

Os seguintes algoritmos e procedimentos foram tirados do livro do dragão [1], todos do capítulo 4.

1 Geral

1.1 Elminicação da recursão à esquerda

$A \rightarrow A\alpha \mid \beta$
Remoção:
 $A \rightarrow \beta A'$
 $A' \rightarrow \alpha A' \mid \varepsilon$

1.2 Fatoração à esquerda

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$
Remoção:
 $A \rightarrow \alpha A'$
 $A' \rightarrow \beta_1 \mid \beta_2$

1.3 Conjunto PRIMEIRO (FIRST)

Para computar $PRIMEIRO(X)$ para todos os símbolos gramaticais X , aplique as seguintes regras até que nenhum terminal ou ε possa ser adicionado a qualquer conjunto PRIMEIRO.

1. Se X for um terminal, então $PRIMEIRO(X)$ é X .
2. Se $X \rightarrow \varepsilon$ for uma produção, adicionar ε a $PRIMEIRO(X)$.
3. Se X for um não-terminal e $X \rightarrow Y_1 Y_2 \dots Y_k$ uma produção, colocar a em $PRIMEIRO(X)$ se, para algum i , a estiver em $PRIMEIRO(Y_i)$ e ε estiver em todos $PRIMEIRO(Y_1), \dots, PRIMEIRO(Y_{i-1})$; isto é, se $Y_1 \dots Y_{i-1} \xRightarrow{*} \varepsilon$. Se ε estiver em $PRIMEIRO(Y_j)$ para todos os $j = 1, 2, \dots, k$, adicione, então, ε a $PRIMEIRO(X)$. Por exemplo, todo o que estiver em $PRIMEIRO(Y_1)$ estará certamente em $PRIMEIRO(X)$. Se Y_1 não derivar ε , então não adicionamos mais nada a $PRIMEIRO(X)$; mas se $Y_1 \xRightarrow{*} \varepsilon$, precisamos adicionar $PRIMEIRO(Y_2)$ e assim por diante.

1.4 Conjunto SEGUINTE (FOLLOW)

Para computar $SEGUINTE(A)$ para todos os não-terminais A , aplique as seguintes regras até que nada mais possa ser adicionado a qualquer conjunto $SEGUINTE$.

1. Colocar $\$$ em $SEGUINTE(S)$, onde S é o símbolo de partida e $\$$ o marcador de fim de entrada à direita.
2. Se existir uma produção $A \rightarrow \alpha B \beta$, então tudo em $PRIMEIRO(\beta)$, exceto ε , é colocado em $SEGUINTE(B)$.
3. Se existir uma produção $A \rightarrow \alpha B$ ou uma produção $A \rightarrow \alpha B \beta$ onde $PRIMEIRO(\beta)$ contém ε (isto é, $\beta \xRightarrow{*} \varepsilon$), então tudo em $SEGUINTE(A)$ está em $SEGUINTE(B)$.

2 Top-down (LL)

2.1 Construção de tabelas sintáticas preditivas

Entrada. Gramática G .

Saída. Tabela sintática M .

Método.

1. Para cada produção $A \rightarrow \alpha$ da gramática, execute os passos 2 e 3.
2. Para cada terminal em $PRIMEIRO(\alpha)$, adicione $A \rightarrow \alpha$ a $M[A, a]$.
3. Se ε estiver em $PRIMEIRO(\alpha)$, adicione $A \rightarrow \alpha$ a $M[A, b]$, para cada terminal b em $SEGUINTE(A)$. Se ε estiver em $PRIMEIRO(\alpha)$ e $\$$ em $SEGUINTE(A)$, adicione $A \rightarrow \alpha$ a $M[A, \$]$.
4. Faça cada entrada indefinida de M ser **erro**.

2.2 Algoritmo para análise sintática preditiva não recursiva

Entrada. Uma cadeia w e uma tabela sintática M para a gramática G .

Saída. Se w estiver em $L(G)$, uma derivação mais à esquerda de w ; caso contrário, uma indicação de erro.

