INE5318
Construção de Compiladores

AULA 4: Análise Sintática

Ricardo Azambuja Silveira
INE-CTC-UFSC
E-Mail: silveira@inf.ufsc.br
URL: www.inf.ufsc.br/~silveira

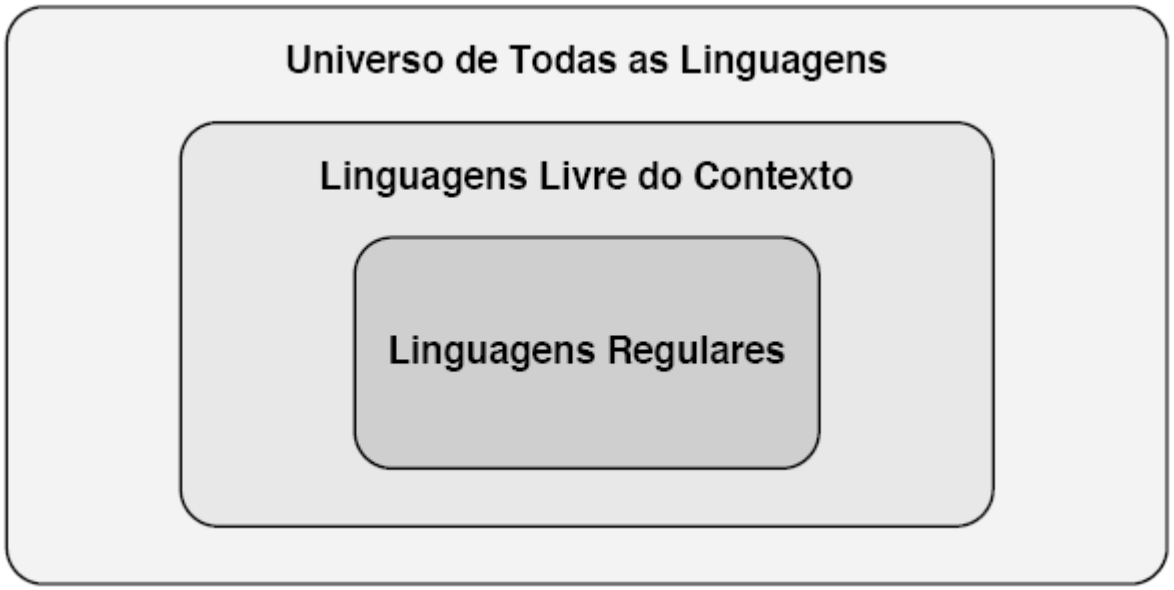
Definições preliminares

- Parser (Analisador Sintático)
  - algoritmo que recebendo como entrada uma sentença x, verifica se x pertence à linguagem produzindo uma árvore de derivação para x, ou emite mensagem de erro
  - Na prática recebe do analisador léxico a seqüência de tokens que constitui a sentença x
- Seja G = (Vn, Vt, P, S) uma G.L.C. com as produções de P numeradas de 1 a p e uma derivação $S \Rightarrow^{+} x$, o Parse de x em G, é a seqüência formada pelo número das produções utilizada

Definições preliminares

- Parse Ascendente (Botton-up)
  - seqüência invertida dos números das produções utilizadas na derivação mais a direita de x em G
    - $S \Rightarrow^{+}_{dir} x$
- Parse Descedente (Top-down)
  - seqüência dos números das produções utilizadas na derivação mais a esquerda de x em G
    - $S \Rightarrow^{+}_{esq} x$

Classes de linguagens



Linguagem livre do contexto

- É a classe mais geral de linguagens cuja produção é da forma
  - $A \rightarrow \alpha$
- Em uma derivação, a variável A deriva α sem depender ("livre") de qualquer análise dos símbolos que antecedem ou sucedem A (o "contexto") na palavra que está sendo derivada

Linguagem livre do Contexto

- Linguagem Livre do Contexto (LLC) ou Tipo 2
  - Linguagem gerada por uma gramática livre do contexto G onde:
  - GERA(G) = { w ∈ T* | S ⇒⁺ w }
- Definição formal da GLC:
  - G = (V, T, P, S)
  - Onde qualquer regra de produção é da forma
    - A → α
    - Onde:
      - A é variável de V (no lado esquerdo há uma variável)
      - α é palavra de (V ∪ T)*

Tipos de gramática

- Classificação ou hierarquia de CHOMSKY
  - Gramática Regular:
    - $P = \{A \rightarrow aX \mid A \in V, a \in T \wedge X \in \{V \cup \{\varepsilon\}\}\}$
    - No lado direito da produção há no máximo um não-terminal sempre acompanhado de uma variável terminal
  - Gramática Livre de Contexto:
    - $P = \{A \rightarrow \beta \mid A \in V \wedge \beta \in (V \cup T)^*\}$
    - O lado esquerdo da produção contém exatamente uma variável
  - Gramática Sensível ao Contexto:
    - $P = \{\alpha \rightarrow \beta \mid |\alpha| \leq |\beta|, \alpha \in (V \cup T)^*V(V \cup T)^* \wedge \beta \in (V \cup T)^+\}$

