UNIVERSITÀ
DEGLI STUDI
DI TORINO

Analizzatori lessicali

a.a. 2018-2019

Riferimenti bibliografici

Compilatori: principi, tecniche e strumenti
A.V. Aho, M. S. Lam, R. Sethi, J.D. Ullman

Analisi lessicale [cap.3]

- 3.3 Descrizione dei token
  - 3.3.4 Definizioni regolari
  - 3.3.5 Estensioni delle espressioni regolari
- 3.4 Riconoscimento dei token
  - 3.4.1 solo figura 3.11
  - 3.4.2 Riconoscimento di parole riservate e identificatori
  - 3.4.3. Completamento dell'esempio
  - 3.4.4 Architettura di un analizzatore lessicale basato su diagrammi di transizione
- 3.8 Progettazione di un generatore di analizzatori lessicali
  - 3.8.1 Struttura dell'analizzatore generato
  - 3.8.2 Riconoscimento dei pattern basato su NFA
  - 3.8.3 DFA per analizzatori lessicali

Definizioni regolari

Possiamo introdurre dei nomi simbolici per individuare specifiche espressioni regolari. Chiamiamo *definizioni regolari* una sequenza di definizioni del tipo:

$$
\begin{aligned}
d_1 &\rightarrow r_1 \\
d_2 &\rightarrow r_2 \\
&\ldots \\
d_K &\rightarrow r_k
\end{aligned}
$$

. .

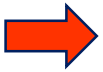
dove d_i è un nuovo simbolo distinto dagli altri e dai simboli dell'alfabeto Σ, r_i è un'espressione regolare su $\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$

$$
\begin{aligned}
\textit{letter\_} &\rightarrow A \mid B \mid C \mid \ldots \mid Z \mid a \mid b \mid \ldots \mid z \mid \_ \\
\textit{digit} &\rightarrow 0 \mid 1 \mid \ldots \mid 9 \\
\textit{id} &\rightarrow \textit{letter\_}(\textit{letter\_} \mid \textit{digit})^*
\end{aligned}
$$

Estensioni delle espressioni regolari

Da quando Kleene negli anni ’50 ha introdotto le espressioni regolari, sono state proposte molte estensioni per migliorarne la capacità espressiva.

*letter_* → A | B |...| Z | a | b |...| z | _
digit → 0 | 1 |...| 9
id → *letter_*(*letter_* | *digit*)*



*letter_* → [A-Za-z_]
digit → [0-9]
id → *letter_*(*letter_* | *digit*)*

digit → 0 | 1 | . . . | 9
digits → *digit digit**
opexp → (E(+ | - | ε) *digits*) | ε
number → *digits* (. *digits* | ε) *opexp*

digit → [0-9]
digits → *digit* $^+$
number → *digits* (. *digits*)*?* (E [+-]? *digits*)?

Estensioni delle espressioni regolari

- **Espressioni regolari in Unix:**
  Esempio:

  `'[A-Z][a-z]*[][A-Z][A-Z]'`

  e' compatibile con (matches) `Ithaca NY`
  non e' compatibile con `Palo Alto CA`

- Domanda: Quale espressione e' compatibile con
  `Palo Alto CA`?

Generatori di analizzatori lessicali

- I generatori di analizzatori lessicali non solo hanno reso veloce la loro implementazione, ma reso facili le modiche: a volte basta cambiare una o due espressioni regolari, senza entrare nel codice per correggere errori.

