# Compiladores

# Análise Sintática Ascendente

**Professor: Flávio Braga**

**Acadêmicos: Douglas Garcia Valle da Silva**

**Edilson Barini Tizolin**

**João Chicati Peres**

**Nelson Junior**

# Análise Sintática Ascendente

- Árvore de derivação à partir das folhas
- Shift-Reduce.
- Analisador Sintático LR à mão, é muito difícil.
- Existem ferramentas para auxiliarem nesta fase.

# Análise Sintática Ascendente

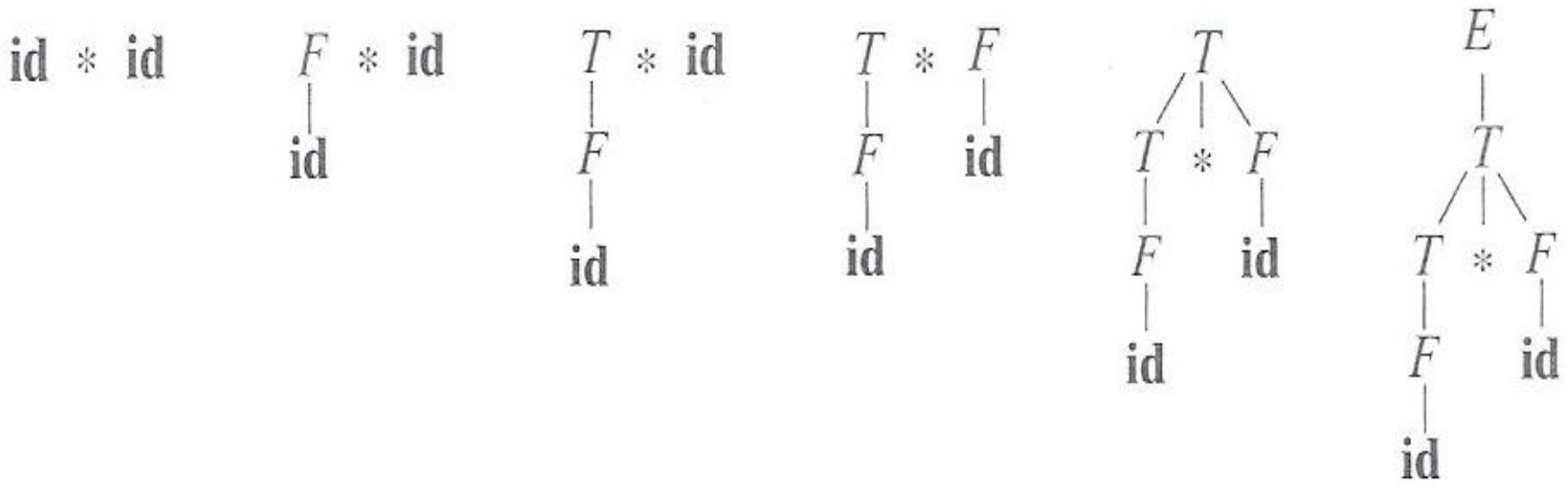



Figura 4.25 Uma análise ascendente para id * id.

# Reduções

- Podemos pensar na análise ascendente como o processo de "reduzir" uma cadeia *w* para o símbolo inicial da gramática.
- Principais decisões:
  - Determinar quando reduzir
  - Determinar a produção a ser usada

# Reduções

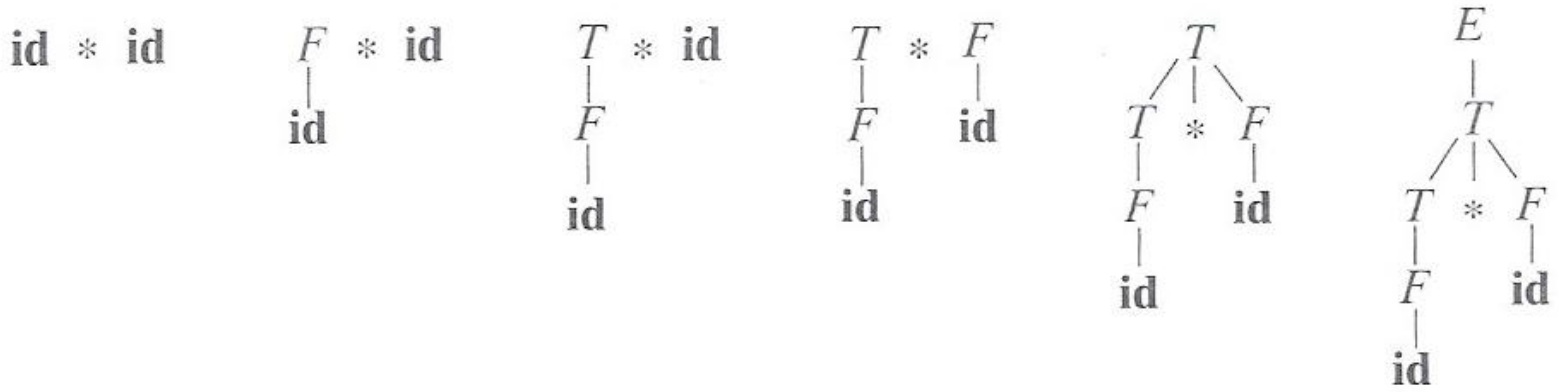



FIGURA 4.25 Uma análise ascendente para **id** * **id**.

- id * id, F * id, T * id, T * F, T, E
- E => T => T *F => T * id => F * id => id * id

# Poda do Handle

- Handle de uma uma cadeia de símbolos é uma subcadeia que casa com o corpo de uma produção.

| Forma Sentencial à Direita | Handle | Produção de Redução |
|---|---|---|
| $\mathbf{id}_1 * \mathbf{id}_2$ | $\mathbf{id}_1$ | $F \to \mathbf{id}$ |
| $F * \mathbf{id}_2$ | $F$ | $T \to F$ |
| $T * \mathbf{id}_2$ | $\mathbf{id}_2$ | $F \to \mathbf{id}$ |
| $T * F$ | $T * F$ | $E \to T * F$ |

Figura 4.26 Handles durante a análise de $\mathbf{id}_1 * \mathbf{id}_2$.

