Gramática tipo 2 (livre de contexto)

Uma Gramática $G = \langle N, \Sigma, P, S \rangle$ é uma gramática livre de contexto (g.l.c.) ou do tipo 2 sss P possui somente produções da forma:

$A \rightarrow \alpha$,

em que $A \in N$ e $\alpha \in (N \cup \Sigma)^*$

Ex. $G_3 = \langle \{S\}, \{0,1\}, \mathbf{P}, S \rangle$, em que

$P = \{$
$S \rightarrow 0S0,$
$S \rightarrow 1S1,$
$S \rightarrow \varepsilon$
$\}$

Gramática tipo 2 (livre de contexto)

As **gramáticas** livres de contexto (g.l.c.) caracterizam as **linguagens** livres de contexto.

Uma **linguagem** L é livre de contexto ou do tipo 2 sss L = L (G) para alguma gramática livre de contexto.

Essas gramáticas são assim chamadas porque uma **regra** **A → β** indica que o não terminal **A**, independentemente, do contexto em que estiver inserido, pode ser substituído por **β**

GLCs – Árvores de Derivação em uma GLC

Para $G = \langle N, \Sigma, P, S \rangle$ definida por:

1) Todo vértice tem um rótulo, que é um símbolo pertencente a $N \cup \Sigma \cup \{\varepsilon\}$.

2) O rótulo raíz é S.

3) Se um vértice interior (não folha) tem um rótulo A, então A deve estar em N

4) Se o vértice com rótulo A possui filhos rotulados com $X_1, X_2, \ldots, X_k$ é uma produção em P.

5) Se um vértice tem rótulo ε, então é uma folha e é o único filho de seu pai.

GLCs – Árvores de Derivação em uma GLC

Exemplo: G_1 = <{S,A}, {a,b}, P,S> definida por:

P = {
S→aAS | a,
A→SbA | SS | ba
}

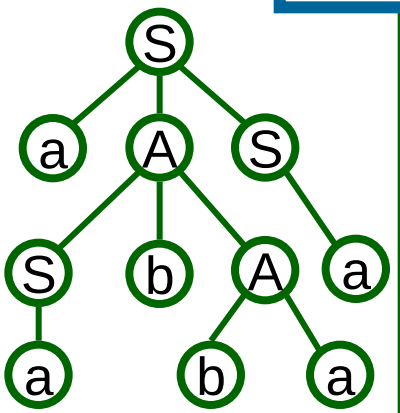
1) Todo vértice tem um rótulo, que é um símbolo pertencente a N U Σ U{ε}.

2) O rótulo raíz é S.

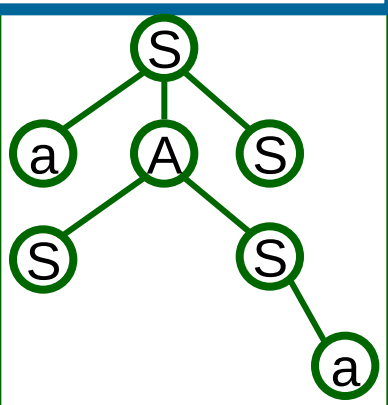
3) Se um vértice interior (não folha) tem um rótulo A, então A deve estar em N

4) Se o vértice com rótulo A possui filhos rotulados com X_1, X_2, ..., X_k é uma produção em P.

5) Se um vértice tem rótulo ε, então é uma folha e é o único filho de seu pai.



Forma sentencial, resultado:
aabbaa



Forma sentencial, resultado:
aSaS

GLCs – Árvores de Derivação em uma GLC

Exemplo: G_2 = <{S}, {a,b}, P,S> definida por:

P = {
S→aSb | ε
}

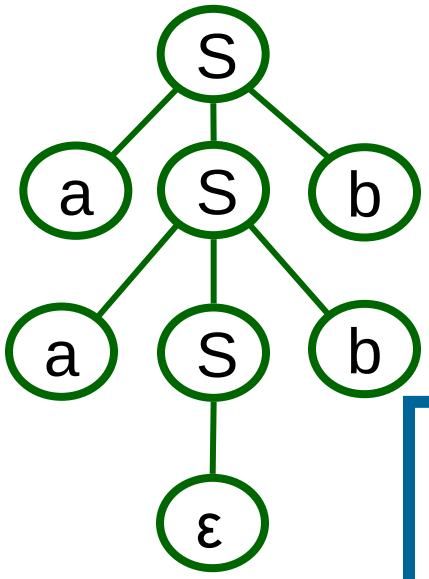
1) Todo vértice tem um rótulo, que é um símbolo pertencente a N U Σ U{ε}.

2) O rótulo raíz é S.

3) Se um vértice interior (não folha) tem um rótulo A, então A deve estar em N

4) Se o vértice com rótulo A possui filhos rotulados com X_1, X_2, ..., X_k é uma produção em P.

5) Se um vértice tem rótulo ε, então é uma folha e é o único filho de seu pai.



Forma sentencial, resultado:
aabb

Árvores de Derivação em uma GLC

RELEITURAS

Uma árvore de derivação em uma gramática G é uma descrição natural da derivação de uma forma sentencial particular de G.

A seqüência de folhas, da esquerda para a direita, constitui uma forma sentencial de G, chamada de resultado da árvore de derivação.

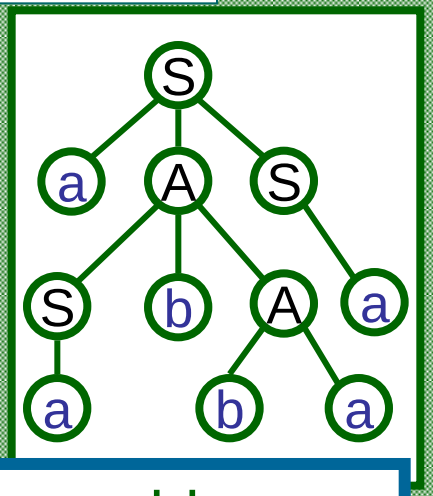
$S \Rightarrow_G^* \alpha$ sss α é o resultado de alguma árvore de derivação de G.

Uma derivação é dita mais à esquerda | direita, se, em cada passo da derivação, uma regra é aplicada à variável mais à esquerda | direita.

Árvores de Derivação em uma GLC

Considere a árvore de derivação em G_1

$P=\{S \rightarrow aAS \mid a,\ A \rightarrow SbA \mid SS \mid ba\}$



Temos então:

Resultado w = aabbaa

A derivação mais à esquerda de w associada a essa árvore:

S =>aAS=>aSbAS=>aabAS=>aabbaS=>aabbaa

A derivação mais à direita de w associada a essa árvore:

S =>aAS=>aAa=>aSbAa=>aSbbaa=>aabbaa

Árvores de Derivação em uma GLC

Outros aspectos

Se w Є L (G) para alguma GLC G, então w é o resultado de pelo menos uma árvore de derivação particular de G.

Associadas a uma árvore de derivação particular,

W tem

uma única derivação mais à esquerda e
uma única derivação mais à direita.

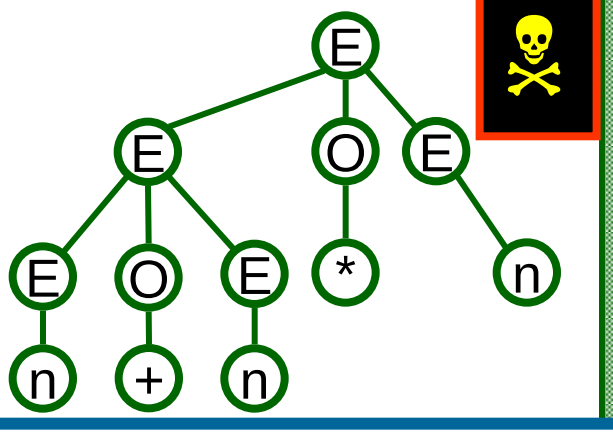
Uma gramática livre de contexto G
é **ambígua** sss existe w Є L (G) tal que w é **resultado** de mais de uma árvore de derivação.

gramática ambígua

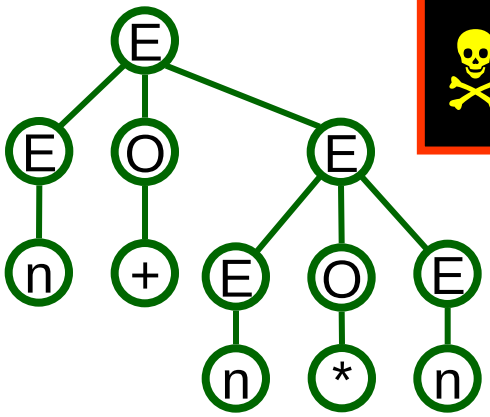
Árvores de Derivação em uma GLC

Exemplo de gramática ambígua

G_1=<{E,O},{(,),n,+,-,*,/},
{E→EOE|(E)|-E|n, O→+|-|*|/},E> é ambígua porque a sentença n+n*n é resultado de duas árvores de derivação em G.



precedência errada



precedência certa

Configuração no autômato: α, y

- α **é o conteúdo da pilha**
- **y é o restante da entrada**
- **topo da pilha é o símbolo à esquerda da pilha deitada**

Transições

M = <{q}, Σ, N ∪ Σ, δ, q, S, ∅ > e **δ** é definida como, a seguir:

- Para cada **ñ terminal A ϵ N**, se as regras de **A** são:

  A → α_1|...|α_n, temos: **δ (q, ε, A) = {(q, α_1), ... , (q, α_n)}**

- Para cada **terminal a ϵ Σ**, temos: **δ (q, a, a) = {(q, ε)}**

1 – expansão para A → β : [A α , y] ├ [β α , y]

2 – terminal a no topo: [a α , ay] ├ [α , y]

Configuração inicial: [S, x] e final: [ε, ε]

Análise sintática descendente para a cadeia (a+a)*a:

E => T=> T*F=> F*F =>(E)*F=>(E+T)*F=>(T+T)*F=> (F+T)*F => (a+T)*F => (a+F)*F => (a+a)*F=>(a+a)*a

Outra forma de acompanhar o reconhecimento de (a+a)*a pelo APND

pilha	fita
E	(a+a)*a
T	(a+a)*a
T*F	(a+a)*a
F*F	(a+a)*a
(E)*F	(a+a)*a
E)*F	a+a)*a
E+T)*F	a+a)*a
T+T)*F	a+a)*a
F+T)*F	a+a)*a
a+T)*F	a+a)*a

pilha	fita
+T)*F	+a)*a
T)*F	a)*a
F)*F	a)*a
a)*F	a)*a
)*F	)*a
*F	*a
F	a
a	a
ε	ε

P = {
E->E+T | T,
T->T*F | F
F->(E) | a
}

δ (q, ε, E)= {(q, E+T), (q,T)}
δ (q, ε, T)= {(q, T*F), (q,F)}
δ (q, ε, F)= {(q, (E)), (q,a)}
δ (q, +, +)= {(q, ε)}
δ (q, *, *)= {(q, ε)}
δ (q, (, ()= {(q, ε)}
δ (q,),))= {(q, ε)}
δ (q, a, a)= {(q, ε)}

Capítulo 3 - Análise sintática ascendente (bottom-up)

Configuração no autômato:

α, y

α é o conteúdo da pilha
y é o restante da entrada
topo da pilha é o símbolo à direita

Transições

$M = <\{q\}, \Sigma, N \cup \Sigma, \delta, q, S, \varnothing>$ e δ é definida por duas ações alternativas:

- **Redução:** reduzindo **ao não terminal esquerdo**, desempilhando os símbolos da pilha que correspondem ao lado direito da regra e empilhando o não terminal do lado esquerdo da regra
- **Empilhamento do terminal (entrada):** empilhando o terminal lido da entrada

1 – redução: $A \rightarrow \beta$, ..., $B \rightarrow \beta$, ..., $Z \rightarrow \beta$

$[\alpha\beta, y] \vdash [\alpha A, y]$ ou $\vdash [\alpha B, y]$ ou ... $\vdash [\alpha Z, y]$

então $\delta(q, \varepsilon, \beta) = \{(q, A), (q, B), ..., (q, Z)\}$ o terminal não é lido

2 – empilhamento do terminal a:

$[\alpha, ay] \vdash [\alpha a, y]$, então $\delta(q, a, \alpha) = \{(q, a)\}$

Configuração inicial: $[\varepsilon, x]$ e **final:** $[S, \varepsilon]$

Análise sintática ascendente para a cadeia (a+a)*a:

P: $E \rightarrow E+T \mid T$,
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid a$

Acompanhar o reconhecimento de (a+a)*a pelo APND no sentido ascendente (bottom-up)

pilha	fita
ε	(a+a)*a$
(	a+a)*a$
(a	+a)*a$
(F	+a)*a$
(T	+a)*a$
(E	+a)*a$
(E+	a)*a$
(E+a	)*a$
(E+F	)*a$
(E+T	)*a$

pilha	fita
(E	)*a$
(E)	*a$
F	*a$
T	*a$
T*	a$
T*a	$
T*F	$
T	$
E	$

1 – redução:

$A \rightarrow \beta$, $B \rightarrow \beta$, ..., $Z \rightarrow \beta$

$[\alpha\beta, y] \vdash [\alpha A, y]$

ou $\vdash [\alpha B, y]$

ou ... $\vdash [\alpha Z, y]$

então

$\delta(q, \varepsilon, \beta) = \{(q, A), (q, B), ..., (q, Z)\}$

o terminal não é lido

2 – empilhamento do terminal a:

$[\alpha, ay] \vdash [\alpha a, y]$,

então $\delta(q, a, \alpha) = \{(q, a)\}$

Configuração inicial: $[\varepsilon, x]$

Configuração final: $[S, \varepsilon]$

Cap. 3 Análise sintática - Coletando informação sobre gramáticas

Informações usadas para **escolher** qual das **regras** com um **não terminal à esquerda, digamos A**, deverá ser **aplicada** ou **reduzida**, durante o processo da análise sintática

Atenção: os algoritmos que coletam essas informações aplicam-se a gramáticas sem regras inúteis

Informação 1. se **A** gera ou não a cadeia vazia **ε**.

