Limbaje formale, automate şi compilatoare

Curs 7

Limbaje formale şi automate

- Limbaje de tipul 3
  - Gramatici regulate
  - Automate finite
    - Deterministe
    - Nedeterministe
  - Expresii regulate
    - a, $a \in \Sigma$, ϵ, $\emptyset$
    - $E_1.E_2$, $E_1|E_2$, E_1^*, (E_1)
- Limbaje de tipul 2
  - Gramatici de tipul 2

Plan

- Istoric
- Paşii compilării
- Analiza lexicală
  - Descriere lexicală
  - Interpretare
  - Interpretare orientată dreapta
  - Descriere lexicală bine formată
- Analiza sintactică ascendentă
  - Parser ascendent general
  - Analiză LR
  - LR(0)

Istoric – 1940

- Programe scrise în instrucțiuni procesor
- Calculatoare puține
- Programatori puțini

Istoric - 1950

- Fortran (1957):
  - Primul compilator (expresii aritmetice, instrucțiuni, proceduri)
  - Încă este folosit pentru aplicații complexe computațional sau pentru testarea performanței
- Algol (1958):
  - Gramatici BNF (Backus-Naur Normal Form), bloc de instrucțiuni, recursie
  - Precursorul sintaxei curente
- Lisp (1958)
  - Programare funcțională
  - Structuri aroborescente, gestiunea automată a spațiului de stocare, dynamic typing
- COBOL (1959)
  - Sintaxă similară limbii engleze
  - Business oriented
  - Pune accent pe citire și scriere de date in format text și numeric

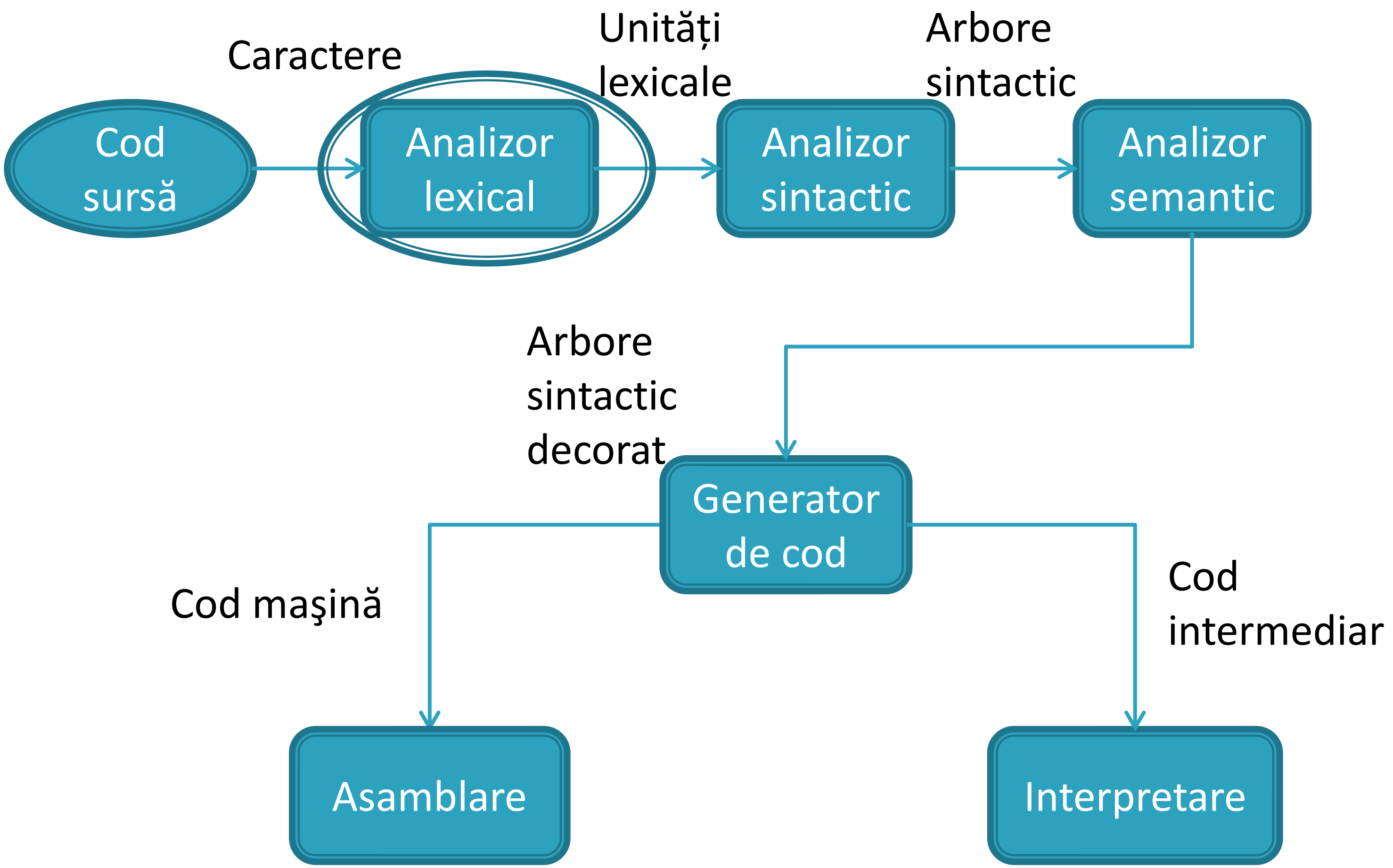
Istoric – 1960 – 1970

- Simula (1965)
  - Bazat pe ALGOL 60
  - Primul limbaj orientat obiect
  - Obiecte, clase, moștenire, funcții virtuale, etc.
- Programare structurată (1968)
  - Edsger Dijkstra – GOTO Considered Harmful
- Pascal (1970)
- C (1973)
  - IRQ, variabile dinamice, multitasking

Istoric – 1980 – prezent

- ADA (1980)
  - primul limbaj standardizat
- Objective C (1984)
  - Inspirat de Smalltalk
  - Orientare obiect
- C++ (1985)
  - *C with Classes*;
  - Orientare-obiect, excepții, template-uri
  - Inspirat de Simula
- Java (1995)
  - just-in-time compilation
- C# (2000)
  - Tehnologia .NET

Compilare


Caractere
Unități
lexicale
Arbore
sintactic
Cod
sursă
Analizor
lexical
Analizor
sintactic
Analizor
semantic
Arbore
sintactic
decorat
Generator
de cod
Cod maşină
Cod
intermediar
Asamblare
Interpretare

Analiza lexicală

- **Def. 1** Fie Σ un alfabet (al unui limbaj de programare). O *descriere lexicală* peste Σ este o expresie regulată $E = (E_1 | E_2 | \ldots | E_n)^+$, unde n este numărul unităților lexicale, iar E_i descrie o unitate lexicală, $1 \le i \le n$.

- **Def. 2** Fie E o descriere lexicală peste Σ ce conține n unități lexicale şi $w \in \Sigma^+$. Cuvântul w este *corect relativ la descrierea* E dacă $w \in L(E)$. O *interpretare* a cuvântului $w \in L(E)$ este o secvență de perechi $(u_1, k_1), (u_2, k_2), \ldots, (u_m, k_m)$, unde $w = u_1 u_2 \ldots u_m$, $u_i \cdot L(E_{ki})$ $1 \le i \le m$, $1 \le ki \le n$.

