Análise Sintática

Prof^a Jerusa Marchi

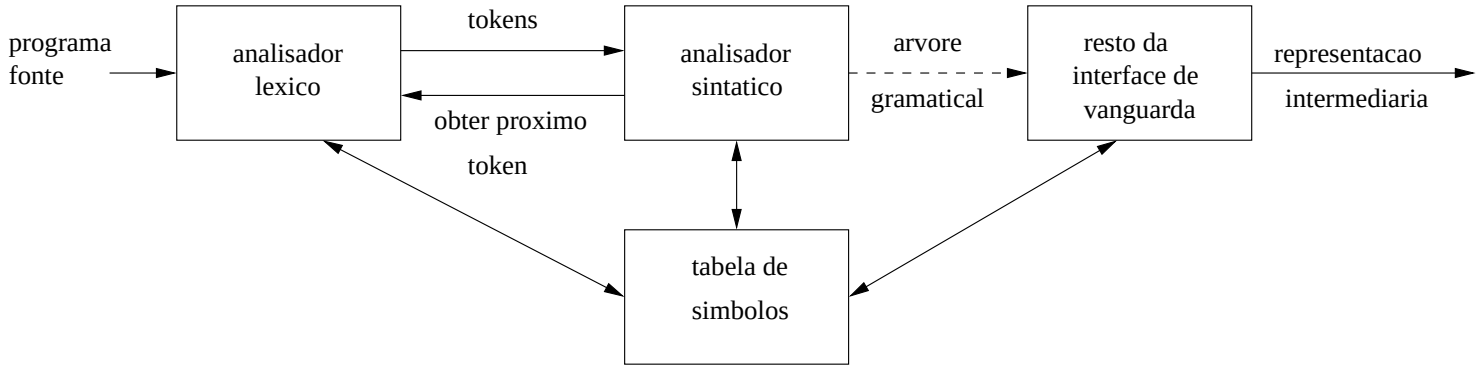
Departamento de Informática e Estatística

Universidade Federal de Santa Catarina

e-mail: jerusa.marchi@ufsc.br

Análise Sintática

- Função: Verificar se as construções usadas no programa estão gramaticalmente corretas, ou seja, se a estrutura sintática é especificada na G.L.C



Análise Sintática

- O análisador sintático muitas vezes é chamado de *Parser*
- A análise sintática, de *Parsing*
- A árvore gerada como resultado, de *Parse*
- Os métodos de análise são chamados de *Teoria de Parsing*

Análise Sintática e Detecção de Erros

- O analisador deve ser projetado para emitir mensagens para quaisquer erros de sintaxe encontrados no programa
- Frequentemente boa parte da detecção de erros num compilador gira em torno da fase de análise sintática

Análise Sintática e Detecção de Erros

- Exemplos de Erros:
  - Léxicos: grafia errada de um identificador, palavra-chave ou operador (ellipseSize - elipseSize)
  - Sintáticos: expressão aritmética com parênteses não balanceados, case sem switch em Java
  - Semânticos: operador aplicado a um operando incompatível ou um return em um método void
  - Lógicos: chamada infinitamente recursiva

Detecção e Tratamento de Erros

- Metas do tratador de erros:
  - Deve relatar a presença de erros de forma clara e acurada (tipo, localização)
  - Deve se recuperar de erros suficientemente rápido a fim de ser capaz de detectar erros subsequentes
  - Não deve retardar significativamente o processamento de programas corretos

Detecção e Tratamento de Erros

- Uma estratégia de recuperação de erros deve considerar os tipos de erros mais propensos a ocorrer e razoáveis de processar
- O tratador de erros deverá reportar o erro da seguinte forma:
  - Imprimir a linha ilegal (com um apontador para a posição no qual o erro foi encontrado)
  - Imprimir uma mensagem compreensível de diagnóstico
  - Exemplo:

```
21: printf("Hello World!")
main.c: error line 21: ";"  expected
```

Estratégias para a recuperação de erros

- O analisador ao encontrar um erro pode
  - abortar o processo de compilação e indicar o erro
  - tentar se recuperar do erro, a fim de continuar processando o restante do programa

Estratégias para a recuperação de erros

- Modo pânico
  - método mais simples de implementar
  - ao descobrir um erro, o analisador sintático descarta símbolos de entrada, um de cada vez, até que seja encontrado um token de sincronização (como "end" ou ";")
- Recuperação em nível de frase
  - ao descobrir um erro, o analisador sintático pode realizar uma correção local, como por exemplo, substituir uma ’,’ (vírgula) por ’;’ (ponto e vírgula), eliminar um ’;’ estranho ou incluir um ’;’ ausente
  - problema: erros que ocorreram antes do ponto de detecção

Estratégias para a recuperação de erros

- Produções de erro
  - aumento da gramática com as produções que geram construções ilegais
  - se uma produção de erro for usada pelo analisador sintático pode-se gerar diagnósticos apropriados para indicar a construção ilegal que foi reconhecida na entrada.
- Correção global
  - alterações mínimas para tentar corrigir o código com problemas
  - dada uma sentença errada x, um algoritmo escolhe uma árvore de derivação para uma cadeia y, tal que o número de inserções, substituições e exclusões seja o menor possível para transformar x em y
  - técnica de interesse teórico
  - alto custo de implementação

Estratégias para a recuperação de erros

- Com o desenvolvimento da tecnologia, (desktops e memória), as técnicas de recuperação de erro se tornaram praticamente desnecessárias
- O custo do tratamento de erro e o tempo necessário a sua execução são elevados
- Em geral, é mais eficiente abortar o processo de compilação e indicar o erro corrente

Análise Sintática

- Se a compilação foi bem sucedida, o analisador sintático constrói uma árvore de derivação, que é passada ao restante da interface de vanguarda
- A árvore de derivação pode ser explícita (armazenada em uma estrutura de dados) ou implícita (nas chamadas das rotinas que aplicam as regras de produção)

Análise Sintática

- Estratégias de desenvolvimento:
  - Top-Down ou Descendente - constrói a árvore de derivação a partir do símbolo inicial da gramática (Derivação)
  - Bottom-Up ou Ascendente - constrói a árvore a partir dos tokens do texto até o símbolo inicial da gramática (Redução)
- Em ambas as estratégias, a entrada é consumida da esquerda para a direita, um token de cada vez

Análise Sintática

- Descendentes
  - Não Determinísticos
    - Descendente Recursivo com Retrocesso
  - Determinísticos
    - Preditivo LL(1)

Análise Sintática

- Ascendentes
  - Não Determinísticos
    - Shift-Reduce com Back-track
  - Determinísticos
    - Família LR
      - SLR(1)
      - LALR(1)
      - LR(1)
    - Precedência
      - Operadores
      - Simples
      - Estendida

Análise Sintática

Considerações:

- Técnicas Não-Determinísticas
  - Exigem implementação com back-track (retrocesso)
  - Não limitam a classe de GLC que pode ser analisada
  - Complexidade exponencial

Análise Sintática

Considerações:

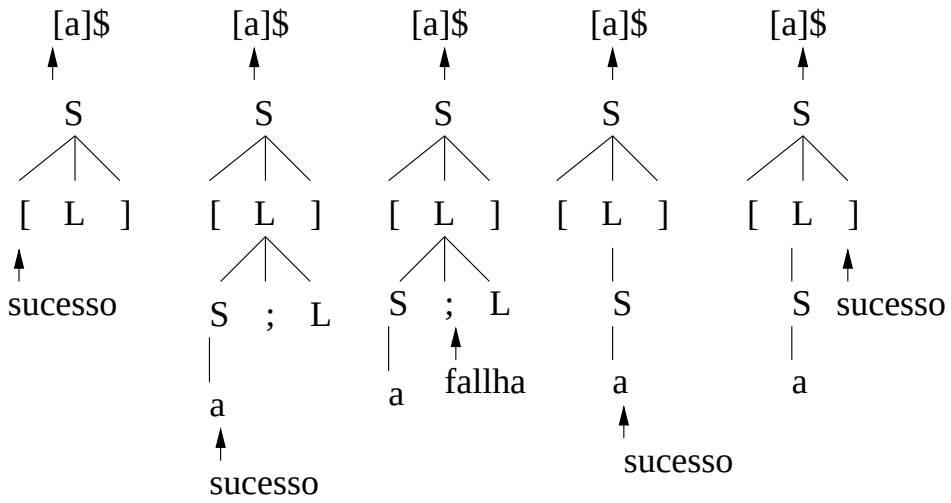
- Determinísticos
  - Implementação sem back-track (determinismo)
  - Limitam a classe de GLC que pode ser analisada
  - Algoritmos eficientes - complexidade linear (espaço requerido proporcional ao tamanho da gramática e tempo de análise proporcional ao tamanho da sentença)
  - Parser's automatizáveis

Análise Sintática Top-Down

- Também chamada de Análise Sintática Descendente
- Objetivo: encontrar uma derivação mais à esquerda para uma cadeia de entrada, contruindo a árvore gramatical a partir da raiz, criando os nós da árvore gramatical em pré-ordem (busca em profundidade).

Análise Sintática Top-Down

- Analisador Sintático Recursivo com Retrocesso
  - expande a árvore sempre pelo não-teminal mais à esquerda
  - quando existe mais de uma regra de produção para o não-terminal, todas as alternativas são testadas até se obtenha sucesso ou ocorra uma falha
  - Exemplo: G = ({S,L}, {a, ;, [,]}, S, P)
    onde P = { S ::= a | [L]
    L ::= S; L | S }



Recursivo com Retrocesso

- Implementação:
  - Consiste em um conjunto de procedimentos, um para cada não-teminal da gramática
  - A execução começa com a ativação do procedimento referente ao símbolo inicial da gramática
  - O procedimento pára e retorna sucesso se toda a cadeia de entrada puder ser derivada
  - As produções são escolhidas de modo arbitrário

Recursivo com Retrocesso

- Exemplo de procedimento para um não-terminal A

  Procedimento $A()$

  1. Escolha uma produção-A, $A ::= X_1 X_2 ... X_n$
  2. Para $i := 1$ até n faça
     - Se (X_i é um não-terminal) então ativa procedimento $X_i()$;
     - Senão Se (X_i for igual ao símbolo de entrada a)
       então avance para o próximo símbolo de entrada
     - Senão retorne erro

Recursivo com Retrocesso

- Analisadores Sintáticos Recursivos com Retrocesso apresentam os seguintes problemas:
  - Uma gramática recursiva à esquerda pode levar o analisador sintático de descendência recursiva, mesmo com retrocesso a um laço infinito (ou seja, exandir o não terminal pela 2^a vez sem consumir símbolos de entrada
  - O retrocesso leva a repetição da leitura de partes da sentença de entrada
    - tempo
    - necessidade de retroceder ações semânticas (armazenamento de informações na tabela de símbolos)
    - dificuldade para precisar onde ocorreu um erro
- Solução: Tratar a gramática para que identifique univocamente qual produção deve ser expandida

Análise Sintática Top-Down

- Analisador Sintático Preditivo LL(1)
  - Idéia: evitar o retrocesso fazendo com que o token identifique exatamente qual a produção que deve ser aplicada na expansão de um não-terminal
  - Olha adiante (1 símbolo) - First e Follow
  - É necessário:
    - que a gramática **não** seja recursiva à esquerda
    - seja fatorada à esquerda
    - ou seja, que os não-terminais com mais de uma regra de produção tenham os primeiros terminais deriváveis identificados univocamente

Analisador Sintático Preditivo

- Exemplo:

$$\begin{aligned} E &::= E \vee T | T \\ T &::= T \wedge F | F \\ F &::= \neg F | id \end{aligned}$$

- Eliminando a recursividade à esquerda:

$$\begin{aligned} E &::= TE' \\ E' &::= \vee TE' | \varepsilon \\ T &::= FT' \\ T' &::= \wedge FT' | \varepsilon \\ F &::= \neg F | id \end{aligned}$$

Analisador Sintático Preditivo

- FIRST e FOLLOW
  - Funções que auxiliam na construção de analisadores sintáticos descendentes e ascendentes
  - Objetivo: Inferir a partir da estrutrutura gramatical qual é a relação entre símbolos terminais e não-terminais
    - FIRST - conjunto de símbolos terminais que iniciam uma forma setencial
      - define quando qualquer produção pode ser usada
    - FOLLOW - conjunto de símbolos terminais que podem aparecer após um símbolo não-terminal em alguma forma sentencial
      - define quando uma produção do tipo $A ::= \varepsilon$ deve ser usada

FIRST

- Para todo $X \in (T \cup N)$, o FIRST(X) é obtido pela aplicação das seguintes regras: (até não haver mais terminais ou ε que possam ser acrescentados a algum dos conjuntos FIRST)
  1. Se $X \in T$ então FIRST(X) =$\{X\}$
  2. Se $X \in N$ então
     (a) Se $X ::= aY \in P$ então $a \in$ FIRST(X)
     (b) Se $X ::= \varepsilon \in P$, então $\varepsilon \in$ FIRST(X)
     (c) Se $X ::= Y_1 Y_2 ... Y_k \in P$, então FIRST($Y_1$) $\in$ FIRST(X)
        i. Se $\varepsilon \in$ FIRST(Y_1), então FIRST(Y_2) $\in$ FIRST(X)
        ii. Se $\varepsilon \in$ FIRST(Y_2), ...
        iii. Se $\varepsilon \in$ FIRST(Y_k) e ... e $\varepsilon \in$ FIRST(Y_1) , então $\varepsilon \in$ FIRST(X)

FIRST

- Exemplos:

$$S ::= Ab \mid ABc$$
$$B ::= bB \mid Ad \mid \varepsilon$$
$$A ::= aA \mid \varepsilon$$

$$\text{First}(S) = \{a, b, c, d\}$$
$$\text{First}(B) = \{a, b, d, \varepsilon\}$$
$$\text{First}(A) = \{a, \varepsilon\}$$

$$S ::= ABC$$
$$A ::= aA \mid \varepsilon$$
$$B ::= bB \mid ACd$$
$$C ::= cC \mid \varepsilon$$

$$\text{First}(S) = \{a, b, c, d\}$$
$$\text{First}(A) = \{a, \varepsilon\}$$
$$\text{First}(B) = \{b, a, c, d\}$$
$$\text{First}(C) = \{c, \varepsilon\}$$

