ANALISI LESSICALE

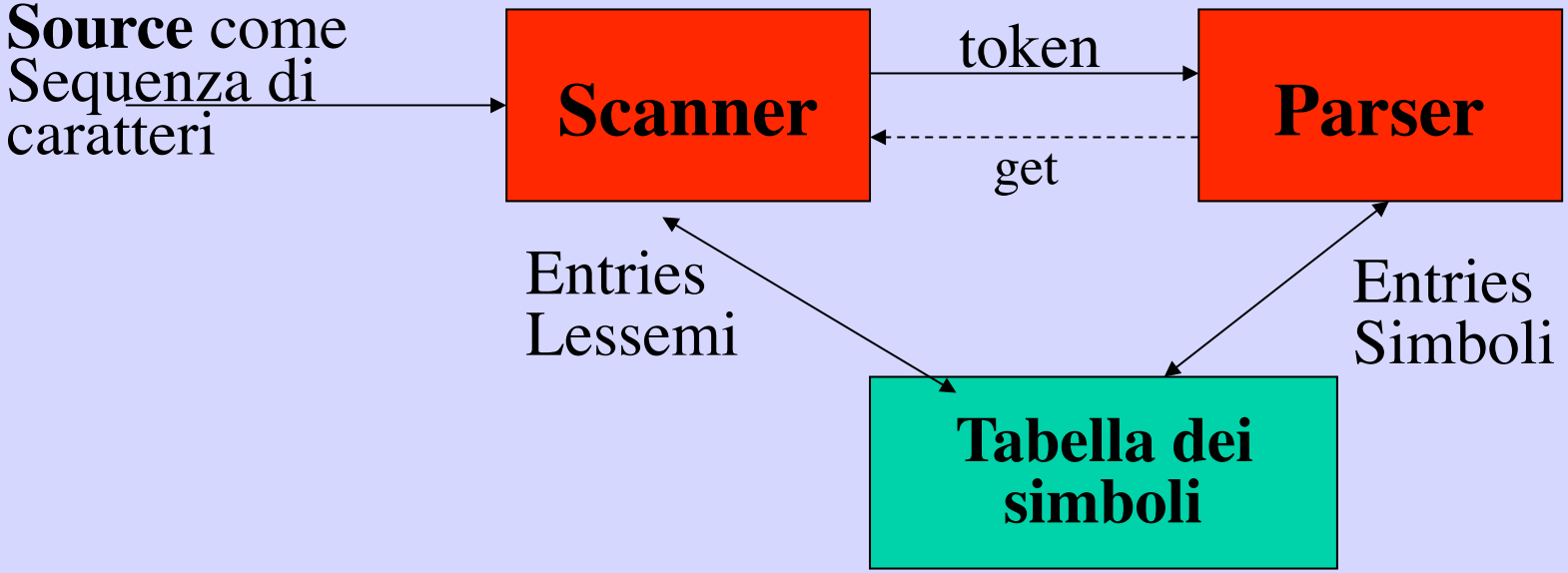
Controllare e riconoscere il linguaggio utilizzato per scrivere i simboli (identificatori, numerali, keywords, separatori, delimitatori,...)

linguaggio lessicale in generale semplice

linguaggio regolare: $\mathfrak{R}$
grammatiche regolari o lineari
diagrammi di transizione

Dove e Come:

Una struttura una passata



Analisi lessicale (Scanner)
guidata dall'analisi sintattica (Parser)

Definizioni: token, lessema, pattern

token = *categoria* lessicale

lessema = *valore* di una categoria

pattern = *regole* lessicali con cui definire ogni categoria lessicale

esempio

const pigreco = 3.1416

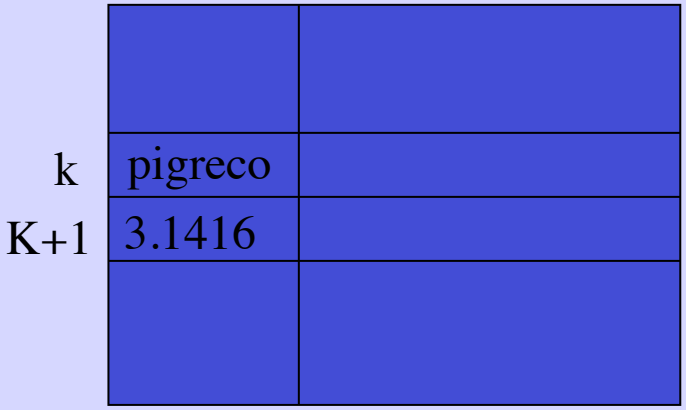
const id rel num

Lessema:

Dove finiscono i valori?

In effetti generiamo coppie: **<token**,attributo>

<const,**>** **<id**,k> **<rel**,=> **<num**,k+1>



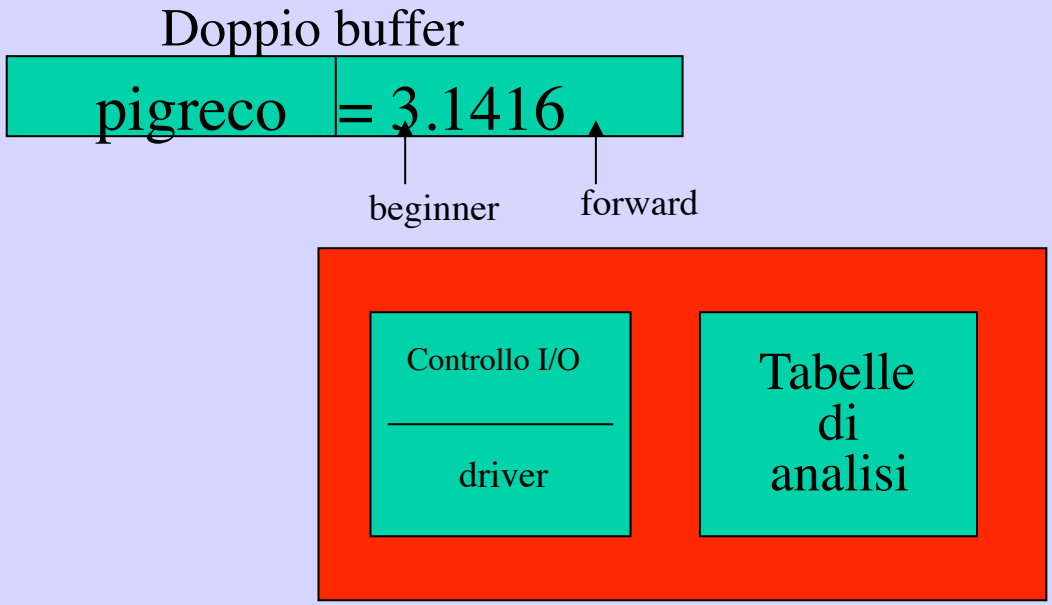
Symboltable

Pattern:

riconoscitori generati automaticamente

- I **pattern** sono definiti con vari formalismi:
  - diagrammi di transizione
  - grammatiche regolari

