Universidade de Santa Cruz do Sul – UNISC
Departamento de informática

COMPILADORES

Análise sintática

Parte 05

Prof. Geovane Griesang
geovanegriesang@unisc.br

Análise sintática

Data	Conteúdo
23/09/2013	3. Análise Sintática: 3.1 analisadores ascendentes e descendentes;
30/09/2013	3. Análise Sintática: 3.2 análise preditiva;
07/10/2013	3. Análise Sintática: 3.3 análise de precedência de operadores;
14/10/2013	3. Análise Sintática: 3.4 análise LR;
21/10/2013	3. Análise Sintática: 3.5 recuperação de erros;
28/10/2013	3. Análise Sintática: 3.6 geradores de analisadores sintáticos;
04/11/2013	Atividade EaD. 3. Análise Sintática: continuação.
11/11/2013	Prova Individual e sem consulta 2.

Análise sintática

Legenda:

$\sum$ = sigma (somatório)
δ = delta
ε = épsilon
λ = lambda
α = alfa
β = beta
γ = gamma
ξ = xi

Análise sintática

Análise ascendente

Análise LR: *Left to Right with Rightmost derivation*

Atualmente, o tipo mais usado de analisadores sintático ascendentes é baseado em um conceito chamado LR(K).

L representa a escanção de entrada da esquerda para direita.

R representa a construções mais à direita ao reverso.

k representa os símbolos a frente do fluxo de entrada que auxiliam nas decisões de análise.

Análise sintática

Análise ascendente

Análise LR

Os analisadores sintáticos LR são controlados por uma tabela, de modo muito parecido com os analisadores LL não-recursivos.

Para uma gramática ser LR, é suficiente que uma analisador *shift-reduce* seja capaz de reconhecer o *handle* quando ele aparecer no topo da pilha.

Análise sintática

Análise ascendente

Análise LR

O uso desse tipo de analisador é atraente por vários motivos:

- Os analisadores LR são capazes de reconhecer praticamente todas as construções sintáticas definidas por Gramáticas Livre de Contexto (GLC) da maioria das linguagens de programação.

Existem gramáticas que não são LR, mas geralmente elas podem ser evitadas para construções típicas das ling. de programação.

Análise sintática

Análise ascendente

Análise LR

...

- O LR, além de ser o método de análise *shift-reduce* sem retrocesso mais geral, pode ser implementado com o mesmo grau de eficiência, espaço e tamanho que outro métodos *shift-reduce*.

- Um analisador LR detecta um erro sintático tão logo ele aparece na cadeia de entrada em uma escansão da entrada da esquerda para direita.

Análise sintática

Análise ascendente

Análise LR

...

- A classe de gramáticas que podem ser reconhecidas com o uso dos métodos LR é um superconjunto próprio da classe de gramáticas que podem ser reconhecidas com os métodos preditivos e LL.

  Para uma gramática ser LR(k), ela deve ser capaz de reconhecer a ocorrência do lado direito de uma produção em forma sentencial mais à direita, com k símbolos a frente da entrada.

  Esse requisito é menos rigoroso que aquele das gramáticas LL(k).

Análise sintática

Análise ascendente

Análise LR

...

Para uma gramática ser LR(k), ela deve ser capaz de reconhecer a ocorrência do lado direito de uma produção em forma sentencial mais à direita, com k símbolos a frente da entrada.

... Relembrando ...

Na LL(k), para reconhecer o uso de uma produção, o analisador vê apenas os k primeiros símbolos que o seu lado direito deriva.

Assim, não é surpresa que as gramáticas LR possam descrever mais linguagens do que as gramáticas LL.

Análise sintática

Análise ascendente

Análise LR

A principal desvantagem do método LR está relacionada com a geração do analisador: sua construção a mão, p/ uma linguagem de programação típica é muito trabalhosa.

Portanto, é necessário o uso de uma ferramenta especializada, um gerador de analisador LR.

Felizmente, muitos desses geradores estão disponíveis, como por exemplo, o Yacc. Esse gerador recebe como entrada um GLC e produz automaticamente como saída um analisador sintático para esta gramática.

Análise sintática

Entrada: id * id

E	→	E + T | T
T	→	T * F | F
F	→	(E) | id

id * id

F → id, F * id

T → F, T * id

E → T, E * id ou F → id, T * F

Análise ascendente

Reduções - Exemplo

A derivação segue a seguinte sequencia de cadeias:

id * id, F * id, T * id, T * F, T, E

Neste ponto temos duas opções de redução:

podemos reduzir a cadeia T para E, segundo a produção E → T, e a cadeia consistindo no segundo id, que representa o lado direito de F → id.

Escolhemos a segunda opção e reduzimos id para F, produzindo a cadeia T * F.

Entrada: id * id

E	→	E + T | T
T	→	T * F | F
F	→	(E) | id

Análise sintática

Análise ascendente

Análise LR – Itens e o autômato LR

Como o analisador sintático *shift-reduce* sabe quando transferir p/ a pilha e quando reduzir?

Ex.: Dado o conteúdo de pilha $T e o próximo símbolo da entrada *, como o analisador sabe que T no topo da pilha não é um *handle*, de modo que a ação apropriada seja transferir o "*" para a pilha e não reduzir T para E?

		id * id
		F → id, F * id
		T → F, T * id
E → T, E * id	ou	F → id, T * F

Análise sintática

Análise ascendente

Análise LR – Itens e o autômato LR

Um analisador LR decide sobre as ações *shift-reduce* mantendo estados para acompanhar onde se encontra a análise.

Os estados representam conjuntos de "*itens*".

Um item LR(0) de uma gramática G é a produção de G com um ponto em alguma posição do seu lado direito.

A produção A → XYZ gera os quatro itens:

A → ▪XYZ

A → X▪YZ

A → XY▪Z

A → XYZ▪

A produção A → ε gera apenas um item: A → ▪

Análise sintática

$A \rightarrow \cdot XYZ$

$A \rightarrow X \cdot YZ$

$A \rightarrow XY \cdot Z$

$A \rightarrow XYZ \cdot$

Análise ascendente

Análise LR - Itens e o autômato LR: Representação dos conjuntos de itens

Um gerador de analisadores sintáticos que produz um analisador ascendente deve representar os itens e conjunto de itens convenientemente.

Um item pode ser representado por um par de inteiros, onde o primeiro representa o número de uma das produções da gramática subjacente, e o segundo indica a posição do ponto.

Conjuntos de itens podem ser representados por uma lista desses pares.

Análise sintática

$A \rightarrow \cdot XYZ$

$A \rightarrow X \cdot YZ$

$A \rightarrow XY \cdot Z$

$A \rightarrow XYZ \cdot$

Análise ascendente

Análise LR - Itens e o autômato LR: Representação dos conjuntos de itens

O conjunto de itens necessários frequentemente incluem itens de "fechamento" ("*closure*"), onde o ponto está no início do corpo.

Esses itens sempre podem ser reconstruídos a partir dos outros itens no conjunto, e não temos de incluí-los na lista.

Análise sintática

A → ▪XYZ

A → X▪YZ

A → XY▪Z

A → XYZ▪

Análise ascendente

Análise LR – Itens e o autômato LR

Um item indica quanto de uma produção já foi visto em determinado ponto no processo de reconhecimento sintático. Ex.:

O item A→▪XYZ indica o início da busca de uma cadeia derivável de XYZ na entrada.

O item A → X▪YZ indica que no ponto atual onde se encontra a análise, uma cadeia X já foi encontrada, e que esperamos em seguida ver uma cadeia derivável de YZ.

O item A → XYZ▪ indica o fim da busca, ou seja, já derivamos o lado direito XYZ de A e que pode ser o momento de reduzir XYZ para A.

Análise sintática

Análise ascendente

Análise LR – Itens e o autômato LR

Uma coleção de conj. de itens LR(0), chamada *coleção LR(0) canônica*, oferece a base para construção de um autômato finito determinista, que é usado para dirigir as decisões durante a análise.

Esse autômato é chamado de *autômato LR(0)*.

Em particular, cada estado do autômato LR(0) representa um *conjunto de itens na coleção LR(0) canônica*.

Análise sintática

Análise ascendente

Exemplo: Para construir a *coleção LR(0) canônica* para uma gramática, definimos uma gramática estendida e duas funções:

função de fechamento (*function* CLOSURE).

função de transição (function GOTO).

Se G é uma gramática com símbolo inicial S, então G', é uma gramática *estendida* para G com o novo símbolo inicial S' e a produção S' → S.

O objetivo dessa nova produção é simplificar a identificação de quando o reconhecedor sintático deve parar e anunciar a aceitação da cadeia de entrada.

Ou seja, a aceitação ocorre se e somente se o analisador estiver para reduzir por S' → S.

Análise sintática

Análise ascendente

Análise LR – Fechamento de conjunto de itens

Se I é um conj. de itens para a gramática G, então CLOSURE(I) é o conj. de itens construídos a partir de I pelas duas regras:

1. Inicialmente, acrescente todo item de I no CLOSURE(I).

2. Se $A \rightarrow \alpha \cdot B\beta$ está em CLOSEURE(I) e $B \rightarrow \gamma$ é uma produção, então adicione o item $B \rightarrow \cdot\gamma$ em CLOSURE(I), se ele ainda não está lá.

