Universidade de Santa Cruz do Sul – UNISC
Departamento de informática

COMPILADORES

Análise sintática

Parte 02

Prof. Geovane Griesang
geovanegriesang@unisc.br

Análise sintática

FIRST e FOLLOW
+
LL(1)

Data	Conteúdo
23/09/2013	3. Análise Sintática: 3.1 analisadores ascendentes e descendentes;
30/09/2013	3. Análise Sintática: 3.2 análise preditiva; **Próxima aula (EaD)**
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	Prova Individual e sem consulta 2.

Análise sintática

Derivações

A construção de uma árvore de derivação pode tornar-se precisa pela adoção de uma abordagem derivativa, em que as produções são tratadas como regras de reescrita.

A partir de um símbolo inicial, a cada passo da reescrita substitui-se um não-terminal pelo corpo de uma das produções.

Esse processo de aplicação das regras de reescrita é conhecido como derivação.

Análise sintática

Derivações

Exemplo: Considere a gramática abaixo, com um único não-terminal E, que representa produção $E \rightarrow - E$:

$$E \rightarrow E + E \mid E * E \mid - E \mid (E) \mid id$$

A produção $E \rightarrow - E$ diz que, se E representa uma expressão, então $- E$ também deve representar uma expressão.

A substituição de um único E por $-E$ é descrita por:

$$E \Rightarrow - E$$

Que é lido como "E deriva $-E$".

E → E + E | E * E | – E | (E) | id

Análise sintática

Derivações

Continuação...

A produção E → (E) pode ser aplicada para substituir qualquer instância de E em qualquer cadeia de símbolos da gramática por (E), por exemplo, E*E ⇒ (E)*E ou E*E ⇒ E*(E).

A partir de um único E, podemos aplicar as produções repetidamente em qualquer ordem para obter uma sequencia de substituições.

E ⇒ – E ⇒ – (E) ⇒ – (id)

Chegamos a essa sequencia de substituições de uma derivação de – (id) a partir de E.

E → E + E | E * E | – E | (E) | id

Análise sintática

Derivações

Continuação...

Uma definição geral de derivação, considere um não-terminal A no meio de uma sequencia de símbolos da gramática, como em $\alpha A \beta$, onde α e β são símbolos de uma gramática.

Suponha que A → γ seja uma produção. Então, escrevemos $\alpha A \beta \Rightarrow \alpha \gamma \beta$.

O símbolo ⇒ significa "deriva em um passo".

Quando uma sequencia de passos de derivação $\alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n$, dizemos que α_1 deriva α_n.

Análise sintática

Derivações

Continuação...

Muitas vezes, queremos dizer "deriva em zero ou mais passos". Para isto, podemos usar a simbologia $\overset{*}{\Rightarrow}$.

1. $\alpha \overset{*}{\Rightarrow} \alpha$, para qualquer cadeia α, *e*
2. *Se* $\alpha \overset{*}{\Rightarrow} \beta$ *e* $\beta \Rightarrow \gamma$, então $\alpha \overset{*}{\Rightarrow} \gamma$.

Desta mesma forma, $\overset{+}{\Rightarrow}$ significa "deriva em um ou mais passos".

E → E + E | E * E | – E | (E) | id

Análise sintática

Derivações

Continuação...

1. Em derivações *mais a esquerda*, o não-terminal mais a esquerda em cada forma sentencial sempre é escolhido. Se $\alpha \Rightarrow \beta$ é um passo em que o não-terminal mais a esquerda em α é substituído, escrevemos $\alpha \overset{\Rightarrow}{}_{lm} \beta$.

$$E \overset{\Rightarrow}{}_{lm} - E \overset{\Rightarrow}{}_{lm} - (E) \overset{\Rightarrow}{}_{lm} - (E + E) \overset{\Rightarrow}{}_{lm} - (id + E) \overset{\Rightarrow}{}_{lm} - (id + id)$$

2. Em derivações *mais a direita*, o não-terminal mais a direita em cada forma sentencial sempre é escolhido. Escrevemos $\alpha \overset{\Rightarrow}{}_{rm} \beta$.

