stand' move

Instantaneous descriptions

A TM changes its configuration with each move. We use the notion of instantaneous description (ID) to describe configurations

An **instantaneous description** (ID) of M is a string of the form

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n$$

where

- q is M's state
- $X_1 X_2 \cdots X_n$ is the "visited" portion of M's tape visitata finora
- the tape head of M is reading the i-th tape symbol

Computation of a TM

To represent a **computation step** of M we use the binary relation $\vdash_M$ defined on the set of IDs

If $\delta(q, X_i) = (p, Y, L)$, then

X_i sovrascritto

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \vdash_M X_1 X_2 \cdots p X_{i-1} Y X_{i+1} \cdots X_n$$

If $\delta(q, X_i) = (p, Y, R)$, then

mossa a sinistra, ora leggo X_{i-1}

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \vdash_M X_1 X_2 \cdots X_{i-1} Y p X_{i+1} \cdots X_n$$

Special cases if the tape head is at the two ends of the written tape

(in libro)

Computation of a TM

To represent the **computations** of M, we use the reflexive and transitive closure of $\underset{M}{\vdash}$, written $\overset{*}{\underset{M}{\vdash}}$

For input string $w \in \Sigma^*$, the initial ID is $q_0 w$

For a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, an accepting computation has the form

$$q_0 w \overset{*}{\underset{M}{\vdash}} \alpha p \beta$$

with $p \in F$ and $\alpha, \beta \in \Gamma^*$

We will come back to this definition after some examples

Example

Let us specify a TM M with $L(M) = \{0^n 1^n \mid n \geqslant 1\}$

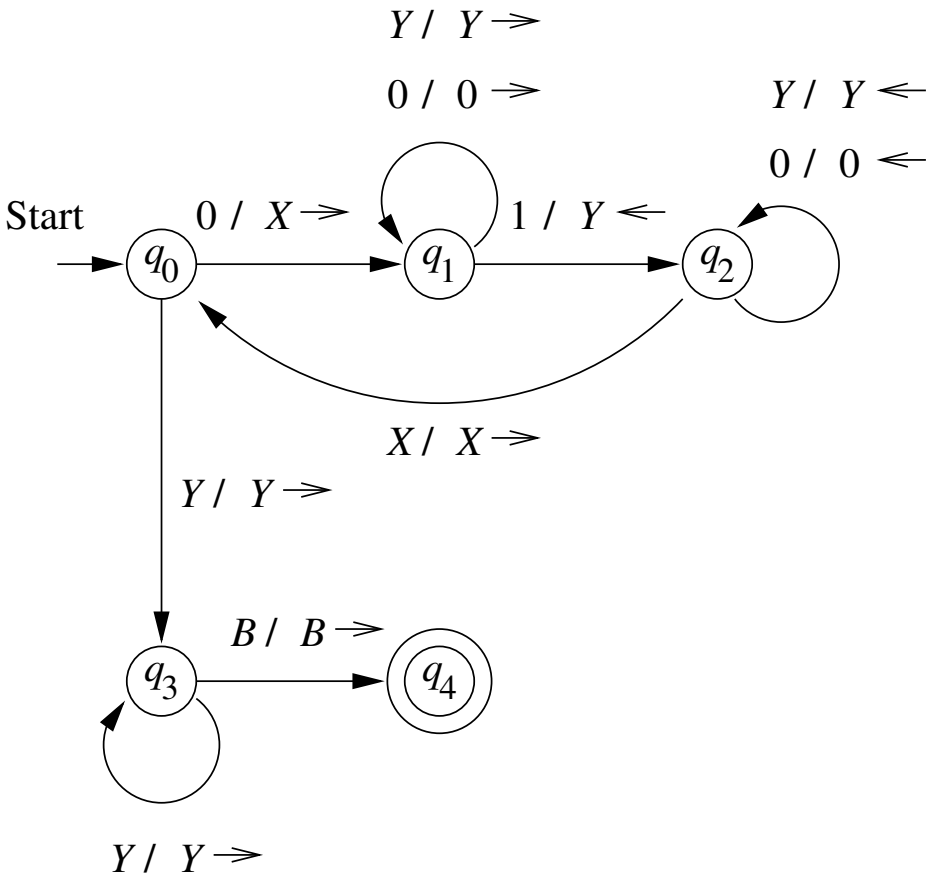
$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

The transition function δ is represented by the following table

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	(q_4, B, R)
$\star q_4$					

Example

We can also represent δ by means of the following **transition diagram**



Example

If input w has the form $\mathbf{0^*1^*}$, then at each ID the tape is of the form $\boldsymbol{X^*0^*Y^*1^*}$ $\Gamma = \Sigma \cup \{X, Y\}$ ⇒ MARKERS

M implements the following strategy

- in q_0 it replaces the leftmost 0 with X and moves to q_1
- in q_1 it proceeds from left to right, goes over 0 and Y looking for the leftmost 1, replaces it with Y and moves to q_2
- in q_2 it proceeds from right to left, goes over Y and 0 looking for the rightmost X, and moves back to q_0
- in q_0, if it finds one more 0 it resumes the above cycle, otherwise it moves to q_3
- in q_3 it overrides all of the Y's and accepts if there is no 1

Observe how input string is overwritten during the computation

Example

Given the string input 0011, M performs the following computation (sequence of ID)

$$\begin{aligned} q_0 0011 &\vdash X q_1 011 \vdash X 0 q_1 11 \\ &\vdash X q_2 0 Y 1 \vdash q_2 X 0 Y 1 \\ &\vdash X q_0 0 Y 1 \vdash X X q_1 Y 1 \\ &\vdash X X Y q_1 1 \vdash X X q_2 Y Y \\ &\vdash X q_2 X Y Y \vdash X X q_0 Y Y \\ &\vdash X X Y q_3 Y \vdash X X Y Y q_3 B \\ &\vdash X X Y Y B q_4 B \end{aligned}$$

TM with "output"

We have defined a TM as a recognition device. Alternatively, we can use these devices to compute **functions** on natural numbers.