Informação 2. quais são os símbolos terminais **iniciadores** das cadeias geradas a partir de **A**:

se **A =>*α**, que terminais podem aparecer como primeiro símbolo de **α**.

Informação 3. quais são os símbolos terminais **seguidores** de A: ou seja, se **S =>*αAβ**, que terminais podem aparecer como primeiro símbolo de **β**.

Informação 1: Geração da cadeia vazia.
Esse algoritmo determina, para todos os **nãoterminais A** de uma **gramática G**, se **A** gera ou não a **cadeia vazia ε**.

Algoritmo 1:
Identificação dos **nãoterminais** que derivam a **cadeia ε.**

O algoritmo trabalha com uma **lista L de regras** e uma **tabela** de não terminais que vão sendo alteradas durante a execução.

Atenção! Se a **gramática não tem** regras e/ou símbolos inúteis, o algoritmo termina com todos os nãoterminais da tabela marcados com:

sim (nãoterminal **gera ε**)

ou

não (nãoterminal **não gera ε**).

Algoritmo 1: Identificação dos **nãoterminais** que derivam a **cadeia ε.**

0. Inicialmente a lista L contém todas as regras de G, exceto aquelas que contém um símbolo terminal. (Regras cujo lado direito têm um terminal não servem para derivar **ε**.)

1. Se existe um **nãoterminal A** sem regras e **A** não está marcado como sim, então marque **A** com **não**. (não interessa se **A** originalmente tinha regras, ou se todas as regras foram retiradas de **L**)

2. Se um **nãoterminal A** tem uma **regra A→ε**,

retire de **L** todas as **regras** com **A** do **lado esquerdo**, e
retire todas as **ocorrências de A** do **lado direito das regras de L**.
Marque **A** com **sim**.

(Note que uma regra **B → C** se transforma em **B →ε**,

se a ocorrência de **C** do lado direito for retirada.)

3. Se uma **regra** tem do **lado direito** um **nãoterminal marcado não**, retire a **regra**. (Regras cujo lado direito têm um nãoterminal que não deriva

ε não servem para derivar **ε**.)

4. Repita os passos 1, 2, 3 até que nenhuma nova informação seja adicionada

Aplicando o algoritmo 1 na gramática

Lista L de regras

1. $P \to PABCD$
2. ~~$A \to \varepsilon$~~
3. $A \to aA$
4. ~~$B \to \varepsilon$~~
5. $B \to Bb$
6. $C \to c$
7. ~~$C \to AB$~~
8. $D \to d$

Tab. Sim/Não

P	Não
A	Sim
B	Sim
C	Sim
D	Não

Passo 0.
Preenchimento da lista L.
As regras 3, 5, 6, e 8 não pertencem a L.

GRAMÁTICA

1. $P \to ABCD$
2. $A \to \varepsilon$
3. $A \to aA$
4. $B \to \varepsilon$
5. $B \to Bb$
6. $C \to c$
7. $C \to AB$
8. $D \to d$

Passo 1: D é um não terminal sem regras na lista L. Então, na tabela é marcado como Não.

Passo 2: Não terminais que levam a ε: A e B
Então, regras retiradas de L: 2 e 4
Regra 7: Símbolos A e B riscados
então, C leva a ε e deve ser retirada de L.
Na tabela: A, B e C são marcados sim.

Passo 3: Neste ponto, sobrou apenas a regra 1.
No lado esquerdo há D que é um ñ terminal Não
então P é retirada da lista L.

Novo loop:
passo 1:
P é um ñ terminal sem regras na lista L.
Então, na tabela, P é marcado como Não
passo 2: não se aplica mais.
passo 3: não se aplica mais

Cálculo dos iniciadores. *Formalmente, podemos definir os símbolos* **iniciadores** *de um* **nãoterminal A** *através de um conjunto* **First(A)**

First(A) = {**a** |**a** é um terminal e **A =>*aα**, para alguma cadeia **α** qualquer}.

O algoritmo vai calcular os conjuntos First(A) para todos os nãoterminais A de uma gramática G.

Cap. 3 Análise sintática - Coletando informação sobre gramáticas

First(A) = {**a** |**a** é um terminal e **A =>*aα**, para alguma cadeia **α** qualquer}.

O algoritmo se baseia nos seguintes pontos:

· Se há uma regra **A** → **aα**, então **a** ∈ **First(A)**.
A derivação correspondente é **A =>aα**.

· Se há uma regra **A**→**B_1...B_maα**, e para todo **i=1, ..., m**, **B_i =>* ε**, então também temos **a ∈ First(A).**
Neste caso, **a** não é o primeiro símbolo, mas passa a ser quando todos os **B_i** "desaparecerem", isto é, forem substituídos por **ε**.
A derivação correspondente é **A => B_1...B_m aα =>* aα**.

· Se há uma regra **A**→**Bα**, e se **a ∈First (B)**, temos também **a∈First (A)**.
Se **a ∈First (B)**, temos **B =>* a β**, e
a derivação correspondente é **A=>Bα=>*a β α**.

· Se há uma regra **A**→**B_1...B_mBα**, e para todo **i=1, ..., m**, **B_i=>*ε**,
Então se **a ∈First (B)**, temos também **a∈First (A),**
de forma semelhante, já que os **B_i** "desaparecem".
Se tivermos **B =>* a β**, a derivação correspondente é
A=>B_1...B_mBα=>Bα =>* aβα.

Cap. 3 Análise sintática - Coletando informação sobre gramáticas

First(A) = {**a** |**a** é um terminal e **A =>*aα**, para alguma cadeia **α** qualquer}.

Algoritmo 2: Cálculo de First(A), para todos os ñ-terminais A de uma gramática G.

0. Inicialmente, para todos os **nãoterminais A da gramática G**, todos os conjuntos **First(A)** estão **vazios**.

1. Para cada regra $A \rightarrow B_1 \ldots B_m a \alpha$, tal que para todo **i=1, ..., m**, B_i **=>*** **ε**, acrescente **a** a **First(A)**.

2. Para cada regra $A \rightarrow B_1 \ldots B_m B \alpha$, tal que, para todo **i=1, ..., m**, B_i **=>*** **ε**, acrescente **First(B)** a **First(A)**.

3. Repita o **passo 2** enquanto
houver alteração em algum dos **conjuntos First**.

Aplicando o algoritmo 2 na gramática:

passo 1: aplica-se às regras 3, 5, 6, 8

regra 3: $a \in First(A)$

regra 5: $b \in First(B)$

regra 6: $c \in First(C)$

regra 8: $d \in First(D)$

passo 2: aplica-se as regras 1 e 7

regra 1: A B C com ε, então first (P) recebe first (D)

A B com ε, então first (P) recebe first (C)

A com ε, então first (P) recebe first (B)

A não é ε, então first (P) recebe first (A)

regra 7: A com ε, então first (C) recebe first (B)

A não é ε, então first (C) recebe first (A)

passo 2: verificando se algum conjunto first será alterado

Está encerrado o algoritmo 2.

Gramática

1. $P \to ABCD$
2. $A \to \varepsilon$
3. $A \to aA$
4. $B \to \varepsilon$
5. $B \to Bb$
6. $C \to c$
7. $C \to AB$
8. $D \to d$

First (passo 0)

P	d, c, b, a
A	a
B	b
C	c, b, a
D	d

Cálculo dos seguidores.

Follow(A) = {**a** | **a** é um **terminal** e **S$ =>*αAaβ**, para **cadeias α**,**β** **qsqr**}.

Considerações iniciais: o que é $?

Um **ñ-terminal** pode aparecer no extremo direito de uma derivação.
Nesta situação, **não há símbolo seguidor**.

Como contornar ? Acrescentando **um novo símbolo terminal "$", que indica o final da cadeia.**

Do ponto de vista de implementação:
$ é considerado como o **token marca de fim-de-arquivo** do arquivo fonte.
Pode ser também entregue pelo analisador léxico.

No tratamento do algoritmo: **$ deve ser incluído em Follow (S).**
$ será encontrado como **seguidor de qualquer** símbolo **ñ-terminal**, que **apareça no fim de uma cadeia** gerada a partir de **S**.

Em particular, sempre teremos **$** como **seguidor de S**.

Formalmente, podemos definir os **símbolos seguidores** de um **ñ-terminal A** através de um **conjunto Follow(A).**

Follow(A) = {**a** | **a** é um **terminal** e **S$ =>*αAaβ**, para **cadeias α,β quaisquer**}.

Vamos analisar o **algoritmo** que calcula os **conjuntos Follow(A)** para todos os **ñ-terminais A** de uma gramática **G**.

No que se segue,
$\gamma = B_1 \ldots B_m$ é uma cadeia de **ñ-terminais** que derivam **ε**, portanto, **γ =>* ε**.
Suporemos também que **S$ =>*δAφ**.

Cap 3 - Coletando informação sobre gramáticas - *Cálculo dos seguidores*

Follow(A) = {**a** | **a** é um **terminal** e **S$ =>*αAaβ**, para **cadeias α**,**β qsqr**}.

O algoritmo se baseia nos seguintes pontos:

Como **S$ =>* S$** , **$ Є Follow(S).**

Se há uma regra **A →αBγaβ, então a Є Follow(B).**
A derivação correspondente é
S$ =>*δAφ=>*δαBγaβφ =>*δαBaβφ

Se há uma regra **A →αBγCβ, então se a Є First(C),**
temos também a Є Follow(B).
Se a Є First(C), temos C =>*aμ.
A derivação correspondente é então
S$ =>*δAφ=>*δαBγCβφ =>*δαBCβφ =>*δαB aμ βφ

Se há uma regra **A →αBγ, então se a Є Follow(A),**
temos também a Є Follow(B).
Se a Є Follow(A), temos S$ =>*δAaφ .
A derivação correspondente é
S$ =>*δAaφ=>*δαBγaφ =>*δαBaφ .

Follow(A) = {**a** | **a** é um **terminal** e **S$ =>*αAaβ**, para **cadeias α**,**β** **qsqr**}.

Algoritmo 3: Cálculo de **Follow(A)** para todos os **ñ-terminais A** de uma gramática **G**.

0. Inicialmente, para todos os **ñterminais A** da gramática G, todos os conjuntos Follow(**A**) estão vazios, excetuando-se Follow(**S**)= { **$** }.

1. Se há uma regra **A → αBγaβ** e **γ = B_1 ... B_m =>* ε**, **então acrescente a a Follow(B).**

2. Se há uma regra **A → αBγCβ** e **γ = B_1 ... B_m =>* ε**, **então acrescente First(C) a Follow(B).**

3. Se há uma regra **A → αBγ** e **γ = B_1 ... B_m =>* ε**, **então acrescente Follow(A) a Follow(B).**

4. Repita o passo 3 enquanto houver modificação em algum dos conjuntos.

Aplicando o algoritmo 3 à gramática

passo 1: aplica-se a regra 5, $B \to Bb$, B é seguido por b.

passo 2: aplica-se às regras 1 e 7

regra 1: $P \to A\,B\,C\,D$

- follow (C) recebe first (D)
- se C é empty, follow (B) recebe first (D)
  se C não é empty, follow (B) recebe first (C)
- se B e C empty, follow (A) recebe first (D)
  se B empty, follow (A) recebe first (C)
  se B não empty, follow (A) recebe first (B)

regra 7: $C \to A\,B$

- follow (A) recebe first (B)

passo 3: aplica-se as regras 1, 3 e 7, sendo que 3 vai ser redundante.

regra 1: follow (D) recebe follow (P)

regra 7: $C \to A\,B$ nesta regra B pode estar Empty ou não.

