Linguaggi e Traduttori: Analisi lessicale

Armando Tacchella

Sistemi e Tecnologie per il Ragionamento Automatico (STAR-Lab)
Dipartimento di Informatica Sistemistica e Telematica (DIST)
Università di Genova

A.A. 2006/2007 - Primo semestre

Outline

Introduzione e motivazioni

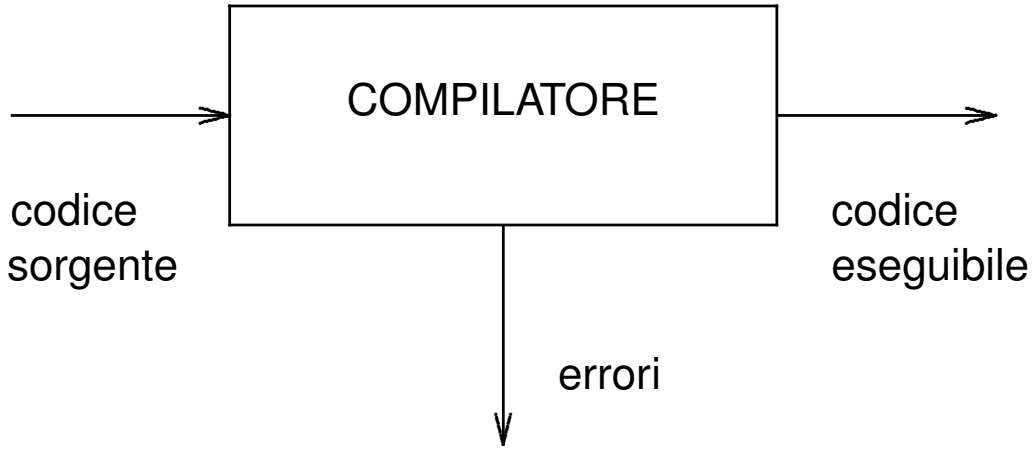
Strumenti formali

- Espressioni regolari (RE)
- Automi a stati finiti (DFA)

Relazione tra RE e DFA

- Automi a stati finiti non deterministici (NFA)
- Costruzione di Thomson: da RE a NFA
- Conversione da NFA a DFA