Tipos de gramática

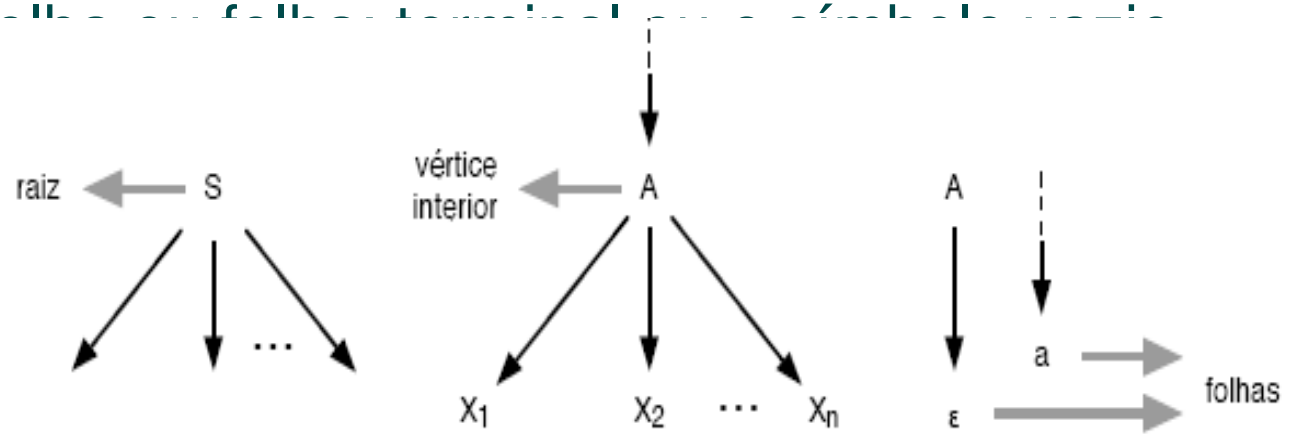
- O analisador léxico lê o programa fonte como uma linguagem regular a fim de identificar os tokens
- O analisador sintático lê a seqüência de tokens fornecida pelo analisador léxico como uma linguagem livre de contexto para identificar a sua estrutura sintática construindo a árvore de derivação

Árvore de Derivação

- Derivação de palavras na forma de árvore
  - partindo do símbolo inicial como a raiz
  - terminando em símbolos terminais como folhas
    - A raiz é o símbolo inicial
    - os vértices interiores são os não-terminais
    - as folhas são os terminais

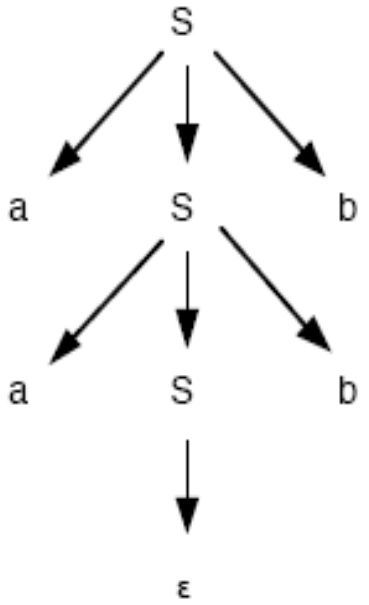
Definições

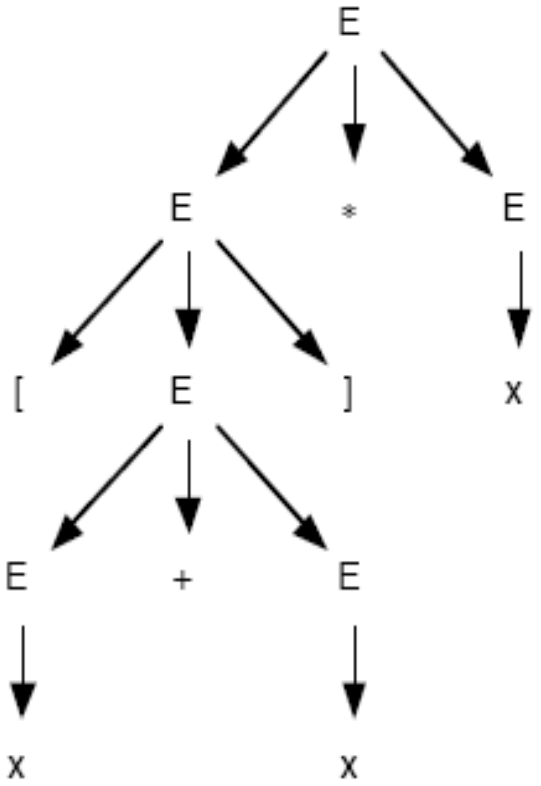
- Raiz: símbolo inicial da gramática
- Vértices interiores: variáveis
  - se A é um vértice interior e X1, X2,…,Xn são os "filhos" de A, então
    - A → X1X2…Xn é uma produção da gramática
    - X1, X2,…,Xn são ordenados da esquerda para a direita
- Vértice folha ou folha terminal ou símbolo vazio
  - se va