- follow (B) recebe follow (C)
- se B está empty, follow (A) recebe follow (C)

Gramática

1. $P \to ABCD$
2. $A \to \varepsilon$
3. $A \to aA$
4. $B \to \varepsilon$
5. $B \to Bb$
6. $C \to c$
7. $C \to AB$
8. $D \to d$

Follow (passo ∅)

P	$
A	d, c, b, a
B	b, d, c, a
C	d
D	$

Cap 3 - Coletando informação sobre gramáticas – *Cálculo dos iniciadores de cadeia*

Um conceito que também é necessário é o conceito de conjunto de símbolos iniciadores de uma cadeia qualquer **α**, ou seja, First(**α**).
A definição é feita recursivamente:

se **α** = **ε**, First(**α**)=First(**ε**)=**Ø**

se **α** é um terminal **a**, First(**α**)=First(**a**)= {**a**};

se **α** é um ñterminal **A**, First(**A**) é calculado pelo **Algoritmo 2**;

se **α** é uma cadeia **Aβ**, e o primeiro símbolo é um ñterminal **A**, que deriva **ε**, First(**α**)=First(**Aβ**)=First(**A**) U First(**β**)

se **α** é uma cadeia **Aβ**, e o primeiro símbolo é um ñterminal **A**, que **não** deriva **ε**, First(**α**)=First(**Aβ**)=First(**A**)

se **α** é uma cadeia **aβ**, cujo primeiro símbolo é um terminal **a**, First(**α**)=First(**aβ**)= { **a** }

Cap 3 - Análise sintática LL(1)

A **idéia** é utilizar **duas informações**.

Resposta à questão da **escolha da regra**
a ser usada no **processo de análise descendente**.

- o **ñterminal A** a ser expandido e
- o **primeiro símbolo a** do resto da entrada.

Uma **tabela M** com essas duas entradas, como **índices**,
dá a regra a ser utilizada: **M**[**A**, **a**].

Essa técnica só pode ser usada para uma classe restrita de gramáticas, a **classe das gramáticas LL(1).**

Cap 3 - Análise sintática LL(1)

O nome **LL(1)** indica que:

(L=*left-to-right*):
a **cadeia de entrada** é examinada da **esquerda para a direita**

(L=*leftmost*):
o **analisador** procura **construir** uma **derivação esquerda**

1:
exatamente **1** símbolo do **resto da entrada** é examinado.

Cap 3 - Análise sintática LL(1)

Nossa conhecida gramática de expressão **não é LL(1)**, ela está na **classe das sLR(1),** também denominada **LR(0).** Essa restrição se dá porque ela apresenta recursão à esquerda, nas regras **E → E + T e T → T * F,** causando **indeterminismo no LL(1).**

Assim, ela foi transformada na **gramática LL(1)** ao lado, para o êxito no funcionamento do LL(1), construído para a gramática de expressão.

1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → a

⇒

1. E → T E'
2. T → F T'
3. F → (E)
4. F → a
5. E' → + T E'
6. E' → ε
7. T' → * F T'
8. T' → ε

Cap 3 - Análise sintática LL(1)

Como **construir M**, a **tabela** de análise **LL(1)**

Caso mais simples:

- símbolo **a** (primeiro da entrada restante) e
- **a** o **primeiro símbolo** derivado do **ñterminal** **A** a ser expandido e ∈ **First(A)**.

Neste caso, **a** deve pertencer a **First(α)**, onde **A** → **α** é uma das alternativas de regra para **A**.

No exemplo da gramática de expressão, a regra **F → (E)** foi usada com **(**.

First(A)	Follow(A)	ñterminais ε
E: (,a	E: $,)	E: não
T:(,a	T:$,),+	T: não
F:(,a	F:$,),+,*	F: não
E': +	E': $,)	E': sim
T': *	T': $,),+	T': sim

Regras

1. E → T E'
2. T → F T'
3. **F → (E)**
4. F → a
5. E' → + T E'
6. E' → **ε**
7. T' → * F T'
8. T' → **ε**

First ou Follow(α)

First(TE')={(,a}
First(FT')={(,a}
First((E))={(}
First(a)={a}
First(+TE')={+}
Follow(E')={$,)}
First(*FT')={*}
Follow(T')={$,+,)}

Cap 3 - Análise sintática LL(1)

Como **construir M**, a **tabela** de análise **LL(1)**

Outra possibilidade

A não é o responsável pela geração do simb. **a**, e sim outro ñterminal encontrado depois de **A**.

Neste caso, devemos ter o terminal **a** ∈ **Follow(A)**.

No exemplo da gramática de expressão, a regra **T' → ε** foi usada com **+**.

First(A)	Follow(A)	ñterminais ε
E: (,a	E: $,)	E: não
T:(,a	T:$,),+	T: não
F:(,a	F:$,),+,*	F: não
E': +	E': $,)	E': sim
T': *	T': $,),+	T': sim

Regras
1. E → T E'
2. T → F T'
3. F → (E)
4. F → a
5. E' → + T E'
6. E' → ε
7. T' → * F T'
8. **T' → ε**

First ou Follow(α)
First(TE')={(,a}
First(FT')={(,a}
First((E))={(}
First(a)={a}
First(+TE')={+}
Follow(E')={$,)}
First(*FT')={*}
Follow(T')={$,+,)}

Cap 3 - Análise sintática LL(1)

Como **construir M**, a **tabela** de análise **LL(1)**

Devemos **examinar** todas as **regras da gramática**:

Para cada **regra i**: **A** → **α**, temos **M**[**A**, **a**]=**i**, para cada **a** em **First**(**α**).

Para cada **regra i**: **A** → **α**, se **α** **=>*** **ε,** temos **M**[**A**, **a**]=**i**, para cada **a** em **Follow**(**A**).

First(**A**)	Follow(**A**)	ñterminais **ε**
E: (,a	E: $,)	E: não
T:(,a	T:$,),+	T: não
F:(,a	F:$,),+,*	F: não
E': +	E': $,)	E': sim
T': *	T': $,),+	T': sim

Regras	First ou Follow(α)
1. E → T E'	First(TE')={(,a}
2. T → F T'	First(FT')={(,a}
3. F → (E)	First((E))={(}
4. F → a	First(a)={a}
5. E' → + T E'	First(+TE')={+}
6. E' → ε	Follow(E')={$,)}
7. T' → * F T'	First(*FT')={*}
8. T' → ε	Follow(T')={$,+,)}

Cap 3 - Análise sintática LL(1)

Como **construir M**, a **tabela** de análise **LL(1)**

Devemos **examinar** todas as **regras da gramática**:

Para cada **regra i**: **A** → **α**, temos **M**[**A**, **a**]=**i**, para cada **a** em **First**(**α**).

Para cada **regra i**: **A** → **α**, se **α** **=>*** **ε,** temos **M**[**A**, **a**]=**i**, para cada **a** em **Follow**(**A**).

	(	a	+	*	)	$
E	1	1	-	-	-	-
T	2	2	-	-	-	-
F	3	4	-	-	-	-
E'	-	-	5	-	6	6
T'	-	-	8	7	8	8

Não tem conflito na tabela, então, é LL(1) !

First(**A**)

E: (,a
T:(,a
F:(,a
E': +
T': *

Follow(**A**)

E: $,)
T:$,),+
F:$,),+,*
E': $,)
T': $,),+

ñterminais **ε**

E: não
T: não
F: não
E': sim
T': sim

Regras

1. E → T E'
2. T → F T'
3. F → (E)
4. F → a
5. E' → + T E'
6. E' → ε
7. T' → * F T'
8. T' → ε

First ou Follow(α)

First(TE')={(,a}
First(FT')={(,a}
First((E))={(}
First(a)={a}
First(+TE')={+}
Follow(E')={$,)}
First(*FT')={*}
Follow(T')={$,+,)}

Cap 3 - Análise sintática LL(1)

Como **construir M**, a **tabela** de análise **LL(1)**

Devemos **examinar** todas as **regras da gramática**:

Para cada **regra i**: **A** → **α**, temos **M**[**A**, **a**]=**i**, para cada **a** em **First**(**α**).

Para cada **regra i**: **A** → **α**, se **α** **=>*** **ε**, temos **M**[**A**, **a**]=**i**, para cada **a** em **Follow**(**A**).

Se a gramática é LL(1), cada entrada de M receberá no máximo um valor.

As entradas de M que não receberem nenhum valor devem ser marcadas como entradas de erro.

Se todas as entradas de M foram definidas uma única vez, dizemos que a gramática é LL(1), como no caso acima.

	(	a	+	*	)	$
E	1	1	-	-	-	-
T	2	2	-	-	-	-
F	3	4	-	-	-	-
E'	-	-	5	-	6	6
T'	-	-	8	7	8	8

Regras

1. E → T E'
2. T → F T'
3. F → (E)
4. F → a
5. E' → + T E'
6. E' → ε
7. T' → * F T'
8. T' → ε

First ou Follow(α)

First(TE')={(,a}
First(FT')={(,a}
First((E))={(}
First(a)={a}
First(+TE')={+}
Follow(E')={$,)}
First(*FT')={*}
Follow(T')={$,+,)}

Análise da expressão: a + a * a

Regras

1. $E \rightarrow T\ E'$
2. $T \rightarrow F\ T'$
3. $F \rightarrow (\ E\ )$
4. $F \rightarrow a$
5. $E' \rightarrow +\ T\ E'$
6. $E' \rightarrow \varepsilon$
7. $T' \rightarrow *\ F\ T'$
8. $T' \rightarrow \varepsilon$

Tabela LL(1)

	(	a	+	*	)	$
E	1	1	-	-	-	-
T	2	2	-	-	-	-
F	3	4	-	-	-	-
E’	-	-	5	-	6	6
T’	-	-	8	7	8	8

$M = <\{q\}, \Sigma, N \cup \Sigma, \delta, q, S, \varnothing>$ e
δ é definida como, a seguir:

Para cada ñ terminal $A \in N$, se as regras de A são: $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$, temos:
$\delta(q, \varepsilon, A) = \{(q, \alpha_1), (q, \alpha_2), \ldots, (q, \alpha_n)\}$

Para cada terminal $a \in \Sigma$, temos: $\delta(q, a, a) = \{(q, \varepsilon)\}$

	Pilha	Entrada	escolha da regra
	E	a+a*a	M[E, a] = 1
	TE'	a+a*a	M[T, a] = 2
	FT'E'	a+a*a	M[F, a] = 4
	aT'E'	a+a*a	–
	T'E'	+a*a	M[T', +] = 8
	E'	+a*a	M[E', +] = 5
	+TE'	+a*a	–
	TE'	a*a	M[T, a] = 2
	FT'E'	a*a	M[F, a] = 4
	aT'E'	a*a	–
	T'E'	*a	M[T', *] = 7
	*FT'E'	*a	–
	FT'E'	a	M[F, a] = 4
	aT'E'	a	–
	T'E'	ε	M[T', $] = 8
	E'	ε	M[E', $] = 6
	ε	ε	–

Cap 3 - Análise sintática LL(1)

Gramática não é LL(1)

Devemos **examinar** todas as **regras da gramática**:

Para cada **regra i**: **A** → **α**, temos **M**[**A**, **a**]=**i**, para cada **a** em **First**(**α**).

Para cada **regra i**: **A** → **α**, se **α** **=>*** **ε**, temos **M**[**A**, **a**]=**i**, para cada **a** em **Follow**(**A**).

Se a gramática é LL(1), cada entrada de M receberá no máximo um valor.

As entradas de M que não receberem nenhum valor devem ser marcadas como entradas de erro.

	(	a	+	*	)	$
E	1,2	1,2	-	-	-	-
T	3,4	3,4	-	-	-	-
F	5	6	-	-	-	-

First(**A**)

E: (,a
T:(,a
F:(,a

Follow(**A**)

E: $,),+
T:$,),*,+
F:$,),+,*

ñterminais **ε**

E: não
T: não
F: não

Regras

1. E → E+T
2. E → T
3. T → T*F
4. T → F
5. F → (E)
6. F → a

First(**α**)

First(E+T)={(,a}
First(T)={(,a}
First(T*F)={(,a}
First(F)={(,a}
First((E))={(}
First(a)={a}

Cap 3 - Análise sintática LL(1)

Gramática não é LL(1)

Devemos **examinar** todas as **regras da gramática**:

Para cada **regra i**: **A** → **α**, temos **M**[**A**, **a**]=**i**, para cada **a** em **First**(**α**).

Para cada **regra i**: **A** → **α**, se **α** **=>*** **ε**, temos **M**[**A**, **a**]=**i**, para cada **a** em **Follow**(**A**).

Se a gramática é LL(1), cada entrada de M receberá no máximo um valor.

As entradas de M que não receberem nenhum valor devem ser marcadas como entradas de erro.

	(	a	+	*	)	$
E	**1,2**	**1,2**	-	-	-	-
T	**3,4**	**3,4**	-	-	-	-
F	5	6	-	-	-	-

Regras	First(**α**)
1. E → E+T	First(E+T)={(,a}
2. E → T	First(T)={(,a}
3. T → T*F	First(T*F)={(,a}
4. T → F	First(F)={(,a}
5. F → (E)	First((E))={(}
6. F → a	First(a)={a}

Caso alguma entrada de M seja definida mais de uma vez, dizemos que houve um conflito, e que a gramática não é LL(1), como neste caso acima.

Gramática não é LL(1)

Se alguma entrada de M é definida **mais de uma vez**, dizemos que houve um **conflito**, e que a **gramática não é LL(1),** como neste caso à direita.