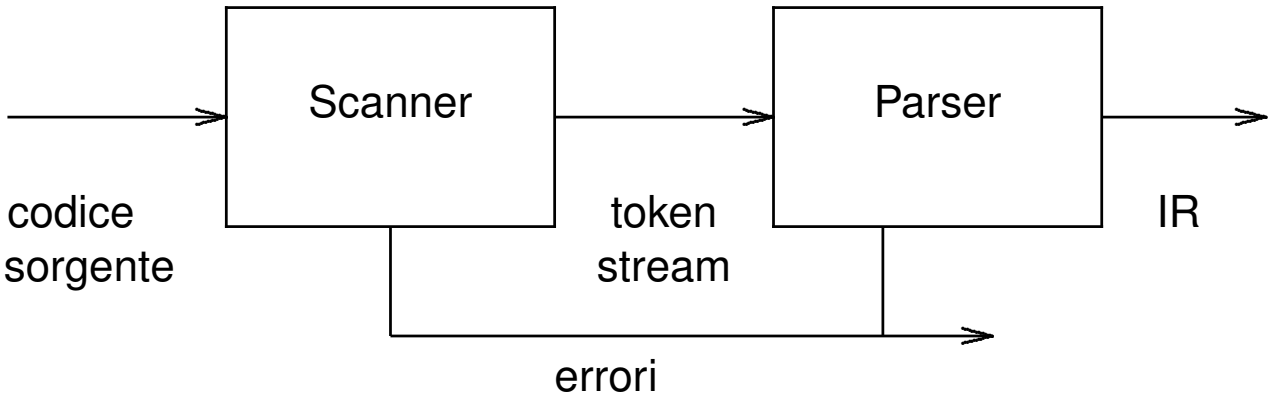
Il Front End



Lo scopo del Front End è quello di analizzare il codice sorgente

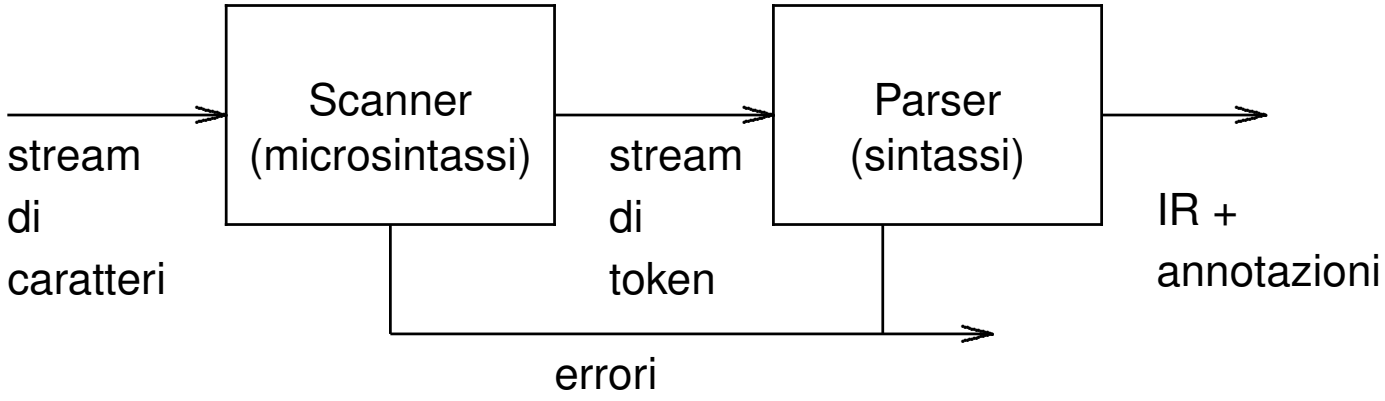
- Eseguire un test di appartenenza:
  programma $\in$ linguaggio?
- Verificare se il programma è semanticamente ben fondato
- Costruire la rappresentazione intermedia (IR)

Realizzazione del Front End



- Specifica della sintassi con una notazione formale
  - espressioni regolari per l'analisi lessicale
  - grammatiche context-free per l'analisi sintattica
- Sintesi automatica dei riconoscitori lessical e sintattici

