Teoria Compilării

Drăgulici Dumitru Daniel

Facultatea de matematică și informatică,
Universitatea București

2011

Bibliografie

- Aho A.V., Ullman J.D.:
  The Theory of Parsing, Translation and Compiling,
  Prentice Hall Inc., vol. I (1973), vol. II (1974)
- Aho A.V., Lam M.S., Sethi R., Ullman J.D.:
  Compilers. Principles, Techniques, & Tools
  Second Edition,
  Pearson Education Inc., 2006
- Adrian Atanasiu:
  Bazele matematice ale scrierii compilatoarelor,
  Ed. Olimp, 1996
- Adrian Atanasiu:
  Bazele informaticii, suport de curs pentru anul II seral,
  Tipogr. Univ. din București, 1987
- Internet:
  www.wikipedia.org
  www.scribd.com

Cuprins

Etapele compilării

Cod sursă (șir de caractere)

$\Downarrow$

$\boxed{\text{Analiza lexicală}}$

$\Downarrow$

Șir de tokeni

$\Downarrow$

$\boxed{\text{Analiza sintactică}}$

$\Downarrow$

Arbore de derivare

$\Downarrow$

$\boxed{\text{Analiza semantică}}$

$\Downarrow$

Arbore cu decorații

$\Downarrow$

$\boxed{\text{Generare cod}}$

$\Downarrow$

Cod obiect

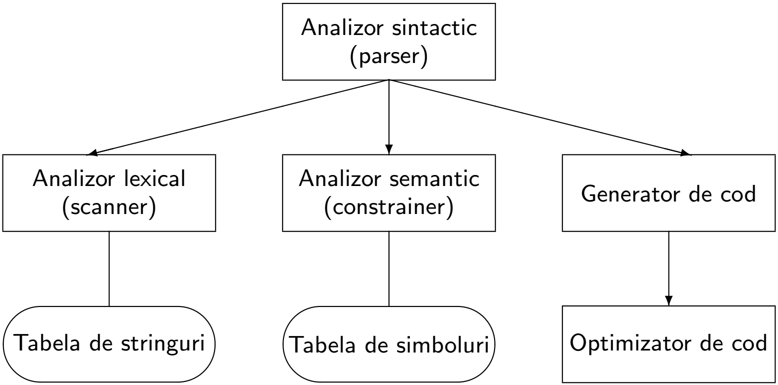
$\Downarrow$

$\boxed{\text{Optimizare cod}}$

$\Downarrow$

Cod obiect optimizat

Structura unui compilator



Cuprins

Analiza lexicală

Analiza lexicală (**scanarea**) are ca scop identificarea **tokenilor** (**unități lexicale**, **atomi lexicali**) succesivi din care este alcătuit codul sursă analizat.

Fiecare token are:

- un **tip** (**clasă**); exemple de tipuri de token în limbajele de programare uzuale:
    identificator: `ionescu`, `Nelu`, `_abc23x`;
    literal întreg: `123`, `0215`, `0x32f1`;
    literal flotant: `12.34`, `12.34e-5`, `12e5`;
    literal caracter: `'a'`, `'A'`, `'\n'`, `'\0'`;
    literal string: `"abc De12"`;
    operator: `+=`, `+`;
    separator: `;`, `{`, `}`;
    comentariu: `/*abc def*/`, `//abc def`;
    spațiu;
  (exemplele sunt din limbajele C/C++).
- o **valoare**; valoarea este secvența de caractere care formează tokenul.

De exemplu tokenul `_abc23x` are tipul *identificator* și valoarea șirul de caractere `_abc23x`.

Analiza lexicală

Analiza lexicală se realizează folosind un algoritm liniar, bazat pe instrumente de limbaje regulate.

O modalitate ușoară de lucru este să definim tipurile de token folosind expresii regulate și să recunoaștem tokenii folosind automate finite deterministe.

Expresii regulate

Definiție: **Expresii regulate** *(***ER***):*

- *Fie* V, $V' = \{+, {}^\star, \cdot, (,)\}$ *alfabete disjuncte.*

  Mulțimea $E(V)$ *a* ER *peste* V *se definește recursiv astfel:*

  1) $V \cup \{\lambda\} \cup \{\emptyset\} \subseteq E(V)$*;*

  2) dacă $\alpha, \beta \in E(V)$ *atunci* $\alpha + \beta$, $\alpha \cdot \beta$, $\alpha^\star$, $(\alpha) \in E(V)$*;*

  3) $E(V)$ *nu conține alte elemente.*

  Vom considera următoarea ordine a priorităților: $+ < \cdot <^\star$ *și vom omite uneori parantezele, dacă nu sunt necesare; de asemenea, uneori în loc de* $\cdot$ *vom folosi juxtapunerea.*

- *Fiecărei* ER $e \in E(V)$ *îi corespunde limbajul* $\overline{e} \subseteq V^\star$ *definit astfel:*

  1) $\overline{\emptyset} = \emptyset$*;*

  2) dacă $a \in V \cup \{\lambda\}$ *atunci* $\overline{a} = \{a\}$*;*

  3) dacă $\alpha, \beta \in E(V)$ *atunci* $\overline{\alpha + \beta} = \overline{\alpha} \cup \overline{\beta}$, $\overline{\alpha\beta} = \overline{\alpha}\,\overline{\beta}$, $\overline{\alpha^\star} = \overline{\alpha}^\star$, $\overline{(\alpha)} = \overline{\alpha}$.

Exemplu:

Fie $V = \{\underline{\texttt{A}}, \ldots, \underline{\texttt{Z}}, \underline{\texttt{a}}, \ldots, \underline{\texttt{z}}, \underline{\texttt{0}}, \ldots, \underline{\texttt{9}}, \underline{\texttt{+}}, \underline{\texttt{*}}, \underline{\texttt{/}}, \underline{\texttt{(}}, \underline{\texttt{)}}, \underline{\texttt{=}}, \underline{\texttt{;}}, \underline{\texttt{␣}}\}$.

Considerăm următoarele ER:

$idf = lit\ (\ lit + cif\ )^{\star}$

$int = cif\ cif^{\ \star}$

$com = \underline{\texttt{/*}}\ (\ notstar + \underline{\texttt{*}}\ notslash\ )^{\star}\ \underline{\texttt{*/}}$

$spa = \underline{\texttt{␣}}\ \underline{\texttt{␣}}^{\ \star}$

$op = \underline{\texttt{+}} + \underline{\texttt{*}} + \underline{\texttt{/}} + \underline{\texttt{(}} + \underline{\texttt{)}} + \underline{\texttt{=}}$

$delim = \underline{\texttt{;}}$

$lit = (\underline{\texttt{A}} + \ldots + \underline{\texttt{Z}} + \underline{\texttt{a}} + \ldots + \underline{\texttt{z}})$

$cif = (\underline{\texttt{0}} + \ldots + \underline{\texttt{9}})$

$notstar = (lit + cif + \underline{\texttt{+}} + \underline{\texttt{/}} + \underline{\texttt{(}} + \underline{\texttt{)}} + \underline{\texttt{=}} + \underline{\texttt{;}} + \underline{\texttt{␣}})$

$notslash = (lit + cif + \underline{\texttt{+}} + \underline{\texttt{*}} + \underline{\texttt{(}} + \underline{\texttt{)}} + \underline{\texttt{=}} + \underline{\texttt{;}} + \underline{\texttt{␣}})$

Presupunem că doar primele 6 ER definesc clase (tipuri) de tokeni (puteam să nu mai dăm un nume ultimelor ER şi să inserăm direct formula lor).

Expresii regulate

Exemplu:

Notăm clasele de token cu respectiv: *1* (*idf*), *2* (*int*), *3* (*com*), *4* (*spa*), *5* (*op*), *6* (*delim*).

Exemplu:

Notăm clasele de token cu respectiv: *1* (*idf*), *2* (*int*), *3* (*com*), *4* (*spa*), *5* (*op*), *6* (*delim*).

Fie codul sursă (cuvânt $w \in V^\star$):

```
abc=(12+x30)+1;␣␣y␣=abc;
```

Exemplu:

Notăm clasele de token cu respectiv: *1* (*idf*), *2* (*int*), *3* (*com*), *4* (*spa*), *5* (*op*), *6* (*delim*).

Fie codul sursă (cuvânt $w \in V^\star$):

`abc=(12+x30)+1;␣␣y␣=abc;`

Atunci, în urma analizei lexicale va rezulta următoarea **interpretare** a lui w:

⟨abc, *1*⟩ ⟨=, *5*⟩ ⟨(, *5*⟩ ⟨12, *2*⟩ ⟨+, *5*⟩ ⟨x30, *1*⟩ ⟨), *5*⟩ ⟨+, *5*⟩ ⟨1, *2*⟩ ⟨;, *6*⟩ ⟨␣␣, *4*⟩ ⟨y, *1*⟩ ⟨␣, *4*⟩ ⟨=, *5*⟩ ⟨abc, *1*⟩ ⟨;, *6*⟩

Exemplu:

Observaţii:

- În loc de ⟨abc, *1*⟩ *puteam găsi:* ⟨a, *1*⟩⟨b, *1*⟩⟨c, *1*⟩,
  sau: ⟨ab, *1*⟩⟨c, *1*⟩, etc.;

  în general dorim să identificăm cei mai lungi tokeni care se pot forma de la poziţia curentă încolo (o interpretare cu acestă proprietate s.n. **orientată către dreapta**);
- Pentru identificarea tokenilor ca mai sus se poate folosi un AFD.

Automate finite deterministe

Definiție: **Automate finite deterministe** *(AFD):*

- *Un* AFD *(parțial) este un sistem* $A = \langle Q, V, \delta, q_0, F\rangle$*, unde:*

  Q *este o mulțime finită, nevidă (mulțimea* stărilor*);*

  V *este o mulțime finită, nevidă (*alfabetul de intrare*);*

  $\delta : Q \times V \longrightarrow\!\!\!\!\!\!\circ\;\; Q$ *este o funcție parțială (*funcţia de tranziţie*);*

  $q_0 \in Q$ *(*starea inițială*);*

  $F \subseteq Q$ *(mulțimea* stărilor finale*).*

- *Extindem* δ *la o funcție parțială* $\hat{\delta} : Q \times V^\star \longrightarrow\!\!\!\!\!\!\circ\;\; Q$ *astfel:*

  $\hat{\delta}(q, \lambda) = q$ *pentru orice* $q \in Q$*,*

  $\hat{\delta}(q, \alpha a) = \delta(\hat{\delta}(q, \alpha), a)$*, dacă există* $\hat{\delta}(q, \alpha)$ *și* $\delta(\hat{\delta}(q, \alpha), a)$*.*

  Notăm $\hat{\delta}$ *tot cu* δ*. Va rezulta:*

  $\delta(q, \alpha\beta) = \delta(\delta(q, \alpha), \beta)$ *pentru orice* $q \in Q$ *și* $\alpha, \beta \in V^\star$*.*

- Limbajul recunoscut *de* A *este următoarea submulțime a lui* $V^\star$*:*

  $L(A) = \{\alpha \in V^\star \,:\, \delta(q_0, \alpha) \in F\}$*.*

- *Un* AFD A *este* total*, dacă* δ *este total definită (ca funcție* $\delta : Q \times V \longrightarrow Q$*); va rezulta că și* $\hat{\delta}$ *este total definită (ca funcție* $\hat{\delta} : Q \times V^\star \longrightarrow Q$*).*

Automate finite deterministe

Observație:

Pentru orice AFD A există un AFD total A' echivalent cu el, adică a.î. $L(A) = L(A')$.

El se poate construi din A adăugând o nouă stare nefinală și ducând în ea toate tranzițiile nedefinite, inclusiv tranziții de la ea la ea însăși, cu toate simbolurile alfabetului.

Automate finite deterministe

Un AFD se poate reprezenta printr-o diagramă de tip graf orientat, unde:

- vârfurile reprezintă stări; se desenează sub forma unor discuri ce conțin o etichetă de identificare a vârfului; starea inițială are o săgeată de intrare, stările finale au conturul dublat (sau o săgeată de ieșire);

- arcele reprezintă tranziții; se desenează prin săgeți ce unesc stările sursă și destinație corespunzătoare, în dreptul cărora se scrie simbolul cu care se face tranziția; pentru simplitate, dacă mai multe tranziții au aceeași sursă și destinație, ele se pot desena printr-o singură săgeată, în dreptul căreia se scriu toate simbolurile asociate lor (sau o mulțime, ER, etc.).

Automate finite deterministe

Propoziție:

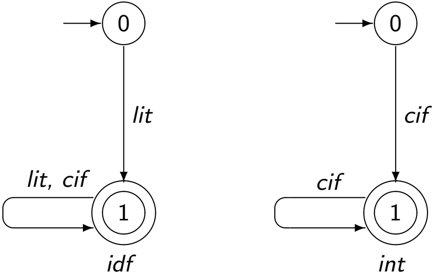
Pentru orice limbaj $L \subseteq V^{\star}$ sunt echivalente:
1) Există un AFD A cu alfabetul V a.î. $L = L(A)$;
2) Există o ER $e \in E(V)$ a.î. $L = \overline{e}$.

Un limbaj ce satisface oricare din proprietățile echivalente de mai sus s.n. **limbaj regulat**.
Există și alte proprietăți echivalente cu cele două, anume:
- L este recunoscut de un automat finit nedeterminist (AFN);
- L este recunoscut de un automat finit nedeterminist cu λ-tranziții (AFNL);
- L este generat de o gramatică regulată (GR);

(a se vedea cursul de limbaje formale, anul I).

Automate finite deterministe

De exemplu, AFD echivalente cu primele două ER din exemplul anterior sunt:



(prin "*lit*", "*cif*" și "*lit, cif*" am notat acum totalitatea simbolurilor din V descrise de respectiv aceste ER).
Riguros, trecerea de la ER la AFD echivalent se face folosind un algoritm ...

Algoritm ER $\longrightarrow$ AFD

Algoritm (ER $\longrightarrow$ AFD):

Intrare: $e \in E(V)$.
Ieșire: $A = \langle Q, V, \delta, q_0, F \rangle$ AFD a.î. $L(A) = \overline{e}$.

Notație:
Considerăm un simbol nou $\sharp \notin V$.
Numim **extensia** lui e expresia regulată $(e)\sharp \in E(V \cup \{\sharp\})$.

Algoritm ER $\longrightarrow$ AFD

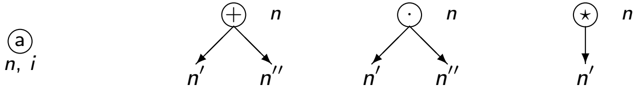
Pasul 1:

Se construiește arborele asociat ER extinse; în acest arbore nu vom mai considera și nodurile asociate parantezelor (nu mai sunt necesare).
Nodurile neterminale ale arborelui vor corespunde unor operatori $+$, $\cdot$ sau $\star$; cele terminale vor corespunde unor simboluri din $V \cup \{\lambda\} \cup \{\sharp\}$.

Asociem fiecărui nod (neterminal sau terminal) un identificator distinct, de ex. $n1$, $n2$, etc.
Asociem frunzelor, de la stânga la dreapta, numere întregi consecutive, numărând de la 1; aceste numere se vor numi **poziții**.
Obs: frunze diferite vor avea asociate numere (poziții) diferite, chiar dacă simbolul din alfabet corespunzător este aceeași.

Deci nodurile arborelui vor fi de forma:

$a \in V \cup \{\lambda\}$ $\qquad$ $e_1 + e_2$ $\qquad$ $e_1 \cdot e_2$ $\qquad$ $e^\star$



(n, n', n'' sunt identificatori, i sunt poziții (doar în cazul frunzelor)).

Algoritm ER $\longrightarrow$ AFD

Pasul 2:

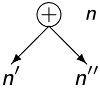
Se calculează mulțimile:

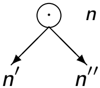
$\mathit{firstpos}(n)$, $\mathit{lastpos}(n)$, $\mathit{nullable}(n)$, pentru fiecare nod n;

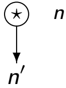
$\mathit{followpos}(i)$, pentru fiecare poziție i.