Exemplu

- w = `alpha := beta = 542`
- Interpretări ale cuvântului w:
  - (`alpha`, Id), (`:=`, Asignare), (`beta`, Id), (`=`, Egal), (`542`, Intreg)
  - (`alp`, Id), (`ha`, Id), (`:=`, Asignare), (`beta`, Id), (`=`, Egal), (`542`, Intreg)
  - (`alpha`, Id), (`:`, Dp), (`=`, Egal), (`beta`, Id), (`=`, Egal), (`542`, Intreg)

Analiza lexicală

- **Def. 3** Fie E o descriere lexicală peste Σ şi $w \in L(E)$. O interpretare a cuvântului w, $(u_1, k_1)(u_2, k_2), \dots(u_m, k_m)$, este *interpretare drept -orientată* dacă $(\forall\ i)\ 1 \leq i \leq m$, are loc:

  $|u_i| = \max\{|v|, v \in L(E_1| E_2|\dots| E_n) \cap \mathrm{Pref}(u_i u_{i+1}\dots u_m)\}$.

  (unde Pref(w) este mulțimea prefixelor cuvântului w).

- Există descrieri lexicale E în care nu orice cuvânt din L(E) admite o interpretare drept-orientată.
- $E = (a \mid ab \mid bc)^+$ şi $w = abc$.

Analiza lexicală

- **Def. 4** O descriere lexicală E este *bine -formată* dacă orice cuvânt w din limbajul L(E) are exact o interpretare drept-orientată.

- **Teoremă** Dată o descriere lexicală E este decidabil dacă E este bine formată

- **Def. 5** Fie E o descriere lexicală bine formată peste Σ. Un *analizor lexical* (*scanner*) pentru E este un program ce recunoaşte limbajul L(E) şi produce, pentru fiecare $w \in L(E)$, interpretarea sa drept-orientată.

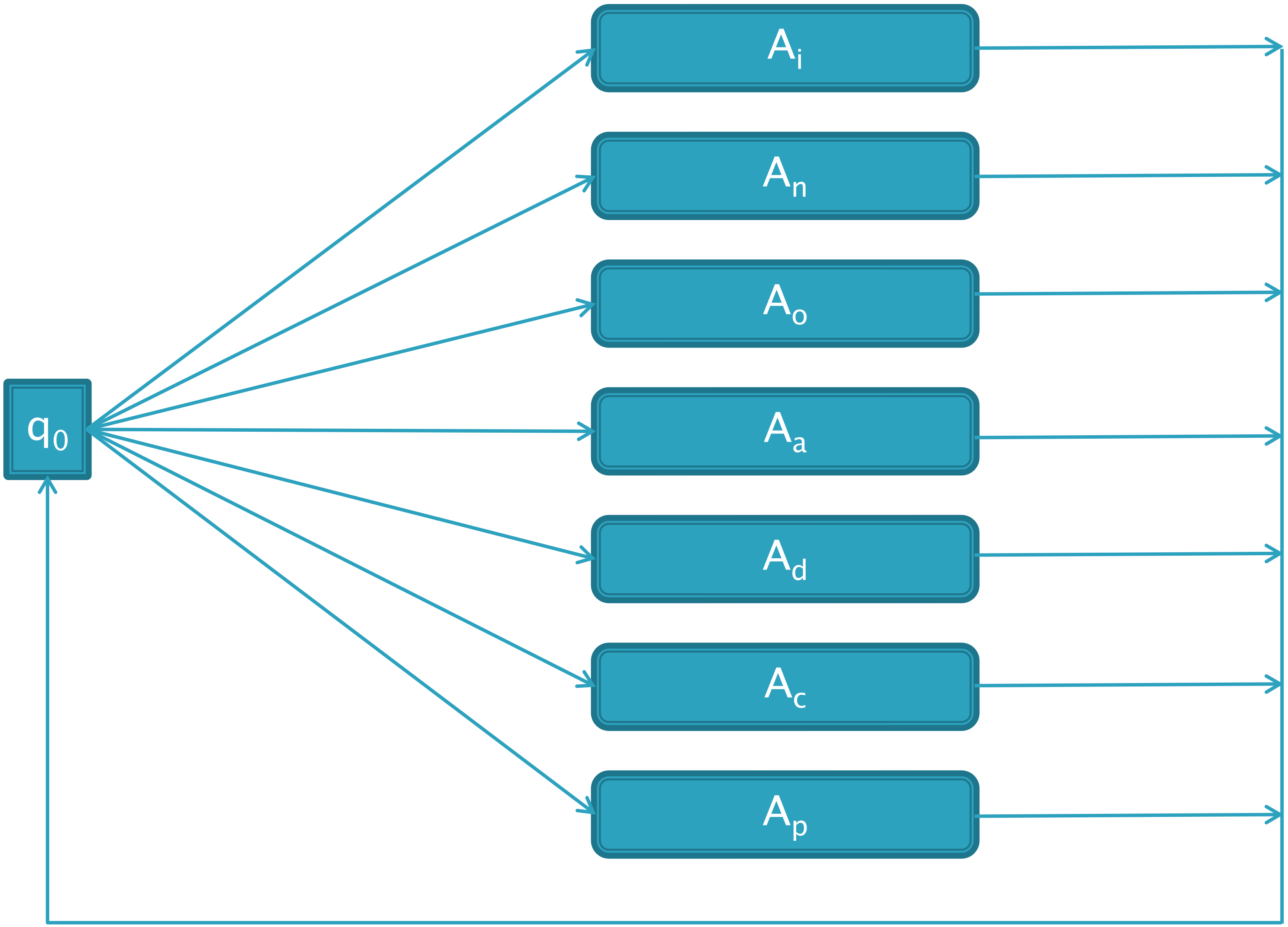
Analiza lexicală

- Fie o descriere lexicală E peste Σ. Crearea unui analizor lexical pentru E inseamnă:

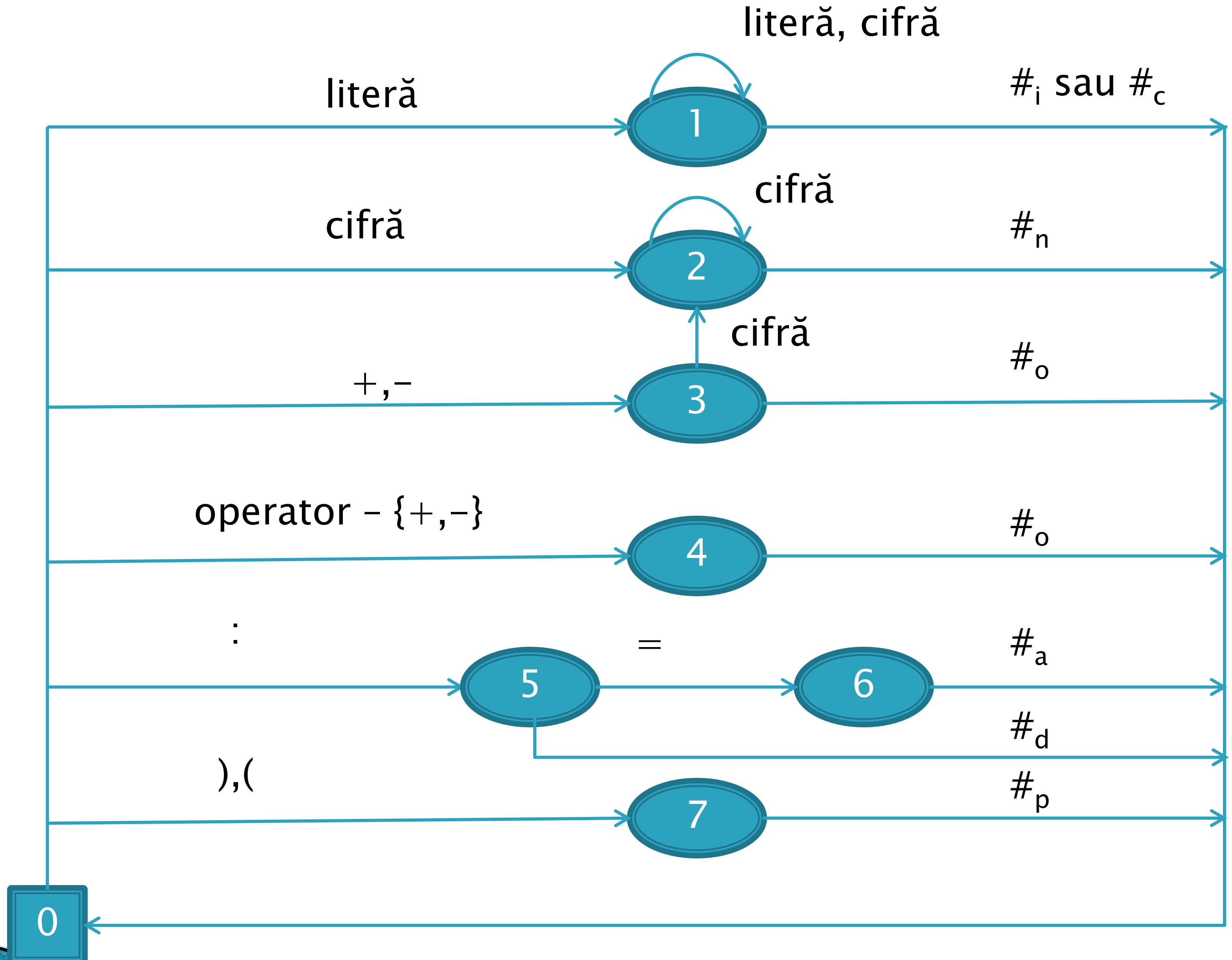
  - 1. Se construieşte automatul finit echivalent A
  - 2. Din A se obține automatul determinist echivalent cu E, fie acesta A'.
  - 3. (Opțional) Automatul minimal echivalent cu A'.
  - 4. Implementarea automatului A'.

Exemplu de analizor lexical

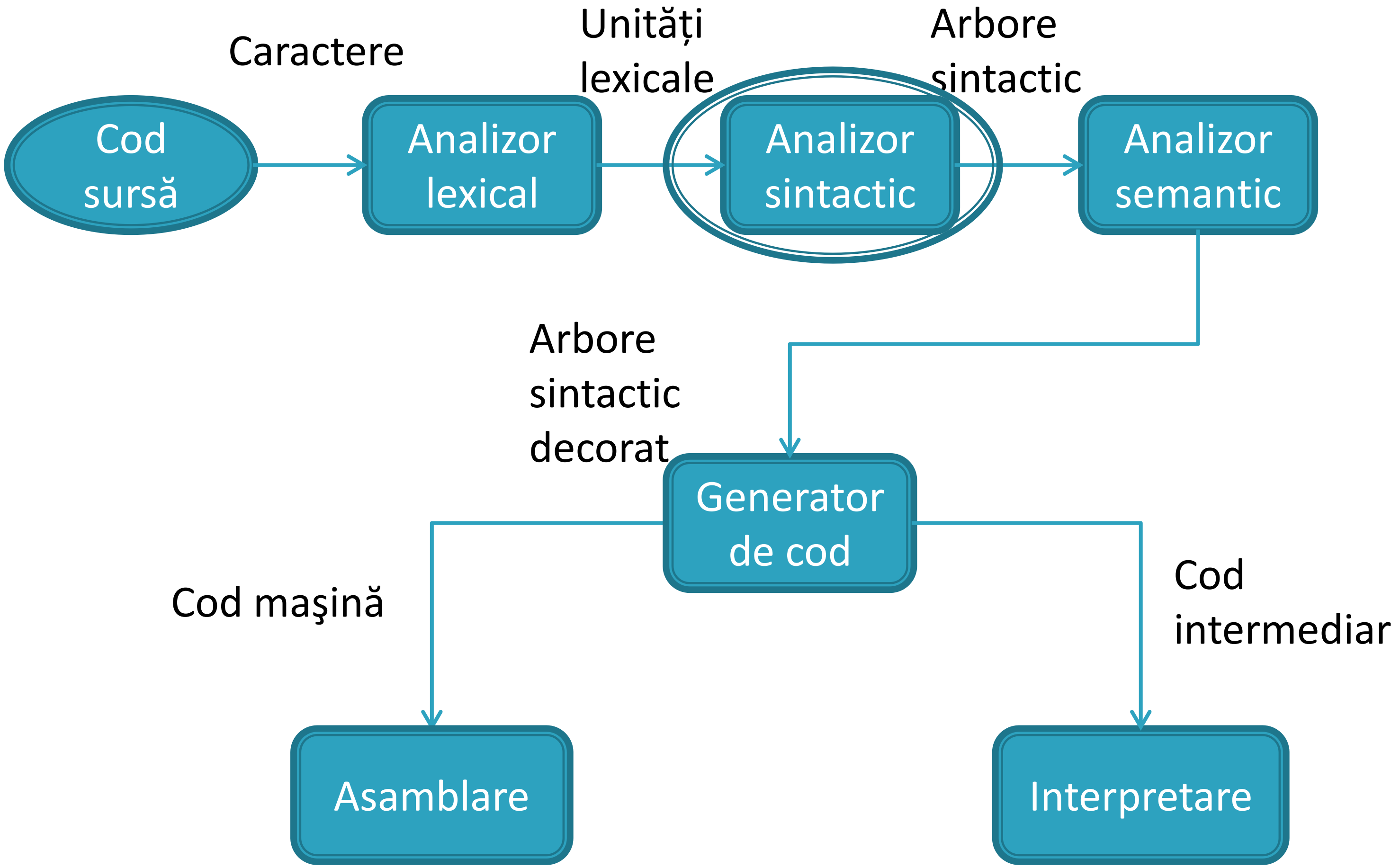
- Fie descrierea lexicală:
  - litera → a | b |…|z
  - cifra → 0 | 1 |…| 9
  - identificator → litera (litera| cifra)*
  - semn → + | -
  - numar →(semn | ϵ) cifra+
  - operator → + | -| * | / | < | > | <= | >= | < >
  - asignare → :=
  - doua_puncte → :
  - cuvinte_rezervate → if| then|else
  - paranteze →) | (



