Análise sintática descendente preditiva. Parte I – análise recursiva

- Professor Eraldo Pereira Marinho
- Compiladores 2º semestre 2020
- UNESP/IGCE Rio Claro

Noções preliminares – árvore de derivação

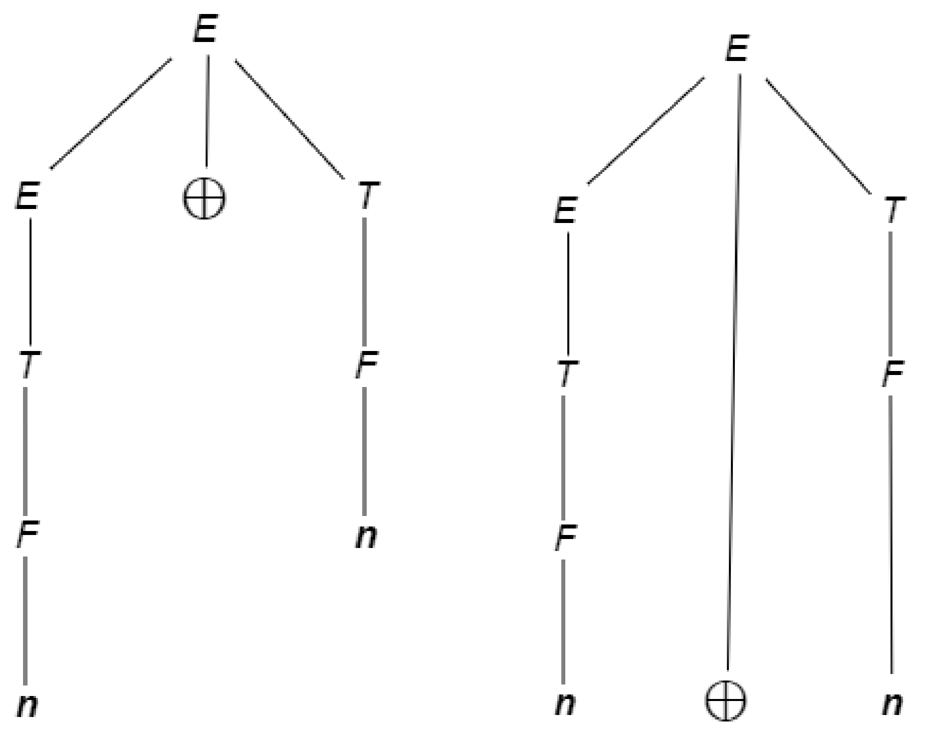
$$E \rightarrow E \oplus T \mid T$$
$$T \rightarrow T \otimes F \mid F$$
$$F \rightarrow (E) \mid \boldsymbol{n}$$

Derivação esquerda (usual):

$$E \Rightarrow^e E \oplus T \Rightarrow^e T \oplus T$$
$$\Rightarrow^e F \oplus T \Rightarrow^e \boldsymbol{n} \oplus T$$
$$\Rightarrow^e \boldsymbol{n} \oplus F \Rightarrow^e \boldsymbol{n} \oplus \boldsymbol{n}$$

Derivação direita (canônica):

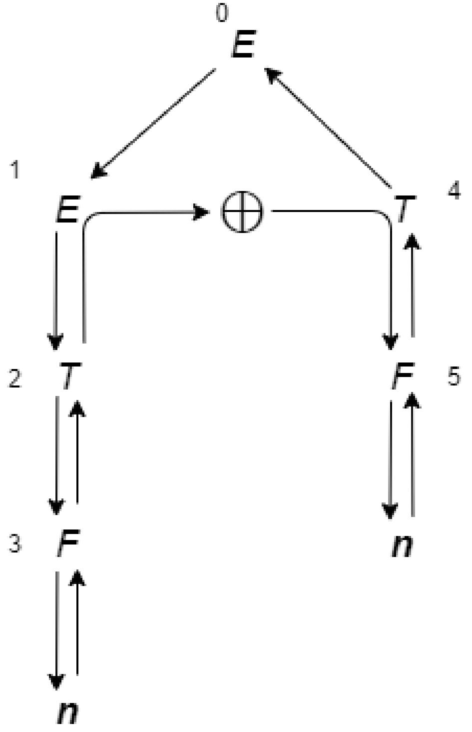
$$E \Rightarrow^d E \oplus T \Rightarrow^d E \oplus F$$
$$\Rightarrow^d E \oplus \boldsymbol{n} \Rightarrow^d T \oplus \boldsymbol{n}$$
$$\Rightarrow^d F \oplus \boldsymbol{n} \Rightarrow^d \boldsymbol{n} \oplus \boldsymbol{n}$$

