Universidade de Santa Cruz do Sul – UNISC
Departamento de informática

COMPILADORES

Análise sintática

Analisadores Sintáticos LR mais poderosos
- LR canônico, e
- LALR

Prof. Geovane Griesang
geovanegriesang@unisc.br

Análise sintática

Análise ascendente – Análise LR

Toda gramática SLR é não-ambígua, mas existem diversas gramáticas não-ambíguas que não são SLR.

Vamos considerar as seguintes produções

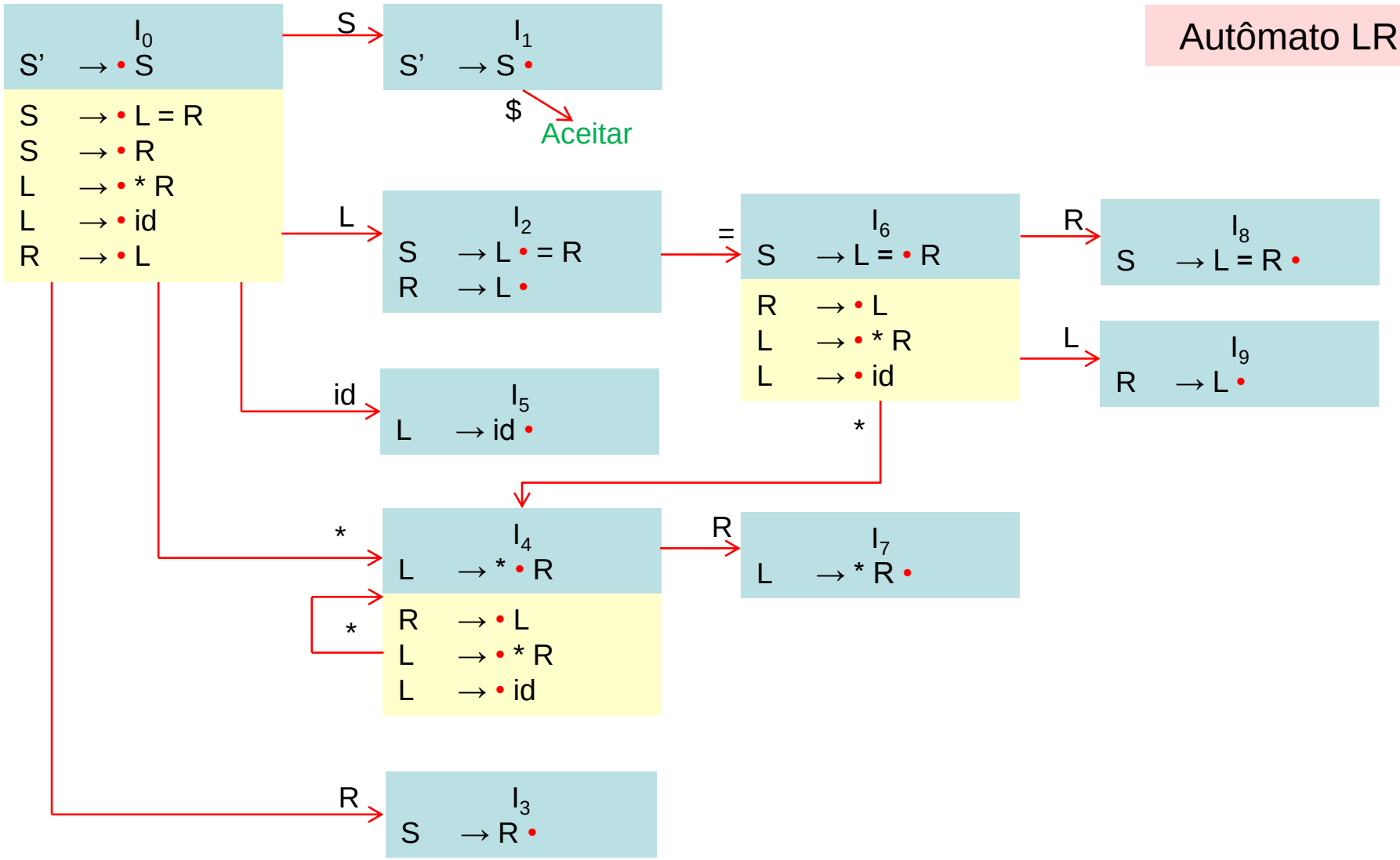
S	→	L = R | R
L	→	* R | id
R	→	L

(1)	S	→	L = R
(2)	S	→	R
(3)	L	→	* R
(4)	L	→	id
(5)	R	→	L

Análise sintática

S → L = R | R
L → * R | id
R → L

Autômato LR(0)