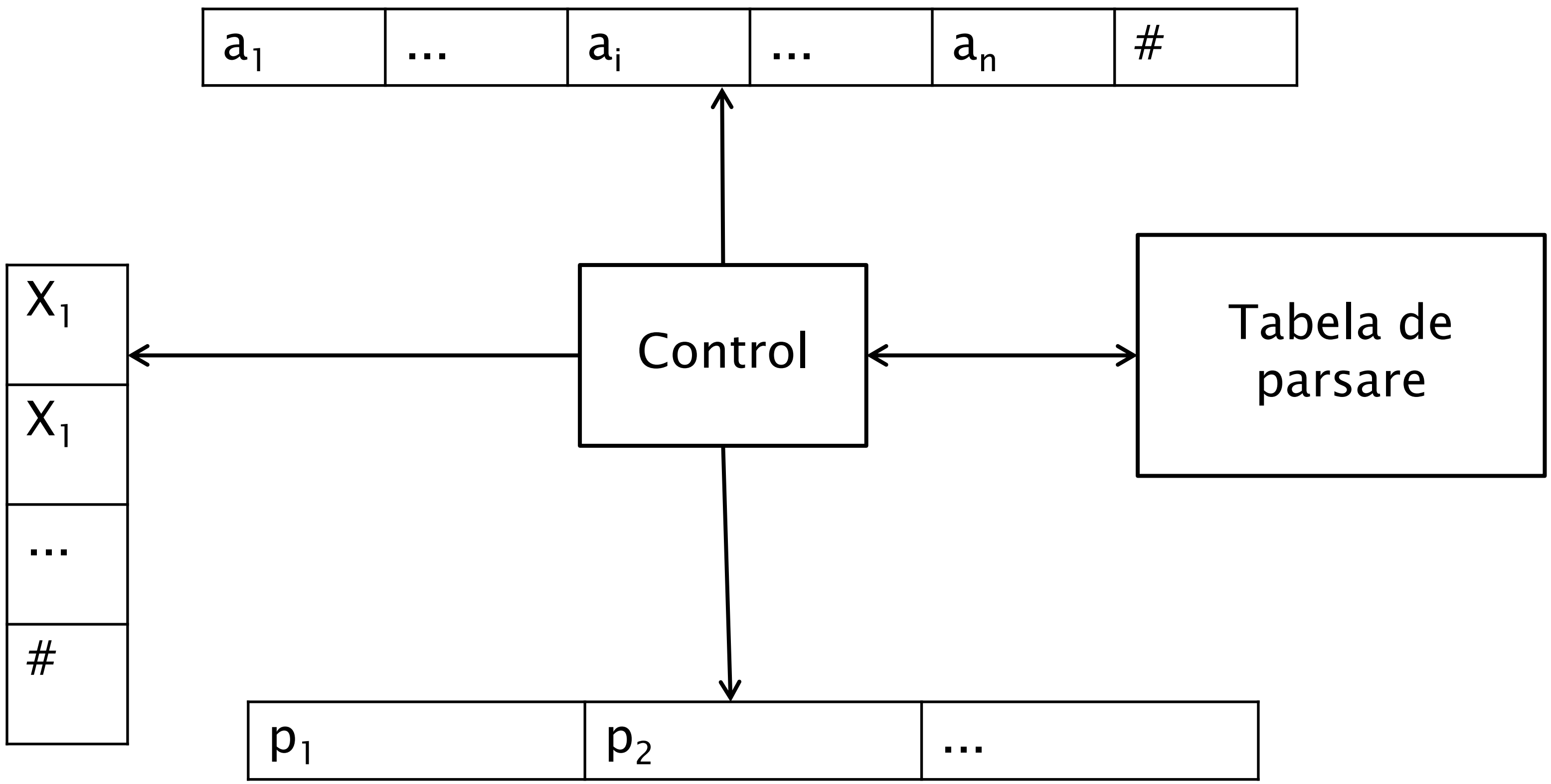
q_0
A_i
A_n
A_o
A_a
A_d
A_c
A_p


literă, cifră
literă
$\#_i$ sau $\#_c$
1
cifră
cifră
$\#_n$
2
cifră
+,−
$\#_o$
3
operator − {+,−}
$\#_o$
4
:
=
$\#_a$
5
6
$\#_d$
),(
$\#_p$
7
0

Compilare


Caractere
Unități lexicale
Arbore sintactic
Cod sursă
Analizor lexical
Analizor sintactic
Analizor semantic
Arbore sintactic decorat
Generator de cod
Cod maşină
Cod intermediar
Asamblare
Interpretare

Parser ascendent general



Configuraţii

- O configuraţie ($\#\gamma$, $u\#$, π) este interpretată în felul următor:
  - –$\#\gamma$ este conţinutul stivei cu simbolul # la baza.
  - –$u\#$ este conţinutul intrării.
  - –π este conţinutul ieşirii.
- $C_0 = \{(\#, w\#, \varepsilon) | w \in T^*\}$ mulţimea configuraţiilor iniţiale.

Tranziţii

- Parserul ascendent ataşat gramaticii G este perechea (C_0, $\vdash$) unde C_0 este mulţimea configuraţiilor iniţiale, iar $\vdash$ este o relaţie de tranziţie definită astfel:
  - $(\# \gamma, au\#, \pi) \vdash (\# \gamma a, u\#, \pi)$ (***deplasare)*** pentru orice $\gamma \in \Sigma^*$, $a \in T$, $u \in T^*$, $\pi \in P^*$.
  - $(\#\alpha\beta, u\#, \pi) \vdash (\#\alpha A, u\#, \pi r)$ dacă $r = A \rightarrow \beta$ (***reducere).***
  - Configuraţia $(\#S, \#, \pi)$ unde $\pi \neq \varepsilon$, se numeşte ***configuraţie de acceptare.***
  - Orice configuraţie, diferită de cea de acceptare, care nu este în relaţia $\vdash$ cu nici o altă configuraţie este o configuraţie eroare.
- Parsere de deplasare/reducere.

Exemplu

- Fie gramatica $S \to aSb \mid \varepsilon$. Tranziţiile sunt:
  - $(\#\gamma, u\#, \pi) \vdash (\#\gamma S, u\#, \pi 2)$
  - $(\#\gamma aSb, u\#, \pi) \vdash (\#\gamma S, u\#, \pi 1)$
  - $(\#\gamma, au\#, \pi) \vdash (\#\gamma a, u\#, \pi)$
  - $(\#\gamma, bu\#, \pi) \vdash (\#\gamma b, u\#, \pi)$
- O succesiune de tranziţii se numeşte calcul
  - $(\#, \#, \varepsilon) \vdash (\#S, \#, 2)$
  - $(\#, aabb\#, \varepsilon) \vdash (\#a, abb\#, \varepsilon) \vdash (\#aa, bb\#, \varepsilon) \vdash (\#aaS, bb\#, 2) \vdash (\#aaSb, b\#, 2) \vdash (\#aS, b\#, 21) \vdash (\#aSb, \#, 21) \vdash (\#S, \#, 211)$

