1

I modelli (operazionali) non deterministici

- Solitamente si tende a pensare ad un algoritmo come una sequenza di operazioni *determinata*: in un certo stato e con certi ingressi non sussistono dubbi sulla "mossa" da eseguire
- Siamo sicuri che ciò sia sempre auspicabile?

Confrontiamo:

if x > y **then** max := x **else** max := y

con:

```
if    x >= y then max := x
      y >= x then max := y
fi
```

2

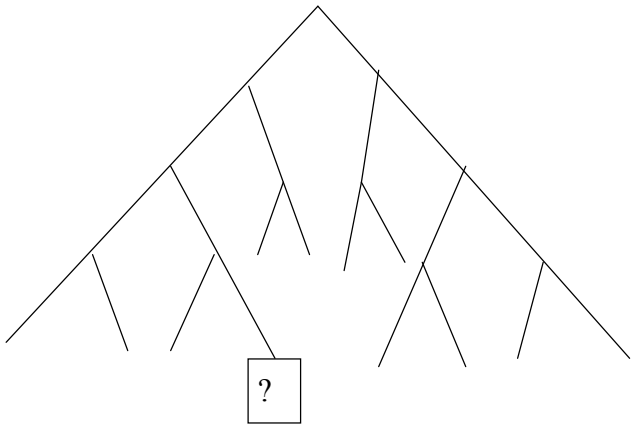
- E` solo una questione di eleganza?
- Pensiamo al costrutto **case** del Pascal & affini:
- perché non un

```
case
x = y        then S1
z > y +3     then S2
….           then …
endcase
```

?

3

Un'altra forma di nondeterminismo "nascosto": la ricerca "cieca"



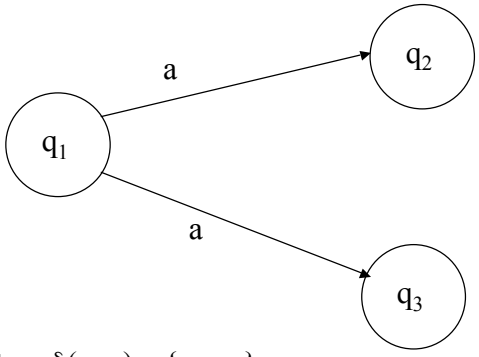
4

- In realtà i vari algoritmi di ricerca sono una "simulazione" di algoritmi "sostanzialmente nondeterministici":
- L'elemento cercato si trova nella radice dell'albero?
- Se sì OK. Altrimenti
  - Cerca nel sottoalbero di sinistra
    o
  - cerca nel sottoalbero di destra
- scelte o priorità tra le diverse strade sono spesso arbitrarie
- Se poi fossimo in grado di assegnare i due compiti in parallelo a due diverse macchine ---->
- Nondeterminismo come modello di computazione o almeno di progettazione di calcolo parallelo
  (Ad esempio Ada ed altri linguaggi concorrenti sfruttano il nondeterminismo)

5

Tra i tanti modelli nondeterministici (ND): versioni ND dei modelli già noti

- FSA ND (ne vedremo tra poco la "comodità")

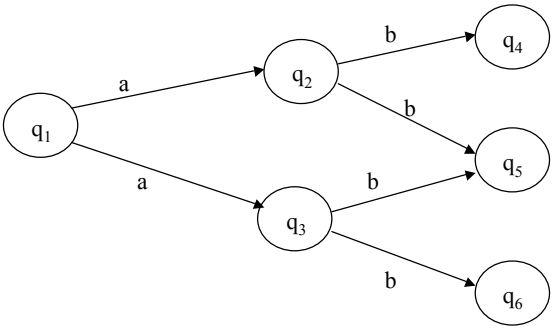


Formalmente: $\delta(q_1,a) = \{q_2, q_3\}$

$\delta : Q \times I \rightarrow \wp(Q)$

6

δ^* : formalizzazione della sequenza di mosse



$\delta(q_1,a) = \{q_2, q_3\}$, $\delta(q_2,b) = \{q_4, q_5\}$, $\delta(q_3,b) = \{q_6, q_5\}$