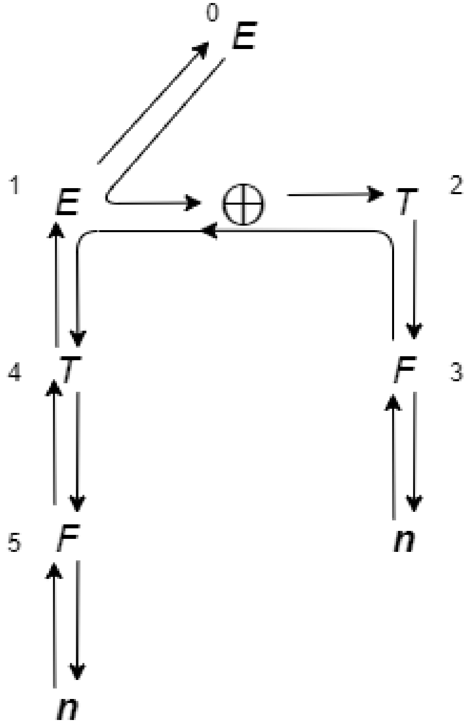


Noções preliminares – caminhamento na árvore de derivação

$$E^0 \Rightarrow E^1 \oplus T \Rightarrow T^2 \oplus T \Rightarrow F^3 \oplus T$$
$$\Rightarrow \boldsymbol{n} \oplus T^4 \Rightarrow \boldsymbol{n} \oplus F^5 \Rightarrow \boldsymbol{n} \oplus \boldsymbol{n}$$

$$E^0 \Rightarrow^d E^1 \oplus T^2 \Rightarrow^d E^1 \oplus F^3$$
$$\Rightarrow^d E^1 \oplus \boldsymbol{n} \Rightarrow^d T^4 \oplus \boldsymbol{n}$$
$$\Rightarrow^d F^5 \oplus \boldsymbol{n} \Rightarrow^d \boldsymbol{n} \oplus \boldsymbol{n}$$

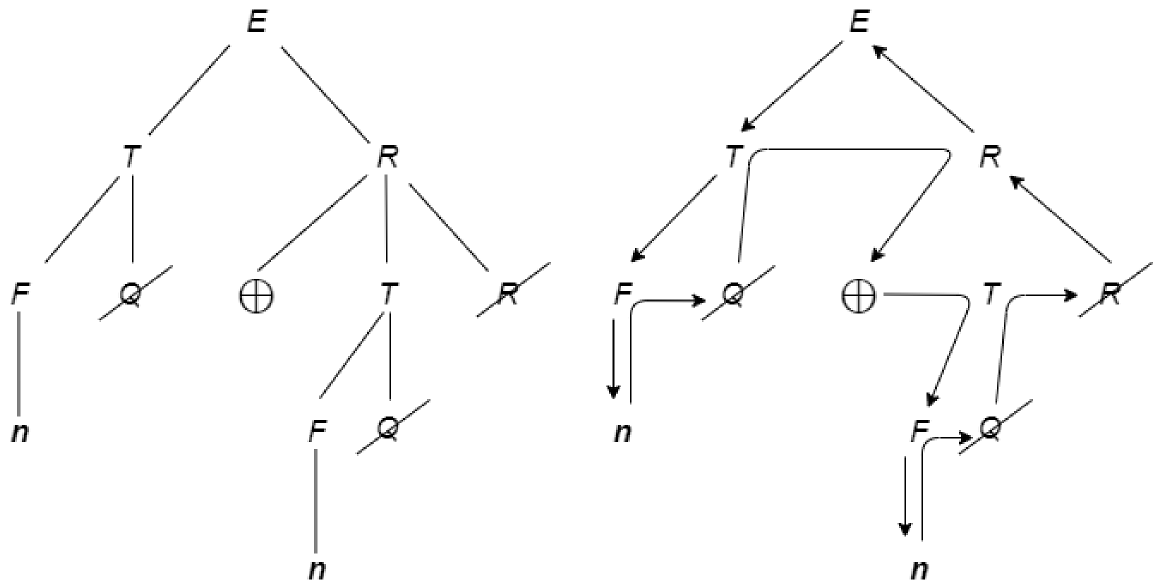




Noções preliminares – caminhamento na árvore de derivação

$$E \rightarrow TR$$
$$T \rightarrow FQ$$
$$R \rightarrow \bigoplus TR \mid \varepsilon$$
$$Q \rightarrow \bigotimes FQ \mid \varepsilon$$
$$F \rightarrow (E) \mid \boldsymbol{n}$$

$$E \Rightarrow TR \Rightarrow FQR \Rightarrow \boldsymbol{n}QR \Rightarrow \boldsymbol{n}R$$
$$\Rightarrow \boldsymbol{n} \oplus TR \oplus FQR \Rightarrow \boldsymbol{n} \oplus \boldsymbol{n}QR$$
$$\Rightarrow \boldsymbol{n} \oplus \boldsymbol{n}R \Rightarrow \boldsymbol{n} \oplus \boldsymbol{n}$$



Noções preliminares – determinismo na escolha do lado direito

Percebemos que a gramática

$$E \to TR$$
$$T \to FQ$$
$$R \to \oplus\, TR \mid \varepsilon$$
$$Q \to \otimes\, FQ \mid \varepsilon$$
$$F \to (E) \mid \boldsymbol{n}$$

tem uma certa vantagem, se pensarmos em desenvolver uma máquina que emule o processo de derivação, em comparação com

$$E \to E \oplus T \mid T$$
$$T \to T \otimes F \mid F$$
$$F \to (E) \mid \boldsymbol{n}$$

Por que? A resposta é que, nesta última, uma tal máquina não tem como decidir de forma simples o que será escolhido, já da primeira produção. $E \to E \oplus T$ ou $E \to T$?