I_x

$A \rightarrow \alpha \cdot B\beta$

$B \rightarrow \cdot\gamma$

Análise sintática

I_x

$A \rightarrow \alpha \cdot B\beta$

$B \rightarrow \cdot\gamma$

Análise ascendente

Análise LR – Fechamento de conjunto de itens

Intuitivamente, $A \rightarrow \alpha \cdot B\beta$ em CLOSURE(I) indica que, em algum ponto no processo de reconhecimento, os itens correspondentes as produções-B devem ser acrescentados para dirigir a busca por B.

A subcadeia derivável de $B\beta$ na entrada terá um prefixo derivável de B aplicando umas das produções-B.

Portanto, incluímos itens para todas as produções-B; ou seja, se $B \rightarrow \gamma$ é uma produção, também incluímos $B \rightarrow \cdot\gamma$ em CLOSURE(I), este processo é conhecido como fechamento de estado.

Análise sintática

Análise ascendente

Análise LR – Fechamento de conjunto de itens

Exemplo:

E’ → E
E → E + T | T
T → T * F | F
F → (E) | id

P/ ver como o fechamento de um estado é computado, E’ → ·E é colocado em CLOSURE(I) pela regra (1).

Acrescente todo item de I no CLOSURE(I).

I_0

E’ → ·E

Análise sintática

Análise ascendente

Análise LR – Fechamento de conjunto de itens

Exemplo:

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

I_0

E' → ·E

E → ·E + T
E → ·T

Como existe um não-terminal E à direita de um ponto, acrescentamos as produções-E com pontos mais à esquerda do lado direito dos itens, ex.:

E → ·E + T e E → ·T.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

Análise LR – Fechamento de conjunto de itens

Exemplo:

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

I_0

E' → ·E

E → ·E + T
E → ·T
T → ·T * F
T → ·F

... Agora, existe um não-terminal T imediatamente à direita de um ponto no segundo item incluído (E → ·T), de modo que acrescentamos as produções-T, T → ·T * F e T → ·F.

Análise sintática

E’	→	E
E	→	E + T | T
T	→	T * F | F
F	→	(E) | id

Análise ascendente

Análise LR – Fechamento de conjunto de itens

Exemplo:

E’	→	E
E	→	E + T | T
T	→	T * F | F
F	→	(E) | id

I_0

E’	→	·E
E	→	·E + T
E	→	·T
T	→	·T * F
T	→	·F
F	→	·(E)
F	→	·id

Em seguida, o não-terminal F à direita de um ponto nos força a incluir os itens F→ ·(E) e F → ·id, mas nenhum outro item precisa ser acrescentado.

Análise sintática

I_x

A → α·Bβ

B → ·γ

Análise ascendente

Análise LR – Fechamento de conjunto de itens

Um modo conveniente de implementar a função de fechamento (*closure*) é manter um arranjo *booleano* *added*, indexado pelos não-terminais de G, de modo que *added[B]* seja definido como verdadeiro (*true*) se e quando incluirmos o item B → ·γ para cada produção-B, B → γ.

```
SetOfItems CLOSURE(I){
        J=I;
        repeat
                for (cada item A → α·Bβ em J)
                        for (cada produção B → γ de G)
                                if (B → ·γ não está em J)
                                        Adicione B → ·γ em J;
        until mais nenhum item seja adicionado a J em um passo do loop;
        return J;
```

Análise sintática

I_x

A → α·Bβ

B → ·γ

Análise ascendente

Análise LR – Fechamento de conjunto de itens

Observe que, se uma produção-B for incluída no fechamento de I com o ponto mais à esquerda do seu lado direito, então todas as produções-B serão igualmente incluídas no fechamento. Ex.: A → α·Bβ.

Uma lista de não-terminais B, cujas produções foram incluídas dessa forma, é suficiente. Dividimos todos os conjuntos de itens de interesse em duas classes:

1. Itens de base (*kernel*): o item inicial, S' → ·S, e todos itens cujos pontos não estão mais à esquerda em seus lados direitos.
2. Itens que não são de base: todos os itens com seus pontos mais à esquerda em seus lados direitos, exceto S' → ·S. Ex.: B → ·γ.

Análise sintática

I_x

A → $\alpha \cdot B\beta$

B → $\cdot\gamma$

Análise ascendente

Análise LR – Fechamento de conjunto de itens

Além do mais, cada conjunto de itens de interesse é formado a partir do fechamento de um conjunto de itens de base.

Os itens adicionados no fechamento nunca podem ser itens de base naturalmente.

Assim, podemos representar os conjuntos de itens em que realmente estamos interessados com pouca memória se desprezarmos todos que não são itens de base, sabendo que esses podem ser gerados novamente no processo de fechamento. Ex.: B → ·γ, exceto: S' → ·S.

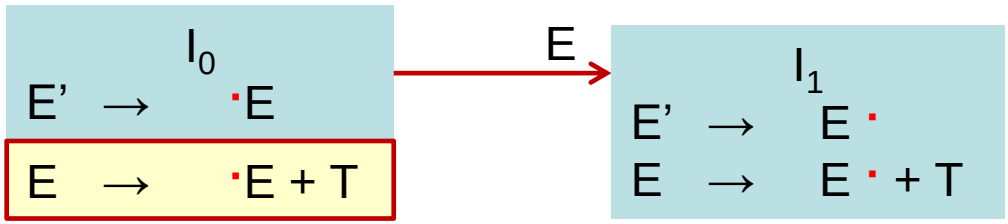
Análise sintática

Análise ascendente

Análise LR – Função de transição

A segunda função usada na construção da coleção *canônica LR(0)* é a função de transição, GOTO(I, X), onde I é o conjunto de itens de base e X é um símbolo da gramática.

GOTO(I, X) é definido como fechamento do conjunto de todos os itens [A→*α*X·*β*] tais que
[A → *α*·X*β*] está em I.



A função de transição GOTO é utilizada para definir as transações no *autômato LR(0)* para uma gramática.

Análise sintática

E’ → E
E → E + T | T
T → T * F | F
F → (E) | id

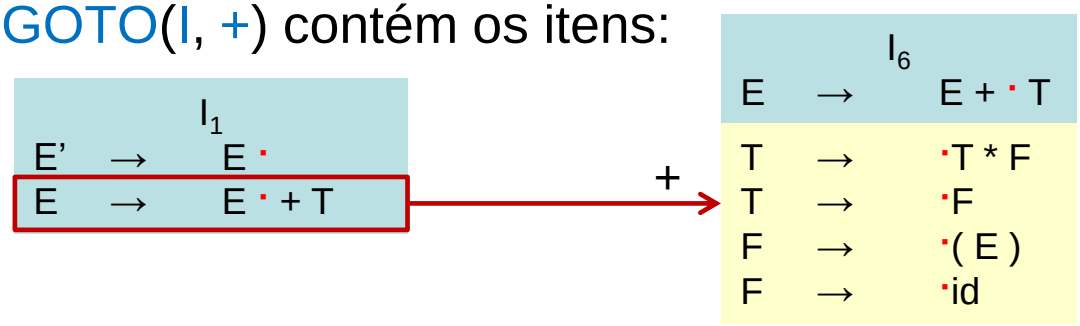
A → α·Xβ
A → αX·β

Análise ascendente

Análise LR – Função de transição

Estados do autômato correspondem aos conjuntos de itens, e GOTO(I, +) especifica a transação do estado I para um novo estado I sob a entrada X.

Se I representa o conjunto com dois itens {[E’ → E·],[E → E·+T]}, então GOTO(I, +) contém os itens:



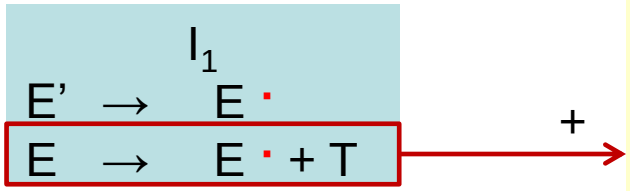
Calculamos GOTO(I, +) examinando os itens em I com o + imediatamente à direita do ponto.

Análise sintática

E’ → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

Análise LR – Função de transição



E → E + · T
T → ·T * F
T → ·F
F → ·(E)
F → ·id

Itens em I:
{[E’ → E·],
[E → E·+T]}

Calculamos GOTO(I, +) examinando os itens em I com o + imediatamente à direita do ponto.

O item E’ → E· não é desse tipo, mas E’ → E· + T sim.

Movemos o ponto para imediatamente à direita de + e obtemos E→E + ·T, e então calculamos o fechamento desse conjunto unitário.

Análise sintática

Análise ascendente

Análise LR – Função de transição

Agora estamos prontos para construir C, a *coleção canônica de conjuntos de itens LR(0)* para uma gramática estendida G'.

```
void Itens(G'){
        C = CLOSURE({[S' → ·S]});
        repeat
                for (cada conjunto de itens I em C)
                        for (cada símbolo da gramática X)
                                if (GOTO(I, X) não é vazio e não está em C)
                                        Adicione GOTO(I, X) em C;
        until nenhum novo conjunto de itens seja adicionado em C em uma rodada;
```

Análise sintática

Análise ascendente

Análise LR – Uso do autômato LR(0)

A ideia central por trás da análise "LR Simples" ou SLR, é a construção do autômato LR(0) a partir da gramática dada.

