Análise Sintática

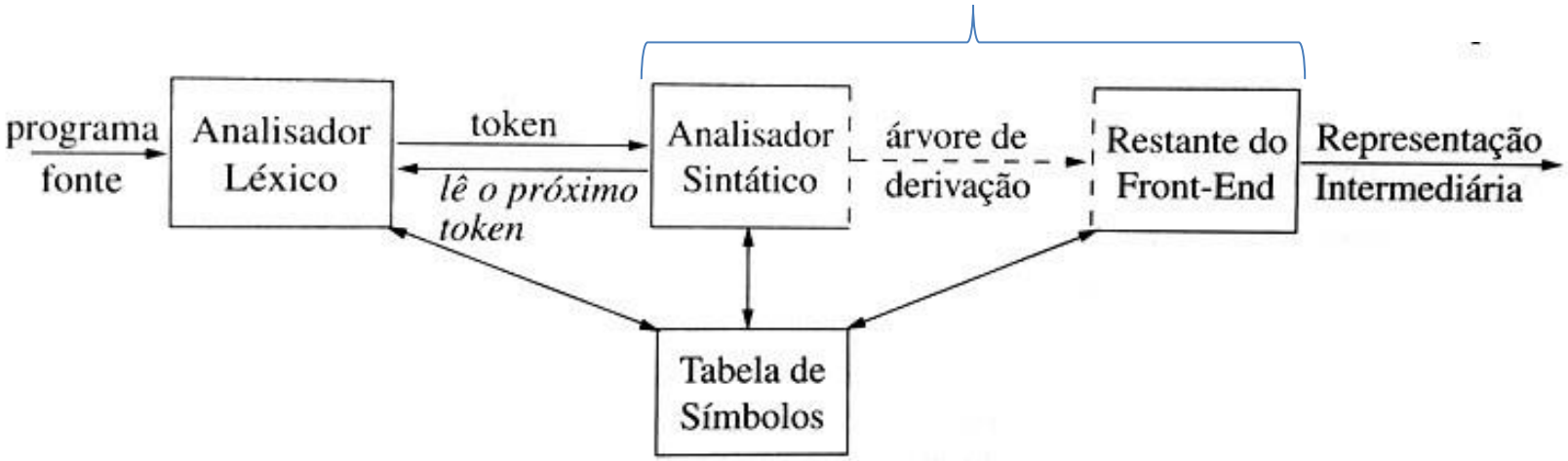
Prof. Leandro I. Pinto

Introdução

- As gramáticas oferecem benefícios significativos para projetistas de linguagens e de compiladores;
  - Provê especificação sintática precisa;
  - A partir de determinadas classes de gramáticas, podemos construir automaticamente um analisador sintático eficiente;
  - Facilita a tradução de programas fonte para código objeto

Analisador sintático

Podem ser implementados juntos



Analisador sintático

- Existem 3 estratégias gerais de análise sintática;
  - Universal
    - Cocke-Younger-Kasami, Earley
  - Descendente
    - Constrói a árvore da raiz até as folhas (derivação);
  - Ascendente
    - Começam nas folhas até alcançar a raiz (redução);
- Métodos descendentes e ascendentes são mais eficientes e, portanto, mais utilizados nos compiladores.
  - Os mais eficientes funcionam apenas para subclasses de gramáticas;
  - Más várias dessas são expressivas o suficiente para as linguagens de programação modernas.

Derivações

- As produções são tratadas como regras de reescrita;
- A partir do símbolo inicial, substitui-se cada não-terminal por uma de suas produções;

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

- Podemos dizer que "E deriva –E" com a notação $E \Rightarrow -E$.
- Ou: $E \Rightarrow (E) \Rightarrow (E + E) \Rightarrow (id + id)$, onde $E \overset{*}{\Rightarrow} (id + id)$ "E deriva (id+id) em vários passos.