	(	a	+	*	)	$
E	**1,2**	**1,2**	-	-	-	-
T	**3,4**	**3,4**	-	-	-	-
F	5	6	-	-	-	-

No caso dessa gramática, foi concluído por inspeção que ela **não é LL(1)**.

Regras	First(α)
1. **E** → **E**+T	First(E+T)={(,a}
2. E → T	First(T)={(,a}
3. **T** → **T***F	First(T*F)={(,a}
4. T → F	First(F)={(,a}
5. F → (E)	First((E))={(}
6. F → a	First(a)={a}

recursão à esquerda em 1 e 3

As duas características mais óbvias são a **recursão à esquerda** e a **possibilidade de fatoração**.

Derivação à esquerda para a cadeia (a+a)*a:

E => T=> T*F=> F*F =>(E)*F=>(E+T)*F=>(T+T)*F=> (F+T)*F => (a+T)*F => (a+F)*F => (a+a)*F=>(a+a)*a

Outra forma de acompanhar o reconhecimento de (a+a)*a pelo APND

Regras

1. E → E+T
2. E → T
3. T → T*F
4. T → F
5. F → (E)
6. F → a

pilha	fita
E ? 1 ? 2	(a+a)*a
T ? 3 ? 4	(a+a)*a
T*F ? 3 ? 4	(a+a)*a
F*F	(a+a)*a
(E)*F	(a+a)*a
E)*F ?1?2	a+a)*a
E+T)*F?1,2	a+a)*a
T+T)*F?3,4	a+a)*a
F+T)*F	a+a)*a
a+T)*F	a+a)*a

pilha	fita
+T)*F	+a)*a
T)*F ?3?4	a)*a
F)*F	a)*a
a)*F	a)*a
)*F	)*a
*F	*a
F	a
a	a
ε	ε

δ (q, ε, E)= **{(q, E+T), (q,T)}**
δ (q, ε, T)= **{(q, T*F), (q,F)}**
δ (q, ε, F)= {(q, (E)), (q,a)}
δ (q, +, +)= {(q, ε)}
δ (q, *, *)= {(q, ε)}
δ (q, (, ()= {(q, ε)}
δ (q,),))= {(q, ε)}
δ (q, a, a)= {(q, ε)}

	(	a	+	*	)	$
E	1,2	1,2	-	-	-	-
T	3,4	3,4	-	-	-	-
F	5	6	-	-	-	-

Recursão à esquerda

Na gramática à direita, encontramos recursão à esquerda nas regras 1 e 3.

Se uma gramática permite uma derivação A =>* A α para algum nãoterminal A e para alguma cadeia não vazia α, a gramática é dita recursiva à esquerda.

No caso mais simples, que vamos examinar aqui, existe na gramática uma regra A → A α , responsável diretamente pela derivação mencionada.

1. E → E+T
3. T → T*F

Regras	First(α)
1. E → E+T	First(E+T)={(,a}
2. E → T	First(T)={(,a}
3. T → T*F	First(T*F)={(,a}
4. T → F	First(F)={(,a}
5. F → (E)	First((E))={(}
6. F → a	First(a)={a}

	(	a	+	*	)	$
E	1,2	1,2	-	-	-	-
T	3,4	3,4	-	-	-	-
F	5	6	-	-	-	-

Recursão à esquerda

Na gramática à direita, encontramos recursão à esquerda nas regras 1 e 3.

1. E → E+T
2. E → T
3. T → T*F
4. T → F
5. F → (E)
6. F → a

	(	a	+	*	)	$
E	1,2	1,2	-	-	-	-
T	3,4	3,4	-	-	-	-
F	5	6	-	-	-	-

Se uma gramática permite uma derivação A **=>*** A **α**
para algum nãoterminal A e para alguma cadeia não vazia **α**, a gramática é dita recursiva à esquerda.

Se A não é um nãoterminal inútil, deve existir (pelo menos) uma regra da forma **A** → **β**, sem recursão à esquerda (aquela que aterra a recursão)

No caso mais simples, que vamos examinar aqui, existe na gramática uma regra A → A **α** , responsável diretamente pela derivação mencionada.

A→ A α
A → β

A combinação dessas duas regras faz com que First(A) e, portanto, First(A **α**) contenham todos os símbolos de First(**β**), levando necessariamente a um **conflito**.

1. E → E+T
3. T → T*F

3. T → T*F
4. T → F

Cap 3 - Análise sintática LL(1)

Recursão à esquerda

A **eliminação da recursão** à esquerda é tentada, procurando transformar a gramática em uma gramática LL(1).

As regras

A → A α

A → β

permitem a geração de cadeias da forma **β α α α ...,**

essas mesmas cadeias podem ser geradas pelas regras abaixo.

A → β A'

A' → α A'

A ' → ε

Esta outra forma de geração usa **recursão à direita**, que não cria problemas.

A → A α
A → β

⇒

A → β A'
A' → α A'
A ' → ε

E → E+T
E → T

⇒

E → T E'
E' → + T E'
E' → ε

T → T*F
T → F

⇒

T → F T'
T' → * F T'
T' → ε

1. E → E+T
2. E → T
3. T → T*F
4. T → F
5. F → (E)
6. F → a

⇒

1. E → T E'
2. T → F T'
3. F → (E)
4. F → a
5. E' → + T E'
6. E' → ε
7. T' → * F T'
8. T' → ε

Possibilidade de fatoração

Outra combinação que ocorre é a de duas regras que começam pelo mesmo símbolo:

Regras como

$A \rightarrow \alpha\ \beta$
$A \rightarrow \alpha\ \gamma$

com **First(α)≠Ø**.

Neste caso, existe uma **interseção** entre **First(α β)** e **First(α γ),**
e a **gramática não pode ser LL(1).**

Isto acontece porque não é possível decidir, **olhando apenas o primeiro símbolo derivado de α**, **qual a regra que deve ser aplicada**.

A **solução** a ser tentada é **simples**, e **envolve a fatoração**:
*a **cadeia inicial** α é **posta em evidência**.*

Lembrando fatoração:
$\alpha\ \beta \mid \alpha\ \gamma$
$\alpha(\beta \mid \gamma)$ **em que α fica em evidência**

$A \rightarrow \alpha\ \beta \mid \alpha\ \gamma$

$A \rightarrow \alpha\ A'$
$A' \rightarrow \beta \mid \gamma$

Isto implica em adiar a decisão entre **β** e **γ**
para quando o primeiro símbolo derivado de **β** ou de **γ** estiver visível.

Cap 3 - *Recursão à esquerda e Possibilidade de fatoração*

As regras:

$A \rightarrow A\ \alpha$
$A \rightarrow \beta$

permitem a geração de cadeias da forma $\beta\ \alpha\ \alpha\ \alpha$ ...,

Eliminando a recursão à esquerda:

$A \rightarrow \beta\ A'$
$A' \rightarrow \alpha\ A'$
$A' \rightarrow \varepsilon$

Esta outra forma de geração usa **recursão à direita**, que não cria problemas.

duas ou mais regras que começam pelos mesmos símbolos:

Regras como

$A \rightarrow \alpha\ \beta$
$A \rightarrow \alpha\ \gamma$

com **First(α)≠Ø**.

A **solução** a ser tentada é **simples**, e **envolve a fatoração**: *a **cadeia inicial** α é **posta em evidência**.*

$A \rightarrow \alpha\ A'$
$A' \rightarrow \beta \mid \gamma$

***Cuidado**: estas duas técnicas permitem transformar algumas gramáticas em LL(1). Entretanto, devemos observar que algumas GLCs não têm gramáticas equivalentes LL(1), e nesse caso a aplicação dessas (ou de outras) técnicas não terá êxito.*

Cap 3 - Análise Sintática Descida Recursiva

A técnica de descida recursiva *é* uma forma de análise descendente ***em que cada símbolo nãoterminal se transforma em um* procedimento**.

Uma **chamada** de um procedimento **A** serve para encontrar a **maior cadeia que pode ser derivada do nãoterminal A**,
a partir do ponto inicial de análise.

Se a gramática considerada é **LL(1)**, a construção dos procedimentos é **automática**.

Quando a tabela **LL(1)** prevê o uso da regra **A → X1X2...Xn**,
o procedimento **A** faz uma série de chamadas correspondentes a
X1, X2, ...Xn, por uma das duas formas de ação:

se $\mathbf{X_i}$ é um **nãoterminal** - o **procedimento** $\mathbf{X_i}$ é chamado;

se $\mathbf{X_i}$ é um **terminal** - **check(Xi)** é chamada, executando os 2 passos:
primeiro: exigindo que o terminal correntemente lido seja $\mathbf{X_i}$, e
segundo: acionando o **analisador léxico** para obter o próximo símbolo.

O processo de construção é demonstrado a seguir.

Cap 3 - Análise Sintática Descida Recursiva

Construção de **um analisador de descida recursiva** para a **gramática** da direita

Há três procedimentos disponíveis:

scan (analisador léxico) ,
que, a **cada chamada**, **atualiza a variável global** **`simb`**, **que é o próximo símbolo do tipo simbs**.

erro,
deve ser implementado para encaminhar as ações apropriadas a cada situação.

check será substituída por **scan**, quando não for necessário testar o símbolo.

```
type simbs=(abre,a,mais,vezes,fecha,dolar);
simb: simbs;
procedure check(s:simbs);
begin if simb = s then scan else erro end;
```

Regras

1. $E \rightarrow T\ E'$
2. $T \rightarrow F\ T'$
3. $F \rightarrow (\ E\ )$
4. $F \rightarrow a$
5. $E' \rightarrow +\ T\ E'$
6. $E' \rightarrow \varepsilon$
7. $T' \rightarrow *\ F\ T'$
8. $T' \rightarrow \varepsilon$

Tabela LL(1)

	(	a	+	*	)	$
E	1	1	-	-	-	-
T	2	2	-	-	-	-
F	3	4	-	-	-	-
E’	-	-	5	-	6	6
T’	-	-	8	7	8	8

A seguir,
a implementação
para esta gramática

Cap 3 - Análise Sintática Descida Recursiva-continua

```
type simbs=(abre,a,mais,vezes,fecha,dolar);
simb: simbs;
procedure check(s:simbs);
begin if simb = s then scan else erro end;
```

```
procedure E;
begin
case simb of
abre,a: {E → TE' }
begin T; Elinha; end;
mais,vezes,fecha,dolar:
erro;
end;{de case}
end;{de E}
```

```
procedure T;
begin
case simb of
abre,a: {T → FT' }
 begin F; Tlinha; end;
mais,vezes,fecha,dolar:
 erro;
end; {de case}
end; {de T}
```

1. E → T E'	5. E' → + T E'
2. T → F T'	6. E' → ε
3. F → (E)	7. T' → * F T'
4. F → a	8. T' → ε

```
procedure F;
begin
case simb of
abre: {F →(E) }
 begin
 scan;E;check(fecha);
 end;
a: {F → a}
 scan;
mais,vezes,fecha,dolar:
 erro;
end; {de case}
end; {de F}
```

```
procedure Elinha;
begin
case simb of
mais: {E' → +TE' }
 begin
   scan; T; Elinha;
 end;
fecha,dolar: ;{E'→ε}
abre,a,vezes: erro;
end; {de case}
end; {de Elinha}
```

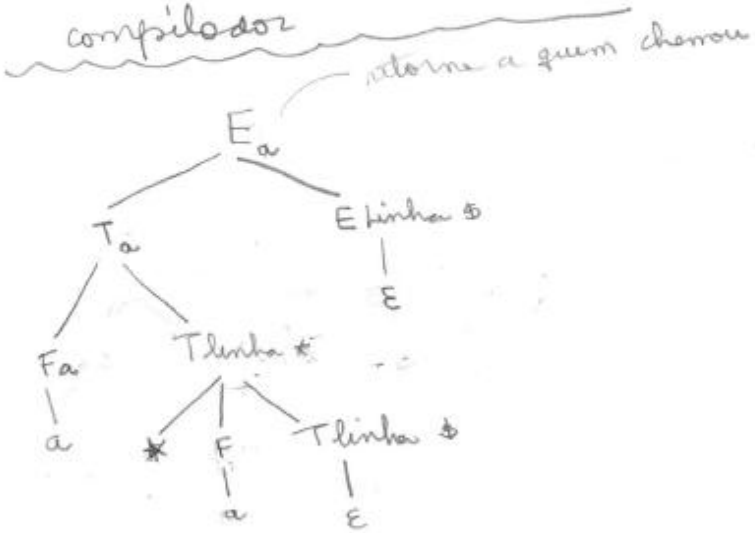
```
procedure Tlinha;
begin
case simb of
vezes: {T' → *FT'}
 begin
  scan; F; Tlinha;
 end;
mais,fecha,dolar:{T'→ε}
 ;
abre,a: erro;
end; {de case}
end; {de Tlinha}
```

$\boxed{a * a}$

começa com procedimento E
com variável simb = a
entra no case abre, a:
chama T;