$\delta^*(q_1,ab) = \{q_4, q_5, q_6\}$

$$\delta^*(q,\varepsilon) = \{q\}$$

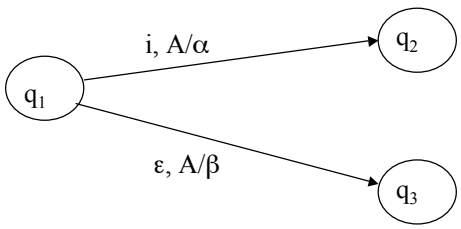
$$\delta^*(q, y \cdot i) = \bigcup_{q' \in \delta^*(q,y)} \delta(q', i)$$

Come accetta un FSA ND?

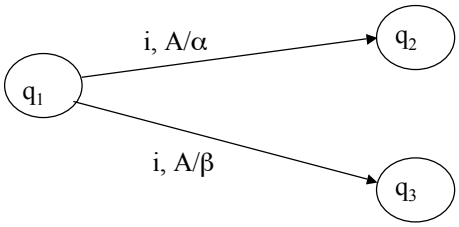
- $x \in \mathcal{L} \Leftrightarrow \delta^*(q_0, x) \cap F \neq \varnothing$
  - Tra i vari modi di funzionamento dell'automa è sufficiente che *almeno uno* di essi abbia successo per accettare la stringa di ingresso
- Sono possibili convenzioni diverse, per esempio: $x \in \mathcal{L} \Leftrightarrow \delta^*(q_0, x) \subseteq F$
  - Tra i vari modi di funzionamento dell'automa *tutti* devono successo per accettare la stringa di ingresso

Gli AP nondeterministici (APND)

- In realtà essi nascono ND di natura:



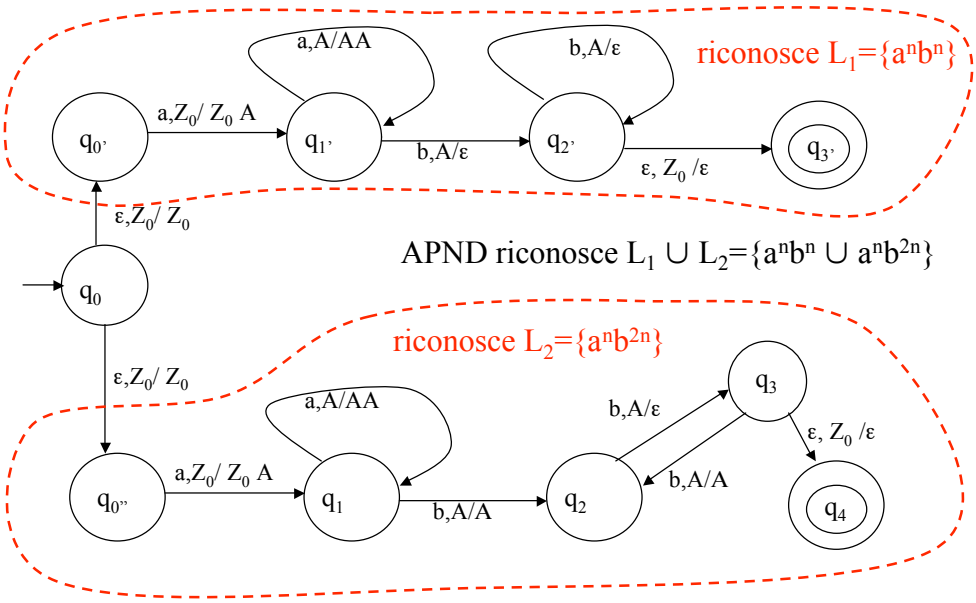
- Tanto vale rimuovere la restrizione del determinismo e generalizzare:



$$\delta : Q \times (I \cup \{\varepsilon\}) \times \Gamma \rightarrow \wp_F(Q \times \Gamma^*)$$

- Perché l'indice F?
- Al solito l'APND accetta x se *esiste* una sequenza
- c_0 |-*- <q, ε, γ>, q ∈ F
- |-- è non più univoca!