Exemplo

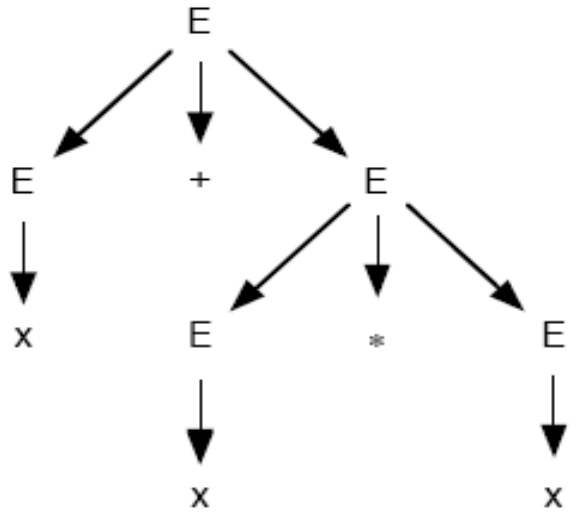
- Árvore de Derivação: aabb e [x+x]∗x





derivações distintas de uma mesma palavra

- Árvore de Derivação × Derivações: x+x∗x
  - E ⇒ E+E ⇒ x+E ⇒ x+E∗E ⇒ x+x∗E ⇒ x+x∗x mais a esquerda
  - E ⇒ E+E ⇒ E+E∗E ⇒ E+E∗x ⇒ E+x∗x ⇒ x+x∗x mais a direita
  - E ⇒ E+E ⇒ E+E∗E ⇒ x+E∗E ⇒ x+x∗E ⇒ x+x∗x Etc…
  - Derivação mais à Esquerda (Direita) é a seqüência de produções aplicada sempre à variável mais à avra

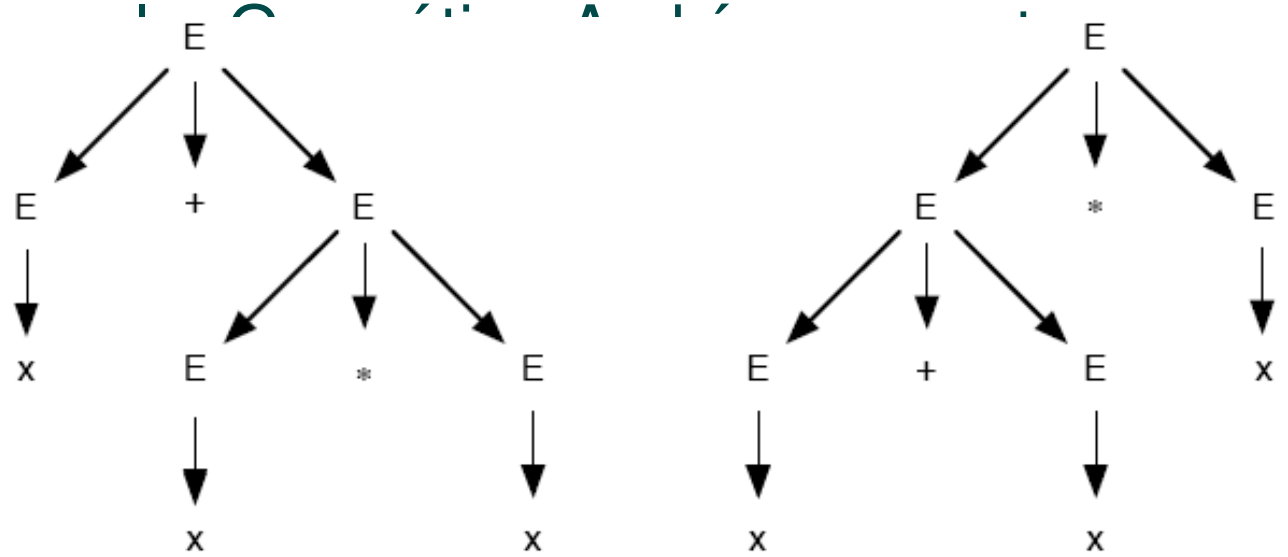


Tipos de gramáticas

- Gramática ambígua
- Gramática sem ciclos
- Gramática ε-livre
- Gramática fatorada à esquerda
- Gramática recursiva à esquerda
- Gramática simplificada

GLC Ambígua

- Uma Gramática livre do contexto é ambígua se existe pelo menos uma palavra que possui duas ou mais árvores de derivação
  - Ex



Exemplo

- Mais de uma derivação à esquerda (direita) para x+x∗x
  - Derivação mais à esquerda
    - E ⇒ E+E ⇒ x+E ⇒ x+E∗E ⇒ x+x∗E ⇒ x+x∗x
    - E ⇒ E∗E ⇒ E+E∗E ⇒ x+E∗E ⇒ x+x∗E ⇒ x+x∗x
  - Derivação mais à direita
    - E ⇒ E+E ⇒ E+E∗E ⇒ E+E∗x ⇒ E+x∗x ⇒ x+x∗x
    - E ⇒ E∗E ⇒ E∗x ⇒ E+E∗x ⇒ E+x∗x ⇒ x+x∗x