Análise sintática

Análise sintática de descida recursiva - FIRST e FOLLOW

A construção de analisadores descendentes e ascendentes é auxiliada por duas funções, *FIRST* e *FOLLOW*, associadas a uma gramática G.

Durante a análise descendente, *FIRST* e *FOLLOW* nos permitem escolher qual produção aplicar, com base no próximo símbolo de entrada.

Durante a recuperação de erro no modo pânico, conjunto de *tokens* produzidos por *FOLLOW* podem ser usados como *tokens* de sincronismo.

Revisão: modo pânico ou desespero

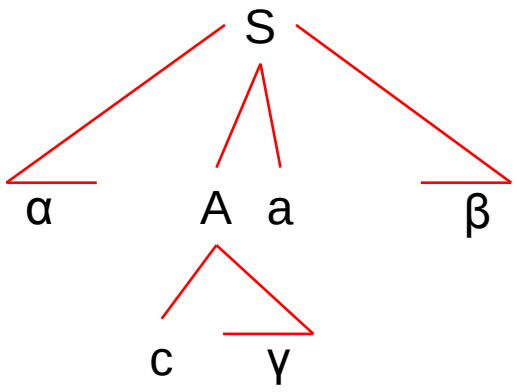
Ao descobrir um erro, o analisador sintático descarta símbolos de entrada, um de cada vez, até que seja encontrado um *token* pertencente a um conjunto designado de *tokens de sincronização*.

Análise sintática

Análise sintática de descida recursiva - FIRST e FOLLOW

Defina *FIRST(α)*, onde α é qualquer cadeia de símbolos da gramática, como sendo o conjunto de símbolos terminais que iniciam as cadeias derivadas de α.

Se $\alpha \rightarrow \varepsilon$, então ε também está em FIRST(α). Por exemplo, na imagem abaixo, $A \overset{*}{\Rightarrow} c\gamma$, portanto c está em FIRST(A).



O símbolo terminal *c* está em *FIRST(A)* e o símbolo terminal está em *a* está em *FOLLOW(A)*.

Análise sintática

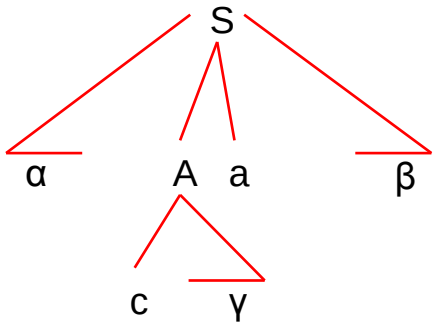
Análise sintática de descida recursiva - FIRST e FOLLOW

Para que possamos ter uma ideia de como *FIRST* pode ser usado durante o reconhecimento preditivo, considera-se as duas produções-A, $A \rightarrow \alpha \mid \beta$, onde *FIRST(α)* e *FIRST(β)* são conjuntos disjuntos (não têm elementos em comum), mas nunca em ambos.

Desta forma, pode-se escolher uma das alternativas de A examinando o próximo símbolo de entrada *a*. desde que a só possa estar em *FIRST(α)* ou *FIRST(β),* mas nunca em ambos.

Por exemplo, se a estiver em *FIRST(β)*, escolha a produção-$A \rightarrow \beta$.

Análise sintática



FIRST e FOLLOW

Defina *FOLLOW(A)*, para o não-terminal *A*, como sendo o conjunto de terminais *a* que podem aparecer imediatamente a direita de *A* em alguma forma sentencial.

Ou seja, o conjunto de terminais *a* tais que exista uma derivação na forma $S \Rightarrow \alpha A a \beta$, para algum α e β, conforme imagem acima.

Portanto, pode haver símbolos entre *A* e *a*, em algum momento durante a derivação, mas, nesse caso, eles derivam ε e desaparecem. Além disso, se *A* for símbolo não-terminal mais a direita em alguma forma sentencial, então $ estará em *FOLLOW(A)*;

$ é um símbolo "marcador de fim de cadeia de entrada" e, como tal, não faz parte de nenhuma gramática.