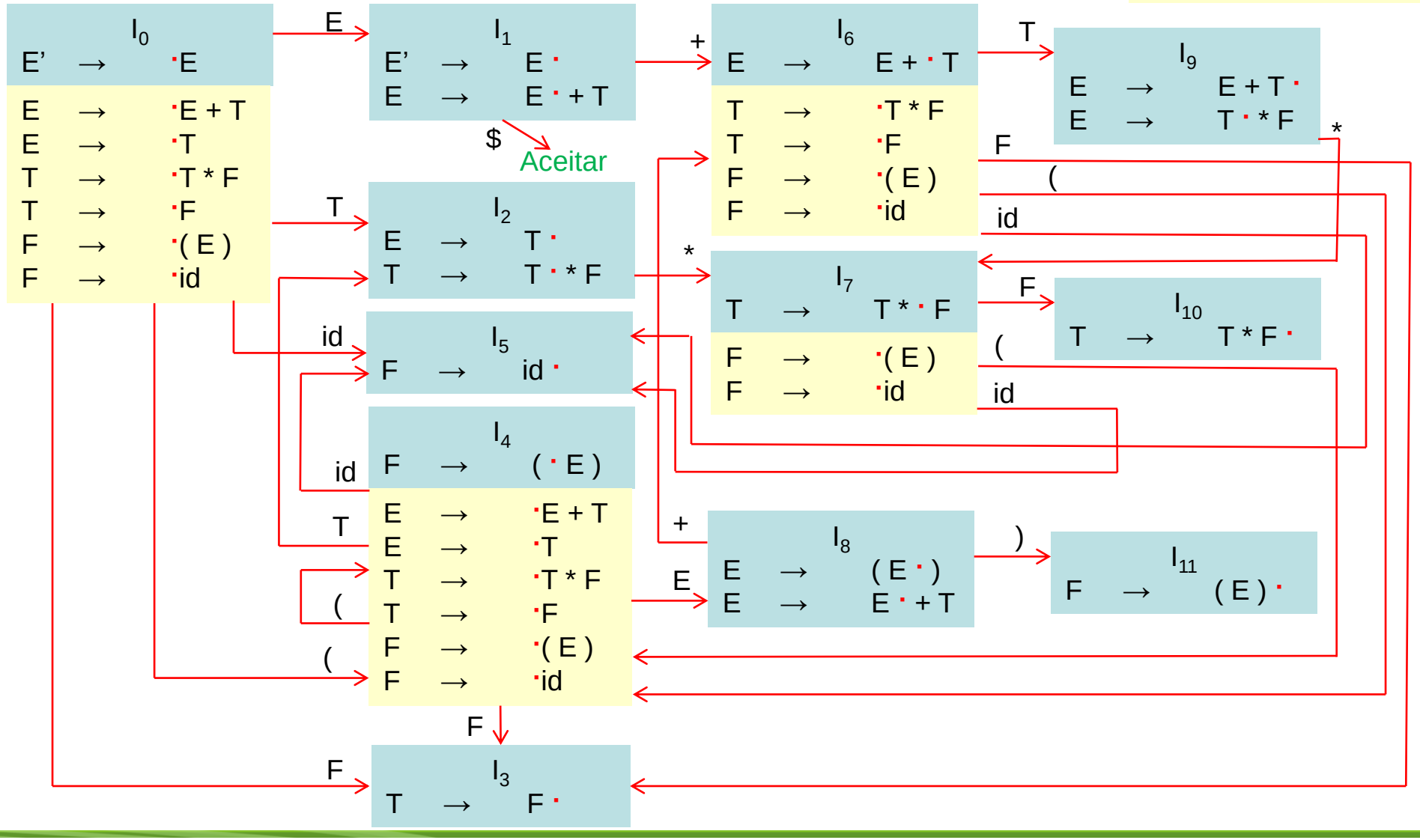
Os estados desses autômatos são os conjuntos de itens da *coleção LR canônica*, e as transições são dadas pela função de transição GOTO.

O estado inicial do autômato LR é dado por CLOSURE({[S' → ·S]}), onde S' é o símbolo inicial da gramática estendida.

Todos os estados são estados de aceitação. Nos referimos ao "*estado j*" como o estado correspondente ao conjunto de itens I_j.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id



Análise sintática

Análise ascendente

Análise LR – Uso do autômato LR(0)

Como os autômatos LR(0) podem ajudar-nos a decidir por uma das ações *shift-reduce*? (veja as decisões entre *shift-reduce* a seguir...)

Suponha que a cadeia γ dos símbolos da gramática leve o autômato LR(0) do estado inicial 0 para algum estado j.

Desta forma, avance sobre o próximo símbolo a se o estado j possuir uma transação sob a.

Caso contrário, escolha reduzir;

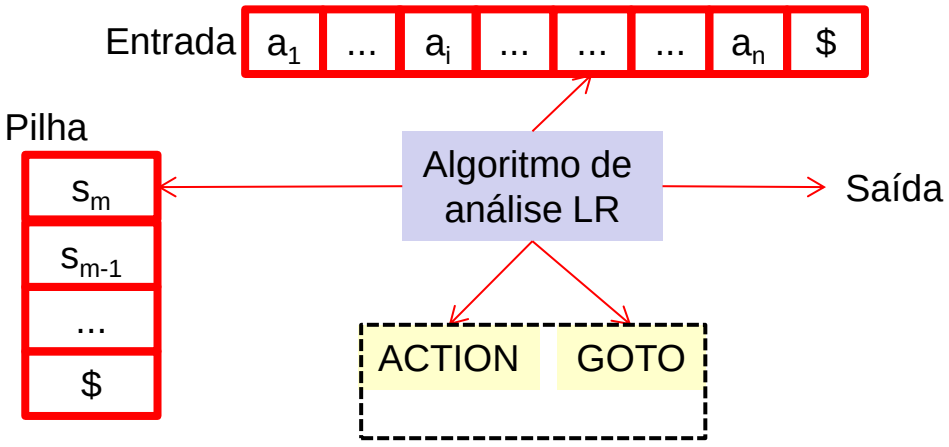
Os itens no estado j nos dirão qual produção usar.

Análise sintática

Análise ascendente

Análise LR – Uso do autômato LR(0)

O algoritmo LR abaixo utiliza sua pilha para acompanhar os estados e também os símbolos da gramática.



Na verdade...

O símbolo da gramática pode ser recuperado a parir do estado, de modo que a pilha possua estados ao invés de símbolos.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

Análise LR – Uso do autômato LR(0)

Ações de uma analisador *shift-reduce* p/ a entrada id * id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	0	$	id * id $	empilha 5 e avança
(2)	0 5	$ id	* id $	reduz segundo F → id
(3)	0 3	$ F	* id $	reduz segundo T→ F
(4)	0 2	$ T	* id $	empilha 7 e avança
(5)	0 2 7	$ T *	id $	empilha 5 e avança
(6)	0 2 7 5	$ T * id	$	reduz segundo F → id
(7)	0 2 7 10	$ T * F	$	reduz segundo T → T * F
(8)	0 2	$ T	$	reduz segundo E → T
(9)	0 1	$ E	$	aceitar

Usamos uma pilha p/ conter os estados

Os símbolos que correspondem aos estados estão na coluna "Símbolos".

A pilha da linha (1) têm o estado inicial 0 do autômato.

Análise sintática

I_0

E' → ·E

E → ·E + T

E → ·T

T → ·T * F

T → ·F

F → ·(E)

F → ·id

id →

I_5

F → id ·

E' → E

E → E + T | T

T → T * F | F

F → (E) | id

Análise ascendente

Análise LR – Uso do autômato LR(0)

Ações de uma analisador *shift-reduce* p/ a entrada id * id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	0	$	id * id $	empilha 5 e avança
(2)	0 5	$ id	* id $	reduz segundo F → id
(3)	0 3	$ F	* id $	reduz segundo T→ F
(4)	0 2	$ T	* id $	empilha 7 e avança
(5)	0 2 7	$ T *	id $	empilha 5 e avança
(6)	0 2 7 5	$ T * id	$	reduz segundo F → id
(7)	0 2 7 10	$ T * F	$	reduz segundo T → T * F
(8)	0 2	$ T	$	reduz segundo E → T
(9)	0 1	$ E	$	aceitar

O símbolo da linha 1 é o marcador de fundo da pilha $.

O próximo símbolo de entrada é id e o estado 0 tem uma transição sob id p/ o estado 5.

Portanto, fazemos a transferência.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

I_5
F → id ·

Análise LR – Uso do autômato LR(0)

Ações de uma analisador *shift-reduce* p/ a entrada id * id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	0	$	id * id $	empilha 5 e avança
(2)	0 5	$ id	* id $	reduz segundo F → id
(3)	0 3	$ F	* id $	reduz segundo T→ F
(4)	0 2	$ T	* id $	empilha 7 e avança
(5)	0 2 7	$ T *	id $	empilha 5 e avança
(6)	0 2 7 5	$ T * id	$	reduz segundo F → id
(7)	0 2 7 10	$ T * F	$	reduz segundo T → T * F
(8)	0 2	$ T	$	reduz segundo E → T
(9)	0 1	$ E	$	aceitar

Na linha 2, o estado 5 é empilhado. (símbolo id).