Linguagem Inerentemente Ambígua

- Não existe um procedimento geral para eliminara ambigüidade de uma gramática
- Uma linguagem é inerentemente ambígua se qualquer GLC que a define é ambígua
  - Exemplo:{ w | w = $a^n b^n c^m d^m$ ou w = $a^n b^m c^m d^n$, n ≥ 1, m ≥ 1 }

Gramática sem ciclos

- Gramática sem ciclos é uma GLC que não possui nenhuma derivação da forma:
  - $A \Rightarrow^{+} A$ para algum $A \in N$

Gramática ε - livre

- É uma gramática que não possui nenhuma produção do tipo
  - A →ε
- exceto, possívelmente a produçao
  - S → ε
  - onde S é o símbolo inicial

Exclusão das produções vazias

- Exclusão de produções vazias (da forma A → ε) pode determinar modificações diversas nas produções
- Algoritmo
  - Etapa 1: variáveis que constituem produções vazias
    - A → ε: variáveis que geram diretamente ε
    - B → A: sucessivamente, variáveis que indiretamente geram ε
  - Etapa 2: exclusão de produções vazias
    - considera apenas as produções não-vazias
    - cada produção cujo lado direito possui uma variável que gera ε, determina uma produção adicional, sem essa variável
  - Etapa 3: geração da palavra vazia, se necessário

Gramática fatorada à esquerda

- É uma GLC que não apresenta produções do tipo:
  - A $\alpha\beta_1 \mid \alpha\beta_2$
- o parser não sabe qual das regras deve aplicar para α

Gramática fatorada

Uma GLC está fatorada se ela é determinística, isto é, não possui produções cujo lado direito inicie com o mesmo conjunto de símbolos ou com símbolos que gerem seqüências que iniciem com o mesmo conjunto de símbolos. Por exemplo, uma gramática fatorada não poderia apresentar as seguintes regras:

$$A \rightarrow aB|aC\,,$$

pois ambas inicial com o terminal a.

Outro exemplo de gramática não-fatorada é:

$$S \rightarrow A|B \quad A \rightarrow ac \quad B \rightarrow ab\,.$$

Gramática fatorada

- Para fatorar:
  1. as produções que apresentam não-determinismo direto, da forma $A \rightarrow \alpha\beta|\alpha\gamma$ serão substituidas por

$$\begin{aligned} A &\rightarrow \alpha A' \\ A' &\rightarrow \beta|\gamma \end{aligned}$$

     sendo A' um novo não-terminal;
  2. o não-determinismo indireto é retirado fazendo, nas regras de produção, as derivações necessárias para torna-lo um não-determinismo direto, resolvido com o passo anterior.

Gramática fatorada

Exemplos:

1. $P : S \rightarrow aA|aB, A \rightarrow aA|a, B \rightarrow b$. Solução:
   $P' : S \rightarrow aS', S' \rightarrow A|B, A \rightarrow aA', A' \rightarrow A|\varepsilon, B \rightarrow b$;
2. $P : S \rightarrow Ab|ab|baA, A \rightarrow aab|b$. Solução:
   (não-determinismo indireto)
   $P' : S \rightarrow aabb|bb|ab|baA, A \rightarrow aab|b$ e
   $P'' : S \rightarrow aS'|bS'', S' \rightarrow abb|b, S'' \rightarrow b|aA, A \rightarrow aab|b$

Gramática recursiva à esquerda

- É uma GLC que permite a derivação
  - $A \Rightarrow^{+} A\alpha$ para algum $A \in N$
- ou seja, um não terminal deriva ele mesmo de forma direta ou indireta como o símbolo mais a esquerda
-

Recursão à Esquerda

- Algoritmo para eliminar recursão à esquerda:
  - Etapa 1: simplificação da gramática
  - Etapa 2: renomeação das variáveis em uma ordem crescente qualquer
  - Etapa 3: produções na forma $Ar \rightarrow As\alpha$, na qual $r \leq s$
  - Etapa 4: exclusão das recursões da forma $Ar \rightarrow Ar\alpha$

Eliminação da recursão

- Eliminação da Recursão à Esquerda

  Um não-terminal A é *recursivo* se $A \Rightarrow^{+} \alpha A \beta$, $\alpha, \beta \in V^{*}$. Se $\alpha = \varepsilon$, então A é recursivo à esquerda. Se $\beta = \varepsilon$, é recursivo à direita. A recursividade pode ser direta ou indireta.

  Uma gramática é *recursiva à esquerda* se possui pelo menos um não-terminal recursivo à esquerda. Se possui pelo menos um não-terminal recursivo à direita, ela é chamada *recursiva à direita*.

Eliminação da recursão

- A importância da recursividade à esquerda é que alguns tipos de compiladores podem executar o processo de reconhecimento como chamadas de rotinas (procedimentos ou funções). Assim, uma gramática recursiva à esquerda, tal como por exemplo $A \rightarrow Aa|a$, acaba gerando um laço infinito $A \Rightarrow Aa \Rightarrow Aaa \Rightarrow Aaaa \Rightarrow \cdots$ e o processo de reconhecimento não finaliza nunca.