Un "banale" esempio



11

Alcune immediate ma significative conseguenze

- Gli APND possono riconoscere un linguaggio non riconoscibile dagli AP deterministici ---->
  - **sono più potenti**
- La costruzione precedente può essere facilmente generalizzata ottenendo una dimostrazione *costruttiva* (come altre precedenti) di chiusura rispetto all'unione degli APND
  - proprietà non sussistente per gli AP deterministici
- La chiusura rispetto all'intersezione invece continua a non sussistere ($\{a^n b^n c^n\} = \{a^n b^n c^*\} \cap \{a^* b^n c^n\}$ non è riconoscibile mediante una pila, neanche in modo ND)
  - I due controesempi precedenti $\{a^n b^n c^n\}$ e $\{a^n b^n\} \cup \{a^n b^{2n}\}$ non sono poi così simili tra loro …

12

- Se una famiglia di linguaggi è chiusa rispetto all'unione e non rispetto all'intersezione

  $\Rightarrow$

  non può essere chiusa rispetto al complemento

  (perché?)
- Ciò mette in evidenza il profondo cambiamento causato dal nondeterminismo rispetto alla complementazione di un problema - in generale -:
  se il modo di funzionamento della macchina è univoco e se la sua computazione giunge al termine è sufficiente scambiare la risposta positiva con quella negativa per ottenere la soluzione di un "problema complemento" (ad esempio *presenza* invece di *assenza* di errori in un programma)

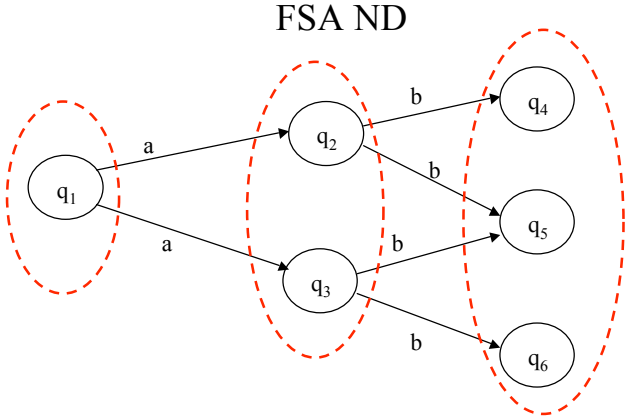
13

- Nel caso degli APND pur essendo possibile, come per gli APD, far sì che una computazione giunga sempre al termine, potrebbero darsi due computazioni
  - c_o |-*- <q_1, ε, γ_1>
  - c_o |-*- <q_2, ε, γ_2>
  - $q_1 \in F$, $q_2 \notin F$
- In questo caso x è accettata
- Però se scambio F con Q-F, x continua ad essere accettata: nell'ambito del nondeterninismo scambiare il sì con il no non funziona!

- E gli altri tipi di automi?

14

FSA ND



Partendo da q_1 e leggendo ab l'automa si trova in uno stato che appartiene all'insieme $\{q_4,q_5,q_6\}$

Chiamiamo nuovamente "stato" l'insieme dei possibili stati in cui si può trovare l'automa ND durante il suo funzionamento.

Sistematizzando ...

15

- Dato un FSA ND ne costruisco *automaticamente* uno equivalente determistico ---->
- gli automi FSA ND non sono più potenti dei loro fratelli deterministici (diversamente dagli AP)

  (e allora a che cosa servono?)
- $A_{ND} = <Q_N, I, \delta_N, q_{0N}, F_N>$
- $A_D = <Q_D, I, \delta_D, q_{0D}, F_D>$
  - $Q_D = \wp(Q_N)$
  - $q_{0D} = \{q_{0N}\}$
  - $\delta_D(q_D, i) = \bigcup_{q_N \in q_D} \delta(q_N, i)$
  - $F_D = \{\overline{Q} \subseteq Q \mid \overline{Q} \cap F_N \neq \varnothing\}$

16

- E` bensì vero che, per ogni automa FS ND ne posso trovare (e *costruire*) uno equivalente deterministico
- Ciò non significa che sia superfluo usare gli FSA ND:
  - Può essere più facile "progettare" un AND e poi ricavarne automaticamente uno deterministico, risparmiandosi la fatica di costruirlo noi stessi deterministico fin da subito (ne vedremo un'applicazione tra breve)
  - Da un AND a 5 stati (ad esempio), ne ricavo, nel caso pessimo, uno con 2^5 stati!
- Resta da esaminare la MT ...