# Analisador Sintático Shift-Reduce

- Forma de análise ascendente
- Utiliza uma pilha
  - Inicialmente pilha vazia
  - Escansão da esquerda para a direta da cadeia de entrada
  - Analisado sintático transfere zero ou mais símbolos da entrada para a pilha.
  - O analisador repete este ciclo até detectar um erro ou até que a pilha contenha apenas o símbolo inicial da gramatica e a entrada vazia.

# Analisador Sintático Shift-Reduce

| Pilha | Entrada | Ação |
|---|---|---|
| $ | $\mathbf{id}_1 * \mathbf{id}_2$ $ | transfere |
| $ $\mathbf{id}_1$ | $* \mathbf{id}_2$ $ | reduz segundo $F \rightarrow \mathbf{id}$ |
| $ $F$ | $* \mathbf{id}_2$ $ | reduz segundo $T \rightarrow F$ |
| $ $T$ | $* \mathbf{id}_2$ $ | transfere |
| $ $T *$ | $\mathbf{id}_2$ $ | transfere |
| $ $T * \mathbf{id}_2$ | $ | reduz segundo $F \rightarrow \mathbf{id}$ |
| $ $T * F$ | $ | reduz segundo $T \rightarrow T * F$ |
| $ $T$ | $ | reduz segundo $E \rightarrow T$ |
| $ $E$ | $ | aceita |

FIGURA 4.28 Configurações de um analisador shift-reduce sob a entrada $\mathbf{id}_1 * \mathbf{id}_2$.

# Analisador Sintático Shift-Reduce

- Quatro ações possíves
  - **Shift:** Transfere o próximo símbolo da entrada para o topo da pilha
  - **Reduce:** O extremo direito da cadeia a ser reduzida deve estar no topo da pilha e decida 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.

# Conflitos durante a Análise Sintática Shift-Reduce

- Conflito Shift/Reduce
  - Mesmo conhecendo todo o conteúdo da pilha, e o próximo símbolo da entrada.
- Conflito Reduce/Reduce
  - Não consegue definir quais das várias reduções disponíveis escolher.

# Introdução à Análise LR Simples: SLR

- Atualmente analisadores sintáticos ascendentes é baseado em um conceito – reconhecedores LR(k).
  - L: escansão da entrada da esquerda para a direita
  - R: construção das derivações mais à direita ao reverso
  - k: símbolos à frente no fluxo de entrada que auxiliam nas decisões da análise
- SLR : "LR Simples" – Simple LR

# Por que Analisadores sintáticos LR

- Atrativos
  - Capacidade de reconhecimento de quase todas as construções sintáticas definidas pelas GLC.
  - Método shift-reduce sem retrocesso e implementação com mesmo grau de eficiência, espaço e tempo de outros métodos shift-reduce.
  - Detecta um erro sintático logo que aparece na cadeia de entrada.
  - Reconhece o superconjunto da classe de gramaticas decompostas pelos métodos preditivos.

# Por que Analisadores sintáticos LR

- Contraponto:
  - Construção do analisador LR é muito trabalhosa.
  - A solução seria um gerador de analisadores LR. Exs.: YACC.

# Itens e o Autômato LR(0)

- Um analisador sintático LR mantém estados para acompanhar onde se encontra a análise durante as ações shift-reduce.
- Estados representam conjunto de “itens”.
- Item LR(0) de uma gramática G é uma produção de G com um ponto em alguma posição do seu lado direito.

A→.XYZ A→X.YZ

A→XY.Z A→XYZ.

# Itens e o Autômato LR(0)

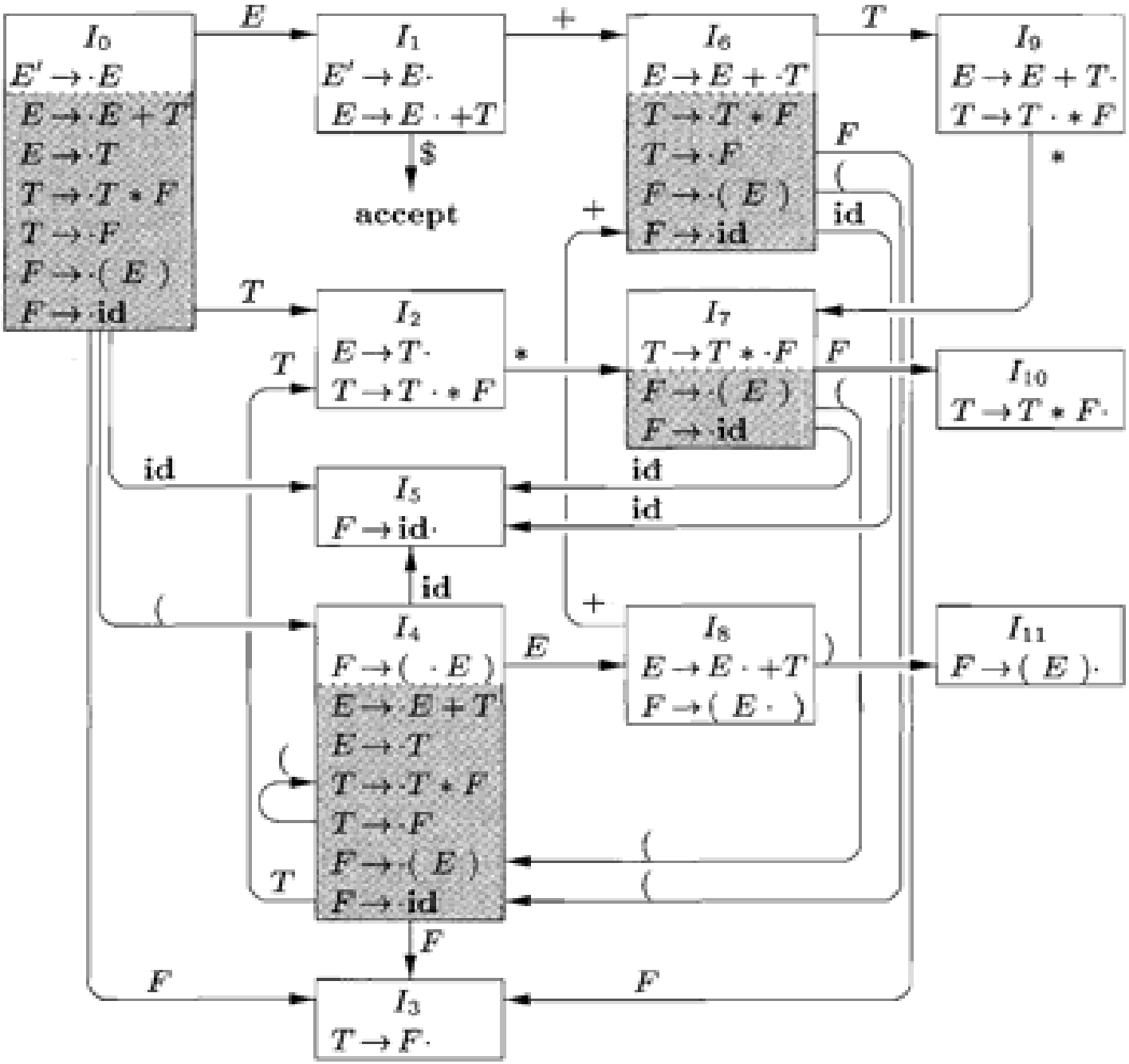
- Coleção LR(0) canônica, chamada coleção de conjuntos de itens LR(0), oferece base para construção de um AFD (autômato LR(0)), usado para dirigir decisões durante a análise.
- Para a seguinte gramatica:

$$E' \rightarrow E$$

$$E \rightarrow E+T \mid T$$

$$T \rightarrow T*F \mid F$$

$$F \rightarrow (E) \mid id$$


$I_0$
$E' \to \cdot E$
$E \to \cdot E + T$
$E \to \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot ( E )$
$F \to \cdot \mathbf{id}$
$I_1$
$E' \to E \cdot$
$E \to E \cdot + T$
$I_2$
$E \to T \cdot$
$T \to T \cdot * F$
$I_3$
$T \to F \cdot$
$I_4$
$F \to ( \cdot E )$
$E \to \cdot E + T$
$E \to \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot ( E )$
$F \to \cdot \mathbf{id}$
$I_5$
$F \to \mathbf{id} \cdot$
$I_6$
$E \to E + \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot ( E )$
$F \to \cdot \mathbf{id}$
$I_7$
$T \to T * \cdot F$
$F \to \cdot ( E )$
$F \to \cdot \mathbf{id}$
$I_8$
$E \to E \cdot + T$
$F \to ( E \cdot )$
$I_9$
$E \to E + T \cdot$
$T \to T \cdot * F$
$I_{10}$
$T \to T * F \cdot$
$I_{11}$
$F \to ( E ) \cdot$
accept
$E$
$T$
$F$
$+$
$*$
$($
$)$
$\mathbf{id}$
$\$$

# Construção do conjunto canônico LR(0) para uma gramatica

Duas operações:

- 1. Acrescentar à gramática a produção S'→S (em que S é o símbolo inicial da gramática)
  - Permite a identificação do fim da análise, mais especificamente, S'→S.
  - O analisador aceitará uma cadeia quando ocorrer uma redução pela regra adicionada S'→S
- 2. Computar as funções fechamento e transição para a nova gramática

# Fechamento de Conjunto de Itens

- Função fechamento:
- Seja I um conjunto de itens LR(0)
  - Todo item em I pertence ao fechamento (I)
  - Se A→α.Xβ está em FECHAMENTO(I) e B→.γ é uma produção, então adiciona-se B→.γ ao conjunto.

Em outras palavras:

- Inicializa o conjunto I com a regra inicial da gramática, colocando-se o indicador (.) no início da regra;
- Para cada regra no conjunto, adicionam-se as regras dos não terminais que aparecem precedidos pelo indicador (.)

# Fechamento de Conjunto de Itens

- Exemplo da aplicação de fechamento
- S' → S
- S → a | [L]
- L → L;S | S

- I={S → [.L]}
- FECHAMENTO(I)={S→[.L], L→.L;S, L→.S, S→.a, S→.[L]}

# A Função de transição

- Função transição:
  - GOTO(I,X): consiste em avançar o indicador (.) através do símbolo gramatical X das produções correspondentes em I e calcular a função fechamento para o novo conjunto.
- Exemplo da aplicação de transição
- I={S→[L.],  L→L.;S}
- transição (I,;)={L→L;.S,  S→.a,  S→.[L]}