Eliminação da recursão

1. recursões diretas: substituir cada regra

   $A \rightarrow A\alpha_1 | A\alpha_2 | \cdots | A\alpha_n | \beta_1 | \beta_2 | \cdots | \beta_m$, onde nenhum

   β_i começa por A, por:

$$A \rightarrow \beta_1 A' | \beta_2 A' | \cdots | \beta_m A'$$

$$A' \rightarrow \alpha_1 A' | \alpha_2 A' | \cdots | \alpha_n A' | \varepsilon$$

Eliminação da recursão

2. recursões indiretas:

   (a) ordene os não-terminais de G em uma ordem qualquer $(A_1, A_2, \ldots, A_n)$;

   (b)

   para i de 1 até n faça
   para j de 1 até $(i-1)$ faça
   troque $A_i \rightarrow A_j\gamma$ por $A_i \rightarrow \delta_1\gamma|\delta_2\gamma|\cdots|\delta_k\gamma$, onde $\delta_1, \delta_2, \ldots, \delta_k$ são os lados direitos das A_j-produções (ou seja, $A_j \rightarrow \delta_1|\cdots|\delta_k$);
   fim_para
   elimine as recursões diretas das A_i-produções;
   fim_para.

Eliminação da recursão

- Exemplo: $P : S \rightarrow Aa, A \rightarrow Sb|cA|a$. Solução: $(A_1 = S, A_2 = A)$, $n = 2$, e

  1. $i = 1$ (não faz j pois o laço é de 1 até 0);

     Eliminando as recursões diretas das S-produções: não faz nada (pois não tem);

  2. $i = 2$ e $j = 1$:

     Trocar produções tipo $A \rightarrow S\gamma$:
     $P' : S \rightarrow Aa, A \rightarrow Aab|cA|a$;

     Eliminar as recursões diretas das A-produções:
     $P'' : S \rightarrow Aa, A \rightarrow cAA'|aA', A' \rightarrow abA'|\varepsilon$.

Classe de analisadores

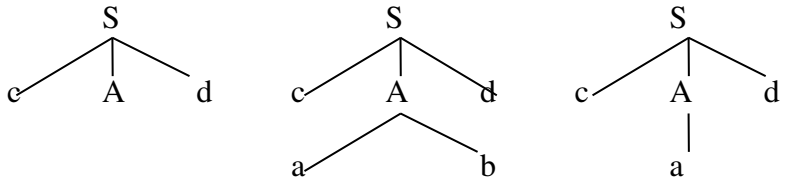
- Descendentes (top-down)
  - Com back-track
  - Sem back-track
    - Recursivo
    - Preditivo (LL)
- Ascendentes (bottom-up)
  - Shift-reduce
  - Shift-reduce com análise de precedência
  - LR
    - LR(0)
    - SLR(1)
    - LALR(1)

Analisadores descendentes

- Constrói a derivação mais à esquerda da sentença de entrada a partir do símbolo inicial da gramática. podem ser implementados:
- com back-track(analisadores não-deterministicos)
  - Força bruta
  - Maior conjunto de gramáticas (ambíguas)
  - Maior tempo para análise
  - Dificuldade para recuperação de erros
  - Dificuldade para análise semântica e geração de código
- sem back-track (analisadores determinísticos)
  - Classe limitada de gramáticas

Analisadores c/ back track

- O backtracking é necessário no seguinte tipo de situação: considere a sentença *cad* e a gramática:
  - S → cAd
  - A → ab | a



inicialmente, é construída uma árvore de sintaxe tendo S como raiz. S é expandido para cAd. O primeiro caracter da entrada (c) é reconhecido, sendo necessário então expandir o não-terminal A. Com a expansão de A, é reconhecido o segundo caracter a. O ponteiro da entrada é movido para o terceiro caracter (d). No reconhecimento deste caracter, há uma falha, já que tenta-se marcar d com b (da derivação de A). Neste momento, é necessário que se retorne para A, a fim de tentar realizar o reconhecimento através da segunda alternativa de derivação de A

Analisadores c/ back track

- O processo é ineficiente, pois leva à repetição da leitura de partes da sentença de entrada, não sendo muito utilizado no reconhecimento de linguagens de programação.
- Quando ocorre um erro, não é possível determinar o ponto exato onde este ocorreu, devido à tentativa de produções alternativas.
- Uma gramática recursiva à esquerda pode gerar um ciclo infinito de expansão de não-terminais.