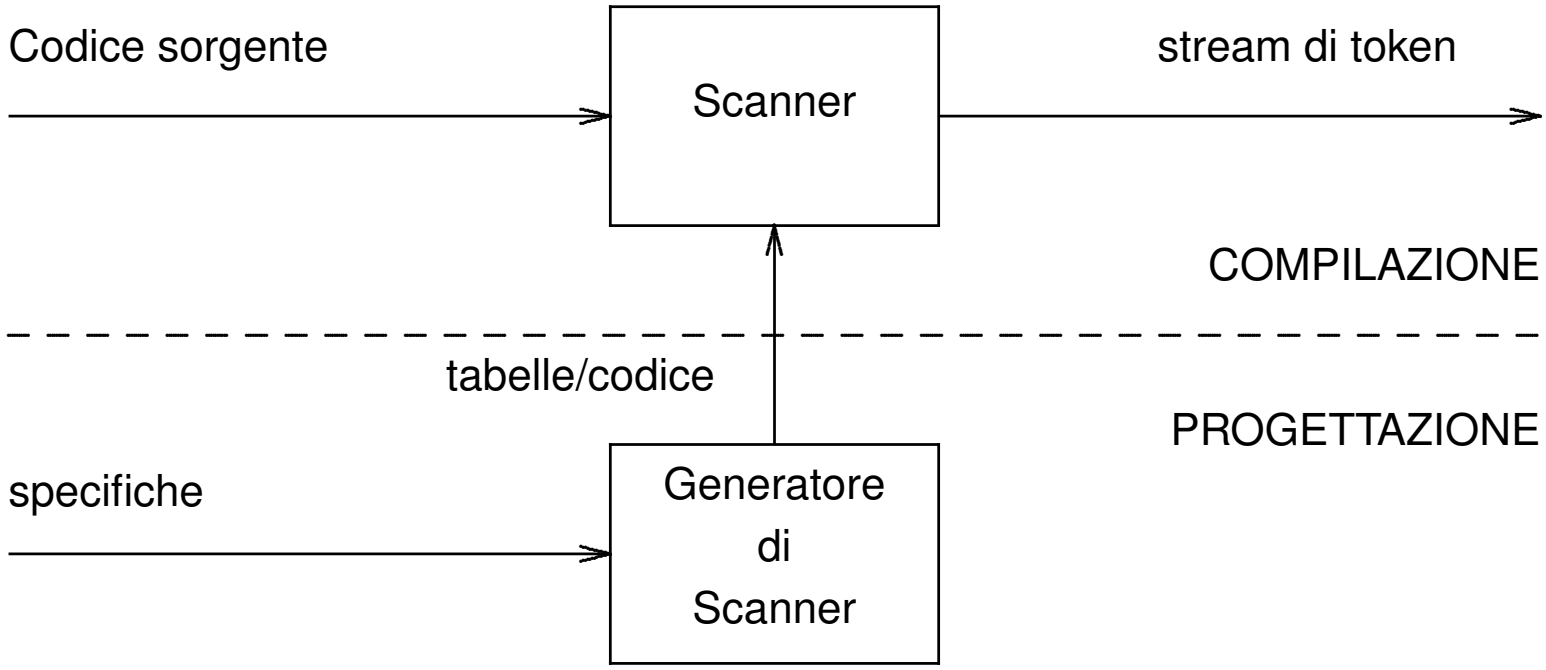
Microsintassi e sintassi



token = <parte del discorso, lessema>

- L'analizzatore lessicale identifica i token
- L'analizzatore sintattico controlla il rispetto della grammatica
- L'analisi sintattica è più complessa
- La separazione semplifica l'implementazione

Visione d'insieme



- Le specifiche per il generatore sono espressioni regolari
- L'analizzatore lessicale è un'implementazione degli automi a stati finiti che corrispondono alle specifiche

Outline

Operazioni su insiemi (Ripasso?)

Operazione	Definizione
Unione di L ed M $L \cup M$	$L \cup M = \{s \mid s \in L \text{ o } s \in M\}$
Concatenazione di L ed M LM	$LM = \{st \mid s \in L \text{ e } t \in M\}$
Chiusura di L L^*	$L^* = \bigcup_{0 \leq i} L^i$

Espressioni regolari (RE)

Dato un alfabeto (insieme di simboli) Σ

- ϵ è una RE che denota l'insieme vuoto $\{\}$
- Se $s \in \Sigma$ allora s è una RE che denota $\{s\}$
- Se x e y sono RE che denotano $L(x)$ ed $L(y)$ allora
  - $x|y$ è una RE che denota $L(x) \cup L(y)$
  - xy è una RE che denota $L(x)L(y)$
  - x^* è una RE che denota $L(x)^*$

Utilizzo delle RE

Dato un generico alfabeto Σ utilizzando le RE possiamo specificare

- singole parole con la concatenazione
- insiemi di parole con l'unione
- insiemi di parole aventi la stessa struttura con la chiusura

Ad esempio, dato $\Sigma = \{\text{caratteri ASCII}\}$ possiamo scrivere:

- una RE per ogni singola parola chiave del C: *if*, *then*, *void*, *int*, ecc.
- una RE che racchiude tutte le parole chiave del C: $if|then|void|int|\ldots$
- una RE che riconosce identificatori validi in C: date $x = a|b|\ldots|z|A|B|\ldots|\_$ e $y = x|0|\ldots|9$, l'espressione xy^* definisce (infiniti) identificatori

Outline

Automi a stati finiti deterministici (DFA)

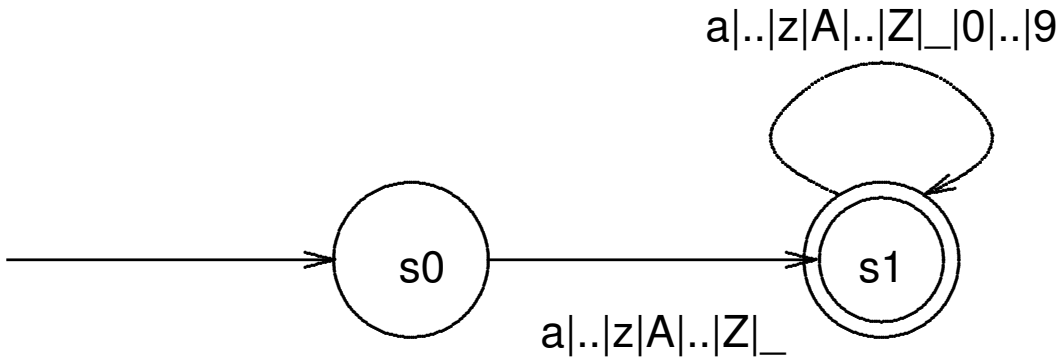
Un DFA è una quintupla $\langle S, \Sigma, \delta, s_0, S_F \rangle$, dove:

- S è un insieme di finito stati (o configurazioni)
- Σ è un insieme finito di simboli
- δ è una funzione $\delta : S \times \Sigma \to S$
- s_0 è lo stato di partenza $s_0 \in S$
- S_F è l'insieme degli stati finali $S_F \subseteq S$

Intuitivamente, data una sequenza di simboli w:

- l'automa si "accende" nello stato s_0
- dato lo stato corrente s, l'automa "consuma" un carattere c di w, ed effettua una transizione nello stato $\delta(s, c)$
- quando w è stata "consumata", se lo stato corrente appartiene a S_F allora w è "accettata" dall'automa, altrimenti è "rifiutata"

Utilizzo dei DFA



Un DFA per riconoscere gli identificatori validi del C.
Formalmente:

- $\Sigma = \{a, \ldots, z, A, \ldots, Z, \_, 0, \ldots, 9\}$
- $S = \{s_0, s_1\}$
- δ è la funzione rappresentata dal diagramma
- $S_F = \{s_1\}$

Identificatori accettati: “_”, “a21”, “pippo”
Identificatori rifiutati: “15”, “7a”

Sintesi dei DFA

Schema del riconoscitore

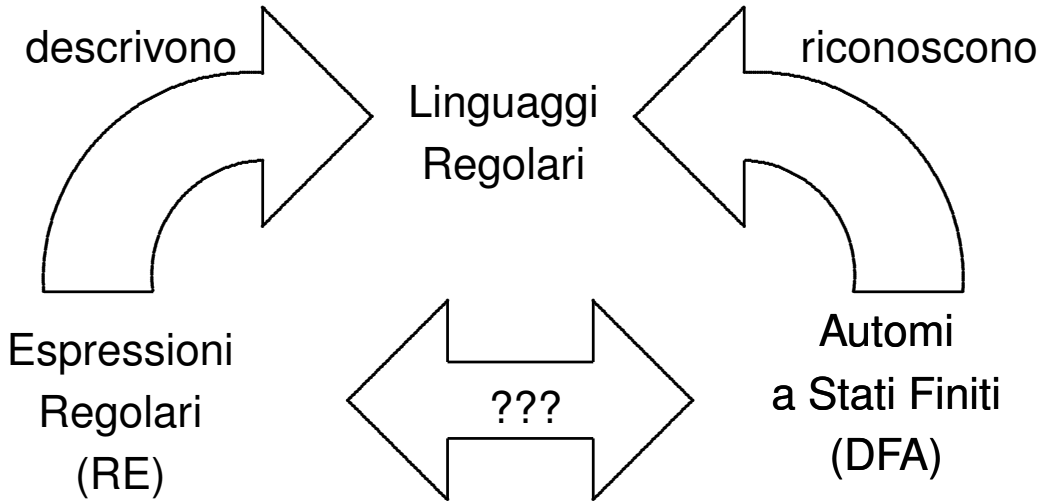
```
ISIDENTIFIER(inputStream) : bool
 1 char ← GETCHAR(inputStream)
 2 state ← s0
 3 while (char ≠ EOF)
 4     state ← δ(state, char)
 5     char ← GETCHAR(inputStream)
 6 if state ∈ SF
 7     then return TRUE
 8     else return FALSE
```