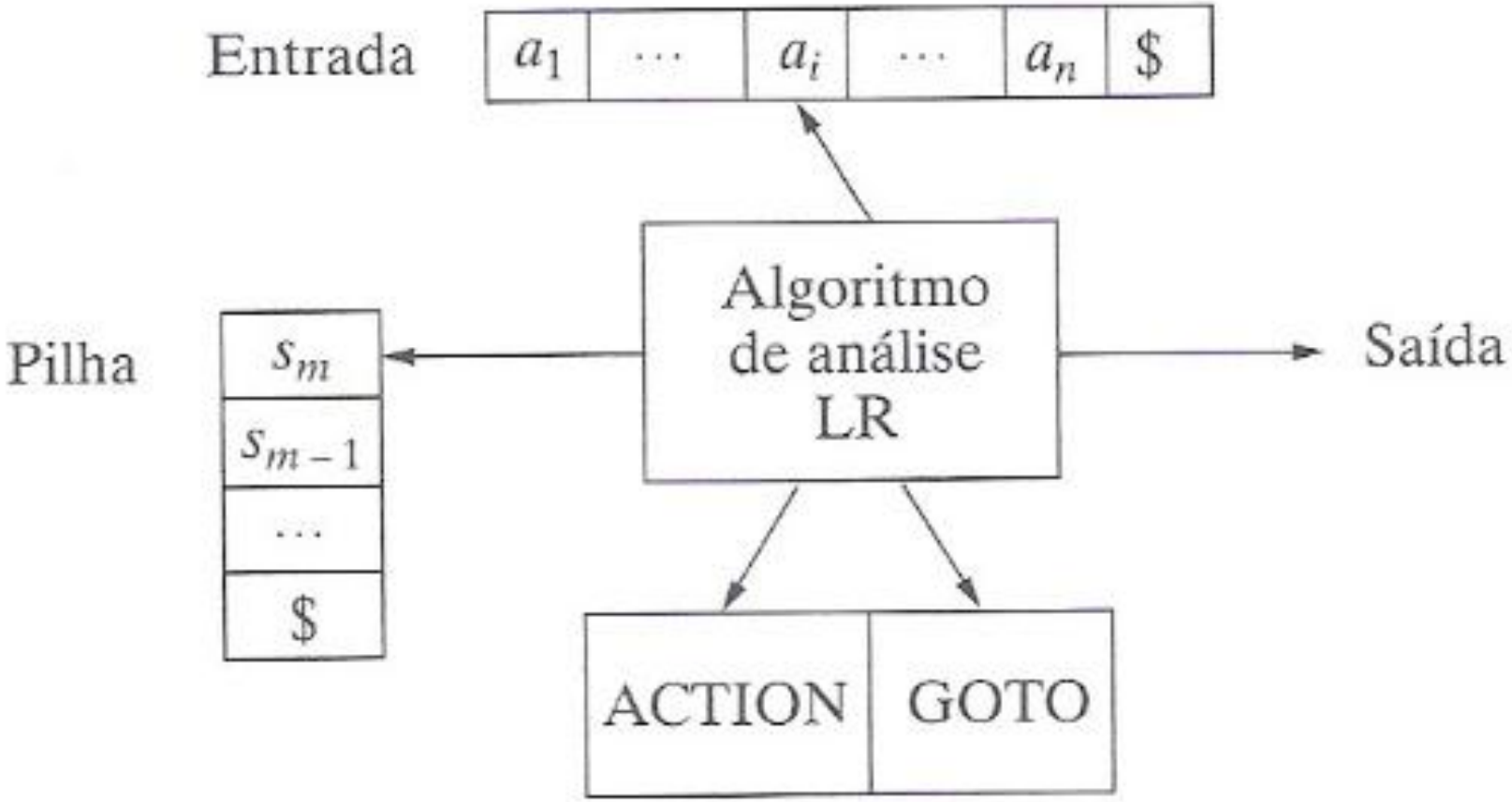
# Uso do Autômato LR(0)

- A idéia por trás do analisador SLR é a análise a partir do autômato LR(0).
- – Nesse autômato, os estados são conjuntos de itens da coleção canônica e as transições são dadas pela função GOTO
- O estado inicial do autômato LR é dado pelo fecho do estado inicial de G'. Um estado j referese ao estado correspondendo ao conjunto de itens $I_j$
- As decisões de shiftreduce são feitas da seguinte maneira:
- 1.Supor que uma string w de símbolos de G' inicia o autômato LR com uma transição do estado inicial 0 para o estado j.
- 2. Então, deslocase (shift) o próximo símbolo da entrada a se j tem uma transição com a.
- 3.Se não existe tal transição, faz a redução (reduce). Os itens que compõem j indicam qual produção usar na redução.

# Algoritmo para obter o conjunto canônico de itens LR(0)

```
void itens(G'){
    C=FECHAMENTO({[S' → .S]});
    repeat
        for (cada conjunto de itens I em C)
            for (cada símbolo de 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;
```

# Modelo de um Analisador Sintático LR



# O Algoritmo da Análise Sintática LR

Seja '$a$' o primeiro símbolo de w$;
**while** (1){ /* repita indefinidamente */
    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 → ß){
        desempilha símbolos |ß| da pilha;
        faça o estado t agora ser o topo da pilha;
        empilhe GOTO[t,A] na pilha;
        imprima a produção A → ß;
    }**else if** (ACTION[s,a]=accept) pare; /*a análise terminou */
    **else** chame uma rotima de recuperação de erro;
    }

# Comportamento do Analisador LR

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

| Estados | Ações | | | | | | Transições | | |
|---|---|---|---|---|---|---|---|---|---|
| | id | + | * | ( | ) | $ | E | T | F |
| 0 | e5 | | | e4 | | | 1 | 2 | 3 |
| 1 | | e6 | | | | OK | | | |
| 2 | | r2 | e7 | | r2 | r2 | | | |
| 3 | | r4 | r4 | | r4 | r4 | | | |
| 4 | e5 | | | e4 | | | 8 | 2 | 3 |
| 5 | | r6 | r6 | | r6 | r6 | | | |
| 6 | e5 | | | e4 | | | | 9 | 3 |
| 7 | e5 | | | e4 | | | | | 10 |
| 8 | | e6 | | | e11 | | | | |
| 9 | | r1 | e7 | | r1 | r1 | | | |
| 10 | | r3 | r3 | | r3 | r3 | | | |
| 11 | | r5 | r5 | | r5 | r5 | | | |

- $e_i$ “empilhar i”
- $r_i$ “reduzir pela regra i”

# Comportamento do Analisador LR

- Movimentos de um analisador LR para a entrada id*id+id.

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

# Construindo tabelas de análise SLR

- Algoritmo
- ENTRADA: Uma gramáticaestendida G'.
- SAÍDA: As funções ACTION e GOTO da tabela de análise SLR para G'.
- MÉTODO:

# CONSTRUINDO TABELAS DE ANÁLISE SLR

- 1. Construa C = { I0, I1,... In}, a coleção de conjuntos de itens LR(0) para G'.
- 2. O estado i é construído a partir de Ii. As ações de reconhecimento sintático para o estado i são determinadas da seguinte forma:

  (a) Se o item [A → α.aβ] está em Ij e GOTO(Ii,a)=Ij, então defina ACTION[i,a] como "shift j". O 'a' deve ser terminal.