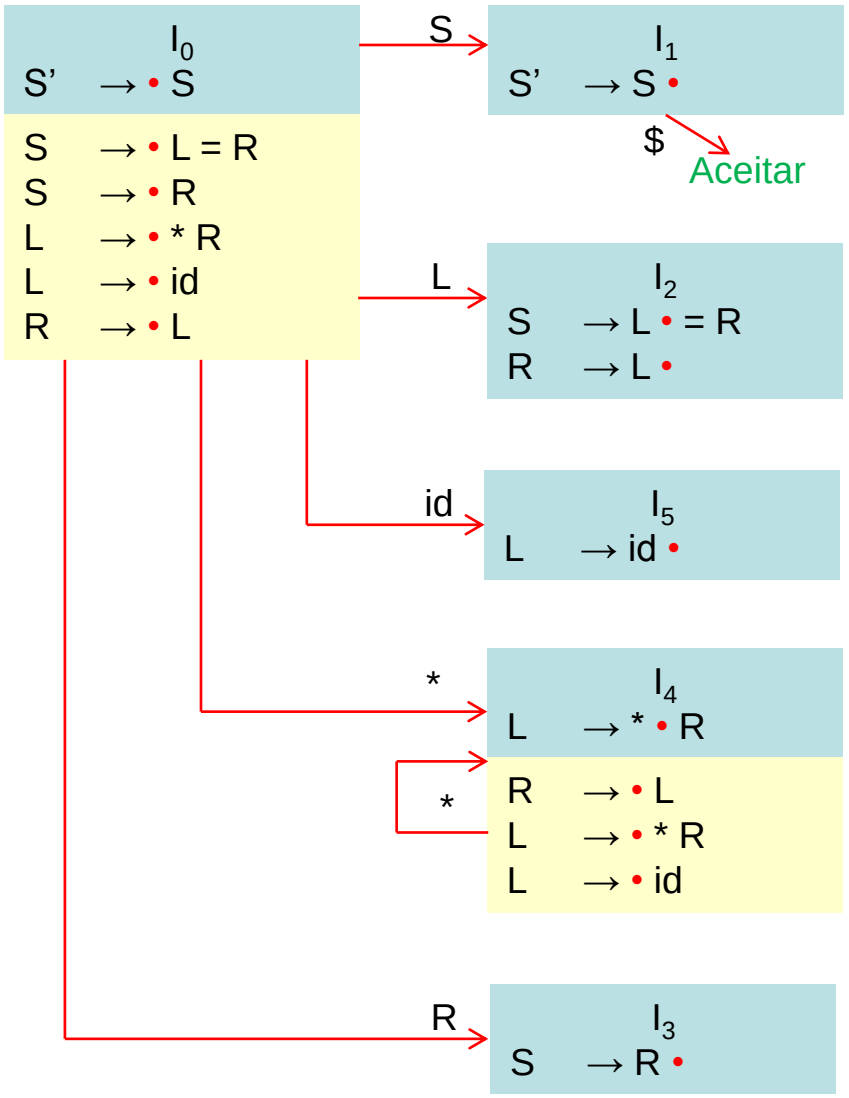


Análise sintática

S → L = R | R
L → * R | id
R → L

Tabela LR(0)

Estado	ACTION				GOTO		
	=	*	id	$	S	L	R
0		s4	s5		1	2	3
1				Acc			
2							
3							
4		s4					
5							
6							
7							
8							
9							

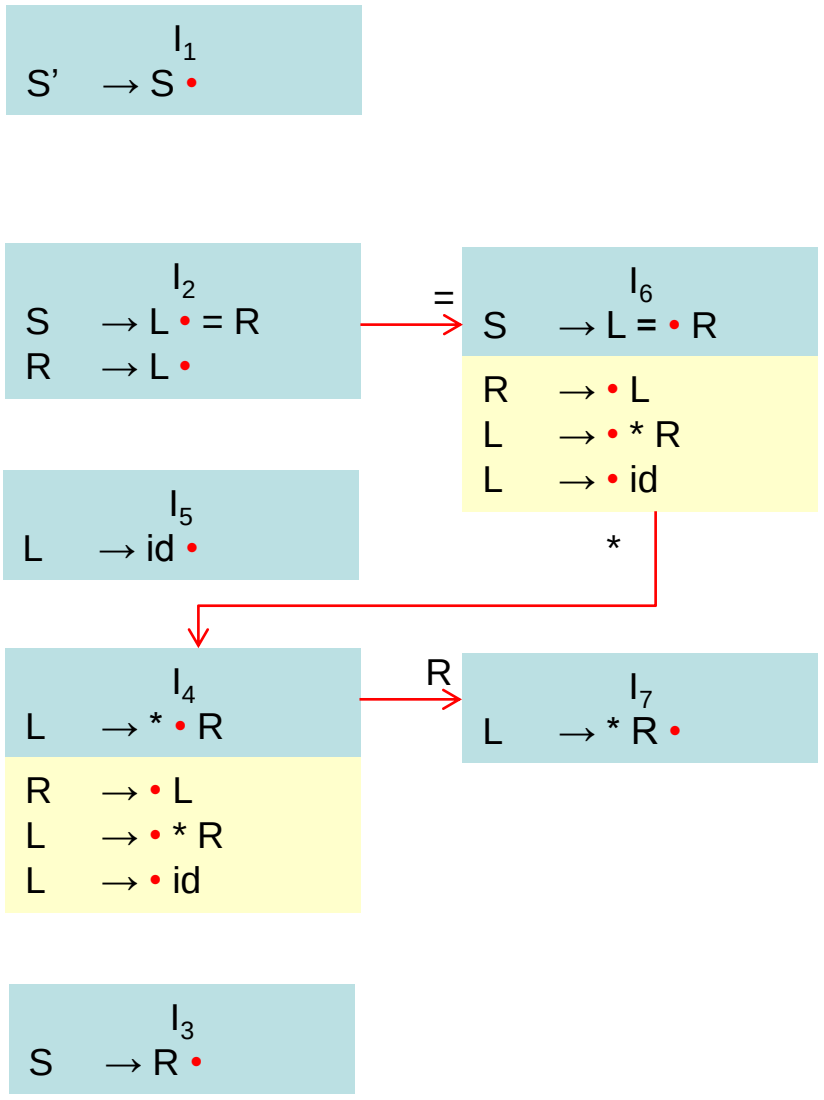


Análise sintática

S → L = R | R
L → * R | id
R → L

Tabela LR(0)

Estado	ACTION				GOTO		
	=	*	id	$	S	L	R
0		s4	s5		1	2	3
1				Acc			
2	s6						
3							
4		s4					7
5							
6		s4					
7							
8							
9							

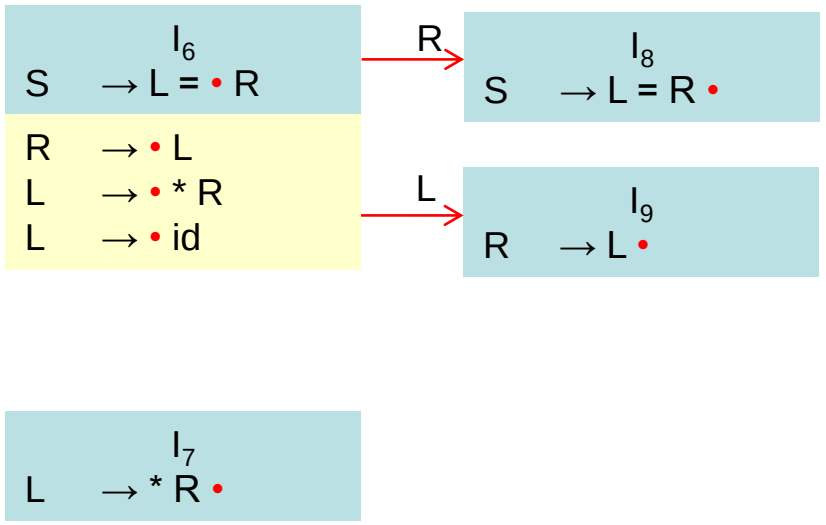


Análise sintática

S → L = R | R
L → * R | id
R → L

Tabela LR(0)

Estado	ACTION				GOTO		
	=	*	id	$	S	L	R
0		s4	s5		1	2	3
1				Acc			
2	s6						
3							
4		s4					7
5							
6		s4				9	8
7							
8							
9							