17

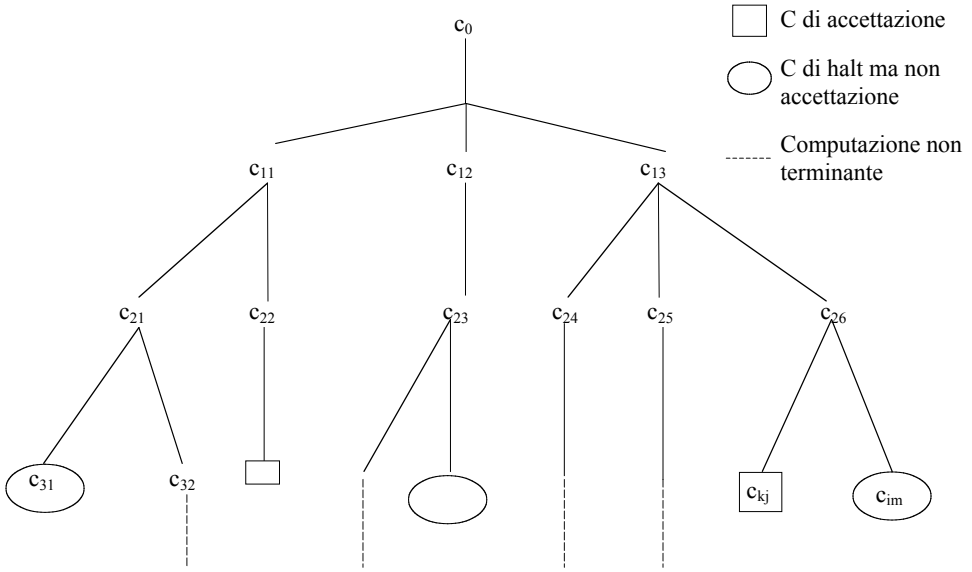
Le MT nondeterministiche

$$< \delta, [\eta] >: Q \times I \times \Gamma^k \rightarrow \wp(Q \times \Gamma^k \times \{R, L, S\}^{k+1} [\times O \times \{R, S\}])$$

- E' necessario l'indice F?
- Configurazioni, transizioni, sequenze di transizioni e accettazione sono definite come al solito
- Il nondeterminismo aumenta la potenza delle MT?

18

Albero delle computazioni



19

- x è accettata da una MT ND se e solo se esiste una computazione della MND che termina in uno stato di accettazione
- può una MT deterministica stabilire se una sua "sorella" ND accetta x, ossia accettare a sua volta x se e solo se la MND la accetta?
- Si tratta di percorrere o "visitare" l'albero delle computazioni ND per stabilire se esiste in esso un cammino che termina in uno stato di accettazione
- Questo è un (quasi) normale e ben noto problema di visita di alberi, per il quale esistono classici algoritmi di visita
- Il problema è perciò ridotto ad implementare un algoritmo di visita di alberi mediante MT: compito noioso ma sicuramente fattibile … a meno del "quasi" di cui sopra ...

20

- Tutto facile se l'abero delle computazioni è finito
- Però potrebbe darsi il caso che alcuni cammini dell'albero siano infiniti (descrivono computazioni che non terminano)
- In tal caso, un algoritmo di visita depth-first (ad esempio, in preordine sinistro) potrebbe "infilarsi in un cammino infinito" senza scoprire che in un altro punto dell'albero ne esiste uno finito che porta all'accettazione.
- Il problema è però facilmente risolvibile adottando ad esempio un algoritmo di visita di tipo breadth-first (che usa una struttura a coda invece di una a pila per accumulare i vari nodi da esaminare).