- Possiamo costruire riconoscitori basati sui patterns



Grammatiche regolari

Espressioni Regolari: E_Σ

Espressioni regolari E_Σ su Σ sono il

minimo insieme definito ricorsivamente da:

1. $\varepsilon \in E_\Sigma$
2. $a \in E_\Sigma, \ \forall a \in \Sigma$
3. $e_1.e_2 \in E_\Sigma, \forall e_1,e_2 \in E_\Sigma$
4. $e_1|e_2 \in E_\Sigma, \forall e_1,e_2 \in E_\Sigma$
5. $e^* \in E_\Sigma, \forall e \in E_\Sigma$
6. $e^i \in E_\Sigma, \forall e \in E_\Sigma$

Espressioni Regolari:
significato

Hanno significato L su 2^{Σ^*}

$\Sigma = \{a,b\}$, $\Sigma^i = \Sigma \times .\Sigma^{i-1}$, $\Sigma^0 = \{\lambda\}$, $\Sigma^2 = \{aa, ab, bb, ba\}$
$\Sigma^* = \cup_{i\in N} \Sigma^i = \{\lambda, a, b, aa, ab, bb, ba, aaa, \ldots\}$
$2^{\Sigma^*} = \{u \mid u \subseteq \Sigma^*\} = \{\{\lambda\},\{a\},\{b\},\ldots,\{\lambda,a\},\ldots,\{\lambda,a,b,ab\},\ldots,\}$

1. $[\varepsilon] = \{\lambda\}$
2. $[a] = \{a\}$
3. $[e_1.e_2] = \{uv \mid u \in [e1], v \in [e2]\} = [e1] \times . [e2]$
4. $[e_1 | e_2] = \{u \mid u \in [e1] \cup [e2]\}$
5. $[e^*] = \{u \mid u \in \cup_{i\in N} [e]^i\}$
6. $[e^i] = \{u \mid u \in [e]^i\}$ *abbreviazione*

Espressioni Regolari:
Esempi

0 | 1 | … | 9

[0 | 1 | … | 9] = {0,1,…,9}

(0 | 1 | … | 9)*

[(0 | 1 | … | 9)*] = {0,1,…,9, 00, 01,…, 3808, ...}

(1 | … | 9).(0 | 1 | … | 9)*

[(1 | … | 9).(0 | 1 | … | 9)*] = {1,…,9, 10, 11,…, 3808, ...}

Grammatica

su Σ

$G = \langle \mathbf{V}, \Sigma, \mathbf{s} \in \mathbf{V}, \mathbf{P} \rangle$

V *non terminali* (token ed ausiliari)

Σ *terminali*

s *simbolo distinto*

$\mathbf{P} \subseteq V \times E_{\Sigma \cup V}$ *produzioni*

Grammatica

quando *grammatica regolare?*

G e' una grammatica, affinche' sia
regolare dobbiamo aggiungere:

- V sono totalmente ordinati,<
- per ogni v::=e ∈P,
  e ∈ $E_{\Sigma \cup \{v'<v\}}$

Grammatica

Esempio: Relazione tra grammatiche, espressioni

Un lessico per i numerali con grammatiche regolari

digit::= 0 | 1 | … | 9
simple::= digit digit*
fract::= simple.simple
exp::= fract E simple |
fract E (+|-) simple
num::= simple | fract | exp

d::= 0 | 1 | … | 9
s::= d d*
f::= s.s
e::= f E s |
f E (+|-) s
n::= s | f | e

Solo produzioni:

- chi e' Σ ?
- quali strutture sono in G ?

Omettiamo il simbolo "." dell'operatore di giustapposizione

Grammatiche: Significato, equazioni tra Linguaggi

$$e \in \mathbf{E}_{\Sigma} \Longrightarrow \mathbf{L}(e) \in \mathbf{2}^{\Sigma^*}$$

$$v ::= e \Longrightarrow \mathbf{v} = \mathbf{L}(e) \in (V \times \mathbf{2}^{\Sigma^*})$$

$$\{v_1 ::= e_1, \ldots, v_k ::= e_k\} \Longrightarrow \{\underline{\mathbf{L}}(v_1), \ldots, \underline{\mathbf{L}}(v_k) \mid \forall i, \underline{\mathbf{L}}(v_i) \in \mathbf{2}^{\Sigma^*} \textit{ and } \underline{\mathbf{L}}(v_i) \equiv \mathbf{L}(e_{i\{\underline{\mathbf{L}}(v_j)/v_j \mid v_j < v_i\}})\}$$

$$G = \langle V, \Sigma, s \in V, P \rangle \Longrightarrow \underline{\mathbf{L}}(G) = \underline{\mathbf{L}}(s)$$

Grammatiche Regolari: Significato, Esempio

d::= 0 | 1 | … | 9
s::= d d*
f::= s.s
e::= f E s |
f E (+|-) s
n::= s | f | e

d = L(0|1|…|9) = {0,…,9}

{L(d)={0,…,9},
L(s)= L(d d*$_{\{0,\ldots,9\}/d}$)={0,…,9}{0,…,9}*
={0,…,9}$^+$
…

L(n) = {0,…,9}$^+$
∪
{u.v | u,v ∈ {0,…,9}$^+$}
∪
{u.vEw | u,v,w ∈ {0,…,9}$^+$}
…

Due Esercizi

1. Espressione regolare per tutte e solo le stringhe contenenti un numero dispari di *a* sull'alfabeto $\Sigma=\{a\}$

Soluzione: (aa)*a

2. Espressione regolare per tutte e solo le stringhe contenenti contigui diversi su $\Sigma=\{a,b,c\}$

Soluzione:

- Difficile: Le espressioni regolari (e le grammatiche regolari) non sono l'unico, nè il più espressivo formalismo con cui esprimere un Linguaggio regolare, $\Re$
  - Alri formalismi:
    - (a) grammatiche context free
    - (b) automi a stati finiti (diagrammi sintattici)
  - Vediamo come si comportano:

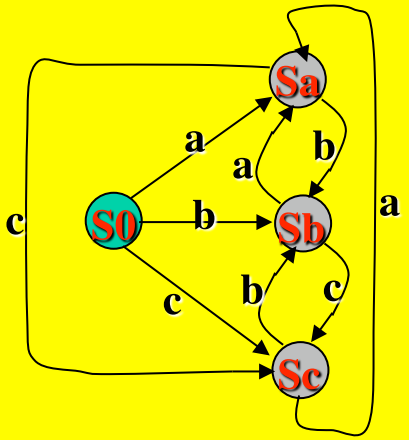
Due Esercizi - continua1

(a) Grammatiche context free: G=<{S,Sa,Sb,Sc},{a,b,c},S,P} &
P={
S::= Sa | Sb | Sc
Sa::= a(Sb | Sc) /*commento: sottolinguaggio stringhe inizianti con “a” e seguite da stringhe di L(G) non inizianti con “a”. */
Sb::= b(Sa | Sc) /*commento: sottolinguaggio stringhe inizianti con “b” e seguite da stringhe di L(G) non inizianti con “b”. */
Sc::= c(Sa | Sb) /*commento: sottolinguaggio stringhe inizianti con “c” e seguite da stringhe di L(G) non inizianti con “c”. */
}

- G non è regolare
- Molto difficile ottenere da G un’espressione regolare per L(G)
- G è context free e può fornire un riconoscitore a pila per L(G)
- I riconoscitori a pila sono più dispendiosi dei riconoscitori a stati finiti

Due Esercizi - continua2

(a) Automi:
A=<{S0,Sa,Sb,Sc},{a,b,c},M,S0,{Sa,Sb,Sc}>
& M=grafo a fianco



- Automa A Deterministico a stati finiti
- Automi usati come riconoscitori per $\Re$, formano anche un formalismo per esprimere $\Re$
- "Facile" passare da un automa ad un'espressione regolare equivalente

L(Sa): a (ε | ba | (ba)* b (cb)* (a | ca) | c (a | (bc)* (a | b (cb)* ca | b (cb)* (ab)* a | b (cb)* (ab)* (cb)* ca))*

- Automi Deterministici e Nondeterministici: li studiamo sistematicamente nelle pagine che seguono

Due Esercizi - continua3

Completare la soluzione dell'esercizio2 calcolando:

- L'espressione regolare per L(Sb)
- Siano Ea, Eb, Ec le espressioni che esprimono i linguaggi L(Sa), L(Sb), L(Sc), si esprima E per il linguaggio L(S)

Si estenda la soluzione dell'esercizio1 al caso in cui l'alfabeto sia:

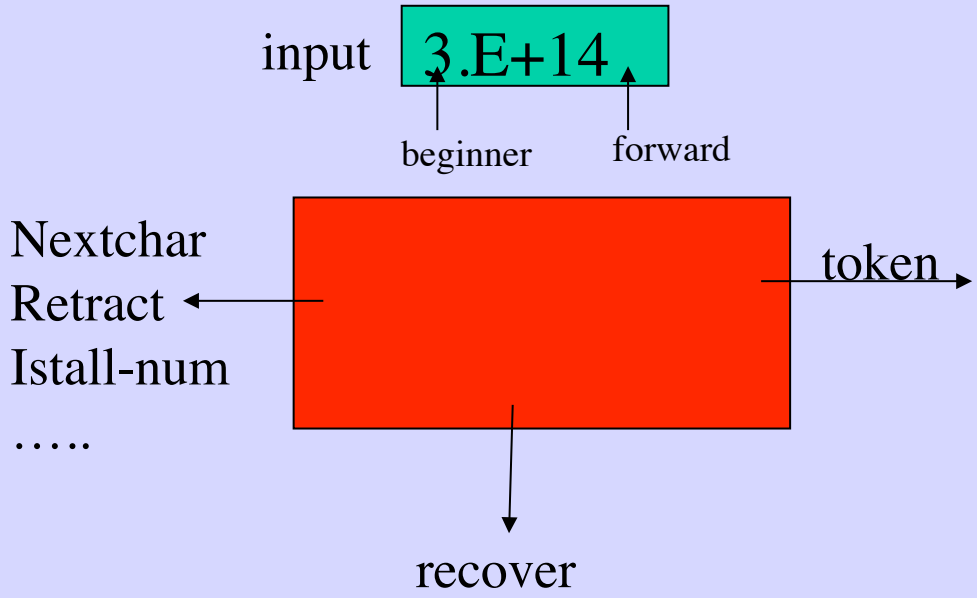
$\Sigma=\{a, b, c\}$

Si considerino gli esercizi alle pagine del corso indirizzate da:

www.di.unipi.it/~bellia/FormaliWeb/FormaliWeb/materiale2004/materiale/LFC/ESERCIZI/lessico.html

Riconoscere il lessico espresso da un pattern

d::= 0 | 1 | … | 9
s::= d d*
f::= s.s
e::= f E s |
f E (+|-) s
n::= s | f | e



Riconoscere il lessico

AUTOMI

Automa a stati finiti

$\langle S, \Sigma, move: S \times \Sigma' \rightarrow S', s_0 \in S, F \subseteq S \rangle$

per S finito

NFA (nondeterministico)

$\Sigma' = \Sigma \cup \{\varepsilon\}$ mentre $S' = 2^S$

DFA (deterministico)

$\Sigma' = \Sigma$ mentre $S' = S$

AUTOMI a stati finiti: la funzione move: grafi, o tabelle

Possiamo usare grafi simili ai diagrammi [G] o tabelle [T] per descrivere la funzione finita *move*

Ad ogni stato S corrisponde:

G	un vertice del grafo
T	un'entry della tabella con tante colonne quanta la cardinalita' di $\sum$ (+1 per nondeter.) tante righe quanti gli stati

AUTOMI a stati finiti:
la funzione move: transizione

Ad ogni $\langle\langle s,a\rangle,S\rangle \in move$ corrisponde:

G un'arco <s,t> etichettato *a* per ogni $t \in S$

T il valore *S* per l'entry $\langle s,a\rangle$

La funzione move:

automa - un esempio

Esempio: **Consideriamo l'automa L**

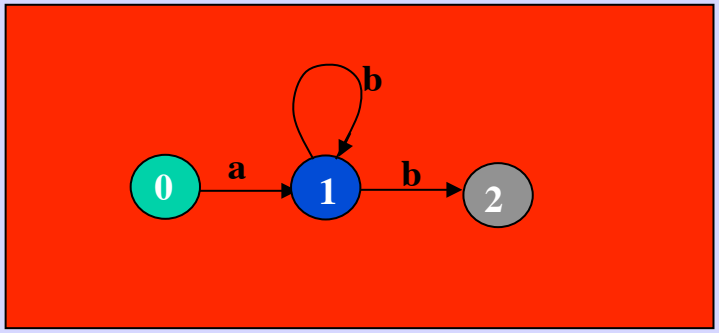
$<S = \{0,1,2\},$
$\Sigma = \{a,b\},$
$move = \{<<0,a>\{1\}>, <<1,b>\{1,2\}>\}$
$s_0 = 0,$
$F = \{2\} >$

E' questo un automa deterministico
oppure automa nondeterministico ?