Funzione di transizione

δ	$a\|..\|z\|$ $A\|..\|Z\|\_$	$0\|..\|9$	altro
s_0	s_1	err	err
s_1	s_1	s_1	err

Outline

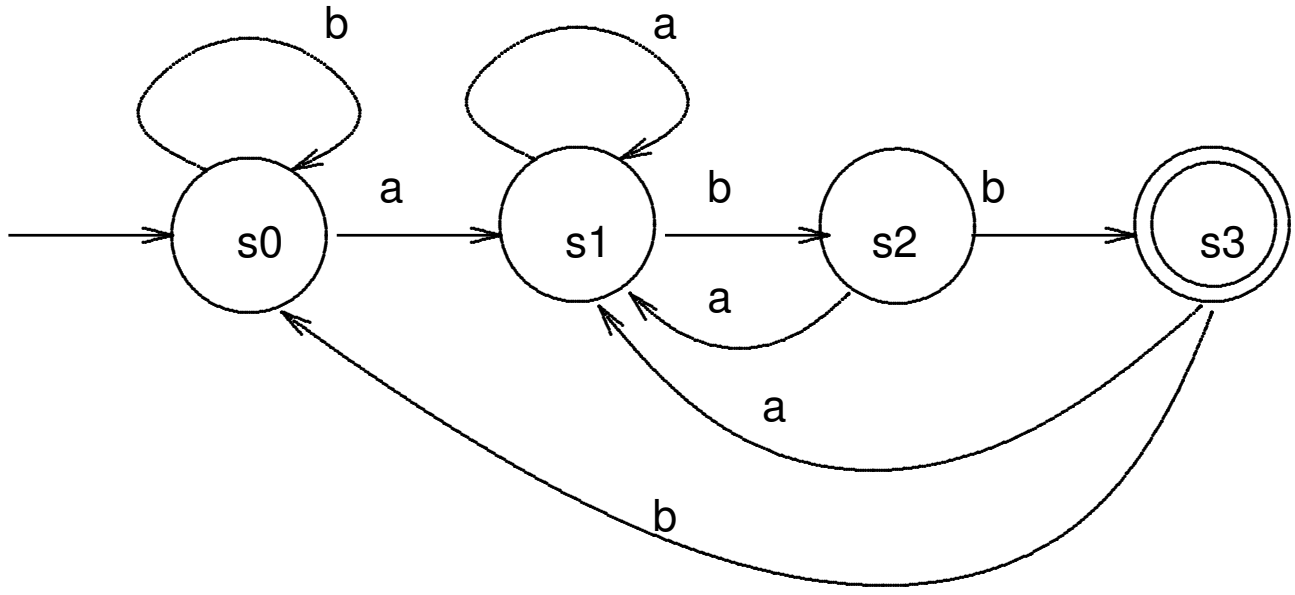
Visione d'insieme



- Le RE descrivono (tutti e soli) i linguaggi regolari
- I DFA riconoscono (tutti e soli) i linguaggi regolari
- Quale corrispondenza tra RE e DFA?
- Chiave per la sintesi automatica di scanner

Da RE a DFA: problematiche

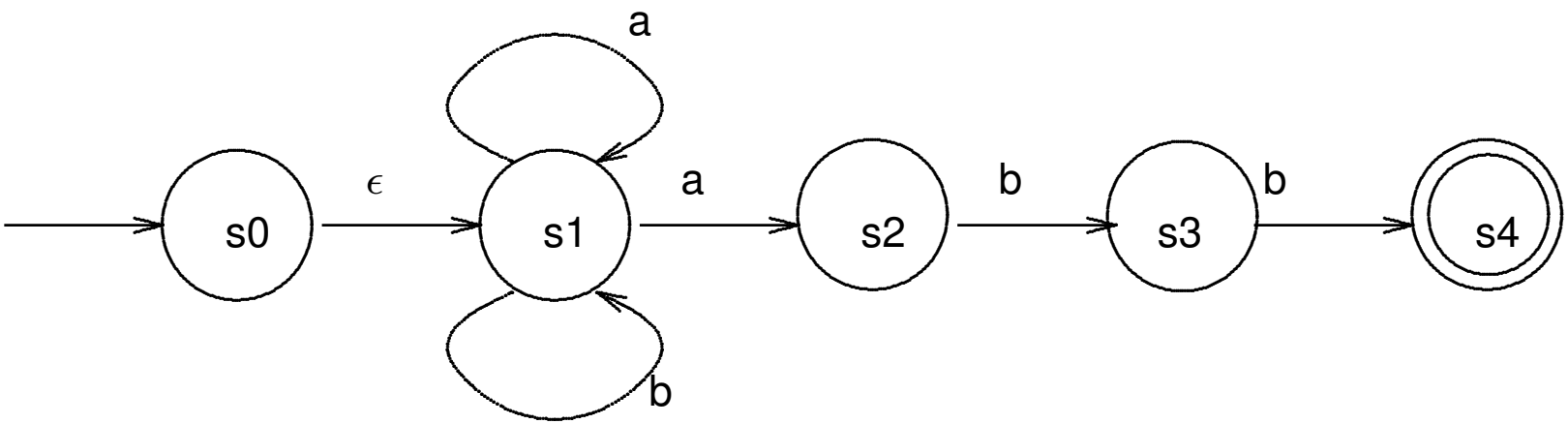
La RE $(a|b)^*abb$ corrisponde al DFA:



- La corrispondenza non è immediata
- La conversione diretta è difficilmente automatizzabile

Da RE a DFA: non determinismo

Un'altro automa per la RE $(a|b)^*abb$



Un riconoscitore intrinsecamente diverso:

- s_0 ha una transizione con ϵ
- s_1 ha due transizioni uscenti con a

Si tratta di un automa a stati finiti non deterministico

Automi a stati finiti non deterministici (NFA)

Un NFA accetta una stringa x se e solo se esiste una sequenza di transizioni da s_0 a $s \in S_F$ (ignorando ϵ) che corrisponde ai simboli di x.