Não há transição a partir do estado 5, sob a entrada *, por isso reduzimos.

A partir do item [F → id ·] no estado 5, efetuamos uma redução segundo a produção F → id

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

I_5
F → id ·

Análise LR – Uso do autômato LR(0)

Ações de uma analisador *shift-reduce* p/ a entrada id * id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	0	$	id * id $	empilha 5 e avança
(2)	0 5	$ id	* id $	reduz segundo F → id
(3)	0 3	$ F	* id $	reduz segundo T→ F
(4)	0 2	$ T	* id $	empilha 7 e avança
(5)	0 2 7	$ T *	id $	empilha 5 e avança
(6)	0 2 7 5	$ T * id	$	reduz segundo F → id
(7)	0 2 7 10	$ T * F	$	reduz segundo T → T * F
(8)	0 2	$ T	$	reduz segundo E → T
(9)	0 1	$ E	$	aceitar

... o lado direito da produção da linha (2) é id, correspondendo ao estado 5, e seu lado direito é o F.

Com os estados, ao desempilhar o estado 5 correspondendo ao símbolo id, o estado 0 passa a ser o topo.

Análise sintática

I_0

E' → ·E

E → ·E + T
E → ·T
T → ·T * F
T → ·F
F → ·(E)
F → ·id

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

F →

I_3

T → F ·

Análise ascendente

Análise LR – Uso do autômato LR(0)

Ações de uma analisador *shift-reduce* p/ a entrada id * id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	0	$	id * id $	empilha 5 e avança
(2)	0 5	$ id	* id $	reduz segundo F → id
(3)	0 3	$ F	* id $	reduz segundo T→ F
(4)	0 2	$ T	* id $	empilha 7 e avança
(5)	0 2 7	$ T *	id $	empilha 5 e avança
(6)	0 2 7 5	$ T * id	$	reduz segundo F → id
(7)	0 2 7 10	$ T * F	$	reduz segundo T → T * F
(8)	0 2	$ T	$	reduz segundo E → T
(9)	0 1	$ E	$	aceitar

... Procuramos uma transição sob F, o lado esquerdo da produção.

... O estado 0 possui uma transição em F p/ o estado 3, onde empilhamos o estado 3, com o símbolo F. Ver linha (3).

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

I_3
T → F ·

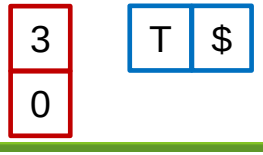
Análise LR – Uso do autômato LR(0)

Ações de uma analisador *shift-reduce* p/ a entrada id * id

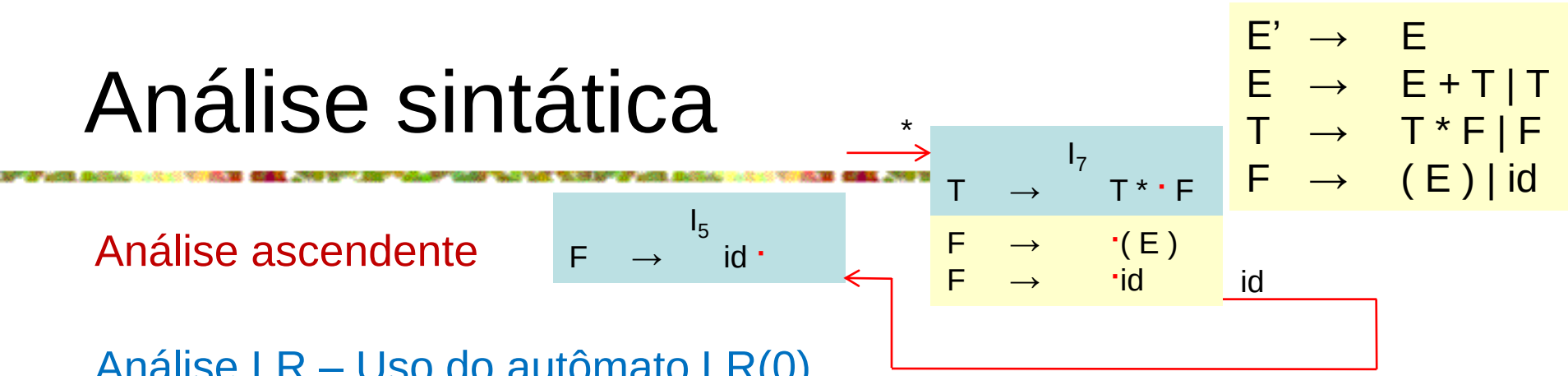
Linha	Pilha	Símbolos	Entrada	Ação
(1)	0	$	id * id $	empilha 5 e avança
(2)	0 5	$ id	* id $	reduz segundo F → id
(3)	0 3	$ F	* id $	reduz segundo T→ F
(4)	0 2	$ T	* id $	empilha 7 e avança
(5)	0 2 7	$ T *	id $	empilha 5 e avança
(6)	0 2 7 5	$ T * id	$	reduz segundo F → id
(7)	0 2 7 10	$ T * F	$	reduz segundo T → T * F
(8)	0 2	$ T	$	reduz segundo E → T
(9)	0 1	$ E	$	aceitar

Não há transição a partir do estado 3, sob a entrada *, por isso reduzimos.

A partir do item [T → F ·] no estado 3, efetuamos uma redução segundo a produção T → F



Análise sintática



Análise ascendente

Análise LR – Uso do autômato LR(0)

Ações de uma analisador *shift-reduce* p/ a entrada id * id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	0	$	id * id $	empilha 5 e avança
(2)	0 5	$ id	* id $	reduz segundo F → id
(3)	0 3	$ F	* id $	reduz segundo T→ F
(4)	0 2	$ T	* id $	empilha 7 e avança
(5)	0 2 7	$ T *	id $	empilha 5 e avança
(6)	0 2 7 5	$ T * id	$	reduz segundo F → id
(7)	0 2 7 10	$ T * F	$	reduz segundo T → T * F
(8)	0 2	$ T	$	reduz segundo E → T
(9)	0 1	$ E	$	aceitar

Como outro exemplo, considere a linha (5) com o estado 7, referente ao símbolo *, no topo da pilha.

Esse estado 7 possui uma transação para o estado 5 sob a entrada id, de modo que empilhamos o estado 5.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

I_5
F → id ·

Análise LR – Uso do autômato LR(0)

Ações de uma analisador *shift-reduce* p/ a entrada id * id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	0	$	id * id $	empilha 5 e avança
(2)	0 5	$ id	* id $	reduz segundo F → id
(3)	0 3	$ F	* id $	reduz segundo T→ F
(4)	0 2	$ T	* id $	empilha 7 e avança
(5)	0 2 7	$ T *	id $	empilha 5 e avança
(6)	0 2 7 5	$ T * id	$	reduz segundo F → id
(7)	0 2 7 10	$ T * F	$	reduz segundo T → T * F
(8)	0 2	$ T	$	reduz segundo E → T
(9)	0 1	$ E	$	aceitar

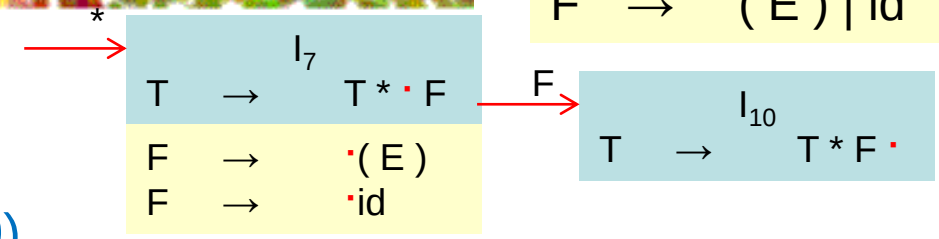
... o estado 5 não possui transições, de modo que reduzimos segundo F → id.