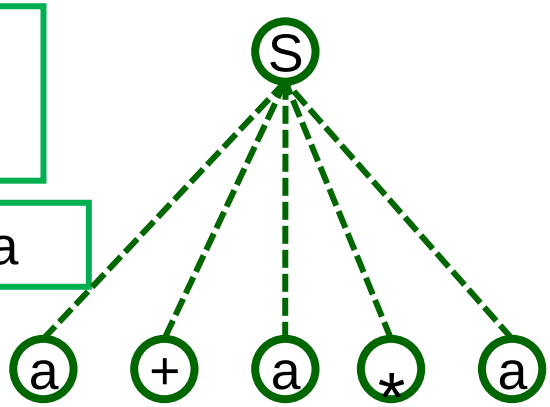
no procedimento T:
simb ainda está com a
entra no case abre, a:
chama procedimento F:

no procedimento F;
simb ainda está com a
entra no case a:
já casou a de simb com o do case
chama scan que vai guarda vezes em simb
e vai retorna a T
em T vai chama Tlinha
no procedimento Tlinha
vai entrar no case vezes e casa com o simb
chama scan que vai guardar a em simb
chama F
no procedimento F:
entra no case a:
já casou a de simb com o do case
chama scan que vai guardar dolar
chama Tlinha
vai entrar no case mais, fecha, dolar
e retorna a chamada anterior de Tlinha
que retorna a T
que retorna a E
que chama E linha
entra no case fecha, dolar:
que retorna a E
que retorna a quem chamou o
compilador



Cap 3 - Análise Sintática sLR(1) também chamada LR(0)

Análise ascendente:
árvore de derivação de baixo para cima



cadeia x

Precisamos decidir qual regra **A** → **β** deve ser reduzida

Cadeia analisada da esquerda para direita

Configuração no autômato [**α, y**]
- **α** conteúdo da pilha
- **y** restante a ser lida

Conteúdo da pilha INVERTIDO (símbolo mais à direita no TOPO)

Transições:

1 - redução por **A** → **β** permite passar de [**α β, y**] para [**α A, y**]

2 - empilhamento de um terminal **a** permite passar de [**α, ay**] para [**αa, y**]

Método sLR(1)

s - a variante de **LR(1)** mais simples

L - cadeia de entrada lida da esquerda para a direita

R – construção de derivação invertida (símbolos do topo são os mais da direita right most)

1 – apenas 1 simbolo é examinado

sLR(1) = LR(0)

Término identificado pela **regra 0: S' →S**
Essa **regra 0** foi criada para **simplificar a identificação do fim da análise**, no **momento da sua redução**.

Construção baseada em itens.

Um item **A** → **α** • **β**

Significa que a cadeia está sendo derivada pela regra **A** → **αβ**

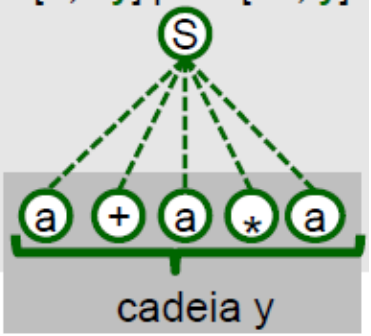
Os símbolos de **α** **já** foram lidos e faltam **ainda** os símbolos de **β**

Dessa forma o • indica o progresso da análise

- **A** → •**γ** significa início da busca por **γ**
- **A** → **γ**• significa fim da busca por **γ** e haverá a redução da regra **A** →**γ**
- **A** → **α**• **β** significa que a cadeia está sendo derivada pela regra **A** → **αβ** , os símbolos de **α** já foram lidos e faltam ainda os símbolos de **β**

Transições
1 - redução por **A** → **β** permite passar de [**α β**, **y**] para [**α A**, **y**]
2 - empilhamento de um terminal **a** permite passar de [**α**, **ay**] para [**αa**, **y**]

Método sLR(1)
s - a variante de LR(1) mais simples
L - cadeia de entrada lida da esquerda p/direita
R – construção de derivação invertida
(símbolos do topo são os mais da direita right most)
1 – apenas 1 simbolo é examinado



sLR(1)=LR(0) Método em 6 passos

as transições com $A \rightarrow \beta$ { [$\alpha\ \beta$, y] para [$\alpha\ A$, y]
[α, a y] para [αa, y] }

1. acréscimo da regra 0. S' →S, ajuda a simplificar a identificação de término quando acontece a redução pela regra 0.

2. Construção baseada em itens:

$A \rightarrow \alpha \bullet \beta$ { Os símbolos de α já foram lidos e faltam ainda os símbolos de β

Se $A \rightarrow \bullet \gamma$ significa início da busca por γ (neste caso α é ε)

Se $A \rightarrow \gamma \bullet$ significa fim da busca por γ e redução da regra $A \rightarrow \gamma$ { (neste caso β é ε)

3. Estado é representado por um conjunto de itens
Estado inicial contém o item S' →• S,
e se procede o fechamento deste estado, { **S' →• S** }

4. Fechamento de um Estado
se $A \rightarrow \alpha \bullet B\ \beta$ então acrescentamos $B \rightarrow \bullet\ \gamma$
se $B \rightarrow \bullet\ C\ \lambda$ então acrescentamos $C \rightarrow \bullet\ \mu$
e assim por diante

5. Construção dos estados seguintes:
Para cada item $A \rightarrow \alpha \bullet X\ \beta$, no estado,
deve existir no estado seguinte um item $A \rightarrow \alpha X \bullet \beta$

6. Transições de estado p com itens $\bullet\ X$ levam a um estado q com item $X\ \bullet$

sLR(1)=LR(0) Método em 6 passos

as transições com $A \rightarrow \beta$ { [$\alpha\ \beta$, y] para [$\alpha\ A$, y]
[α, a y] para [αa, y] }

1. acréscimo da regra 0. S' →S, ajuda a simplificar a identificação de término quando acontece a redução pela regra 0.

2. Construção baseada em itens:

$A \rightarrow \alpha \bullet \beta$ { Os símbolos de α já foram lidos e faltam ainda os símbolos de β

Se $A \rightarrow \bullet \gamma$ significa início da busca por γ (neste caso α é ε)

Se $A \rightarrow \gamma \bullet$ significa fim da busca por γ e redução da regra $A \rightarrow \gamma$ { (neste caso β é ε)

3. Estado é representado por um conjunto de itens
Estado inicial contém o item S' →• S,
e se procede o fechamento deste estado, { **S' →• S** }

Cuidado c/ os passos 5 e 6 !

4. Fechamento de um Estado
se $A \rightarrow \alpha \bullet B\ \beta$ **então acrescentamos** $B \rightarrow \bullet\ \gamma$
se $B \rightarrow \bullet\ C\ \lambda$ **então acrescentamos** $C \rightarrow \bullet\ \mu$
e assim por diante

5. Construção dos estados seguintes:
Para cada item $A \rightarrow \alpha \bullet X\ \beta$**, no estado,**
deve existir no estado seguinte um item $A \rightarrow \alpha X \bullet \beta$

Todos os itens $A \rightarrow \alpha X \bullet \beta$ do estado seguinte devem corresponder a todos os itens $A \rightarrow \alpha \bullet X\ \beta$ do estado anterior

6. Transições de estado p com itens $\bullet\ X$ **levam a um estado q com item** $X\ \bullet$

Este novo item deve gerar propagação de itens, caso início de β seja um **ñterminal**

Cap.3-Análise Sintática ascendente sLR(1) tbm chamada de LR(0)

Gramática

0. $S' \to E$
1. $E \to E+T$
2. $E \to T$
3. $T \to T * F$
4. $T \to F$
5. $F \to (E)$
6. $F \to a$

Estado 0

$S' \to .E$	E	1
$E \to .E+T$	E	1
$E \to .T$	T	2
$T \to .T*F$	T	2
$T \to .F$	F	3
$F \to .(E)$	(	4
$F \to .a$	a	5

Estado 1

$S' \to E.$	r	0
$E \to E.+T$	+	6

Estado 2

$E \to T.$	r	2
$T \to T.*F$	*	7

Estado 3

$T \to F.$	r	4

Estado 4

$F \to (.E)$	E	8
$E \to .E+T$	E	8
$E \to .T$	T	2
$T \to .T*F$	T	2
$T \to .F$	F	3
$F \to .(E)$	(	4
$F \to .a$	a	5

Estado 5

$F \to a.$	r	6

Estado 6

$E \to E+.T$	T	9
$T \to .T*F$	T	9
$T \to .F$	F	3
$F \to .(E)$	(	4
$F \to .a$	a	5

Estado 7

$T \to T*.F$	F	10
$F \to .(E)$	(	4
$F \to .a$	a	5

Estado 8

$F \to (E.)$	)	11
$E \to E.+T$	+	6

Estado 9

$E \to E+T.$	r	1
$T \to T.*F$	*	7

Estado 10

$T \to T*F.$	r	3

Estado 11

$F \to (E).$	r	5

Follow

S'	$
E	$ +)
T	* $ +)
F	* $ +)

transições 2 tipos

- se $\delta(p,a)=q$ empilha a e q
- se $\delta(p,a)=r_i$ observando que $i.\ B \to \gamma$ reduz $|\gamma|$ } n.º de pops depois determina $q = \delta(\text{topo}, B)$ e empilha B e q

tabela sLR(1)

	E	T	F	(	a	+	*	)	$
0	1	2	3	4	5				
1						6			r_0
2						r_2	7	r_2	r_2
3						r_4	r_4	r_4	r_4
4	8	2	3	4	5				
5						r_6	r_6	r_6	r_6
6		9	3	4	5				
7			10	4	5				
8						6		11	
9						r_1	7	r_1	r_1
10						r_3	r_3	r_3	r_3
11						r_5	r_5	r_5	r_5

Cap.3-Análise Sintática ascendente sLR(1), tbm chamada de (LR(0)

Regras

0. **S' → E**
1. **E → E + T**
2. **E → T**
3. **T → T * F**
4. **T → F**
5. **F → (E)**
6. **F → id**

Follow(**A**)

E: $,),+
T:$,),*,+
F:$,),+,*

Estado 0

S' → .E	E	1
E → .E+T	E	1
E → .T	T	2
T → .T*F	T	2
T → .F	F	3
F → .(E)	(	4
F → .a	a	5

Estado 1

S' → E.	r	0
E → E.+T	+	6

Estado 2

E → T.	r	2
T → T.*F	*	7

Estado 3

T → F.	r	4

Estado 4

F → (.E)	E	8
E → .E+T	E	8
E → .T	T	2
T → .T*F	T	2
T → .F	F	3
F → .(E)	(	4
F → .a	a	5

Estado 5

F → a.	r	6

Estado 6

E → E+.T	T	9
T → .T*F	T	9
T → .F	F	3
F → .(E)	(	4
F → .a	a	5

Estado 7

T → T*.F	F	10
F → .(E)	(	4
F → .a	a	5

F → (E.)	)	11
E → E.+T	+	6

Estado 9

E → E+T.	r	1
T → T.*F	*	7

Estado 10

T → T*F.	r	3

Estado 11

F → (E).	r	5

Tabela sLR(1) ouLR(0)

	E	T	F	(	id	+	*	)	$
0	1	2	3	4	5	-	-	-	-
1	-	-	-	-	-	6	-	-	*r0*
2	-	-	-	-	-	*r2*	7	*r2*	*r2*
3	-	-	-	-	-	*r4*	*r4*	*r4*	*r4*
4	8	2	3	4	5	-	-	-	-
5	-	-	-	-	-	*r6*	*r6*	*r6*	*r6*
6	-	9	3	4	5	-	-	-	-
7	-	-	10	4	5	-	-	-	-
8	-	-	-	-	-	6	-	11	-
9	-	-	-	-	-	*r1*	7	*r1*	*r1*
10	-	-	-	-	-	*r3*	*r3*	*r3*	*r3*
11	-	-	-	-	-	*r5*	*r5*	*r5*	*r5*

Exercício:

A gramática a seguir é sLR(1) ?