Analisadores s/ back-track

- Limitações da gramática
  - Não possuir Recursão à Esquerda;
  - Estar Fatorada, isto é, se $A \rightarrow \alpha_1 \mid \alpha_2 \mid ... \mid \alpha_n$ são as A-produções de uma determinada G.L.C. então
    - $First(\alpha_1) \cap ... \cap First(\alpha_n) = \varnothing$
  - Para todo $A \in Vn \mid A \Rightarrow^* \varepsilon$, $First(A) \cap Follow(A) = \varnothing$
    - é necessário uma gramática cujas produções derivadas a partir de um mesmo não-terminal tenham seus lados direitos iniciados por diferentes seqüências de caracteres, permitindo que o analisador "saiba" qual das alternativas a seguir, antes mesmo de começar a derivação

Analisador descendente recursivo

- consiste basicamente na construção de um conjunto de procedimentos (normalmente recursivos), um para cada símbolo não terminal da gramática em questão
- Desvantagens
  - não é geral, ou seja, os procedimentos são específicos para cada gramática
  - tempo de análise é maior
  - necessidade de uma linguagem que permita recursividade para sua implementação
  - Dificuldade de validação
- Vantagens
  - simplicidade de implementação
  - facilidade para inserir as diferentes funções do processo de

Exemplo

- Parser Descendente Recursivo da Gramática G:
  - E → TE'
  - E' → + TE' | ε
  - T → (E) | id

```
(* Programa Principal *)
begin
        scanner (simb);
        E (simb);
end;
procedure E (simb);
begin

        T (simb);
        Elinha (simb);
end;

procedure  Elinha (simb);
begin
        if simb = "+"
        then begin
              scanner (simb);
              T (simb);
              Elinha (simb);
           end;
end;


procedure T (simb);
begin

   ...

end;
```

Parser preditivo (LL)

- Implementa um parser descendente recursivo mais eficiente.
- Consiste de:
  - Entrada contendo a sentença a ser analisada.
  - Pilha usada para simular a recursividade
    - prevê a parte da entrada que está para ser analisada
    - inicializada com $ e o símbolo inicial da gramática em questão).
  - Tabela de Parsing contendo as ações a serem efetuadas.
    - É uma matriz M(A, a), onde $A \in Vn \wedge a \in Vt$.
  - Algoritmo de análise sintática.

Estrutura de um parser preditivo

- O termo Preditivo deve-se ao fato de que a pilha sempre contém a descrição do restante da sentença (se ela estiver correta); isto é, prevê a parte da sentença que deve estar na entrada para que a sentença esteja correta

Parser preditivo (LL)

- tem como função determinar a partir de X (topo da pilha) e de a (o próximo símbolo) a ação a ser executadar:
- Se X = a = $ → o analisador anuncia o final da análise.
- Se X = a ≠ $ → o analisador retira X do topo da pilha e a da entrada. (reconhecimento sintático de a)
- Se X ∈ Vt e X ≠ a → situação de erro ('X' era o símbolo esperado). devem ser ativados os procedimentos de recuperação de erro ou a análise encerrada
- Se X ∈ Vn → o analisador consulta a tabela de parsing M(X,a), a qual poderá conter o número de uma produção ou um indicativo de erro:
  - Se M(X,a) contém o número de uma produção, e esta produção tipo X → UVW, então X que está no topo da pilha deve ser substituído por WVU (com U no topo);
  - - Se M(X, a) contém um indicativo de erro, então o diagnosticar o erro

Algoritmo

```
repita
(* X – topo da pilha *)
(* a – próximo símbolo da entrada *)

se x é terminal ou $
então se X = a
        então retira X do topo da pilha
                retira a da entrada
        senão erro( )
senão (* X é não terminal *)
        se M(X, a) = X → Y1 Y2 ... Yk
        então retira X da pilha
                coloca Yk Yk-1 ... Y2 Y1 na pilha
                (* deixando Y1 sempre no topo *)
        senão erro( )
até X = $ (* pilha vazia, análise concluída *)
```

Tabela de parsing

- Em cada entrada da tabela M existe uma única produção viabilizando a análise determinística da sentença de entrada.
- Para isso é necessário que a gramática:
  - Não possuir recursão à esquerda;
  - Estar fatorada (é determinística)
  - Para todo $A \in Vn \mid A \Rightarrow^{*} \varepsilon$, $First(A) \cap Follow(A) = \varnothing$

Tabela de parsing

- As G.L.C. que satisfazem estas condições são denominadas G.L.C. LL(K)
  - podem ser analisadas deterministicamente da esquerda para a direita (Left-to-right)
  - o analisador construirá uma derivação mais à esquerda (Leftmost derivation)
  - sendo necessário a cada passo o conhecimento de K símbolos de lookahead (símbolos de entrada que podem ser vistos para que uma ação seja determinada).
- Somente G.L.C. LL(K) podem ser analisadas pelos analisadores preditivos (
- os analisadores preditivos são também denominados analisadores LL(K)
  - na prática usa-se K = 1, obtendo-se desta forma Analisadores

Tabela de parsing LL

- Idéia Geral
  - se $A \rightarrow \alpha \in P \wedge a \in$ First(α), então, se A está no topo da pilha e a é o próximo símbolo da entrada, devemos expandir (derivar) A, usando a produção $A \rightarrow \alpha$.
  - Mas, e se $\alpha = \varepsilon$ ou $\alpha \Rightarrow^* \varepsilon$!? (note que ε nunca aparecerá na entrada) neste caso, se $a \in$ Follow(A) expandimos (derivamos) A através da produção $A \rightarrow \alpha$.