FOLLOW

- Para calcular o FOLLOW(A) para todos os não-terminais A, aplique as seguintes regras, até que nada mais possa ser acrescentado a nenhum dos conjuntos FOLLOW
  1. Se S é o símbolo inicial da gramática, então $\$ \in$ FOLLOW(S)
  2. Se $A ::= \alpha B \beta \in P$ e $\beta \neq \varepsilon$, então adicione FIRST(β) em FOLLOW(B)
  3. Se $A ::= \alpha B$ (ou $A ::= \alpha B \beta$, onde $\varepsilon \in$ FIRST(β)) $\in P$, então adicione FOLLOW(A) em FOLLOW(B)
     - FIRST(β) $\rightarrow$ FIRST(da sequência β)

FOLLOW

- Exemplos:

$$S ::= Ab \mid ABc$$
$$B ::= bB \mid Ad \mid \varepsilon$$
$$A ::= aA \mid \varepsilon$$

$$\text{Follow}(S) = \{\$\}$$
$$\text{Follow}(B) = \{c\}$$
$$\text{Follow}(A) = \{b, a, d, c\}$$

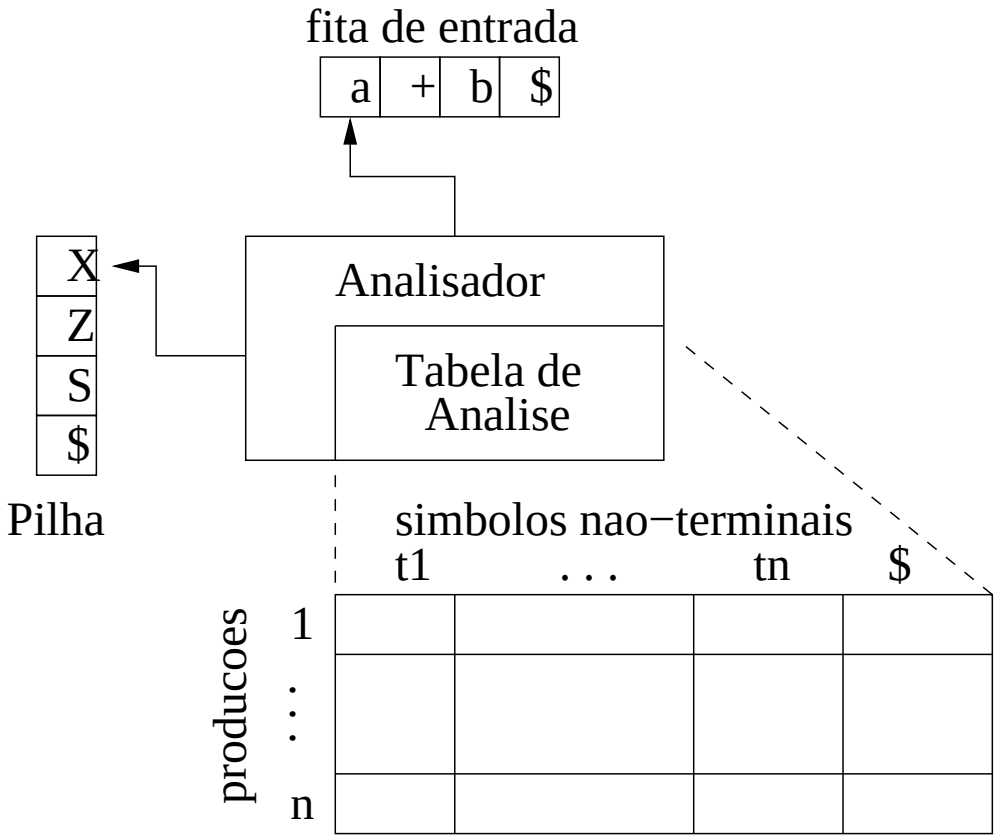
$$S ::= ABC$$
$$A ::= aA \mid \varepsilon$$
$$B ::= bB \mid ACd$$
$$C ::= cC \mid \varepsilon$$

$$\text{Follow}(S) = \{\$\}$$
$$\text{Follow}(A) = \{b, a, c, d\}$$
$$\text{Follow}(B) = \{c, \$\}$$
$$\text{Follow}(C) = \{d, \$\}$$

Análise Sintática Top-Down

- Analisador Sintático Preditivo
  - Idéia: eliminar a recursão (pilha implícita) utilizando uma pilha explícita (autômato de pilha)
  - O autômato de pilha é controlado por uma **tabela de análise**

Analisador Sintático Preditivo



Analisador Sintático Preditivo

- A cada passo o analisador executa 3 possíveis ações
  1. Se os símbolos no topo da pilha e sob o cabeçote de leitura forem iguais a $\$$ (fim de sentença), então pára aceitando a sentença de entrada $X = a = \$$
  2. Se o símbolo no topo da pilha for igual ao símbolo sob o cabeçote de leitura mas diferente de $\$$ então desempilha o topo e avança o cabeçote de leitura para o próximo símbolo na fita de entrada $X = a \neq \$$
  3. Se o símbolo no topo da pilha for um não-terminal, o programa consulta a entrada refente na tabela sintática M. Essa entrada será uma produção da gramática ou uma entrada de erro. Se for uma produção, substitui o topo pelo corpo da produção. $M[X, a] = \{X ::= UVW\}$ desempilha X e empilha W, V, U (U é o topo)

Analisador Sintático Preditivo

- Analisador Sintático Preditivo
  - O reconhecimento preditivo é a determinação da produção a ser aplicada, cujo lado direito irá substituir o símbolo não-terminal que se encontra no topo da pilha
  - O analisador busca a produção a ser aplicada na tabela de análise, levando em conta o não-terminal no topo da pilha e o token sob o cabeçote de leitura

Analisador Sintático Preditivo

- Exemplo:

$$\begin{aligned} E &::= E \vee T | T \\ T &::= T \wedge F | F \\ F &::= \neg F | id \end{aligned}$$

- Eliminando a recursividade à esquerda:

$$\begin{aligned} 1: \quad E &::= TE' \\ 2,3: \quad E' &::= \vee TE' | \varepsilon \\ 4: \quad T &::= FT' \\ 5,6: \quad T' &::= \wedge FT' | \varepsilon \\ 7,8: \quad F &::= \neg F | id \end{aligned}$$