Historically, this was the original definition by A. Turing

We encode each natural number in **unary notation** according to the scheme

$$n =_1 0^n$$

$7 \Rightarrow 0000000$

Example

The following TM M computes the **proper subtractor** function

$$m \dot{-} n = max(m - n, 0)$$

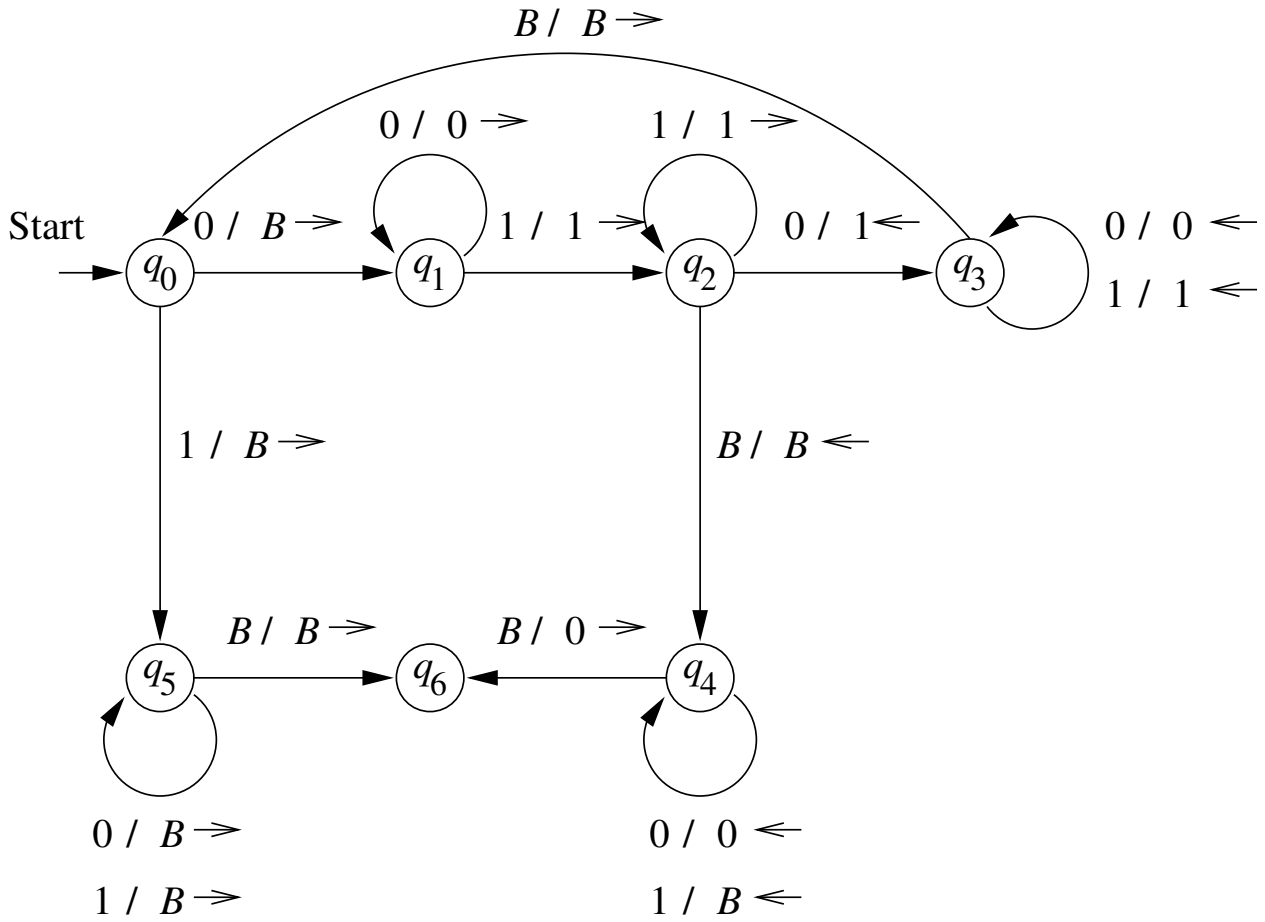
starting with $0^m 1 0^n$ on its tape and **halting** with $0^{m \dot{-} n}$.

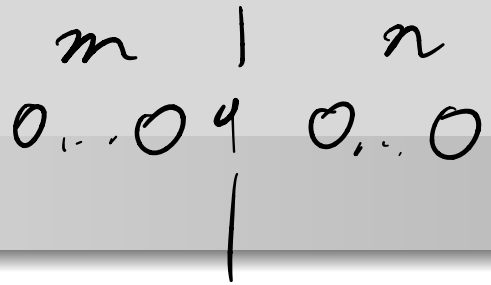
No set of final states for TMs with output

	0	1	B
$\rightarrow q_0$	(q_1, B, R)	(q_5, B, R)	
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)
$\star q_6$			

Example

The transition diagram is





Example

The TM M performs the following loop

- find the leftmost 0 and replace with B (states q_0, q_3)
- search right for the first 0 placed after symbols 1, and replace it with 1 (states q_1, q_2)