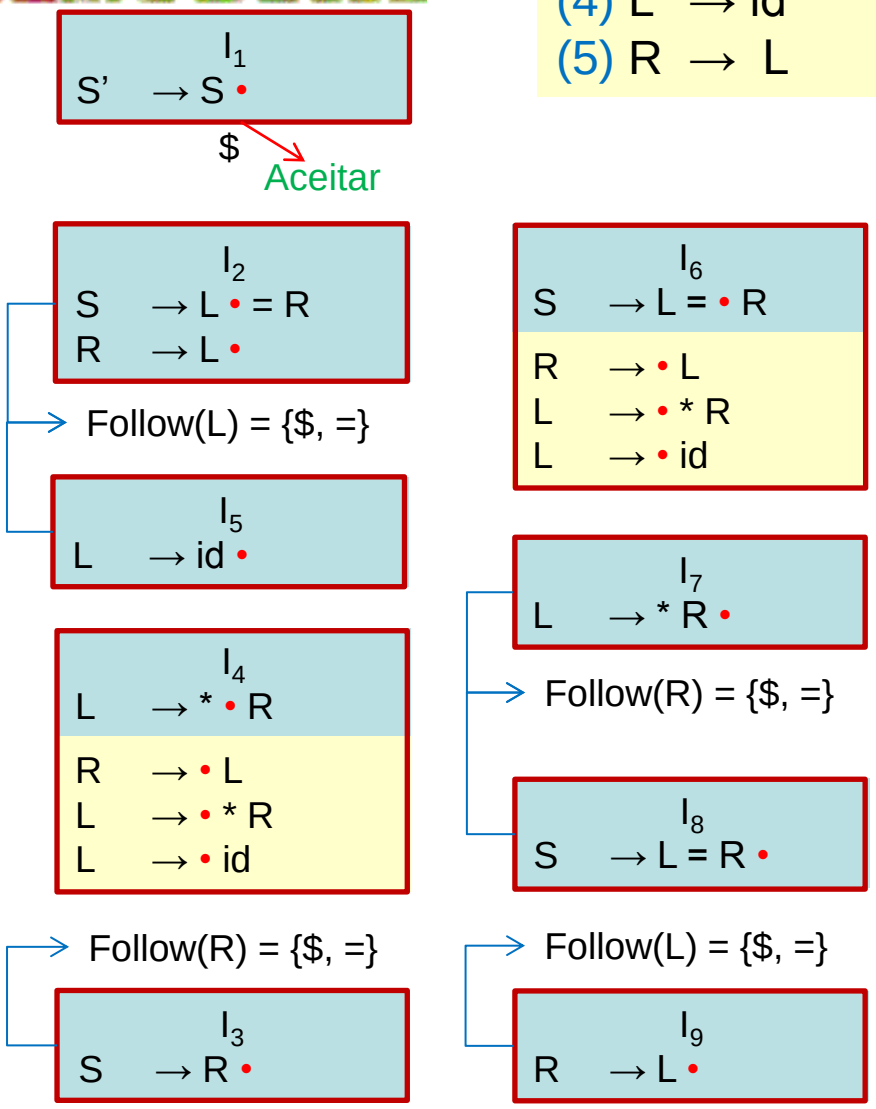
Análise sintática

Follow(S) = {$}
Follow(L) = {$, =}
Follow(R) = {$, =}

(1) S → L = R
(2) S → R
(3) L → * R
(4) L → id
(5) R → L

Tabela LR(0)

Estado	ACTION				GOTO		
	=	*	id	$	S	L	R
0		s4	s5		1	2	3
1				Acc			
2	s6			r5			
3	r2			r2			
4		s4					7
5	r4			r4			
6		s4				9	8
7	r3			r3			
8	r1			r1			
9	r5			r5			



Análise sintática

S → L = R | R
L → * R | id
R → L

Análise ascendente – Análise LR

Vamos considerar o conjunto de itens I_2.

I_2: S → L • = R
R → L •

O primeiro item deste conjunto faz com que ACTION[2, =] seja "*shift 6*".

Estado	ACTION				GOTO		
	=	*	id	$	S	L	R
2	s6			r2			

Análise sintática

Follow(S) = {$}
Follow(L) = {$, =}
Follow(R) = {$, =}

S → L = R | R
L → * R | id
R → L

I_2: S → L • = R
R → L •

Análise ascendente – Análise LR

Como FOLLOW(R) contém =, o segundo item define ACTION[2,=] como "*reduce* segundo a produção R → L".

(5) R → L

Como existe uma ação de reduzir e transferir p/ o estado 2 e a entrada = em ACTION[2, =], o estado 2 caracteriza um conflito *shift/reduce* sob o símbolo de entrada =.

Estado	ACTION				GOTO		
	=	*	id	$	S	L	R
2	s6			r5			

r5

Análise sintática

S → L = R | R
L → * R | id
R → L

I_2: S → L • = R
R → L •

Análise ascendente – Análise LR

A gramática anterior não é ambígua.

Esse conflito aparece porque o método para construção de analisadores SLR não é suficientemente poderoso para lembrar o contexto a esquerda a fim de decidir que ação o reconhecedor deve tomar quando = aparece na entrada, tendo visto uma cadeia redutível para L.

Análise sintática

$$
\begin{aligned}
S &\rightarrow L = R \mid R \\
L &\rightarrow * R \mid id \\
R &\rightarrow L
\end{aligned}
$$

I_2: $S \rightarrow L \bullet = R$
$R \rightarrow L \bullet$

Análise ascendente – Análise LR

Os métodos LR canônico e o LALR conseguem reconhecer a gramática em questão, e terão sucesso com uma gama maior de gramáticas.

Porém, existem gramáticas não ambíguas p/ os quais todos os métodos de construção de analisadores LR produzirão tabelas de ação de análise de conflitos.

Felizmente, tais gramáticas geralmente podem ser evitadas em aplicações de linguagens de programação.

Análise sintática

Prefixos viáveis

Por que os autômatos LR(0) podem ser usados para guiar as decisões do tipo *shift-reduce*?

Os autômatos LR(0) p/ uma gramática caracteriza as cadeias de símbolos da gramática que podem aparecer na pilha de um analisador *shift-reduce* para a gramática.

Conteúda da pilha deve ser um prefixo de uma forma sentencial à direita.

Se a pilha contém α e o restante da entrada é x, então uma sequencia de reduções levará αx para S.

Em termos de derivações, $S \Rightarrow \alpha x$.

Análise sintática

Prefixos viáveis

Nem todos os prefixos das formas sentenciais à direita podem aparecer na pilha, pois o analisador sintático não deve avançar além do *handle*.

$$E \overset{*}{\underset{rm}{\Rightarrow}} F * id \underset{rm}{\Rightarrow} (E) * id$$

Em vários momentos durante o reconhecimento, a pilha terá (, (E, e (E), mas não deve conter (E)*, pois (E) é um *handle*, que o analisador sintático precisa reduzir para S antes de transferir o *.