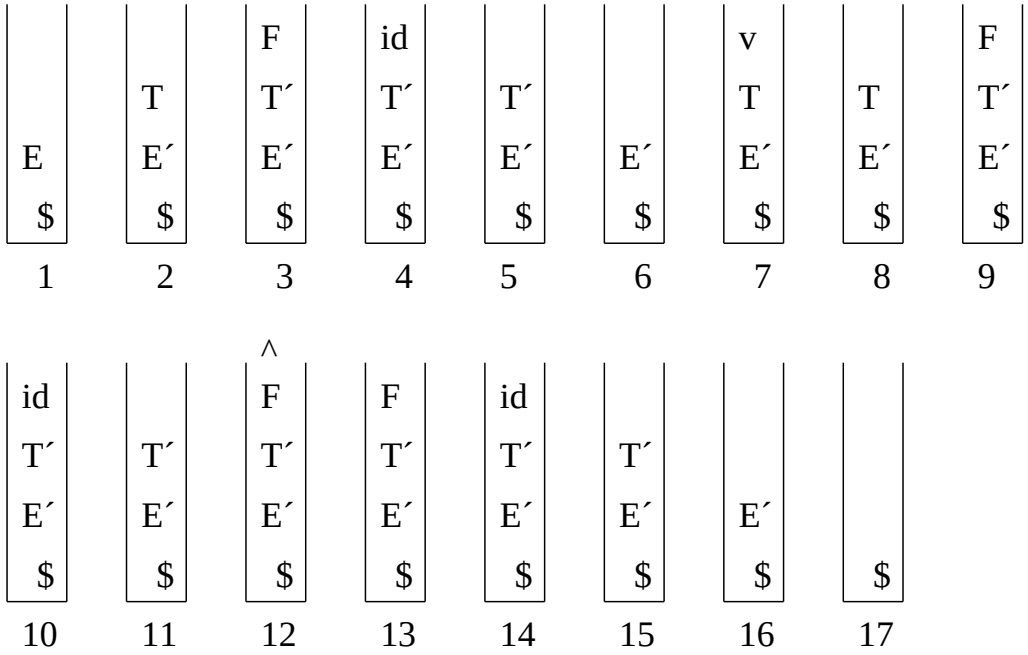
Analisador Sintático Preditivo

- Tabela de Análise:

	id	$\vee$	$\wedge$	$\neg$	\$
E	$1 : TE'$	/	/	$1 : TE'$	/
E'	/	$2 : \vee TE'$	/	/	$3 : \varepsilon$
T	$4 : FT'$	/	/	$4 : FT'$	/
T'	/	$6 : \varepsilon$	$5 : \wedge FT'$	/	$6 : \varepsilon$
F	$7 : id$	/	/	$8 : \neg F$	/

Analisador Sintático Preditivo

- A pilha após cada passo do algoritmo (para a entrada $id \vee id \wedge id$):



Analisador Sintático Preditivo

- Construção da Tabela de Análise
  - Condição: A gramática deve ser LL(1)
    - sentenças geradas pela gramática são passíveis de ser analisadas da esquerda para a direita (**L**eft to right) produzindo uma derivação mais à esquerda (**L**eftmost derivation) levando em conta apenas **1** símbolo de entrada
    - Para isto a gramática precisa:
      1. Não ser recursiva à esquerda
      2. Estar fatorada
      3. Para todo $A \in N$ tal que $A \stackrel{*}{\Rightarrow} \varepsilon$, First($A$) $\cap$ Follow(A) $= \emptyset$

Analisador Sintático Preditivo

- Algoritmo: construção da Tabela de Análise
  Entrada GLC $G = (N, T, P, S)$
  Saída Tabela de Análise $M[N, T + 1]$

  Para cada produção $A ::= \alpha$ de G faça

  1. Para cada terminal a em FIRST(α), inclua $A ::= \alpha$ em $M[A, a]$
  2. Se ε pertence a FIRST(α)
     (a) inclua $A ::= \alpha$ em $M[A, b]$ para cada terminal b em FOLLOW(A)
  3. Para toda a entrada $M[A, a]$ que não tiver produção assinalada, defina $M[A, a]$ como **erro**. (entrada vazia na tabela)

Analisador Sintático Preditivo

- Exemplo

$$
\begin{aligned}
1: &\quad E ::= && TE' \\
2,3: &\quad E' ::= && \vee TE' | \varepsilon \\
4: &\quad T ::= && FT' \\
5,6: &\quad T' ::= && \wedge FT' | \varepsilon \\
7,8: &\quad F ::= && \neg F | id
\end{aligned}
$$

	id	$\vee$	$\wedge$	$\neg$	\$
E	1	/	/	1	/
E'	/	2	/	/	3
T	4	/	/	4	/
T'	/	6	5	/	6
F	7	/	/	8	/

Análise Sintática Bottom-up

- Na Análise Sintática Bottom-up (Ascendente ou Redutiva) a árvore de derivação é construída a partir das folhas (sentença de entrada)
- A sentença é **reduzida** ao símbolo inicial da gramática
  - Equivale a fazer uma derivação mais a direita invertida

Reduções

- Em cada passo da redução, uma subcadeia específica, casando com o lado direito de uma produção é substituida pelo não terminal na cabeça da produção
- As principais decisões relacionadas com a análise ascendente em cada passo do reconhecimento são:
  - determinar quando reduzir
  - determinar a produção a ser aplicada para que a análise prossiga

Reduções

$$E ::= E + T \mid E - T \mid T$$

$$T ::= T * F \mid T/F \mid F$$

$$F ::= (E) \mid id$$

- $id * id \rightarrow F * id \rightarrow T * id \rightarrow T * F \rightarrow T \rightarrow E$

Poda do Handle

- um “handle” de uma cadeia de símbolos é uma subcadeia que casa com o corpo de uma produção, e cuja redução para o não-terminal do lado esquerdo representa um passo da derivação à direita ao inverso.

Forma Setencial à direita	handle	Produção de Redução
$id_1 * id_2$	id_1	$F \rightarrow id$
$F * id_2$	F	$T \rightarrow F$
$T * id_2$	id_2	$F \rightarrow id$
$T * F$	$T * F$	$E \rightarrow T * F$

Tipos de Analisadores

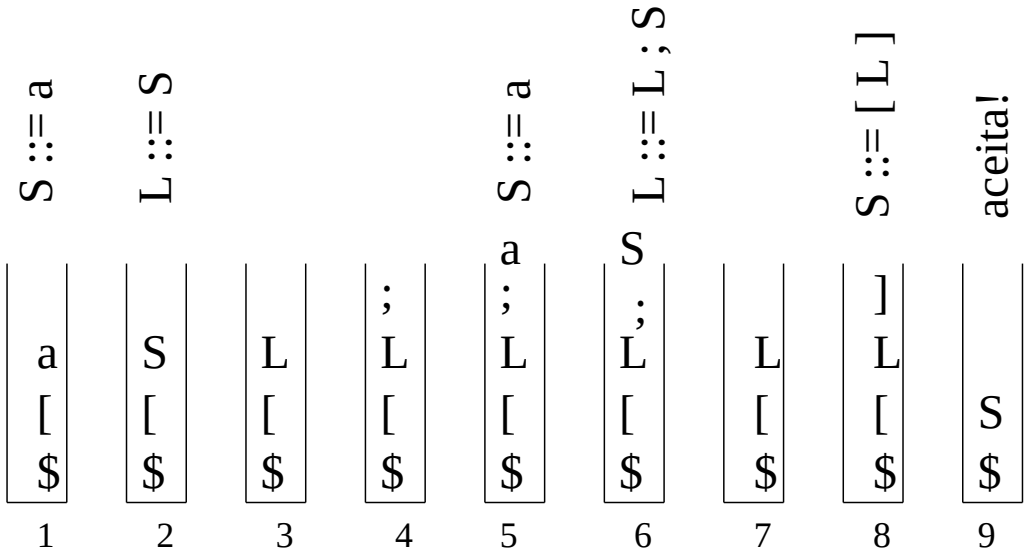
- Shift-Reduce
- Família LR
  - SLR - Simple LR
  - LALR - Lookahead LR
  - LR Canônicos