Conclusioni

- Nondeterminismo: utile astrazione per descrivere problemi/algoritmi di ricerca; situazioni in cui non esistono elementi di scelta, o sono tra loro indifferenti; computazioni parallele
- In generale non aumenta la potenza di calcolo, almeno nel caso delle MT (che sono l'automa più potente tra quelli visti finora) però può fornire descrizioni più compatte
- Aumenta la potenza degli automi a pila
- Può essere applicato a diversi modelli di calcolo (praticamente a tutti); in alcuni casi sono stati definiti modelli "intrinsecamete nondeterministici" (inventati apposta per descrivere fenomeni nondeterministici)
- Per semplicità ci siamo concentrati soprattutto su riconoscitori nondeterministici ma il concetto può essere esteso anche agli automi traduttori.
- NB: il concetto di ND non è da confondersi con un fenomeno *stocastico* (esistono modelli stocastici -e.g. catene di Markov- che sono ben diversi da quelli nondeterministici)

Grammatiche

- Gli automi sono un modello riconoscitivo/traduttivo/*elaborativo* (di linguaggi):
  - essi "ricevono" una stringa nel loro ingresso e la elaborano in vari modi
- Passiamo ora ad esaminare un modello *generativo*:
  una grammatica produce o *genera* stringhe (di un linguaggio)
- Concetto generale di *grammatica* o *sintassi* (matematicamente, alfabeto è sinonimo di vocabolario e grammatica è sinonimo di sintassi):
  insieme di regole per costruire frasi di un linguaggio (stringhe)
  - si applica a qualsiasi nozione di linguaggio nel senso più lato.
- In modo sostanzialmente simile ai normali meccanismi linguistici una grammatica formale genera stringhe di un linguaggio attraverso un processo di *riscrittura*:

23

- “Una frase è costituita da un soggetto seguito da un predicato
  Un soggetto a sua volta può essere un sostantivo, oppure un pronome, oppure ….
  Un predicato può essere costituito da un verbo seguito da un complemento
  …”
- “Un programma è costituito da una parte dichiarativa e da una parte esecutiva
  La parte dichiarativa …
  La parte esecutiva è costituita da una sequenza di istruzioni
  Un’istruzione può essere semplice o composta
  …”

24

- “Un messaggio di posta elettronica è costituito da una testata e da un corpo
  La testata contiene indirizzo, …”
- …

- In generale questo tipo di regole linguistiche descrive un “oggetto principale” (libro, programma, messaggio, protocollo, ….) come una sequenza di “oggetti componenti” (soggetti, testate, parti dichiarative, …). Ognuno di questi viene poi “raffinato” rimpiazzandoli con altri oggetti più dettagliati e così via finche` non si giunge ad una sequenza di elementi base (bit, caratteri, …)
  Le varie riscritture possono presentare delle alternative: un soggetto può essere un nome o un pronome o altro,
  un’istruzione può essere di assegnamento o di I/O, ...

25

La definizione formale di grammatica

- $G = <V_N, V_T, P, S>$
  - V_N : alfabeto o vocabolario *nonterminale*
  - V_T : alfabeto o vocabolario *terminale*
  - $V = V_N \cup V_T$
  - $S \in V_N$: elemento particolare di V_N detto *assioma o simbolo iniziale*
  - $P \subseteq (V_N)^+ \times V^*$: insieme delle regole di riscrittura, o produzioni sintattiche.

    per comodità scriveremo $P = \{<\alpha, \beta>\}$ come $P = \{\alpha \rightarrow \beta\}$

26

Esempio

- $V_N = \{S, A, B, C, D\}$
- $V_T = \{a,b,c\}$
- S
- $P = \{S \rightarrow AB, BA \rightarrow cCd, CBS \rightarrow ab, A \rightarrow \varepsilon\}$

Relazione di derivazione immediata

- Definizione:

  $\alpha \Rightarrow \beta$ con $\alpha \in V^+$, $\beta \in V^*$

  $\leftrightarrow$

  $\alpha = \alpha_1\alpha_2\alpha_3 \quad , \quad \beta = \alpha_1\beta_2\alpha_3 \quad \wedge\ \alpha_2 \rightarrow \beta_2 \in P$

  α_2 si riscrive come β_2 nel contesto $<\alpha_1, \alpha_3>$

- Rispetto alla grammatica precedente:

  aaBAS $\Rightarrow$ aacCdS

- Definiamo poi come al solito la chiusura riflessiva e transitiva della relazione di derivazione immediata $\Rightarrow$: $\overset{*}{\Rightarrow}$