  (b) Se o item [A → α.] está em Ii, então defina ACTION [i,a] como "reduce A → α" para todo a em FOLLOW(A); A pode não ser S', o símbolo inicial da gramatica.

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

  Se houver conflito resultante das regras anteriores, dizemos que a gramática não é SLR(1). Neste caso, o algoritmo deixa de produzir um analisar sintático.
- 3. Astransições goto para o estado i são construídas para todos os não-terminais A usando a regra: Se GOTO( Ii,A)=i= Ij, então GOTO[i,A]=j.
- 4. Entradas não definidas pelas regras (2) e (3) caracterizam "erro".
- 5. O estado inicial construído a partir do conjunto de itens contendo [S' → .S].

# Prefixo Viáveis

- Automatos LR(0) guiam nas decisões do tipo shift-reduce, pois caracteriza as cadeias de símbolos da gramática que podem aparecer na pilha de um AS shift-reduce para a gramática.
- O analisador não deve avançar além do handle, visto que nem todos os prefixos das formas sentenciais à direita (prefixos viáveis) podem a aparecer na pilha. Por exemplo:

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

- Durante o reconhecimento sintático, a pilha conterá (, (E e (E), mas não deve conter (E)*, pois (E) é um handle, que precisa ser reduzido para F antes de transferir o *.

# Prefixo Viáveis

- Um prefixo viável é um prefixo de uma forma sentencial à direita que não continua além da extremidade direita do handle mais à direita dessa forma sentencial.
- A análise SLR baseia-se de que autômatos LR(0) reconhecem prefixos viáveis. A → β1.β2 é válido para um prefixo viável αβ1 se existir uma derivação.

$$S' \underset{rm}{\overset{*}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha \beta_1 \beta_2 w$$

# Prefixo Viáveis

- O fato de $A \rightarrow \beta1.\beta2$ ser válido para $\alpha\beta1$ traz muito a respeito sobre transferir ou reduzir quando encontramos $\alpha\beta1$ na pilha de análise.
- Se $\beta2 \mathrel{!}= \varepsilon$, sugere que ainda não transferimos o handle para a pilha, próximo movimento seria a transferência.
- Se $\beta2 = \varepsilon$, tudo indica $A \rightarrow \beta1$ é o handle e podemos reduzir segundo essa produção.
- Naturalmente, dois itens válidos podem nos dizer para fazer diferentes coisas para o mesmo prefixo viável.

# Analisadores Sintáticos LR mais poderosos

- Para esse analise será levado em consideração o lookahead, e serão vistos dois métodos para isso:
- 1 – LR Canônico  ou LR usa o lookahead, essa método usa uma tabela a partir dos itens LR(1)
- 2 – Look Ahead LR ou LALR, possui menos estados, e a incorporação dos itens lookahead o deixa mais poderoso, e geral.

# Itens Canônicos

- É a técnica mais genérica para construir tabelas de analise LR.
- É fomentada porque em algumas situações, quando o estado i, aparece no topo da pilha, o prefixo viável para uma produção não pode ser seguido de “a” (exemplo), assim a reduçao não é permitida

# Construindo Conjunto de Itens LR(1)

- O método inclui mais dois procedimentos, na construção canônica:
- Clousure e Goto

# O Algoritmo

```
SetOfItems CLOSURE(I) {
    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;
}

SetOfItems GOTO(I,X) {
    inicializa J para ser o conjunto vazio;
    for ( cada item [A → α·Xβ,a] em I )
        adicione item [A → α X·β,a] ao conjunto J;
    return CLOSURE(J);
}
void items(G') {
    inicializa C como CLOSURE({[S' → ·S,$]});
    repeat
        for ( cada conjunto de itens I em C )
            for ( cada símbolo X da gramática )
                if ( GOTO(I,X) não é vazio e não está em C )
                    adicione GOTO(I,X) em C;
    until não haja mais conjuntos de itens para serem incluídos em C;
}
```

# Considerando a seguinte gramática:

- S' → S
- S → CC
- C → cD | d

- Vamos gerar os Itens LR(1):

# Passo a passo:

- $I_{0:}$ $S \rightarrow . S, \$$
  $S \rightarrow .CC, \$$
  $C \rightarrow cC, c/d$
  $C \rightarrow .d, c/d$
- $I_{1:}$ $S' \rightarrow S., \$$

# Passo a passo:

- $I_{2:}$ $S \rightarrow C.C,\$$
  $C \rightarrow .cC,\$$
  $C \rightarrow .d,\$$

- $I_{3:}$ $C \rightarrow c.C,c/d$
  $C \rightarrow .cC,c/d$
  $C \rightarrow .d,c/d$