The loop ends in two possible ways

- M cannot find a 0 to the right of the 1's ($m > n$); then M turns all of the 1's into a single 0 followed by B's $\Rightarrow$ convention results to in unary notation
- M cannot find a 0 to be replaced by B ($m \leqslant n$); then $m \dot{-} n = 0$ and M replaces all 0's and 1's into B

Notation for TM

We use notational **conventions** similar to those of other automata

- a, b, c, ..., a_1, a_2, ..., a_i, ... input symbols
- X, Y, Z tape symbols
- u, w, x, y, z strings over the input alphabet
- α, β, γ, ..., strings over tape alphabet
- p, q, r, ..., q_1, q_2, ..., q_i, ... states

Language accepted by a TM

A TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ **accepts** the language

$$L(M) = \{w \mid w \in \Sigma^*,\ q_0 w \overset{*}{\underset{M}{\vdash}} \alpha p \beta,\ p \in F,\ \alpha, \beta \in \Gamma^*\}$$

The class of languages accepted by TMs is called **recursively enumerable** (RE)

This term derives from formalisms that historically preceded TM

TM and halting

A TM **halts** if it enters a state q with tape symbol X and $\delta(q, X)$ is not defined (there is no next move)

If a TM accepts a string, we can assume that it always halts : just make $\delta(q, X)$ undefined for every final state q

If a TM does not accept, **we can't** assume that it will halt (in a non-final state)

The class of languages accepted by some TM that halts for every input are called **recursive** (REC)

28/11

Es1 (tema 02/2019, es 3, 6 punti)

a. $L_1, L_2 \notin REG$ (regolari); $L_1 \cup L_2$ non può essere in REG:
FALSO ⇒ diamo controesempio (dimostrare proprio dice: non REG, dobbiamo visto in classe)
$\Sigma = \{a, b\}$, $L_1, L_2 \subseteq \Sigma^*$; $L_1 = \{w \mid \#_a(w) \geq \#_b(w)\}$ ⇒ non reg.
(non possiamo contare e confrontare num. così) ⇒
⇒ $L_2 = \{w \mid \#_a(w) \leq \#_b(w)\}$: non reg. ⇒ $L_1 \cup L_2 = \Sigma^* \in REG$

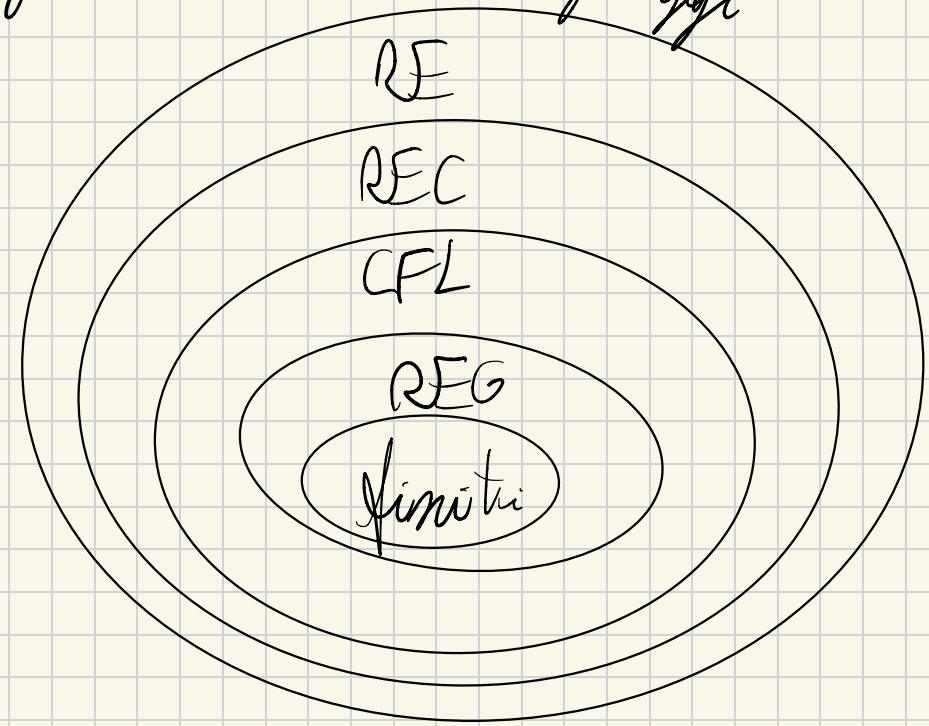
b. $L_1 \in CFL$, ogni sottoinsieme di L_1 è CFL: FALSO
$L_1 = \{a^i b^j c^k \mid i,j,k \geq 1\}$, $L \subseteq L_1$: L: $i = j = k$ ⇒
⇒ non CFL per pumping lemma (già visto in classe)

c. $L_1 \in CFL$, $L_2 \in REG$: $L_1 \cap L_2$ può essere REG: VERO

d. // // $L_1 \cap L_2 \notin REG$ possibile: VERO
$L_1 = \{a^n b^n \mid n \geq 1\}$ CFL, non REG), $L_2 = \Sigma^*$ REG ⇒
⇒ $L_1 \cap L_2 = L_1 \in CFL$

linguaggi

altri linguaggi



Oggetto $x \Rightarrow P(x)$: PREDICATO (proprietà) di $x \Rightarrow$
$\Rightarrow$ DECISION PROBLEM: decidere se $P(x)$ vero per
input $x \Rightarrow$ linguaggio associato: $L_P = \{x \mid P(x) \text{ vero}\}$
Molti problemi matematici non sono decision
problems, ma possono essere trasformati $\Rightarrow$
$\Rightarrow$ prob. generali non è + facile di decision problem
associato