Suponhamos, novamente, que queremos derivar $\boldsymbol{n} \oplus \boldsymbol{n}$ usando esta última gramática. Qual produção a máquina escolheria primeiramente se a única informação que essa teria era que o token lido antecipadamente seria $\tau = \boldsymbol{n}$? Ou seja, $E \Rightarrow T$ ou $E \Rightarrow E \oplus T$? A máquina não saberia qual dessas produções escolheria. Daí viria uma alternativa que seria a heurística: $E \Rightarrow T \Rightarrow F \Rightarrow \boldsymbol{n}$, mas aí a máquina receberia um novo token, que seria $\tau = \oplus$. Aí, a máquina teria que devolver esse token e voltar a $\tau = \boldsymbol{n}$ e escolher a segunda opção, $E \Rightarrow E \oplus T \Rightarrow T \oplus T \Rightarrow F \oplus T \Rightarrow \boldsymbol{n} \oplus T \Rightarrow \boldsymbol{n} \oplus F \Rightarrow \boldsymbol{n} \oplus \boldsymbol{n}$. O mesmo já não ocorreria com a primeira gramática.

Forma normal de Greibach

- Os primeiros passos para um estudo sistemático da análise sintática é a análise sintática descendente recursiva, que requer uma forma normal de gramática livre do contexto, as gramáticas LL(1)
- Comecemos pela definição da forma normal de Greibach:
- Uma gramática $G = (N, \Sigma, S, P)$, livre do contexto, é dita na forma normal de Greibach se e somete se suas produções são todas da forma $A \rightarrow \boldsymbol{\sigma}\alpha$, onde $\boldsymbol{\sigma} \in \Sigma$ e $\alpha \in (\Sigma \cup N)^*$. Em outras palavras, o lado direito das produções deve ser explicitamente iniciados por terminais, $\in \Sigma$
- O teorema de Greibach garante que toda boa gramática pode ser rescrita na forma normal de Greibach

Teorema de Greibach

Teorema 1. Seja $G = (N, \Sigma, S, P)$ uma boa gramática. Neste caso, é sempre possível apresentar uma gramática $\mathrm{G}' = (N', \Sigma, S, P')$, $N \subseteq N'$ com suas produções P' na forma normal de Greibach.

Prova. Sendo G uma boa gramática, então não há produções nulas, nem produções unitárias e muito menos produções quebradas, sem vínculos. Contudo, pode haver produções na forma

$$A \to A\alpha \mid \beta; \quad A \in N, \alpha, \beta \in (\Sigma \cup N)^*, \beta \neq A\gamma$$

Por indução, temos que A pode derivar as seguintes situações: $A \Rightarrow \beta$, $A \Rightarrow A\alpha \Rightarrow \beta\alpha$ e, para um número $n + 1$ de passos de derivação, $A \Rightarrow^n A\alpha^n \Rightarrow \beta\alpha^n$. Assim, podemos dizer que existe um $R \in N'$ tal que $R \Rightarrow^n \alpha^n$, o que é possível se $R \to \alpha R \mid \alpha$. Neste caso, podemos substituir as produções acima pelas novas produções

$$A \to \beta R \mid \beta, \qquad R \to \alpha R \mid \alpha$$

A demonstração ainda não está completa pois pode haver mais de uma produção com recursão esquerda em A. Suponhamos que existam as produções

$$A \to A\alpha_1 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \cdots \mid \beta_n$$

É fácil generalizar o raciocínio anterior para mostrarmos que

$$A \Rightarrow^* A\alpha_{j_k} \cdots \alpha_{j_2}\alpha_{j_1} \Rightarrow \beta_i \alpha_{j_k} \cdots \alpha_{j_2}\alpha_{j_1}$$

Observemos que j_k é um inteiro qualquer em $\{1, 2, \ldots, m\}$ invocado no último nível de recursão (substituição), nível k. Desta última indução, temos que um novo conjunto de substituições será necessário para eliminarmos a recursão esquerda:

$$A \to \beta_1 R \mid \cdots \mid \beta_n R \mid \beta_1 \mid \cdots \mid \beta_n, \qquad R \to \alpha_1 R \mid \cdots \mid \alpha_m R \mid \alpha_1 \mid \cdots \mid \alpha_m$$

Caso haja nova recursão esquerda nos β, o método acima é reaplicado até não mais recursão esquerda. Quando não há recursão esquerda, é possível obter, eventualmente, β_i na forma $\beta_i = B\gamma_i$.

Teorema de Greibach - continuação

Dando continuidade à demonstração, produções do tipo $A \to B\alpha$, com $B \not\Rightarrow^* A\gamma$, devem ser recursivamente substituídas até não mais haver produções com lado direito iniciando com não terminal. Uma vez que não temos mais recursão esquerda, usamos o seguinte algoritmo:

Entrada: uma gramática LC sem recursão esquerda, $G = (N, \Sigma, S, P)$

Saída: uma gramática na FNG, $G' = (N, \Sigma, S, P')$

Método:

Inicialize P' com todas as produções P que já estejam na FNG

Repita

    Para toda produção de P, na forma $A \to B\alpha$, faça:

        Para toda produção $B \to \beta$ de P', faça:

            Acrescente $A \to \beta\alpha$ ao conjunto P'

        Fim de faça

    Fim de faça

Até não haver mais alterações em P'

Fácil verificar que, ao fim deste algoritmo, todas as produções de P' já se encontram na FNG ■

Uma coisa interessante no algoritmo acima é que o mesmo não funciona para gramáticas com recursão esquerda (por que?).