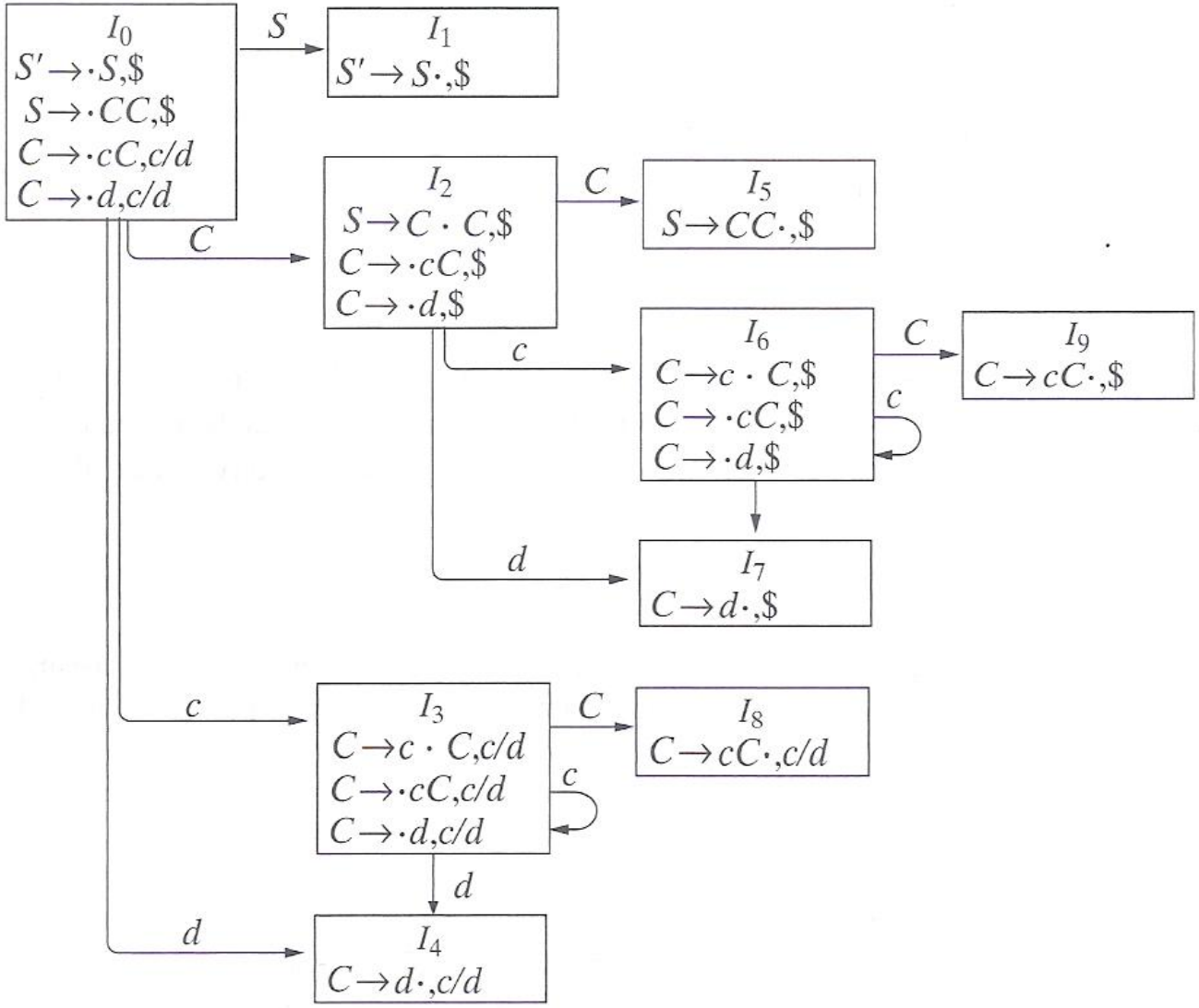
# Passo a passo:

- $I_{4:}$ $C \rightarrow d., c/d$
- $I_{5:}$ $S \rightarrow CC.,c/d$
- $I_{6:}$ $C \rightarrow c.C,\$$
  $C \rightarrow .cC,\$$
  $C \rightarrow .d,\$$

# Passo a Passo

- $I_{7:}$ C → d., $
- $I_{8:}$ C → cC.,c/d
- $I_{9:}$ C → cC.,$

# GRAFO GOTO


$I_0$
$S' \to \cdot S,\$$
$S \to \cdot CC,\$$
$C \to \cdot cC,c/d$
$C \to \cdot d,c/d$
S
$I_1$
$S' \to S\cdot,\$$
C
$I_2$
$S \to C \cdot C,\$$
$C \to \cdot cC,\$$
$C \to \cdot d,\$$
C
$I_5$
$S \to CC\cdot,\$$
c
$I_6$
$C \to c \cdot C,\$$
$C \to \cdot cC,\$$
$C \to \cdot d,\$$
C
$I_9$
$C \to cC\cdot,\$$
c
d
$I_7$
$C \to d\cdot,\$$
c
$I_3$
$C \to c \cdot C,c/d$
$C \to \cdot cC,c/d$
$C \to \cdot d,c/d$
C
$I_8$
$C \to cC\cdot,c/d$
c
d
d
$I_4$
$C \to d\cdot,c/d$

# Tabela LR(1) Canônicas de Análise

- Para criar esse tabela é utilizado um metodo especifico

- Entrada: Uma gramática estendida G’
- Saída: As funções ACTION e GOTO da tabela LR canônica de analise G’.

# Método para criação

**MÉTODO:**

1. Construa $C' = \{I_0, I_1, \cdots, I_n\}$, a coleção de conjuntos de itens LR(1) para $G'$.
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 como a seguir.
   (a) Se $[A \rightarrow \alpha \cdot a\beta, b]$ 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 $[A \rightarrow \alpha \cdot, a]$ estiver em $I_i$, $A \neq S'$, então defina ACTION[$i$, $a$] como "*reduce* $A \rightarrow \alpha$".
   (c) Se $[S' \rightarrow S \cdot, \$]$ estiver em $I_i$, então defina ACTION[$i$, \$] como "*accept*".
   Se quaisquer ações de conflito resultarem das regras anteriores, dizemos que a gramática não é LR(1), e o algoritmo falha nesse caso.
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 $[S' \rightarrow \cdot S, \$]$.

| 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 | | |

# Construção da tabela de Análise Sintática LALR

- Método para construção do analisador sintático
- Método mais utilizado, por ter tabelas menores do que LR Canônicas
- A maioria das construções podem ser representadas por gramáticas LALR
- É mais simples e fácil fazer com LALR do com LR canônicas

# Construção da tabela de Análise Sintática LALR

- Primeiramente para esse construção são procurados conjuntos com núcleos idênticos:
- Ex: $I_4$ e $I_7$ possuem núcleos idênticos:

- $I_{4:}$ C → d., c/d
- $I_{7:}$ C → d., $
- A união dos estados é feita, e então é revisada a função de transição para tratar dos erros

# Construção da tabela de Análise Sintática LALR

- As uniões dos estados nunca podem gerar conflitos shift/reduce, não sabendo qual ação tomar, pois se houver há um erro na gramática.