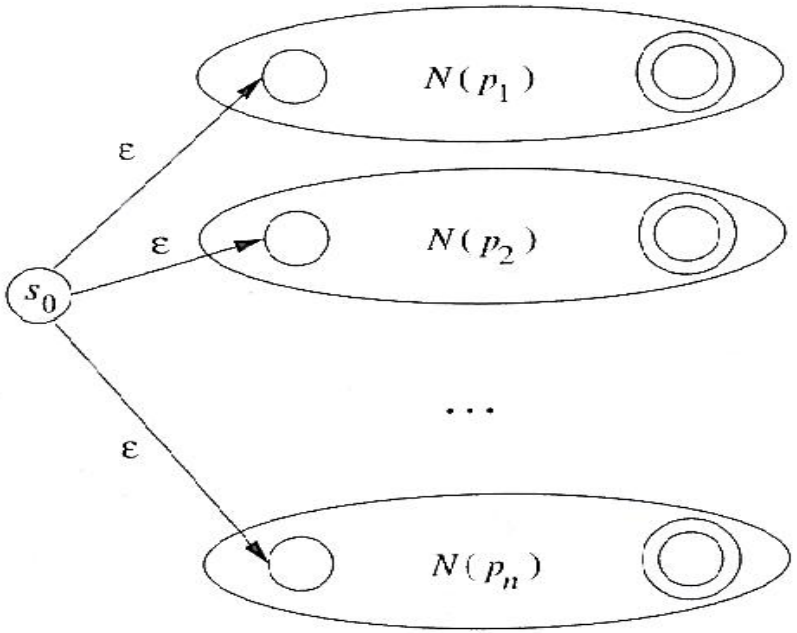
- I comandi “Lex” e “Flex” di UNIX, che sono *generatori di analizzatori lessicali*, accettano in input una lista di espressioni regolari (stile UNIX).

- Questi comandi, per generare un programma efficiente che scompone il codice sorgente in token, sfruttano il processo di conversione da espressione regolare ad automa.

Generatori di analizzatori lessicali

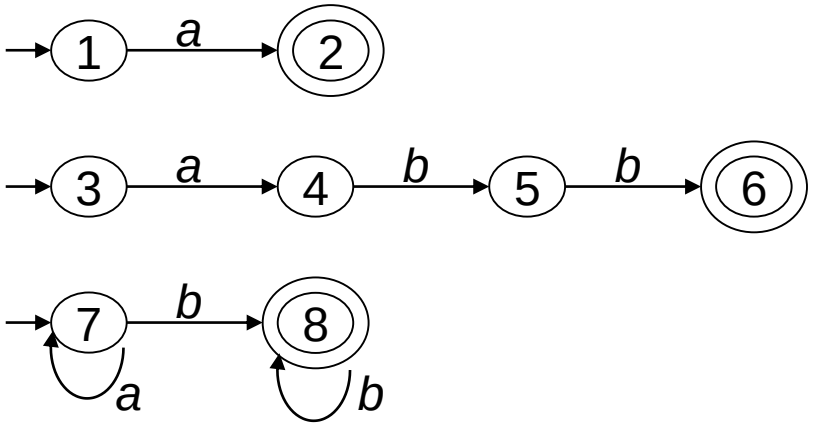
1. Trasformano ogni espressione regolare in un NFA

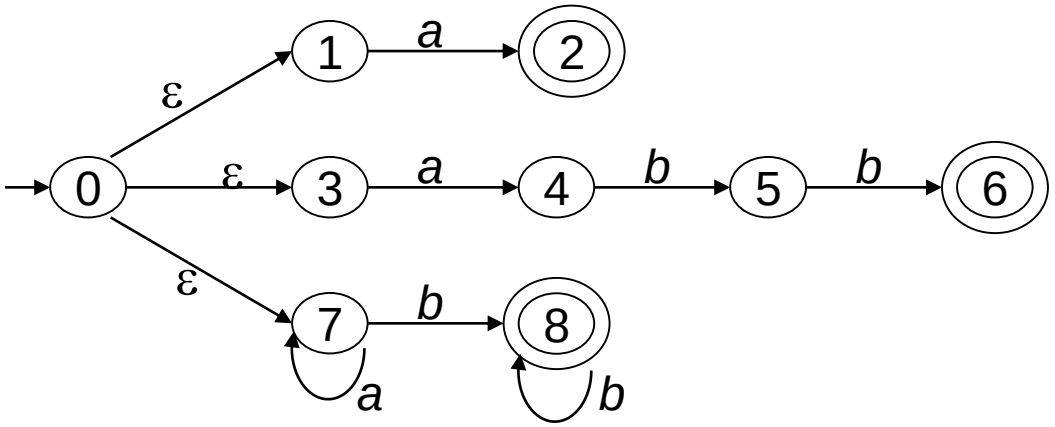
2. Combinano gli NFA in un unico automa aggiungendo un nuovo stato iniziale con transizioni ε verso ognuno degli stati iniziali degli automi N_i relativi ai pattern p_i.