Conflicte

- Parserul este nedeterminist:
  - Pentru o configuraţie de tipul $(\#\alpha\beta, au\#, \pi)$, $S\rightarrow\beta$, există două posibilităţi (conflict **deplasare/reducere**):
    - $(\#\alpha\beta, au\#, \pi) \vdash (\#\alpha S, au\#, \pi r)$ (reducere cu $S\rightarrow\beta$)
    - $(\#\alpha\beta, au\#, \pi) \vdash (\#\ \alpha\beta a, u\#, \pi)$ (deplasare)
  - Pentru o configuraţie $(\#\gamma, u\#, \pi)$ cu $\gamma=\alpha_1\beta_1=\alpha_2\beta_2$ şi $A\rightarrow\beta_1$, $B\rightarrow\beta_2$, reguli (conflict **reducere/reducere**)
    - $(\#\alpha_1\beta_1, u\#, \pi) \vdash (\#\ \alpha_1 A, au\#, \pi r_1)$
    - $(\#\alpha_2\beta_2, u\#, \pi) \vdash (\#\ \alpha_2 B, au\#, \pi r_2)$

Corectitudine

- Spunem că un cuvânt $w \in T^*$ este acceptat de un parser ascendent dacă există măcar un calcul de forma
  - $(\#, w\#, \varepsilon) \vdash^{+} (\#S, \#, \pi)$
- Pentru ca parserul descris să fie corect, trebuie ca el să accepte toate cuvintele din L(G) şi numai pe acestea.
- **Teorema**
  - Parserul ascendent general ataşat unei gramatici G este corect: pentru orice $w \in T^*$, $w \in L(G)$ dacă şi numai dacă în parser are loc calculul $(\#, w\#, \varepsilon) \vdash^{+} (\#S, \#, \pi)$.

Analiza sintactică LR

- Gramatici **LR(k)**:**L**eft to right scanning of the input, constructing a **R**ightmost derivation in reverse, using **k** symbols lookahead
- Definiţie
  - O gramatică G se numeşte gramatică LR(k), $k \geq 0$, dacă pentru orice două derivări de forma:
    - $S' \Rightarrow S \;{}_{dr}\!\Rightarrow^{*} \alpha A u \;{}_{dr}\!\Rightarrow \alpha\beta u = \delta u$
    - $S' \Rightarrow S \;{}_{dr}\!\Rightarrow^{*} \alpha' A' u' \;{}_{dr}\!\Rightarrow \alpha'\beta' u' = \alpha\beta v = \delta v$
  - pentru care $k{:}u = k{:}v$, are loc $\alpha=\alpha'$, $\beta=\beta'$, $A=A'$

Analiza sintactică LR

- **Teorema 1**
  - Dacă G este gramatică LR(k), $k \geq 0$, atunci G este neambiguă.
- Un limbaj L este (în clasa) $\mathcal{LR}$(k) dacă există o gramatică LR(k) care îl generează
- **Teorema 2**
  - Orice limbaj $\mathcal{LR}$(k) este limbaj de tip 2 determinist.
- **Teorema 3**
  - Orice limbaj de tip 2 determinist este limbaj LR(1).
- **Teorema 4**
  - Pentru orice limbaj $\mathcal{LR}$(k), $k \geq 1$, există o gramatică LR(1) care generează acest limbaj, adică $LR(0) \subset LR(1) = LR(k)$, $k \geq 1$.

Gramatici LR(0)

- **Definiţie**
  - Fie G = (V, T, S, P) o gramatică independentă de context redusă. Să presupunem că simbolul • nu este în Σ. Un **articol** pentru gramatica G este o producţie $A\rightarrow\gamma$ în care s-a adăugat simbolul • într-o anume poziţie din γ. Notăm un articol prin $A\rightarrow\alpha\bullet\beta$ dacă $\gamma = \alpha\beta$. Un articol în care • este pe ultima poziţie se numeşte **articol complet**.
- **Definiţie**
  - Un **prefix viabil** pentru gramatica G este orice prefix al unui cuvânt $\alpha\beta$ dacă $S \;{}_{dr}\!\Rightarrow^* \alpha Au \;{}_{dr}\!\Rightarrow \alpha\beta u$. Dacă $\beta = \beta_1\beta_2$ şi $\varphi = \alpha\beta_1$ spunem că articolul $A \rightarrow \beta_1\bullet\beta_2$ este **valid** pentru **prefixul viabil** φ.

Exemplu

- Exemplu $S \to A$, $A \to aAa \mid bAb \mid c \mid \varepsilon$.
  - Articole: $S\to\bullet A$, $S\to A\bullet$, $A\to\bullet aAa$, $A\to a\bullet Aa$, $A\to aA\bullet a$, $A\to aAa\bullet$, $A\to\bullet bAb$, $A\to b\bullet Ab$, $A\to bA\bullet b$, $A\to bAb\bullet$, $A\to\bullet c$, $A\to c\bullet$, $A\to\ \bullet$.
- Articole valide pentru prefixe viabile:

Prefixul viabil	**Articole valide**	**Derivarea corespunzătoare**
ab	$A\to b\bullet Ab$ $A\to\bullet aAa$ $A\to\bullet bAb$	$S\Rightarrow A\Rightarrow aAa\Rightarrow abAba$ $S\Rightarrow A\Rightarrow aAa\Rightarrow abAba\Rightarrow abaAaba$ $S\Rightarrow A\Rightarrow aAa\Rightarrow abAba\Rightarrow abbAbba$
ε	$S\to\bullet A$ $A\to\bullet bAb$ $A\to\bullet c$	$S\Rightarrow A$ $S\Rightarrow A\Rightarrow bAb$ $S\Rightarrow A\Rightarrow c$

Gramatici LR(0)