Algoritm ER $\longrightarrow$ AFD

Mai întâi construim mulțimile $firstpos(n)$, $lastpos(n)$, $nullable(n)$, astfel:

tip nod n	$firstpos(n)$	$lastpos(n)$	$nullable(n)$
a $a \in V \cup \{\lambda\}$ n, i	$\{i\}$	$\{i\}$	$true$ d.d $a = \lambda$
 $\oplus$ n n' n''	$firstpos(n')$ $\cup firstpos(n'')$	$lastpos(n')$ $\cup lastpos(n'')$	$nullable(n')$ or $nullable(n'')$

 $\cdot$ n
 n' n''
	if $nullable(n')$ then
	 $firstpos(n')$
	 $\cup firstpos(n'')$
	 else
	 $firstpos(n')$
		if $nullable(n'')$ then
		 $lastpos(n')$
		 $\cup lastpos(n'')$
		 else
		 $firstpos(n'')$
			$nullable(n')$
			 and
			 $nullable(n'')$

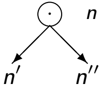
 $\star$ n n'	$firstpos(n')$	$lastpos(n')$	true

Algoritm ER $\longrightarrow$ AFD

Apoi construim mulțimile *followpos*(i), astfel:

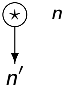
- pentru orice poziție i, inițializez *followpos*$(i) = \emptyset$;

- pentru orice nod



și orice $i \in$ *lastpos*(n'), reunesc *firstpos*(n'') la *followpos*(i);

- pentru orice nod



și orice $i \in$ *lastpos*(n'), reunesc *firstpos*(n') la *followpos*(i).

Algoritm ER $\longrightarrow$ AFD

Observație:

- *firstpos*(n)/*lastpos*(n) este mulţimea poziţiilor pe care le poate avea primul simbol/ultimul simbol al unui şir generat de subarborele cu rădăcina n;
nullable(n)=*true* d.d. subarborele cu rădăcina n poate genera şirul vid.
followpos(i) este mulţimea poziţiilor pe care le pot avea simbolurile ce urmează simbolului de pe poziţia i într-un şir generat de arborele ER;

- toate mulţimile *firstpos*, *lastpos*, *nullable*, *followpos* se pot calcula printr-o singură parcurgere în adâncime a arborelui (la paşii de revenire).

Algoritm ER $\longrightarrow$ AFD

Pasul 3:

Se construiește $A = \langle Q, V, \delta, q_0, F\rangle$ AFD a.î. $L(A) = \overline{e}$, astfel:

- elementele lui Q sunt mulțimi de poziții;
- $q_0 = \mathit{firspos}(\text{rădăcină})$;
- Q, F și δ se construiesc astfel:

```
Q ← {firstpos(rădăcină)}, firstpos(rădăcină) stare nemarcată;
if [firstpos(rădăcină) conține poziția lui ♯] then F ← {firstpos(rădăcină)}
                                              else F ← ∅;
while [există T ∈ Q nemarcată] do begin
        marchează T;
        for [fiecare a ∈ V] do begin
              U ← {j ∈ followpos(i) : i ∈ T, i etichetat cu a};
              if U ∉ Q then begin
                    Q ← Q ∪ {U}; U stare nemarcată;
                    if [U conține poziția lui ♯] then F ← F ∪ {U}
              end;
              δ(T, a) ← U
        end
end
```

Algoritm ER $\longrightarrow$ AFD

Exemplu:

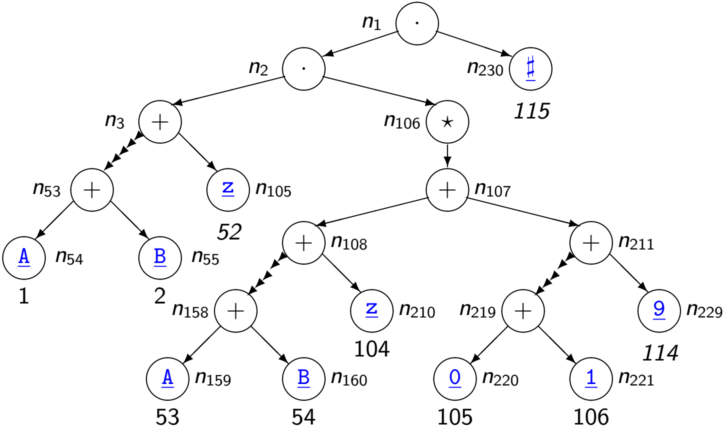
Să aplicăm algoritmul pentru ER *idf* din exemplul anterior.

Expandând ER îmbricate şi considerând ER extinsă, aceasta se scrie:
$idf = ((\underline{\mathtt{A}} + \ldots + \underline{\mathtt{Z}} + \underline{\mathtt{a}} + \ldots + \underline{\mathtt{z}})\ ((\underline{\mathtt{A}} + \ldots + \underline{\mathtt{Z}} + \underline{\mathtt{a}} + \ldots + \underline{\mathtt{z}}) + (\underline{\mathtt{0}} + \ldots + \underline{\mathtt{9}}))^{\star})\underline{\sharp}$

Construim arborele asociat ER extinse, numim nodurile şi numerotăm poziţiile:

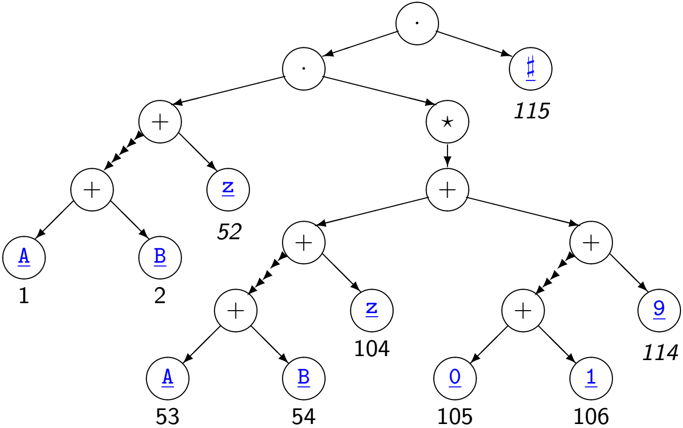
Algoritm ER $\longrightarrow$ AFD

Exemplu:



Algoritm ER $\longrightarrow$ AFD

Exemplu:

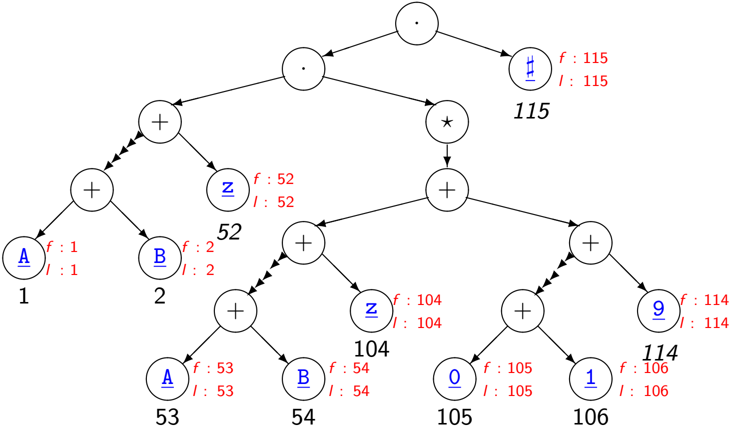


Calculăm mulţimile *firstpos* (f), *lastpos* (l), *nullable* (vom nota doar nodurile pentru cre *nullable* = *true*, scriind $n : t$), *followpos*, facând o parcurgere în adâncime a arborelui (când lucrăm direct pe desen nu mai sunt necesari identificatorii vârfurilor şi i-am omis).

Vom scrie cu roşu valorile nou adăugate la fiecare pas.

Algoritm ER $\longrightarrow$ AFD

Exemplu:



followpos(1):

...

followpos(52):

followpos(53):

...

followpos(104):

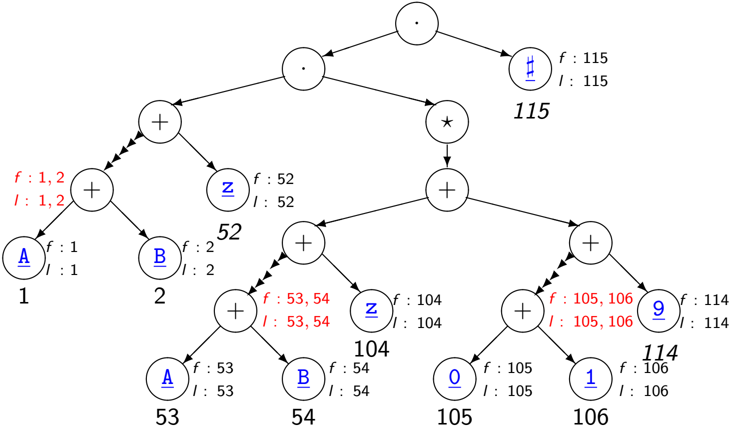
followpos(105):

...

followpos(114):

Algoritm ER $\longrightarrow$ AFD

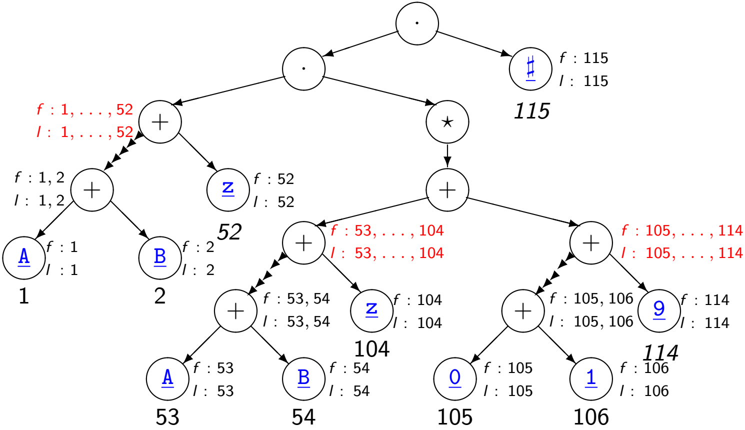
Exemplu:



followpos(1):	*followpos*(53):	*followpos*(105):
...	...	...
followpos(52):	*followpos*(104):	*followpos*(114):

Algoritm ER $\longrightarrow$ AFD

Exemplu:



followpos(1):

. . .

followpos(52):

followpos(53):

. . .

followpos(104):

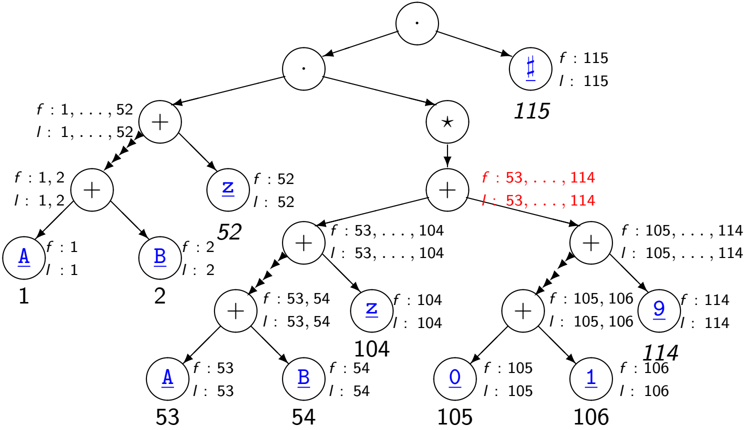
followpos(105):

. . .

followpos(114):

Exemplu:

Algoritm ER $\longrightarrow$ AFD



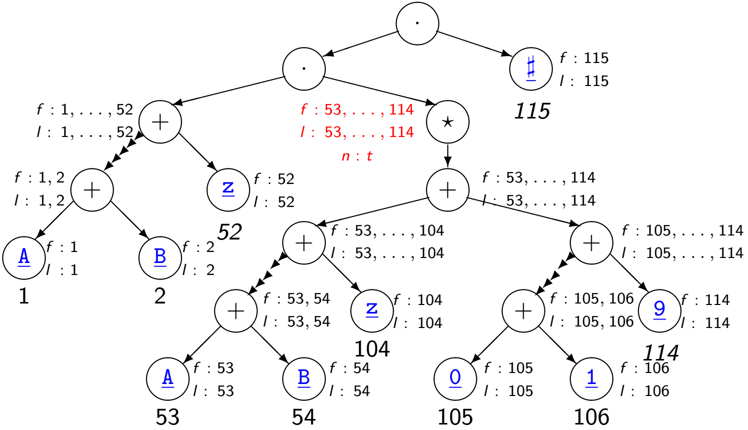
followpos(1): . . . *followpos*(52):

followpos(53): . . . *followpos*(104):

followpos(105): . . . *followpos*(114):

Algoritm ER $\longrightarrow$ AFD

Exemplu:



followpos(1):

. . .

followpos(52):

followpos(53): 53, . . . , 114

. . .

followpos(104): 53, . . . , 114

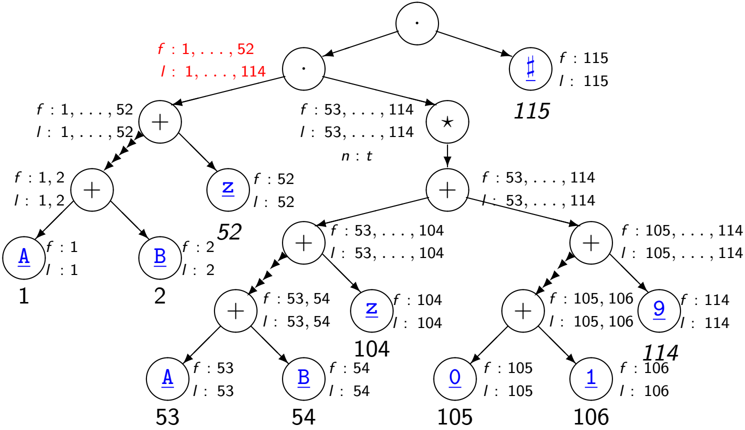
followpos(105): 53, . . . , 114

. . .

followpos(114): 53, . . . , 114

Algoritm ER $\longrightarrow$ AFD

Exemplu:



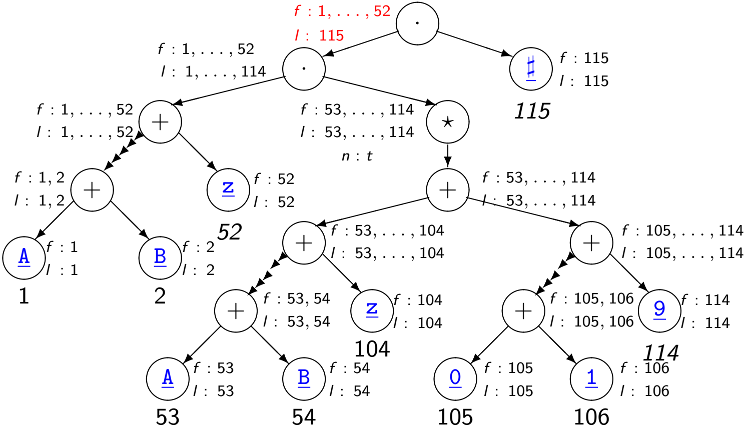
followpos(1): $53,\ldots,114$ | *followpos*(53): $53,\ldots,114$ | *followpos*(105): $53,\ldots,114$

…

followpos(52): $53,\ldots,114$ | *followpos*(104): $53,\ldots,114$ | *followpos*(114): $53,\ldots,114$

Algoritm ER $\longrightarrow$ AFD

Exemplu:



followpos(1): 53, . . . , 114, 115 *followpos*(53): 53, . . . , 114, 115 *followpos*(105): 53, . . . , 114, 115

. . . .

followpos(52): 53, . . . , 114, 115 *followpos*(104): 53, . . . , 114, 115 *followpos*(114): 53, . . . , 114, 115

Algoritm ER $\longrightarrow$ AFD

Din calculele de mai sus este suficient să reţinem că
$firstpos(\text{rădăcină}) = \{1, \ldots, 52\}$
$followpos(i) = 53, \ldots, 115$ pentru orice i
poziţia lui $\sharp$ este 115

Construim automatul, mai exact q_0, Q, F, δ:

- avem $q_0 = \{1, \ldots, 52\}$
- iniţializăm Q, F, δ cu $\emptyset$, apoi avem succesiv (am scris cu roşu starile şi tranziţiile noi adăugate, am subliniat stările marcate, iar la ultima stare marcată am consemnat deasupra simbolurile corespunzătoare poziţiilor):