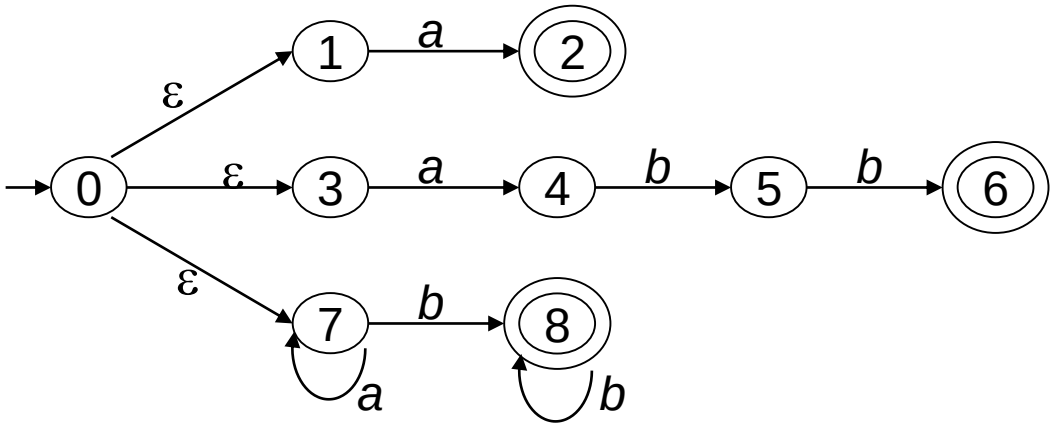
Generatori di analizzatori lessicali

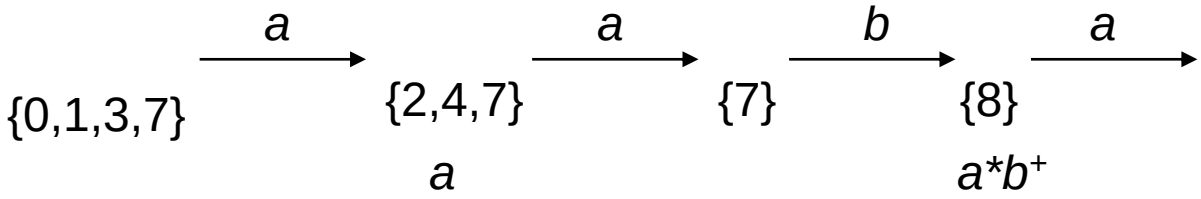
Esempio: si abbiano le tre espressioni regolari *a, abb* e a^*b^+