Os prefixos que podem aparecer do lado direito de um forma sentencial são chamados de prefixos viáveis.

Análise sintática

Analisadores Sintáticos LR mais poderosos

A partir de agora, vamos trabalhar com analisadores LR incorporando nos itens o primeiro símbolo da entrada, ainda não lido.

Existem dois métodos LR com estas características:

1. O método "LR canônico", ou apenas "LR", faz uso do(s) símbolo(s) *lookahread.*

   Este método usa uma tabela construída a partir do conjunto de itens denominados itens LR(1).

Análise sintática

Analisadores Sintáticos LR mais poderosos

2. O método “Look Ahead LR”, ou “LALR”, cuja tabela é construída a partir dos conjuntos de itens LR(0), e possui muito menos estados que os analisadores típicos, baseados no conjunto canônico de itens LR(1).

   A incorporação de símbolos *lookahread* nos itens LR(0) introduz muito mais poder ao método e o torna mais geral, permitindo-lhe tratar muito mais gramáticas do que o método SLR.

   Além de o método LALR ser mais poderoso que o método SLR, suas tabelas ACTION e GOTO não são maiores que as tabelas ACTION e GOTO pata o SLR, o que torna o método perfeito na maioria das situações.

Análise sintática

Analisadores Sintáticos LR mais poderosos

Vimos a técnica mais genérica para construir uma tabela de análise LR a partir de uma gramatica.

No método SLR, o estado *i* faz uma redução segundo a produção A→α se no conjunto de itens I_i tiver o item [A→α•] e *a* estiver em FOLLOW (A).

Em alguns casos, quando o estado *i* aparece no topo da pilha, o prefixo viável *βα* na pilha é tal que *βA* não pode ser seguido por *a* em nenhuma forma sentencial à direita.

Assim, a redução por A → α deverá ser inválida na entrada *a*.

Análise sintática

Follow(S) = {$}
Follow(L) = {$, =}
Follow(R) = {$, =}

S → L = R | R
L → * R | id
R → L

Analisadores Sintáticos LR mais poderosos

Analisadores Sintáticos LR mais poderosos – Exemplo:

I_2: S → L• = R
R → L•

... no estado 2, tínhamos o item R → L•, que poderia corresponder à A→α anterior, e *a* poderia ser o sinal = que esta em FOLLOW(R).

Assim, o analisador SLR faz uma redução segundo a produção R→L no estado 2 com = como próxima entrada.

A ação transferir também é exigida, devido ao item R→L•=R no estado 2.

Não existe uma forma sentencial à direita na gramatica do exemplo que começa com R = ...

Análise sintática

S → L = R | R
L → * R | id
R → L

I_2: S → L• = R
R → L•

Análise ascendente – Análise LR

Analisadores Sintáticos LR mais poderosos – Exemplo:

Assim, o estado 2 correspondente somente ao prefixo viável L, não deve efetuar a redução de L para R.

É possível incorporar mais informações no estado para nos auxiliar na remoção de algumas dessas reduções inválidas por $A \rightarrow \alpha$.

Dividindo os estados quando necessário, podemos fazer com que cada estado de um analisador LR indique exatamente quais símbolos podem seguir um *handle* α, para qual o existe uma redução possível para A.

Análise sintática

S → L = R | R
L → * R | id
R → L

I_2: S → L• = R
R → L•

Análise ascendente – Análise LR

Analisadores Sintáticos LR mais poderosos – Exemplo:

A informação extra é incorporada ao estado redefinindo-se os itens para incluir um símbolo terminal como um segundo componente.

A forma geral de um item agora é um par [A → α • β, a], onde o primeiro componente A → αβ é uma produção, e *a* é um terminal ou o marcador do fim da entrada, *$*.

Chamamos esse objeto de um item LR(1).

O número 1 refere-se ao tamanho do segundo componente, denominado *lookahead* do item.

Análise sintática

S → L = R | R
L → * R | id
R → L

I_2: S → L• = R
R → L•

Análise ascendente – Análise LR

Analisadores Sintáticos LR mais poderosos – Exemplo:

O *lookahead* não tem efeito algum em um item da forma [A→α• β, a], onde β não é ε, mas um item na forma [A→α •, a] requer uma redução segundo a produção A → α somente se o próximo símbolo da entrada for a.

Assim, somos forçados a reduzir segundo A → α somente sob aqueles símbolos de entrada *a* para os quais [A→α •, a] é um item LR(1) no estado do topo da pilha.

O conjunto desses símbolos *a* é um subconjunto de FOLLOW(A), mas poderia ser um subconjunto próprio.

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos

Formalmente, dizemos que o item LR(1) [A → α • β, a] é válido para um prefixo viável γ se houver uma derivação $S \overset{*}{\underset{rm}{\Rightarrow}} \delta Aw \underset{rm}{\Rightarrow} \delta\alpha\beta w$, onde:

1. γ = δα, e
2. Ou *a* é o primeiro símbolo de *w*, ou *w* é ϵ e *a* é $.

Exemplo: Vamos considerar a gramática:

S → BB
B → aB I b

Análise sintática

S → BB
B → aB I b

Análise ascendente – Análise LR

LR mais poderosos

Existe uma derivação mais à direita $S \overset{*}{\underset{rm}{\Rightarrow}} aaBab \underset{rm}{\Rightarrow} aaaBab$.

Vemos que o item [B → a • B, a] é válido para um prefixo viável γ = aaa permitindo δ = aa, A=B, w = ab, α = a, e β = B na definição anterior.

Há também uma derivação mais à direita $S \overset{*}{\underset{rm}{\Rightarrow}} BaB \underset{rm}{\Rightarrow} BaaB$.

Por essa derivação, vemos que o item [B→a• B, $] é válido para o prefixo viável Baa.

Análise sintática

S → BB
B → aB I b

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1)

O método para construir a coleção canônica de conjuntos de itens LR(1) válidos é basicamente o mesmo daquele para a construção da coleção canônica de conjuntos de itens LR(0).

Só precisamos modificar dois procedimentos CLOSURE e GOTO.