Analisador Sintático Shift-Reduce

- Idéia: Substituir o lado direito da produção (corpo) pelo não-terminal correspondente (cabeça)
- Implementação: Usa autômatos de pilha (a entrada é a sentença do texto seguida de $ e a pilha contém somente o símbolo $
- Processo: empilha a sentença de entrada até que se tenha na pilha o lado direito de alguma produção. Então, este é substituído pelo símbolo não-terminal (lado esquerdo da produção). Repete-se o processo até que a sentença de entrada seja totalmente lida e a pilha fique reduzida ao símbolo inicial da gramática ou até que um erro seja detectado
- A derivação é do tipo **rm** (Rightmost derivation)

Analisador Sintático Shift-Reduce

- Exemplo: G = ({S,L}, {a, ;, [,]}, S, P) onde P = { S ::= a | [L]
  L ::= S; L | S }
- Sentença de entrada: $[a; a]\$$



- Chamamos de **handle** a sequência de símbolos que corresponde à definição de um não-terminal

Analisador Sintático Shift-Reduce

- Um analisador Shift-Reduce pode realizar 4 possíveis ações:
  - Shift - transfere o próximo símbolo de entrada para o topo da pilha
  - Reduce - o extremo direito da cadeia a ser reduzida deve estar no topo da pilha. Localize o extremo esquerno da cadeia no interior da pilha e decida por qual não terminal esta cadeia será substituída.
  - Accept - Anuncia o término bem-sucedido da análise
  - Error - Ao descobrir um erro de sintaxe chame uma rotina de recuperação de erro ou aborte

Analisadores Sintáticos LR(K)

- “LR" significa: Left to right with a Rightmost derivation
- (k) significa: considerando **k** símbolos sob o cabeçote de leitura
- Tipos de analisadores LR
  - SLR (Simple LR) - Fáceis de implementar, porém aplicáveis a uma classe restrita de gramáticas;
  - LR Canônicos - mais poderoso e mais caro, aplicados a um grande número de linguagens livres de contexto;
  - LALR (Lookahead LR) - poder e custo intermediário, porém funciona para a maioria das gramáticas de linguagens de programação

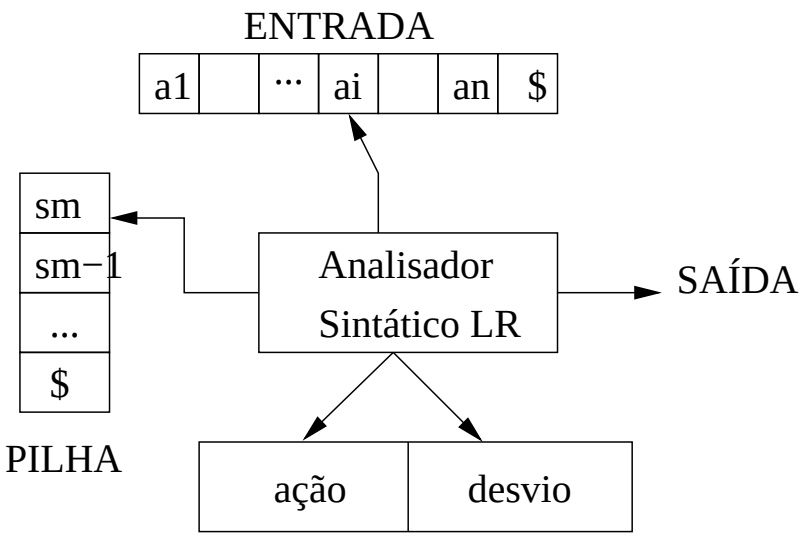
Analisadores Sintáticos LR(K)

- Vantagens:
  - reconhecem, praticamente, todas as construções sintáticas definidas por gramáticas livres de contexto da maioria das linguagens de programação
  - o método LR evita o retrocesso do método de análise shift reduce
  - possibilita a dectecção de erros tão cedo quanto possível numa varredura da entrada da esquerda para a direita
  - A classe de gramáticas que podem ser reconhecidas usando métodos LR é um superconjunto próprio da classe de gramáticas que podem ser reconhecidas com os métodos preditivos ou LL
    - Para uma gramática ser LR(k), ela deve ser capaz de reconhecer a ocorrência do lado direito de uma produção em uma forma sentencial mais à direita, com k símbolos à frente na entrada
    - Esse requisito é muito menos rigoroso do que aquele para as gramáticas LL(k)

Analisadores Sintáticos LR(K)

- Desvantagens:
  - Difíceis de implementar manualmente
  - Uso de uma ferramenta especializada - Yacc
    - o gerador de analisador sintático recebe a gramática livre de contexto e produz automaticamente como saída um analisador sintático para essa gramática.
    - se a gramática é ambígua ou possui outras construções difíceis de analisar, então o gerador localiza essas construções e disponibiliza mensagens com diagnósticos apropriados

Funcionamento dos Analisadores LR



- Consiste em: uma entrada, uma saída, uma pilha, um algoritmo de análise sintática e uma tabela sintática que possui duas partes (ação e desvio)
- O algoritmo é o mesmo para todos os tipos de analisadores LR, o que varia é a tabela

Funcionamento dos Analisadores LR

- O algoritmo de análise lê caracteres de um buffer de entrada, e ao contrário do A.S. Shift-Reduce que transfere um símbolo para a pilha, um A.S. LR transfere um *estado*
- Cada *estado* resume a informação contida na pilha abaixo dele
- A pilha contém uma sequência de estados $s_0, s_1, ..., s_m$

Funcionamento dos Analisadores LR

- O algoritmo do analisador LR se comporta como segue:
  - dado s_m (estado corrente da pilha) e a_i (símbolo corrente de entrada), consulta ação[s_m, a_i], que pode ter um dos quatro seguintes valores:
    - empilhar s_j : empilha o estado j e avança (o estado j representa a na pilha)
    - reduzir $A ::= \beta$: desempilha $\mid \beta \mid$, seja s o estado no topo da pilha, desvio[s, A];
    - aceitar: aceita a entrada e termina a análise, ou
    - erro

Funcionamento dos Analisadores LR

- Exemplo:

$$(1) E ::= E + T$$
$$(2) E ::= T$$
$$(3) T ::= T * F$$
$$(4) T ::= F$$
$$(5) F ::= (E)$$
$$(6) F ::= id$$

- O cógigo para cada ação da tabela é :
  - s_i significa empilhar o estado i
  - r_j significa reduzir através da produção de número j
  - acc significa aceitar
  - entrada vazia significa um erro

Funcionamento dos Analisadores LR

Estado	ação						desvio		
	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		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Funcionamento dos Analisadores LR

	Pilha	Símbolo	entrada	ação
(1)	0		$id * id + id\$$	empilhar 5 e avança
(2)	05	id	$*id + id\$$	reduzir por F::= id
(3)	03	F	$*id + id\$$	reduzir por T:: F
(4)	02	T	$*id + id\$$	empilhar 7 e avança
(5)	027	$T*$	$id + id\$$	empilhar 5 e avança
(6)	0275	$T * id$	$+id\$$	reduzir por F::= id
(7)	02710	$T * F$	$+id\$$	reduzir por T::= T * F
(8)	02	T	$+id\$$	reduzir por E::= T
(9)	01	E	$+id\$$	empilhar 6 e avança
(10)	016	$E+$	$id\$$	empilhar 5 e avança
(11)	0165	$E + id$	$\$$	reduzir por F::= id
(12)	0163	$E + F$	$\$$	reduzir por T::= F
(13)	0169	$E + T$	$\$$	reduzir por E::= E + T
(14)	01	E	$\$$	aceitar