Análise sintática

Análise sintática de descida recursiva - FIRST e FOLLOW

Para calcular *FIRST(X)* de todos os símbolos *X* da gramática, aplique as seguintes regras até que não haja mais terminais ou ε que possam ser acrescentados a algum dos conjuntos *FIRST*.

1. Se *X* é um símbolo terminal, então *FIRST(X)* = *{X}*.

2. Se *X* é um símbolo não-terminal e $X \rightarrow Y_1 Y_2 ... Y_k$ é uma produção para algum $k \geq 1$, então acrescente *a* a *FIRST(X)* se, para algum *i*, *a* estiver em $FIRST(Y_i)$, e ε estiver em todos os $FIRST(Y_1)$, ..., $FIRST(Y_{i-1})$; ou seja, $Y_1 ... Y_{i-1} \Rightarrow \varepsilon$.

   Se ε está em $FIRST(Y_j)$ para todo $j=1,2,...,k$, então adicione ε a *FIRST(X)*.
   Por exemplo, tudo em $FIRST(Y_1)$ certamente está em *FIRST(X)*.

   Se Y_1 não derivar ε, então, não acrescentamos mais nada a FIRST(X), mas se $Y_1 \Rightarrow \varepsilon$, então adicionamos $FIRST(Y_2)$, e assim por diante.

3. Se $X \rightarrow \varepsilon$ é uma produção, então acrescente ε a *FIRST(X)*.

Análise sintática

Análise sintática de descida recursiva - FIRST e FOLLOW

Agora, podemos calcular *FIRST* para qualquer cadeia $X_1\ X_2\ ...\ X_n$:

- Adicione a $FIRST(X_1...X_n)$ todos os símbolos não-ε de $FIRST(X_1)$.
- Inclua os símbolos não-ε de $FIRST(X_2)$, se ε estiver em $FIRST(X_1)$.
- Inclua os símbolos não-ε de $FIRST(X_3)$, se ε estiver em $FIRST(X_1)$ e em $FIRST(X_2)$; e assim por diante.
- Acrescente ε a $FIRST(X_1X_2...X_n)$ se, para todo i, ε estiver em $FIRST(X_i)$.

Análise sintática

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

FIRST e FOLLOW – Mesmo exemplo da aula passada

1. FIRST(F) = *FIRST(T)* = *FIRST(E)* = {*(*, *id*}

Para entender, observe que as duas produções para F possuem corpos que se iniciam com esses dois símbolos terminais, id e o (.

F → (E) | id

T possui somente uma produção, e seu corpo começa com *F*.

Como *F* não deriva ε, *FIRST(T)* deve ser igual ao *FIRST(F)*.

O mesmo argumento se aplica a *FIRST(E)*.

FIRST(F) = FIRST(T) = FIRST(E) = {(, id}

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

Análise sintática

FIRST e FOLLOW – Mesmo exemplo da aula passada

2. FIRST(E’) = {+, ε} E’ → + T E’ | ε

O motivo é que uma das duas produções para *E’* tem um corpo que começa com o terminal +, e o corpo da outra é ε.

Sempre que um não-terminal deriva ε, incluímos ε em *FIRST* para esse não-terminal.

3. FIRST(T’) = {*, ε} T’ → * F T’ | ε

Raciocínio semelhante àquele usado no *FIRST(E’)*.

A. sintática

FIRST(F) = FIRST(T) = FIRST(E) = {(, id}
FIRST(E') = {+, ε} e FIRST(T') = {*, ε}

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

FIRST e FOLLOW – Mesmo exemplo da aula passada

4. FOLLOW(E) = *FOLLOW(E')* = {), $}

E	→	T E'
E'	→	+ T E' | ε
F	→	(E) | id

Como *E* é o símbolo inicial da gramática, *FOLLOW(E)* deve incluir *$*.

O corpo *(E)* da produção-F explica porque o *)* está em *FOLLOW(E)*.

Para *E'*, observe que esse não-terminal aparece apenas nas extremidades dos corpos das produções-E.