Análise sintática

S → BB
B → aB I b

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) para G'

```
SetOfItems CLOSURE(1){

        repeat
                for (cada item [A → α • Bβ, a] em I)
                        for (cada produção B → γ em G')
                                for (cada terminal b em FIRST(βa))
                                        adicione [B → • γ, b] no conjunto I;
        until não conseguir adicionar mais itens em I;
        return I;
}
```

Análise sintática

S → BB
B → aB I b

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) para G'

```
SetOfItems GOTO(I,X){

        Inicializa J para ser o conjunto vazio;
                for (cada item [A → α• Xβ, a] em I)
                        Adicione item [A →  αX• β, a] ao conjunto J;
                return CLOSURE(J);
}
```

Análise sintática

S → BB
B → aB I b

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) para G’

```
void items(G’){

        inicializa C como CLOSURE({[S’ → •S, $]});
        repeat
                for (cada conjunto de itens I em C)
                        for (cada símbolo X da gramatica)
                                if (GOTO(I, X) não é vazio e não esta em C)
                                        adicione GOTO(I, X) em C;
        until não haja mais conjuntos de itens para serem incluídos em C;
}
```

Análise sintática

S → BB
B → aB I b

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1)

P/ apreciar a nova definição da função CLOSURE, em particular, porque *b* precisa estar em FIRST(βa), considere um item da forma [A → α • Bβ, a] no conjunto de itens válidos para algum prefixo viável γ.

Então, existe uma derivação mais à direita $S \overset{*}{\underset{rm}{\Rightarrow}} \delta Aax \underset{rm}{\Rightarrow} \delta\alpha B\beta ax$, onde γ=δα.

Suponha que βx derive a cadeia terminais by.

Então, cada produção da forma B → η para algum η, temos a derivação $S \overset{*}{\underset{rm}{\Rightarrow}} \gamma Bby \underset{rm}{\Rightarrow} \gamma\eta by$. Assim, [B→ •η, b] é válido para γ.

Análise sintática

S → BB
B → aB I b

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1)

Observe que b pode ser o primeiro terminal derivado de β, ou é possível que β derive ε na derivação $\beta ax \overset{*}{\underset{rm}{\Rightarrow}} by$, e b possa, portanto, ser a.

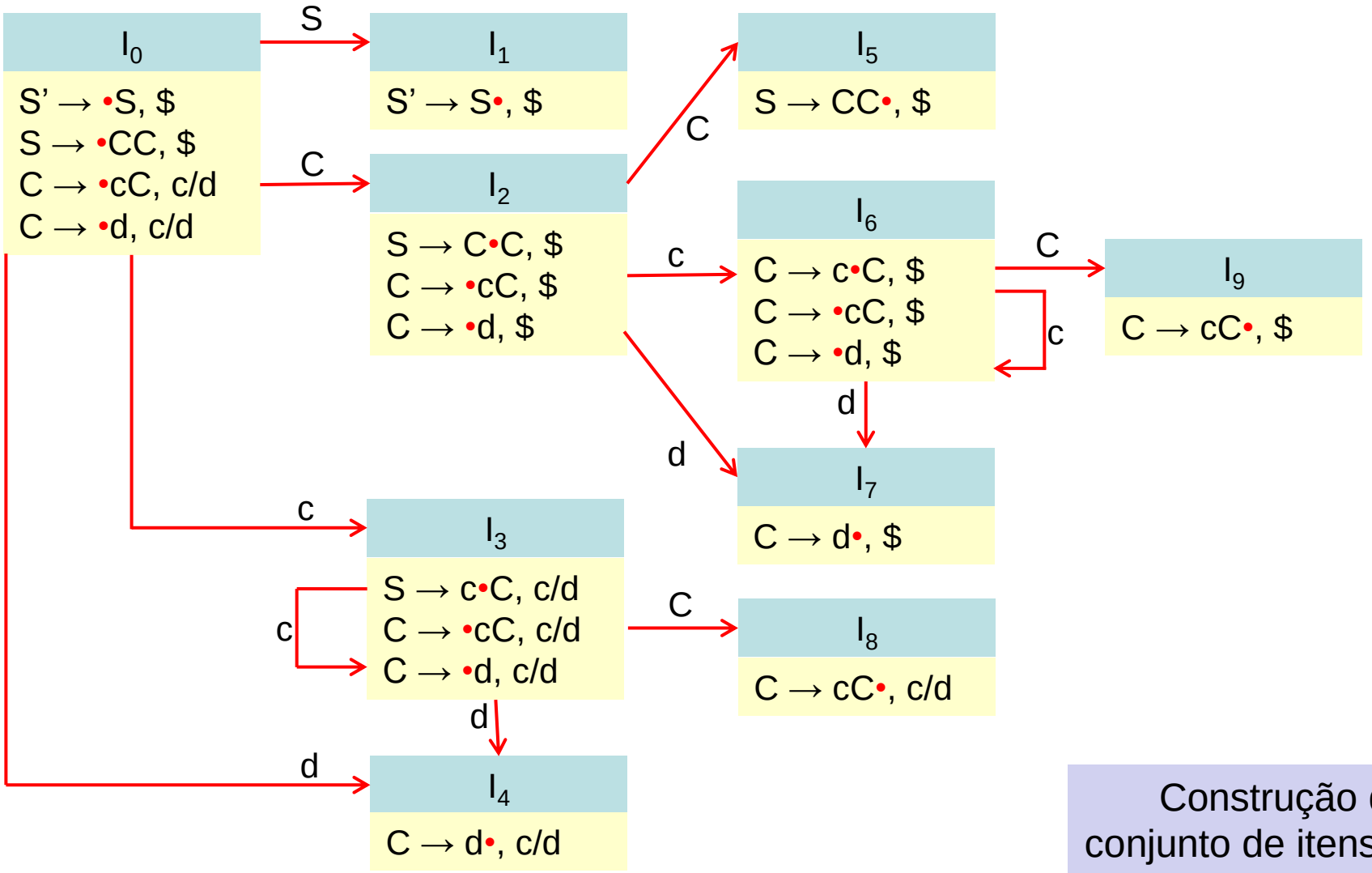
Para resumir as duas possiblidades, dizemos que b pode ser qualquer terminal em FIRST(βax).

Portanto, observe que x não pode ser o primeiro terminal de by, de modo que FIRST(βax)=FIRST(βa).

Agora, mostramos a construção dos seguintes itens LR(1) ...

Análise sintática

S’ → S
S → CC
C → cC | d



Construção de conjunto de itens LR(1)

Análise sintática

S’ → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) - Algoritmo

Entrada: Uma gramática estendida G’
Saída: Os conjuntos de itens LR(1) que são conjuntos de itens válidos para um ou mais prefixos viáveis de G’.
Método: As funções CLOSURE e GOTO e a rotina principal itens para construir os conjuntos de itens.

Iniciamos calculando o fechamento de {[S’ → •S, $]}.

Para fazer o fechamento, casamos o item [S’→•S] com o item [A→α•Bβ, a] na função CLOSURE.

Ou seja, A = S’, α = ε, B = S, β = ε, e a = $.

I_0
S’ → •S, $
S → •CC, $
C → •cC, c/d
C → •d, c/d

Análise sintática

S’ → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) - Algoritmo

A função CLOSURE nos diz para adicionar [B→ •γ, b] para cada produção B → γ e terminal b em FIRST(βa).