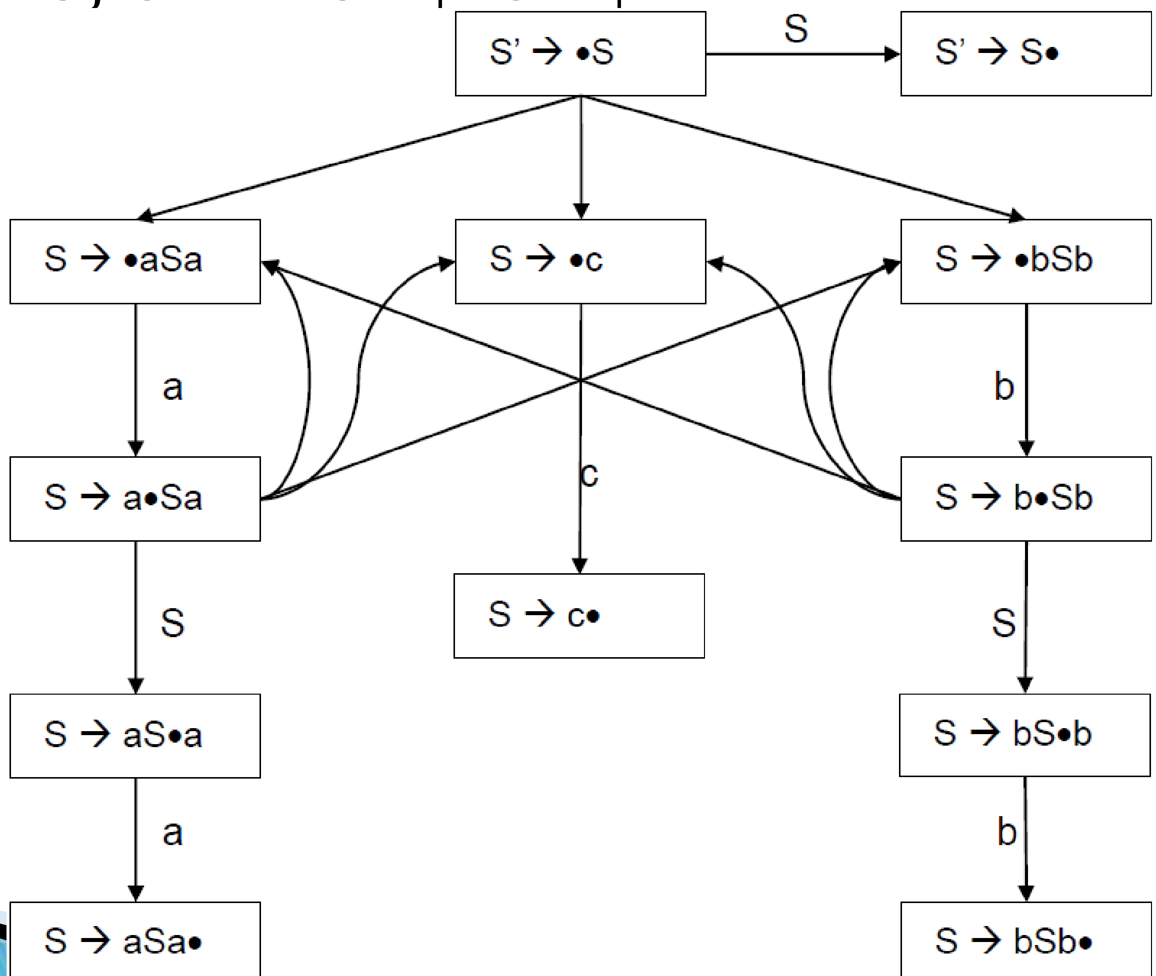
- **Lema**
  - Fie G o gramatică şi $A \to \beta_1 \bullet B\beta_2$ un articol valid pentru prefixul viabil γ. Atunci, oricare ar fi producţia $B \to \beta$, articolul $B \to \bullet\beta$ este valid pentru γ.
- **Teorema** (caracterizare LR(0))
  - Gramatica G este gramatică LR(0) dacă şi numai dacă, oricare ar fi prefixul viabil γ, sunt îndeplinite condiţiile:
    - 1.nu există două articole complete valide pentru γ.
    - 2.dacă articolul $A \to \beta\bullet$ este valid pentru γ, nu există nici un articol $B \to \beta_1 \bullet a\beta_2$, $a \in T$, valid pentru γ.

Gramatici LR(0)

- **Teorema**
  - Fie G = (V, T, S, P) o gramatică independentă de context. Mulţimea prefixelor viabile pentru gramatica G este limbaj regulat.
- **Demonstraţie**
  - G' este G la care se adaugă $S' \to S$.
  - $M = (Q, \Sigma, \delta, q_0, Q)$, unde:
    - Q este mulţimea articolelor gramaticii G',
    - $\Sigma = V \cup T$, $q_0 = S' \to \bullet S$
    - $\delta: Q x(\Sigma \cup \{\varepsilon\}) \to 2^Q$ definită astfel:
      - $\delta(A \to \alpha \bullet B\beta, \varepsilon) = \{B \to \bullet\alpha \mid B \to \gamma \in P\}$.
      - $\delta(A \to \alpha \bullet X\beta, X) = \{ A \to \alpha X \bullet \beta\}$, $X \in \Sigma$.
      - $\delta(A \to \alpha \bullet a\beta, \varepsilon) = \emptyset$, $\forall a \in T$.
      - $\delta(A \to \alpha \bullet X\beta, Y) = \emptyset$, $\forall X,Y \in \Sigma$ cu $X \neq Y$.
- Se arată că are loc:
  - $(A \to \alpha \bullet \beta \in \delta\,\hat{}(q_0, \gamma) \Leftrightarrow \gamma$ este prefix viabil şi $A \to \alpha \bullet \beta$ este valid pentru γ.

Exemplu

- S’ → S, S → aSa |bSb | c



Automatul LR(0)

- Algoritmul 1(procedura închidere(t))
- Intrare:
  - Gramatica G = (V, T, S, P);
  - Mulţimea t de articole din gramatica G;
- Ieşire: $t' = \text{închidere}(t) = \{q \in Q \mid \exists p \in t,\ q \in \delta(p,\varepsilon)\} = \delta(t,\varepsilon)$

Automatul LR(0)

- t’ = t ; flag = true;
- while(flag) {
  - flag = false;
  - for ($A\rightarrow\alpha\bullet B\beta \in$ t’) {
    - for ($B\rightarrow\gamma \in P$)
      - if ($B\rightarrow\bullet\gamma\notin$t’) {
      - t’ = t’$\cup\{B\rightarrow\bullet\gamma\}$;
      - flag = true;
      - }//endif
    - }//endforB
  - }//endforA
- }//endwhile
- return t’;

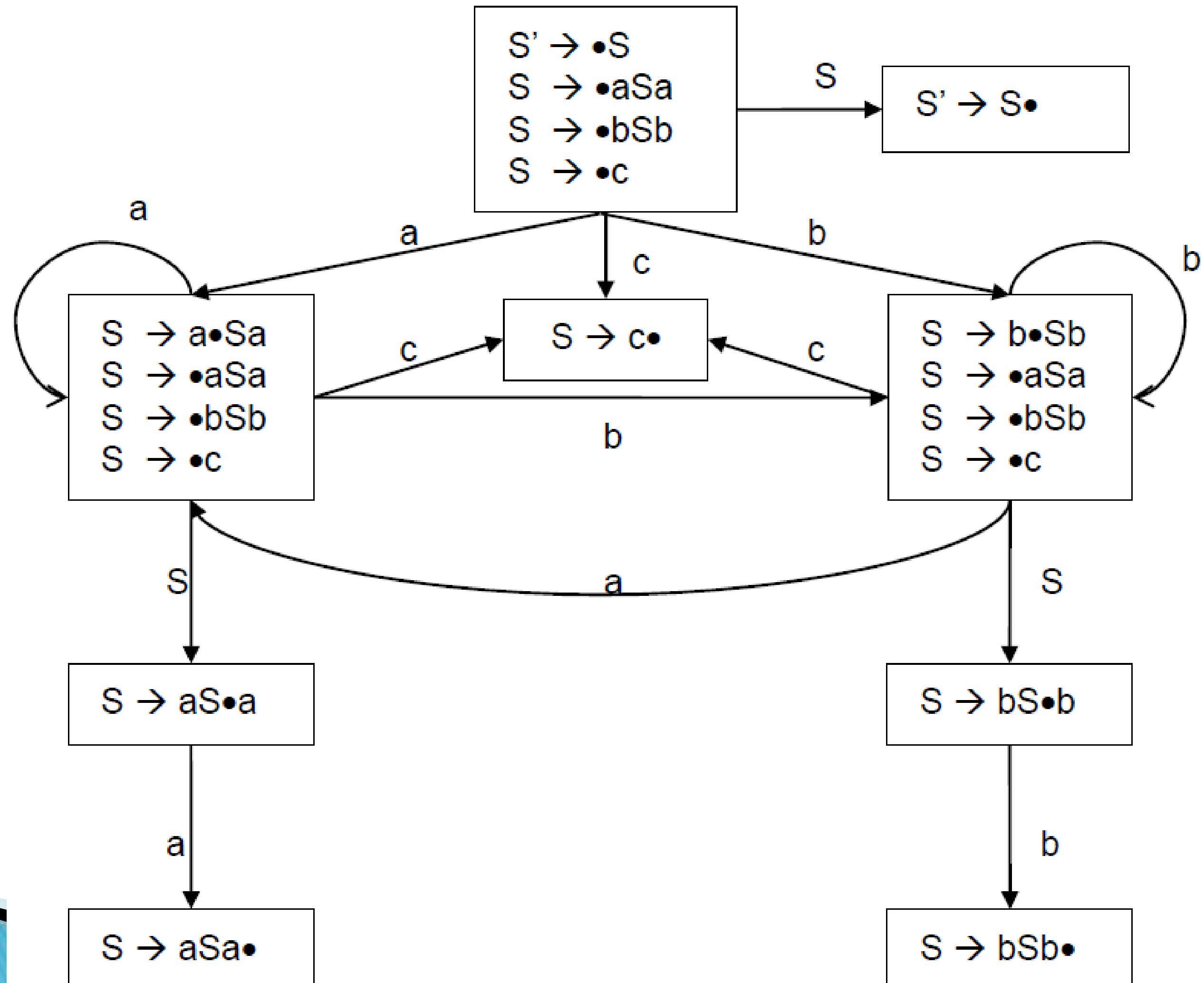
Automatul LR(0)