Derivações

- Derivação mais à esquerda

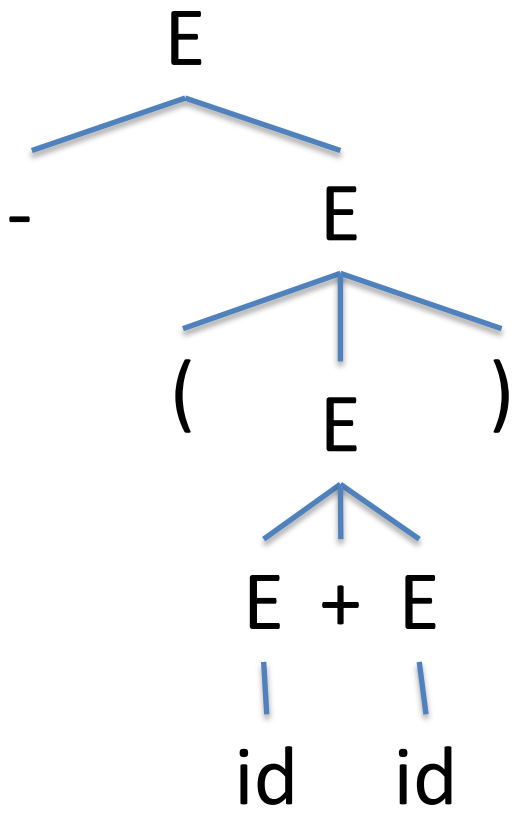
$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(id + E)$$
$$\Rightarrow -(id + id)$$

- Derivação mais à direita

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(E + id)$$
$$\Rightarrow -(id + id)$$

Árvore de Derivação

- Representação gráfica de uma derivação



Ambiguidade

- Uma gramática é ambígua quando produz mais de uma árvore de derivação para alguma sentença;
- Dê duas derivações para $a + b * c$ considerando as produções $E \rightarrow E + E | E * E | - E | (E) | id$.

- Para a maioria dos analisadores sintáticos, é desejável que a gramática não seja ambígua.

Eliminando a ambiguidade

- Remova a ambiguidade de:

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

- Ajuda na verificação de prioridades/precedência:
  - Ex.: * sobre +

Eliminação da recursão a esquerda

- Alguns métodos de análise sintática não permitem recursão a esquerda;
  - Análise sintática descendente.
- A gramática possui recursão a esquerda imediata quando existe uma produção na forma: $A \rightarrow A\alpha|\beta$
- A recursão imediata pode ser eliminada:

$$A \rightarrow \beta A'$$
$$A' \rightarrow \alpha A'|\epsilon$$

- Essa regra é suficiente para muitas gramáticas.

Eliminação da recursão a esquerda

- Agrupe as produções

$$A \rightarrow A\alpha_1 | A\alpha_2 | \ldots | A\alpha_m | \beta_1 | \beta_2 | \ldots | \beta_n$$

- Substitua as produções por:

$$A \rightarrow \beta_1 A' | \beta_2 A' | \ldots | \beta_n A'$$
$$A' \rightarrow \alpha_1 A' | \alpha_2 A' | \ldots | \alpha_m A' | \epsilon$$

Resolva:

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid id$$

Fatoração a Esquerda

- Transformação na gramática para se tornar adequada para um reconhecedor sintático preditivo, ou descendente;
- Quando a escolha entre duas alternativas de produções não é clara, podemos reescrever para adiar a decisão até que se tenhamos lido uma entrada longa o suficiente para decidir.

Fatoração a Esquerda

Ache a cadeia α em comum mais longa:

$$A \rightarrow \alpha\beta_1 | \alpha\beta_2 | \ldots | \alpha\beta_n | \gamma$$

Substitua as produções por:

$$A \rightarrow \alpha A' | \gamma$$
$$A' \rightarrow \beta_1 | \beta_2 | \ldots | \beta_n$$

Aplique repetidamente essa transformação até que não haja duas alternativas para um não terminal com um prefixo comum.

Ex.:

$$S \rightarrow iEtS \mid iEtSeS \mid a$$
$$E \rightarrow b$$

Referências

- [Ref 1] AHO, Alfred V et al.
  (). Compiladores: princípios, técnicas e ferramentas. 2. ed. São Paulo: Pearson/Addison Wesley, 2007. 634 p. ISBN 9788588639249 (broch.).