Método. Inicialmente, o analisador sintático está numa configuração na qual possui somente $\$S$ na pilha, com S , o símbolo de partida de G ao topo e $w\$$ no *buffer* de entrada. O programa, que utiliza a tabela sintática preditiva M para realizar uma análise sintática da entrada é mostrado abaixo.

faça ip apontar para o primeiro símbolo de $w\$$;

repetir

seja X o símbolo ao topo da pilha e a o símbolo apontado por ip ;

se X for um terminal ou $\$$ **então**

se $X = a$ **então**

remover X da pilha e avançar ip

senão *erro()*

senão /* X é um não-terminal */

se $M[X, a] = X \rightarrow Y_1 Y_2 \dots Y_k$ **então início**

remover X da pilha;

empilhar Y_k, Y_{k-1}, \dots, Y_1 , com Y_1 ao topo da pilha;

escrever a produção $X \rightarrow Y_1 Y_2 \dots Y_k$;

fim

senão *erro()*

até que $X = \$$ /* a pilha está vazia */

3 Bottom-up SLR

3.1 A Operação de Fechamento

Se I for um conjunto de itens para uma gramática G , então o $fechamento(I)$ é o conjunto de itens construídos a partir de I por essas duas regras:

1. Inicialmente, cada item I é adicionado ao $fechamento(I)$.
2. Se $A \rightarrow \alpha \cdot B\beta$ estiver em $fechamento(I)$ e $B \rightarrow \gamma$ for uma produção, adicionar o item $B \rightarrow \cdot\gamma$ a I , se já não estiver lá. Aplicamos esta regra até que não possam ser adicionados novos itens ao $fechamento(I)$.

3.2 A Operação Desvio

$Desvio(I, X)$ é definida como o fechamento do conjunto de todos os itens $[A \rightarrow \alpha X \cdot \beta]$ tais que $[A \rightarrow \alpha \cdot X\beta]$ esteja em I .

3.3 A construção dos conjuntos de itens $LR(0)$ (SLR)

procedimento itens(G');

início

$C := \{ fechamento(\{[S' \rightarrow \cdot S]\}) \};$

repetir

para cada conjunto de itens I em C e cada símbolo gramatical X tal que $desvio(I, X)$ não seja vazio e não esteja em C **faça** incluir $desvio(I, X)$ a C

até que não haja mais conjuntos de itens a serem incluídos a C

fim

3.4 Construção de tabelas sintáticas

Entrada. Uma gramática aumentada G' .

Saída. As funções sintáticas SLR $acao$ e $desvio$ para G' .

Método.

1. Construir $C = \{I_0, I_1, \dots, I_n\}$, a coleção de conjuntos de itens $LR(0)$ para G' .
2. O estado i é construído a partir de I_i . As ações sintáticas para o estado i são determinadas como se segue:
 - a) Se $[A \rightarrow \alpha \cdot a\beta]$ estiver em I_i e $desvio(I_i, a) = I_j$, então estabelecer $acao[i, a]$ em "empilhar j ". Aqui, a precisa ser um terminal.
 - b) Se $[A \rightarrow \alpha \cdot]$ estiver em I_i , então estabelecer a $acao[i, a]$ em "reduzir através de $A \rightarrow \alpha$ ", para todo a em $SEGUITNE(A)$; aqui, A não pode ser S' .
 - c) Se $[S' \rightarrow S \cdot]$ estiver em I_i então fazer $acao[i, \$]$ igual a "aceitar".

Se quaisquer ações conflitantes forem geradas pelas regras anteriores, dizemos que a gramática não é SLR(1). O algoritmo falha em produzir um analisador sintático neste caso.

3. As transições de desvio para o estado i são construídas para todos os não-terminais A usando-se a seguinte regra: se $desvio(I_i, A) = I_j$, então $desvio[i, A] = j$.
4. Todas as entradas definidas pelas regras (2) e (3) são tornadas "erro".
5. O estado inicial do analisador sintático é aquele construído a partir do conjunto de itens contendo $[S' \rightarrow \cdot S]$.