- Le transizioni su ϵ non consumano l'input
- L'esecuzione di un NFA prevede che si "indovini" la mossa corretta ad ogni passo
  - viene sempre fatta la scelta giusta
  - se una sequenza di scelte giuste accetta x allora x fa parte del linguaggio riconosciuto dall'automa

Relazione tra NFA e DFA

- I DFA sono un caso particolare di NFA
  - non ammettono transizioni su ϵ
  - le transizioni sono descritte da una funzione
- Un DFA può essere simulato con un NFA (ovviamente!)
- Un NFA può essere simulato con un DFA (meno ovvio)
  - Simulare sovrapposizioni di stati
  - Nel caso pessimo, esplosione esponenziale (spazio)
  - Tempo di esecuzione comunque proporzionale a $|x|$

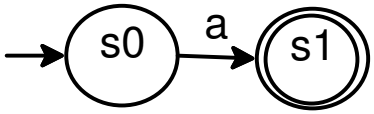
RE, NFA e DFA nei compilatori

Per generare automaticamente il codice dell'analizzatore lessicale:

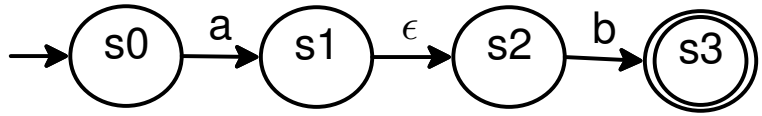
1. Scrivere le specifiche con le RE (a mano)
2. Convertire le RE in un NFA (Thomson's construction)
3. Convertire l'NFA in un DFA (Subset construction)
4. (Minimizzare il DFA risultante)
5. Generare il codice

Outline

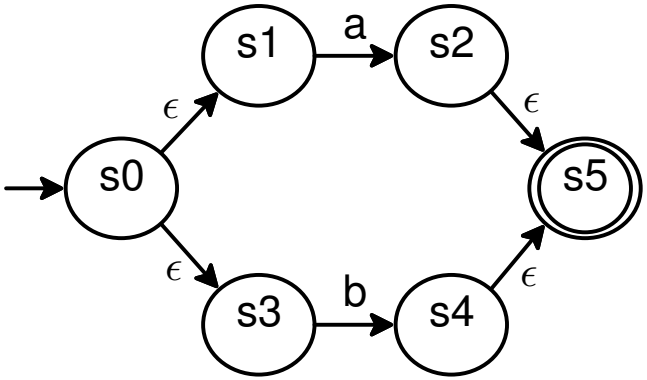
Elementi di base



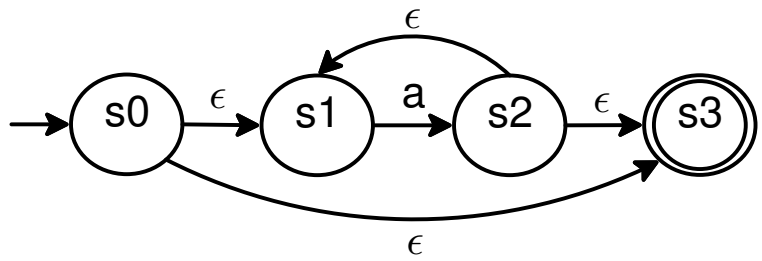
NFA per *a*



NFA per *ab*



NFA per *a*|*b*

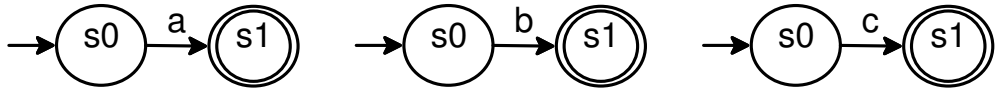


NFA per *a**

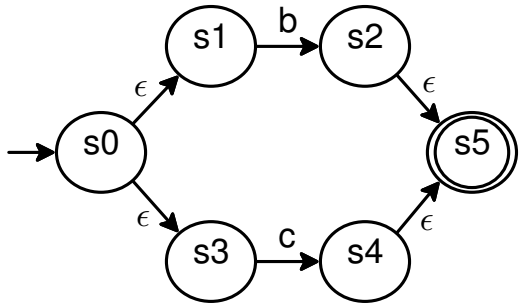
Esempio di costruzione di Thomson (1/2)