...quando removemos da pilha o estado 5 correspondente ao lado direito id, o estado 7 passa a ser o topo da pilha.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente



Análise LR – Uso do autômato LR(0)

Ações de uma analisador *shift-reduce* p/ a entrada id * id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	0	$	id * id $	empilha 5 e avança
(2)	0 5	$ id	* id $	reduz segundo F → id
(3)	0 3	$ F	* id $	reduz segundo T→ F
(4)	0 2	$ T	* id $	empilha 7 e avança
(5)	0 2 7	$ T *	id $	empilha 5 e avança
(6)	0 2 7 5	$ T * id	$	reduz segundo F → id
(7)	0 2 7 10	$ T * F	$	reduz segundo T → T * F
(8)	0 2	$ T	$	reduz segundo E → T
(9)	0 1	$ E	$	aceitar

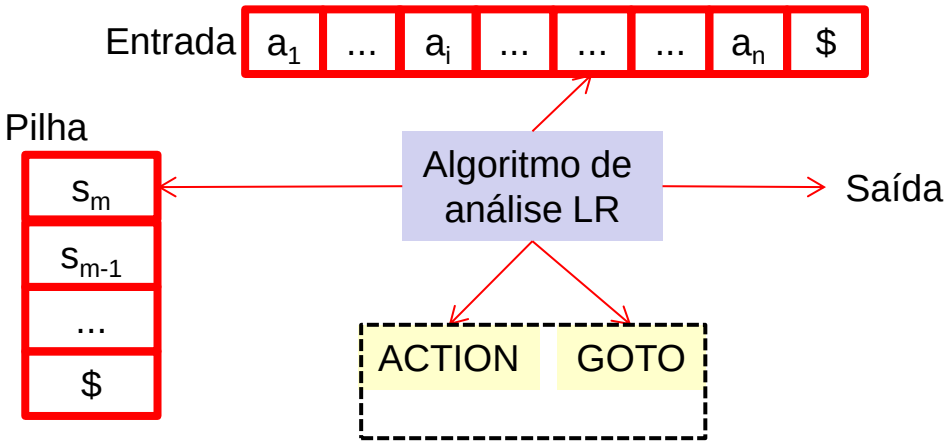
Como o estado 7 possui uma transação sob F para o estado 10, empilhamos o estado 10 (símbolo F).

Análise sintática

Análise ascendente

Análise LR – O algoritmo de análise LR

Consiste em uma entrada, uma saída, uma pilha, o algoritmo de análise sintática e uma tabela de análise construída de duas partes (ACTION e GOTO).



O algoritmo é o mesmo para todos os analisadores sintáticos LR.

Só a tabela de análise é diferente de um analisador para o outro.

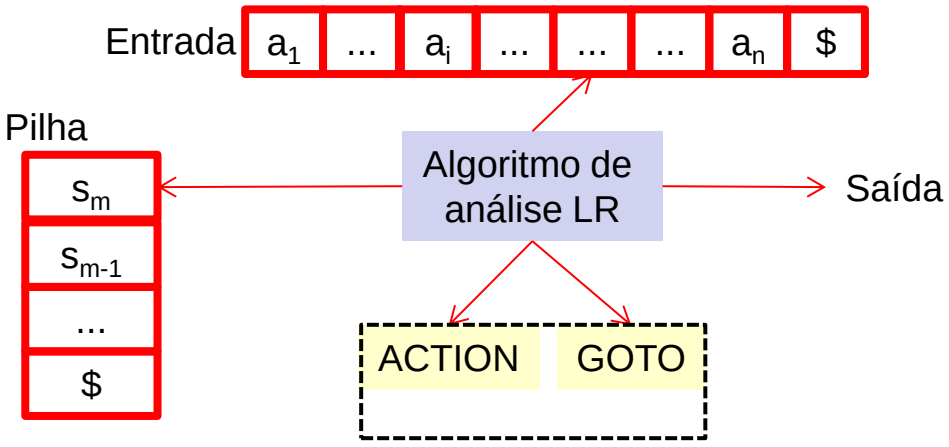
O algoritmo lê caracteres de um *buffer* de entrada um de cada vez.

Análise sintática

Análise ascendente

Análise LR – O algoritmo de análise LR

Consiste em uma entrada, uma saída, uma pilha, o algoritmo de análise sintática e uma tabela de análise construída de duas partes (ACTION e GOTO).



O analisador sintático *shift-reduce* transfere um símbolo para pilha, já o analisador sintático LR transfere um estado.

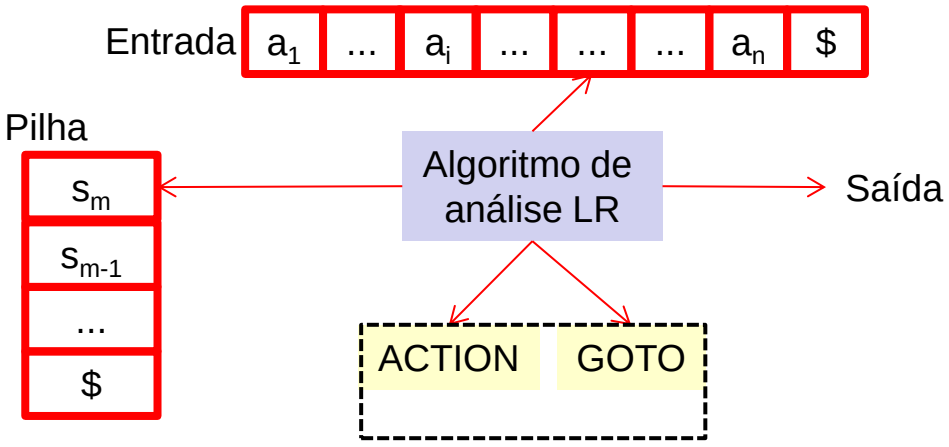
Cada estado resume a informação contida na pilha abaixo dele.

Análise sintática

Análise ascendente

Análise LR – O algoritmo de análise LR

Consiste em uma entrada, uma saída, uma pilha, o algoritmo de análise sintática e uma tabela de análise construída de duas partes (ACTION e GOTO).



A pilha contém uma sequencia de estados, $s_0 s_1 \ldots s_m$, onde s_m está no topo.

No método SLR, a pilha contém estados do autômato LR(0).

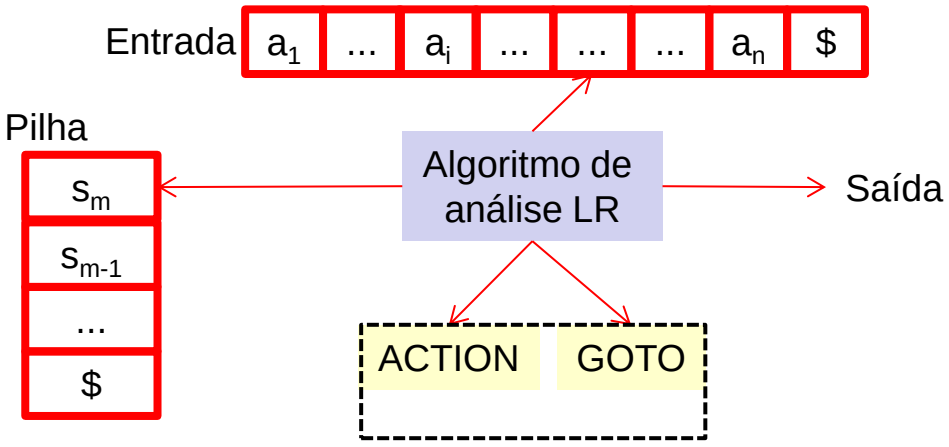
Por construção, a cada estado é associado 1 símbolo da gramática.

Análise sintática

Análise ascendente

Análise LR – O algoritmo de análise LR

Consiste em uma entrada, uma saída, uma pilha, o algoritmo de análise sintática e uma tabela de análise construída de duas partes (ACTION e GOTO).



Lembrem-se de que os estados correspondem a conj. de itens, e que há uma transição do estado i para o estado j se GOTO(I_i, X) = I_j.

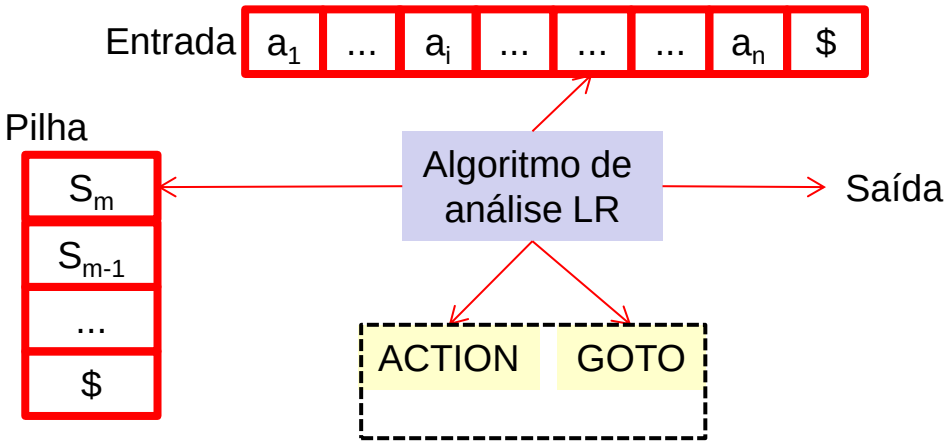
Todas as transições para o estado j devem ser para o mesmo símbolo da gramática X.

Análise sintática

Análise ascendente

Análise LR – O algoritmo de análise LR

Consiste em uma entrada, uma saída, uma pilha, o algoritmo de análise sintática e uma tabela de análise construída de duas partes (ACTION e GOTO).



Desta forma, cada estado, exceto o estado inicial 0, possui um único símbolo da gramática associado a ele.

Análise sintática

Análise ascendente

Análise LR – Estrutura da tabela LR

A tabela LR consiste em duas partes: ACTION, uma função de ação de análise, e GOTO, uma função de transição.

1. A função ACTION recebe como argumentos um estado i e um estado a (ou $). O valor de ACTION(i, a) pode ter um dos quatro formatos:

   a) *Shift j*, onde j é um estado.