Assim, *FOLLOW(E')* deve ser o mesmo que *FOLLOW(E)*;

A. sintática

FIRST(F) = FIRST(T) = FIRST(E) = {(, id}
FIRST(E') = {+, ε} e FIRST(T') = {*, ε}
FOLLOW(E) = FOLLOW(E') = {), $}

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

FIRST e FOLLOW – Mesmo exemplo da aula passada

5. FOLLOW(T) = {+,), $}

E	→	T E'
E'	→	+ T E' | ε

T aparece ao lado direito das produções-E apenas seguido por *E'*.

Tudo exceto ε que está em *FIRST(E')* deve ser incluído em FOLLOW(T). Isso explica o símbolo "**+**" (FIRST(E') = {+, ε}).

Como *FIRST(E')* contém ε (ou seja, E' $\Rightarrow^{*}$ ε), e *E'* é a cadeira inteira seguindo *T* nos corpos das produções-E, tudo em *FOLLOW(E)* também deve ser incluído em *FOLLOW(T)*. Isso explica os símbolos $ e).

(Pensar em substituir E' por ε: E → T e E' → + T). FOLLOW(E) = {), $}.

A. sintática

FIRST(F) = FIRST(T) = FIRST(E) = {(, id}
FIRST(E') = {+, ε} e FIRST(T') = {*, ε}
FOLLOW(E) = FOLLOW(E') = {), $}
FOLLOW(T) = FOLLOW(T') = {+,), $}
FOLLOW(F) = {+, *,), $}.

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

FIRST e FOLLOW – Mesmo exemplo da aula passada

5. continuação...

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

E → T E'
E' → + T E' | ε
T → F T'
T' → * F T' | ε

Quanto a T', por aparecer apenas nas extremidades das produções-T, é necessário fazer o *FOLLOW(T')* = *FOLLOW(T)*.

6. FOLLOW(F) = {+, *, *)*, *$*}.

T → F T'
T' → * F T' | ε

O raciocínio deste caso é semelhante àquele para T no ponto anterior, ou seja, FIRST(T'). Pensar T → F ~~T'~~ e T' → * F ~~T'~~ | ε. FOLLOW de T e T'.

Análise sintática

Gramáticas LL(1)

Os analisadores sintáticos preditivos, ou seja, os analisadores de descida recursiva que não precisam de retrocesso, podem ser construídos para uma classe de gramáticas chamada LL(1).

O primeiro "L" em LL(1) significa que a cadeia de entrada é escandida da esquerda para a direita (*L = "left-to-right"*).

O segundo "L" representa uma derivação mais a esquerda (L = "*leftmost*").

O "1" pelo uso de um símbolo a frente na entrada utilizado em cada passo para tomar as decisões quanto a ação de análise.

Análise sintática

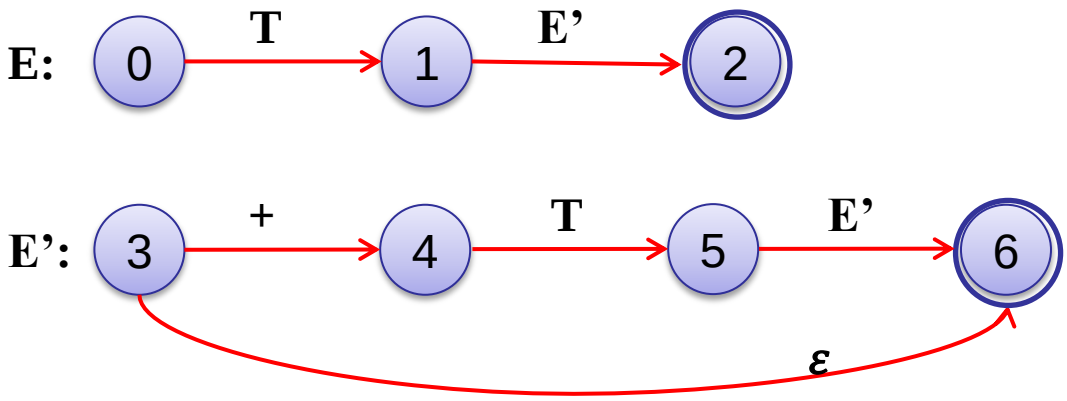
E	→	T E’
E’	→	+ T E’ | ε
T	→	F T’
T’	→	* F T’ | ε
F	→	(E) | id