Q :

F :

δ :

Exemplu:

Din calculele de mai sus este suficient să reţinem că
$firstpos$(rădăcină) $= \{1, \ldots, 52\}$
$followpos(i) = 53, \ldots, 115$ pentru orice i
poziţia lui $\sharp$ este 115

Construim automatul, mai exact q_0, Q, F, δ:

- avem $q_0 = \{1, \ldots, 52\}$
- iniţializăm Q, F, δ cu $\emptyset$, apoi avem succesiv (am scris cu roşu starile şi tranziţiile noi adăugate, am subliniat stările marcate, iar la ultima stare marcată am consemnat deasupra simbolurile corespunzătoare poziţiilor):

Q : $\{1, \ldots, 52\}$

F :

δ :

Exemplu:

Din calculele de mai sus este suficient să reţinem că

$firstpos$(rădăcină) $= \{1, \ldots, 52\}$
$followpos(i) = 53, \ldots, 115$ pentru orice i
poziţia lui $\underline{\sharp}$ este 115

Construim automatul, mai exact q_0, Q, F, δ:

- avem $q_0 = \{1, \ldots, 52\}$
- iniţializăm Q, F, δ cu $\emptyset$, apoi avem succesiv (am scris cu roşu starile şi tranziţiile noi adăugate, am subliniat stările marcate, iar la ultima stare marcată am consemnat deasupra simbolurile corespunzătoare poziţiilor):

$$Q : \underline{\overset{\underline{\mathtt{A}} \;\ldots\; \underline{\mathtt{z}}}{\{1, \ldots, 52\}}}, \{53, \ldots, 115\}$$

$$F : \{53, \ldots, 115\}$$

$$\delta : \{1, \ldots, 52\} \underset{\underline{\mathtt{A}}, \ldots, \underline{\mathtt{z}}}{\longrightarrow} \{53, \ldots, 115\}$$

Exemplu:

Din calculele de mai sus este suficient să reţinem că

$\mathit{firstpos}(\text{rădăcină}) = \{1, \ldots, 52\}$

$\mathit{followpos}(i) = 53, \ldots, 115$ pentru orice i

poziţia lui $\underline{\sharp}$ este 115

Construim automatul, mai exact q_0, Q, F, δ:

- avem $q_0 = \{1, \ldots, 52\}$
- iniţializăm Q, F, δ cu $\emptyset$, apoi avem succesiv (am scris cu roşu starile şi tranziţiile noi adăugate, am subliniat stările marcate, iar la ultima stare marcată am consemnat deasupra simbolurile corespunzătoare poziţiilor):

$$Q: \underline{\{1, \ldots, 52\}}, \underline{\{\overset{\mathtt{A}}{53}, \ldots, \overset{\mathtt{z}}{104}, \overset{\mathtt{0}}{105}, \ldots, \overset{\mathtt{9}}{114}, \overset{\sharp}{115}\}}$$

$$F: \{53, \ldots, 115\}$$

$$\delta: \{1, \ldots, 52\} \xrightarrow[\mathtt{A}, \ldots, \mathtt{z}]{} \{53, \ldots, 115\}, \ \{53, \ldots, 115\} \xrightarrow[\mathtt{A}, \ldots, \mathtt{z},\ \mathtt{0}, \ldots, \mathtt{9}]{} \{53, \ldots, 115\}$$

Algoritm ER $\longrightarrow$ AFD

Exemplu:

Numerotând starile: $\{1, \ldots, 52\} = 0$, $\{53, \ldots, 115\} = 1$, avem automatul definit prin:

$V = \{\underline{\texttt{A}}, \ldots, \underline{\texttt{z}}, \underline{\texttt{0}}, \ldots, \underline{\texttt{9}}\}$
$Q = \{0, 1\}$
$q_0 = 0$
$F = \{1\}$

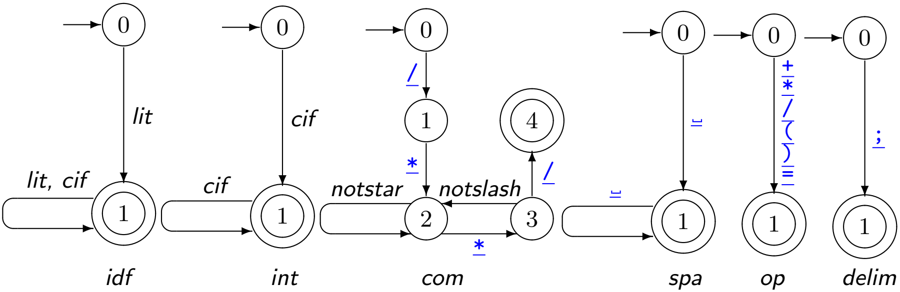
δ	A	...	z	0	...	9
0	1	...	1	1	...	1
1	1	...	1	1	...	1

adică tocmai automatul desenat mai înainte.

Obs: pentru a uşura lucrul cu grupuri numeroase de simboluri ale alfabetului folosite la fel (şi a evita notaţiile cu "..."), se pot introduce **clase** de simboluri (în exemplele noastre: *lit*, *cif*, *notstar*, *notslash*).

Exemplu:

Procedând în acelaşi fel, obţinem următoarele AFD echivalente cu respectiv cele 6 ER ce defineau clasele de tokeni în exemplele anterioare (exerciţiu):



(prin "*lit*", "*cif*" şi "*lit, cif*" am notat şi acum totalitatea simbolurilor din V descrise de respectiv aceste ER).

Analiza lexicală

Uneori AFD obținute prin diverse procedee pot fi micșorate, în sensul reducerii numărului de stări (nu e cazul celor 6 AFD de mai înainte).

În acest scop se poate aplica un algoritm de minimizare AFD (a se vedea cursul de limbaje formale, anul I).

Analiza lexicală

Odată construit câte un AFD pentru fiecare tip de token, putem face analiza lexicală folosind în paralel aceste AFD, sau putem folosi un AFD unic (construit din AFD respective), pentru care am asociat câte un tip de token fiecărei stări finale.

AFD unic se poate construi întotdeauna, în baza următoarelor rezultate și algoritmi:

Propoziție:

Clasa limbajelor regulate este închisă la reuniunea finită, mai exact: Dacă $L_1, \ldots, L_n \subseteq V^\star$ sunt limbaje regulate ($n \in \mathbf{N}$, $n \geq 1$), atunci $L_1 \cup \cdots \cup L_n$ este limbaj regulat.

Demonstrația se face considerând câte un instrument de limbaje regulate (AFD, AFN, AFNL, ER, GR) pentru fiecare dintre $L_1, \ldots, L_n$ și construind un alt instrument de limbaje regulate (AFD, AFN, AFNL, ER, GR) despre care se demonstrează că descrie $L_1 \cup \cdots \cup L_n$ (a se vedea cursul de limbaje formale, anul I).
În funcție de alegerile făcute, se obțin diverse demonstrații.
De exemplu din n AFD-uri se construiește un alt AFD; sau din n AFD-uri se construiește un AFNL (e mai ușor), etc.

Ne va fi utilă următoarea variantă a propoziției anterioare:

Propoziție:

Fiind date AFD totale $A_i = \langle Q_i,\ V,\ \delta_i,\ q_0^i,\ F_i\rangle$, $i = 1, \ldots, n$, *a.î.* $L(A_i) \cap L(A_j) = \emptyset$ *pentru orice* $i \neq j$, *există un AFD* $A = \langle Q,\ V,\ \delta,\ q_0,\ F\rangle$ *și o funcție* $tok : F \longrightarrow \{1, \ldots, n\}$ *a.î.:*

- $L(A) = L(A_1) \cup \cdots \cup L(A_n)$;
- *pentru orice* $\alpha \in V^\star$ *și* $i \in \{1, \ldots, n\}$ *avem* $\alpha \in L(A_i)$ *d.d.* $\delta(q_0, \alpha) \in F$ *și* $tok(\delta(q_0, \alpha)) = i$.

Algoritmul de construcție a lui A este următorul (demonstrațiile sunt simple - exercițiu):

Algoritm reuniune disjunctă AFD

$Q \leftarrow \{\langle q_0^1, \ldots, q_0^n \rangle\}$, $\langle q_0^1, \ldots, q_0^n \rangle$ stare nemarcată;
$q_0 \leftarrow \langle q_0^1, \ldots, q_0^n \rangle$;
if [există $i \in \{1, \ldots, n\}$ a.î. $q_0^i \in F_i$] then begin $F \leftarrow \{q_0\}$; $tok(q_0) \leftarrow i$ end
else $F \leftarrow \emptyset$;
while [există $q = \langle q^1, \ldots, q^n \rangle \in Q$ nemarcată] do begin
  marchează q;
  for [fiecare $a \in V$] do begin
    $r \leftarrow \langle \delta_1(q^1, a), \ldots, \delta_n(q^n, a) \rangle \overset{not}{=} \langle r^1, \ldots, r^n \rangle$;
    if $r \notin Q$ then begin
      $Q \leftarrow Q \cup \{r\}$; r stare nemarcată;
      if [există $i \in \{1, \ldots, n\}$ a.î. $r^i \in F_i$] then begin
        $F \leftarrow F \cup \{r\}$; $tok(r) \leftarrow i$
      end
    end;
    $\delta(q, a) \leftarrow r$
  end
end

Obs: faptul că $L(A_i) \cap L(A_j) = \emptyset$ pentru orice $i \neq j$ garantează că mai sus întotdeauna va exista cel mult un $i \in \{1, \ldots, n\}$ a.î. $q_0^i \in F_i$, respectiv $r^i \in F_i$, și astfel funcția *tok* este bine definită.

Algoritm reuniune disjunctă AFD

Exemplu:

Să aplicăm algoritmul celor 6 AFD ce defineau clase de tokeni în exemplele anterioare. Pentru aceasta le completăm mai întâi la nişte AFD totale:

lit (1)

δ	*lit*	*cif*	*altesb*
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	*cif*	*altesb*
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	/	*	$V \setminus \{/, *\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	␣	*altesb*
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	+,*,/,(,),=	*altesb*
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	;	*altesb*
0	1	2
1	2	2
2	2	2

F: 1

Am scris automatele mai compact: din fiecare tabel se deduce mulţimea stărilor şi funcţia de tranziţie, starea iniţială este *0*, cele finale sunt scrise dedesubt, "*lit*", "*cif*", "*notstar*", "*notslash*" desemnează mulţimile respective de simboluri, iar "*altesb*" desemnează în cazul fiecărui automat mulţimea simbolurilor din *V* neacoperite de celelalte coloane. De asemenea, în dreptul numelui fiecărui automat am scris între paranteze numărul clasei de tokeni pe care o defineşte.

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	lit	cif	altesb
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	cif	altesb
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	/	*	$V \setminus \{/, *\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	␣	altesb
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	+,*,/,(,),=	altesb
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	;	altesb
0	1	2
1	2	2
2	2	2

F: 1

Avem (am numerotat starile noului AFD începând de la 0, am scris cu roşu starile şi tranziţiile noi adăugate, am subliniat stările marcate şi am scris între [ ] valoarea *tok* pentru fiecare stare finală):

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	lit	cif	altesb
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	cif	altesb
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	/	*	$V \setminus \{/, *\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	␣	altesb
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	+,*,/,(,),=	altesb
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	;	altesb
0	1	2
1	2	2
2	2	2

F: 1

Q : $\langle 0, 0, 0, 0, 0, 0 \rangle = 0$

$q_0 = 0$;

F :

δ :

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	lit	cif	altesb
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	cif	altesb
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	$\underline{/}$	$\underline{*}$	$V \setminus \{\underline{/}, \underline{*}\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	$\underline{␣}$	altesb
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	$\underline{+}, \underline{*}, \underline{/}, \underline{(}, \underline{)}, \underline{=}$	altesb
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	$\underline{;}$	altesb
0	1	2
1	2	2
2	2	2

F: 1

$Q:\ \underline{\langle 0,0,0,0,0,0\rangle = 0}, \langle 1,2,5,2,2,2\rangle = 1, \langle 2,1,5,2,2,2\rangle = 2,$
$\langle 2,2,1,2,1,2\rangle = 3, \langle 2,2,5,2,1,2\rangle = 4, \langle 2,2,5,1,2,2\rangle = 5,$
$\langle 2,2,5,2,2,1\rangle = 6$

$q_0 = 0;$

$F:\ 1\ [1],\ 2\ [2],\ 3\ [5],\ 4\ [5],\ 5\ [4],\ 6\ [6]$

$\delta:\ 0 \xrightarrow{lit} 1,\ 0 \xrightarrow{cif} 2,\ 0 \xrightarrow{\underline{/}} 3,\ 0 \xrightarrow{\underline{*}} 4,\ 0 \xrightarrow{\underline{␣}} 5,\ 0 \xrightarrow{\underline{+},\underline{(},\underline{)},\underline{=}} 4,\ 0 \xrightarrow{\underline{;}} 6$

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	lit	cif	altesb
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	cif	altesb
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	/	*	$V \setminus \{/, *\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	␣	altesb
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	+,*,/,(,),=	altesb
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	;	altesb
0	1	2
1	2	2
2	2	2

F: 1

$Q: \ \langle 0,0,0,0,0,0\rangle = 0,\ \langle 1,2,5,2,2,2\rangle = 1,\ \langle 2,1,5,2,2,2\rangle = 2,$
$\langle 2,2,1,2,1,2\rangle = 3,\ \langle 2,2,5,2,1,2\rangle = 4,\ \langle 2,2,5,1,2,2\rangle = 5,$
$\langle 2,2,5,2,2,1\rangle = 6,\ \langle 2,2,5,2,2,2\rangle = 7$

$q_0 = 0;$

$F: \ 1\ [1],\ 2\ [2],\ 3\ [5],\ 4\ [5],\ 5\ [4],\ 6\ [6]$

$\delta: \ 0 \xrightarrow{lit} 1,\ 0 \xrightarrow{cif} 2,\ 0 \xrightarrow{/} 3,\ 0 \xrightarrow{␣} 5,\ 0 \xrightarrow{+,*,(,),=} 4,\ 0 \xrightarrow{;} 6,$

$1 \xrightarrow{lit, cif} 1,\ 1 \xrightarrow{V \setminus (lit \cup cif)} 7$

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	lit	cif	altesb
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	cif	altesb
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	/	*	$V \setminus \{/, *\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	␣	altesb
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	+,*,/,(,),=	altesb
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	;	altesb
0	1	2
1	2	2
2	2	2

F: 1

$Q: \langle 0,0,0,0,0,0\rangle = 0,\ \langle 1,2,5,2,2,2\rangle = 1,\ \langle 2,1,5,2,2,2\rangle = 2,$
$\langle 2,2,1,2,1,2\rangle = 3,\ \langle 2,2,5,2,1,2\rangle = 4,\ \langle 2,2,5,1,2,2\rangle = 5,$
$\langle 2,2,5,2,2,1\rangle = 6,\ \langle 2,2,5,2,2,2\rangle = 7$

$q_0 = 0;$

$F: 1\ [1],\ 2\ [2],\ 3\ [5],\ 4\ [5],\ 5\ [4],\ 6\ [6]$

$\delta: 0 \xrightarrow{lit} 1,\ 0 \xrightarrow{cif} 2,\ 0 \xrightarrow{/} 3,\ 0 \xrightarrow{␣} 5,\ 0 \xrightarrow{+,*,(,),=} 4,\ 0 \xrightarrow{;} 6,$
$1 \xrightarrow{lit,cif} 1,\ 1 \xrightarrow{V\setminus(lit\cup cif)} 7,\ 2 \xrightarrow{cif} 2,\ 2 \xrightarrow{V\setminus cif} 7$

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	lit	cif	altesb
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	cif	altesb
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	$\underline{/}$	$\underline{*}$	$V \setminus \{\underline{/}, \underline{*}\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	$\underline{␣}$	altesb
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	$\underline{+},\underline{*},\underline{/},\underline{(},\underline{)},\underline{=}$	altesb
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	$\underline{;}$	altesb
0	1	2
1	2	2
2	2	2