Generatori di analizzatori lessicali



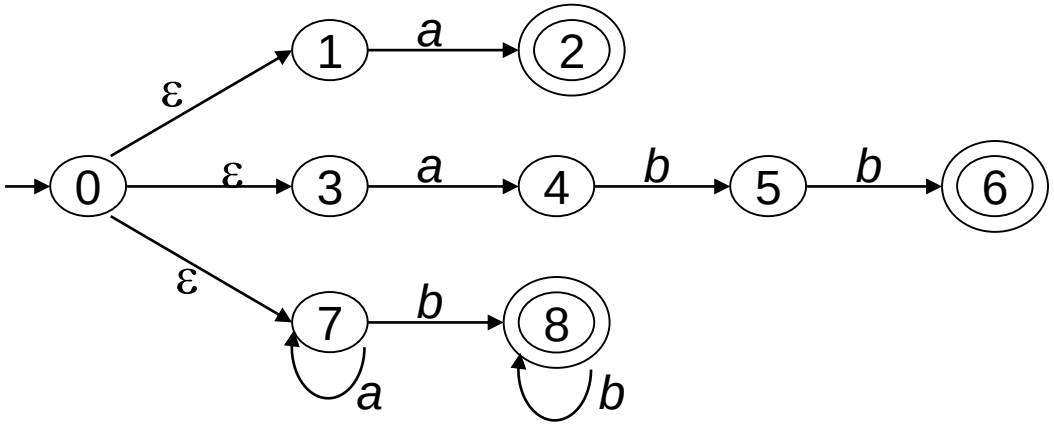


CONFLITTO !

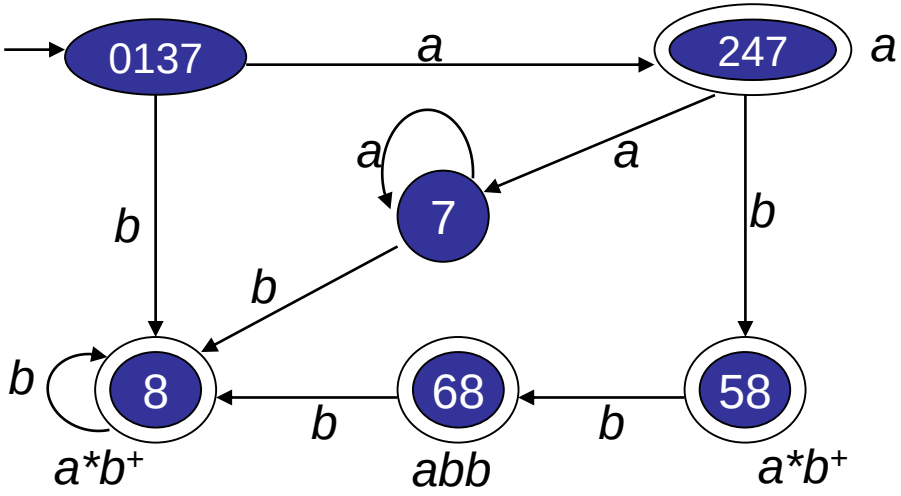
Ad esempio, Lex riconosce il prefisso più lungo che soddisfa un pattern e, se una stringa soddisfa più pattern, viene considerata un lessema del primo specificato.

Generatori di analizzatori lessicali

Alcuni analizzatori convertono l'NFA relativo a tutti i pattern in un DFA equivalente mediante la costruzione per sottinsiemi.



	a	b
[0,1,3,7]	[2,4,7]	[8]
[2,4,7]	[7]	[8]
[7]	[7]	[5,8]
[8]	-	[8]
[5,8]	-	[6,8]
[6,8]	-	[8]



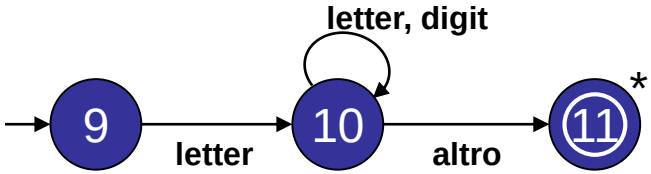
Ottimizzazione dei riconoscitori basati su DFA

1. Costruire un DFA a partire da un'espressione regolare in modo diretto, cioè senza costruire un NFA come passo intermedio. Il DFA così costruito in alcuni casi ha meno stati del DFA costruito passando da un NFA.

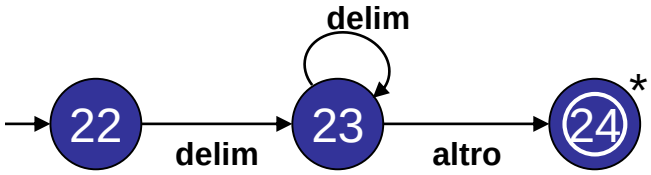
2. Rappresentare in modo più compatto le tabelle di transizione.