- Algoritmul 2 Automatul LR(0)
  - Intrare: Gramatica G = (N, T, S, P) la care s-a adăugat S’ → S;
  - Ieşire: Automatul determinist M = (T, Σ, g, t_0, T) echivalent cu M.

- $t0$=închidere($S' \to \bullet S$); $T=\{t_0\}$; marcat(t_0)=false;
- while($\exists$ t ϵ T && !marcat(t)) { // marcat(t) = false
  - for(X ϵ Σ) {// $\Sigma = N \cup T$
    - t' = $\emptyset$;
    - for($A \to \alpha \bullet X\beta$ ϵt)
      - t' = t' $\cup$ $\{B \to \alpha X \bullet \beta \mid A \to \alpha \bullet X\beta$ ϵt$\}$;
      - if(t'$\neq\emptyset$){
        - t' = închidere(t');
        - if(t'$\notin$T) {
          - T = T$\cup${ t' };
          - marcat(t') = false;
        - }//endif
      - g(t, X) = t';
      - }//endif
    - }//endfor
  - }//endfor
  - marcat(t) = true;
- }// endwhile

Automatul LR(0) – Exemplu

- S’ → S, S → aSa | bSb | c



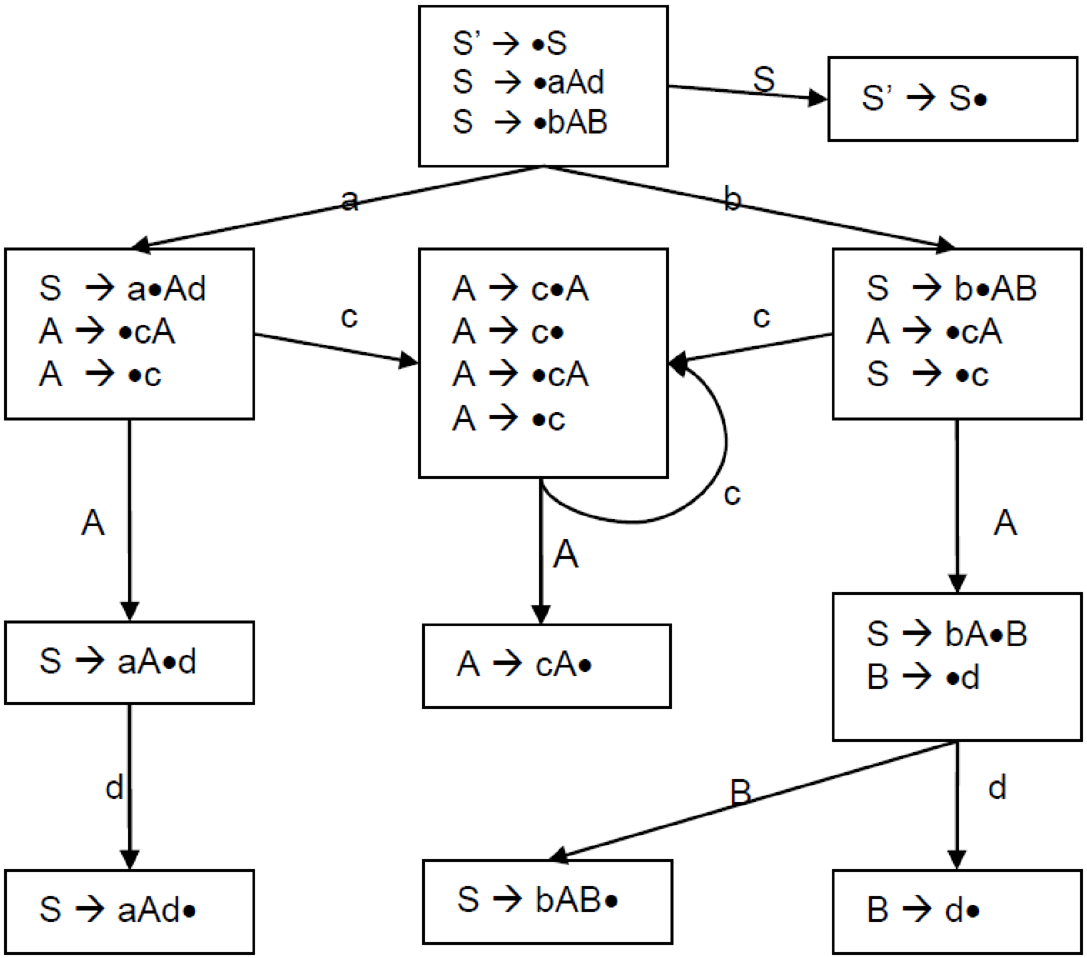
Test LR(0)

- **Definiţie** Fie G o gramatică şi M automatul LR(0) ataşat lui G.
  - Spunem că o stare a lui M are un conflict **reducere/reducere** dacă ea conţine două articole complete distincte $A \to \alpha\bullet$, $B \to \beta\bullet$.
  - Spunem că o stare a lui M are un conflict **deplasare/reducere** dacă ea conţine un articol complet $A \to \alpha\bullet$ şi un articol cu terminal după punct de forma $B \to \beta\bullet a\gamma$.
  - Spunem că o stare este **consistentă** dacă ea nu conţine conflicte şi este **inconsistentă** în caz contrar.