Exemplo

Seja a gramática LR(1) de expressões:

$$E \to E + T \mid E - T \mid T$$
$$T \to T * F \mid T \div F \mid F$$
$$F \to (E) \mid \boldsymbol{n}$$

E é o símbolo inicial e $\boldsymbol{n}$ é uma constante numérica qualquer. Com base na demonstração do teorema anterior, temos que

$$E \to TR, \qquad R \to +TR \mid -TR \mid \varepsilon$$
$$T \to FQ, \qquad Q \to * FQ \mid \div FQ \mid \varepsilon$$
$$F \to (E) \mid \boldsymbol{n}$$

Substituindo os prefixos com F nas demais produções, e observando que $T \Rightarrow^* F$, temos

$$E \to (E)QR \mid \boldsymbol{n}QR, \qquad R \to +TR \mid -TR \mid \varepsilon$$
$$T \to (E)Q \mid \boldsymbol{n}Q, \qquad Q \to * FQ \mid \div FQ \mid \varepsilon$$
$$F \to (E) \mid \boldsymbol{n}$$

Que já estão basicamente na forma normal de Greibach, a menos das produções épsilon. Contudo, se quisermos eliminar essas produções, iremos praticamente dobrar a complexidade espacial da gramática. O que importa é que todas as produções iniciam com terminais, o que era nosso objetivo neste exemplo.

Gramática LL(1) – informal

A gramática de expressões do exercício anterior não precisava ficar explicitamente na forma normal de Greibach – seria suficiente eliminar as recursões esquerdas para atingirmos nosso próximo propósito. Tomemos a forma intermediária da gramática de expressões, onde não havia mais recursão esquerda:

$$E \rightarrow TR,$$
$$R \rightarrow +TR \mid -TR \mid \varepsilon$$
$$T \rightarrow FQ,$$
$$Q \rightarrow * FQ \mid \div FQ \mid \varepsilon$$
$$F \rightarrow (E) \mid \boldsymbol{n}$$

Agora, observemos que a topologia de árvore de derivação, induzida pelas produções acima, é a mesma da hierarquia de chamada e implementação de funções C: $E \rightarrow TR$ pode ser substituída por `void E(void){ T(); R(); }`. O mesmo se dá para as produções que partem de R:

```
void R(void) {
       if (lookahead == '+') {
                 match ('+'); T(); R();
       } else if (lookahead == '-') {
                 match ('-'); T(); R();
       } else {
                 /*simula épsilon, caso venha lookahead == EOF ou lookahead == ')' */
                 ;
       }
}
```

O operador match

O operador match, que aparece na listagem anterior para simular as produções que partem de R tem o seguinte template:

```
1: void match(int expected_token) {
2:         if (lookahead == expected_token) {
3:                 lookahead = gettoken(source);
4:         } else {
5:                 fprintf(stderr, "token mismatch\n");
6:                 exit(ERRTOKEN);
7:         }
8: }
```

Então, conforme a listagem acima, match é a forma com que o parser solicita o próximo token, armazenador no operador lookahead. A parte correspondente ao erro não precisa ser do jeito que aparece nas linhas 5 e 6, e podem ser muito mais sofisticadas.

A função FIRST

A gramática de expressões, conhecida na forma LL(1), foi escrita anteriormente como uma fase intermediária para passar a forma LR(1) para a forma normal de Greibach. Retomemos então a gramática LL(1) de expressões:

$$E \to TR, \qquad R \to +TR \mid -TR \mid \varepsilon$$
$$T \to FQ, \qquad Q \to * FQ \mid \div FQ \mid \varepsilon$$
$$F \to (E) \mid \boldsymbol{n}$$

Confrontemos com a forma normal de Greibach:

$$E \to (E)QR \mid \boldsymbol{n}QR, \qquad R \to +TR \mid -TR \mid \varepsilon$$
$$T \to (E)Q \mid \boldsymbol{n}Q, \qquad Q \to * FQ \mid \div FQ \mid \varepsilon$$
$$F \to (E) \mid \boldsymbol{n}$$

Observemos que, desta última forma, os primeiros tokens que podem ocorrer numa sentença válida gerada por esta gramática são '(' e $\boldsymbol{n}$. Assim, uma sentença como "$3 + 4$" é válida, enquanto uma sentença como "$)3 -+ -4$" não é válida, não somente pelos posteriores erros de sintaxe, pela repetição de operadores aditivos, mas por iniciar a mesma com ')'. Neste caso dizemos que '(' e n são membros de $\mathrm{FIRST}(E)$. Adicionalmente, é fácil verificar que $\mathrm{FIRST}(E) = \mathrm{FIRST}(T) = \mathrm{FIRST}(F) = \{(, \boldsymbol{n}\}$. De modo análogo, as cadeias derivadas de R iniciam com os tokens em $FIRST(R) = \{\varepsilon, +, -\}$.

Com base no que foi ensaiado acima, ε aparece como FIRST. Apesar de não fazer parte do alfabeto, ε é considerado um terminal, um terminal nulo. função $\mathrm{FIRST}: (\Sigma \cup N)^* \mapsto 2^{\Sigma \cup \{\varepsilon\}}$, dada a gramática $G = (N, \Sigma, S, P)$, é a função que leva uma forma sentencial $\alpha \in (\Sigma \cup N)^*$ numa coleção de tokens em $2^{\Sigma \cup \{\varepsilon\}}$, ou seja, $\mathrm{FIRST}(\alpha) \subseteq \Sigma \cup \{\varepsilon\}$. No exemplo, $\Sigma = \{(,), \boldsymbol{n}, +, -, *, \div\}$.