Em termos da presente gramática, B → γ deve ser S → CC, e como β é ε e a é $, b só pode ser $.

Assim, acrescentamos [S → • CC, $] ao estado I_0.

I_0
S’ → •S, $
S → •CC, $
C → •cC, c/d
C → •d, c/d

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) - Algoritmo

Continuamos a calcular o fechamento incluindo todos os itens [C → •γ, b] para b em FIRST(C$).

Ou seja, casando [S → •CC, $] com [A → α•Bβ, a], temos A=S, α=ε, B=C, β=C, e a=$.

Como C não deriva a cadeia vazia, FIRST(C$) = FIRST(C).

First(S') = {c, d}	Follow(S') = {$}
First(S) = {c, d}	Follow(S) = {$}
First(C) = {c, d}	Follow(C) = {c,d,$}

Análise sintática

First(S') = {c, d}
First(S) = {c, d}
First(C) = {c, d}

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) - Algoritmo

Como FIRS(C) contém os terminais c e d, acrescentamos os seguintes itens [C → • cC, c], [C → • cC, d], [C → • d, c] e [C → • d, d] no estado I_0.

Nenhum dos novos itens possui um não-terminal imediatamente à direita do ponto, de modo que completamos o primeiro conjunto de itens LR(1).

O conjunto inicial de itens é dado pelo estado I_0 a seguir.

I_0
S' → •S, $
S → •CC, $
C → •cC, c/d
C → •d, c/d

Análise sintática

S’ → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) - Algoritmo

Agora, calculamos GOTO(I_0, X) para os diversos valores de X.

Para X = S, temos de fazer o fechamento do item [S’ → S•, $].

Nenhum fechamento adicional é possível, pois o ponto “•” aparece na extremidade direita.

Assim, temos o próximo conjunto de itens representado no estado I_1.

I_1
S’ → S•, $

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) - Algoritmo

Para X = C, fechamos [S → C•C, $].

Apresentamos as produções-C com o segundo componente $ e, como não há mais nenhum item para ser incluído, o estado I_2 é dado por:

I_2
S → C•C, $
C → •cC, $
C → •d, $

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) - Algoritmo

Em seguida, considere X = C.

Temos de fazer o fechamento de {[C → •cC, c/d}].

Então, acrescentamos as produções-C com o segundo componente c/d, produzindo o estado I_3.

I_3
S → c•C, c/d
C → •cC, c/d
C → •d, c/d

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) - Algoritmo

Finalmente, considere X = d, com o segundo componente c/d, produzimos o conjunto de itens:

I_4
C → d•, c/d

Terminamos considerando o GOTO sob I_1.

Não obtemos novos conjuntos de I_1, mas I_2 possui transições sob C, c e d. Para GOTO(I2, C), temos:

I_5
S → CC•, $

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) - Algoritmo

Nenhum fechamento é necessário.

Para calcular GOTO(I_2, c), consideramos o fechamento de {[C → c•C, $}], para obter:

I_6
C → c•C, $
C → •cC, $
C → •d, $

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) - Algoritmo

Observe que I_6 difere de I_3 somente nos segundos componentes.

Veremos que é comum que vários conjuntos de itens LR(1) para uma gramatica tenham os mesmos primeiros componentes e difiram em seus segundos componentes.

I_3
S → c•C, **c/d**
C → •cC, **c/d**
C → •d, **c/d**

I_6
C → c•C, **$**
C → •cC, **$**
C → •d, **$**

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) - Algoritmo

Continuar com a função GOTO para I_2, GOTO(I_2, d) é visto como sendo:

I_7
C → d•, $

Então, passando agora para I_3, os GOTOs de I_3 sob c e d são I_3 e I_4, respectivamente, e GOTO(I_3, C):

I_8
C → cC•, c/d

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Construindo conjuntos de itens LR(1) - Algoritmo

I_4 e I_5 não possuem GOTOs, pois todos os itens têm seus pontos mais à direita de seu lado direito.

Os GOTOs de I_6 sob c e d são I_6 e I_7, respectivamente, e GOTO(I_6, C) é:

I_9
C → cC•, $

Por fim, os conjuntos de itens restantes não geram GOTOs, de modo que terminamos.

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos – Tabelas LR(1) canônicas de análise - Algoritmo

Esta seção apresenta as regras para construir as funções ACTION e GOTO, a partir dos conjuntos de itens LR(1) para os reconhecimentos LR(1).

Essas funções são representadas por uma tabela, como antes.

A única diferença está nos valores das entradas.

Entrada: Uma gramática estendida G'
Saída: As funções ACTION e GOTO da tabela LR canônica de análise para G'.
Método:

1. Construa C'={I_0, I_1,..., I_n}, a coleção de conjuntos de itens LR(1) para G'.

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos – Tabelas LR(1) canônicas de análise - Algoritmo

Método: ...

2. O estado i do analisador sintático é construído a partir de I_i.
A ação de análise do reconhecedor para o estado i é determinada:

a) Se [A → α•aβ, b] está em I_i e GOTO(I_i, a) = I_j, então defina ACTION[i, a] como "*shift j*". a deve ser um terminal.

b) Se [A → α•, a] estiver em I_i, A ≠ S', então defina ACTION[i, a] como "*reduce A → α*".

c) Se [S'→ S•, $] estiver em I_i, então defina ACTION[i, $] como "*accept*". Se quaisquer ações de conflito resultarem das regras anteriores, dizemos que a gramatica não é LR(1), e o algoritmo falha nesse caso.

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos – Tabelas LR(1) canônicas de análise - Algoritmo

Método: ...

3. As funções de transições para o estado i são construídas para todos os não-terminais A usando a regra: se GOTO(I_i, A) = I_j, então GOTO[i, A] = j.

4. Todas as entradas da tabela ACTION e GOTO não definidas pelas regras (2) e (3) são entradas de "error".

5. O estado inicial do analisador sintático corresponde ao conjunto de itens LR(1) que contém o item [S' → •S, $].

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos – Tabelas LR(1) canônicas de análise - Algoritmo

A tabela de reconhecimento sintático construído a partir das funções de ação e de transição produzida pelo algoritmo de construção dos conjuntos de itens LR(1) é chamada de tabela de análise LR(1) canônica.

Um analisador LR usando essa tabela é chamado de analisador sintático LR(1) canônico.

Se a função de ação de análise não possuir múltiplas entradas definidas, então a gramatica analisada é chamada de gramatica LR(1).

Como no LR(0), podemos omitir o "(1)" se ele for entendido.

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas LR(1) canônicas de análise - Algoritmo

A tabela de análise canônica LR(1) para a gramatica estudada é ilustrada ao lado:

As produções 1, 2 e 3 são:

S → CC,
C → cC e
C → d, respectivamente.