La funzione move:

grafo o tabella - un esempio

<S= {0,1,2},
Σ= {a,b},
move= {<<0,a>{1}>,
<<1,b>{1,2}>}
s_0=0,
F= {2} >



	a	b	ε
0	{1}		
1		{1,2}	
2			

AUTOMI a stati finiti:

significato: $\Re$

DOVE POSSIAMO DARE SIGNIFICATO AD UN AUTOMA A STATI FINITI

A = $\langle S, \Sigma, move: S \times \Sigma' \rightarrow S', s_0 \in S, F \subseteq S \rangle$

sui linguaggi regolari $\Re$ su 2^{Σ^*}

AUTOMI a stati finiti:
significato: funzione di decisione

Usiamo funzioni di decisione

$sem(A) = f \in \Sigma^* \dashrightarrow \{accept, noaccept\}$

tale che: per ogni $c_1c_2....c_n \in \Sigma^*$

Informalmente

$f(c_1c_2....c_n) = accept$
se e solo se $c_1c_2....c_n$ appartiene al linguaggio

AUTOMI a stati finiti:
funzione di decisione: relazione su $(\Sigma^* \times S)^2$

definiamo la relazione $==>: (\Sigma^* \times S)^2$

$\gamma, s \; ==> \gamma', s'$

se e solo se:

$\gamma = c\gamma'$ and $s' \in move(c,s)$

oppure

$\gamma = \gamma'$ and $s' \in move(\varepsilon,s)$

AUTOMI a stati finiti:
funzione di decisione: definizione

Sia ==>* la chiusura transitiva di ==>

Allora:

$$\mathrm{SEM}(A)(\gamma) = f(\gamma) = \begin{cases} \textit{accept} \ \textbf{se} \ \gamma, s_0 ==>^* \lambda, s \in F \\ \textit{noaccept} \ \textbf{se} \ \text{per nessun } s \in F, \ \gamma, s_0 ==>^* \lambda, s \end{cases}$$

Automa:

Linguaggio, Equivalenza, Minimale

A = **<S, Σ, *move*: SxΣ'--> S', s_0∈S, F⊆S>**

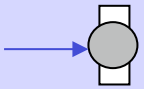
- **Linguaggio: L**
  - **L(A)** = {γ ∈ Σ* | sem(A)(γ)=accept}
- **Equivalenza: E**
  - **E(A)** = {A' | sem(A')=sem(A)}
- **Minimale: M**
  - **M(A)** =<S_m, _, _, _,_>∈ E(A):
    - #S_m ≤ #S_A per tutti A)=<S_A, _, _, _,_> ∈ E(A)

Linguaggio Regolare: Automa

- Un automa esprime un Linguaggio regolare
- Tutti i linguaggi regolari sono esprimibili con:
  - Automi
  - Automi deterministici

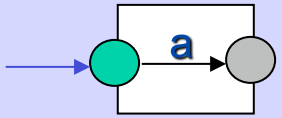
Da E_Σ a NFA
approccio di K. Thompson - 1

1. ε



$<\{s_0\}, \{\}, \{\}, s_0 \in S, \{s_0\}>$

2. *a* per ogni $a \in \Sigma$

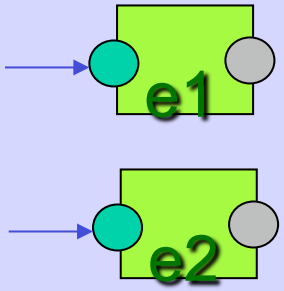


$<\{s_0,s_1\}, \{a\}, \{<<s_0,a>,s_1>\}, s_0 \in S, \{s_1\}>$

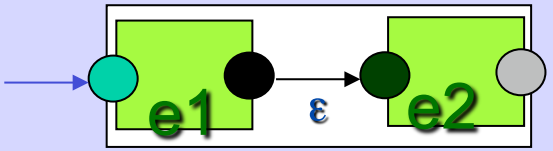
Approccio di K. Thompson - 2

3. e1.e2 per ogni e1,e2∈E

e1: $<S_1, \Sigma_1, M_1, s_1, \{f_1\}>$
e2: $<S_2, \Sigma_2, M_2, s_2, \{f_2\}>$



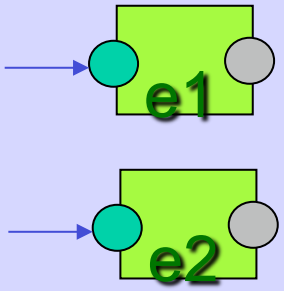
e1.e2: $< S_1 \cup S_2, \Sigma_1 \cup \Sigma_2,$
$M_1 \cup M_2 \cup \{<< f_1, \varepsilon >, \{s_2\}>\},$
$s_1, \{f_2\}>$



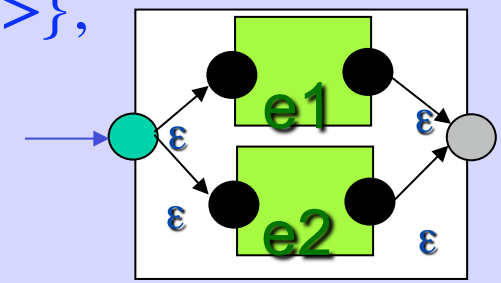
Approccio di K. Thompson - 3

4. e1|e2 per ogni e1,e2∈E

e1: $\langle S_1, \Sigma_1, M_1, s_1, \{f_1\}\rangle$
e2: $\langle S_2, \Sigma_2, M_2, s_2, \{f_2\}\rangle$



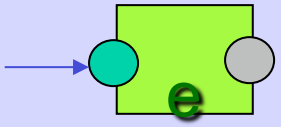
e1|e2: $\langle S_1 \cup S_2 \cup \{s_{new}, f_{new}\}, \Sigma_1 \cup \Sigma_2,$
$M_1 \cup M_2 \cup \{\langle\langle s_{new}, \varepsilon\rangle, \{s_1, s_2\}\rangle,$
$\langle\langle f_1, \varepsilon\rangle, \{f_{new}\}\rangle$
$\langle\langle f_2, \varepsilon\rangle, \{f_{new}\}\rangle\},$
$s_{new}, \{f_{new}\}\rangle$