F: 1

$Q:\ \langle 0,0,0,0,0,0\rangle = 0,\ \langle 1,2,5,2,2,2\rangle = 1,\ \langle 2,1,5,2,2,2\rangle = 2,$
$\langle 2,2,1,2,1,2\rangle = 3,\ \langle 2,2,5,2,1,2\rangle = 4,\ \langle 2,2,5,1,2,2\rangle = 5,$
$\langle 2,2,5,2,2,1\rangle = 6,\ \langle 2,2,5,2,2,2\rangle = 7,\ \langle 2,2,2,2,2,2\rangle = 8$

$q_0 = 0;$

$F:\ 1\ [1],\ 2\ [2],\ 3\ [5],\ 4\ [5],\ 5\ [4],\ 6\ [6]$

$\delta:\ 0 \xrightarrow{lit} 1,\ 0 \xrightarrow{cif} 2,\ 0 \xrightarrow{\underline{/}} 3,\ 0 \xrightarrow{\underline{␣}} 5,\ 0 \xrightarrow{\underline{+},\underline{*},\underline{(},\underline{)},\underline{=}} 4,\ 0 \xrightarrow{\underline{;}} 6,$
$1 \xrightarrow{lit,cif} 1,\ 1 \xrightarrow{V\setminus(lit\cup cif)} 7,\ 2 \xrightarrow{cif} 2,\ 2 \xrightarrow{V\setminus cif} 7,\ 3 \xrightarrow{\underline{*}} 8,\ 3 \xrightarrow{V\setminus\{\underline{*}\}} 7$

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	*lit*	*cif*	*altesb*
0	*1*	*2*	*2*
1	*1*	*1*	*2*
2	*2*	*2*	*2*

F: 1

int (2)

δ	*cif*	*altesb*
0	*1*	*2*
1	*1*	*2*
2	*2*	*2*

F: 1

com (3)

δ	/	*	$V \setminus \{/, *\}$
0	*1*	*5*	*5*
1	*5*	*2*	*5*
2	*2*	*3*	*2*
3	*4*	*2*	*2*
4	*5*	*5*	*5*
5	*5*	*5*	*5*

F: 4

spa (4)

δ	␣	*altesb*
0	*1*	*2*
1	*1*	*2*
2	*2*	*2*

F: 1

op (5)

δ	+,*,/,(,),=	*altesb*
0	*1*	*2*
1	*2*	*2*
2	*2*	*2*

F: 1

delim (6)

δ	;	*altesb*
0	*1*	*2*
1	*2*	*2*
2	*2*	*2*

F: 1

$$Q: \ \langle 0,0,0,0,0,0\rangle = 0,\ \langle 1,2,5,2,2,2\rangle = 1,\ \langle 2,1,5,2,2,2\rangle = 2,$$
$$\langle 2,2,1,2,1,2\rangle = 3,\ \langle 2,2,5,2,1,2\rangle = 4,\ \langle 2,2,5,1,2,2\rangle = 5,$$
$$\langle 2,2,5,2,2,1\rangle = 6,\ \langle 2,2,5,2,2,2\rangle = 7,\ \langle 2,2,2,2,2,2\rangle = 8$$

$q_0 = 0;$

$F: \ 1\ [1],\ 2\ [2],\ 3\ [5],\ 4\ [5],\ 5\ [4],\ 6\ [6]$

$$\delta: \ 0 \xrightarrow{lit} 1,\ 0 \xrightarrow{cif} 2,\ 0 \xrightarrow{/} 3,\ 0 \xrightarrow{␣} 5,\ 0 \xrightarrow{+,*,(,),=} 4,\ 0 \xrightarrow{;} 6,$$
$$1 \xrightarrow{lit,cif} 1,\ 1 \xrightarrow{V\setminus(lit\cup cif)} 7,\ 2 \xrightarrow{cif} 2,\ 2 \xrightarrow{V\setminus cif} 7,\ 3 \xrightarrow{*} 8,\ 3 \xrightarrow{V\setminus\{*\}} 7,\ 4 \xrightarrow{V} 7$$

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	lit	cif	altesb
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	cif	altesb
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	/	*	$V \setminus \{/, *\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	␣	altesb
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	+,*,/,(,),=	altesb
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	;	altesb
0	1	2
1	2	2
2	2	2

F: 1

$$Q: \langle 0,0,0,0,0,0\rangle = 0,\ \langle 1,2,5,2,2,2\rangle = 1,\ \langle 2,1,5,2,2,2\rangle = 2,$$
$$\langle 2,2,1,2,1,2\rangle = 3,\ \langle 2,2,5,2,1,2\rangle = 4,\ \langle 2,2,5,1,2,2\rangle = 5,$$
$$\langle 2,2,5,2,2,1\rangle = 6,\ \langle 2,2,5,2,2,2\rangle = 7,\ \langle 2,2,2,2,2,2\rangle = 8$$

$$q_0 = 0;$$

$$F:\ 1\ [1],\ 2\ [2],\ 3\ [5],\ 4\ [5],\ 5\ [4],\ 6\ [6]$$

$$\delta:\ 0 \xrightarrow{lit} 1,\ 0 \xrightarrow{cif} 2,\ 0 \xrightarrow{/} 3,\ 0 \xrightarrow{␣} 5,\ 0 \xrightarrow{+,*,(,),=} 4,\ 0 \xrightarrow{;} 6,$$
$$1 \xrightarrow{lit,cif} 1,\ 1 \xrightarrow{V\setminus(lit\cup cif)} 7,\ 2 \xrightarrow{cif} 2,\ 2 \xrightarrow{V\setminus cif} 7,\ 3 \xrightarrow{*} 8,\ 3 \xrightarrow{V\setminus\{*\}} 7,\ 4 \xrightarrow{V} 7,$$
$$5 \xrightarrow{␣} 5,\ 5 \xrightarrow{V\setminus\{␣\}} 7$$

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	*lit*	*cif*	*altesb*
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	*cif*	*altesb*
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	$\underline{/}$	$\underline{*}$	$V \setminus \{\underline{/}, \underline{*}\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	$\underline{␣}$	*altesb*
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	$\underline{+},\underline{*},\underline{/},\underline{(},\underline{)},\underline{=}$	*altesb*
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	$\underline{;}$	*altesb*
0	1	2
1	2	2
2	2	2

F: 1

$$Q:\ \underline{\langle 0,0,0,0,0,0\rangle = 0},\ \underline{\langle 1,2,5,2,2,2\rangle = 1},\ \underline{\langle 2,1,5,2,2,2\rangle = 2},$$
$$\underline{\langle 2,2,1,2,1,2\rangle = 3},\ \underline{\langle 2,2,5,2,1,2\rangle = 4},\ \underline{\langle 2,2,5,1,2,2\rangle = 5},$$
$$\underline{\langle 2,2,5,2,2,1\rangle = 6},\ \langle 2,2,5,2,2,2\rangle = 7,\ \langle 2,2,2,2,2,2\rangle = 8$$

$$q_0 = 0;$$

$$F:\ 1\ [1],\ 2\ [2],\ 3\ [5],\ 4\ [5],\ 5\ [4],\ 6\ [6]$$

$$\delta:\ 0 \xrightarrow{lit} 1,\ 0 \xrightarrow{cif} 2,\ 0 \xrightarrow{\underline{/}} 3,\ 0 \xrightarrow{\underline{␣}} 5,\ 0 \xrightarrow{\underline{+},\underline{*},\underline{(},\underline{)},\underline{=}} 4,\ 0 \xrightarrow{\underline{;}} 6,$$
$$1 \xrightarrow{lit,cif} 1,\ 1 \xrightarrow{V\setminus(lit\cup cif)} 7,\ 2 \xrightarrow{cif} 2,\ 2 \xrightarrow{V\setminus cif} 7,\ 3 \xrightarrow{\underline{*}} 8,\ 3 \xrightarrow{V\setminus\{\underline{*}\}} 7,\ 4 \xrightarrow{V} 7,$$
$$5 \xrightarrow{\underline{␣}} 5,\ 5 \xrightarrow{V\setminus\{\underline{␣}\}} 7,\ {\color{red}6 \xrightarrow{V} 7}$$

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	*lit*	*cif*	*altesb*
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	*cif*	*altesb*
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	/	*	$V \setminus \{/, *\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	␣	*altesb*
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	+,*,/,(,),=	*altesb*
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	;	*altesb*
0	1	2
1	2	2
2	2	2

F: 1

$$Q: \underline{\langle 0,0,0,0,0,0\rangle = 0}, \underline{\langle 1,2,5,2,2,2\rangle = 1}, \underline{\langle 2,1,5,2,2,2\rangle = 2},$$
$$\underline{\langle 2,2,1,2,1,2\rangle = 3}, \underline{\langle 2,2,5,2,1,2\rangle = 4}, \underline{\langle 2,2,5,1,2,2\rangle = 5},$$
$$\underline{\langle 2,2,5,2,2,1\rangle = 6}, \underline{\langle 2,2,5,2,2,2\rangle = 7}, \langle 2,2,2,2,2,2\rangle = 8$$

$$q_0 = 0;$$

$$F: 1\ [1],\ 2\ [2],\ 3\ [5],\ 4\ [5],\ 5\ [4],\ 6\ [6]$$

$$\delta: 0 \xrightarrow{lit} 1,\ 0 \xrightarrow{cif} 2,\ 0 \xrightarrow{/} 3,\ 0 \xrightarrow{␣} 5,\ 0 \xrightarrow{+,*,(,),=} 4,\ 0 \xrightarrow{;} 6,$$
$$1 \xrightarrow{lit,cif} 1,\ 1 \xrightarrow{V\setminus(lit\cup cif)} 7,\ 2 \xrightarrow{cif} 2,\ 2 \xrightarrow{V\setminus cif} 7,\ 3 \xrightarrow{*} 8,\ 3 \xrightarrow{V\setminus\{*\}} 7,\ 4 \xrightarrow{V} 7,$$
$$5 \xrightarrow{␣} 5,\ 5 \xrightarrow{V\setminus\{␣\}} 7,\ 6 \xrightarrow{V} 7,\ 7 \xrightarrow{V} 7$$

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	lit	cif	altesb
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	cif	altesb
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	/	*	$V \setminus \{/, *\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	␣	altesb
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	+,*,/,(,),=	altesb
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	;	altesb
0	1	2
1	2	2
2	2	2

F: 1

$$Q: \ \langle 0,0,0,0,0,0\rangle = 0,\ \underline{\langle 1,2,5,2,2,2\rangle = 1},\ \underline{\langle 2,1,5,2,2,2\rangle = 2},$$
$$\underline{\langle 2,2,1,2,1,2\rangle = 3},\ \underline{\langle 2,2,5,2,1,2\rangle = 4},\ \underline{\langle 2,2,5,1,2,2\rangle = 5},$$
$$\underline{\langle 2,2,5,2,2,1\rangle = 6},\ \underline{\langle 2,2,5,2,2,2\rangle = 7},\ \underline{\langle 2,2,2,2,2,2\rangle = 8},$$
$$\langle 2,2,3,2,2,2\rangle = 9,$$

$q_0 = 0;$

$F: \ 1\ [1],\ 2\ [2],\ 3\ [5],\ 4\ [5],\ 5\ [4],\ 6\ [6]$

$$\delta: \ 0 \xrightarrow{lit} 1,\ 0 \xrightarrow{cif} 2,\ 0 \xrightarrow{/} 3,\ 0 \xrightarrow{␣} 5,\ 0 \xrightarrow{+,*,(,),=} 4,\ 0 \xrightarrow{;} 6,$$
$$1 \xrightarrow{lit,cif} 1,\ 1 \xrightarrow{V\setminus(lit\cup cif)} 7,\ 2 \xrightarrow{cif} 2,\ 2 \xrightarrow{V\setminus cif} 7,\ 3 \xrightarrow{*} 8,\ 3 \xrightarrow{V\setminus\{*\}} 7,\ 4 \xrightarrow{V} 7,$$
$$5 \xrightarrow{␣} 5,\ 5 \xrightarrow{V\setminus\{␣\}} 7,\ 6 \xrightarrow{V} 7,\ 7 \xrightarrow{V} 7,\ 8 \xrightarrow{*} 9,\ 8 \xrightarrow{V\setminus\{*\}} 8$$

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	lit	cif	altesb
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	cif	altesb
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	/	*	$V \setminus \{/, *\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	␣	altesb
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	+,*,/,(,),=	altesb
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	;	altesb
0	1	2
1	2	2
2	2	2

F: 1

$Q: \langle 0,0,0,0,0,0\rangle = 0,\ \langle 1,2,5,2,2,2\rangle = 1,\ \langle 2,1,5,2,2,2\rangle = 2,$
$\langle 2,2,1,2,1,2\rangle = 3,\ \langle 2,2,5,2,1,2\rangle = 4,\ \langle 2,2,5,1,2,2\rangle = 5,$
$\langle 2,2,5,2,2,1\rangle = 6,\ \langle 2,2,5,2,2,2\rangle = 7,\ \langle 2,2,2,2,2,2\rangle = 8,$
$\langle 2,2,3,2,2,2\rangle = 9,\ \langle 2,2,4,2,2,2\rangle = 10$

$q_0 = 0;$

$F:\ 1\ [1],\ 2\ [2],\ 3\ [5],\ 4\ [5],\ 5\ [4],\ 6\ [6],\ 10\ [3]$

$\delta:\ 0 \xrightarrow{lit} 1,\ 0 \xrightarrow{cif} 2,\ 0 \xrightarrow{/} 3,\ 0 \xrightarrow{␣} 5,\ 0 \xrightarrow{+,*,(,),=} 4,\ 0 \xrightarrow{;} 6,$
$1 \xrightarrow{lit,cif} 1,\ 1 \xrightarrow{V\setminus(lit \cup cif)} 7,\ 2 \xrightarrow{cif} 2,\ 2 \xrightarrow{V\setminus cif} 7,\ 3 \xrightarrow{*} 8,\ 3 \xrightarrow{V\setminus\{*\}} 7,\ 4 \xrightarrow{V} 7,$
$5 \xrightarrow{␣} 5,\ 5 \xrightarrow{V\setminus\{␣\}} 7,\ 6 \xrightarrow{V} 7,\ 7 \xrightarrow{V} 7,\ 8 \xrightarrow{*} 9,\ 8 \xrightarrow{V\setminus\{*\}} 8,$
$9 \xrightarrow{/} 10,\ 9 \xrightarrow{V\setminus\{/\}} 8$

Algoritm reuniune disjunctă AFD

Exemplu:

lit (1)

δ	*lit*	*cif*	*altesb*
0	1	2	2
1	1	1	2
2	2	2	2

F: 1

int (2)

δ	*cif*	*altesb*
0	1	2
1	1	2
2	2	2

F: 1

com (3)

δ	$\underline{/}$	$\underline{*}$	$V \setminus \{\underline{/}, \underline{*}\}$
0	1	5	5
1	5	2	5
2	2	3	2
3	4	2	2
4	5	5	5
5	5	5	5

F: 4

spa (4)

δ	$\underline{␣}$	*altesb*
0	1	2
1	1	2
2	2	2

F: 1

op (5)

δ	$\underline{+},\underline{*},\underline{/},\underline{(},\underline{)},\underline{=}$	*altesb*
0	1	2
1	2	2
2	2	2

F: 1

delim (6)

δ	$\underline{;}$	*altesb*
0	1	2
1	2	2
2	2	2

F: 1

$$Q:\ \underline{\langle 0,0,0,0,0,0\rangle = 0},\ \underline{\langle 1,2,5,2,2,2\rangle = 1},\ \underline{\langle 2,1,5,2,2,2\rangle = 2},$$
$$\underline{\langle 2,2,1,2,1,2\rangle = 3},\ \underline{\langle 2,2,5,2,1,2\rangle = 4},\ \underline{\langle 2,2,5,1,2,2\rangle = 5},$$
$$\underline{\langle 2,2,5,2,2,1\rangle = 6},\ \underline{\langle 2,2,5,2,2,2\rangle = 7},\ \underline{\langle 2,2,2,2,2,2\rangle = 8}$$
$$\underline{\langle 2,2,3,2,2,2\rangle = 9},\ \underline{\langle 2,2,4,2,2,2\rangle = 10}$$