A função FIRST

Sumariamente, podemos definir a função FIRST conforme o seguinte algoritmo:

Entrada: uma gramática LL(1), $G = (N, \Sigma, S, P)$, um não símbolo de $\Sigma \cup N \cup \{\varepsilon\}$ ou uma forma sentencial α de $(N \cup \Sigma)^*$, onde $A \to \alpha$ é uma produção de P

Saída: Função $\mathrm{FIRST}(X)$

Método:

Inicialize $\mathrm{FIRST}(X) = \emptyset$

Se $X \in N$ e $X \to \varepsilon \in P$, adicione ε a $\mathrm{FIRST}(X)$

Se $X = \alpha_1\alpha_2 \cdots \alpha_n$, para algum $\mathrm{A} \to X \in P$, ou $X \to \alpha_1 \cdots \alpha_n$, então

    Se $\alpha_i \Rightarrow^* \varepsilon$, para $i = 1, \ldots, j-1 < n$, com $\alpha_j \neq \varepsilon$, inclua $\mathrm{FIRST}(\alpha_j)$ em $\mathrm{FIRST}(X)$

Senão

    Se $X \in \Sigma \cup \{\varepsilon\}$, adicione X a $\mathrm{FIRST}(X)$

Fim de se

Retorne $\mathrm{FIRST}(X)$

Fim

A função FOLLOW

Mais uma observação decorre do fato de que tanto R quanto Q retratam sentenças parciais, aparecendo como sufixo de T e como sufixo de F nas produções que partem de E e T, respectivamente.

Retomemos a versão LL(1) da gramática de expressões aritméticas:

$$E \to TR, \qquad R \to +TR \mid -TR \mid \varepsilon$$
$$T \to FQ, \qquad Q \to * FQ \mid \div FQ \mid \varepsilon$$
$$F \to (E) \mid \boldsymbol{n}$$

Observemos que a primeira invocação do símbolo inicial E corresponde à raiz absoluta de uma árvore de derivação para, por exemplo, a expressão $\boldsymbol{n} * (\boldsymbol{n} + \boldsymbol{n})$. Neste caso, dizemos que $E \Rightarrow^* \boldsymbol{n} * (\boldsymbol{n} + \boldsymbol{n})$. Contudo, se quisermos enxergar a sentença produzida por uma gramática, incluindo o caráter de fim de sentença, $\$$, temos que, antes da invocação, o texto é simplesmente $\$$, o que significa que o texto é nulo, ou mais especificamente $\$ = \varepsilon\$$. Ao fim da derivação, o texto é visto, em baixo nível, como $\boldsymbol{n} * (\boldsymbol{n} + \boldsymbol{n})\$$. Fazendo uma redução na árvore, de baixo para cima, até chegarmos no E, isso equivale a ter escrito $E\$$. Ou seja, tudo que é derivado da invocação inicial do símbolo inicial é finalizado por $. Então, dizemos que $ segue E, ou simplesmente que $\$ \in \mathrm{FOLLOW}(E)$.

Raciocínio semelhante se dá com o que ocorre com (E), quando E é invocado de dentro dos parênteses, vemos que tudo que for derivado de E é finalizado com $)$. Assim, como não outro símbolo seguindo E nas formas sentenciais da direita, temos finalmente $\mathrm{FOLLOW}(E) = \{\$,)\}$.

Por outro lado, observe que $E \Rightarrow TR\$$ ou $E \Rightarrow TR)$ pode ocorrer ao longo da derivação de expressões. Então, tudo que for $\mathrm{FOLLOW}(E)$ servirá como $\mathrm{FOLLOW}(T)$ e é exatamente $\mathrm{FOLLOW}(R) = \mathrm{FOLLOW}(E)$, ou seja, $\mathrm{FOLLOW}(E) \subset \mathrm{FOLLOW}(T)$, o que ocorre quando $R \Rightarrow \varepsilon$. Contudo, T em TR vem seguido de tudo que estiver em $\mathrm{FIRST}(R)$ exceto ε. Então, $+$ e $-$ pertencem a $\mathrm{FOLLOW}(T)$. Como a única aparição explícita de T ocorre somente em TR, concluímos que $\mathrm{FOLLOW}(T) = \mathrm{FOLLOW}(R) \cup \mathrm{FIRST}(R) - \{\varepsilon\} = \{+, -,), \$\}$.

A função FOLLOW

Raciocínio semelhante ao que fizemos acima ocorre com Q. Como $T \to FQ$, tudo que segue T segue Q, e esta é a única aparição de Q partindo de outro não terminal, então $\mathrm{FOLLOW}(Q) = \mathrm{FOLLOW}(T) = \{+, -,), \$\}$.

Generalizando, temos a definição de $\mathrm{FOLLOW}: N \mapsto 2^{\Sigma \cup \{\$\}}$, dada a gramática $G = (N, \Sigma, S, P)$, como a função que transforma um não terminal $A \in N$ conforme o seguinte algoritmo:

1. Para cada não terminal $A \in N$, inicialize $\mathrm{FOLLOW}(A) = \emptyset$;
2. Se $A = S$, adicione \$ a $\mathrm{FOLLOW}(A)$;
3. Se A aparece numa produção do tipo $X \to \alpha A$, inclua $\mathrm{FOLLOW}(X)$ a $\mathrm{FOLLOW}(A)$;
4. Se A aparece no lado direito de uma produção de P, na forma $X \to \alpha A \beta$, adicione $\mathrm{FIRST}(\beta) - \{\varepsilon\}$ a $\mathrm{FOLLOW}(A)$ – neste caso, se $\varepsilon \in \mathrm{FIRST}(\beta)$, inclua $\mathrm{FOLLOW}(X)$ a $\mathrm{FOLLOW}(A)$.