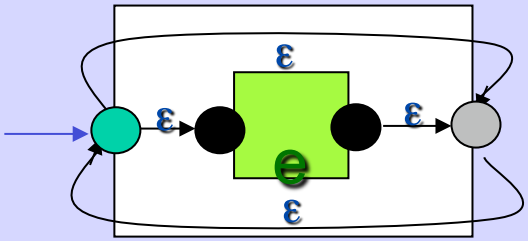
Approccio di K. Thompson - 4

5. e*per ogni e∈E

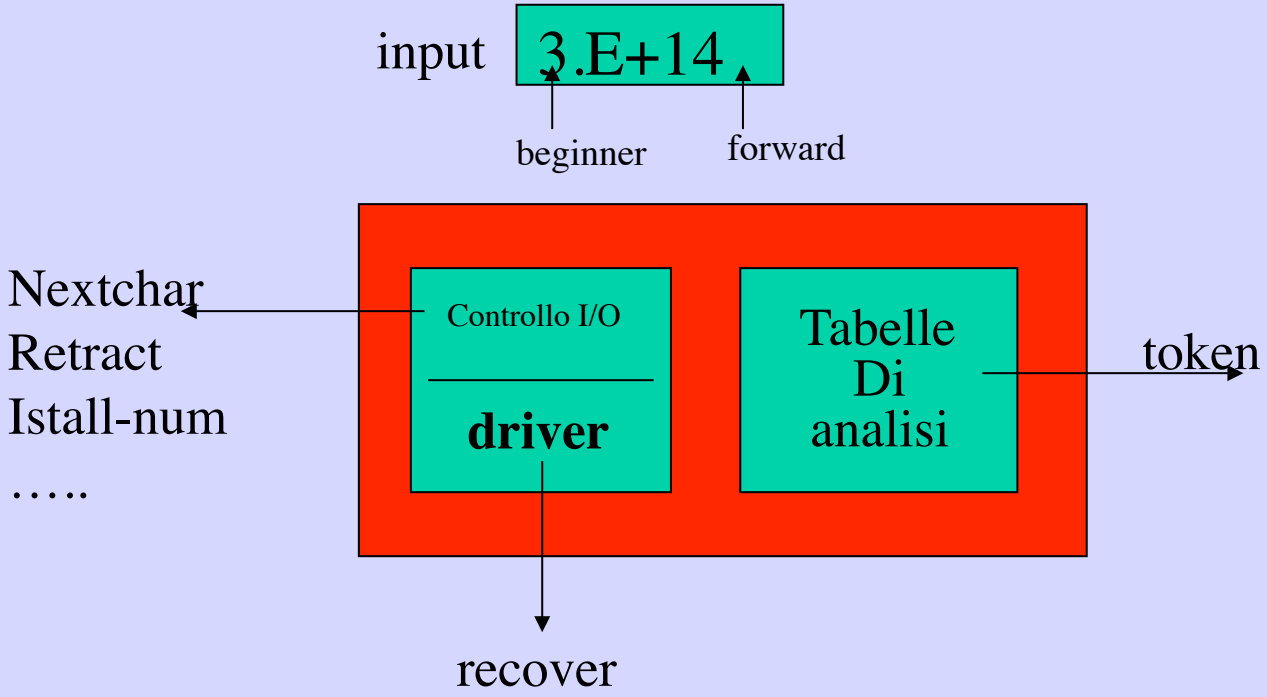
e: <S, Σ,M,s,{f}>



e* : < S ∪ {s_{new}, f_{new}}, Σ,
M ∪{<< s_{new}, ε>,{s, f_{new}}>,
<< f, ε>,{s, f_{new}}>},
s_{new}, {f_{new}}>



Un driver per riconoscere il lessico



Un driver per NFA (DA)

```
Answer Driver()
    {states=push([Clos({s0}),pointerInput()]);
     repeat
        answer='accept';
        [s,input]=pop(states);
        nextchar(c);
        while(c≠eof) and (s≠⊥) {
            S=move[s,c];
            push([Clos(S),pointerInput()]);
            [s,input]=pop(state);
            nextchar(c);}
        if (s∉F) or (c≠eof) answer='noaccept';
     until  emptystack() or (answer='accept');
     return (answer);
    }
```

States: stack di controllo
Push: inserisce [S,&]
S: insieme di stati
&: punatore input
Pop: rimuove uno stato da S e copia & nell'input. Se S e' singoletto l'intera coppia [S,&] e' rimossa dallo stack
⊥: valore calcolato da move[s,c] quando la tabella non ha entry per move[s,c]
Clos: ε-chiusura insieme di stati
Clos(S)=S
∪Clos($\cup_{u \in S}$move(s,ε))

Un driver per NFA: eliminiamo il backtracking - 1

$$f(\gamma) = \begin{cases} \textit{accept} \ \textbf{se} \ \text{move1}^*(\gamma) \cap F \neq \{\} \\ \textit{noaccept} \ \textbf{altrimenti} \end{cases}$$

move

	a	b	ε
0	{1}		
1		{1,2}	
2			

move1

	a	b	ε
0	{1}		
1		{1,2}	
2			
{1,2}		{1,2}	

Eliminiamo il backtracking - 2

realizzare move1*

La funzione *move1*:*

$$move1(S,c) = Clos(\cup_{s \in Clos(S)} \{move(s,c)\})$$

ricorda che: $Clos(S) = S \cup Clos(\cup_{s \in S} move(s,\varepsilon))$

Allora:

$move1^*(c) = move1(\{s_0\},c)$

$move1^*(c_1,\ldots,c_{n-1},c_n) =$
$move1(move1^*(c_1,\ldots,c_{n-1}),cn)$

Eliminiamo il backtracking - 2

un driver lineare per NFA

move1* ci fornisce un riconoscitore

```
Answer move1star()
        {S=Clos({s0});
         nextchar(c);
         while(c≠eof) and (S≠∅) {
                  S=move1(S,c));
                  nextchar(c)};
         if (S∩F) ≠ ∅  return 'accept';
         return 'noaccept';
        }
```

Lineare
(passo exp)

Esercizio 1

Un convertitore NFA in DA

compiliamo move1: approccio

Una struttura (e una notazione)

Star: Set -> (Σ -> Set)

- È una tabella con righe in Set e colonne in Σ
- *Star*[S] - è la riga S della tabella
- *Star*[S]c - è l'entry di riga S e colonna c
- Star non contiene colonna ε

Un'operazione

merge(move,S) = merge-row({move[s] | s∈S})