      A ação tomada pelo analisador sintático efetivamente transfere a entrada a para a pilha, mas usa o estado j para representar a.

Análise sintática

Análise ascendente

Análise LR – Estrutura da tabela LR

b) *Reduce* $A \rightarrow \beta$. A ação do analisador efetivamente reduz β no topo da pilha para A, o lado esquerdo da produção.

c) *Accept*: O analisador aceita a entrada e termina a análise.

d) *Error*: O analisador descobre um erro em sua entrada, passa o controle p/ o recuperador de erro, que toma alguma ação corretiva.

2. Estendemos a função GOTO, definida sobre um conjunto de itens, para os estados: se GOTO(I_i, A) = I_j, então GOTO também mapeia um estado i e um não-terminal A para o estado j.

Análise sintática

Análise ascendente

Análise LR – Configuração do analisador LR

Para descrever o comportamento de uma analisador LR, é recomendável ter uma notação representado o estado completo do analisador: sua pilha e a entrada restante.

Uma configuração p/ um analisador LR é um par: ($s_0 s_1 ... s_m$, $a_i a_{i+1} ... a_n$ \$). O primeiro componente é o conteúdo da pilha, com o topo à direita, e o segundo componente representa a entrada restante.

Essa configuração representa a seguinte forma sentencial à direita: ($X_0 X_1 ... X_m a_i, a_{i+1} ... a_n$). Assim como faria um analisador *shift-reduce*.

Análise sintática

Análise ascendente

Análise LR – Configuração do analisador LR

A única diferença é que, em invés de símbolos da gramática, a pilha contém estados, a partir dos quais os símbolos da gramática podem ser recuperados.

Ou seja, X_i é o símbolo da gramática representado pelo estado s_i.

Observe que s_0, o estado inicial do analisador sintático, não representa um símbolo da gramática, e serve como marcador de fundo da pilha, além de desempenhar um papel importante na análise.

Análise sintática

Análise ascendente

Análise LR – Comportamento do analisador LR

O próximo movimento do analisador, a partir da configuração dada, é determinado pela leitura de a_i, o símbolo de entrada corrente, e s_m, o estado do topo da pilha, e então consultando a entrada ACTION[s_m, a_i] na tabela de ação de análise.

As configurações restantes após cada um dos quatro tipos de movimentação são as seguintes:

1. Se ACTION[S_m, a_i] = *shift s*, o analisador sintático executa um *movimento de transferência* (*shift move*); ele empilha o estado s e avança da entrada, resultando na configuração: ($s_0 s_1 ... s_m$, $a_{i+1} ... a_n$ $).

Análise sintática

Análise ascendente

Análise LR – Comportamento do analisador LR

2. Se ACTION[S_m, a_i]=*reduce* $A \rightarrow \beta$, então o analisador sintático executa um *movimento de reduzir* (*reduce move*), resultando na configuração: ($s_0 s_1 \ldots s_{m-r} s$, $a_i a_{i+1} \ldots a_n$ \$),

   onde r é o comprimento de β, e s = GOTO[s_{m-r}, A].

Aqui, o analisador sintático incialmente desempilhou os r símbolos de estado da pilha, expondo o estado s_{m-r}.

O analisador sintático então empilhou s, a entrada para GOTO[s_{m-r}, A], na pilha. O símbolo corrente da entrada não é alterado em um movimento de reduzir.

Análise sintática

Análise ascendente

Análise LR – Comportamento do analisador LR

2. ...

Para os LR que construiremos, $X_{m-r+1}...X_m$, a sequencia de símbolos da gramática correspondendo aos estados retirados da pilha, sempre irão casar com β, o lado direito da produção sendo reduzida.

A saída de um analisador LR é gerada após um movimento de reduzir, executando a ação semântica, associada à produção que está sendo reduzida.

No momento, vamos considerar que a saída consiste só na impressão da produção de redução.

Análise sintática

Análise ascendente

Análise LR – Comportamento do analisador LR

3. Se ACTION[s_m, a_i] = *accept*, a análise está concluída.

4. Se ACTION[s_m, a_i] = *error*, o analisador sintático descobriu um erro e ativa uma rotina para recuperação de erro.

Todos os analisadores sintáticos LR se comportam dessa maneira.

A única diferença entre eles diz respeito a informação contida nos campos ACTION e GOTO da tabela de análise.

O algoritmo de análise LR é resumido nos próximos slides...

Análise sintática

Análise ascendente

Análise LR – Comportamento do analisador LR - Algoritmo

Entrada: Uma cadeia de entrada w e uma tabela de análise com funções ACTION e GOTO para uma gramática G.

Saída: Se w está em L(G), os passos os passos de uma redução de uma análise ascendente para w.

Caso contrário, uma indicação de erro.

Método: Inicialmente, o analisador sintático possui w$ no buffer de entrada e s_0 em sua pilha, onde s_0 representa o estado inicial.

O analisador sintático então, executa o algoritmo...

Análise sintática

Análise ascendente

Análise LR – Comportamento do analisador LR - Algoritmo

```
[PSEUDO] seja a o primeiro símbolo de w$;
while (1){
        Seja s o estado no topo da pilha;
        if (ACTION[s, a] = shift t)
                empilha t na pilha;
                seja a o próximo símbolo da entrada;
        } else if (ACTION[s, a] = reduce A→ β) {
                desempilha símbolos |β| da pilha;
                faça o estado t agora ser o topo da pilha;
                empilhe GOTO[t, A] na pilha;
                imprima a produção A→ β;
        } else if (ACTION[s, a] = accept) pare; /* a análise terminou*/
        else chame uma rotina de recuperação de erros;
}
```

Análise sintática

E’	→	E
E	→	E + T | T
T	→	T * F | F
F	→	(E) | id

Análise ascendente

Análise LR – Comportamento do analisador LR – Exemplo

A tabela do próximo slide monstra as funções ACTION e GOTO de uma tabela de análise LR para a gramática de expressão estudada nesta aula, e repetida aqui com produções numeradas:

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

Os códigos para as ações são:

1. S_i significa avança na entrada e empilha o estado i na pilha.
2. R_j significa *reduce* segundo a produção de número j.
3. Acc significa *accept*.
4. Entrada em branco significa *error*.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Estado	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		S6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		r6			s11				
9		s1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

O valor de GOTO(s, a) para o terminal *a* está em ACTION da tabela de análise conectado à ação de transferência sob a entrada *a* e o estado *s*.

O GOTO fornece o GOTO[s, a] para os não-terminais A.

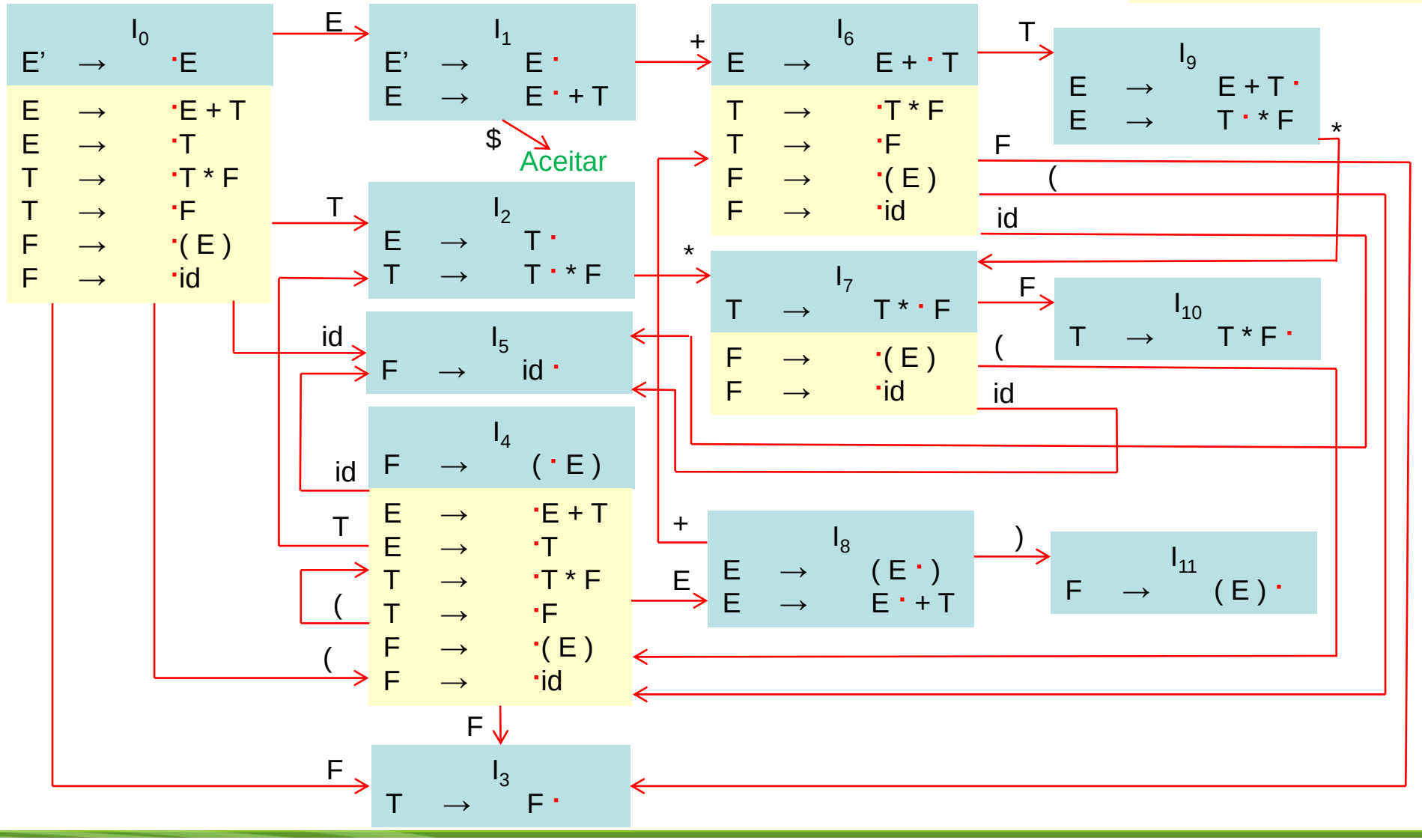
Ok!!!