Linguaggio generato da una grammatica

$$L(G) = \{x \mid x \in V_T^* \wedge S \overset{*}{\Rightarrow} x\}$$

Consiste di tutte le stringhe costituite da soli simboli terminali derivabili (in numero qualsiasi di passi) da S

29

Primo esempio

- $G_1 = <\{S, A, B\}, \{a, b, 0\}, P, S>$
  $P = \{S \rightarrow aA, A \rightarrow aS, S \rightarrow bB, B \rightarrow bS, S \rightarrow 0\}$
- Alcune derivazioni:
  - $S \Rightarrow 0$
  - $S \Rightarrow aA \Rightarrow aaS \Rightarrow aa0$
  - $S \Rightarrow bB \Rightarrow bbS \Rightarrow bb0$
  - $S \Rightarrow aA \Rightarrow aaS \Rightarrow aabB \Rightarrow aabbS \Rightarrow aabb0$
- Mediante una facile generalizzazione:
  - $L(G_1) = \{aa, bb\}^*.0$

30

Secondo esempio

- $G_2 = <\{S\}, \{a, b\}, P, S>$
  $P = \{S \rightarrow aSb \mid ab\}$ (abbreviazione per $S \rightarrow aSb$, $S \rightarrow ab$)
- Alcune derivazioni:
  - $S \Rightarrow ab$
  - $S \Rightarrow aSb \Rightarrow aabb$
  - $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$
- Mediante una facile generalizzazione:
  - $L(G_2) = \{a^n b^n \mid n \geq 1\}$
- Se sostituiamo $S \rightarrow ab$ con $S \rightarrow \varepsilon$ otteniamo:
  - $L(G_2) = \{a^n b^n \mid n \geq 0\}$

31

Terzo esempio

{S→aACD, A→aAC, A→ε, B→b, CD→BDc, CB→BC, D→ε}

S ⇒ aACD ⇒ aCD ⇒ aBDc ⇒ aBc ⇒ abc

S ⇒ aACD ⇒ aaACCD ⇒ aaCCD ⇒ aaCC↓

S ⇒ aACD ⇒ aaACCD ⇒ aaCCD ⇒ aaCBDc ⇒
aaBCDc ⇒ aabCDc ⇒ aabBDcc ⇒ aabbDcc ⇒ aabbcc

S ⇒ aACD ⇒ aaACCD ⇒ aaaACCCD ⇒ aaaCCCD ⇒
aaaCCBDc ⇒ aaaCCbDc ⇒ aaaCCbc↓

...

32

Alcune domande "naturali"

- Quale utilità pratica delle grammatiche (oltre ai "divertimenti" con $\{a^n b^n\}$?)
- Quali linguaggi si possono ottenere con le grammatiche?
- Che relazioni esistono tra grammatiche e automi (o meglio tra i linguaggi *generati* dalle grammatiche e i linguaggi *riconosciuti* dagli automi)?

33

Alcune risposte

- Definizione della sintassi dei linguaggi di programmazione
- Applicazioni duali rispetto a quelle degli automi
- Esempio più semplice: la *compilazione* dei linguaggi (non solo di programmazione): la grammatica definisce il linguaggio; l'automa lo riconosce e traduce.
- In modo un po' più sistematico:

34

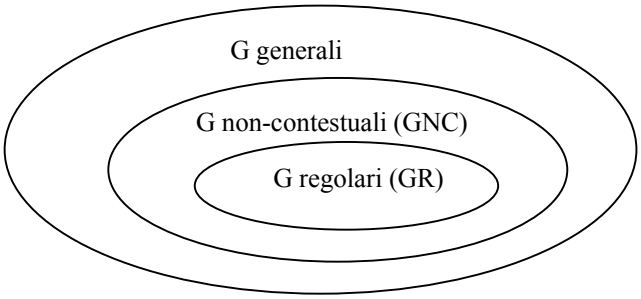
Classi di grammatiche

- Grammatiche non-contestuali (context-free):
  - $\forall\ \alpha \rightarrow \beta \in P$, $|\alpha| = 1$, cioè α è un elemento di V_N.
  - Non-contestuale perché la riscrittura di α non dipende dal contesto in cui si trova.
  - Sono di fatto la stessa cosa della BNF usata per definire la sintassi dei linguaggi di programmazione (quindi vanno bene per definire certi meccanismi sintattici tipici dei linguaggi di programmazione e naturali … ma non tutti)
  - Le G_1 e G_2 precedenti sono non-contestuali, non così la G_3.