$\forall_{c\in\Sigma}$ merge-row({R_1,…,R_k}) c = ($\cup_{1\le i\le k}$Clos(R_i c))

	digit	.	ε
0	1		1
1	3	2	{1,3}
2		2	
3	0	4	
4	4	5	
5	6		
6	6		

move

merge-row({0,1,3})= {0,1,3} | {2,4}

Un convertitore NFA in DA

compiliamo move1: realizzazione

Tabella ***movestar()***

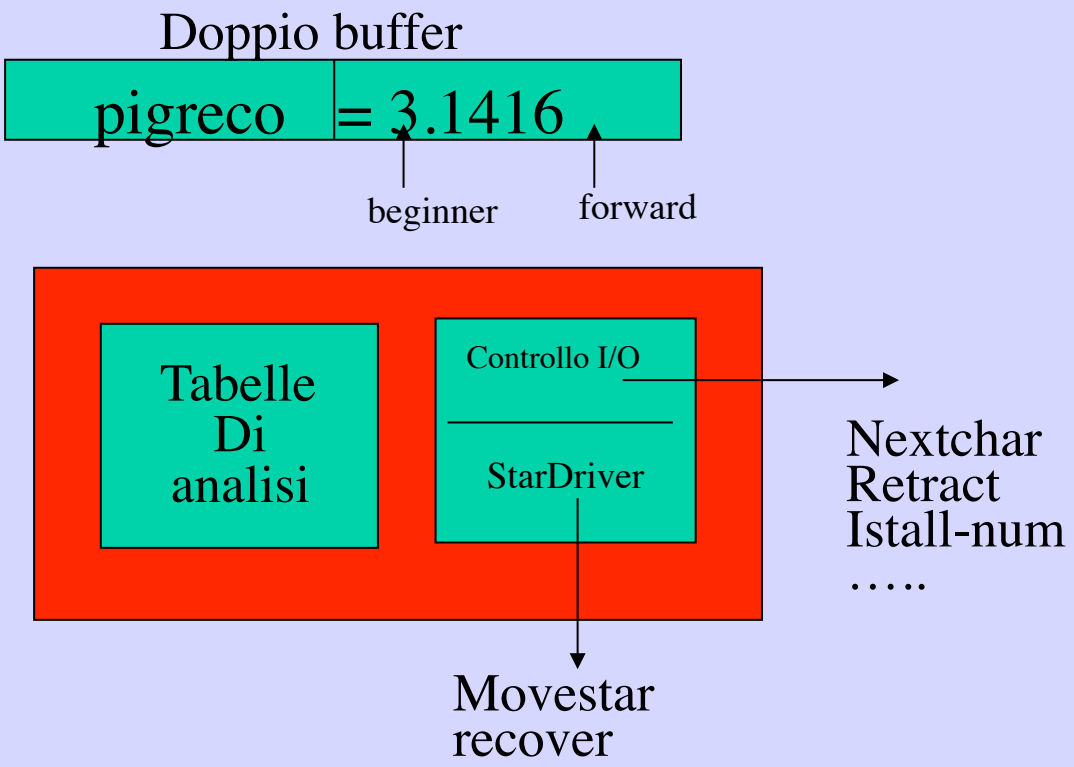
```
{EntryStar = ∅;
 List= Clos({s0});
 while (List≠emptylist) {
        S=firstout(List);
        if (S∉EntryStar) {
               add(S,EntryStar);
               Star[S]= merge(move,S);
               List=List+{Star[S]c | c∈Σ};}
        }
 return Star;
}
```

Esercizio 2

Un driver per DA (solo DA)

```
Answer StarDriver()
        {s= s0;
         nextchar(c);
         while(c≠eof) and (s≠{}) {
                  s=Star[s]c;
                 nextchar(c);}
         if ((s∉F) or (c≠eof)) answer='noaccept'
        else answer='accept';
        return (answer);
        }
```

lineare


Doppio buffer
pigreco = 3.1416
beginner
forward
Tabelle
Di
analisi
Controllo I/O
StarDriver
Nextchar
Retract
Istall-num
…..
Movestar
recover

Conclusioni

- Il *lessico* è un *linguaggio regolare*
- I *linguaggi regolari* sono un sottoinsieme $\Re \subseteq 2^{\Sigma^*}$:

  $F = \{I \subseteq \Sigma^* \mid \#I < n, \text{ per naturale } n\}$

  $\Re$ è la *chiusura* di F rispetto *unione*, *prodotto giustapposto*, *stella di Kleene*

  $\Re = F_{\{\cup, X_\bullet, *\}}$

- Gli *automi a stati finiti* forniscono riconoscitori lineari per $\Re$

Proprietà interessanti (1)

L1 e L2 sono regolari.

Unione: L1∪L2 e' regolare? SI

Prodotto: L1×L2 e' regolare? SI

Intersezione: L1∩L2 e' regolare? SI

Complemento: C(L1) e' regolare? SI

L e' un linguaggio

Decidibilita': L e' regolare? NO

Proprietà interessanti (2)

L e' regolare
$A=<S,\sum,m,s_0,F>$ e' automa per L

Th.(di iterazione – Pumping Lemma)
$\exists k(=\#S)$: Se $x \in L$, $|x|>k$,
Allora: $\exists u,w,v$ tali che $0<|w|\leq k$, $x=u.w.v$,
$u.w^n.v \in L \quad (\forall n \in \aleph)$

Corollario
$\exists k(=\#S)$: Se $x \in L$, $|x|>k$,
Allora: $\exists u,w,v$ tali che: $0<|uw|\leq k$, $|w|\neq 0$, $x=u.w.v$,
$u.w^n.v \in L \ (\forall n \in \aleph)$

Esercizi

fine

Esercizio

Si consideri il lessico dei numerali per interi e numeri in virgola fissa in notazione decimale e senza limite sul numero di cifre.

1. Si diano le espressioni regolari per: i numerali per interi, S, i numerali per quelli in virgola fissa, F, l'unione dei due.
2. Si dia una grammatica regolare per tale lessico
4. Si dia un automa per tale lessico
5. Si dia un automa deterministico per tale lessico
6. Si dia un riconoscitore deterministico per tale lessico
7. Si modifichi tale riconoscitore affinchè sia in grado di riconoscere sequenze di parole di tale lessico separate da caratteri non appartenenti a {0,..,9,'.'}, generando: …<pi,li>…, dove pi=posizione, li=lunghezza della i-esima parola riconosciuta.

7. Si modifichi tale riconoscitore affinchè sia in grado di riconoscere la prima e più lunga (7.1- breve) parola del lessico occorrente in una stringa di caratteri su un alfabeto contenente {0,..,9,'.'}

Esercizio - 1

1. Si diano le espressioni regolari per: i numerali per interi, S, i numerali per quelli in virgola fissa, F, l'unione dei due.