Construindo a tabela de análise SLR

- Itens e Autômatos LR(0)
  - Um **item LR(0)** de uma gramática G é uma produção de G com um ponto em alguma posição do seu lado direito.
  - A produção $A \to XYZ$ gera quatro itens:

$$
\begin{aligned}
A \to & \quad \cdot XYZ \\
A \to & \quad X \cdot YZ \\
A \to & \quad XY \cdot Z \\
A \to & \quad XYZ \cdot
\end{aligned}
$$

  - A produção $A \to \varepsilon$ gera apenas um item $A \to \cdot$

Itens LR(0)

- Intuitivamente um item indica quanto de uma produção já foi vista em determinado ponto no processo de reconhecimento sintático
  - O item $A \rightarrow \cdot XYZ$ indica o início da busca por uma cadeia derivável de XYZ na entrada.
  - O item $A \rightarrow X \cdot YZ$ indica que no ponto atual onde se encontra a análise, uma cadeira derivável de X já foi encontrada e que espera-se em seguida ver uma cadeia derivável de YZ
  - O item $A \rightarrow XYZ \cdot$ 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

Itens LR(0)

- Uma coleção de conjuntos de itens LR(0), chamada **coleção LR(0) Canônica**, oferece a base para a construção de um autômato finito determinístico que é usado para dirigir as decisões durante a análise
- Esse AFD é chamado de autômato LR(0)
  - Cada estado do autômato representa um conjunto de itens na coleção LR(0) Canônica

Coleção LR(0) Canônica

- Para construir a coleção LR Canônica para uma gramática, define-se uma gramática estendida e duas funções (fechamento ou closure e transição ou goto)
- Se G é uma gramática com símbolo inicial S, então G' é uma gramática estendida para G com um novo símbolo inicial S' e a produção S' $\rightarrow$ S (esta nova produção serve para identificar quando o reconhecedor sintático deve parar e anunciar a aceitação da cadeia de entrada, pela redução de S' $\rightarrow$ S)

Coleção LR(0) Canônica

- Se I é um conjunto de itens para uma gramática G, então CLOSURE(I) é o conjunto 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 CLOSURE (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. Aplique essa regra até que nenhum outro item possa ser incluído no CLOSURE(I)

Coleção LR(0) Canônica

- Exemplo:

$$E' \to E$$
$$E \to E + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (E) \mid id$$

CLOSURE

- Exemplo: se I é o conjunto de um item $\{[E' \rightarrow \cdot E]\}$, então CLOSURE($I$) contém o conjunto de itens I_0:

$$E' \rightarrow \cdot E$$
$$E \rightarrow \cdot E + T$$
$$E \rightarrow \cdot T$$
$$T \rightarrow \cdot T * F$$
$$T \rightarrow \cdot F$$
$$F \rightarrow \cdot (E)$$
$$F \rightarrow \cdot id$$

CLOSURE

SetOfItems CLOSURE(I) {
    $J = I$;
    repeat
        for (each item $A \to \alpha \cdot B \beta$ in J)
            for (each production $B \to \gamma$ of G)
                if ($B \to \cdot \gamma$ is not in J)
                    add $B \to \cdot \gamma$ to J;
    until no more items are added to J on one round;
    return J;
}

Coleção LR(0) Canônica

- A segunda função usada na construção da coleção LR (0) Canônica é a função GOTO(I, X) onde I é o conjunto de itens e X é um símbolo da gramática
- GOTO(I,X) é definido como o fechamento do conjunto de todos os itens [$A \to \alpha X \cdot \beta$] tais que [$A \to \alpha \cdot X \beta$] está em I
- Intuitivamente define as transições no autômato LR(0) para a gramática

GOTO

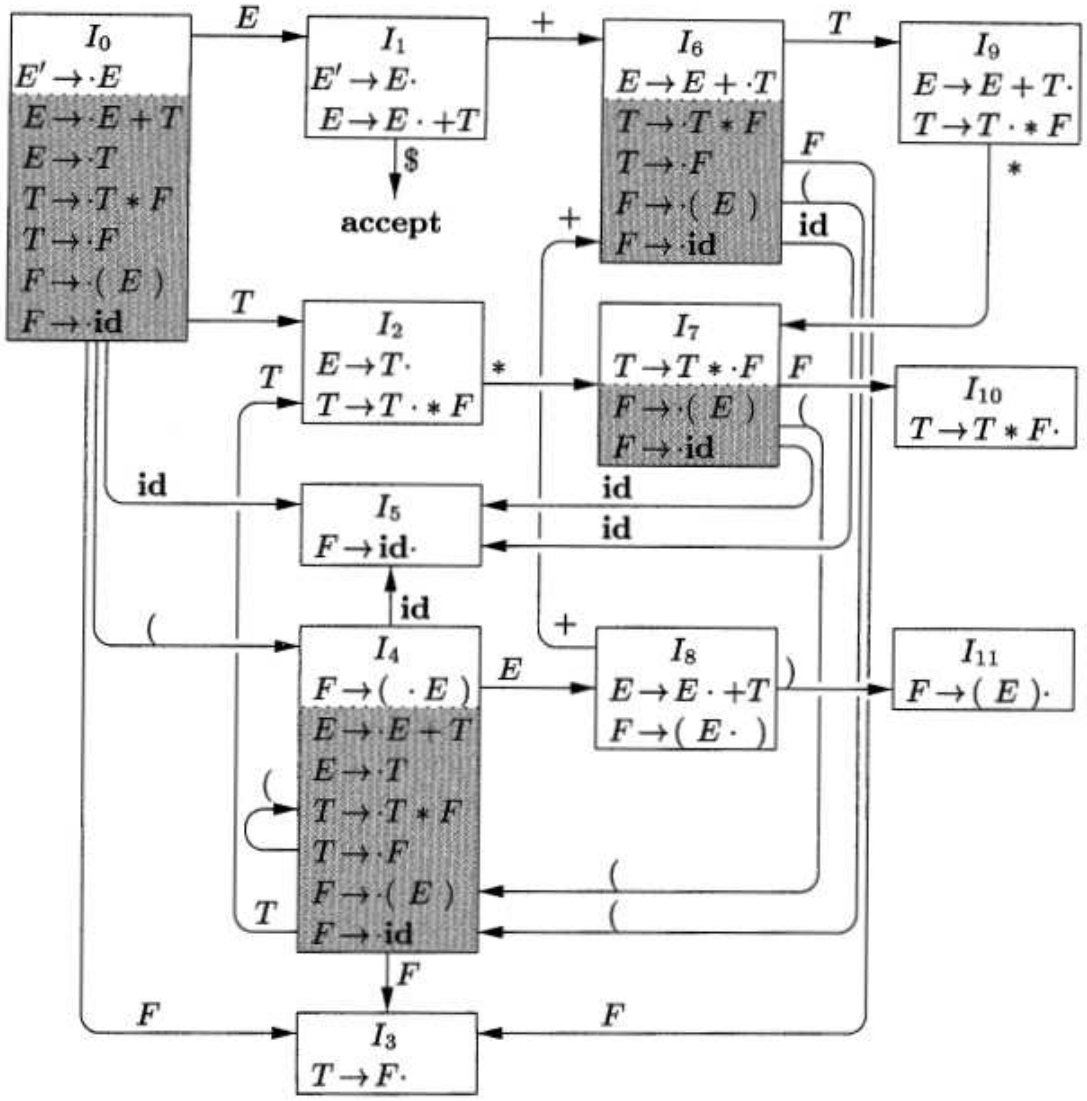
- Exemplo: Se I representa o conjunto com dois itens $\{[E' \to E\cdot], [E \to E \cdot +T]\}$, então GOTO($I$, $+$) contém os itens:

$$E \to E + \cdot T$$
$$T \to \cdot T * F$$
$$T \to \cdot F$$
$$F \to \cdot(E)$$
$$F \to \cdot id$$