$q_0 = 0;$

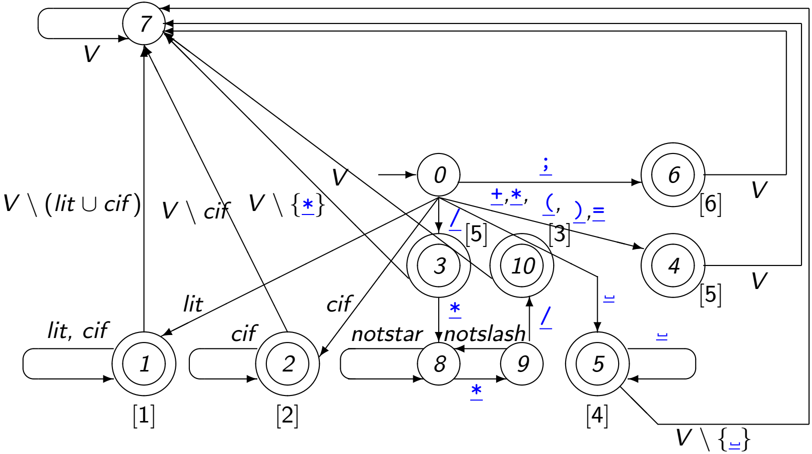
$F:\ 1\ [1],\ 2\ [2],\ 3\ [5],\ 4\ [5],\ 5\ [4],\ 6\ [6],\ 10\ [3]$

$$\delta:\ 0 \xrightarrow{lit} 1,\ 0 \xrightarrow{cif} 2,\ 0 \xrightarrow{\underline{/}} 3,\ 0 \xrightarrow{\underline{␣}} 5,\ 0 \xrightarrow{\underline{+},\underline{*},\underline{(},\underline{)},\underline{=}} 4,\ 0 \xrightarrow{\underline{;}} 6,$$
$$1 \xrightarrow{lit,cif} 1,\ 1 \xrightarrow{V\setminus(lit\cup cif)} 7,\ 2 \xrightarrow{cif} 2,\ 2 \xrightarrow{V\setminus cif} 7,\ 3 \xrightarrow{\underline{*}} 8,\ 3 \xrightarrow{V\setminus\{\underline{*}\}} 7,\ 4 \xrightarrow{V} 7,$$
$$5 \xrightarrow{\underline{␣}} 5,\ 5 \xrightarrow{V\setminus\{\underline{␣}\}} 7,\ 6 \xrightarrow{V} 7,\ 7 \xrightarrow{V} 7,\ 8 \xrightarrow{\underline{*}} 9,\ 8 \xrightarrow{V\setminus\{\underline{*}\}} 8,$$
$$9 \xrightarrow{\underline{/}} 10,\ 9 \xrightarrow{V\setminus\{\underline{/}\}} 8,\ 10 \xrightarrow{V} 7$$

Algoritm reuniune disjunctă AFD

Exemplu:

Desenat, automatul rezultat arată astfel (în dreptul fiecărei stări finale am scris între [ ] clasele de tokeni definite de ea):



Analiza lexicală

Observăm că starea 7 nu este utilizată la recunoaşterea cuvintelor - ea doar captează tranziţiile care nu ar trebui definite, pentru a avea un AFD total. Am putea elimina această stare şi tranziţiile ce intră/ies din ea, obţinând un AFD echivalent (inclusiv în ceea ce priveşte identificarea tipurilor de token), cu mai puţine stări, dar parţial. Se folosesc următoarele rezultate/algoritmi (demonstraţiile sunt simple - exerciţiu):

Definiție:

Fie $A = \langle Q,\ V,\ \delta,\ q_0,\ F\rangle$ *un AFD.*
O stare $q \in Q$ *este:*
• **accesibilă**, *dacă există* $\alpha \in V^\star$ *a.î.* $\delta(q_0, \alpha) = q$*;*
• **coaccesibilă**, *dacă există* $\alpha \in V^\star$ *a.î.* $\delta(q, \alpha) \in F$*.*

Observație:

• *doar stările ce sunt și accesibile și coaccesibile sunt utilizate efectiv la recunoașterea cuvintelor;*
• *starea inițială este accesibilă, stările finale sunt coaccesibile (aplicăm definiția pentru* $\alpha = \lambda$*);*
• *limbajul recunoscut de automat este nevid* $\Leftrightarrow$ *există stări ce sunt și accesibile și cooaccesibile* $\Leftrightarrow$ *există cel puțin o stare finală accesibilă* $\Leftrightarrow$ *starea inițială este coaccesibilă.*

Propoziție:

Fie $A = \langle Q,\ V,\ \delta,\ q_0,\ F\rangle$ *un AFD și tok* $: F \longrightarrow \{1,\ldots,n\}$, $n \in \mathbb{N}$, $n \geq 1$, *o funcție.*

Construim AFD $A' = \langle Q',\ V,\ \delta',\ q_0',\ F'\rangle$ *c si funcția* $tok' : F \longrightarrow \{1,\ldots,n\}$ *astfel:*

- $Q' = \{q_0\} \cup \{q \in Q\ :\ q$ *este și accesibilă și coaccesibilă în* $A\}$;
- $q_0' = q0$;
- *pentru orice* $q \in Q'$ *și* $a \in V$, *dacă* $\delta(q,a)$ *este definit în* A *și* $\delta(q,a)$ *este și accesibilă și coaccesibilă în* A, *atunci* $\delta'(q,a) = \delta(q,a)$, *altfel* $\delta'(q,a)$ *este nedefinit;*
- $F' = \{q \in F\ :\ q$ *este accesibilă în* $A\}$.
- $tok' = tok|_{F'}$ *(dacă* $F' = \emptyset$ *atunci* tok' *este funcția banală).*

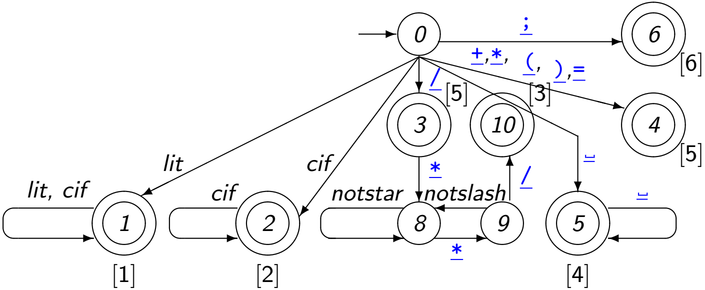
Atunci:

- *pentru orice* $\alpha \in V^\star$ *avem* $\delta(q_0,\alpha) \in F$ *d.d.* $\delta'(q_0',\alpha) \in F'$, *iar în acest caz avem* $tok(\delta(q_0,\alpha)) = tok'(\delta'(q_0',\alpha))$;
- *în particular,* $L(A) = L(A')$.

Deci, pentru recunoașterea tokenilor putem folosi automatul restrâns.

Exemplu:

Aplicând algoritmul din propoziţia precedentă automatului ce definea clasele de tokeni în exemplul precedent, obţinem:



Acesta este automatul pe care îl vom folosi la recunoaşterea tokenilor.

Analiza lexicală

Analiza lexicală (bazată pe automatul de mai înainte) se efectuează astfel:

• pornind cu automatul din starea inițială, luăm pe rând caracterele din codul sursă și facem tranziții cu ele, până se termină codul sursă sau automatul se blochează (ajunge într-o stare din care cu caracterul curent nu mai există tranziție);

• dacă starea în care s-a ajuns (prin terminarea codului sursă sau blocarea automatului) este finală, atunci s-a mai identificat un token; tipul său este dat de starea finală respectivă (funcția *tok* a acesteia), iar valoarea tokenului este șirul consumat la recunoașterea lui; după emiterea tokenului, dacă nu s-a terminat codul sursă, restăm automatul (îl punem din nou în starea inițială) și facem tranziții cu următoarele caractere, începând cu caracterul cu care s-a blocat mai devreme (din starea inițială s-ar putea să existe tranziție cu el) până când codul sursă se termină sau automatul se blocheazădin nou;

• dacă starea respectivă nu este finală, atunci am întâlnit o eroare lexicală; în acest moment putem încheia scanarea semnalând eroare, sau putem încerca o recuperare a ei;

Analiza lexicală

• pentru recuperarea erorii vom parcurge în sens invers drumul de stări urmat în automat la scanarea tokenului curent, introducând înapoi în input caracterele cu care s-au făcut tranzițiile;

• dacă vom întâlni o stare finală, ea va fi ultima stare finală prin care am trecut la scanarea tokenului curent, și atunci vom emite tokenul cu tipul dat de acea stare finală și valoarea dată de șirul consumat până la ea, după care vom reseta automatul și cu caracterul următor (ultimul reintrodus în input) vom face iar tranziții;

• dacă vom reveni în starea inițială fără să fi întâlnit nici o stare finală, atunci eroarea lexicală este atât de mare încât nu conține nici un token ca prefix; în acest caz vom opri scanarea, semnalând eroarea lexicală.

Observație:

Faptul că tranziţiile se fac până automatul se blochează (sau se termină codul sursă) ne garantează, în cazul când ne-am oprit într-o stare finală, că aceasta este ultima stare finală prin care am trecut la scanarea tokenului curent. Astfel, fiecare token identificat este cel mai lung care se poate forma de la poziţia de unde a început scanarea lui, deci se obţine o interpretare orientată spre dreapta.

Analiza lexicală

Putem formaliza algoritmul de analiză lexicală a unui șir (cod sursă) $w \in V^\star$ bazat pe AFD $A = \langle Q, V, \delta, q_0, F \rangle$ și funcția $tok : F \longrightarrow \{1, \ldots, n\}$ astfel:

- Lucrăm cu configurații de foma $(\alpha; s; \beta; \gamma)$ unde:
  - α este o stivă cu vârful la stânga, reprezentând partea rămasă din codul sursă scanat;
  - s poate fi c (continuare), b (blocaj), s (succes), e (eroare);
  - β este o stivă cu vârful la dreapta, conținând traseul (stări intercalate cu caractere) parcurs în automat la scanarea tokenului curent;
  - γ este o stivă cu vârful la dreapta, conținând partea generată din interpretare;
- Configurația inițială este $(w; c; q_0; \lambda)$;
- Configurație finală este una de forma:

$(\lambda; s; q_0; \gamma)$; este un caz de succes, în acest caz γ este interpretarea lui w;
$(\alpha; e; q_0; \gamma)$; este un caz de eșec (eroare lexicală nerecuperabilă);

Analiza lexicală

• Trecerea de la o configurație la alta se face în baza următoarelor reguli (în fiecare moment se poate aplica cel mult una dintre ele):

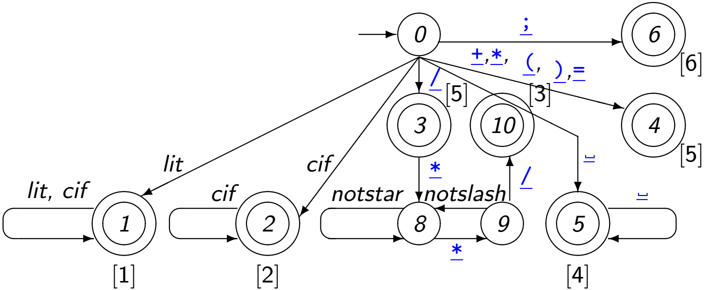
1. $(a\alpha;\ c;\ \beta q;\ \gamma) \vdash (\alpha;\ c;\ \beta qar;\ \gamma)$, dacă $\delta(q, a) = r$;
2. $(a\alpha;\ c;\ \beta q;\ \gamma) \vdash (a\alpha;\ b;\ \beta q;\ \gamma)$, dacă $\delta(q, a)$ nu e definită;
3. $(\lambda;\ c;\ \beta q;\ \gamma) \vdash (\lambda;\ b;\ \beta q;\ \gamma)$;
4. $(\alpha;\ b;\ \beta q;\ \gamma) \vdash (\alpha;\ c;\ q_0;\ \gamma\langle\varphi, tok(q)\rangle)$, dacă $q \in F$ și $\alpha \neq \lambda$; φ este cuvântul din $V^\star$ care se obține din βq prin eliminarea stărilor intercalate între simbolurile din V;
5. $(\lambda;\ b;\ \beta q;\ \gamma) \vdash (\lambda;\ s;\ q_0;\ \gamma\langle\varphi, tok(q)\rangle)$, dacă $q \in F$; φ este cuvântul din $V^\star$ care se obține din βq prin eliminarea stărilor intercalate între simbolurile din V;
6. $(\alpha;\ b;\ \beta raq;\ \gamma) \vdash (a\alpha;\ b;\ \beta r;\ \gamma)$, dacă $q \notin F$;
7. $(\alpha;\ b;\ q_0;\ \gamma) \vdash (\alpha;\ e;\ q_0;\ \gamma)$, dacă $q_0 \notin F$;

Obs: ultimele două reguli sunt considerate doar dacă vrem să încercăm recuperarea erorilor; altfel, în locul lor vom considera regula:

6'. $(\alpha;\ b;\ \beta q;\ \gamma) \vdash (\alpha;\ e;\ \beta q;\ \gamma)$, dacă $q \notin F$.

Exemplu:

Să aplicăm algoritmul de analiză lexicală în cazul când AFD, *tok* şi *w* sunt cele din exemplele anterioare:



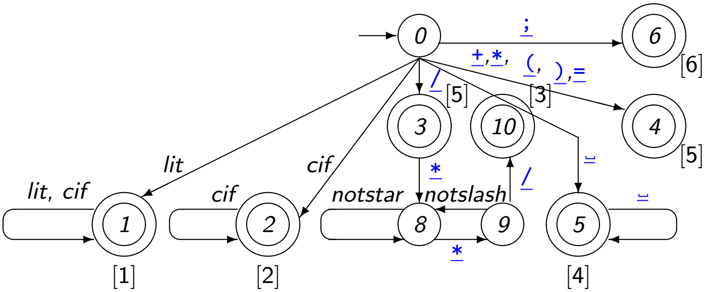
$(\underline{\texttt{abc=(12+x30)+1;␣␣y␣=abc;}};\ c;\ 0;\ \lambda)$

$\overset{1}{\vdash}\ (\underline{\texttt{bc=(12+x30)+1;␣␣y␣=abc;}};\ c;\ 0\underline{\texttt{a}}1;\ \lambda)$

$\overset{1}{\vdash}\ (\underline{\texttt{c=(12+x30)+1;␣␣y␣=abc;}};\ c;\ 0\underline{\texttt{a}}1\underline{\texttt{b}}1;\ \lambda)$

$\overset{1}{\vdash}\ (\underline{\texttt{=(12+x30)+1;␣␣y␣=abc;}};\ c;\ 0\underline{\texttt{a}}1\underline{\texttt{b}}1\underline{\texttt{c}}1;\ \lambda)$

Exemplu:



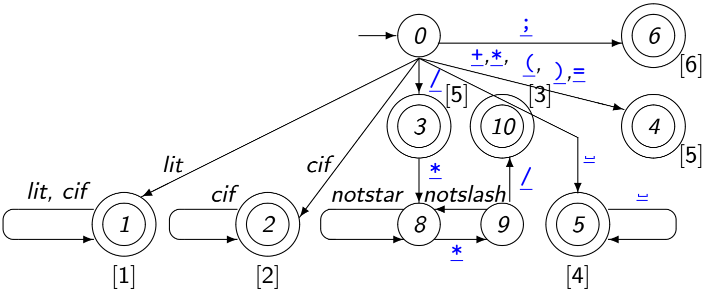
$\overset{2}{\vdash}$ (=(12+x30)+1;␣␣y␣=abc;; b; 0a1b1c1; λ)

$\overset{4}{\vdash}$ (=(12+x30)+1;␣␣y␣=abc;; c; 0; $\langle$abc, $1\rangle$)

$\overset{1}{\vdash}$ ((12+x30)+1;␣␣y␣=abc;; c; 0=4; $\langle$abc, $1\rangle$)

$\overset{2}{\vdash}$ ((12+x30)+1;␣␣y␣=abc;; b; 0=4; $\langle$abc, $1\rangle$)

Exemplu:



$\overset{4}{\vdash}$ ((12+x30)+1;␣␣y␣=abc;; *c*; 0; ⟨abc, *1*⟩⟨=, *5*⟩)