35

- Grammatiche regolari:
  - $\forall\ \alpha \rightarrow \beta \in P, |\alpha| = 1, \beta \in\ V_T \cdot V_N \cup V_T \cup \varepsilon$
  - Le grammatiche regolari sono anche non-contestuali, ma non viceversa
  - La G_1 precedente è regolare, ma non la G_2.

Relazioni tra grammatiche e linguaggi



36

Se ne deduce immediatamente che:

$\mathcal{LR} \subseteq \mathcal{LNC} \subseteq \mathcal{LGen}$

Ma i contenimenti sono stretti?

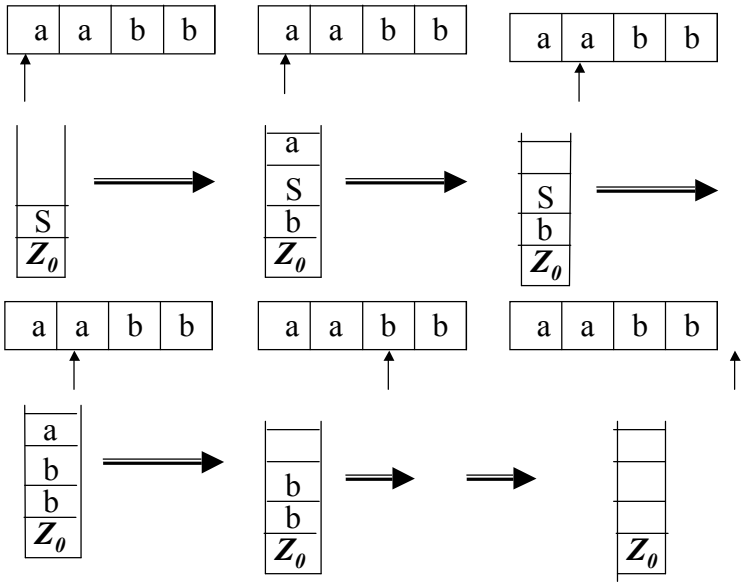
La risposta dal confronto con gli automi

Relazioni tra grammatiche e automi (senza troppe sorprese)

- GR equivalenti agli FSA
- Dato un FSA A:
  - poniamo $V_N = Q$, $V_T = I$, $S = <q_0>$, e per ogni $\delta(q, i) = q'$, poniamo $<q> \rightarrow i<q'>$ inoltre se $q' \in F$, aggiungiamo $<q> \rightarrow i$
  - E' intuitivo (facile induzione) che $\delta^*(q, x) = q'$ se e solo se $<q> \Rightarrow^* x<q'>$
- Viceversa, data una GR :
  - poniamo $Q = V_N \cup \{q_F\}$, $I = V_T$, $<q_0> = S$, $F = \{q_F\}$ e, per ogni $A \rightarrow bC$ poniamo $\delta(A,b) = C$ per ogni $A \rightarrow b$ poniamo $\delta(A,b) = q_F$
  - **NB: L'FSA così ottenuto è nondeterministico: molto più comodo!**

- GNC equivalenti a AP (ND!)
  - giustificazione intuitiva (senza dimostrazione: la dimostrazione costituisce il "cuore" della costruzione di un compilatore):

$S \Rightarrow aSb \Rightarrow aabb$



Ggen equivalenti alle MT

- Data G costruiamo (a grandi linee) una MT *ND (!)*, M, che accetti L(G):
  - M ha un nastro di memoria
  - La stringa di ingresso x si trova nel nastro di ingresso
  - Il nastro di memoria viene inizializzato con S (meglio: Z_0S)
  - Il nastro di memoria (in generale esso conterrà una stringa α, $\alpha \in V^*$) viene scandito alla ricerca di una parte sinistra di qualche produzione di P
  - Quando se ne trova una -*non necessariamente la prima, operando una scelta ND*- essa viene sostituita dalla corrispondente parte destra (se ve n'è più d'una si opera ancora nondeterministicamente)

41

In tal modo:

$$\alpha \Rightarrow \beta \quad \leftrightarrow \quad c = < q_s , Z_0\alpha > \mid -^* - \; < q_s , Z_0\beta >$$

Se e quando sul nastro si trova una stringa y $\in V_T^*$, la si confronta con x. Se coincidono x viene accettata, altrimenti questa particolare sequenza di mosse non porta all'accettazione.