Algoritmo per dec. prob. usando algoritmo per
gen., come subroutine (REDUCTION TECHNIQUE):
- input $\langle A, B, C \rangle$
- subroutine(s) su A B per produrre $C' = A \times B$
- controllare $C' = C$

Recursive and recursively enumerable languages

REC ⊂ RE

Recursive language (REC) : the language is accepted by a TM that halts on each input string (in the language or not)

associato sottoinsieme di tutte TM

Recursively enumerable language (RE) : the language is accepted by a TM that halts when the string belongs to the language

dove associata di TM: tutte possibili

For strings not in the language, the TM may compute forever

A decision problem P is **decidable** if its encoding L_P (see chapter 1) is a recursive language. Alternatively : if there is a TM M that always halts such that $L(M) = L_P$

(cap. 1, slide 40)

Programming techniques for TM

Although the class of TM is very simple, this model has the **same** computational power as a modern computer

We will also see that a TM is able to perform processing on other TMs. This allows us to prove that certain problems are undecidable

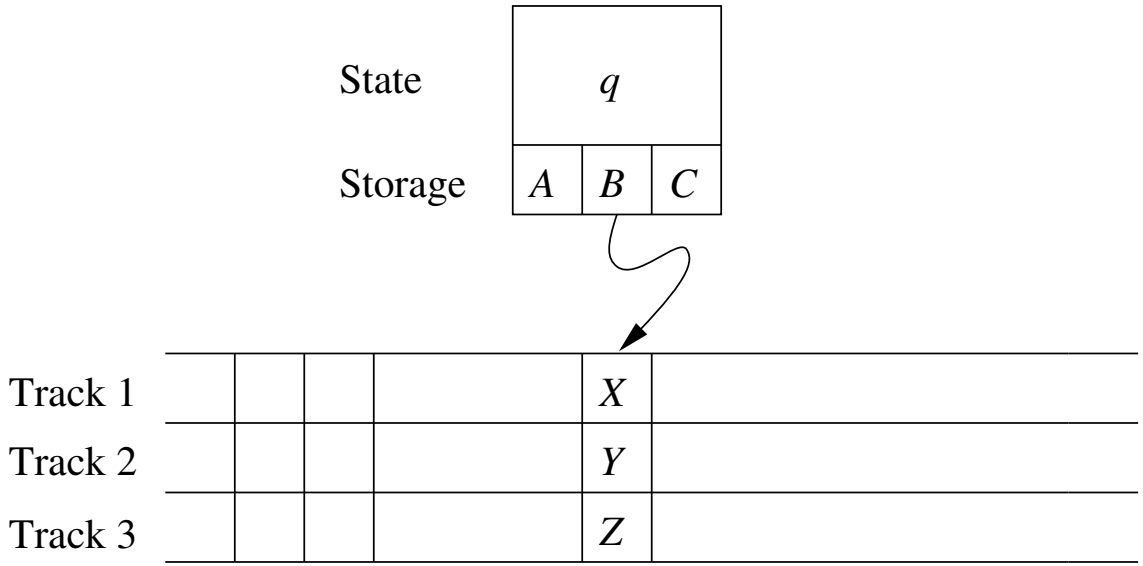
Compare with compilers, which take programs as input and produce new programs as output

We present in the following some **notational variants** of the TM that make TM programming easier

Programming techniques for TM

We reformulate the TM definition using

- a finite number of registers with **random access**, which we place inside each state
- a finite number of tape tracks



State as internal memory

Example : A TM M that "memorizes" the first symbol read and verifies that this does not appear again in the input

$$L(M) = L(01^* + 10^*)$$

Let $M = (Q, \{0,1\}, \{0,1,B\}, \delta, [q_0, B], B, \{[q_1, B]\})$, with $Q = \{q_0, q_1\} \times \{0,1,B\}$

	0	1	B
$\rightarrow [q_0, B]$	$([q_1, 0], 0, R)$	$([q_1, 1], 1, R)$	
$[q_1, 0]$		$([q_1, 0], 1, R)$	$([q_1, B], B, R)$
$[q_1, 1]$	$([q_1, 1], 0, R)$		$([q_1, B], B, R)$
$\star[q_1, B]$			

Tape with multiple tracks

Example : A TM for the language $L = \{wcw \mid w \in \{0,1\}^*\}$

We use a tape track for "marking" those input symbols that we have already tested