Justifique sua resposta.

```
S → V = E | I ( E )
E → E + V | V
V → V = I ( E ) | I
I → a
```

```
S → V = E
S → I ( E )
E → E + V
E → V
V → V = I ( E )
V → I
I → a
```

Solução na página seguinte

Estado 0

S' → . S	S	1
S → . V = E	V	2
S → . I (E)	I	3
V → . V = I (E)	V	2
V → . I	I	3
I → . a	a	4

Estado 1

S' → S .	r	0

Estado 2

S → V . = E	=	5
V → V . = I (E)	=	5

Estado 3

S → I . (E)	(	6
V → I .	r	6

Estado 4

I → a .	r	7

Estado 5

S → V = . E	E	7
V → V = . I (E)	I	8
E → . E + V	E	7
E → . V	V	9
V → . V = I (E)	V	9
V → . I	I	8
I → . a	a	4

Estado 6

S → I (. E)	E	10
E → . E + V	E	10
E → . V	V	9
V → . V = I (E)	V	9
V → . I	I	11
I → . a	a	4

Estado 7

S → V = E .	r	1
E → E . + V	+	12

Estado 8

V → V = I . (E)	(	13
V → I .	r	6

Estado 9

E → V .	r	4
V → V . = I (E)	=	14

Estado 10

S → I (E .)	)	15
E → E . + V	+	12

Estado 11

V → I .	r	6

Estado 12

E → E + . V	V	16
V → . V = I (E)	V	16
V → . I	I	11
I → . a	a	4

Estado 13

V → V = I (. E)	E	17
E → . E + V	E	17
E → . V	V	9
V → . V = I (E)	V	9
V → . I	I	11
I → . a	a	4

Estado 14

V → V = . I (E)	I	18
I → . a	a	4

Estado 15

S → I (E) .	r	2

Estado 16

E → E + V .	r	3
V → V . = I (E)	=	14

Estado 17

V → V = I (E .)	)	19
E → E . + V	+	12

Estado 18

V → V = I . (E)	(	13

Estado 19

V → V = I (E) .	r	5

follow

S	$
E	$) +
V	= $) +
I	(= $) +

	S	E	V	I	=	(	)	+	a	$
0	1		2	3					4	
1										r0
2					5					
3					r6	6	r6	r6		r6
4					r7	r7	r7	r7		r7
5		7	9	8					4	
6		10	9	11					4	
7								12		r1
8					r6	13	r6	r6		r6
9					r4		r4	r4		r4
10							15	12		
11					r6		r6	r6		r6
12			16	11					4	
13		17	9	11					4	
14				18					4	
15										r2
16					r4		r3	r3		r3
17							19	12		
18						13				
19					r5		r5	r5		r5

GRAMÁTICA

0. S' → S
1. S → V = E
2. S → I (E)
3. E → E + V
4. E → V
5. V → V = I (E)
6. V → I
7. I → a

pilha	entrada
0	a=a+a=a(a)$
0a4	=a+a=a(a)$
0I3	
0V2	
0V2=5	a+a=a(a)$
0V2=5a4	+a=a(a)$
0V2=5I8	
0V2=5V9	
0V2=5E7	
0V2=5E7+12	a=a(a)$
0V2=5E7+12a4	=a(a)$
0V2=5E7+12I11	
0V2=5E7+12V16	
0V2=5E7+12V16=14	a(a)$
0V2=5E7+12V16=14a4	(a)$
0V2=5E7+12V16=14I18	
0V2=5E7+12V16=14I18(13	a)$
0V2=5E7+12V16=14I18(13a4	)$
0V2=5E7+12V16=14I18(13I11	
0V2=5E7+12V16=14I18(13V9	
0V2=5E7+12V16=14I18(13E17)19	$
0V2=5E7+12V16	
0V2=5E7	
0S1	
0S' final com regra Ø	

	S	E	V	I	=	(	)	+	a	$
0	1		2	3					4	
1										r_0
2					5					
3					16	6	r_6	r_6		r_6
4					r_7	r_7	r_7	r_7		r_7
5		7	9	8					4	
6		10	9	11					4	
7								12		r_1
8					r_6	13	r_6	r_6		r_6
9					14		r_4	r_4		r_4
10							15	12		
11					r_6		r_6	r_6		r_6
12			16	11					4	
13		17	9	11					4	
14				18					4	
15										r_2
16					14		r_3	r_3		r_3
17							19	12		
18						13				
19					r_5		r_5	r_5		r_5

transições:

1 - se $\delta(p,a) = q$ empilha a e q (empilha)

2 - se $\delta(p,a) = r_i$ (i. $B \to \gamma$) (redução)

- redução (pop de $|\gamma|$)
- $\delta(topo, B) = q$ < empilha B / empilha q

3 - se a redução é 0. S' → E (aceita)

encerra o processo com sucesso.

A decisão sobre os símbolos look ahead é mais criteriosa do que em sLR(1).

O controle é mais fino ! Permite que gramáticas não sLR(1) sejam analisadas por LR(1)

Como calcular os look ahead ?

Sem usar o Follow ?

Essa é a precisão que empodera a LR(1)

Cap 3 - Análise Sintática LR(1)

Como calcular os look aheads ? É ≠ Follow

Definição do item muda
de [A → α . β] para [A → α . β, u]

u é 1 símbolo terminal ($ também pode ser)
↳ este **1** é aquele de **LR(k)**, quer dizer, **k** tem valor **1**,
em outras palavras, | **u** | = **k** = **1**

Problemas com **k > 1** é o **número de colunas que sobe** a n^k , para **n terminais no alfabeto**.

Atenção! Se **k = 0**, então o método é **sLR (1)**, por isso chamado tbm de **LR(0)**

Definição do item muda:

Um item LR(1) é da forma [A → α . β, u]

u é um símbolo terminal ($ também pode ser)

→ este 1 é aquele de **LR(k)**, quer dizer, **k** tem valor **1**,

em outras palavras, | **u** | = k = 1

v = First(βu)

Passo 1 do método destacado para entender o First(βu):
O fechamento de um estado q deve ser feito,
para cada item [A → α•Bβ, u] ϵ q,
acrescentando-se a q,
todos os itens da forma [B → •γ , v],
em que B → γ é uma regra de B, e v ϵ First (βu).

Notar que os símbolos de **v ϵ First (βu)** são os que podem ocorrer depois de **B**, supondo que **u** ocorreu depois de **A**, e que a regra **A → αBβ** foi aplicada na derivação.

Este **processo de fechamento em cada estado** deve ser repetido enquanto **novos itens forem sendo acrescentados ao estado q**.

Construção da coleção de estados

1.O fechamento de um estado q é feito, para cada item[A→α•Bβ, u] ϵ q, acrescentando-se a q todos os itens da forma [B → •γ , v], em que B → γ é uma regra de B, e v ϵ First (βu).

Notar que os símbolos de **v ϵ First (βu)** são os que podem ocorrer depois de **B**, supondo que **u** ocorreu depois de **A**, e que a regra **A → αBβ** foi aplicada na derivação.

Este processo deve ser repetido enquanto novos itens forem sendo acrescentados ao estado q.

2. A transição entre os estados é definida como:
Se um estado p possui itens da forma [A → α•Xβ, u], o estado seguinte q, resultante de δ (p, X), alcançado com X a partir do estado p anterior, é construído pelo fechamento do conjunto de itens da forma [A → αX•β, u], obtidos através de todos os itens com X depois do ponto no estado p.

Entendeu ? Leva u p/q com o item

3. O estado inicial 0 é construído pelo fechamento de { [S' →•S, $] }

4. A coleção dos estados do analisador LR (1) é obtida a partir do estado 0, por todas as transições possíveis.

Ações na análise

Os empilhamentos são feitos obedecendo as transições

As reduções são feitas de acordo

- com os símbolos look aheads do item
- e com o símbolo de entrada a ser lido, quer dizer,

se um estado tem um item [B → γ • , v] e

o símbolo de entrada a ser lido coincide com v,

então a este estado corresponde a ação de redução pela regra B → γ

Regras:

0. $E' \to E$
1. $E \to S\,x$
2. $E \to y\,S\,z$
3. $E \to O\,z$
4. $E \to y\,O\,x$
5. $S \to +$
6. $O \to +$

	first	follow
E'	+, y	$
E	+, y	$
S	+	x, z
O	+	x, z

Estado 0

item regra	v = first(βu)	seguinte	
E' → .E	$	E	1
E → .Sx	$	S	2
E → .ySz	$	y	3
E → .Oz	$	O	4
E → .yOx	$	y	3
S → .+	x	+	5
O → .+	z	+	5

atenção: em $E \to .Sx$, $\beta u = x\$$, $v = x$
em $E \to .Oz$, $\beta u = z\$$, $v = z$

Estado 1

E' → E.	$	r0

Estado 2

		seguinte	
E → S.x	$	x	6

Estado 3

	v = first(βu)	seguinte	
E → y.Sz	$	S	7
E → y.Ox	$	O	8
S → .+	z	+	9
O → .+	x	+	9

atenção:
em $E \to y.Sz$, $\beta u = z\$$, $v = z$
em $E \to y.Ox$, $\beta u = x\$$, $v = x$
assim o seguinte com + não aproveita o estado 5!

Estado 4

E → O.z	$	z	10

Estado 5

S → +.	x	r	5
O → +.	y	r	6

Estado 6

		redução
E → Sx.	$	r1

Estado 7

		seguinte	
E → yS.z	$	z	11

Estado 8

		seguinte	
E → yO.x	$	x	12

Estado 9

		reduções	
S → +.	z	r	5
O → +.	x	r	6

Estado 10

		redução
E → Oz.	$	r3

Estado 11

E → ySz.	$	r2

Estado 12

E → yOx.	$	r4

tabela LR(1)

	E	S	O	x	y	z	+	$
0	1	2	4		3		5	
1								R_0
2				6				
3		7	8				9	
4						10		
5				R_5	R_6			
6								R_1
7						11		
8				12				
9				R_6		R_5		
10								R_3
11								R_2
12								R_4

sem conflito! então a gramática e a linguagem são livre de contexto e LR(1)!

Exercício:
A gramática a seguir é LR(1) ?
Justifique sua resposta.

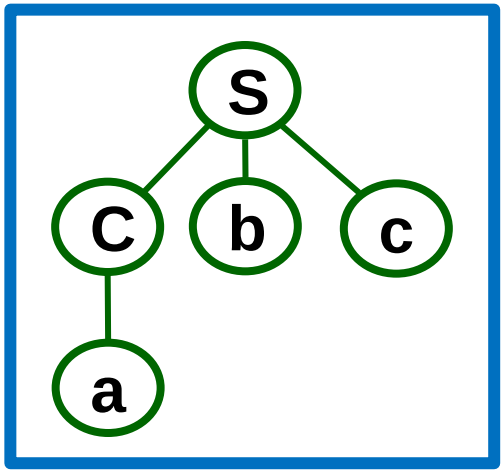
S → Aa | bAc | Bc | bBa

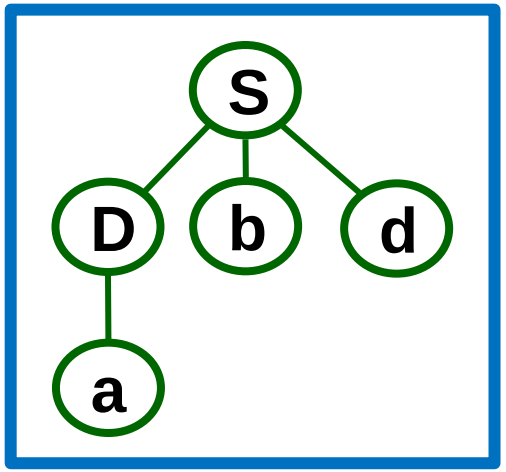
A → d

B → d

Linguagens cujas gramáticas não são LR(1) e nem ambíguas

P = {
S→Cbc | Dbd
C→a
D→a
}





Esta linguagem, apesar de não ser ambígua e de gerar apenas duas cadeias: abc e abd, não é LR(1).

Vemos que para decidir entre C e D, precisamos olhar o 3º símbolo, porque o 2º símbolo não traz informação suficiente.

Linguagens cujas gramáticas não são LR(1) e nem ambíguas

Um outro exemplo, a linguagem

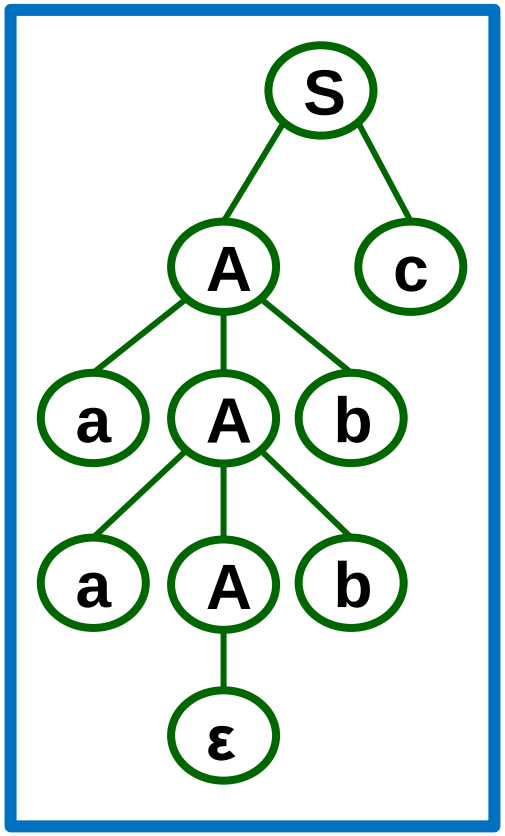
$$\{a^i b^i c \mid i \geq 0\} \cup \{a^i b^{2i} d \mid i \geq 0\}.$$

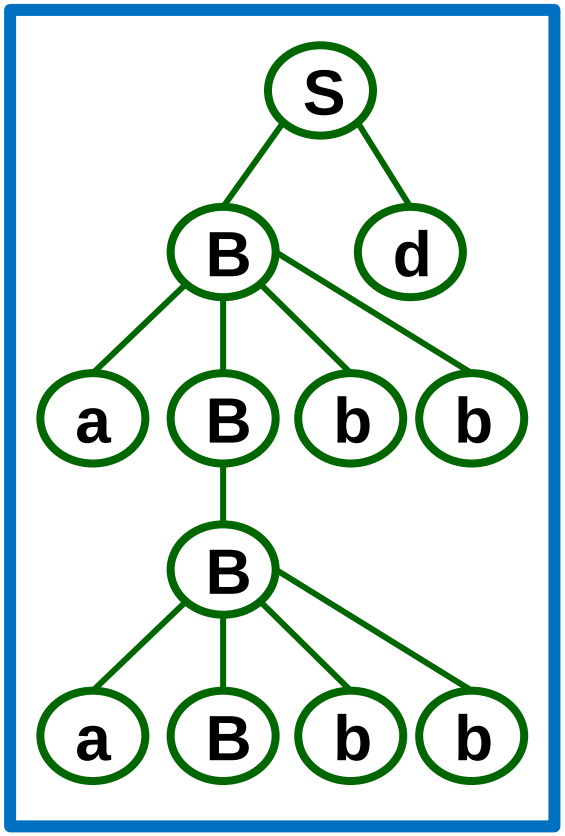
Quando o 1º b é encontrado, não há informação suficiente para decidir qual a redução, se para A ou B ?