Estado	ACTION			GOTO	
	c	d	$	S	C
0	s3	s4		1	2
1			**acc**		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Apresentamos nesta seção o nosso último método para a construção de um analisador sintático, a técnica LALR (*Lookahead LR*).

Esse é o método frequentemente usado na prática, pois suas tabelas são muito menores que as tabelas LR canônicas, e ainda as construções sintáticas mais comuns das linguagens de programação podem ser expressas convenientemente por uma gramatica LALR.

O mesmo é quase verdadeiro para gramaticas SLR, mas existem algumas construções que não podem ser reconhecidas pelas técnicas SLR.

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Para fazer uma comparação do tamanho das tabelas dos analisadores, os métodos SLR e LALR para uma dada gramatica sempre têm o mesmo número de estados, e esse número tipicamente é de várias centenas de estados para uma linguagem como **C**.

A tabela LR canônica tipicamente tem vários **milhares** de estados para uma linguagem de mesmo tamanho.

Assim, é muito mais fácil e mais econômico construir tabelas SLR e LALR do que as tabelas LR canônicas.

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

S' → S
S → CC
C → cC | d

Como uma introdução, vamos novamente considerar a G':

Considere um par de estados de aparência semelhante, como I_4 e I_7.

I_4
C → d•, c/d

I_7
C → d•, $

Cada um desses estados possui só um item cujo primeiro componente é C → d.

Em I_4, os *lookaheads* são c ou d, em I_7, $ é o único *lookahead*.

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

I_4	I_7
C → d•, c/d	C → d•, $

LR mais poderosos – Tabelas de análise LALR

Para ver a diferença entre os papéis de I_4 e I_7 no analisador sintático, observe que a gramatica gera a linguagem regular c*dc*d.

Ao ler uma entrada cc...Cdcc...cd, o analisador transfere o primeiro grupo de cs seguido de um d para a pilha, entrando no estado 4 após ler o d.

O analisador, então, requer uma redução segundo a produção C→d, desde que o próximo símbolo da entrada seja c ou d.

A exigência de que c ou d venha em seguida faz sentido, pois esses são os símbolos que poderiam iniciar as cadeias em c*d.

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

I_4	I_7
C → d•, c/d	C → d•, $

LR mais poderosos – Tabelas de análise LALR

Se $ vier após o primeiro d, temos uma entrada como ccd, que não pertence à linguagem, e o estado 4 declara corretamente um erro se $ for a próxima entrada.

O analisador sintático entra no estado 7 depois de ler o segundo d.

Então, o analisador dever ler $ na entrada, para ele reconhecer uma cadeia no formato c*dc*c.

Assim, faz sentido que o estado 7 reduza segundo a produção C → d sob a entrada $ e declare erro nas entradas c ou d.

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

I_4	I_7
C → d•, c/d	C → d•, $

LR mais poderosos – Tabelas de análise LALR

Vamos agora substituir I_4 e I_7 por um novo estado I_{47}, a união de I_4 e I_7, consistindo no conjunto de três itens representados por [C → d•, c/d/$].

As transições sob d para I_4 ou I_7 a partir de I_0, I_2, I_3 e I_6 agora são dirigidas para I_{47}.

A ação do estado 47 é reduzir para qualquer entrada.

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

I_4	I_7
C → d•, c/d	C → d•, $

LR mais poderosos – Tabelas de análise LALR

O analisador sintático revisado comporta-se essencialmente como o original, embora possa reduzir d para C em circunstâncias nas quais o original declararia erro, por exemplo, para entrada como ccd ou cdcd.

O erro eventualmente será detectado;

Na verdade, independente do método usado, o erro será detectado antes que mais símbolos de entrada sejam transferidos para a pilha.

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

I_4	I_7
C → d•, c/d	C → d•, $

LR mais poderosos – Tabelas de análise LALR

Generalizando, procuramos por conjuntos de itens LR(1) com o mesmo núcleo, ou seja, conjuntos onde os primeiros componentes do par sejam iguais, e os juntamos em um novo conjunto de itens.

Por ex., I_4 e I_7 possuem núcleos iguais {C → d•}, portanto formam tal par.

I_3 e I_6 formam outro par, com o núcleo {C → c•C, C → •cC, C → •d}.

I_3
S → c•C, c/d
C → •cC, c/d
C → •d, c/d

I_6
C → c•C, $
C → •cC, $
C → •d, $

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Existe mais um par, I_8 e I_9, com o núcleo comum {C → cC•}.

I_8
C → cC•, c/d

I_9
C → cC•, $

Observe que, em geral, um núcleo é um conjunto de itens LR(0) para a gramatica dada, e que uma gramatica LR(1) pode produzir mais de dois conjuntos de itens com o mesmo núcleo.

Análise sintática

S’ → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Como o núcleo de GOTO(I, X) depende somente do núcleo de I, as transições dos conjuntos unidos podem também ser unidas.

Assim, à medida que unimos os conjuntos de itens, revisamos também a função de transição.

As funções de ação são modificadas para refletir as ações de erro em todos os conjuntos de itens unidos.

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Vamos supor que tenha uma gramatica LR(1), ou seja, uma gramatica cujos os conjuntos de itens LR(1) não produzam conflitos em sua ação de análise.

Se unirmos todos os estados com o mesmo núcleo, é possível que a fusão resultante apresente conflitos, mas isso é impossível pelo seguinte:

> Suponha que, após a união, exista um conflito no *lookahead* a porque há item [A → α•, a] exigindo uma redução segundo a produção A → α, e exista também outro item [B → β•aγ, b] exigindo um empilhamento.

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Então, algum conjunto de itens que formou a união possui o item [A → α•, a], e como os núcleos de todos esses estados são iguais, ele também tem de possuir um item [B → β•aγ, c] para algum c.

Portanto, o *conflito shift/reduce* sob a já existia, e a gramatica não é LR(1), conforme assumimos.

Assim, a união de estados com núcleos comuns nunca pode gerar *conflitos shift/reduce* que não estavam presentes em um de seus estados originais, porque as ações de transferências dependem apenas do núcleo, e não do *lookahead*.

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

É possível, contudo, que uma fusão produza *conflito reduce/reduce*, como mostra o exemplo a seguir.

S' → S
S → aAd I bBd I aBe I bAe
A → C
B → C

A qual gera as quatro cadeias acd, ace, bcd e bce.

A gramatica é LR(1) por construir os conjuntos de itens LR(1).

Análise sintática

S' → S
S → aAd I bBd I aBe I bAe
A → C
B → C

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Após sua construção, descobrimos que o conjunto de itens {[A → c•, d], [B→ c•, e]} é válido para o reflexo viável ac e que {[A → c•, e], [B → c•, d]} é válido para o prefixo viável bc.