S = digit digit*
F = digit*.digit digit*
Digit = 0|1|…|9

Esercizio - 2

2. Si dia una grammatica regolare per tale lessico

S = digit digit*
F = digit*.digit digit*
Digit = 0|1|…|9

Regolare perchè: digit < S,F

oppure

S = digit digit*
F = digit*.S
Digit = 0|1|…|9

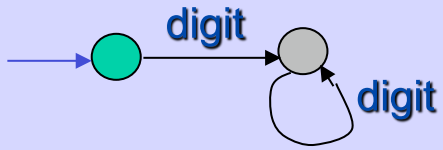
Regolare perchè: digit < S < F

Esercizio - 3

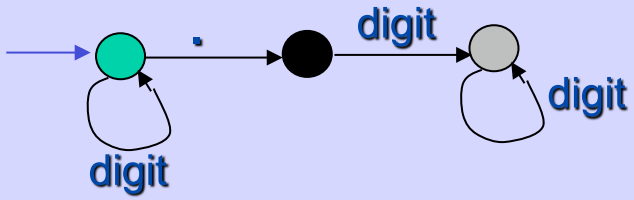
3. Si dia un automa per tale lessico

Osservazione. Possiamo semplificare la costruzione usando *digit* come fosse un carattere

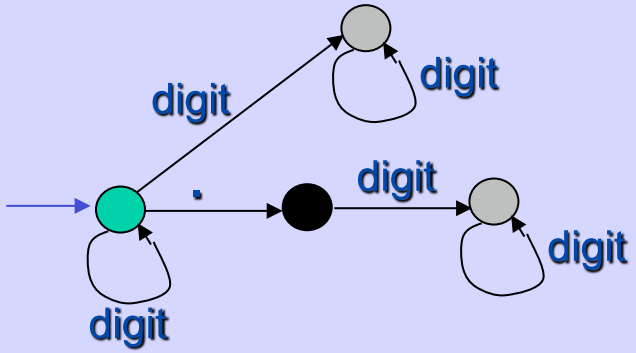
S = digit digit*



F = digit*. digit digit*

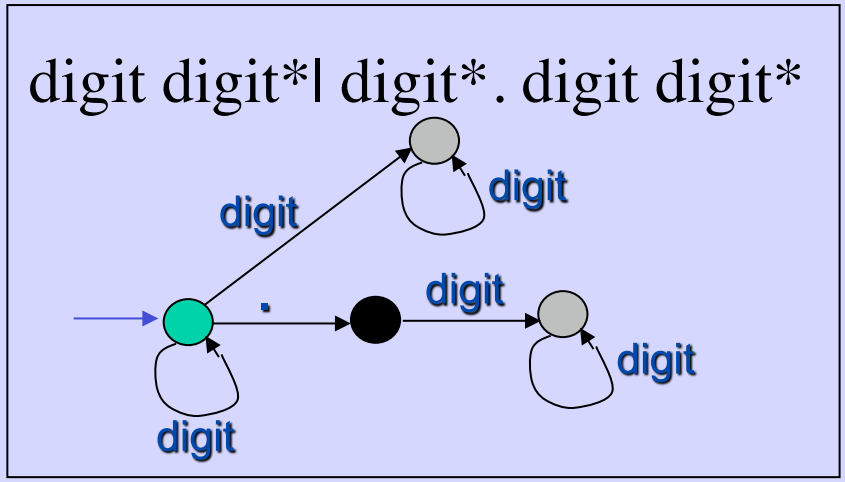


digit digit*| digit*. digit digit*



Esercizio - 4

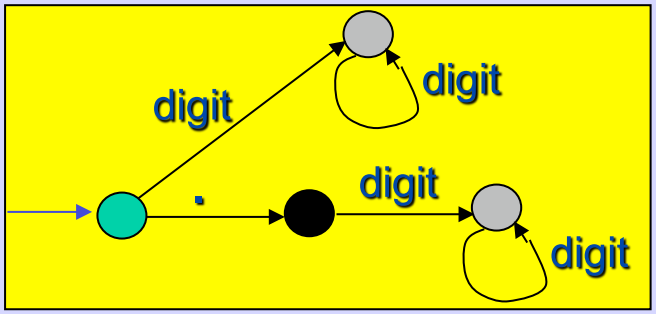
4. Si dia un automa deterministico per tale lessico



	digit	.
0	{0,1}	{2}
1	{1}	
2	{3}	
3	{3}	

Movestar

	digit	.
{0}	{0,1}	{2}
{0,1}	{0,1}	{2}
{2}	{3}	
{3}	{3}	



Esercizio - 5

5. Si dia un riconoscitore deterministico per tale lessico

```
Answer StarDriver()
{s= s0;
 nextchar(c);
 while(c≠eof) and (s≠{}) {
            s=Star[s]c;
            nextchar(c);}
 if ((s∉F) or (c≠eof)) answer='noaccept'
 else answer='accept';
 return (answer);}
```

Tabella di analisi: **Star**

	digit	.
{0}	{0,1}	{2}
{0,1}	{0,1}	{2}
{2}	{3}	
{3}	{3}	

Esercizio - 6

6. Si modifichi tale riconoscitore affinchè sia in grado di riconoscere sequenze di parole di tale lessico separate da caratteri non appartenenti a {0,..,9,'.'}

```
Answer StarDriver()
{input=0;
  answer = <>;
  while(nextchar (c) != eof){
      length=0; cur=input;
      s = s0;
      while(c∈Σ && s≠{}) {
                s=Star[s]c;
                nextchar(c); //incr. anche input
                length++;}
      if (s∈F) answer= answer ++ <cur-1,length>;}
return (answer);}
```

Tabella di analisi: **Star**

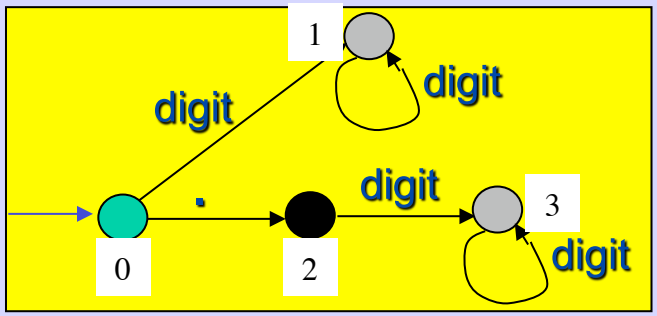
	digit	.
{0}	{0,1}	{2}
{0,1}	{0,1}	{2}
{2}	{3}	
{3}	{3}	

Altre tecniche per costruire scanners

- *Diagrammi di transizione*
- *Automi di items*

Diagrammi di transizione su Σ

- Utilizzati per costruzione *manuale* di riconoscitori
  - lessico molto semplice
- Sono grafi diretti etichettati:
  - funzione di transizione automa riconoscitore deterministico
  - anche non deterministico (allora, backtracking)



- Associa ad ogni stato un *case statement*

```
–case 0: if (c==digit) state = 1;
                else if (c==dot) state = 2;
                        else state = fail(0);
– case 2: if (c==digit) state = 3;
                else state = fail(2);
– case 1: if (c==digit) state = 1;
                else {  retract(1);
                        install-num();
                        return(S);}
– case 3: if (c==digit) state = 3;
                else {  retract(1);
                        install-num();
                        return(F);}
```

Diagrammi di transizione su Σ - continua

- Compone i frammenti

```
int state = 0;
int lexical-value;

token nexttoken()
      {while(1) {
          switch (state) {
                  case 0: …
                  case 1: …
                  case 2: …
                  case 3: …}
                  }
}
```

