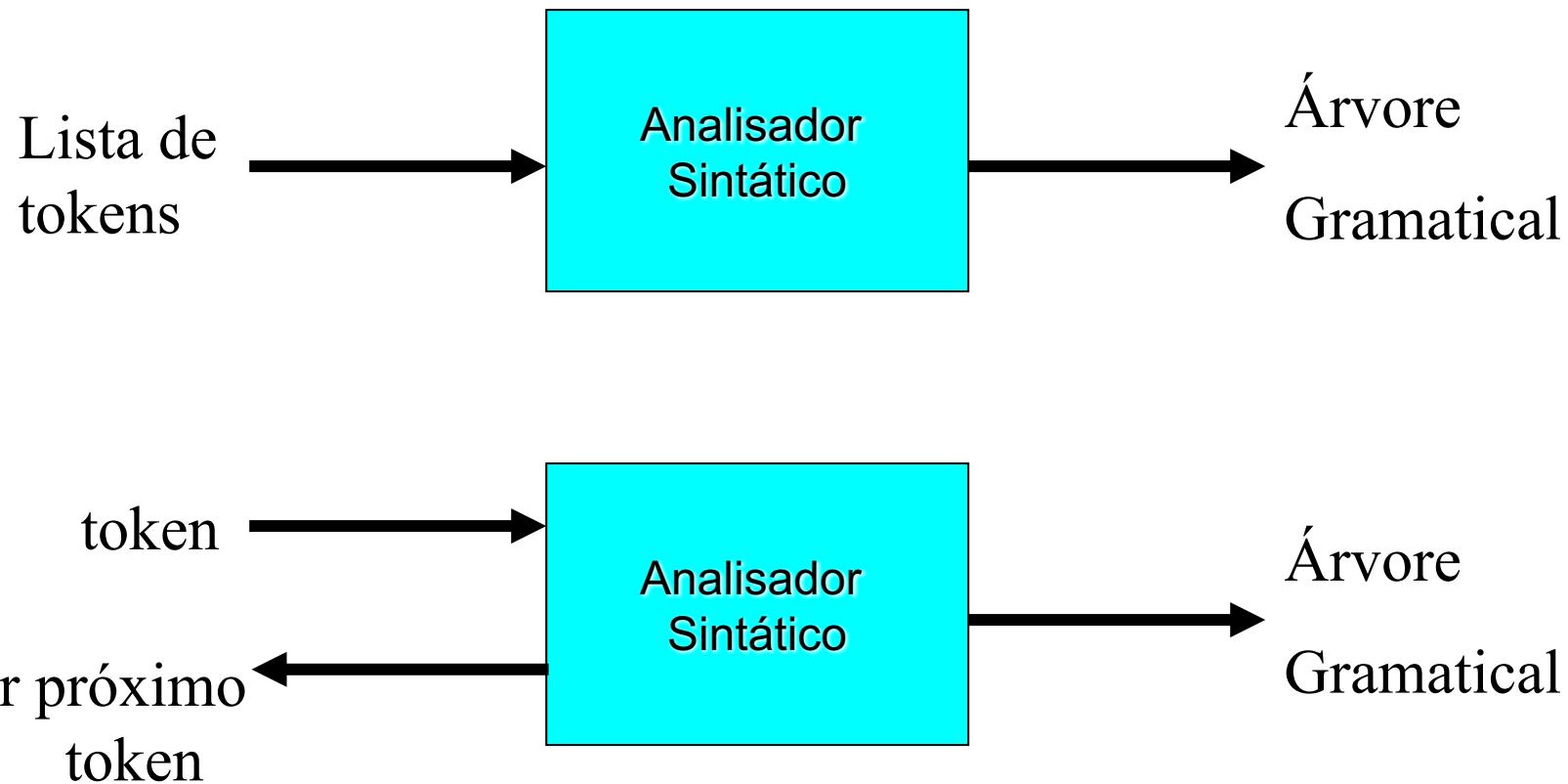


Compiladores

Análise Sintática – Parte 1
Prof. Tales Viegas

<https://fb.com/ProfessorTalesViegas>

Análise Sintática



Funções do Analisador Sintático

- ▶ Comprovar que a sequência de tokens cumpre as regras gramaticais
- ▶ Gerar a árvore gramatical

Vantagens de Utilizar Gramáticas

- ▶ Especificações sintáticas precisas de linguagens
- ▶ Podemos usar um gerador automático de *parser*
- ▶ O processo de construção pode levar a identificar ambiguidades
- ▶ Facilidade de ampliar/modificar a linguagem

Papel dos Analisadores Sintáticos

- ▶ Identificar erros de sintaxe
 A^*/B
- ▶ Tornar clara a estrutura hierárquica da evolução da sentença
 A/B^*C  $(A/B)^*C$ em Fortran
 $A/(B^*C)$ em APL
- ▶ Recuperação de erros de sintaxe
- ▶ Não retardar, de forma significativa, o processamento de programas corretos