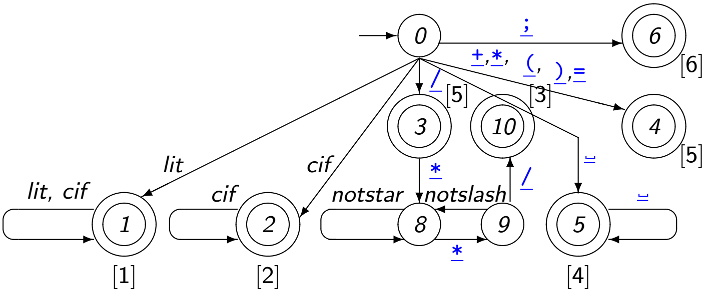
$\overset{1}{\vdash}$ (12+x30)+1;␣␣y␣=abc;; *c*; 0(4; ⟨abc, *1*⟩⟨=, *5*⟩)

$\overset{2}{\vdash}$ (12+x30)+1;␣␣y␣=abc;; *b*; 0(4; ⟨abc, *1*⟩⟨=, *5*⟩)

$\overset{4}{\vdash}$ (12+x30)+1;␣␣y␣=abc;; *c*; 0; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩)

Analiza lexicală

Exemplu:



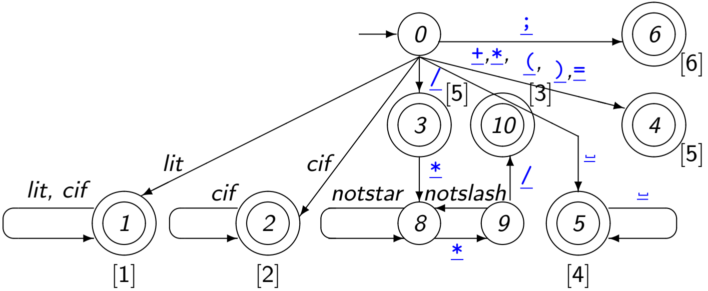
$\overset{1}{\vdash}$ (2+x30)+1;␣␣y␣=abc;; c; 012; $\langle$abc, $1\rangle\langle$=, $5\rangle\langle$(, $5\rangle$)

$\overset{1}{\vdash}$ (+x30)+1;␣␣y␣=abc;; c; 01222; $\langle$abc, $1\rangle\langle$=, $5\rangle\langle$(, $5\rangle$)

$\overset{2}{\vdash}$ (+x30)+1;␣␣y␣=abc;; b; 01222; $\langle$abc, $1\rangle\langle$=, $5\rangle\langle$(, $5\rangle$)

$\overset{4}{\vdash}$ (+x30)+1;␣␣y␣=abc;; c; 0; $\langle$abc, $1\rangle\langle$=, $5\rangle\langle$(, $5\rangle\langle$12, $2\rangle$)

Exemplu:



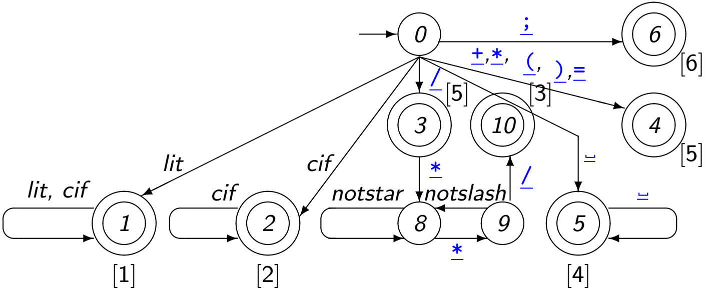
$\overset{1}{\vdash}$ (x30)+1;␣␣y␣=abc;; *c*; 0+4; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩)

$\overset{2}{\vdash}$ (x30)+1;␣␣y␣=abc;; *b*; 0+4; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩)

$\overset{4}{\vdash}$ (x30)+1;␣␣y␣=abc;; *c*; 0; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩)

$\overset{1}{\vdash}$ (30)+1;␣␣y␣=abc;; *c*; 0x1; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩)

Exemplu:



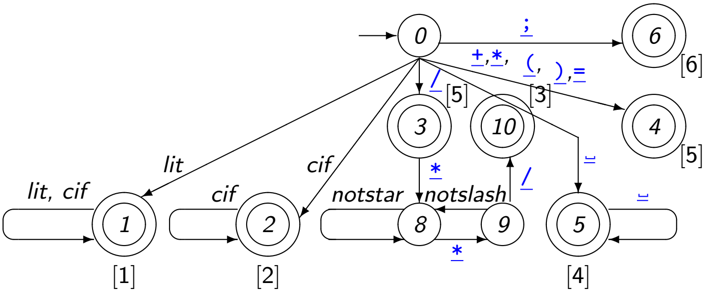
$\vdash^{1}$ (0)+1;␣␣y␣=abc;; c; 0x131; $\langle$abc, $1\rangle\langle$=, $5\rangle\langle$(, $5\rangle\langle$12, $2\rangle\langle$+, $5\rangle$)

$\vdash^{1}$ ()+1;␣␣y␣=abc;; c; 0x13101; $\langle$abc, $1\rangle\langle$=, $5\rangle\langle$(, $5\rangle\langle$12, $2\rangle\langle$+, $5\rangle$)

$\vdash^{2}$ ()+1;␣␣y␣=abc;; b; 0x13101; $\langle$abc, $1\rangle\langle$=, $5\rangle\langle$(, $5\rangle\langle$12, $2\rangle\langle$+, $5\rangle$)

$\vdash^{4}$ ()+1;␣␣y␣=abc;; c; 0; $\langle$abc, $1\rangle\langle$=, $5\rangle\langle$(, $5\rangle\langle$12, $2\rangle\langle$+, $5\rangle\langle$x30, $1\rangle$)

Exemplu:



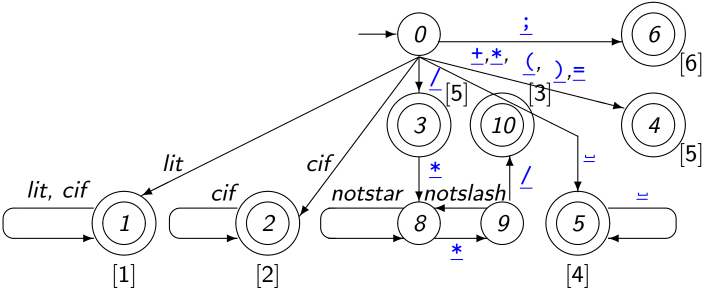
$\overset{1}{\vdash} (\underline{\texttt{+1;␣␣y␣=abc;}};\ c;\ 0\underline{)}4;\ \langle\underline{\texttt{abc}}, 1\rangle\langle\underline{\texttt{=}}, 5\rangle\langle\underline{\texttt{(}}, 5\rangle\langle\underline{\texttt{12}}, 2\rangle\langle\underline{\texttt{+}}, 5\rangle\langle\underline{\texttt{x30}}, 1\rangle)$

$\overset{2}{\vdash} (\underline{\texttt{+1;␣␣y␣=abc;}};\ b;\ 0\underline{)}4;\ \langle\underline{\texttt{abc}}, 1\rangle\langle\underline{\texttt{=}}, 5\rangle\langle\underline{\texttt{(}}, 5\rangle\langle\underline{\texttt{12}}, 2\rangle\langle\underline{\texttt{+}}, 5\rangle\langle\underline{\texttt{x30}}, 1\rangle)$

$\overset{4}{\vdash} (\underline{\texttt{+1;␣␣y␣=abc;}};\ c;\ 0;\ \langle\underline{\texttt{abc}}, 1\rangle\langle\underline{\texttt{=}}, 5\rangle\langle\underline{\texttt{(}}, 5\rangle\langle\underline{\texttt{12}}, 2\rangle\langle\underline{\texttt{+}}, 5\rangle\langle\underline{\texttt{x30}}, 1\rangle\langle\underline{\texttt{)}}, 5\rangle)$

$\overset{1}{\vdash} (\underline{\texttt{1;␣␣y␣=abc;}};\ c;\ 0\underline{+}4;\ \langle\underline{\texttt{abc}}, 1\rangle\langle\underline{\texttt{=}}, 5\rangle\langle\underline{\texttt{(}}, 5\rangle\langle\underline{\texttt{12}}, 2\rangle\langle\underline{\texttt{+}}, 5\rangle\langle\underline{\texttt{x30}}, 1\rangle\langle\underline{\texttt{)}}, 5\rangle)$

Exemplu:



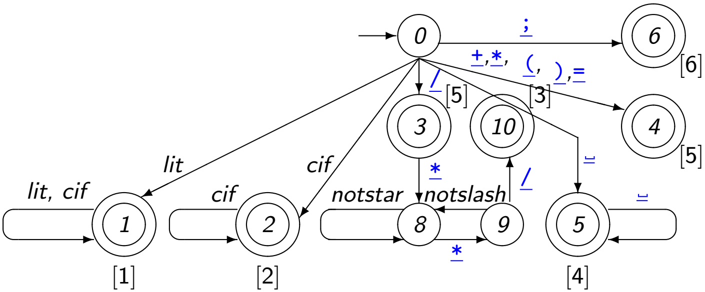
$\overset{2}{\vdash}$ (1;␣␣y␣=abc;; *b*; 0+4; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩)

$\overset{4}{\vdash}$ (1;␣␣y␣=abc;; *c*; 0; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩)

$\overset{1}{\vdash}$ (;␣␣y␣=abc;; *c*; 012; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩)

$\overset{2}{\vdash}$ (;␣␣y␣=abc;; *b*; 012; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩)

Exemplu:



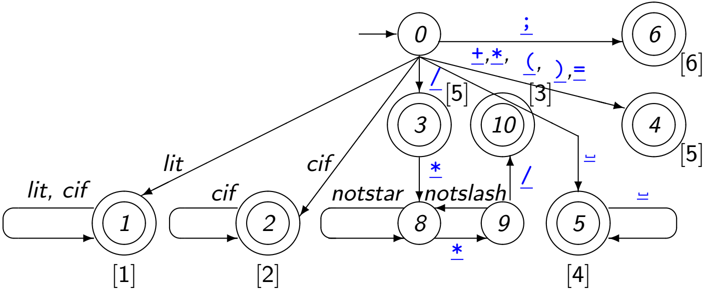
$\overset{4}{\vdash} (\underline{\texttt{;\textvisiblespace\textvisiblespace y\textvisiblespace=abc;}};\ c;\ 0;\ \langle\underline{\texttt{abc}}, 1\rangle\langle\underline{\texttt{=}}, 5\rangle\langle\underline{\texttt{(}}, 5\rangle\langle\underline{\texttt{12}}, 2\rangle\langle\underline{\texttt{+}}, 5\rangle\langle\underline{\texttt{x30}}, 1\rangle\langle\underline{\texttt{)}}, 5\rangle\langle\underline{\texttt{+}}, 5\rangle$
$\langle\underline{\texttt{1}}, 2\rangle)$

$\overset{1}{\vdash} (\underline{\texttt{\textvisiblespace\textvisiblespace y\textvisiblespace=abc;}};\ c;\ 0\underline{\texttt{;}}6;\ \langle\underline{\texttt{abc}}, 1\rangle\langle\underline{\texttt{=}}, 5\rangle\langle\underline{\texttt{(}}, 5\rangle\langle\underline{\texttt{12}}, 2\rangle\langle\underline{\texttt{+}}, 5\rangle\langle\underline{\texttt{x30}}, 1\rangle\langle\underline{\texttt{)}}, 5\rangle\langle\underline{\texttt{+}}, 5\rangle$
$\langle\underline{\texttt{1}}, 2\rangle)$

$\overset{2}{\vdash} (\underline{\texttt{\textvisiblespace\textvisiblespace y\textvisiblespace=abc;}};\ b;\ 0\underline{\texttt{;}}6;\ \langle\underline{\texttt{abc}}, 1\rangle\langle\underline{\texttt{=}}, 5\rangle\langle\underline{\texttt{(}}, 5\rangle\langle\underline{\texttt{12}}, 2\rangle\langle\underline{\texttt{+}}, 5\rangle\langle\underline{\texttt{x30}}, 1\rangle\langle\underline{\texttt{)}}, 5\rangle\langle\underline{\texttt{+}}, 5\rangle$
$\langle\underline{\texttt{1}}, 2\rangle)$

$\overset{4}{\vdash} (\underline{\texttt{\textvisiblespace\textvisiblespace y\textvisiblespace=abc;}};\ c;\ 0;\ \langle\underline{\texttt{abc}}, 1\rangle\langle\underline{\texttt{=}}, 5\rangle\langle\underline{\texttt{(}}, 5\rangle\langle\underline{\texttt{12}}, 2\rangle\langle\underline{\texttt{+}}, 5\rangle\langle\underline{\texttt{x30}}, 1\rangle\langle\underline{\texttt{)}}, 5\rangle\langle\underline{\texttt{+}}, 5\rangle$
$\langle\underline{\texttt{1}}, 2\rangle\langle\underline{\texttt{;}}, 6\rangle)$

Exemplu:



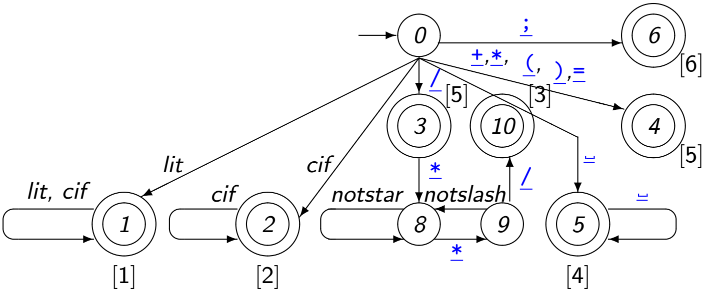
$\stackrel{1}{\vdash}$ (␣y␣=abc;; *c*; 0␣5; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩⟨1, *2*⟩⟨;, *6*⟩)

$\stackrel{1}{\vdash}$ (y␣=abc;; *c*; 0␣5␣5; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩⟨1, *2*⟩⟨;, *6*⟩)

$\stackrel{2}{\vdash}$ (y␣=abc;; *b*; 0␣5␣5; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩⟨1, *2*⟩⟨;, *6*⟩)

$\stackrel{4}{\vdash}$ (y␣=abc;; *c*; 0; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩)

Exemplu:



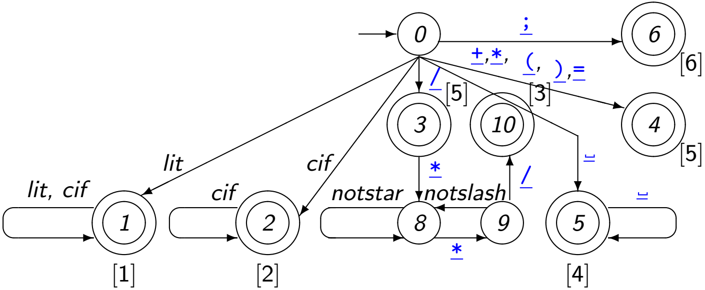
$\overset{1}{\vdash}$ ($\underline{\texttt{␣=abc;}}$; c; $0\underline{\texttt{y}}1$; $\langle\underline{\texttt{abc}}, 1\rangle\langle\underline{\texttt{=}}, 5\rangle\langle\underline{\texttt{(}}, 5\rangle\langle\underline{\texttt{12}}, 2\rangle\langle\underline{\texttt{+}}, 5\rangle\langle\underline{\texttt{x30}}, 1\rangle\langle\underline{\texttt{)}}, 5\rangle\langle\underline{\texttt{+}}, 5\rangle$
$\langle\underline{\texttt{1}}, 2\rangle\langle\underline{\texttt{;}}, 6\rangle\langle\underline{\texttt{␣␣}}, 4\rangle$)

$\overset{2}{\vdash}$ ($\underline{\texttt{␣=abc;}}$; b; $0\underline{\texttt{y}}1$; $\langle\underline{\texttt{abc}}, 1\rangle\langle\underline{\texttt{=}}, 5\rangle\langle\underline{\texttt{(}}, 5\rangle\langle\underline{\texttt{12}}, 2\rangle\langle\underline{\texttt{+}}, 5\rangle\langle\underline{\texttt{x30}}, 1\rangle\langle\underline{\texttt{)}}, 5\rangle\langle\underline{\texttt{+}}, 5\rangle$
$\langle\underline{\texttt{1}}, 2\rangle\langle\underline{\texttt{;}}, 6\rangle\langle\underline{\texttt{␣␣}}, 4\rangle$)