3. Minimizzare il numero di stati dei DFA, combinando gli stati che hanno lo stesso comportamento.

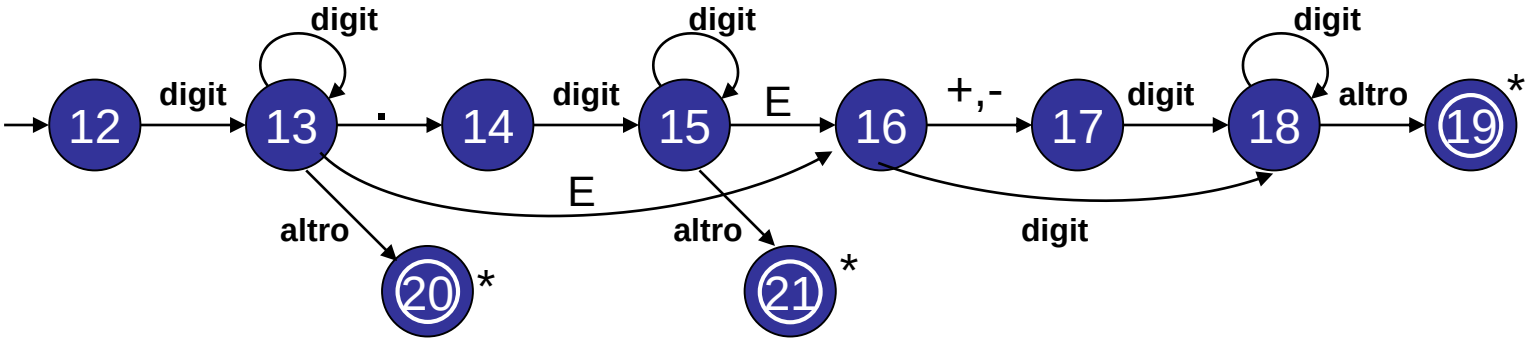
Riconoscimento dei token: esempio



Automa per "*id*" e parole chiave



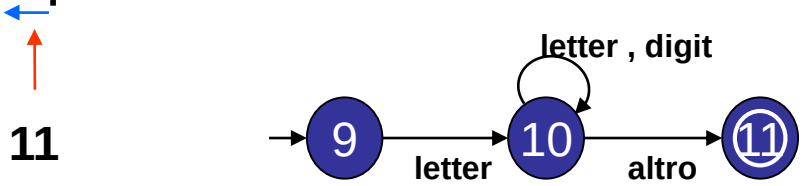
Automa per i delimitatori



Automa per i numeri senza segno

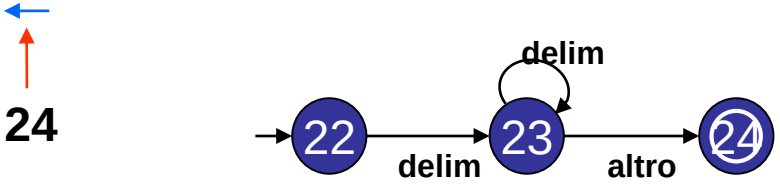
Riconoscimento dei token: esempio

if p1==3.14 then



riconosce un *id*, e verifica che è una parola riservata, ritorna <259, if>

if p1==3.14 then



riconosce una sequenza di delimitatori, ignora

if p1==3.14 then

riconosce un *id* e ritorna il token <257, p1>

Riconoscimento dei token: esempio

if p1==3.14 then

5

riconosce ==, ritorna <258, ==>

if p1==3.14 then

21

riconosce un numero e ritorna <256, 3.14>

if p1==3.14 then

24

riconosce una sequenza di delimitatori, ignora

if p1==3.14 then

11

riconosce un *id*, e verifica che è una parola riservata, ritorna < 260, then>