Gramáticas LL(1)

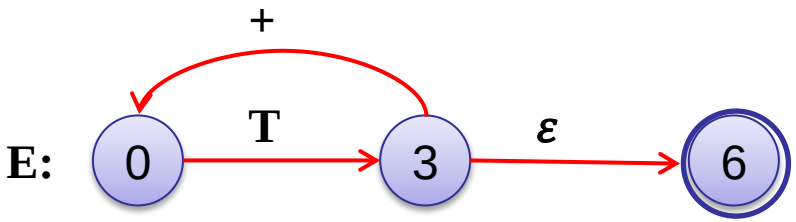
Diagramas de transição para analisadores sintáticos preditivos

Os diagramas de transição são úteis p/ visualizar analisadores preditivos:
Ex.: Diagramas de transição para os não terminais E e E’.

(a)



(b)



Análise sintática

Gramáticas LL(1)

Para construir o diagrama de transição de uma gramática, primeiro deve-se eliminar a recursão a esquerda e faturar a gramática a esquerda.

Revisão: Eliminar recursão a esquerda.

G = (N, T, P, S)

P: $S \rightarrow A\ a$

$A \rightarrow S\ b \mid c\ A \mid a$

P': $S \rightarrow A\ a$

$A \rightarrow A\ a\ b \mid c\ A \mid a$

P'': $S \rightarrow A\ a$

$A \rightarrow c\ A\ A' \mid a\ A'$

$A' \rightarrow a\ b\ A' \mid \varepsilon$

Slides da revisão

Análise sintática

Gramáticas LL(1)

Para construir o diagrama de transição de uma gramática, primeiro deve-se eliminar a recursão a esquerda e fatorar a gramática a esquerda.

Revisão: Fatorar a esquerda.

Exemplo 01: $A \rightarrow a\ B \mid a\ C$

P': $A \rightarrow a\ A'$, $A' \rightarrow B \mid C$

Exemplo 02: $S \rightarrow A \mid B$, $A \rightarrow a\ c$, $B \rightarrow a\ b$

P': $S \rightarrow A \mid B$, $A \rightarrow a\ c$, $B \rightarrow a\ b$

P': $S \rightarrow a\ A \mid a\ B$, $A \rightarrow c$, $B \rightarrow b$

P': $S \rightarrow a\ S'$, $S' \rightarrow A \mid B$, $A \rightarrow c$, $B \rightarrow d$

Slides da revisão

Análise sintática

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

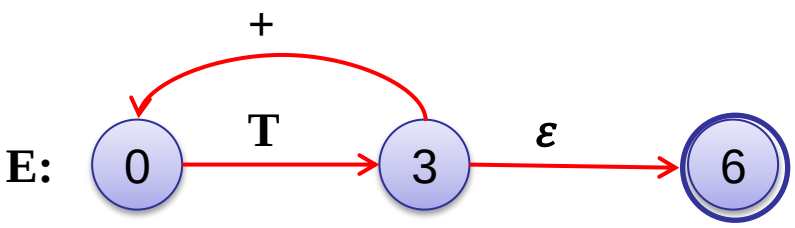
Gramáticas LL(1)

Para construir o diagrama de transição de uma gramática, primeiro deve-se eliminar a recursão a esquerda e fatorar a gramática a esquerda.

Então, para cada não terminal A:

1. Crie os estados inicial e final (retorno).
2. Para cada produção-A $\rightarrow X_1 X_2 \ldots X_k$, crie um caminho do estado inicial para o estado final com as arestas rotuladas com $X_1, X_2, \ldots, X_k$.

   Se A $\rightarrow \varepsilon$, caminho é uma aresta rotulada com ε.