$$M = (Q, \Sigma, \Gamma, \delta, [q_1, B], [B, B], \{[q_0, B]\})$$

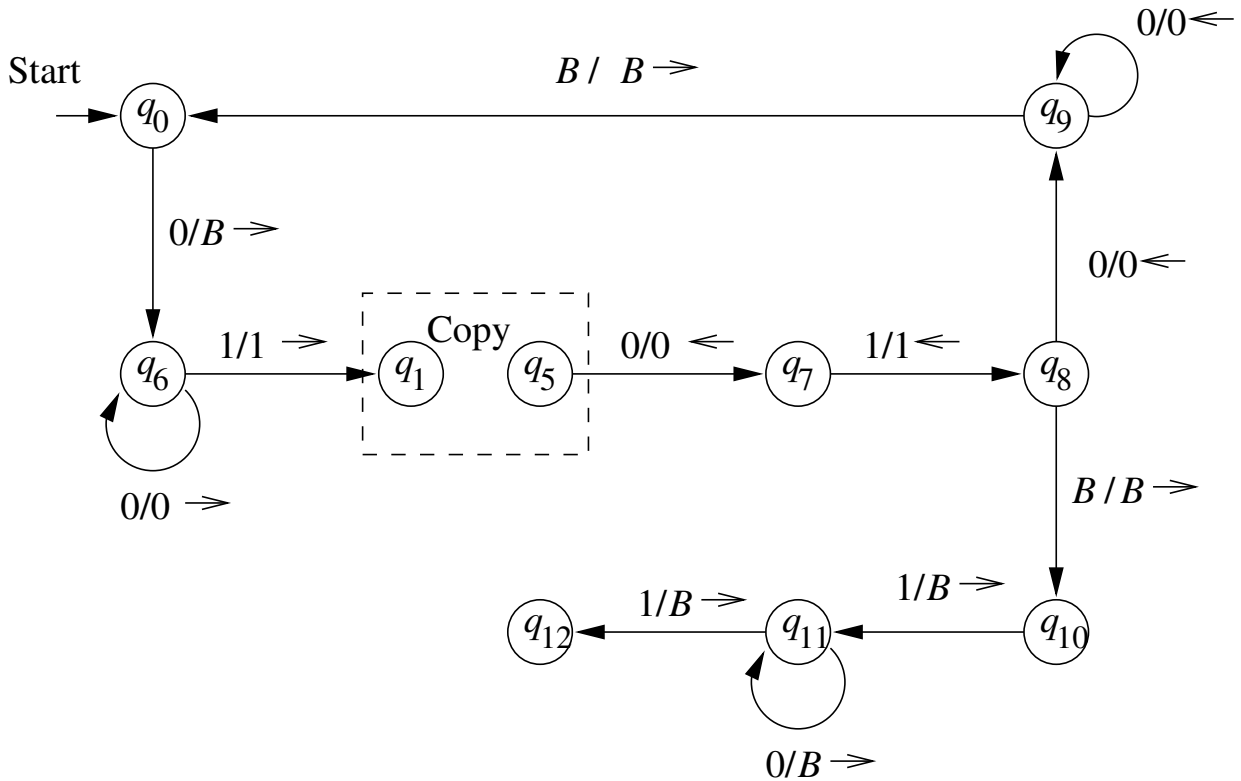
where

- $Q = \{q_1, q_2, \ldots, q_9\} \times \{0, 1, B\}$
- $\Sigma = \{[B, 0], [B, 1], [B, c]\}$
- $\Gamma = \{B, *\} \times \{0, 1, c, B\}$

See the textbook for the specification of the transition function δ

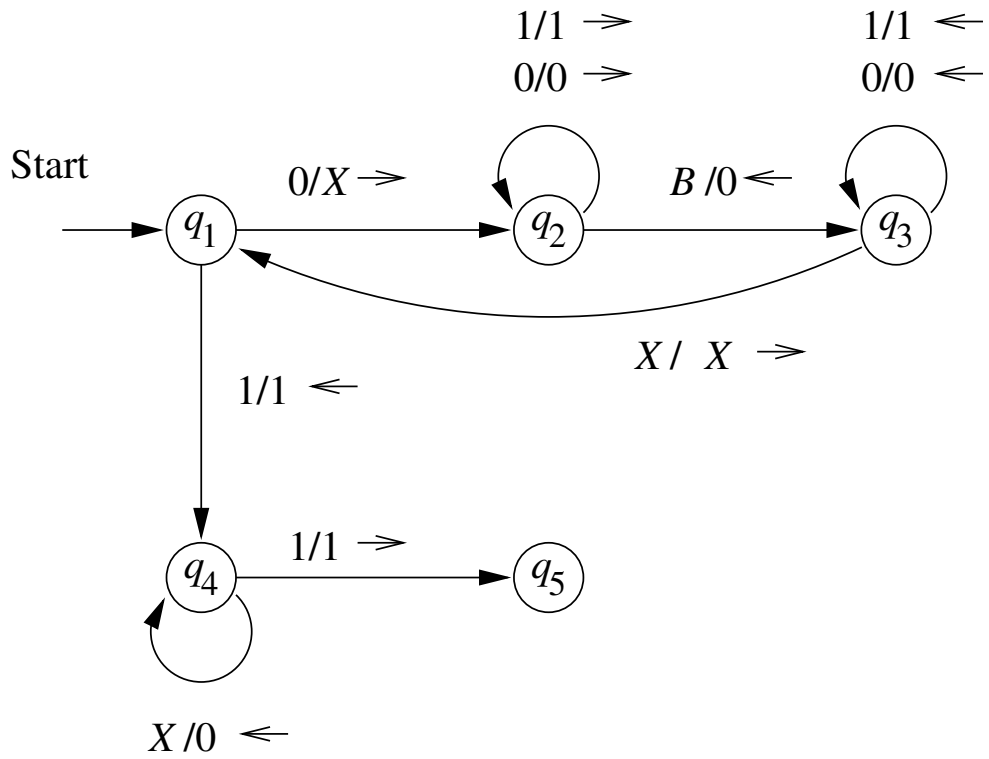
Use of a subroutine

Example : A TM for the computation of the product function $0^m 1 0^n 1 \mapsto 0^{m \cdot n}$. We use a *subroutine* "Copy"



Use of a subroutine

The subroutine "Copy" takes ID $0^{m-k}1q_10^n10^{(k-1)n}$ to ID $0^{m-k}1q_50^n10^{kn}$