Mas como eu consigo os dados desta tabela?

A. sintática

FOLLOW(E) = FOLLOW(E') = {), $}
FOLLOW(T) = FOLLOW(T') = {+,), $}
FOLLOW(F) = {+, *,), $}.

E' → E
E → E + T | T
T → T * F | F
F → (E) | id



Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

Análise LR – Construindo tabelas de análise SLR

Esse método é o mais simples dos LRs. Vamos nos referir a tabela de análise construída por este método como uma tabela SLR, e o analisador LR usando uma tabela SLR como um analisador sintático SLR.

O método SLR começa com os itens LR(0) e autômatos LR(0), ou seja, dada uma gramática G, estendemos G para produzir G', com o novo símbolo inicial S'. A partir de G', construímos C, a coleção canônica de conj. de itens para G' junto com a função GOTO.

Para construir as entradas ACTION e GOTO da tabela de análise usando o algoritmo a seguir, é preciso que conheçamos FOLLOW(A) para cada não-terminal A de uma gramática.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	0	$	id * id + id $	empilha 5 e avança
(2)	0 5	$ id	* id + id $	reduz segundo F → id
(3)	0 3	$ F	* id + id $	reduz segundo T→ F
(4)	0 2	$ T	* id + id $	empilha 7 e avança
(5)	0 2 7	$ T *	id + id $	empilha 5 e avança
(6)	0 2 7 5	$ T * id	+ id $	reduz segundo F → id
(7)	0 2 7 5 10	$ T * F	+ id $	reduz segundo T → T * F
(8)	0 2	$ T	+ id $	reduz segundo E → T
(9)	0 1	$ E	+ id $	empilha 6 e avança
(10)	0 1 6	$ E +	id $	empilha 5 e avança
(11)	0 1 6 5	$ E + id	$	reduz segundo F → id
(12)	0 1 6 3	$ E + F	$	reduz segundo T→ F
(13)	0 1 6 9	$ E + T	$	reduz segundo E → E + T
(14)	0 1	$ E	$	aceitar

Movimentos de um analisador LR p/ a entrada: id * id + id

A coluna Símbolos mostra a sequencia de símbolos de uma gramática que correspondem aos estados contidos na pilha.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	**0**	**$**	**id** * id + id $	empilha 5 e avança
(2)	0 5	$ id	* id + id $	reduz segundo F → id
(3)	0 3	$ F	* id + id $	reduz segundo T→ F
(4)	0 2	$ T	* id + id $	empilha 7 e avança
(5)	0 2 7	$ T *	id + id $	empilha 5 e avança
(6)	0 2 7 5	$ T * id	+ id $	reduz segundo F → id
(7)	0 2 7 5 10	$ T * F	+ id $	reduz segundo T → T * F
(8)	0 2	$ T	+ id $	reduz segundo E → T
(9)	0 1	$ E	+ id $	empilha 6 e avança
(10)	0 1 6	$ E +	id $	empilha 5 e avança
(11)	0 1 6 5	$ E + id	$	reduz segundo F → id
(12)	0 1 6 3	$ E + F	$	reduz segundo T→ F
(13)	0 1 6 9	$ E + T	$	reduz segundo E → E + T
(14)	0 1	$ E	$	aceitar

Ex.: Na linha (1), o analisador SLR está no estado 0, o estado inicial tem símbolos da gramática, e com id, o primeiro símbolo da entrada.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	**0**	**$**	**id** * id + id $	empilha 5 e avança

Ex.: Na linha (1), o analisador SLR está no estado 0, o estado inicial tem símbolos da gramática, e com id, o primeiro símbolo da entrada.

Estado	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3

A ação na linha 0 (estado) e na coluna id do campo ACTION é s5, significando avance empilhando o estado 5.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	0	$	id * id + id $	empilha 5 e avança
(2)	0 **5**	$ **id**	***** id + id $	reduz segundo F → id
(3)	0 3	$ F	* id + id $	reduz segundo T→ F
(4)	0 2	$ T	* id + id $	empilha 7 e avança
(5)	0 2 7	$ T *	id + id $	empilha 5 e avança
(6)	0 2 7 5	$ T * id	+ id $	reduz segundo F → id
(7)	0 2 7 5 10	$ T * F	+ id $	reduz segundo T → T * F
(8)	0 2	$ T	+ id $	reduz segundo E → T
(9)	0 1	$ E	+ id $	empilha 6 e avança
(10)	0 1 6	$ E +	id $	empilha 5 e avança
(11)	0 1 6 5	$ E + id	$	reduz segundo F → id
(12)	0 1 6 3	$ E + F	$	reduz segundo T→ F
(13)	0 1 6 9	$ E + T	$	reduz segundo E → E + T
(14)	0 1	$ E	$	aceitar

Foi isso que ocorreu na linha (2): símbolo do estado 5 foi posto na pilha, depois, id foi removido da entrada.

... Então * torna-se o símbolo de entrada corrente, e a ação do estado 5 sob a entrada * é reduzir segundo a produção F → id.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Linha	Pilha	Símbolos	Entrada	Ação
(1)	0	$	id * id + id $	empilha 5 e avança
(2)	0 5	$ id	* id + id $	reduz segundo F → id
	0	$ **F**	***** id + id $	

Estado	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3

Linha	Pilha	Símbolos	Entrada	Ação
(3)	0 **3**	$ **F**	***** id + id $	reduz segundo T→ F

Um símbolo estado é desempilhado.

O estado 0 é, então, exposto.

Como a transição do estado 0 sob F é 3, o estado 3 é colocado na pilha.

Temos a situação da linha (3).

... cada um dos mov. restantes pode ser determinado de modo semelhante.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

Análise LR – Algoritmo para construção de uma tabela SLR

Entrada: uma gramática estendida G'.

Saída: as funções ACTION e GOTO da tabela de análise SLR para G'.

Método:

1. Construa C = {I_0, I_1..., I_n}, a coleção de conj. de itens LR(0) p/ G'.
2. O estado i é construído a partir de I_i. As ações sintáticas para i são determinadas da seguinte forma:

   a) Se o item [A → $\alpha \cdot a\beta$] está em I_i e GOTO(I_i, a) = I_j, então defina ACTION[i, a] como "shift j". Aqui, a deve ser um terminal.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

Análise LR – Algoritmo para construção de uma tabela SLR

2. ...

b) Se o item [A → α·] está em I_i, então defina ACTION[i, a] como A→α para todo a em FOLLOW(A). Aqui, A pode não ser S', o símbolo inicial da gramática.

c) Se o item [S' → S·] estiver em I_i, então defina ACTION[i, $] como "*accept*".

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

Análise LR – Algoritmo para construção de uma tabela SLR

...

3. As transações goto 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 não definidas pelas regras (2) e (3) caracterizam "error".

5. O estado inicial do analisador é aquele construído a partir do conj. de itens contendo [S' → ·S].

Análise sintática

E’ → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

Análise LR – Algoritmo para construção de uma tabela SLR

A tabela de análise consistindo as funções ACTION e GOTO geradas pelo algoritmo anterior é chamada de *tabela SLR(1) para G*.

Um analisador LR usando esta tabela é chamado de *analisador sintático SLR(1) para G*, e uma gramática tendo uma tabela de análise SLR(1) é considerada *SLR(1)*.

Usualmente, omitimos o “*(1)*” depois de “SLR”, pois não lidaremos aqui com analisadores com mais de um símbolo de *loockhead*.

Análise sintática

E’	→	E
E	→	E + T | T
T	→	T * F | F
F	→	(E) | id

Análise ascendente

Análise LR – Algoritmo para construção de uma tabela SLR - Exemplo

Vamos construir a tabela SLR para a gramática estendida de expressão. A coleção canônica de conj. de itens LR(0) para a gramática estudada.