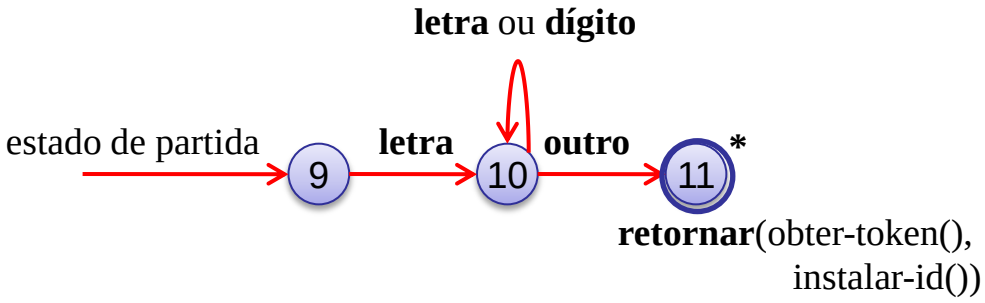


Análise sintática

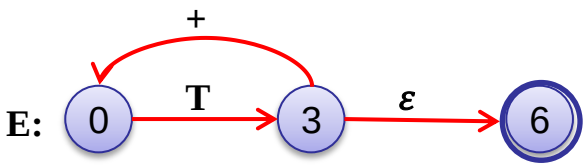
Gramáticas LL(1)

Diagramas de transição para os analisadores preditivos diferem daqueles para os analisadores léxicos.

Analisador léxico:



Analisador sintático:



Análise sintática

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

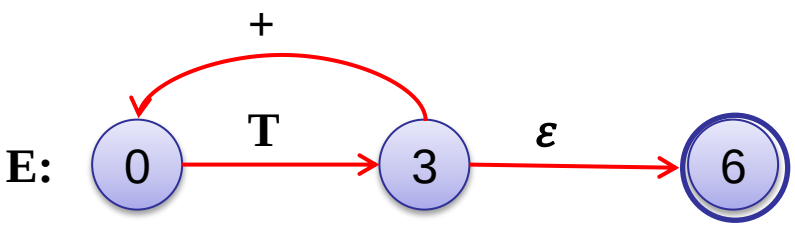
Gramáticas LL(1)

Analisadores sintáticos possuem um diagrama para cada não-terminal.

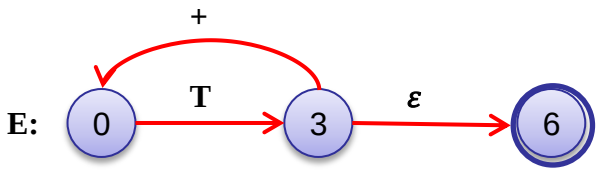
Os rótulos das arestas podem ser *tokens* ou não-terminais.

Uma transação sob um *token* (terminal) significa que efetuaremos essa transação se esse *token* for o próximo símbolo de entrada.

Assim, uma transação sob um não-terminal A representa a ativação do procedimento A.



Análise sintática



Gramáticas LL(1)

Diagramas de transição podem ser simplificados, desde que a sequencia de símbolos da gramática ao longo dos caminhos seja preservada.

Também podemos substituir o diagrama de um não-terminal A por uma aresta rotulada com A.

A classe de gramáticas LL(1) é rica o suficiente para reconhecer a maioria das construções presentes nas linguagens de programação, mas escrever uma gramática adequada p/ a linguagem fonte não é uma tarefa simples.

Exemplo: nenhuma gramática com recursão a esquerda ou ambígua pode ser LL(1).

Análise sintática

Gramáticas LL(1)

A gramática G é LL(1) se e somente se, sempre que $A \rightarrow \alpha \mid \beta$ forem duas produções distintas de G, as seguintes condições serão verdadeiras:

1. Para um terminal a, tanto α quanto β não derivam cadeias começando com a.
2. No máximo um dos dois, α ou β, pode derivar a cadeia vazia.
3. Se $\beta \overset{*}{\Rightarrow} \varepsilon$, então α não deriva nenhuma cadeia vazia começando com um não-terminal em *FOLLOW(A)*. De modo semelhante, se $\alpha \overset{*}{\Rightarrow} \varepsilon$, então β não deriva qualquer cadeia começando com um terminal em *FOLLOW(A)*.