4 Bottom-up LR canônico

4.1 Fechamento LR(1)

função $fechamento(I)$;

início

repetir

para cada item $[A \rightarrow \alpha \cdot B\beta, a]$ em I , cada produção $B \rightarrow \gamma$ em G' , e cada terminal b em $PRIMEIRO(\beta a)$ tal que $[B \rightarrow \cdot \gamma, b]$ não está em I **faça** incluir $[B \rightarrow \cdot \gamma, b]$ em I ;

até que não possam ser adicionados mais itens a I ;

retornar I

fim;

4.2 Desvio LR(1)

função *fechamento*(I);

início

seja J o conjunto de itens $[A \rightarrow \alpha X \cdot \beta, a]$ tais que $[A \rightarrow \alpha \cdot X \beta, a]$ esteja em I

retornar *fechamento*(J)

fim;

4.3 Construção de itens LR(1)

procedimento *itens*(G');

início

$C := \{\text{fechamento}(\{[S' \rightarrow \cdot S, \$]\})\}$;

repetir

para cada conjunto de itens I em C e cada símbolo gramática X tal que *desvio*(I, X) não é vazio e não está em C **faça** incluir *desvio*(I, X) a C

até que não possam ser adicionados itens a C ;

fim;

4.4 Construção de tabelas sintáticas LR(1)

Entrada. Uma gramática aumentada G' .

Saída. As funções sintáticas canônicas LR *acao* e *desvio* para G' .

Método.

1. Construir $C = \{I_0, I_1, \dots, I_n\}$, a coleção de conjuntos de itens LR(1) para G' .
2. O estado i do analisador sintático é construído a partir de I_i . As ações sintática para o estado i são determinadas como se segue:
 - a) Se $[A \rightarrow \alpha \cdot a \beta, b]$ estiver em I_i e *desvio*(I_i, a) = I_j , então estabelecer *acao*[i, a] igual a "empilhar j ". Aqui, a é exigido ser um terminal.
 - b) Se $[A \rightarrow \alpha \cdot, a]$ estiver em I_i , $A \neq S'$, então fazer *acao*[i, a] igual a "reduzir $A \rightarrow \alpha$ ".
 - c) Se $[S' \rightarrow S \cdot, \$]$ estiver em I_i , então fazer *acao*[$i, \$$] igual a "aceitar".

Se um conflito resultar das regras acima, a gramática não é considerada LR(1) e o algoritmo falha.

3. As transições de desvio para o estado i são determinadas como se segue: se *desvio*(I_i, A) = I_j , então *desvio*[i, A] = j .
4. Todas as entradas definidas pelas regras (2) e (3) são tornadas "erro".
5. O estado inicial do analisador sintático é aquele construído a partir do conjunto de itens contendo $[S' \rightarrow \cdot S, \$]$.

5 Programa de análise sintática LR

Inicialmente, o analisador sintático possui s_0 na pilha, onde s_0 é o estado inicial, e $w\$$ no buffer de entrada.

fazer *ip* apontar para primeiro símbolo $w\$$;

repetir para sempre início

seja s o estado ao topo da pilha e a o símbolo apontador por *ip*;

se *acao*[s, a] = *empilha* s' **então início**

empilhar a e em seguida s' no topo da pilha;

```

    avançar  $ip$  para o próximo símbolo de entrada
  fim
  senão se  $acao[s, a] = reduzir A \rightarrow \beta$  então início
    desempilhar  $2 \times |\beta|$  símbolos para fora da pilha; se  $s'$  o estado agora ao topo da pilha; empilhar
     $A$  e em seguida  $desvio[s', A]$ ;
    escrever a produção  $A \rightarrow \beta$ ;
  fim
  senão se  $acao[s, a] = aceitar$  então
    retornar
  fim
  senão erro()
fim

```

Referências

- [1] A. V. AHO, M. S. LAM, R. SETHI, and J. D. ULLMAN, “Compiladores: princípios, técnicas e ferramentas,” *Guanaban Koogan*, vol. 1995, 1986.