Isto pode acontecer para qualquer gramática desta linguagem .

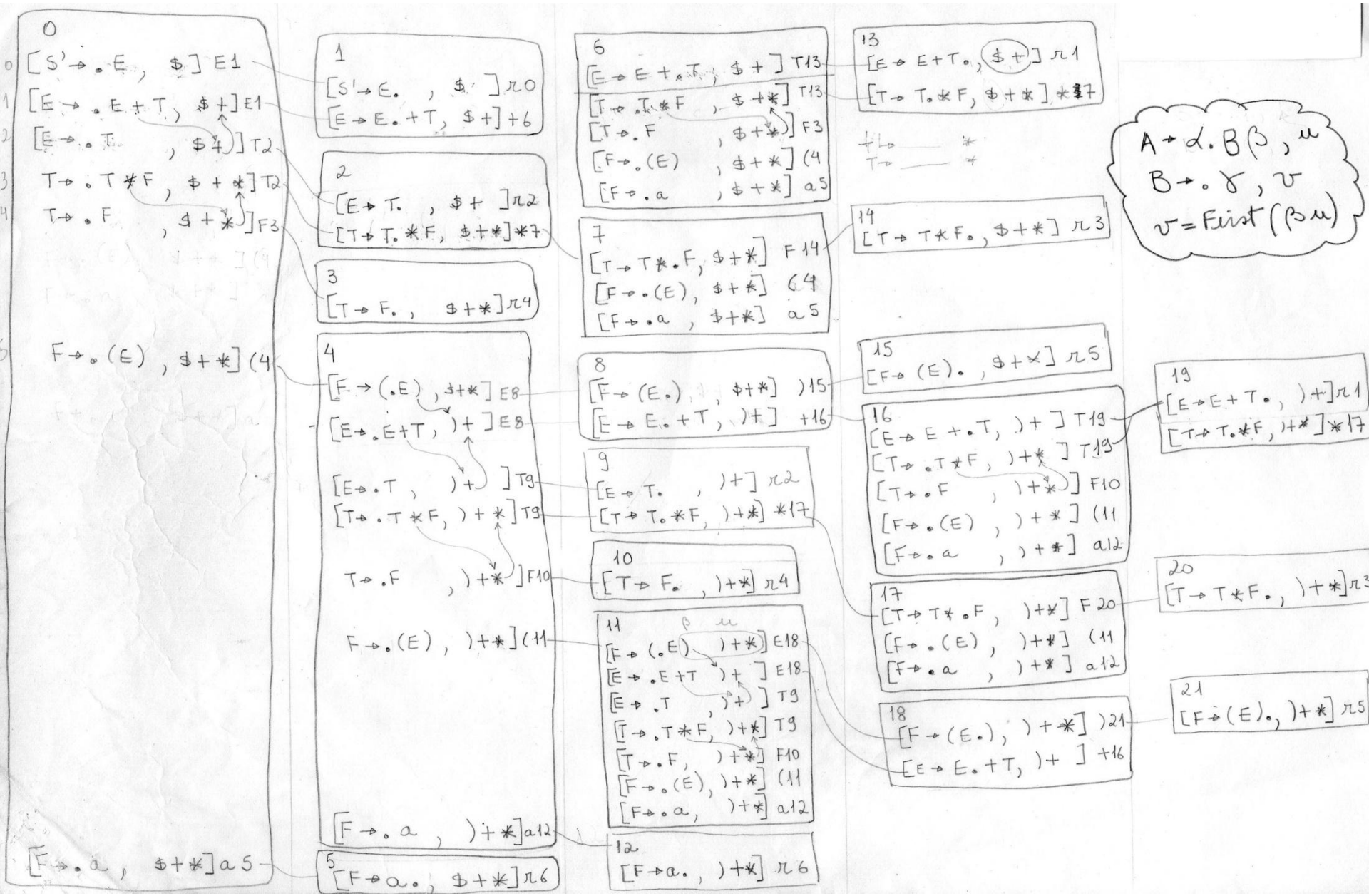
Ao encontrar o primeiro b, é impossível decidir qual regra usar:
- uma regra que associa um b a cada a, ou
- uma regra que associa dois b´s a cada a.

P = {
S→Ac | Bd
A→aAb | ε
B→aBbb | ε
}





Cap.3 Análise Sintática Ascendente LR(1)



Cap.3 Análise Sintática Ascendente LR(1)

Tabela de ações

estados \ símbolos	E	T	F	$	+	*	(	)	a
0	1	2	3				4		5
1				r0	6				
2				r2	r2	7			
3				r4	r4	r4			
4	8	9	10				11		12
5				r6	r6	r6			
6		13	3				4		5
7			14				4		5
8					16			15	
9					r2	17		r2	
10					r4	r4		r4	
11	18	9	10				11		12
12					r6	r6		r6	
13				r1	r1	7			
14				r3	r3	r3			
15				r5	r5	r5			
16		19	10				11		12
17			20				11		12
18					16			21	
19					r1	17		r1	
20					r3	r3		r3	
21					r5	r5		r5	

passos de aceitação da cadeia (a+a)*a

6

Pilha	Entrada	ação
0	(a+a)*a	s4
0(4	a+a)*a	s12
0(4a12	+a)*a	r6
0(4F10		r4
0(4T9		r2
0(4E8		s16
0(4E8+16	a)*a	s12
0(4E8+16a12	)*a	r6
0(4E8+16F10		r4
0(4E8+16T19		r1
0(4E8		s15
0(4E8)15	*a	r5
0F3		r4
0T2		s7
0T2*7	a	s5
0T2*7a5	$	r6
0T2*7F14		r3
0T2		r2
0E1		r0

transições:

1 - se $\delta(p,a) = q$ empilha a e q (empilha)

2 - se $\delta(p,a) = r_i$ (i. $B \to \gamma$) (reduz)

- reduz (pop de $|\gamma|$)
- δ(topo, B) = q < empilha B, empilha q

3 - se a redução é 0. $S' \to E$ (aceita)

encerra o processo com sucesso.

Nas reduções, os lookaheads do item são utilizados para indicar as ações no estado

Cap 3 - Análise Sintática laLR(1)

Desde o analisador LL(1) estamos enriquecendo nossas ferramentas, método a método.

O analisador sLR(1) - considerado um LR(0) - é capaz de reconhecer linguagens que não são LL(1). Suas limitações de um lookahead em relação ao LR(1) podem impedi-lo de reconhecer linguagens reconhecidas pelo método LR(1).

O laLR(1), devido a um critério mais apurado, nos lookaheads, tem vantagem sobre o sLR(1).

Podemos classificar as linguagens como a seguir:
uma linguagem LL(1) é na maioria das vezes sLR(1);
uma linguagem sLR(1) é laLR(1);
uma linguagem laLR(1) é LR(1).

Cap 3 - Análise Sintática laLR(1)

O laLR(1) é um método de análise que calcula os lookaheads, gerando menos estados que o LR(1).

Para a linguagem Pascal,
O laLR(1) gera 300 estados e o LR(1) 3000 estados.

O laLR(1) é usado há mais de 42 anos,
popularizado pelo Yacc (bison no Linux), uma ferramenta de geração de compilador, cujo parâmetro é a gramática da linguagem.

A idéia é simples, baseada no LR(1),
unindo estados que tenham o mesmo núcleo de itens,
antes os lookaheads são desprezados,
e depois vão sendo calculados.

Este método junta sLR(1) com LR(1):
- antes, são definidos os estados pelo sLR(1)
- depois, são calculados os lookaheads pelo First (βu).

Processo de construção

1. Construímos os estados LR(0) para a gramática, e as funções de transição correspondentes.

(são exatamente os estados e as transições de sLR(1) !)

2. Acrescentamos os lookaheads (os itens), começando por acrescentar o item [S' → •S, $] ao estado 0.
Acrescentamos então o estado 0 a uma lista de estados a serem tratados.

O que significa tratar um estado q ?
2.a Remover o estado q da lista;
2.b fazer o fechamento do estado q propagando os lookaheads;
2.c para cada item da forma [A → α•Xβ, u], em q,
verificar se o item [A → αX•β, u] já se encontra em q',
caso não seja encontrado:

$\delta(q, X) = q'$

2.c.i o item deve ser acrescentado a q',
2.c.ii se o estado q' não faz parte da lista, então é acrescentado para tratamento posterior.

Um estado pode ser tratado várias vezes, inclusive, durante o seu próprio tratamento, ele pode ser reincluído na lista de estados e o passo 2 deve continuar até que a lista fique vazia.

Regras:

0. $E' \to E$
1. $E \to Sx$
2. $E \to ySz$
3. $E \to Oz$
4. $E \to yOx$
5. $S \to +$
6. $O \to +$

	first	follow
E'	$+, y$	\$
E	$+, y$	\$
S	$+$	x, z
O	$+$	x, z

Estado 0

itens		seguinte	redução
$E' \to .E$	\$	E	1
$E \to .Sx$	\$	S	2
$E \to .ySz$	\$	y	3
$E \to .Oz$	\$	O	4
$E \to .yOx$	\$	y	3
$S \to .+$	x	$+$	5
$O \to .+$	z	$+$	5

Estado 1

		seguinte ou redução
$E' \to E.$	\$	$r0$

Estado 2

itens	u	seguinte	redução
$E \to S.x$	\$	x	6

Estado 3

$E \to y.Sz$	\$	S	7
$E \to y.Ox$	\$	O	8
$S \to .+$	z	$+$	5
$O \to .+$	x	$+$	5

Estado 4

$E \to O.z$	\$	z	9

Estado 5

$S \to +.$	x, z	r	5
$O \to +.$	z, x	r	6

Lista: ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ ~~11~~

Estado 6

$E \to Sx.$	\$	r	1

Estado 7

$E \to yS.z$	\$	z	10

Estado 8

$E \to yO.x$	\$	x	11

Estado 9

$E \to Oz.$	\$	r	3

Estado 10

$E \to ySz.$	\$	r	2

Estado 11

$E \to yOx.$	\$	r	4

tabela laLR(1)

	E	S	O	x	y	z	+	\$
0	1	2	4		3		5	
1				6				r_0
2								
3		7	8				5	
4						9		
5				r_5 r_6		r_5 r_6		
6								r_1
7						10		
8				11				
9								r_3
10								r_2
11								r_4

conflito → então não é laLR(1)

Exercício laLR(1) pg 1/3

LALR

gramática

. $S' \to E$
. $E \to E + P$
. $E \to P$
. $P \to \text{if } B \text{ th } P \text{ el } P$
. $P \to a$
. $B \to E = E$
. $B \to \text{if } B \text{ th } B \text{ el } B$

Estado ∅

	l.a.	ações
$S' \to . E$	$	E1
$E \to . E + P$	$ +	E1
$E \to . P$	$ +	P2
$P \to . \text{if } B \text{ th } P \text{ el } P$	$ +	if3
$P \to . a$	$ +	a4

~~∅ 1 2 3 4 5 6 7 8 2 4 9~~
~~10 11 5 12 13 3 4 14 2 8 15~~
~~16 6 5 17 18 7 8 19 10 20 16~~
~~11 12 13 21 15 14 2 15 16~~
~~5 17 18 19 9 20 21~~

1.	l.a.	ações
$S' \to E.$	$	r∅
$E \to E. + P$	$ +	+5

2.		
$E \to P.$	$, +, =, th, el	r2

3.		
$P \to \text{if}. B \text{ th } P \text{ el } P$	$ + el, th, =	B6
$B \to . E = E$	th	E7
$B \to . \text{if } B \text{ th } B \text{ el } B$	th	if8
$E \to . E + P$	=, +	E7
$E \to . P$	=, +	P2
$P \to . \text{if } B \text{ th } P \text{ el } P$	=, +	if8
$P \to . a$	=, +	a4

4.		
$P \to a.$	$, +, =, el, th	r4

Atenção!

quando estivermos tratando 3,

não incluir os l.a.s no 1.º item, provenientes dos 6.º e 7.º itens.