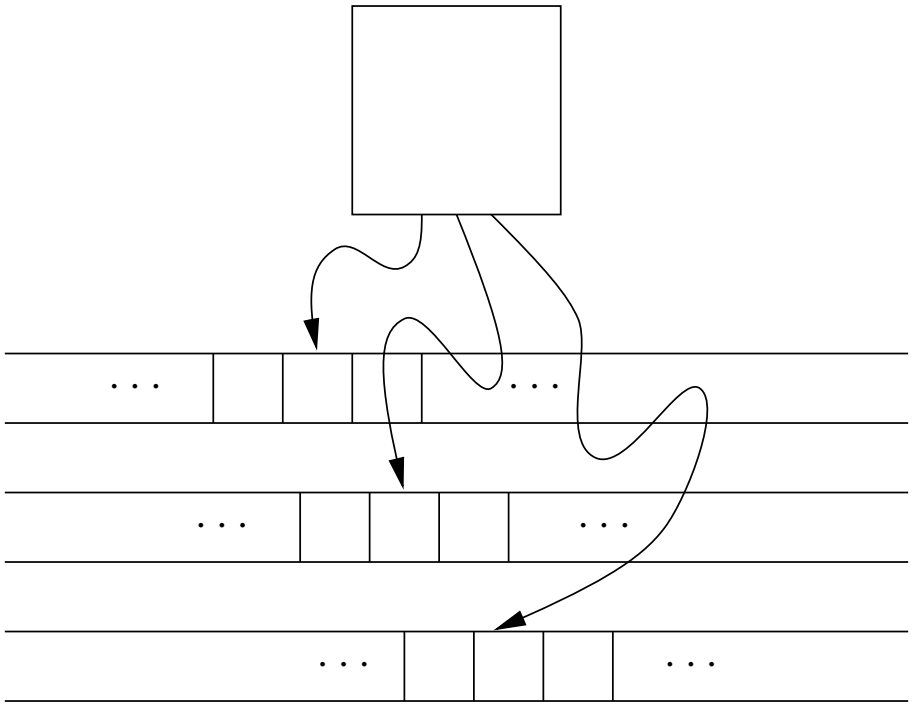
TM extensions

Let us now present some **extensions** of the TM definition

For each extension, we prove that the **computational capacity** is the same as the one of the classic definition of TM

Multi-tape TM

We use a finite number of **independent** tapes for the computation, with the input on the first tape



Multi-tape TM

In a single move the multi-tape TM performs the following actions

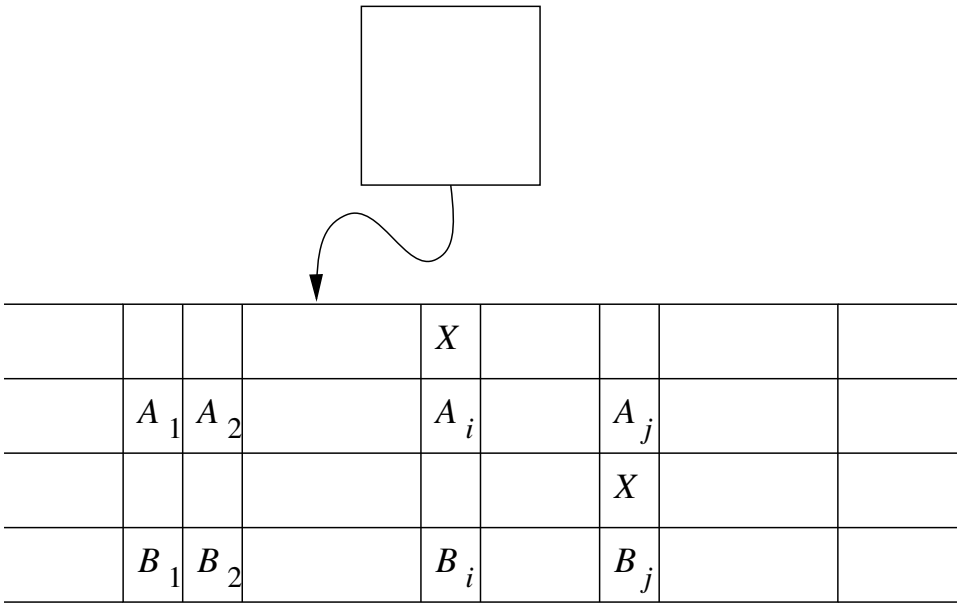
- state update, on the basis of read tape symbols
- for each tape :
  - write a symbol in current cell
  - move the tape head independently of the other heads (L = left, R = right, or S = stay)

Note that the stay option is not available in a TM

Multi-tape TM

Theorem A language accepted by a multi-tape TM M is RE

Proof (sketch) We can simulate M using a TM N with a multi-track tape



Multi-tape TM

We use $2k$ tracks to simulate k tapes : even tracks used for tape content, odd tracks used for tape head position

N visits all k head positions to **simulate** a single move of M

- left to right pass : the number of visited tape heads and the content of the corresponding cells are stored into the state of N
- right to left pass : for each tape head of M, the corresponding action is simulated by N

N updates its state in the same way as M □

Multi-tape TM

Theorem The TM N in the proof of the previous theorem simulates the first n moves of the TM M with k tapes in time $\mathcal{O}(n^2)$

Proof (sketch) After n moves of M, tape head markers in N have **mutual distance** not exceeding $2n$

It follows that any one of the first n moves of M can be simulated by N in a number of moves not exceeding $4n + 2k$, which amounts to $\mathcal{O}(n)$ since k is a constant $\square$