Nenhum desses conjuntos possui conflitos e seus núcleos são iguais. Contudo, a sua união...

A → c•, d/e
B → c•, d/e

gera um *conflito reduce/reduce*, pois as reduções segundo as produções A → c e B → c exigem as entradas d e e.

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Agora, estamos preparados para entender o primeiro dos dois algoritmos usados na construção da tabela LALR.

A ideia geral é construir os conjuntos com núcleos comuns. Desta forma, construímos a nova tabela de análise a partir da coleção de conjuntos de itens unidos.

O método que estamos prestes a descrever serve principalmente como uma definição das gramaticas LALR(1).

A construção de toda a coleção de conjunto de item LR(1) requer espaço e tempo considerável para ser usada na prática.

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR - Algoritmo

Entrada: Uma gramática estendida G'
Saída: As funções ACTION e GOTO da tabela LALR de análise para G'.
Método: 1. Construa C = { I_0, I_1, ... , I_n}, a coleção de conjuntos de itens LR(1).

2. Para todos os núcleos presentes em conjuntos de itens LR(1), determine aqueles conjuntos que tenham o mesmo núcleo, e os substituta pela sua união.

3. Considere que C' = {J_0, J_1, ..., J_m} seja o conjunto de itens LR(1) resultante. As ações de análise para o estado i são construídas a partir de J_i, da mesma forma que o algoritmo para LR(1). Se houver um conflito de ação de análise, o algoritmo deixa de produzir um reconhecedor sintático, e a gramatica é considerada como não sendo LALR(1).

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR - Algoritmo

Entrada: Uma gramática estendida G'
Saída: As funções ACTION e GOTO da tabela LALR de análise para G'.
Método: 4. A tabela GOTO é construída da forma a seguir.

Se J é união de um ou mais conjuntos de itens LR(1), ou seja, $J = I_1 \cap I_2 \cap ... I_K$, então os núcleos de $GOTO(I_1, X)$, ..., $GOTO(I_K, X)$ são os mesmos, desde que I_1, I_2, ..., I_K possuam todos o mesmo núcleo.

Considere que K seja a união de todos os conjuntos de itens tendo o mesmo núcleo que $GOTO(I_1, X)$.

Então, GOTO(J, X) = K.

Análise sintática

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

A tabela produzida algoritmo anterior é denominada tabela de análise LALR para G.

Se não houver conflito de ação de análise, então a gramatica dada é considerada uma gramatica LALR(1).

A coleção de conjuntos de itens construídos na etapa (3) é chamada de coleção LALR(1).

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Voltando a gramática usada como exemplo inicial (introduzida em LR(1)), e conforme mencionado, existem três pares de conjuntos de itens que podem ser unidos.

I_3 e I_6 são substituídos por sua união:

I_3
S → c•C, **c/d**
C → •cC, **c/d**
C → •d, **c/d**

∩

I_6
C → c•C, **$**
C → •cC, **$**
C → •d, **$**

=

I_{36}: C → c•C, c/d/$
C → •cC, c/d/$
C → •d, c/d/$

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

I_4 e I_7 são substituídos por sua união:

I_4
C → d•, c/d

∩

I_7
C → d•, $

=

I_{47}: C → d•, c/d/$

e I_8 e I_9 são substituídos por sua união:

I_8
C → cC•, c/d

∩

I_9
C → cC•, $

=

I_{89}: C → cC•, c/d/$

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas LALR(1) de análise - Algoritmo

As funções de ação e transição LALR para os conjuntos de itens unidos aparecem na Tabela a seguir.

Estado	ACTION			GOTO	
	c	d	$	S	C
0	s36	s47		1	2
1			**acc**		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r3	r3	r3		

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Para verificar como as funções de transição são calculadas, considere GOTO(I_{36}, C).

No conjunto original de itens LR(1), GOTO(I_3, C) = I_8, e I_8 agora faz parte de I_{89}, de modo que fazemos com que GOTO(I_{36}, C) seja I_{89}.

Poderíamos ter chegado a essa mesma conclusão se considerássemos I_6, a outra parte de I_{36}.

Ou seja, GOTO(I_6, C) = I_9, e I_9 agora faz parte de I_{89}.

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Ou seja, GOTO(I_6, C) = I_9, e I_9 agora faz parte de I_{89}.

Como outro exemplo, considere GOTO(I_2, c), uma entrada que é processada após a ação de transferência de I_2 sob a entrada c.

Nos conjuntos originais de itens LR(1), GOTO(I_2, c) = I_6.

Como I_6 agora faz parte de I_{36}, GOTO(I_2, c) torna-se I_{36}.

Assim, a entrada na Tabela para o estado 2 e a entrada c é feita s36, significando transferir e empilhar o estado 36 na pilha.

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Dada uma cadeia da linguagem c*dc*d, tanto o analisador sintático LR da quanto o analisador sintático LALR efetuam exatamente a mesma sequência de transferências e reduções, embora os nomes dos estados na pilha possam ser diferentes.

Por exemplo, se o analisador sintático LR coloca I_3 ou I_6 na pilha, o analisador sintático LALR empilhará I_{36}.

Esse relacionamento em geral se mantém para uma gramática LALR. Os analisadores LR e LALR imitarão um ao outro nas entradas corretas.

Análise sintática

S' → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

Dada uma entrada errônea, o analisador LALR pode prosseguir efetuando algumas reduções na pilha após o analisador LR ter declarado um erro. Porém, o analisador sintático LALR nunca transferirá um novo símbolo após o analisador LR declarar um erro.

Por exemplo, na entrada ccd seguida por $, o analisador LR colocará na pilha os estados: **0 3 3 4**.

No estado 4 descobre-se um erro, pois $ é o próximo símbolo da entrada e este estado possui uma ação de erro sob $. Ao contrário, o analisador LALR fará os movimentos, colocando na pilha os estados: **0 36 36 47**.

Análise sintática

S’ → S
S → CC
C → cC | d

Análise ascendente – Análise LR

LR mais poderosos – Tabelas de análise LALR

O estado 47 sob a entrada $ possui uma ação de redução segundo a produção C → d.

Agora, a ação do estado 89 sob a entrada $ é reduzir segundo a produção C → cC. Portanto a pilha torna-se: **0 36 89**.

Uma redução semelhante é efetuada. Após esta redução, a pilha passa a ter a configuração: **0 2**.

Finalmente, o estado 2 possui uma ação de erro sob a entrada $, de modo que o erro agora é descoberto.

COMPILADORES

Obrigado!!

Prof. Geovane Griesang
geovanegriesang@unisc.br