# ALGORITMO

**ENTRADA:** Uma gramática estendida $G'$.

**SAÍDA:** Funções ACTION e GOTO da tabela LALR de análise para $G'$.

**MÉTODO:**

1. Construa $C = \{ I_0 , I_1, \ldots , I_n\}$, 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 substitua pela sua união.
3. Considere que $C' = \{J_0, J_1, \ldots , J_m\}$ 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 maneira que no Algoritmo 4.56. Se houver um conflito de ação de análise, o algoritmo deixa de produzir um reconhecedor sintático, e a gramática é considerada como não sendo LALR(1).
4. A tabela GOTO é construída da forma a seguir. Se $J$ é a união de um ou mais conjuntos de itens LR(1), ou seja, $J = I_1 \cap I_2 \cap \cdots \cap I_k$, então os núcleos de GOTO($I_1$, $X$), GOTO($I_2$, $X$), ..., 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($J$, $X$) = $K$.

A tabela produzida pelo Algoritmo (4.59) é denominada *tabela de análise LALR* para $G$. Se não houver conflitos de ação de análise, então a gramática dada é considerada uma *gramática LALR(1)*. A coleção de conjuntos de itens construídos na etapa (3) é chamada de *coleção LALR(1)*.

# TABELA LALR

| 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 | r2 | r2 | r2 | | |

# Construção da tabela de Análise Sintática LALR

- Embora os analisadores LALR e LR empilharem estados com nomes diferentes, eles fazem exatamente a mesma coisa, só que o LALR faz de forma mais simples
- Esta não é a melhor forma de criar tabelas LALR

# Construção eficiente de tabelas de Análise LALR

- Podem ser feitas algumas modificações no algoritmo de construção para evitar a construção completa da tabela dos itens LR(1)
- Primeiro pode-se representar qualquer conjunto de itens LR(0) ou LR(1) por sua base, ou seja os itens iniciais tais como:
- S' → .S

# Construção eficiente de tabelas de Análise LALR

- Pode-se construir as bases através do processo de propagação espontânea dos lookaheads.

- E também as bases da LALR(1) podem ser criadas pela função de fechamento CLOSURE

# EXEMPLIFICANDO

- Usando a gramática não SLR extendida:
- S' → .S
- S → L = R | R
- L → *R | id
- R → L

# EXEMPLIFICANDO

- Itens gerados para essa base:

$I_0$: $S' \rightarrow \cdot S$

$I_1$: $S' \rightarrow S\cdot$

$I_2$: $S \rightarrow L\cdot = R$
$R \rightarrow L.$

$I_3$: $S \rightarrow R\cdot$

$I_4$: $L \rightarrow *\cdot R$

$I_5$: $L \rightarrow \mathbf{id}\cdot$

$I_6$: $S \rightarrow L = \cdot R$

$I_7$: $L \rightarrow * R\cdot$

$I_8$: $R \rightarrow L\cdot$

$I_9$: $S \rightarrow L = R\cdot$

# EXEMPLIFICANDO OS LOOKAHEADS

**ENTRADA**: A base $K$ de um conjunto $I$ de itens LR(0) e um símbolo da gramática $X$.

**SAÍDA**: Os lookaheads gerados espontaneamente pelos itens de $I$ para os itens base em GOTO($I$, $X$) e os itens de $I$ a partir dos quais os lookaheads são propagados para os itens base em GOTO($I$, $X$).

**MÉTODO:** O algoritmo é dado na Figura 4.45.

**for** ( cada item $A \rightarrow \alpha \cdot \beta$ em $K$ ) {
    $J$ := CLOSURE($\{[A \rightarrow \alpha \cdot \beta, \#]\}$ );
    **if** ( $[B \rightarrow \gamma \cdot X\delta, a]$ está em $J$, e $a$ não é # )
        conclui que lookahead $a$ é gerado espontaneamente para o item
            $B \rightarrow \gamma X \cdot \delta$ em GOTO($I$, $X$);
    **if** ( $[B \rightarrow \gamma \cdot X\ \delta, \#]$ está em $J$ )
        conclui que os lookaheads se propagam de $A \rightarrow \alpha \cdot \beta$ em $I$ para
            $B \rightarrow \gamma X \cdot \delta$ em GOTO($I$, $X$);
}

# Propagando os Lookaheads

- Após determinar os lookaheads, deve-se propaga-los através do metodo:

**ENTRADA**: Uma gramática estendida $G'$.

**SAÍDA**: *As bases* dos conjuntos de itens da coleção LALR(1) para $G'$.

**MÉTODO**:

1. Construa as bases dos conjuntos de itens LR(0) para $G$. Se espaço não for problema, o modo mais simples é construir os conjuntos de itens LR(0), como na Seção 4.6.2, e depois remover os itens que não são *base*. Se o espaço for bastante restrito, podemos em vez disso armazenar apenas os itens base para cada conjunto, e calcular a função de transição para um conjunto de itens $I$ computando primeiro o fechamento de $I$.
2. Aplique o Algoritmo 4.62 à base de cada conjunto de itens LR(0) e símbolo $X$ da gramática para determinar quais lookaheads são gerados espontaneamente para os itens base em GOTO($I$, $X$), e de quais itens em $I$ os lookaheads são propagados para os itens base em GOTO($I$, $X$).
3. Inicie uma tabela que dê, para cada item base em cada conjunto de itens, os lookaheads associados. Inicialmente, cada item tem associado a ele apenas os lookaheads que determinamos no passo (2) como sendo gerados espontaneamente.
4. Faça passadas repetidas sobre os itens base em todos os conjuntos. Quando visitamos um item $i$, pesquisamos os itens base aos quais $i$ propaga seus lookaheads, usando as informações tabuladas no passo (2). O conjunto corrente de lookaheads para $i$ é acrescentado aos que já estão associados a cada um dos itens aos quais $i$ propaga seus lookaheads. Continuamos fazendo passadas pelos itens base até que não seja mais possível propagar novos lookaheads.