Primeiro, considere o conj. de itens I_0:

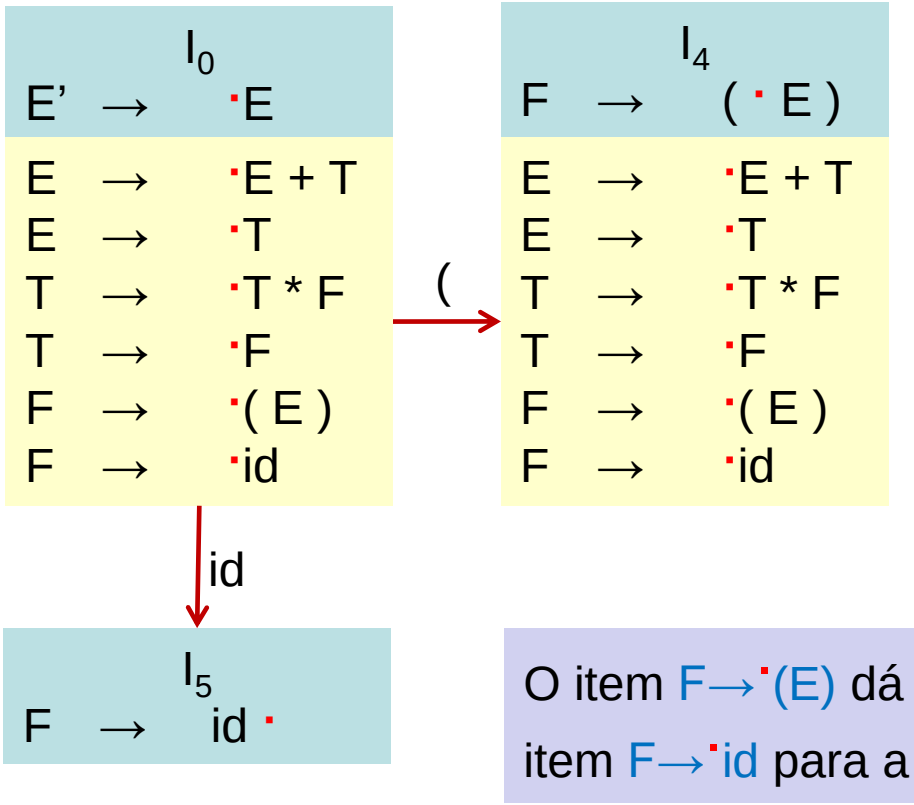
I_0		
E’	→	·E
E	→	·E + T
E	→	·T
T	→	·T * F
T	→	·F
F	→	·(E)
F	→	·id

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

Análise LR – Algoritmo para construção de uma tabela SLR - Exemplo



Estado	ACTION					
	id	+	*	(	)	$
0	s5			s4		

O item F→·(E) dá origem a entrada ACTION[0, (] = *shift 4*, e o item F→·id para a entrada ACTION[0, id] = *shift 5*.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

Análise LR – Algoritmo para construção de uma tabela SLR - Exemplo

Os demais itens I_0 não produzem entradas na parte ACTION referente ao estado 0.

Agora, considere I_1:

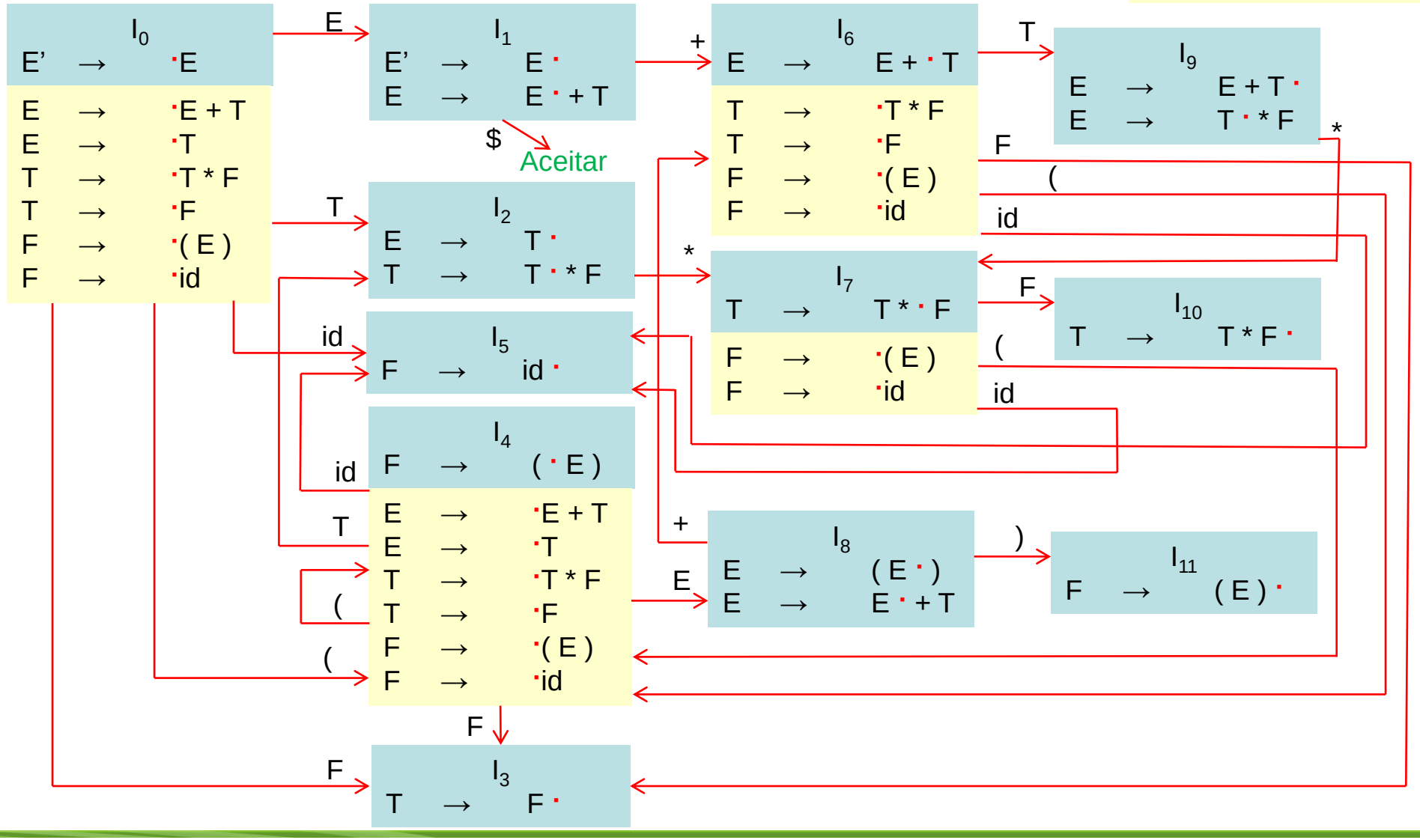
I_1

E' → E ·
E → E · + T

O primeiro gera ACTION[1, $] = *accept*, e o segundo gera ACTION[1, +] = *shift 6*.

Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id



Análise sintática

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

Análise ascendente

Análise LR – Algoritmo para construção de uma tabela SLR - Exemplo

Agora, considere I_2:

I_2

E → T ·
T → T · * F

Com o FOLLOW(E) = {$, +,)}, o primeiro item faz que ACTION[2, $] = ACTION[2, +] = ACTION[2,)] = *reduce* segundo a produção E → T.

O segundo item gera ACTION[2, *] = *shift 7*.

Continuando esse processo, obtemos a tabela ACTION e GOTO.

Análise sintática

Análise ascendente

Análise LR – Algoritmo para construção de uma tabela SLR - Exemplo

Toda gramática SLR(1) é não-ambígua, mas existem muitas gramáticas não-ambíguas que não são SLR(1).

Considere as seguintes produções:

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

Pense em L e R como significando valor l e valor r, respectivamente, e * como um operador indicando "conteúdo de".

Análise sintática

S → L = R | R
L → * R | id
R → L

Análise ascendente

Análise LR – Algoritmo para construção de uma tabela SLR - Exemplo

A coleção LR canônica de conj. de itens LR(0) para a gramática estudada:

I_0: S' → · S
S → · L = R
S → · R
L → · * R
L → · id
R → · L

I_1: S' → S·

I_2: S → L· = R
R → L·

I_3: S → R·

I_4: L → * · R
R → · L
L → · * R
L → · id

I_5: L → id ·

I_6: S → L = · R
R → · L
L → · * R
L → · id

I_7: L → * R ·

I_8: R → L ·

I_9: S → L = R ·

Análise sintática

S → L = R | R
L → * R | id
R → L

Análise ascendente

Análise LR – Algoritmo para construção de uma tabela SLR - Exemplo

Considere o conj. de itens I_2.

I_2: S → L · = R
R → L ·

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

Como FOLLOW(R) contém = (para ver porquê, considere a derivação $S \Rightarrow L = R \Rightarrow * R = R$), o segundo item define ACTION[2,=] como "*reduce* segundo a produção R → L".

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

Análise sintática

S → L = R | R
L → * R | id
R → L

Análise ascendente

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.

Próxima aula

Os métodos LR canônico e o LALR conseguem reconhecer a gramática em questão, e terão sucesso com uma gama maior de gramáticas. Porém, observe que existem gramáticas não ambíguas para os quais todos os métodos de construção de analisadores sintáticos LR produzirão tabelas de ação de análise de conflitos.

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

COMPILADORES

Obrigado!!

Prof. Geovane Griesang
geovanegriesang@unisc.br