É imediato que $\varepsilon \notin \mathrm{FOLLOW}(X)$.

O que vem a ser afinal de contas uma gramática LL(1)?

Uma gramática $G = (N, \Sigma, S, P)$, livre do contexto, é dita LL(1) se é possível implementar um analisador sintático, codificado em C, que simule de forma única o processo de derivação dessa gramática, pela substituição de não terminais por procedimentos/funções C e seus terminais pela invocação do operador match desse terminal na forma de constante predefinida, sem retrocessos, ou nenhuma outra forma heurística, ou seja, de modo determinístico.

Em decorrência, uma gramática LL(1) tem que ter uma chave (token) única para indexar cada lado direito das regras de produção, ou seja, $A \to \alpha |\ \beta$

Corresponde, virtualmente, ao trecho de código abstrato

```
void A(void) {
        if ( lookahead ∈ FIRST(α) ) {
                match(lookahead); /∗ continue denotando a forma sentencial α ∗/
        else if ( lookahead ∈ FIRST(β) ) {
                match(lookahead); /∗ continue denotando a forma sentencial β ∗/
        } else {
                /∗ codifique aqui a rotina de depuração de erro ∗/
        }
}
```

Desta forma, com base no algoritmo acima, o lado direito α é escolhido se e somente se o token previsto, $lookahead \in \mathrm{FIRST}(\alpha)$, enquanto o lado direito β é escolhido se e somente se $lookahead \in \mathrm{FIRST}(\beta)$ o que só é possível se e somente se $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$. Isso torna determinística a escolha do lado direito das produções.

Por exemplo, a gramática $S \to \boldsymbol{a}\ S \mid \boldsymbol{a}\ S\ \boldsymbol{b}\ S \mid \boldsymbol{c}$ não é LL(1). Por que?

Ainda sobre LL(1)

Uma forma conveniente de enxergarmos o processo de análise sintática descendente preditiva recursiva como uma emulação do processo de derivação à esquerda é o seguinte.

Se $S \rightarrow \alpha$ é uma produção inicial, então podemos idealizar a seguinte sucessão de descrições instantâneas:

$$.S \Rightarrow \alpha \vdash S \Rightarrow .\alpha \vdash^* S \Rightarrow \alpha. \equiv S. \Rightarrow \alpha$$

O ponto precedendo um símbolo significa e iminência de visitar o mesmo, sincronizado com $lookahead$ – seja por chamada recursiva do procedimento/função associado, seja pela iminência de invocar o operador match do símbolo esperado em $\mathrm{FIRST}(\alpha)$. O ponto após uma forma sentencial significa que o último componente, apêndice, foi visitado e isso significa o retorno de uma recursão ou saída do operador match.

Se tivermos uma produção com a seguinte configuração instantânea,

$$X \rightarrow \alpha.A$$

e $\exists\ A \rightarrow \varepsilon \in P$, então, a escolha $X \Rightarrow \alpha.A \Rightarrow \alpha.\varepsilon \vdash \alpha\varepsilon. \equiv \alpha. \equiv X.$ é feita se e somente se $lookahead$, abstraindo o ponto prefixando A, não pertence a $\mathrm{FIRST}(A)$ mas pertence a $\mathrm{FOLLOW}(A)$. O que não pode ocorrer é $A \rightarrow \varepsilon$ para $lookahead \in \mathrm{FIRST}(A)$, sabendo que $lookahead \neq \varepsilon$, o que tornaria possível ter lookahead $\in \mathrm{FIRST}(A) \cap \mathrm{FOLLOW}(A) \neq \emptyset$ – em outras palavras, teríamos um $lookahead$ sendo FIRST e FOLLOW ao mesmo tempo.

Definição (finalmente) de gramática LL(1)

Uma gramática livre do contexto, $G = (N, \Sigma, S, P)$, é dita LL(1) se e somente se:

1. Todas as produções de P não possuem recursão esquerda – nem direta nem cíclica – em outras palavras, $A \not\Rightarrow^* A\alpha$;
2. Uma produção do tipo $A \rightarrow \alpha|\beta$ implica ter $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$;
3. Se $A \rightarrow \alpha|\varepsilon$, com $\alpha \neq \varepsilon$, então a escolha $A \rightarrow \varepsilon$ só possível quando $lookahead \in \text{FOLLOW}(A)$, com $\text{FIRST}(A) \cap \text{FOLLOW}(A) = \emptyset$.

Exemplo de LL(1)

Surge o seguinte teorema:

Teorema 2. A gramática de expressões simplificadas,

$$G = (\{E, T, F, R, Q\}, \{\boldsymbol{n}, (,), +, -, *, \div\}, E, P),$$

com

$$P = \{ E \to TR, R \to +TR, R \to -TR, R \to \varepsilon, T \to FQ, Q \to * FQ, Q \to \div FQ, Q \to \varepsilon, F \to (E), F \to \boldsymbol{n}\}$$

é uma gramática LL(1).

Prova. Segue imediatamente da definição de gramática LL(1) ■

Exemplo de LL(1) – continuação

Tomemos a gramática LL(1) de expressões na sua forma mais abstrata,

$$E \to TR$$
$$R \to \bigoplus TR \mid \varepsilon$$
$$T \to FQ$$
$$Q \to \bigotimes FQ \mid \varepsilon$$
$$F \to (E) \mid \boldsymbol{n}$$

onde $\bigoplus = {}' + {}' \mid {}' - {}'$ e $\bigotimes = {}' * {}' \mid \div$ abstraem os operadores aditivos e multiplicativos, respectivamente.