NFA per la RE $a(b|c)^*$

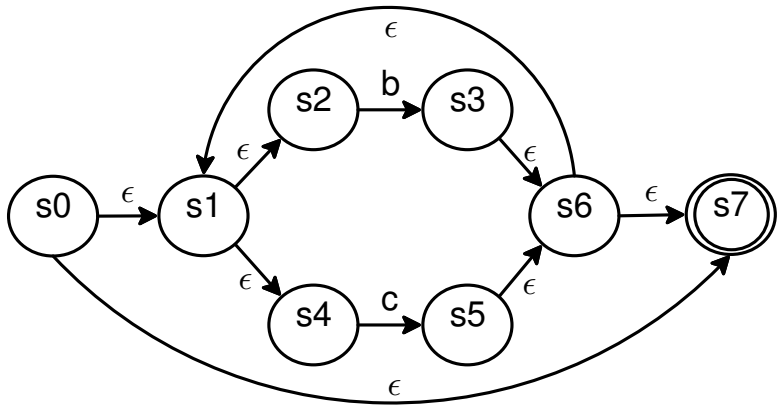
1. NFA per *a*, *b* e *c*



2. NFA per $b|c$

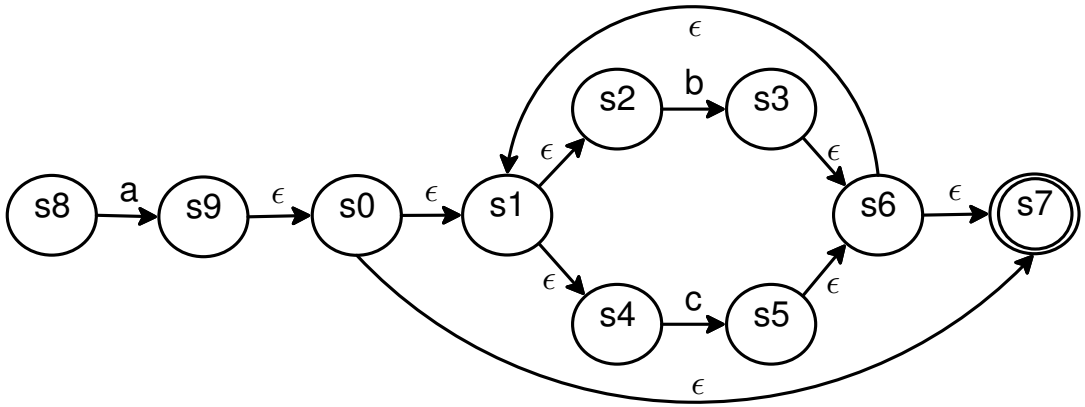


3. NFA per $(b|c)^*$

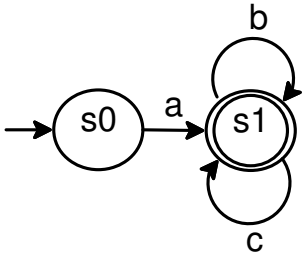


Esempio di costruzione di Thomson (2/2)

4. NFA per $a(b|c)^*$



È possibile costruire un automa molto più semplice a mano



però la costruzione di Thomson può essere automatizzata!

Outline

Subset Construction: NFA $\rightarrow$ DFA

- Si tratta di simulare un NFA
- Due funzioni chiave:
  - MOVE(s_i, a): insieme degli stati raggiungibili da s_i tramite a
  - ϵ-CLOSURE: insieme degli stati raggiungibili da s_i tramite ϵ
- Algoritmo:
  - Lo stato iniziale del DFA q_0 è la ϵ-CLOSURE dello stato iniziale del NFA s_0, ossia $q_0 = \epsilon$-CLOSURE(s_0)
  - Lo stato successivo è l'immagine di q_0, ossia la ϵ-CLOSURE del risultato di MOVE(s_i, α) per ogni $\alpha \in \Sigma$
  - Ripeto il procedimento finché nessun nuovo stato q_i viene creato