Coleção LR(0) Canônica

```
void items(G') {
        C = CLOSURE({[S' → ·S]});
        repeat
                for ( each set of items I in C )
                        for ( each grammar symbol X )
                                if ( GOTO(I, X) is not empty and not in C )
                                        add GOTO(I, X) to C;
        until no new sets of items are added to C on a round;
}
```

Autômatos LR(0)



Construção da Tabela SLR

- O método SLR começa com os itens LR(0) e autômatos LR(0) como visto
  - Estender a gramática G para produzir G'
  - A partir de G', constrói-se C, a coleção Canônica de conjuntos de itens para G' junto com a função GOTO
  - Para construir a tabela é necessário ainda conhecer FOLLOW(A) par cada não terminal A de G

Construção da Tabela SLR

Algorithm 4.46: Constructing an SLR-parsing table.

INPUT: An augmented grammar G'.

OUTPUT: The SLR-parsing table functions ACTION and GOTO for G'.

METHOD:

1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(0) items for G'.

2. State i is constructed from I_i. The parsing actions for state i are determined as follows:

   (a) If $[A \to \alpha \cdot a\beta]$ is in I_i and $\text{GOTO}(I_i, a) = I_j$, then set $\text{ACTION}[i, a]$ to "shift j." Here a must be a terminal.

   (b) If $[A \to \alpha \cdot]$ is in I_i, then set $\text{ACTION}[i, a]$ to "reduce $A \to \alpha$" for all a in $\text{FOLLOW}(A)$; here A may not be S'.

   (c) If $[S' \to S \cdot]$ is in I_i, then set $\text{ACTION}[i, \$]$ to "accept."

   If any conflicting actions result from the above rules, we say the grammar is not SLR(1). The algorithm fails to produce a parser in this case.

Construção da Tabela SLR

3. The goto transitions for state i are constructed for all nonterminals A using the rule: If $\text{GOTO}(I_i, A) = I_j$, then $\text{GOTO}[i, A] = j$.
4. All entries not defined by rules (2) and (3) are made "error."
5. The initial state of the parser is the one constructed from the set of items containing $[S' \to \cdot S]$.

Construção da Tabela SLR

Estado	ação						desvio		
	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		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Construção da Tabela SLR

- A tabela construída é chamada Tabela SLR(1) para G
- O analisador resultante é chamado de analisador SLR(1)

LR mais poderosos

- Incorpora nos Itens o primeiro símbolo ainda não lido da entrada - Lookahead
  - LR Canônico - Faz uso do Lookahead e usa uma tabela construída a partir do conjunto de itens denominados LR(1)
  - LALR - tabela construída a partir dos conjuntos LR(0) e possui menos estados que analisadores sintáticos típicos (baseados no conjunto Canônico LR(1))
- A introdução do Lookahead nos itens LR(0) dá mais poder ao método e o torna mais geral, permitindo tratar mais gramáticas que o método SLR
- As tabelas não são maiores que as geradas pelo SLR, o que torna o médoto prefirido na maioria das situações

Analisadores LR Canônicos

- No SLR, o estado i faz uma redução segundo a produção $A \to \alpha$ se no conjunto de itens I_i tiver o item $[A \to \alpha\cdot]$ e a estiver em FOLLOW(A).
- Objetivo: Contornar o problema do SLR quando o estado i aparece no topo da pilha, o prefixo viável $\beta\alpha$ na pilha é tal que βA não pode ser seguido por a em uma forma sentencial à direita

Analisadores LR Canônicos

- Exemplo: Exemplo de gramática não SLR(1)

$$S \rightarrow L = R \mid R$$
$$L \rightarrow *R \mid id$$
$$R \rightarrow L$$

Analisadores LR Canônicos

- Coleção Canônica de Itens:

$$
\begin{array}{lll}
I_0: & S' \to \cdot S & I_1: \quad S' \to S\cdot \\
 & S \to \cdot L = R & \\
 & S \to \cdot R & I_2: \quad S \to L\cdot = R \\
 & L \to \cdot * R & \quad\;\; R \to L\cdot \\
 & L \to \cdot id & \\
 & R \to \cdot L & I_3: \quad S \to R\cdot
\end{array}
$$

  - Há um conflito shif/reduce na tabela para a entrada $[2, =] = \{s_6, r_3\}$

Analisadores LR Canônicos

- É possível incorporar mais informações no estado para auxiliar na remoção de algumas dessas reduções inválidas por $A \rightarrow \alpha$.
- A informação extra é incorporada ao estado redefinindo-se os itens para incluir um símbolo terminal como um segundo componente (lockahead)
- O item, chamado de LR(1), assume a forma $[A \rightarrow \alpha \cdot \beta, a]$ onde a é um terminal ou marcador de final de sentença
- O lockahead só tem efeito para produções onde β é igual a ε, nestas, uma redução pela produção $A \rightarrow \alpha$ se o próximo símbolo da entrada for a

Analisadores LR Canônicos

- Para construir a coleção canônica de conjuntos LR(1) válidos, são modificados os procedimentos CLOSURE e GOTO:

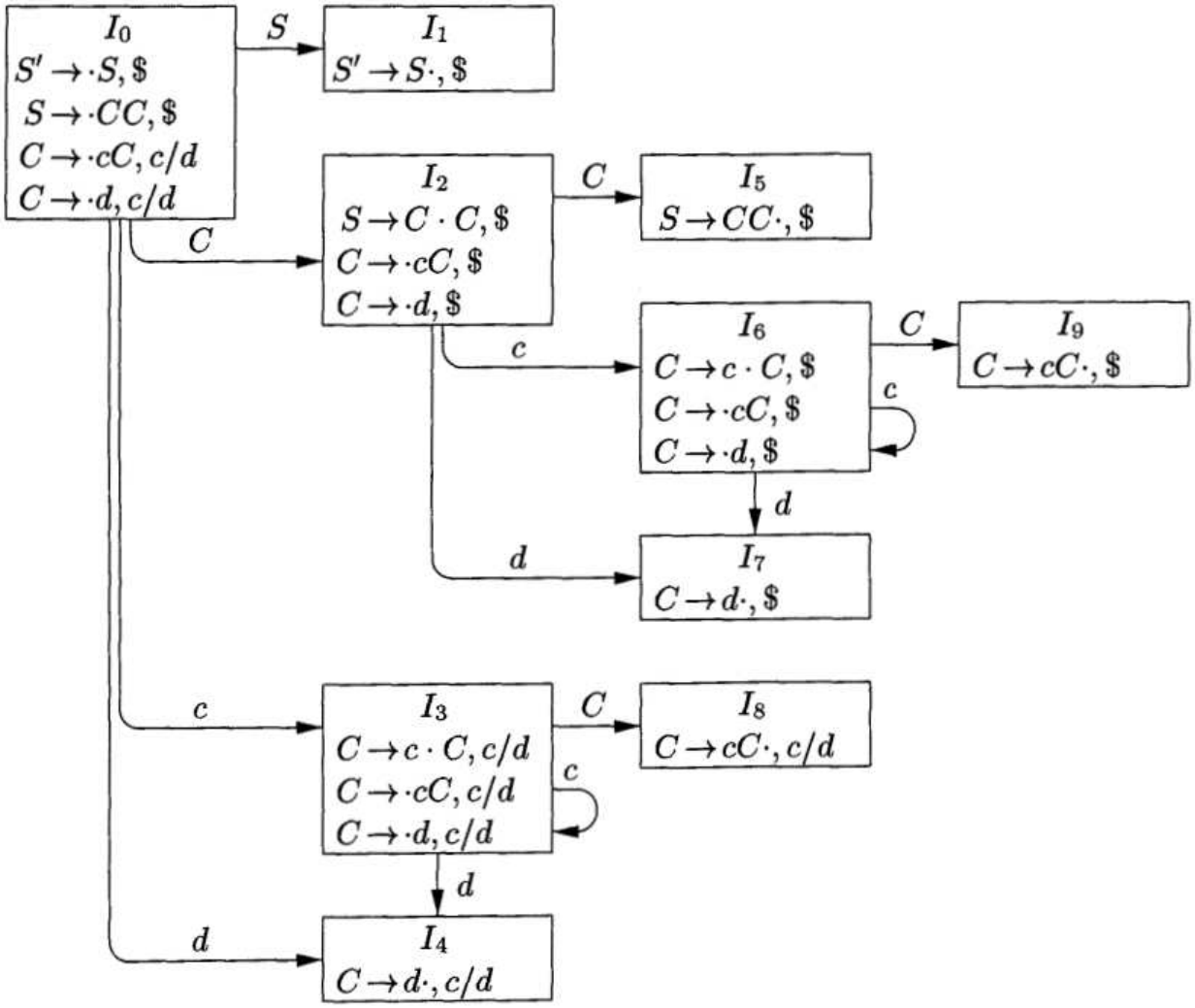
Analisadores LR Canônicos

```
SetOfItems CLOSURE(I) {
        repeat
                for ( each item [A → α·Bβ, a] in I )
                        for ( each production B → γ in G' )
                                for ( each terminal b in FIRST(βa) )
                                        add [B → ·γ, b] to set I;
        until no more items are added to I;
        return I;
}

SetOfItems GOTO(I, X) {
        initialize J to be the empty set;
        for ( each item [A → α·Xβ, a] in I )
                add item [A → αX·β, a] to set J;
        return CLOSURE(J);
}

void items(G') {
        initialize C to CLOSURE({[S' → ·S, $]});
        repeat
                for ( each set of items I in C )
                        for ( each grammar symbol X )
                                if ( GOTO(I, X) is not empty and not in C )
                                        add GOTO(I, X) to C;
        until no new sets of items are added to C;
}
```

Analisadores LR Canônicos



Analisadores LR Canônicos

Algorithm 4.56: Construction of canonical-LR parsing tables.

INPUT: An augmented grammar G'.

OUTPUT: The canonical-LR parsing table functions ACTION and GOTO for G'.

METHOD:

1. Construct $C' = \{I_0, I_1, \cdots, I_n\}$, the collection of sets of LR(1) items for G'.

2. State i of the parser is constructed from I_i. The parsing action for state i is determined as follows.

   (a) If $[A \to \alpha \cdot a\beta, b]$ is in I_i and GOTO$(I_i, a) = I_j$, then set ACTION$[i, a]$ to "shift j." Here a must be a terminal.

   (b) If $[A \to \alpha \cdot, a]$ is in I_i, $A \neq S'$, then set ACTION$[i, a]$ to "reduce $A \to \alpha$."

   (c) If $[S' \to S \cdot, \$]$ is in I_i, then set ACTION$[i, \$]$ to "accept."

   If any conflicting actions result from the above rules, we say the grammar is not LR(1). The algorithm fails to produce a parser in this case.

3. The goto transitions for state i are constructed for all nonterminals A using the rule: If GOTO$(I_i, A) = I_j$, then GOTO$[i, A] = j$.

4. All entries not defined by rules (2) and (3) are made "error."

5. The initial state of the parser is the one constructed from the set of items containing $[S' \to \cdot S, \$]$.

Analisadores LR Canônicos

STATE	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		

Analisadores LALR

- Método mais usado na prática pois suas tabelas são muito menores que as tabelas LR canônicas
- Além disso, construções sintáticas mais comuns das linguagens de programação podem ser expressas convenientemente por uma gramática LALR
  - o que nem sempre é verdade para gramáticas SLR, pois podem haver construções que não são reconhecidas

Analisadores LALR

- Em comparação com o número de estados, analisadores LALR tem o mesmo número de estados que analisadores SLR (algumas centenas de estados) enquanto que analisadores LR Canônicos tem alguns milhares de estados para uma linguagem como C, p.ex.
- logo, tabelas LALR e SLR são mais econômicas.

Analisadores LALR

- Motivação
  - Estados I_4 e I_7 do autômato LR(1) - dois estados que reduzem $C \to d$ com os símbolos $\{c, d, \$\}$
- Idéia:
  - Juntar os estados em um único I_{47} que reduz para qualquer entrada
  - Encontrar conjuntos de itens com mesmo núcleo (conjuntos onde os primeiros componentes dos pares sejam iguais)

Analisadores LALR

Algorithm 4.59: An easy, but space-consuming LALR table construction.

INPUT: An augmented grammar G'.

OUTPUT: The LALR parsing-table functions ACTION and GOTO for G'.

METHOD:

1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(1) items.

2. For each core present among the set of LR(1) items, find all sets having that core, and replace these sets by their union.

3. Let $C' = \{J_0, J_1, \ldots, J_m\}$ be the resulting sets of LR(1) items. The parsing actions for state i are constructed from J_i in the same manner as in Algorithm 4.56. If there is a parsing action conflict, the algorithm fails to produce a parser, and the grammar is said not to be LALR(1).

4. The GOTO table is constructed as follows. If J is the union of one or more sets of LR(1) items, that is, $J = I_1 \cap I_2 \cap \cdots \cap I_k$, then the cores of $\text{GOTO}(I_1, X)$, $\text{GOTO}(I_2, X), \ldots, \text{GOTO}(I_k, X)$ are the same, since $I_1, I_2, \ldots, I_k$ all have the same core. Let K be the union of all sets of items having the same core as $\text{GOTO}(I_1, X)$. Then $\text{GOTO}(J, X) = K$.

Analisadores LALR

STATE	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	r2	r2	r2		