$\overset{4}{\vdash}$ ($\underline{\texttt{␣=abc;}}$; c; 0; $\langle\underline{\texttt{abc}}, 1\rangle\langle\underline{\texttt{=}}, 5\rangle\langle\underline{\texttt{(}}, 5\rangle\langle\underline{\texttt{12}}, 2\rangle\langle\underline{\texttt{+}}, 5\rangle\langle\underline{\texttt{x30}}, 1\rangle\langle\underline{\texttt{)}}, 5\rangle\langle\underline{\texttt{+}}, 5\rangle$
$\langle\underline{\texttt{1}}, 2\rangle\langle\underline{\texttt{;}}, 6\rangle\langle\underline{\texttt{␣␣}}, 4\rangle\langle\underline{\texttt{y}}, 1\rangle$)

$\overset{1}{\vdash}$ ($\underline{\texttt{=abc;}}$; c; $0\underline{\texttt{␣}}5$; $\langle\underline{\texttt{abc}}, 1\rangle\langle\underline{\texttt{=}}, 5\rangle\langle\underline{\texttt{(}}, 5\rangle\langle\underline{\texttt{12}}, 2\rangle\langle\underline{\texttt{+}}, 5\rangle\langle\underline{\texttt{x30}}, 1\rangle\langle\underline{\texttt{)}}, 5\rangle\langle\underline{\texttt{+}}, 5\rangle$
$\langle\underline{\texttt{1}}, 2\rangle\langle\underline{\texttt{;}}, 6\rangle\langle\underline{\texttt{␣␣}}, 4\rangle\langle\underline{\texttt{y}}, 1\rangle$)

Exemplu:



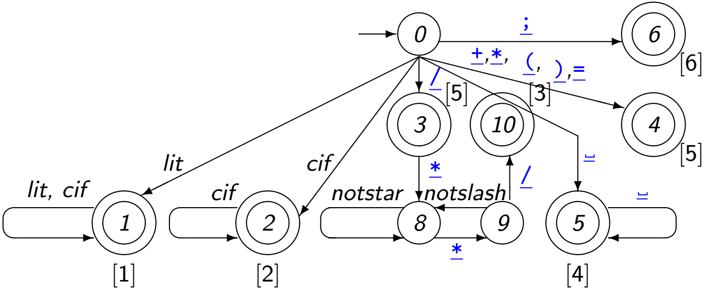
$\overset{2}{\vdash}$ (=abc;; *b*; 0␣5; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩
⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩⟨y, *1*⟩)

$\overset{4}{\vdash}$ (=abc;; *c*; 0; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩
⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩⟨y, *1*⟩⟨␣, *4*⟩)

$\overset{1}{\vdash}$ (abc;; *c*; 0=4; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩
⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩⟨y, *1*⟩⟨␣, *4*⟩)

$\overset{2}{\vdash}$ (abc;; *b*; 0=4; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩
⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩⟨y, *1*⟩⟨␣, *4*⟩)

Exemplu:



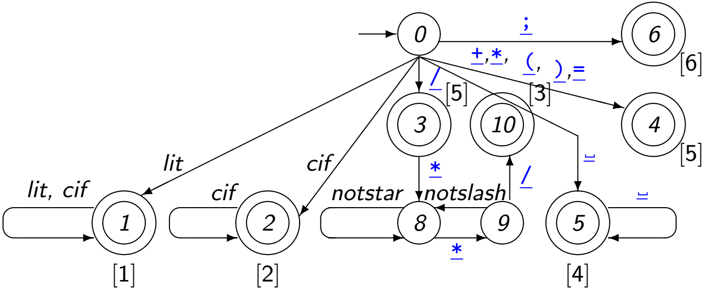
$\overset{4}{\vdash}$ (abc;; *c*; 0; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩
⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩⟨y, *1*⟩⟨␣, *4*⟩⟨=, *5*⟩)

$\overset{1}{\vdash}$ (bc;; *c*; 0a1; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩
⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩⟨y, *1*⟩⟨␣, *4*⟩⟨=, *5*⟩)

$\overset{1}{\vdash}$ (c;; *c*; 0a1b1; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩
⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩⟨y, *1*⟩⟨␣, *4*⟩⟨=, *5*⟩)

$\overset{1}{\vdash}$ (;; *c*; 0a1b1c1; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩
⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩⟨y, *1*⟩⟨␣, *4*⟩⟨=, *5*⟩)

Exemplu:



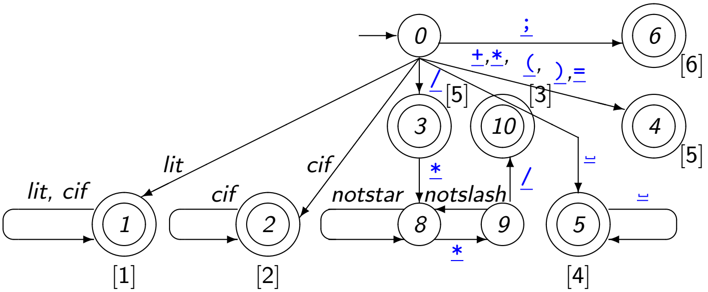
$\overset{2}{\vdash}$ (;; *b*; 0a1b1c1; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩
⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩⟨y, *1*⟩⟨␣, *4*⟩⟨=, *5*⟩)

$\overset{4}{\vdash}$ (;; *c*; 0; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩
⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩⟨y, *1*⟩⟨␣, *4*⟩⟨=, *5*⟩⟨abc, *1*⟩)

$\overset{1}{\vdash}$ (λ; *c*; 0;6; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩
⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩⟨y, *1*⟩⟨␣, *4*⟩⟨=, *5*⟩⟨abc, *1*⟩)

$\overset{3}{\vdash}$ (λ; *b*; 0;6; ⟨abc, *1*⟩⟨=, *5*⟩⟨(, *5*⟩⟨12, *2*⟩⟨+, *5*⟩⟨x30, *1*⟩⟨), *5*⟩⟨+, *5*⟩
⟨1, *2*⟩⟨;, *6*⟩⟨␣␣, *4*⟩⟨y, *1*⟩⟨␣, *4*⟩⟨=, *5*⟩⟨abc, *1*⟩)

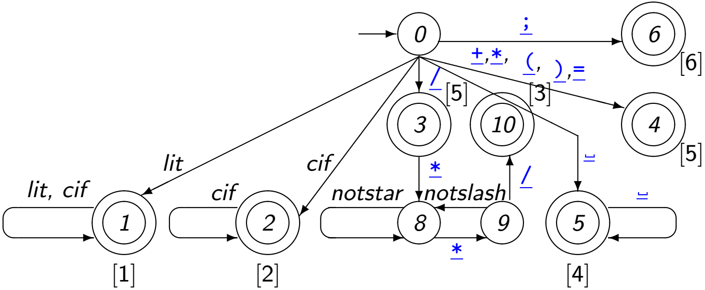
Exemplu:



$\overset{5}{\vdash} (\lambda;\ s;\ 0;\ \langle \underline{\texttt{abc}}, 1\rangle\langle \underline{\texttt{=}}, 5\rangle\langle \underline{\texttt{(}}, 5\rangle\langle \underline{\texttt{12}}, 2\rangle\langle \underline{\texttt{+}}, 5\rangle\langle \underline{\texttt{x30}}, 1\rangle\langle \underline{\texttt{)}}, 5\rangle\langle \underline{\texttt{+}}, 5\rangle$
$\langle \underline{\texttt{1}}, 2\rangle\langle \underline{\texttt{;}}, 6\rangle\langle \underline{\sqcup\sqcup}, 4\rangle\langle \underline{\texttt{y}}, 1\rangle\langle \underline{\sqcup}, 4\rangle\langle \underline{\texttt{=}}, 5\rangle\langle \underline{\texttt{abc}}, 1\rangle\langle \underline{\texttt{;}}, 6\rangle)$

Exemplu:

Să aplicăm algoritmul şi pentru $w =$ `ab12/*x12*` (codul conţine o eroare lexicală: comentariu netrminat):



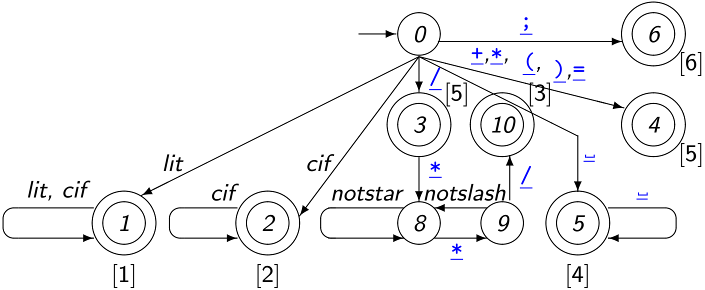
($\texttt{ab12/*x12*}$; c; 0; λ)

$\overset{1}{\vdash}$ ($\texttt{b12/*x12*}$; c; 0$\texttt{a}$1; λ)

$\overset{1}{\vdash}$ ($\texttt{12/*x12*}$; c; 0$\texttt{a}$1$\texttt{b}$1; λ)

$\overset{1}{\vdash}$ ($\texttt{2/*x12*}$; c; 0$\texttt{a}$1$\texttt{b}$1$\texttt{1}$1; λ)

Exemplu:



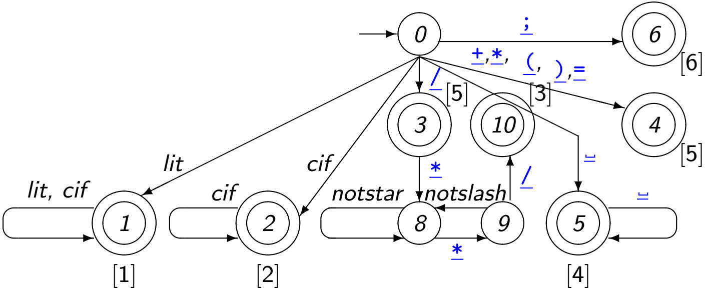
$\overset{1}{\vdash}$ (/*x12*; *c*; 0a1b11121; λ)

$\overset{2}{\vdash}$ (/*x12*; *b*; 0a1b11121; λ)

$\overset{4}{\vdash}$ (/*x12*; *c*; 0; $\langle$ab12, *1*$\rangle$)

$\overset{1}{\vdash}$ (*x12*; *c*; 0/3; $\langle$ab12, *1*$\rangle$)

Exemplu:



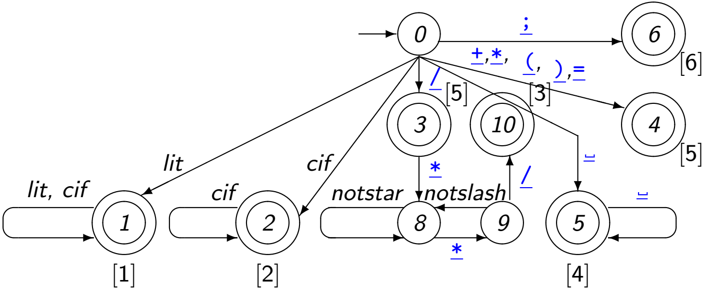
$\overset{1}{\vdash}$ (x12∗; c; 0/3∗8; ⟨ab12, *1*⟩)

$\overset{1}{\vdash}$ (12∗; c; 0/3∗8x8; ⟨ab12, *1*⟩)

$\overset{1}{\vdash}$ (2∗; c; 0/3∗8x818; ⟨ab12, *1*⟩)

$\overset{1}{\vdash}$ (∗; c; 0/3∗8x81828; ⟨ab12, *1*⟩)

Exemplu:



$\overset{1}{\vdash} (\lambda;\ c;\ 0\underline{/}3\underline{*}8\underline{x}8\underline{1}8\underline{2}8\underline{*}9;\ \langle \underline{ab12}, 1\rangle)$

$\overset{3}{\vdash} (\lambda;\ b;\ 0\underline{/}3\underline{*}8\underline{x}8\underline{1}8\underline{2}8\underline{*}9;\ \langle \underline{ab12}, 1\rangle)$

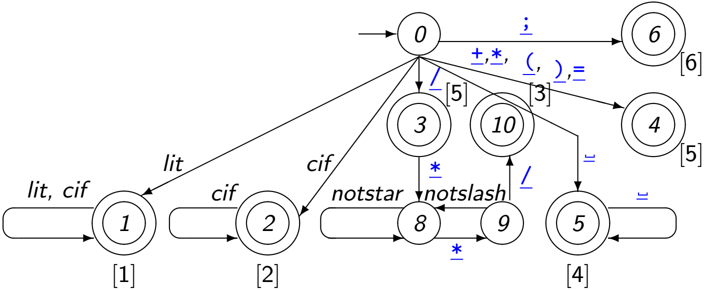
(în acest moment, dacă nu am dori recuperarea erorilor, am efectua

$\overset{6'}{\vdash} (\lambda;\ e;\ 0\underline{/}3\underline{*}8\underline{x}8\underline{1}8\underline{2}8\underline{*}9;\ \langle \underline{ab12}, 1\rangle)$ *și am încheia analiza)*

$\overset{6}{\vdash} (\underline{*};\ b;\ 0\underline{/}3\underline{*}8\underline{x}8\underline{1}8\underline{2}8;\ \langle \underline{ab12}, 1\rangle)$

$\overset{6}{\vdash} (\underline{2*};\ b;\ 0\underline{/}3\underline{*}8\underline{x}8\underline{1}8;\ \langle \underline{ab12}, 1\rangle)$

Exemplu:



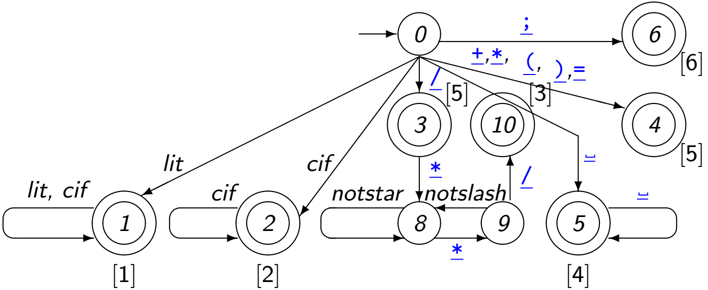
$\overset{6}{\vdash} (\underline{\texttt{12*}};\ b;\ 0\underline{\texttt{/}}3\underline{\texttt{*}}8\underline{\texttt{x}}8;\ \langle \underline{\texttt{ab12}}, 1\rangle)$

$\overset{6}{\vdash} (\underline{\texttt{x12*}};\ b;\ 0\underline{\texttt{/}}3\underline{\texttt{*}}8;\ \langle \underline{\texttt{ab12}}, 1\rangle)$

$\overset{6}{\vdash} (\underline{\texttt{*x12*}};\ b;\ 0\underline{\texttt{/}}3;\ \langle \underline{\texttt{ab12}}, 1\rangle)$

$\overset{4}{\vdash} (\underline{\texttt{*x12*}};\ c;\ 0;\ \langle \underline{\texttt{ab12}}, 1\rangle\langle \underline{\texttt{/}}, 5\rangle)$

Exemplu:



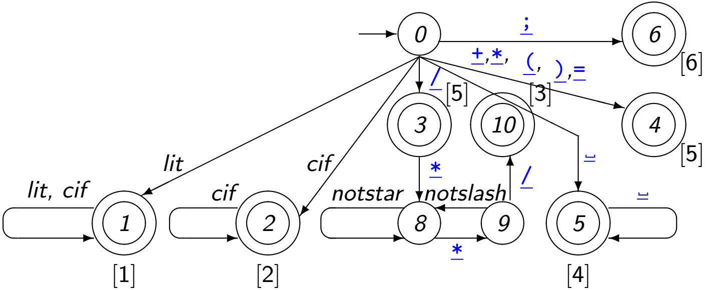
$\overset{1}{\vdash} (\underline{\texttt{x12*}};\ c;\ 0\underline{\texttt{*}}4;\ \langle \underline{\texttt{ab12}}, \mathit{1}\rangle\langle \underline{\texttt{/}}, \mathit{5}\rangle)$

$\overset{2}{\vdash} (\underline{\texttt{x12*}};\ b;\ 0\underline{\texttt{*}}4;\ \langle \underline{\texttt{ab12}}, \mathit{1}\rangle\langle \underline{\texttt{/}}, \mathit{5}\rangle)$