Lembrando:

O fechamento de um estado q é feito, acrescentando ao estado q, para cada item $[A \to \alpha . B \beta, u] \in q$, todos os itens da forma $[B \to . \gamma, v]$, onde $B \to \gamma$ é uma regra de B, e $v \in First(\beta u)$.

5.	l.a.	ações
$E \to E + . P$	$ + = th el	P9
$P \to . \text{if } B \text{ th } P \text{ el } P$	$ + = th el	if3
$P \to . a$	$ + = th el	a4

6.		
$P \to \text{if } B . \text{th } P \text{ el } P$	$ + = th el	th 10

7.		
$B \to E . = E$	th el	=11
$E \to E . + P$	=, +	+5

8.		
$B \to \text{if}. B \text{ th } B \text{ el } B$	th el	B12
$P \to \text{if}. B \text{ th } P \text{ el } P$	= + el	B12
$B \to . E = E$	th	E7
$B \to . \text{if } B \text{ th } B \text{ el } B$	th	if8
$E \to . E + P$	=, +	E7
$E \to . P$	=, +	P2
$P \to . \text{if } B \text{ th } P \text{ el } P$	=, +	if8
$P \to . a$	=, +	a4

Repare que o ., no 1.º item, não se encontra no início, e nos 6.º e 7.º, sim.

Exercício laLR(1) pg 2/3

9.	l.a.	ação
E → E + P.	$ + = th el	r1

10.		
P → if B th. P el P	$ + el th =	P13
P → . if B th P el P	el	if3
P → . a	el	a4

11.		
B → E = . E	th, el	E14
E → . E + P	th, +, el	E14
E → . P	th, +, el	P2
P → . if B th P el P	th, +, el	if3
P → . a	th, +, el	a4

12.		
B → if B . th B el B	th, el	th15
P → if B . th P el P	=, +, el	th15

13.	l.a.	ação
P → if B th P. el P	$ + el th =	el 16

14.		
B → E = E.	th, el	r5
E → E. + P	th, +, el	+5

15.		
B → if B th. B el B	th, el	B17
P → if B th. P el P	=, +, el	P18
B → . E = E	el	E7
B → . if B th B el B	el	if8
P → . if B th P el P	el	if8
P → . a	el	a4
E → . E + P	= +	E7
E → . P	= +	P18

16.	l.a.	ação
P → if B th P el. P	$ + = el th	P19
P → . if B th P el P	$ + = el th	if3
P → . a	$ + = el th	a4

17.		
B → if B th B. el B	th el	el 20

18.		
P → if B th P. el P	= + el	el16
E → P.	= +	r2

Exercício laLR(1) pg 3/3

	l.a	ação
19. P → if B th P el P.	$ + = el th	r 3

	l.a.	ação
21. B → if B th B el B.	th el	r 6

20. B → if B th B el . B	th el	B 21
B → . if B th B el B	th el	if 8
B → . E = E	th el	E 7
E → . E + P	= +	E 7
E → . P	= +	P 2
P → . if B th P el P	= +	if 8
P → . a	= +	a 4

Ainda ex. anterior, montando a tabela e reconhecendo a cadeia if if a=a th a=a el a=a th a el a+a

Tabela que representa o diagrama

	E	P	B	+	if	th	el	a	=	$
0	1	2			3			4		
1				5						r_0
2				r_2		r_2	r_2		r_2	r_2
3	7	2	6		8			4		
4				r_4		r_4	r_4		r_4	r_4
5		9			3			4		
6						10				
7				5					11	
8	7	2	12		8			4		
9				r_1		r_1	r_1		r_1	r_1
10		13			3			4		
11	14	2			3			4		
12						15				
13							16			
14				5		r_5	r_5			
15	7	18	17		8			4		
16		19			3			4		
17							20			
18				r_2			16		r_2	
19				r_3		r_3	r_3		r_3	r_3
20	7	2	21		8			4		
21						r_6	r_6			

GRAMÁTICA

0. S' → E
1. E → E + P
2. E → P
3. P → if B th P el P
4. P → P → a
5. B → E = E
6. B → if B th B el B

entrada

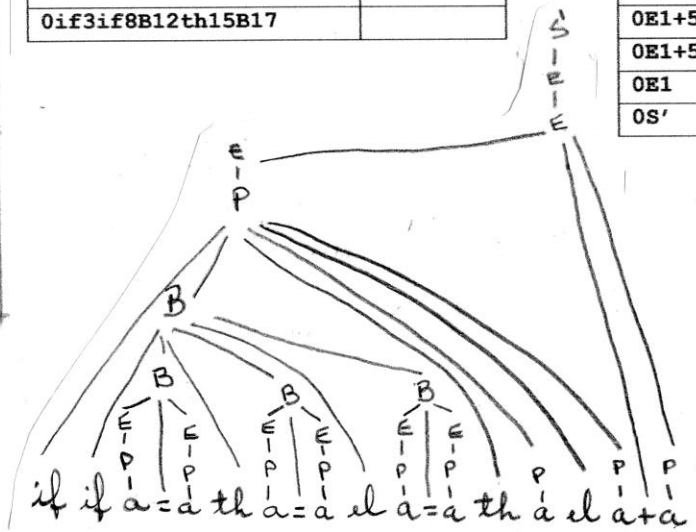
if if a = a th a = a el a = a th a el a + a $

pilha	entrada
0	if if a = a ...
0if3	if a = a th...
0if3if8	a = a th a...
0if3if8a4	= a th a =...
0if3if8P2	
0if3if8E7	
0if3if8E7=11	a th a = a...
0if3if8E7=11a4	th a = a el...
0if3if8E7=11P2	
0if3if8E7=11E14	
0if3if8B12	
0if3if8B12th15	a = a el a...
0if3if8B12th15a4	= a el a =...
0if3if8B12th15P18	
0if3if8B12th15E7	
0if3if8B12th15E7=11	a el a = a...
0if3if8B12th15E7=11a4	el a = a th...
0if3if8B12th15E7=11P2	
0if3if8B12th15E7=11E14	
0if3if8B12th15B17	

continuando a entrada:

el a = a th a el a + a $

pilha	entrada
0if3if8B12th15B17el20	a = a th a...
0if3if8B12th15B17el20a4	= a th a el...
0if3if8B12th15B17el20P2	
0if3if8B12th15B17el20E7	
0if3if8B12th15B17el20E7=11	a th a el a...
0if3if8B12th15B17el20E7=11a4	th a el a +...
0if3if8B12th15B17el20E7=11P2	
0if3if8B12th15B17el20E7=11E14	
0if3if8B12th15B17el20B21	
0if3B6	
0if3B6th10	a el a + a...
0if3B6th10a4	el a + a $
0if3B6th10P13	
0if3B6th10P13el16	a + a $
0if3B6th10P13el16a4	+ a $
0if3B6th10P13el16P19	
0P2	
0E1	
0E1+5	a $
0E1+5a4	$
0E1+5P9	
0E1	
0S'	



Transições:

1- Se $\delta(p,a) = q$, empilha a e q
2- Se $\delta(p,a) = r_i$ e $i.B \to \gamma$ reduz $|\gamma|$ (nº de pop's) verifica na tabela o estado do topo com B $\delta(\text{topo}, B) = q$, então empilha por B e q.
3 - qdo. chegar a r_0 encerra.

Stm → Stm; Stm
Stm → id := Exp
Stm → print (ExpList)
Exp → id
Exp → num
Exp → Exp Binop Exp
Exp → (Stm,Exp)
ExpList → Exp,ExpList
ExpList → Exp
Binop → +
Binop → -
Binop → *
Binop → /

P/facilitar, vamos trocar os símbolos terminais e ñ terminais por uma letra só

Stm S
Exp E
Binop B
ExpList L
id i
num n
:= =
print p

gramática, a seguir com acrescimo da regra 0 : S' → S

Regras:

0. S' → S
1. S → S; S
2. S → i := E
3. S → p (L)
4. E → i
5. E → n
6. E → E B E
7. E → (S,E)
8. L → E, L
9. L → E
10. B → +
11. B → -
12. B → *
13. B → /

Estado 0

S' → . S	$	S	1
S → . S; S	$;	S	1
S → . i = E	$;	i	2
S → . p (L)	$;	p	3

Estado 1

S' → S .	$	r	∅
S → S . ; S	$;	;	4

Estado 2

S → i . = E	$; ,	=	5

Estado 3

S → p . (L)	$; ,	(	6

Estado 4

S → S; . S	$; ,	S	7
S → . S; S	$; ,	S	7
S → . i = E	$; ,	i	2
S → . p (L)	$; ,	p	3

Estado 5

S → i = . E	$; ,	E	8
E → . i	$; + - * / ,	i	9
E → . n	$; + - * / ,	n	10
E → . E B E	$; + - * / ,	E	8
E → . (S,E)	$; + - * / ,	(	11

Estado 6

S → p (. L)	$; ,	L	12
L → . E, L	)	E	13
L → . E	)	E	13
E → . i	)) + - * /	i	9
E → . n	)) + - * /	n	10
E → . E B E	)) + - * /	E	13
E → . (S,E)	)) + - * /	(	11

Estado 7

S → S; S .	$; ,	r	1
S → S . ; S	$; ,	;	4

~~0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31~~

Estado 8

S → i = E .	$; ,	r	2
E → E . B E	$; + - * / ,	B	14
B → . +	in (	+	15
B → . -	in (	-	16
B → . *	in (	*	17
B → . /	in (	/	18

Estado 9

E → i .	$; + - * / ,)	r	4

Estado 10

E → n .	$; + - * / ,)	r	5

Estado 11

E → (. S, E)	$; + - * / ,)	S	19
S → . S; S	) ;	S	19
S → . i = E	) ;	i	2
S → . p (L)	) ;	p	3

Estado 12

S → p (L .)	$; ,	)	20

Estado 13

L → E . , L	)	,	21
L → E .	)	r	9
E → E . B E	)) + - * /	B	14
B → . +	in (	+	15
B → . -	in (	-	16
B → . *	in (	*	17
B → . /	in (	/	18

Estado 14

E → EB.E	$;+-*/)	E	22
E → .i	$;+-*/)	i	9
E → .n	$;+-*/)	n	10
E → .EBE	$;+-*/)	E	22
E → .(S,E)	$;+-*)	(	11

Estado 15

B → +.	in(	r	10

Estado 16

B → −.	in(	r	11

Estado 17

B → *.	in(	r	12

Estado 18

B → /.	in(	r	13

Estado 19

E → (S., E)	$;+-*/,)	,	23
S → S.;S	,;	;	4

Estado 20

S → p(L).	$;,	r	3

Estado 21

L → E,.L	)	L	24
L → .E,L	)	E	13
L → .E	)	E	13
E → .i	,)+-*/	i	9
E → .n	,)+-*/	n	10
E → .EBE	,)+-*/	E	13
E → .(S,E)	,)+-*/	(	11

Estado 22

E → EBE.	$;+-*/)	r	6
E → E.BE	$;+-*/)	B	14
B → .+	in(	+	15
B → .−	in(	−	16
B → .*	in(	*	17
B → ./	in(	/	18

Estado 23

E → (S,.E)	$;+-*/,)	E	25
E → .i	)+-*/	i	9
E → .n	)+-*/	n	10
E → .EBE	)+-*/	E	25
E → .(S,E)	)+-*/	(	11

Estado 24

E → E,L.)		r	8

Estado 25

E → (S,E.)	$;+-*/,)	)	26
E → E.BE	)+-*/	B	14
B → .+	in(	+	15
B → .−	in(	−	16
B → .*	in(	*	17
B → ./	in(	/	18

Estado 26

E → (S,E).	$;+-*/,)	r	7

	S	E	L	B	;	i	=	p	(	)	x	,	+	-	*	/	$
0	1					2		3									
1					4												r_0
2							5										
3									6								
4	7					2		3									
5		8				9			11		10						
6		13	12			9			11		10						
7					r_1							r_1					r_1
8				14	r_2							r_2	15	16	17	18	r_2
9					r_4					r_4		r_4	r_4	r_4	r_4	r_4	r_4
10					r_5							r_5	r_5	r_5	r_5	r_5	r_5
11	19					2		3									
12										20							
13				14						r_9		21	15	16	17	18	
14		22				9			11		10						
15						r_{10}			r_{10}		r_{10}						
16						r_{11}			r_{11}		r_{11}						
17						r_{12}			r_{12}		r_{12}						
18						r_{13}			r_{13}		r_{13}						
19					4							23					
20					r_3							r_3					r_3
21		13	24			9			11		10						
22				14	r_6					r_6			15 r_6	16 r_6	17 r_6	18 r_6	r_6
23		25				9			11		10						
24										r_8							
25				14						26			15	16	17	18	
26					r_7					r_7		r_7	r_7	r_7	r_7	r_7	r_7

← conflito
não é LALR (1)