Tabela de parsing LL

- Algoritmo para Construção da Tabela de Parsing
  - Para cada produção $A \rightarrow \alpha \in P$
    - Para todo $a \in$ First(α), exceto ε,
      - coloque o número da produção $A \rightarrow \alpha$ em M(A, a).
    - Se $\varepsilon \in$ First(α), coloque o número da produção $A \rightarrow \alpha$ em M(A, b), para todo $b \in$ Follow(A).
  - As posições de M que ficarem indefinidas, representarão as situações de erro.

Definições preliminares

- Conjunto first
  - Conjunto dos terminais que iniciam as palavras derivadas por α
  - Seja α uma seqüência qualquer gerada por G, first(α) é o conjunto de símbolos terminais que iniciam α ou seqüências derivadas (direta ou indiretamente) de α.
    - Se $\alpha = \varepsilon$ ou $\alpha \Rightarrow^* \varepsilon$, então $\varepsilon \in$ first(α)
- Conjunto Follow
  - para todo A $\in$ Vn, Folow de A é o conjunto de símbolos terminais que podem aparecer imediatamente após A em alguma forma sentencial da gramática

Analisadores Ascendentes

- Baseia-se em um algoritmo primitivo denominado Algoritmo Geral Shift-Reduce (Avança-Reduz)
- Constituído por
  - Uma Pilha Sintática (inicialmente vazia);
  - Um Buffer de Entrada contendo a sentença a ser analisada);
  - Uma G.L.C. com as produções numeradas de 1 a p);
  - Um procedimento de análise sintática

procedimento de análise sintática

- consiste em transferir (“Shiftar”) os símbolos da entrada (um a um) para a pilha até que o conteúdo da pilha (ou parte dele) coincida com o lado direito de uma produção.
- Quando isto ocorrer, o lado direito da produção deve ser substituído pelo (reduzido ao) lado esquerdo da produção em questão.
- Este processo deve ser repetido ate que toda a sentença de entrada tenha sido analisada.
- As reduções são prioritárias em relação ao shifts
- Se no fim a entrada estiver vazia e a pilha contiver apenas o símbolo inicial de G, a sentença é reconhecida

Exemplo

- Dado a gramática
  - S → aABe
  - A → Abc | b
  - B → d
- A entrada abbcde pode ser reduzida para S de acordo com os seguintes passos:
  - abbcde
  - aAbcde
  - aAde
  - aABe
  - S
- De forma reversa, a seguinte derivação é equivalente à redução realizada:
  - S → aABe → aAde → aAbcde → abbcde

Exemplo

- Com a gramática abaixo e a entrada id1+id2* id3, tem-se a seqüência de reduções apresentada na Tabela
  - $E \rightarrow E + E$
  - $E \rightarrow E * E$
  - $E \rightarrow (E)$
  - $E \rightarrow id$

Forma Sentencial à Direita	*Handle*	Produção para Redução
$id_1+id_2* id_3$	id_1	$E \rightarrow id$
$E+id_2* id_3$	id_2	$E \rightarrow id$
$E+E* id_3$,	id_3	$E \rightarrow id$
E+E*E	E*E	$E \rightarrow E*E$
E+E	E+E	$E \rightarrow E+E$
E		

Características

- pode ser aplicada a qualquer GLC
- Requer muito tempo para analise;
- Só detecta erro sintático após consumir toda a sentença a ser analisada e não identifica o ponto onde ocorreu o erro.
- Pode rejeitar sentenças corretas (não-determinismo), necessitando por isso de backtracking

Técnicas Shift-Reduce sem Back-Tracking

- superam as deficiências da técnica Shift-Reduce com Back-Tracking
  - Tempo de análise diretamente proporcional ao tamanho da sentença;
  - Detecção de erros sintáticos no momento da ocorrência;
  - São técnicas Determinísticas, isto é, em qualquer situação, haverá sempre uma única ação a ser efetuada.
- Custo associado:

Principais técnicas

- Analisadores de Precedência (Simples, estendida e de operadores)
  - algoritmo Shift-Reduce acrescido de relações de precedência entre os símbolos da gramática que definem, de forma determinística a ação a ser efetuada
- Analisadores LR
  - família de analisadores shift e reduce em que as operações shift e reduce são realizados sempre deterministicamente, com base no estado corrente da análise e nas propriedades estruturais da gramática em questão.

Analisadores LR

- analisam a sentença de entrada da esquerda para a direita (Left-to-right) e construírem uma derivação mais à direita (Rightmost derivation) na ordem inversa.
- formada por uma série de técnicas onde as principais são: LR(0), SLR(1), LALR(1) e LR(1), em ordem crescente no sentido de força (abrangência de G.L.C.) e complexidade de implementação
- consiste de duas partes:
  - Um algoritmo de análise sintática (padrão para todas as técnicas da família e independente da gramática)
  - uma Tabela de Parsing (construída de forma especifica para cada técnica e para cada gramática).