$\overset{4}{\vdash} (\underline{\texttt{x12*}};\ c;\ 0;\ \langle \underline{\texttt{ab12}}, \mathit{1}\rangle\langle \underline{\texttt{/}}, \mathit{5}\rangle\langle \underline{\texttt{*}}, \mathit{5}\rangle)$

$\overset{1}{\vdash} (\underline{\texttt{12*}};\ c;\ 0\underline{\texttt{x}}1;\ \langle \underline{\texttt{ab12}}, \mathit{1}\rangle\langle \underline{\texttt{/}}, \mathit{5}\rangle\langle \underline{\texttt{*}}, \mathit{5}\rangle)$

Exemplu:



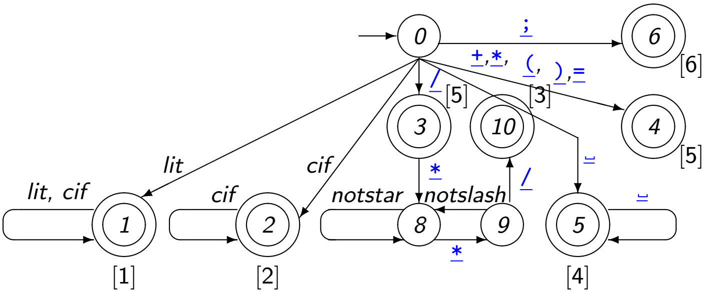
$\overset{1}{\vdash} (\underline{2*};\ c;\ 0\underline{x}1\underline{1}1;\ \langle \underline{ab12}, 1\rangle\langle \underline{/}, 5\rangle\langle \underline{*}, 5\rangle)$

$\overset{1}{\vdash} (\underline{*};\ c;\ 0\underline{x}1\underline{1}\underline{2}1;\ \langle \underline{ab12}, 1\rangle\langle \underline{/}, 5\rangle\langle \underline{*}, 5\rangle)$

$\overset{2}{\vdash} (\underline{*};\ b;\ 0\underline{x}1\underline{1}\underline{2}1;\ \langle \underline{ab12}, 1\rangle\langle \underline{/}, 5\rangle\langle \underline{*}, 5\rangle)$

$\overset{4}{\vdash} (\underline{*};\ c;\ 0;\ \langle \underline{ab12}, 1\rangle\langle \underline{/}, 5\rangle\langle \underline{*}, 5\rangle\langle \underline{x12}, 1\rangle)$

Exemplu:



$\overset{1}{\vdash} (\lambda;\ c;\ 0\underline{*}4;\ \langle \underline{\texttt{ab12}}, \mathit{1}\rangle\langle \underline{/}, \mathit{5}\rangle\langle \underline{*}, \mathit{5}\rangle\langle \underline{\texttt{x12}}, \mathit{1}\rangle)$

$\overset{3}{\vdash} (\lambda;\ b;\ 0\underline{*}4;\ \langle \underline{\texttt{ab12}}, \mathit{1}\rangle\langle \underline{/}, \mathit{5}\rangle\langle \underline{*}, \mathit{5}\rangle\langle \underline{\texttt{x12}}, \mathit{1}\rangle)$

$\overset{5}{\vdash} (\lambda;\ s;\ 0;\ \langle \underline{\texttt{ab12}}, \mathit{1}\rangle\langle \underline{/}, \mathit{5}\rangle\langle \underline{*}, \mathit{5}\rangle\langle \underline{\texttt{x12}}, \mathit{1}\rangle\langle \underline{*}, \mathit{5}\rangle)$

Observație:

În ambele exemple de mai sus, dacă am fi lucrat cu automatul total, la fiecare token am fi parcurs codul sursă până la capăt (deoarece automatul nu se blochează), apoi ne-am fi întors până la starea finală corespunzătoare. Astfel, scanarea ar fi fost mai puțin eficientă.

Aspecte de implementare

Aspecte de implementare a unui analizor lexical (scanner):

- Structura scannerului:

Analizorul lexical poate fi implementat în mai multe feluri.

O idee este să-l construim a.î. să se poată integra în schema de compilator prezentată la început.

Astfel, el va fi implementat ca o componentă care poate fi invocată din exterior și la fiecare invocare să furnizeze un nou token.

De exemplu, el poate fi un obiect (în sensul programării orientate pe obiecte) având o metodă publică "gettoken()", care la fiecare apel va furniza un nou token (tokenul curent); în acest scop, obiectul scanner va reține prin membrii săi dată poziția în codul sursă până la care a ajuns cu scanarea.

Aspecte de implementare

Conceptul de token poate fi şi el implementat ca un obiect (sau structură) cu membri (cel puţin) pentru tipul şi valoarea tokenului. La fiecare apel, metoda "gettoken()" va returna un obiect token.

Tipul poate fi un întreg (se fac nişte convenţii, ex: $0 =$ identificator, $1 =$ întreg. etc.) sau un element al unui tip enumerare.

Valoarea este un string.

Pentru a lucra unitar, "gettoken()" va semnala şi apariţia unei erori tot prin returnarea unui token, special - de ex. tipul să fie -1 iar valoarea să fie un întreg, desemnând poziţia în cod unde a fost întâlnită eroarea (a se vedea şi comentariile următoare referitoare la tabela de stringuri, unde vom arăta că valoarea tokenului poate fi returnată întotdeauna ca un întreg).

În cele ce urmează, dacă nu vom preciza altfel, aceasta este structura de scanner pe care o vom avea în vedere.

Aspecte de implementare

- Tabela de stringuri:

Celelalte componente ale compilatorului nu au nevoie de valorile tokenilor (ca stringuri) ci doar de o informație mai restrânsă, din care să se poată determina dacă doi tokeni diferiți au sau nu aceeași valoare.

Astfel, putem face următoarea optimizare: obiectul scanner va gestiona o tabelă de stringuri, care va fi un membru privat al său (deoarece doar el are nevoie de ea) și în care va reține valorile unice ale tokenilor întâlniți la scanarea codului curent.

Fiecare apel al lui "gettoken()", după ce va identifica noul token, va căuta valoarea lui în tabela de stringuri; dacă nu o va găsi, o va insera; în ambele cazuri, poziția respectivă din tabel (unde a fost găsită/inserată) va fi furnizată ca valoare a tokenului (sub forma unui întreg, pointer, referință, etc.). Astfel, obiectul token returnat poate fi format întotdeauna din doi membri întregi - unul pentru tip, altul pentru valoare (dacă nu vom menționa altfel, așa vom considera în cele ce urmează).

Astfel, rezultă o economie de memorie în cazul manevrării simultane a mai multor tokeni cu aceeași valoare.

Aspecte de implementare

În continuare, putem face diverse convenții/optimizări, de exemplu: fiecare operator să aparțină unui tip distinct de token - în acest caz în tabela de stringuri nu mai trebuie să reținem și operatorii (tipul tokenului va indica în mod unic valoarea), sau toți operatorii să aparțină unui același tip token - în acest caz vom reține în tabela de stringuri și operatorii.

La fel în cazul cuvintelor cheie.

Aspecte de implementare

- Cuvinte cheie:

În diversele limbaje de programare întîlnim cuvinte cheie. Acestea au structura unui identificator, dar sunt tokeni de alt tip, așa că pentru recunoașterea lor nu putem folosi, teoretic, starea finală a identificatorilor.
De exemplu, putem conveni ca fiecare cuvânt cheie să definească un tip distinct de token (cu o unică valoare posibilă) și atunci să necesite o stare finală distinctă.

Existența cuvintelor cheie poate fi modelată în cadrul teoriei generale folosite până acum, în baza următoarelor rezultate:

- *Orice limbaj format dintr-un singur cuvânt este regulat.*
  De exemplu limbajul $\{\alpha\}$, unde $\alpha = a_1 \ldots a_n$, $a_i \in V^\star$, poate fi descris de ER $a_1 \cdot \ldots \cdot a_n$ sau recunoscut de un AFD lanț.

- *Orice limbaj finit este regulat.*
  Într-adevăr, el este o reuniune finită de limbaje regulate; putem construi un AFD pentru el și care în plus să asocieze un tip distinct fiecărui cuvânt component (printr-o funcție "*tok*") cu algoritmul de reuninune disjunctă AFD prezentat mai înainte.

Aspecte de implementare

◦ *Clasa limbajelor regulate este închisă la diferență (adică dacă L_1 și L_2 sunt limbaje regulate, atunci $L_1 \setminus L_2$ este un limbaj regulat).*

Pentru demonstrație se pornește cu două AFD totale pentru cele două limbaje și se construiește un AFD total pentru diferența lor, ca în algoritmul de reuniune disjunctă AFD, cu următoarele diferențe:

- se lucrează doar cu două automate, nu cu n; deci stările noului AFD vor fi de forma $\langle q^1, q^2 \rangle$, cu $q^1 \in Q_1$, $q^2 \in Q_2$;
- nu mai avem de a face cu funcția "*tok*" (deoarece cuvintele recunoscute de noul AFD sunt printre cele recunoscute de primul AFD sursă - deci știm care este);
- În noul AFD vom avea $\langle q^1, q^2 \rangle \in F$ d.d $q^1 \in F_1$ și $q^2 \notin F_2$;

(demonstrațiile sunt simple - exercițiu).

Aspecte de implementare

◦ Dacă notăm cu M limbajul descris de ER "*idf*" din exemplele anterioare și fixăm o submulțime finită $C \subseteq M$, atunci putem considera $M \setminus C$ mulțimea identificatorilor și C mulțimea cuvintelor cheie; ambele sunt limbaje regulate, sunt disjuncte, iar prin tehnicile menționate mai sus putem construi câte un AFD pentru fiecare (care este sau se poate completa la unul total); cu aceste AFD și cu cele care definesc celelalte clase de token considerate (aceste clase sunt disjuncte de M, deci și de $M \setminus C$ și C) putem continua în maniera cunoscută, aplicând algortimul de reuniune disjunctă AFD, pentru a obține un AFD capabil să recunoască tokenii și care va asocia un tip identificatorilor și cate un nou tip fiecărui cuvânt cheie.

Aspecte de implementare

Din punctul de vedere al implementării putem evita însă complicarea automatului prin adăugarea de noi stări finale și putem folosi la recunoașterea cuvintelor cheie starea finală a identificatorilor, în felul următor:

dacă la apelul curent al lui "gettoken()" scanarea s-a oprit (prin blocarea automatului sau terminarea codului sursă) în starea finală a identificatorilor, șirul consumat (valoarea noului token) se caută mai întâi într-o tabelă de cuvinte cheie, care este fixată; dacă nu este găsit, se va returna un token identificator; dacă este găsit, se va returna un token de alt tip, corespunzător cuvântului cheie.

Aspecte de implementare

- Spații:

În diversele limbaje de programare se folosesc și caractere albe (blank, tab, cap de linie, etc.) sau comentarii.
Prezenă lor într-n cod sursă ajută la o mai bună înțelegere a codului de către om, dar nu au efect la execuție.

Pentru a clarifica ideile, vom numi **spațiu** o succesiune maximală de caractere albe și comentarii. În general, într-un program putem insera un spațiu între orice doi tokeni de alt tip, fără a afecta semantica programului (efectul la execuție). Spațiile se pot defini și identifica la fel ca și ceilalți tokeni - ele pot fi definite cu o ER iar pentru recunoașterea lor vom mai adăuga o stare finală în AFD.

Pentru compilator spațiile au ca singur scop să ajute scannerul să identifice ceilalți tokeni (care au efect la execuție). Odată identificați și livrați mai departe, aceștia sunt delimitați logic prin structurile de program ale compilatorului, deci spațiile nu mai sunt necesare - restul compilatorului nu mai are nevoie de spații.

Aspecte de implementare

De aceea, scannerul va fi construit a.î. să recunoască spațiile la fel ca pe ceilalți tokeni, dar să nu le livreze mai departe. Mai exact, dacă la un apel al funcției "gettoken()" automatul s-a blocat în starea finală a spațiilor, să nu se returneze tokenul spațiu, să se rămână în apelul curent, să se scaneze și să se returneze următorul token (având în vedere maximalitatea spațiilor, următorul token nu va fi spațiu).
Pentru a trata unitar și cazul când spațiul a fost ultimul token (deci nu mai există alt token după el care să se returneze) se poate inventa un token - sfârșit de fișier (*EOF*), care să se returneze dacăpartea răsă din cod este λ, indiferent dacă înainte a fost sau nu un token spațiu, iar restul compilatorului să fie scris a.î. să ignore acest ultim token furnizat.

Aspecte de implementare

De exemplu la analizarea codului $w = \underline{\texttt{abc=(12+x30)+1;␣␣y␣=abc;}}$ din exemplele precedente, scannerul va furniza interpretarea:

$\langle \underline{\texttt{abc}}, 1 \rangle$ $\langle \underline{\texttt{=}}, 5 \rangle$ $\langle \underline{\texttt{(}}, 5 \rangle$ $\langle \underline{\texttt{12}}, 2 \rangle$ $\langle \underline{\texttt{+}}, 5 \rangle$ $\langle \underline{\texttt{x30}}, 1 \rangle$ $\langle \underline{\texttt{)}}, 5 \rangle$ $\langle \underline{\texttt{+}}, 5 \rangle$ $\langle \underline{\texttt{1}}, 2 \rangle$ $\langle \underline{\texttt{;}}, 6 \rangle$ $\langle \underline{\texttt{y}}, 1 \rangle$ $\langle \underline{\texttt{=}}, 5 \rangle$ $\langle \underline{\texttt{abc}}, 1 \rangle$ $\langle \underline{\texttt{;}}, 6 \rangle$

sau interpretarea:

$\langle \underline{\texttt{abc}}, 1 \rangle$ $\langle \underline{\texttt{=}}, 5 \rangle$ $\langle \underline{\texttt{(}}, 5 \rangle$ $\langle \underline{\texttt{12}}, 2 \rangle$ $\langle \underline{\texttt{+}}, 5 \rangle$ $\langle \underline{\texttt{x30}}, 1 \rangle$ $\langle \underline{\texttt{)}}, 5 \rangle$ $\langle \underline{\texttt{+}}, 5 \rangle$ $\langle \underline{\texttt{1}}, 2 \rangle$ $\langle \underline{\texttt{;}}, 6 \rangle$ $\langle \underline{\texttt{y}}, 1 \rangle$ $\langle \underline{\texttt{=}}, 5 \rangle$ $\langle \underline{\texttt{abc}}, 1 \rangle$ $\langle \underline{\texttt{;}}, 6 \rangle$ EOF

Analiza lexicală

Exerciții seminar:

1. Aplicați algoritmul ER $\longrightarrow$ AFD pentru ER (a+b)*abb

2. Aplicați algoritmul ER $\longrightarrow$ AFD pentru ER "*com*" din exemplele date.

3. Considerăm clasele de tokeni:
    1. identificatori (ca în exemplele date)
    2. cuvântul cheie: `while`
    3. cuvântul cheie: `for`
    4. operatori: `+`
    5. paranteză deschisă: `(`
    6. paranteză închisă: `)`
    7. delimitator: `;`

   Specificați aceste clase de tokeni prin expresii regulate.
   Construiți AFD unic și funcția *tok* corespunzătoare pentru recunoașterea tokenilor.

4. Faceți demonstrațiile lăsate ca exercițiu în lecție.

Teme laborator:

1. Scrieți un obiect scanner pentru un limbaj de programare uzual, respectând aspectele de implementare menționate în lecție. Integrați acest scanner într-un program care primește la intrare un cod (fișier) sursă și afișază lista tokenilor; pentru fiecare token va afișa tipul și valoarea ca stringuri.
Programul principal va conține un ciclu care la fiecare iterație apeleaza metoda "gettoken()" a scannerului și afișază tokenul returnat. Acest token are doi membri întregi. Stringul corespunzător membrului valoare se poate obține cu o metodă publică ascannerului, care primește întregul ca parametru și returnează o copie a valorii din tabela de stringuri. Stringul corespunzător membrului tip se poate obține cu o altă metodă publică a scannerului ce primește întregul respectiv ca parametru.

2. Scrieți un program care primește la intrare niște ER și o listă de cuvinte cheie și generează codul unui obiect scanner corespunzător (care să se poată intergra de exemplu într-un program ca la tema 1). Se presupune predefinit tipul "identificator".