Assim, as duas primeiras condições são equivalentes a declaração de que *FIRST(α)* e *FIRST(β)* são conjuntos disjuntos.

A intersecção dos conjuntos *FIRST(α)* e *FIRST(β)* é vazia.

Análise sintática

Gramáticas LL(1)

A gramática G é LL(1) se e somente se, sempre que $A \rightarrow \alpha \mid \beta$ forem duas produções distintas de G, as seguintes condições serão verdadeiras:

1. Para um terminal a, tanto α quanto β não derivam cadeias começando com a.
2. No máximo um dos dois, α ou β, pode derivar a cadeia vazia.
3. Se $\beta \overset{*}{\Rightarrow} \varepsilon$, então α não deriva nenhuma cadeia vazia começando com um não-terminal em *FOLLOW(A)*. De modo semelhante, se $\alpha \overset{*}{\Rightarrow} \varepsilon$, então β não deriva qualquer cadeia começando com um terminal em *FOLLOW(A)*.

A terceira condição é equivalente a afirmar que, se ε está em *FIRST(β)*, *FIRST(α)* e *FOLLOW(A)* são conjuntos disjuntos, e, da mesma forma, se ε estiver em FIRST(α), ou seja, a intersecção de *FIRST(α)* com *FOLLOW(A)* é vazia. Da mesma forma *FIRST(β)* com *FOLLOW(A)* também é vazia.

Análise sintática

Gramáticas LL(1)

Analisadores preditivos podem ser construídos para as gramáticas LL(1), pois é possível selecionar a produção apropriada a ser aplicada para um não-terminal examinando-se só o símbolo corrente da entrada restante.

Os construções de fluxo de controle, com suas distintas palavras-chave, geralmente satisfazem as restrições das gramáticas LL(1). Exemplo:

stmt → **if** (expr) stmt **else** stmt
| **while** (expr) stmt
| { stmt_list }

então as palavras-chave **if**, **while** e o símbolo **{** nos dizem qual é a única alternativa que possivelmente poderia ser bem-sucedida se estivermos procurando um stmt.

Análise sintática

Gramáticas LL(1)

Próximo algoritmo coleta as informações dos conjuntos *FIRST* e *FOLLOW* em uma tabela de reconhecimento sintático preditivo M[A, a], um arranjo bidimensional, onde A é um não-terminal e a é um terminal ou o símbolo $, o marcador de fim de entrada.

O algoritmo é baseado na ideia a seguir: a produção $A \rightarrow \alpha$ é escolhida se o próximo símbolo de entrada a estiver em *FIRST(α)*.

A única complicação ocorre quando $\alpha = \varepsilon$ ou, generalizando, $\alpha \overset{*}{\Rightarrow} \varepsilon$.

Nesse caso, devemos novamente escolher $A \rightarrow \alpha$, se o símbolo corrente da entrada estiver em *FOLLOW(A)*, ou se o $ foi alcançado e $ está em *FOLLOW(A)*.

Análise sintática

Gramáticas LL(1)

Algoritmo: Construção da tabela para o reconhecedor sintático preditivo.

Entrada: Gramática G.
Saída: Tabela de análise M.
Método: Para cada produção $A \rightarrow \alpha$ da gramática, faça:

1. Para cada terminal a em *FIRST(A)*, inclua $A \rightarrow \alpha$ em M[A, a].

2. Se ε pertence a FIRST(α), inclua $A \rightarrow \alpha$ em M[A, b] para cada terminal b em FOLLOW(A). Se ε pertence a FIRST(α) e $ pertence a FOLLOW(A), acrescente também $A \rightarrow \alpha$ em [A, $].

Se, depois de realizar esses passos, não houver produção alguma em M[A, a], então defina M[A, a] como *error* (que normalmente representamos por uma entrada vazia na tabela).