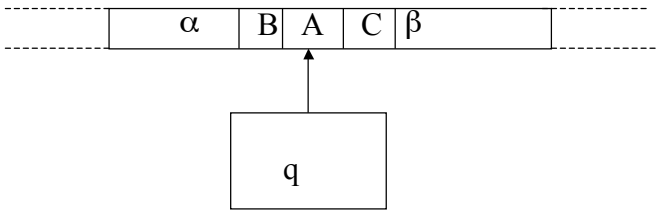
Si noti che:

- L'uso del ND facilita molto la costruzione ma non è essenziale
- E' invece essenziale -e, vedremo, ineluttabile- il fatto che, se x $\notin$ L(G), M potrebbe "tentare infinite strade", alcune delle quali anche non terminanti, senza poter concludere che x $\in$ L(G) (giustamente), ma *neanche il contrario*.
  Infatti la definizione di accettazione richiede che M giunga in una configurazione di accettazione se e solo se x $\in$ L, ma non richiede che M termini la computazione (in uno stato negativo) se x $\notin$ L
- Tornano in ballo il problema-complemento e l'asimmetria tra risolvere un problema in maniera positiva o in maniera negativa.

42

- Data M (per comodità e senza perdere generalità, a nastro singolo) costruiamo (a grandi linee) una G, che generi L(M):
  - In primo luogo G genera *tutte* le stringhe del tipo x\$X, x $\in V_T^*$, X essendo una "copia di x" costituita da caratteri nonterminali (ad esempio, per x = aba, x\$X = aba\$ABA)
    L'obiettivo è di ricavare da x\$X una derivazione x\$X $\Rightarrow^*$ x se e solo se x è accettata da M.
  - L'idea base è di simulare ogni mossa di M mediante una derivazione immediata di G:

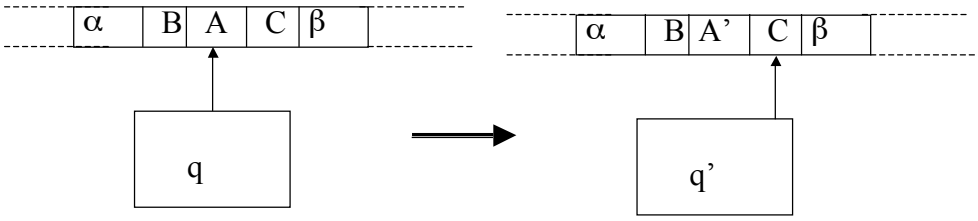
- Rappresento la configurazione



- Mediante la stringa (i casi particolari sono lasciati in esercizio): $\$\alpha BqAC\beta$
- Se è definita:
  - $\delta(q,A) = <q', A', R)$ si definisce in G la produzione $qA \rightarrow A'q'$
  - $\delta(q,A) = <q', A', S)$ si definisce in G la produzione $qA \rightarrow q' A'$
  - $\delta(q,A) = <q', A', L)$ si definisce in G la produzione $BqA \rightarrow q' BA'$

  $\forall$ B dell'alfabeto di M (si ricordi che M è a nastro singolo e quindi ha un solo alfabeto per ingresso, memoria e uscita)

- In tal modo:



- Se e solo se: $x\$\alpha BqAC\beta \Rightarrow x\$\alpha BA'q'C\beta$, ecc.
- Aggiungo infine produzioni che permettano a G di derivare da $x\$\alpha Bq_FAC\beta$ la sola x se - e solo se- M giunge a una configurazione di accettazione ($\alpha Bq_FAC\beta$), cancellando tutto ciò che si trova a destra di \$, \$ compreso.