Nondeterministic TM

In a **nondeterministic** Turing machine, NTM for short, the transition function δ is set-valued :

$$\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$$

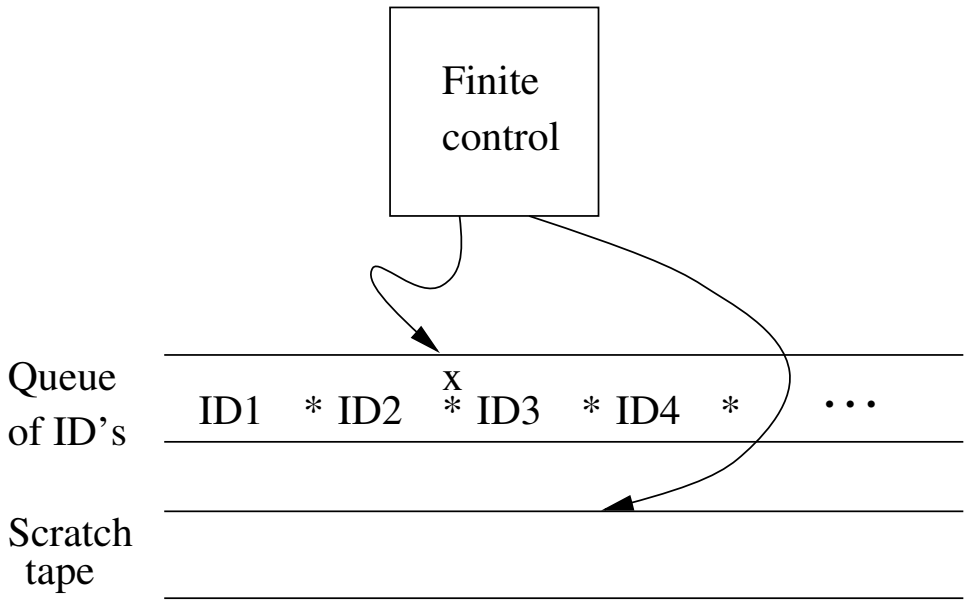
At each step, the NTM chooses one of the triples as the next move

The NTM accepts an input w if **there exists** a sequence of choices that leads from the initial ID for w to an ID with an accepting state

Nondeterministic TM

Theorem For each NTM M_N, there exists a (deterministic) TM M_D such that $L(M_N) = L(M_D)$

Proof (skecth) We specify M_D as a TM with two tapes



Nondeterministic TM

A single ID in the **queue** (first) tape is marked as being processed

M_D performs the following cycle

- copy the marked ID from the queue tape to the **scratch** (second) tape
- for each possible move of M_N, add a new ID at the end of queue tape
- move the marker in the queue tape to the next ID

Nondeterministic TM

Let m be the maximum number of choices for M_N. After n moves, M_N reaches a number of ID bounded by

$$1 + m + m^2 + \cdots + m^n \leqslant nm^n + 1$$

M_D explores all the IDs reached by M_N in n steps before each ID reached in $n + 1$ steps, as in a **breadth first** search

If there exists an accepting ID for M_N on w, M_D reaches this ID in a finite amount of time. Otherwise, M_D does not accept, and may not halt

We therefore conclude that $L(M_N) = L(M_D)$ □

Nondeterministic TM

Observe that the TM M_D in the previous theorem can take an amount of time **exponentially larger** than M_N to accept an input string

We **do not know** if this slowdown is necessary: this very important issue will be the subject of investigation in a next chapter

TM with restrictions

We impose some restrictions on the definition of TM / multi-tape TM:

- tape is unlimited only in one direction
- two tapes used in stack mode

We prove that these models are equivalent to TM

Think about the above definitions as normal forms

These models are especially useful in some proofs that we will present later on

TM with semi-infinite tape

In a TM with **semi-infinite tape**

- there are no cells to the left of the **initial tape position**
- a tape symbol can never be overwritten by the blank B

In a TM with semi-infinite tape each ID is a sequence of tape symbols other than B, i.e., there are no "holes"

TM with semi-infinite tape

We can **simulate** a TM by means of a TM with semi-infinite tape with two tracks

- the upper track represents the initial position X_0 and all tape cells to its right
- the lower track represents all tape cells to the left of X_0, in reverse order
- a special symbol $*$ is used to mark the initial position

X_0	X_1	X_2	$\cdots$
$*$	X_{-1}	X_{-2}	$\cdots$

TM with semi-infinite tape

Theorem Each language accepted by a TM M_2 is also accepted by a TM M_1 with semi-infinite tape

Proof (sketch) First, we modify M_2 in such a way that it uses a **new** tape symbol B' each time B is used to overwrite a tape symbol

Let $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, B, F_2)$ be the modified TM. We define

$$M_1 = (Q_1, \Sigma \times \{B\}, \Gamma_1, \delta_1, q_0, [B, B], F_1)$$

TM with semi-infinite tape

The states of M_1 are $Q_1 = \{q_0, q_1\} \cup (Q_2 \times \{U, L\})$. Symbols U, L indicate whether M_1 is visiting the upper or lower track

The input symbols of M_1 are pairs $[a, B]$ with a an input symbol of M_2

The tape symbols Γ_1 of M_1 are pairs in $\Gamma_2 \times \Gamma_2$ with the addition of pairs $[X, *]$ for each $X \in \Gamma_2$, where $*$ is used to mark the initial position of M_1 tape

The accepting symbols of M_1 are $F_1 = F_2 \times \{U, L\}$

TM with semi-infinite tape

Transitions in δ_1 implement the following moves

- place $*$ on the initial position, in the lower track, and restore the initial conditions of M_2
- when M_1 is not in the initial cell, the moves of M_2 are simulated with
  - the same direction if U appears in the state
  - the reverse direction if L appears in the state
- upon reading $*$
  - if M_2 moves to the right, M_1 simulates the same move
  - if M_2 moves to the left, M_1 simulates the same move but it reverses the direction