Análise sintática

Gramáticas LL(1)

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

FIRST(F) = *FIRST(T)* = *FIRST(E)* = {*(*, *id*}

FIRST(E’) = {+, ε}

FIRST(T’) = {*, ε}

FOLLOW(E) = *FOLLOW(E’)* = {*)*, *$*}

FOLLOW(T) = *FOLLOW(T’)* = {+, *)*, *$*}

FOLLOW(F) = {+, *, *)*, *$*}.

Análise sintática

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

Gramáticas LL(1)

Usando o algoritmo anterior para produzir uma tabela de saída...

Entradas em branco associadas aos símbolos da gramática representam erro nesta tabela, as demais entradas indicam a produção usada para expandir um não-terminal.

Considere a produção E →TE’. Visto que FIRST(TE’) = FIRST(T) = {(, id} essa produção é incluída em M[E, (] e M(E, id).

A produção E’→+TE’ é incluída em M[E’, +], desde que FIRST(+TE’) = {+}.

Visto que no FOLLOW(E’) = {), $}, a produção E’ → ε é incluída em M[E’,)] e em M[E’, $].

Análise sintática

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

Gramáticas LL(1)

Usando o algoritmo anterior para produzir uma tabela de saída...

Não-terminal	Símbolos de entrada					
	id	+	*	(	)	$
E	E → TE’			E → TE’		
E’		E’ → +TE’			E’ → ε	E’ → ε
T	T → FT’			T → FT’		
T’		T’ → ε	T’ → *FT’		T’ → ε	T’ → ε
F	F → id			F → (E)		

Análise sintática

Gramáticas LL(1)

O algoritmo em questão pode ser aplicado a qualquer gramática G para produzir a tabela M de análise correspondente a G.

Para toda gramática LL(1), cada entrada na tabela de análise identifica no máximo uma produção ou sinaliza um erro.

Para algumas gramáticas, contudo, M pode conter algumas entradas que possuem definição múltipla. Ex., se G possui recursão a esquerda ou for ambígua, então M terá pelo menos uma entrada com definição múltipla.

Embora a eliminação da recursão à esquerda e a fatoração à esquerda sejam fáceis de fazer, demos observar que algumas gramáticas livres de contexto não têm gramáticas equivalentes LL(1), e nesse caso nenhuma alteração produzirá uma gramática LL(1).

Análise sintática

Gramáticas LL(1)

Usando o algoritmo anterior para produzir uma tabela de saída, entretanto, a linguagem no exemplo abaixo não possui uma gramática LL(1) para ela.

$$
\begin{aligned}
S &\rightarrow iEtSS' \mid a \\
S' &\rightarrow eS \mid \varepsilon \\
E &\rightarrow b
\end{aligned}
$$

Essa gramática representa uma abstração do *else* vazio.

A entrada M[S', e] contém duas produções $S' \rightarrow eS$ e $S' \rightarrow \varepsilon$.

A gramática é ambígua e a ambiguidade se manifesta pela escolha sobre que produção usar quando um e (*else*) é visto.

Análise sintática

S → iEtSS' | a
S' → eS | ε
E → b

Gramáticas LL(1)

Podemos resolver essa ambiguidade escolhendo S' → eS. Essa escolha corresponde à associação de um *else* com o *then* anterior mais próximo.

Observe a escolha S' → ε impede que e seja até mesmo colocado na pilha ou removido da entrada, e com certeza está errada.

Não-terminal	Símbolos de entrada					
	a	b	e	I	t	$
S	S → a			S → iEtSS'		
S'						S' → ε
E		E → b	S' → eS S' → ε			

Próxima aula

Vamos tentar um método diferente na próxima aula.

Vamos realizar a próxima aula por EaD.

Será deixado um material no EaD.

Entretanto, teremos mais um lista de exercícios (questionário) para ser resolvida até o dia 07/10/2013.

A resolução do questionário caracteriza presença nesta aula.

Depois teremos uma enquete para que a metodologia usada nessa aula possa ser avaliada.

COMPILADORES

Obrigado!!

Prof. Geovane Griesang
geovanegriesang@unisc.br