Neste caso, as possíveis configurações instantâneas para representar a análise sintática descendente preditiva recursiva são

$$.E \to TR \vdash E \to .TR \vdash^* E \to T.R \vdash^* E \to TR. \vdash E. \to TR$$

Ainda

$$.R \to \bigoplus TR \vdash R \to .\bigoplus TR \text{ (se } lookahead = \bigoplus) \vdash R \to \bigoplus .TR \vdash^* R \to \bigoplus T.R \vdash^* R \to \bigoplus TR. \vdash R. \to \bigoplus TR, \text{ ou}$$
$$R \to .\varepsilon \left(\text{se } lookahead \notin \text{FIRST}(R) \wedge lookahead \in \text{FOLLOW}(R)\right) \vdash R \to \varepsilon. \vdash R. \to \varepsilon$$

Mesmo raciocínio com $T \to FQ$ e $Q \to \bigotimes FQ \mid \varepsilon$, mas dando ênfase às produções finais

$$.F \to (E) \vdash F \to .(E) \vdash F \to (.E) \vdash^* F \to (E.) \vdash F \to (E). \vdash F. \to (E)$$
$$.F \to \boldsymbol{n} \vdash F \to .\boldsymbol{n} \vdash F \to \boldsymbol{n}. \vdash F. \to \boldsymbol{n}$$

Exemplo de LL(1) – continuação

Onde as funções FIRST e FOLLOW foram computadas como

$$\text{FIRST}(E) = \text{FIRST}(T) = \text{FIRST}(F) = \{\boldsymbol{n}, (\}$$
$$\text{FIRST}(R) = \{\varepsilon, \oplus\}$$
$$\text{FIRST}(Q) = \{\varepsilon, \otimes\}$$

$\text{FOLLOW}(E) = \{\$,)\}$, visto que podem surgir as situações $E.\$$ e $E.)$

$\text{FOLLOW}(T) = \text{FOLLOW}(E) \cup \text{FIRST}(R) - \{\varepsilon\} = \{\$,), \oplus\}$, visto que em na configuração instantânea $T.R$, a produção $R \to \varepsilon$ ocorre se $lookahead \in \text{FOLLOW}(R)$, ou então T é seguido de $\oplus$, caso $lookahead = \oplus$

$\text{FOLLOW}(R) = \text{FOLLOW}(E) = \{\$,)\}$, uma vez que $E \to TR.$ é seguido de tudo que segue E

$\text{FOLLOW}(Q) = \text{FOLLOW}(T) = \{\$,), \oplus\}$, uma vez que $T \to FQ.$ é seguido de tudo que segue T

Por último, $\text{FOLLOW}(F) = \text{FOLLOW}(T) \cup \text{FIRST}(Q) - \{\varepsilon\}$, visto que em na configuração instantânea $F.Q$, a produção $Q \to \varepsilon$ ocorre se $lookahead \in \text{FOLLOW}(Q)$, ou então F é seguido de $\otimes$, caso $lookahead = \otimes$

É sempre conveniente justificar cada passo de obtenção das funções recursivas acima, principalmente em caso de provas ou de outras atividades para nota.

Exemplo de LL(1) – continuação

Continuando o exemplo, suponhamos agora que queremos derivar 3 + 4, que será visto pelo analisador sintático como $\boldsymbol{n} \oplus \boldsymbol{n}$, utilizando a notação ponto:

$$.E \Rightarrow TR \vdash E \Rightarrow .TR \Rightarrow .FQR\ (lookahead = \boldsymbol{n})$$

$$\Rightarrow .\boldsymbol{n}QR \vdash \boldsymbol{n}.QR\ (lookahead = \oplus) \Rightarrow \boldsymbol{n}.\varepsilon R \vdash \boldsymbol{n}.R$$

$$\Rightarrow \boldsymbol{n}.\oplus TR \vdash \boldsymbol{n} \oplus .TR \Rightarrow \boldsymbol{n} \oplus .FQR\ (lookahead = \boldsymbol{n})$$

$$\Rightarrow \boldsymbol{n} \oplus .\boldsymbol{n}QR \vdash \boldsymbol{n} \oplus \boldsymbol{n}.QR\ (lookahead = \$)$$

$$\Rightarrow \boldsymbol{n} \oplus \boldsymbol{n}.\varepsilon R \vdash \boldsymbol{n} \oplus \boldsymbol{n}.R \Rightarrow \boldsymbol{n} \oplus \boldsymbol{n}.\varepsilon \vdash \boldsymbol{n} \oplus \boldsymbol{n}.$$

$$\vdash E. \Rightarrow TR$$

Este último passo é só para dizer que retornou da recursão inicial, da derivação $.E \Rightarrow TR$

Como converter uma gramática livre do contexto para LL(1)?