- **Teorema** Fie G o gramatică şi M automatul său LR(0). Gramatica G este LR(0) dacă şi numai dacă automatul M nu conţine stări inconsistente

Exemplu

- S → aAd | bAB, A → cA | c, B → d



Algoritmul de analiză LR(0)

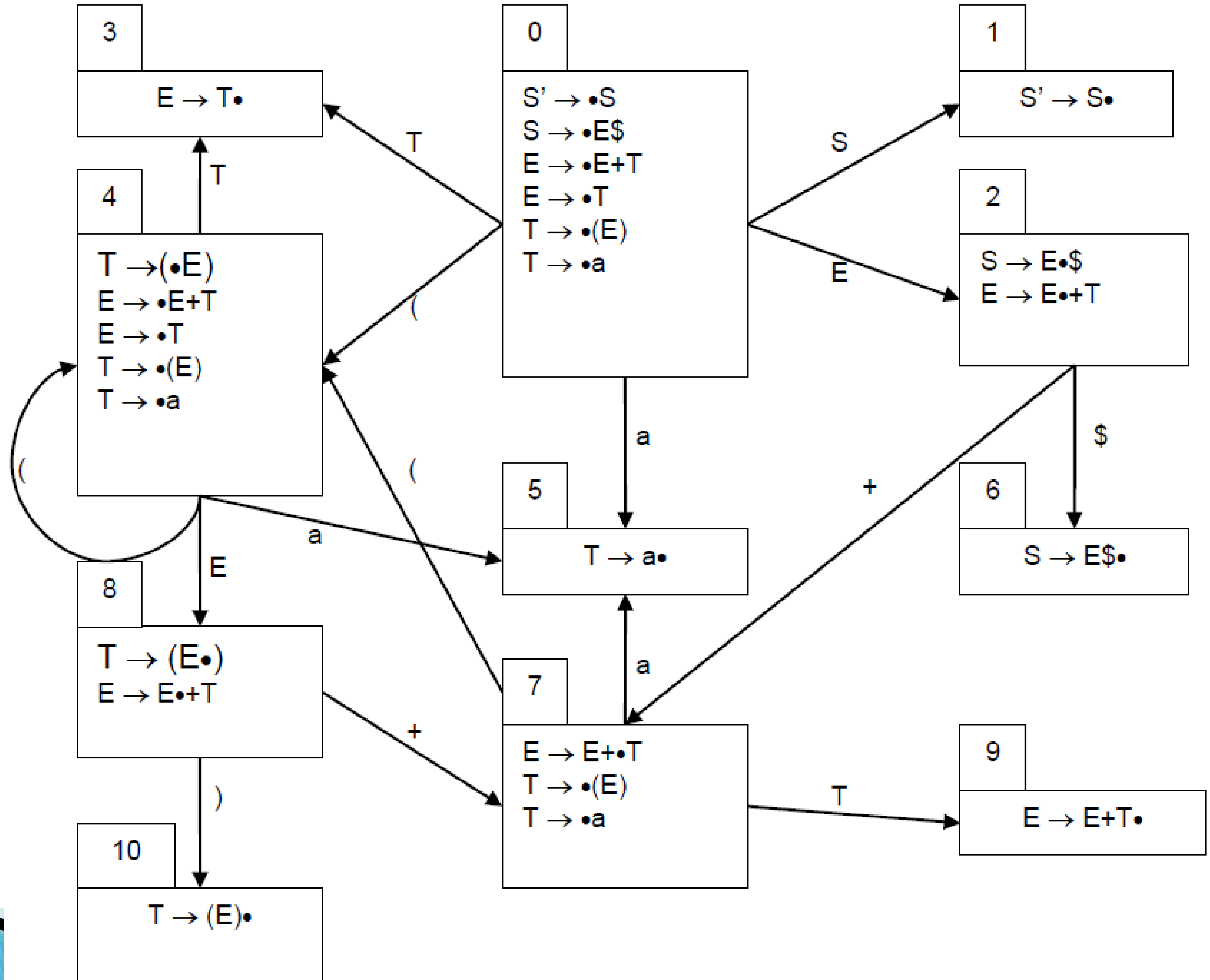
- Tabela de parsare coincide cu automatul LR(0), M.
- Configuraţie: $(\sigma, u\#, \pi)$ unde $\sigma \epsilon t_0 T^*$, $u \epsilon T^*$, $\pi \epsilon P^*$.
- Configuraţia iniţială este $(t_0, w\#, \varepsilon)$,
- Tranziţiile:
  - Deplasare: $(\sigma t, au\#, \pi) \vdash (\sigma t t', u\#, \pi)$ dacă $g(t, a) = t'$.
  - Reducere: $(\sigma t \sigma' t', u\#, \pi) \vdash (\sigma t t'', u\#, \pi r)$ dacă $A \to \beta\bullet \in t'$, $r = A \to \beta$, $| \sigma' t' | = | \beta |$ şi $t'' = g(t, A)$.
  - Acceptare: $(t_0 t_1, \#, \pi)$ este configuraţia de acceptare dacă $S' \to S\bullet \in t1$, π este parsarea acestuia.
  - Eroare: o configuraţie căreia nu i se poate aplica nici o tranziţie

Algoritmul de analiză LR(0)

```
char ps[]= "w#"; //ps este sirul de intrare w
i = 0; // pozitia in sirul de intrare
STIVA.push(t0); // se initializeaza stiva cu t0
while(true) { // se repeta pana la succes sau eroare
    t = STIVA.top();
    a = ps[i] // a este simbolul curent din intrare
    if( g(t, a)≠Ø { //deplasare
        STIVA.push(g(t, a));
        i++; //se inainteaza in intrare
        }
    else {
    if(A → X1X2…Xm •  ∈ t){
        if(A == „S")
            if(a == „#")exit( "acceptare");
            else exit("eroare");
        else // reducere
            for( i = 1; i <= m; i++) STIVA.pop();
                STIVA.push(g(top(STIVA), A));
        } //endif
    else exit("eroare");
    }//endelse
}//endwhile
```

Exemplu

- S’ → S S → E$ E → E+T T →(E) E → T T → a



Exemplu

- S’ → S S → E$ E → E+T T →(E) E → T T → a

Stiva	Intrare	Acţiune	Ieşire
0	a+(a+a)$#	deplasare	
05	+(a+a)$#	reducere	T → a
03	+(a+a)$#	reducere	E → T
02	+(a+a)$#	deplasare	
027	(a+a)$#	deplasare	
0274	a+a)$#	deplasare	
02745	+a)$#	reducere	T → a
02743	+a)$#	reducere	E → T
02748	+a)$#	deplasare	
027487	a)$#	deplasare	
0274875	)$#	reducere	T → a
0274879	)$#	reducere	E → E+T
02748	)$#	deplasare	
02748'10'	$#	reducere	T → (E)
0279	$#	reducere	E → E+T
02	$#	deplasare	
026	#	reducere	S → E$
01	#	acceptare	

Corectitudinea parserului LR(0)

- **Lema 1, 2** Fie G = (N, T, S, P) o gramatică LR(0), $t_0\sigma$, $t_0\tau$ drumuri în automatul LR(0) etichetate cu φ respectiv γ şi u, v ϵ T*. Atunci, dacă în parserul LR(0) are loc $(t_0\sigma, uv\#, \varepsilon) \vdash^+ (t_0\tau, v\#, \pi)$, atunci în G are loc derivarea $\varphi \;{}_{dr}\!\Rightarrow_\pi u$ şi reciproc.

- **Teoremă** Dacă G este gramatică LR(0) atunci, oricare ar fi cuvântul de intrare w ϵ T*, parserul LR(0) ajunge la configuraţia de acceptare pentru w, adică $(t_0\sigma, uv\#, \varepsilon) \vdash^+ (t_0\tau, v\#, \pi)$ dacă şi numai dacă $\varphi \;{}_{dr}\!\Rightarrow_\pi u$

Bibliografie

- Grigoraş Gh., *Construcția compilatoarelor. Algoritmi fundamentali*, Editura Universității “Alexandru Ioan Cuza”, Iaşi, 2005