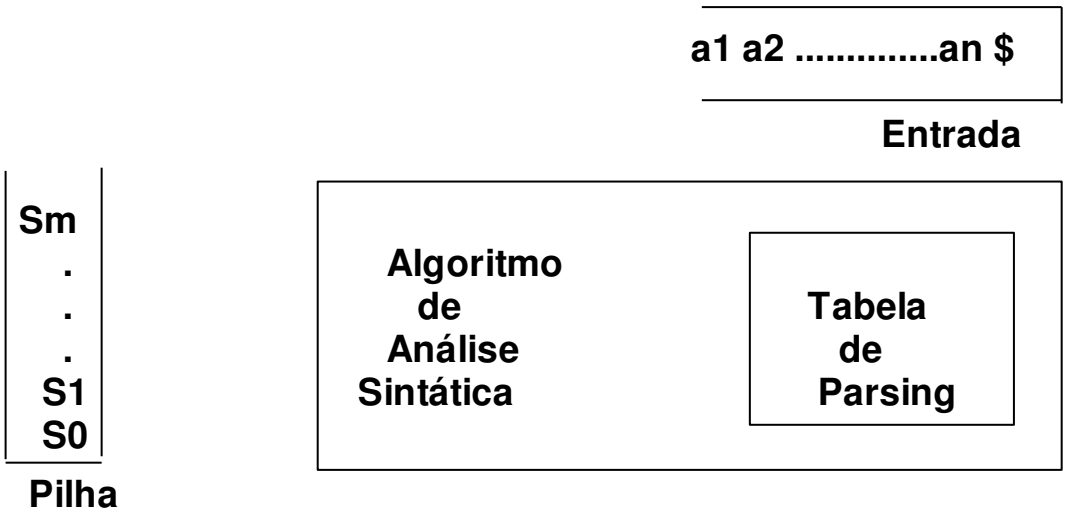
Analizadores LR

- Vantagens
  - Analisam praticamente todas as construções sintáticas de linguagem de programação que podem ser representadas de forma não ambígua por G.L.C..
  - São mais gerais que os outros analisadores ascendentes e que a maioria dos descendentes sem back-track.
  - Possuem a propriedade de detecção imediata de erros sintáticos.
  - O tempo de análise é proporcional ao tamanho da sentença a ser analisada.
- Desvantagens:
  - complexidade de construção da tabela de parsing
  - espaço requerido para seu armazenamento

Estrutura

- Algoritmo de Análise Sintática: padrão para todas as técnicas da família.
- Tabela de Análise Sintática (ou tabela de parsing): específica para cada técnica e para cada gramática.
- a)Pilha de Estados (ou pilha sintática): conterá um histórico da análise efetuada; é inicializada com o estado inicial da análise sintática.
- b)Entrada: conterá a sentença a ser analisada, seguida por $ (a marca de final de sentença).
- c)Uma G.L.C. : com as produções numeradas de 1 a p.

Estrutura


a1 a2 .....an $
Entrada
Sm
.
.
.
S1
S0
Pilha
Algoritmo
de
Análise
Sintática
Tabela
de
Parsing

Algoritmo

- Determinar S_m (o estado do topo da pilha) e a_i (o próximo símbolo da entrada) e consultar a tabela de parsing para decidir a próxima ação a ser efetuada:
  - Halt: fim de análise;
  - Erro: indica a ocorrência de um erro sintático.
  - Shift S: significa o reconhecimento sintático do símbolo da entrada (o símbolo deve ser retirado e o estado S, indicado na tabela de Parsing, deverá ser empilhado)
  - Reduce R: significa que uma redução pela produção número R deve ser efetuada.
    - retira da pilha sintática tantos estados quantos forem os símbolos do lado direito da produção R, e o símbolo do lado esquerdo dessa produção deverá ser tratado como um símbolo de entrada na próxima ação do analisador.

Tabela de parsing LR

- dividida em duas partes:
- a) Tabela Action:
  - contém as transições (Shift, Reduce, Erro e Halt) com os símbolos terminais.
- b) Tabela Goto:
  - contém as transições (Goto e Erro) com os símbolos não-terminais (Um Goto é um Shift especial sobre um símbolo não-terminal).
-

Gramáticas LR

- Uma G.L.C. é LR se cada sentença que ela gera pode ser analisada deterministicamente da esquerda para a direita por um analisador LR.
- Pertencem a classe das gramáticas LR, todas as G.L.C. para as quais é possível construir a tabela de parsing LR sem que hajam conflitos (isto é, em cada posição da tabela haverá no máximo uma ação registrada).
- Gramáticas LR(K): São gramáticas LR, considerando-se K símbolos de Lookahead.
- G.L.C. analisáveis por analisadores SLR(1), LALR(1) e LR(1) são também denominadas, respectivamente, gramáticas SLR(1), LALR(1) e LR(1).

Propriedade das Tabelas LR

- Condição De Erro: Nunca será encontrada para sentenças sintaticamente corretas.
- Cada Shift deverá especificar um único estado, o qual dependerá do símbolo da entrada.
- Cada Reduce só é realizado quando os estados do topo da pilha (os símbolos que eles representam) forem exatamente os símbolos do lado direito de alguma produção de G, isto é, um handle da gramática.
- Halt só será encontrado quando a análise estiver completa.