# Tabela de Propagação

| De | Para |
|---|---|
| $I_0$: $S' \to \cdot S$ | $I_1$: $S' \to S\cdot$<br>$I_2$: $S \to L\cdot = R$<br>$I_2$: $R \to L\cdot$<br>$I_3$: $S \to R\cdot$<br>$I_4$: $L \to *\cdot R$<br>$I_5$: $L \to \mathbf{id}\cdot$ |
| $I_2$: $S \to L\cdot = R$ | $I_6$: $S \to L = \cdot R$ |
| $I_4$: $L \to *\cdot R$ | $I_4$: $L \to *\cdot R$<br>$I_5$: $L \to \mathbf{id}\cdot$<br>$I_7$: $L \to * R\cdot$<br>$I_8$: $R \to L\cdot$ |
| $I_6$: $S \to L = \cdot R$ | $I_4$: $L \to *\cdot R$<br>$I_5$: $L \to \mathbf{id}\cdot$<br>$I_8$: $R \to L\cdot$<br>$I_9$: $S \to L = R\cdot$ |

# Cálculos do Lookahead

| Item do Conjunto | Lookaheads | | | |
|---|---|---|---|---|
| | Início | Passo 1 | Passo 2 | Passo 3 |
| $I_0$: $S' \rightarrow \cdot S$ | $ | $ | $ | $ |
| $I_1$: $S' \rightarrow S\cdot$ | | $ | $ | $ |
| $I_2$: $S \rightarrow L\cdot = R$ | | $ | $ | $ |
| $R \rightarrow L\cdot$ | | $ | $ | $ |
| $I_3$: $S \rightarrow R\cdot$ | | $ | $ | $ |
| $I_4$: $L \rightarrow *\cdot R$ | = | =/$ | =/$ | =/$ |
| $I_5$: $L \rightarrow \mathbf{id}\cdot$ | = | =/$ | =/$ | =/$ |
| $I_6$: $S \rightarrow L = \cdot R$ | | | $ | $ |
| $I_7$: $L \rightarrow * R\cdot$ | | = | =/$ | =/$ |
| $I_8$: $R \rightarrow L\cdot$ | | = | =/$ | =/$ |
| $I_9$: $S \rightarrow L = R\cdot$ | | | | $ |

# Compactação de tabela LR

- Gramáticas podem ter até 100 terminais
- Função ação pode ter até 20.00 entradas
- Solução para compactar os campos ACTION E GOTO.

# Técnicas para compactar tabelas LR

- Muitas linhas da tabela são idênticas
- Pode-se economizar espaço criando apontadores
- Estados com mesmas ações apontam para o mesmo endereço
- Cada terminal recebe um numero

# TÉCNICAS PARA COMPACTAR TABELAS LR

- Pode conseguir mais eficiencia em relação ao espaço com analisador sintático mais lento
- Lista de ações para cada estado
- Pares com SIMBOLO TERMINAL E AÇÃO.
- Ação mais frequente no final da lista
- Entradas de erro são substituídas por reduções

# Exemplo de Compactação

significa accept,
rada em branco significa *error*.

| 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 | | s6 | | | s11 | | | | |
| 9 | | r1 | s7 | | r1 | r1 | | | |
| 10 | | r3 | r3 | | r3 | r3 | | | |
| 11 | | r5 | r5 | | r5 | r5 | | | |

Figura 4.37 Tabela de análise para a gramática de expressão.

# Exemplo de Compactação

- Reduções para os estados 0,4,6,7
- SIMBOLO AÇÃO
- id s5
- ( s4
- any error
- Reduções para os estados 3,5,10,11

  (any, rj)

# Compactação da Tabela GOTO

- Também no formato de lista
- Lista de pares para cada não-terminal
- Formato GOTO [currenteState,A] = nextState
- Exemplo para o Não-terminal F
- currentState nextState
- 7 10
- any 3
- No caso do não-terminal E pode-se escolher entre o 1 e 8 pois duas estradas são necessárias nos dois casos.

# Usando Gramáticas Ambiguas

- Toda gramática ambígua não é LR
- Gramáticas ambíguas são úteis em alguns casos
- Construções de linguagens como expressões
- Isolar construções sintáticas
- Devem ser utilizadas com cuidado

# Recuperação de erros na Análise Sintática LR

- Implementado recuperação de erro
- Modo Pânico
- Analisar até encontrar estado s com transição sob um não-terminal A
- Descartar símbolos até encontrar “a”
- Empilha estado GOTO(s,A) e retorna a analise
- Tenta eliminar a frase que contem o erro

# Recuperação de erros na Análise Sintática LR

- Recuperação em nível de frase
- Examina cada entrada na tabela
- Com base na tabela decide qual o erro mais provável do programador
- Projeta-se rotinas de erro específicas
- Podem incluir inserções, remoções, alterações e transposições

# Exemplo de recuperação de erros na análise LR

- Considerando a gramática e tabela
- E->E+E | E*E | (E) | id

| Estado | Action | | | | | | Goto |
|---|---|---|---|---|---|---|---|
| | **id** | + | * | ( | ) | $ | *E* |
| 0 | s3 | e1 | e1 | s2 | e2 | e1 | 1 |
| 1 | e3 | s4 | s5 | e3 | e2 | acc | |
| 2 | s3 | e1 | e1 | s2 | e2 | e1 | 6 |
| 3 | r4 | r4 | r4 | r4 | r4 | r4 | |
| 4 | s3 | e1 | e1 | s2 | e2 | e1 | 7 |
| 5 | s3 | e1 | e1 | s2 | e2 | e1 | 8 |
| 6 | e3 | s4 | s5 | e3 | s9 | e4 | |
| 7 | r1 | r1 | s5 | r1 | r1 | r1 | |
| 8 | r2 | r2 | r2 | r2 | r2 | r2 | |
| 9 | r3 | r3 | r3 | r3 | r3 | r3 | |

# Exemplo de recuperação de erros na análise LR

- Nos estados só com reduções as entradas de erros foram substituídas por reduções
- As entradas de erros restantes foram substituídas por chamadas a rotinas de erros
- Uma estratégia segura garantirá que pelo menos um símbolo de entrada será removido