Pseudo-codice per la subset construction

$q_0 \leftarrow \epsilon\text{-CLOSURE}(\{s_0\})$
$Q \leftarrow \{q_0\}$
$W \leftarrow \{q_0\}$
while $(W \neq \emptyset)$
  selezionare e rimuovere q da W
  foreach $\alpha \in \Sigma$
    $t \leftarrow \epsilon\text{-CLOSURE}(\text{MOVE}(q,\alpha))$
    $\delta(q,\alpha) \leftarrow t$
    if $(t \notin S)$ **then**
      $S \leftarrow S \cup \{t\}$
      $W \leftarrow W \cup \{t\}$

L'algoritmo termina:

1. S non contiene duplicati
2. $2^{\{\#\text{ stati NFA}\}}$ è finito
3. il loop **while** aggiunge sempre elementi ad S (monotono)

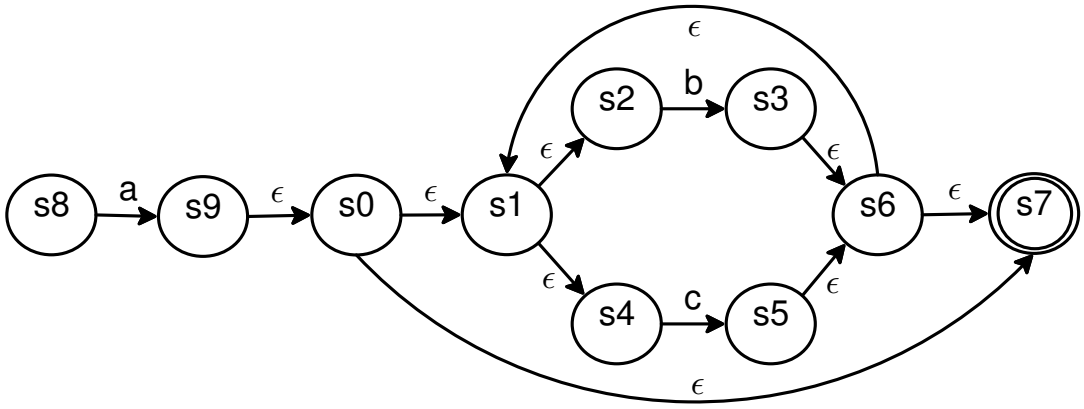
$\Rightarrow$ il loop termina!

S contiene tutti gli stati raggiungibili dell'NFA

- esplora tutti i simboli in tutti gli stati
- costruisce ogni possibile configurazione

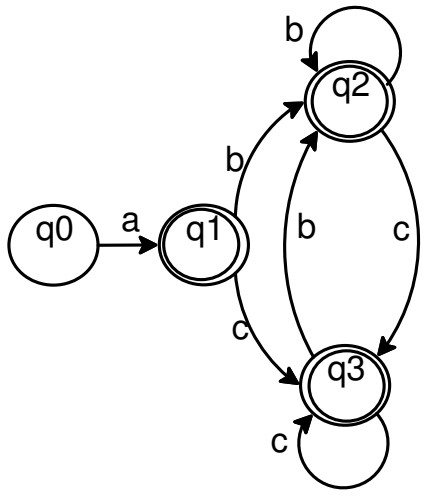
$\Rightarrow$ S e δ sono il DFA risultante

Esempio di conversione NFA→DFA (1)

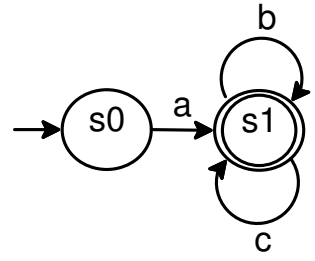


Stati DFA (q)	Stati NFA (s)	ϵ-CLOSURE(MOVE(q,*))		
		a	b	c
q_0	s_8	$s_9, s_0, s_1, s_2, s_4, s_7$	–	–
q_1	$s_9, s_0, s_1, s_2, s_4, s_7$	–	$s_3, s_6, s_7, s_1, s_2, s_4$	$s_5, s_6, s_7, s_1, s_2, s_4$
q_2	$s_3, s_6, s_7, s_1, s_2, s_4$	–	q_2	q_3
q_3	$s_5, s_6, s_7, s_1, s_2, s_4$	–	q_2	q_3

Esempio di conversione NFA→DFA (2)



Generato automaticamente



Generato a mano

L'automa ottenuto:

- ha minori dimensioni del NFA corrispondente,
- ha tutte le transizioni deterministiche, e
- può essere implementato utilizzando il codice visto in precedenza.