Pelo que pudemos perceber da explanação anterior, uma gramática é LL(1) se não tem recursão esquerda e o lado direito de cada produção, digamos, a forma sentencial α em $A \to \alpha$, é única em relação ao token lido antecipadamente em $lookahead$, combinado com o símbolo não terminal, A, da esquerda. Assim, precisamos encontrar uma gramática cujo lado direito das produções $A \to \alpha$ seja uma função $\alpha = f(A, \tau)$, onde τ é o token carregado em $lookahead$. Assim, podemos fazer as seguintes prescrições:

1. Elimine as recursões esquerdas das produções, do mesmo modo que fizemos no início desta aula;

2. Faça a fatoração esquerda nas formas sentenciais direitas das produções – por exemplo, se tivermos $S \to \boldsymbol{a}S \mid \boldsymbol{a}S\boldsymbol{b}S \mid \boldsymbol{c}$, podemos rescrever o lado direito como se fosse uma expressão regular, obtendo $\boldsymbol{a}S \mid \boldsymbol{a}S\boldsymbol{b}S \mid \boldsymbol{c} = \boldsymbol{a}S(\varepsilon \mid \boldsymbol{b}S) \mid \boldsymbol{c}$. Como assumimos como dogma que não iremos misturar expressão regular com gramática, redefinimos o termo entre parênteses como $R \to \boldsymbol{b}S \mid \varepsilon$. Deste modo, a gramática resultante é $S \to \boldsymbol{a}SR \mid \boldsymbol{c}, R \to \boldsymbol{b}S \mid \varepsilon$;

3. Estude FIRST e FOLLOW para saber o conjunto de tokens necessário para selecionar os lados direitos e para saber quando algum não terminal deve se anular – por exemplo, $R \to \varepsilon$, do exemplo dado no Item 2, é necessário sempre que o token τ armazenado em $lookahead$ estiver em $\mathrm{FOLLOW}(R) = \{\$\}$

Observe que a condição do Item 3 remove a ambiguidade que existe na gramática, uma vez que R nunca é zerado quando $lookahead = \boldsymbol{b}$. Por exemplo, derivando $S \Rightarrow .\boldsymbol{a}SR \vdash \boldsymbol{a}.SR \Rightarrow \boldsymbol{a}.\boldsymbol{a}SRR \vdash \boldsymbol{aa}.SRR \vdash^* \boldsymbol{aa}S.RR$ – neste momento, $lookahead$ pode ser \$ ou pode ser $\boldsymbol{b}$. No primeiro caso, temos que $.RR \Rightarrow \varepsilon.R \Rightarrow \varepsilon\varepsilon$. No segundo caso, quando $lookahead = \boldsymbol{b}$, só podemos ter o R prefixado pelo ponto, ficando $.RR \Rightarrow \boldsymbol{b}.R$, o que impede que o primeiro R seja anulado na presença de $lookahead = \boldsymbol{b}$.

Mais um exemplo de analisador descendente recursivo

A gramática

$$S \rightarrow \boldsymbol{a}SR \mid \boldsymbol{c},$$
$$R \rightarrow \boldsymbol{b}S \mid \varepsilon$$

pode ser facilmente transcrita para um analisador sintático recursivo preditivo:

```
void S(void) {
        if (lookahead == a) {
                match(a); S(); R();
        } else {
                match(c);
        }
}
```

```
void R(void) {
        if (lookahead == b) {
                match(b); S();
        } else {
                match(EOF); /* gettoken embutido em
                match vai retornar EOF se chamado
                novamente */
        }
}
```

Testando o exemplo anterior

Para entender o funcionamento de ambas rotinas simulando a referida gramática, tomemos a sequência de tokens $|\boldsymbol{a}|\boldsymbol{a}|\boldsymbol{c}|\$$, onde $\$$ simboliza a constante EOF. Para simplificar, supomos que os espaços foram removidos. A sequência de chamadas é simulada por

$$.S \Rightarrow .\boldsymbol{a}SR \vdash \boldsymbol{a}.SR \Rightarrow \boldsymbol{a}.\boldsymbol{a}SRR \vdash \boldsymbol{aa}.SRR \Rightarrow \boldsymbol{aa}.\boldsymbol{c}RR \vdash \boldsymbol{aac}.RR$$

Neste momento, onde o ponto está prefixando o R mais à esquerda da dupla RR, $lookahead$ já é EOF. Isso corresponde à linha 5 do procedimento $R()$ da listagem anterior.

Então match(EOF) vai ser bem sucedido e vai invocar gettoken, que continuará retornando EOF. Acontece que a linha 5 reproduz $R \to \varepsilon$. Logo, teremos

$$\boldsymbol{aac}.RR \Rightarrow \boldsymbol{aac}.\varepsilon R \vdash \boldsymbol{aac}\varepsilon.R \Rightarrow \boldsymbol{aac}\varepsilon.\varepsilon \vdash \boldsymbol{aac}\varepsilon\varepsilon. \equiv \boldsymbol{aac}.$$

que significa que acabou a análise com sucesso.

Questões valendo 2 horas – vamos ver se prestaram atenção à aula assíncrona?

1. Explique com poucas palavras, e com suas palavras, o que vem a ser a forma normal de Greibach.
2. Por que o algoritmo, utilizado para concluir a demonstração do Teorema de Greibach nem sempre funciona com recursão esquerda? *Dica: teste com a gramática LR(1) de expressões.*
3. Qual a vantagem de se ter a gramática de expressões na forma LL(1) do ponto de vista de implementação simplificada de analisadores sintáticos?
4. Porque $S \to \boldsymbol{a}S \mid \boldsymbol{a}S\boldsymbol{b}S \mid \varepsilon$ não é LL(1)?
5. A gramática $S \to \boldsymbol{a}SR,\ R \to \boldsymbol{b}S \mid \varepsilon$, com S sendo o símbolo inicial, é LL(1)? Justifique.