Funzioni di base utilizzate nella composizione (da implementare)

forward: cursore buffer input
c: carattere del buffer su cui e'posizionato forward
nextchar(): avanza forward di 1
retract(k): indietreggia forward di k posizioni
state: variabile contenente stato correntemente attraversato
Start: variabile contenente stato iniziale corrente riconoscimento
install-num(): aggiorna la symboltable
fail(state)-recover(state): gestiscono il fallimento
digit, dot, ... : predicati su caratteri
…

Automi di Items: Espressioni Regolari - definizioni

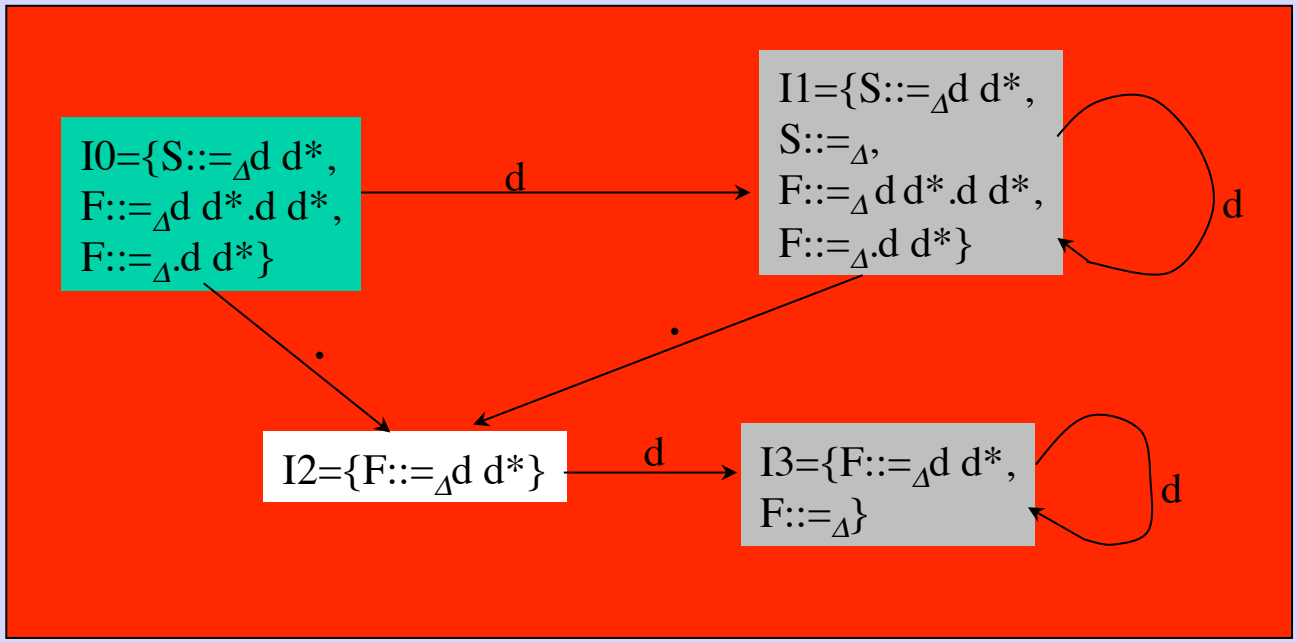
- Utilizzati per derivare *automi riconoscitori deterministici*
  - espressioni regolari e pattern matching (Lex)
- Sia e$\in E_\Sigma$ espressione regolare su alfabeto Σ
  - **Stato iniziale**: *I0=**C**({$_\Delta$e})*,
  - *Definizione della funzione di chiusura C (sulla base):*
    - ***C**(A)= $\cup_{t\in A}$ C(t)*
    - *C($_\Delta\varepsilon$)= {$_\Delta$};*
    - *C($_\Delta$bβ)= {$_\Delta$bβ}, $\forall$ b$\in\Sigma$, $\alpha,\beta\in\Sigma^*$;*
    - *C(e*β)= C{$_\Delta$ee*β}$\cup$C{$_\Delta\beta$};*
    - *C($_\Delta$(e1|e2)β)= C($_\Delta$e1 β)$\cup$C($_\Delta$e2 β);*
    - *addizionali casi per addizionali operatori regolari*
  - *Definizione della **funzione di move** **M**:*
    - **M**(Ii,a)=Ij **sse**
    - a$\in$*first(Ii)*={*a* | $_\Delta a\beta \in Ii$}
    - *Ij = **C**({$_\Delta\beta$ | $_\Delta a\beta \in Ii$})*
  - ***Stati Finali***: {Ij | $_\Delta\in Ij$}

Automi di Items: Grammatiche Regolari - definizioni

- Utilizzati per derivare *automi riconoscitori deterministici*
  - espressioni regolari e pattern matching (Lex)
- Sia P={s::=e} produzioni di G=<S, Σ, *s0*, *P*> regolare
  - **Stato iniziale**: $I0=\mathbf{C}(\{s::=_\Delta e \mid s::=e \in P, s \in S0 \subseteq S\})$,
    S0 contiene le categorie lessicali (token) di interesse
  - Definizione della ***funzione di chiusura*** **C** (sulla base):
    - $\mathbf{C}(\{s::=_\Delta \beta \mid s \in S\}) = \cup_{s \in S} C(s::=_\Delta \beta)$
    - $C(s::=_\Delta \varepsilon) = \{s::=_\Delta\}$;
    - $C(s::=_\Delta b\beta) = \{s::=_\Delta b\beta\}, \forall b \in \Sigma, \ \alpha,\beta \in \Sigma^*$;
    - $C(s::=_\Delta s'\beta) = \mathbf{C}(\{s::=_\Delta e\beta \mid s'::=e \in P\}) \ \forall b \in \Sigma$;
    - $C(s::=_\Delta e^*\beta) = C\{s::=_\Delta ee^*\beta\} \cup C\{s::=_\Delta \beta\}$;
    - $C(s::=_\Delta (e1|e2)\beta) = C(s::=_\Delta e1\ \beta) \cup C(s::=_\Delta e2\ \beta)$;
    - *addizionali casi per addizionali operatori regolari*
  - *Definizione della* ***funzione di move*** **M**:
    - **M**(Ii,a)=Ij **sse**
      - $a \in first(Ii)=\{a \mid s::=_\Delta a\beta \in Ii\}$
      - $Ij = \mathbf{C}(\{s::=_\Delta \beta \mid s::=_\Delta a\beta \in Ii\})$
  - ***Stati Finali***: $\{Ij \mid \exists\, sj \in S, sj::=_\Delta \in Ij\}$

Automi di Items: un esempio

S::= digit digit*
F::= digit*.digit digit*



	digit	.
0	1	2
1	1	2
2	3	
3	3	