TM with semi-infinite tape

It can be shown by induction on the number of steps of a computation that the IDs of M_1 and M_2 match, modulo

- the reversal of the L track of M_1
- its concatenation on the left with the U track of M_1
- the elimination of the $*$ marker

It follows that $L(M_1) = L(M_2)$ $\square$

Multi-Stack machine

We apply to a multi-tape TM the restriction to use each tape in stack mode

- can only overwrite at the top
- can only insert at the top
- can only delete at the top

The resulting model accepts only recursively enumerable language, since it is a restriction of a multi tape TM

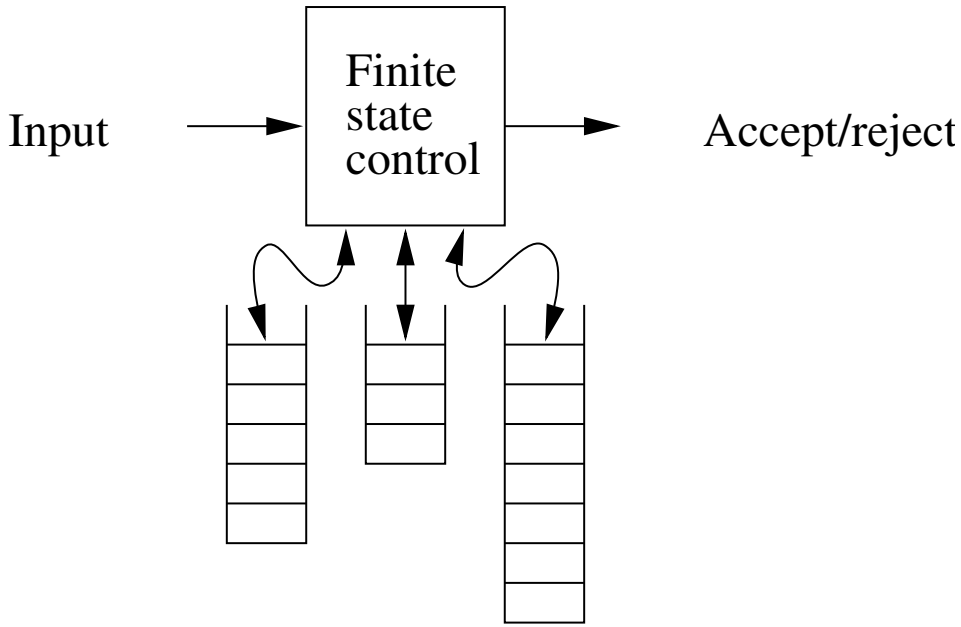
Multi-Stack machine

Let M be a multi-tape TM with tapes used in stack mode. We also assume that

- the input is provided in an **external**, read-only tape and with end-marker \$, and it can only be read from left to right
- M can perform ϵ-moves, but these moves must not be in conflict with each other or with other reading moves (determinism)

Multi-Stack machine

M is called a **multi-stack machine**, and can be viewed as a generalization of the deterministic PDA



Multi-Stack machine

In a multi-stack machine with k stacks, a **transition rule** has the form

$$\delta(q, a, X_1, X_2, \ldots, X_k) = (p, \gamma_1, \gamma_2, \ldots, \gamma_k)$$

In words, when the machine is in state q and reads input symbol $a \in \Sigma \cup \{\epsilon\}$, and with X_i on top of the i-th stack, $1 \leqslant i \leqslant k$, it moves to state p and replaces each X_i with γ_i

Multi-Stack machine

Theorem If a language L is accepted by a TM, then L is accepted by a multi-stack machine with two stacks

Proof (sketch) Let $L = L(M)$ for a TM M. We construct a machine S with two stacks, having special symbols used as **markers** at the bottom of the stack

The basic idea is to

- simulate the tape to the left of the current position with the first stack
- simulate the tape starting from the current position and extending to the right with the second stack

Multi-Stack machine

The transition rules of S implement the following strategy

- copy the input $w\$$ into the first stack
- move the contents of the first stack into the second stack
- if M overwrites X with Y and moves to the right, S pushes Y on the first stack and pops X from the second stack
- if M overwrites X with Y and moves to the left, S pops the symbol Z from the first stack and replaces X with ZY in the second stack
- in addition, S employs some special moves to handle the case where M is located at the end points of the tape (one of the two stacks contains the bottom marker)
- S accepts whenever M accepts □

TM and computer

Theorem If a language L is accepted by a modern computer, then L is accepted by a TM

Proof Omitted

Summary of decision problem for CFLs

We have presented **efficient** algorithms for the solution of the following decision problems for CFLs

- given a CFG G, test whether $L(G) \neq \emptyset$
- given a string w, test whether $w \in L(G)$ for a fixed CFG G

Undecidable decision problem for CFLs

In the next chapters we will develop a mathematical theory to prove the existence of decision problems that **no algorithm can solve**

Let us now anticipate some of these problems, concerning CFLs

- given a CFG G, test whether G is ambiguous
- given a representation for a CFL L, test whether L is inherently ambiguous
- given a representation for two CFLs L_1 and L_2, test whether the intersection $L_1 \cap L_2$ is empty
- given a representation for two CFLs L_1 and L_2, test whether $L_1 = L_2$
- given a representation for a CFL L defined over Σ, test whether $L = \Sigma^*$