Tipos de Analisadores Sintáticos

- ▶ Métodos de Cocke–Younger–Kasami e Early
 - Universais: servem para qualquer gramática (bem eficientes)
- ▶ Métodos descendentes (Top–Down)
 - Constroem a árvore sintática de cima para baixo
 - Analisadores Descendentes Recursivo
 - Analisadores LL(k)

Tipos de Analisadores Sintáticos

- ▶ Métodos Ascendentes (Bottom-Up)
 - Constroem a árvore sintática de baixo para cima
 - Analisadores SR
 - Analisadores LR
 - Analisadores LALR

Tipos de Analisadores Sintáticos

- ▶ Tanto para a análise ascendente quanto para a descendente
 - A entrada é examinada da esquerda para direita, um símbolo por vez
 - Trabalham com subclasses de gramáticas
- ▶ Em geral, as gramáticas são LL e LR
 - Na prática, utilizam LL(1) e LR(1)

Tipos de Analisadores Sintáticos

- ▶ Muitos compiladores são dirigidos pela sintaxe (*parsen driver*)
 - Onde o analisador sintático chama o analisador léxico
- ▶ Há ferramentas para a geração automática de analisadores
 - Como o Flex e o Bison
 - Como o JFlex e o CUP

Recuperação de Erros

► Desespero

- Identificado um erro, o analisador sintático descarta símbolos de entrada até que seja encontrado um token pertencente ao subconjunto de token de sincronização
- Tokens de sincronização: delimitadores, etc.

Recuperação de Erros

▶ Recuperação de Frases

- Ao descobrir um erro, o analisador sintático pode realizar uma correção local na entrada restante (substituir por alguma cadeia que permita a análise prosseguir)
- Exemplo: substituir uma vírgula inadequada por um ponto e vírgula, remover um “:” excedente

Recuperação de Erros

▶ Produções de Erro

- Aumenta-se a gramática, de forma a acomodar os erros mais comuns.
- Quando uma produção de erro é identificada pelo analisador, diagnósticos apropriados são apresentados

Recuperação de Erros

▶ Correção Global

- Usa algoritmos de escolha da sequência mínima de mudanças necessárias para se obter a correção global, com custo adequado
- Exemplo: dada uma cadeia x , o parser procura árvores gramaticais que permitam transformar o x em y (cadeia correta) com um mínimo de modificações.

Análise Sintática Ascendente

- ▶ O algoritmo inicia a leitura do programa fonte, da esquerda para a direita, varre as diferentes hipóteses e obtém a árvore de derivação.
- ▶ A cadeia é reconhecida quando atinge-se o símbolo inicial da gramática

Análise Sintática Ascendente

► Algoritmo

- Adotar a cadeia dada como α
- Procura-se decompor α de forma que

- $\alpha = \beta X_1 X_2 \dots X_n \gamma$

existindo uma produção da forma:

- $X = X_1 X_2 \dots X_n$

chegamos a uma produção da forma:

- $\alpha = \beta X \gamma$

O significado é associar ao não-terminal X , e as subárvore(s) são as ocorrências de $X_1 X_2 \dots X_n$ que foram substituídas.

Se X_i é um terminal, a árvore associada será uma folha de rótulo X_i

- Repete-se o passo anterior até que α seja o símbolo inicial da gramática

Exemplo

- ▶ Seja a gramática de expressões definida por:
 - $E \rightarrow T$
 - $E \rightarrow E + T$
 - $T \rightarrow F$
 - $T \rightarrow T^*F$
 - $F \rightarrow a$
 - $F \rightarrow b$
 - $F \rightarrow (E)$
- ▶ Supondo a cadeia $a+b^*a$, teríamos que aplicar reduções até obter a cadeia de entrada

Observações

- ▶ Procurando as reduções mais à esquerda, obtem-se derivações mais a direita que formam a cadeia
- ▶ Nem sempre as reduções são triviais e levam a uma sequência válida
- ▶ Considerando o item anterior, o algoritmo **deveria** varrer todas as hipóteses possíveis, retrocedendo se necessário
- ▶ O algoritmo ideal sabe qual redução escolher – aplicável a um número menor de gramáticas

Análise Sintática Ascendente

Problemas

- ▶ Identificação da parte α a ser reduzida
- ▶ Identificação da produção a ser associada adequadamente à uma redução. Assim, em uma gramática onde existam produções do tipo:
 - $A \rightarrow \beta$
 - $B \rightarrow \beta$
 - programa tem de saber qual produção deve ser aplicada para redução
 - Há algoritmos que usam o mecanismo de redução diferente

Análise Sintática Descendente

- ▶ Parte-se do axioma (símbolo inicial) da gramática para chegar à sentença verificada. Se esta sentença estiver correta do ponto de vista gramatical, ela é reconhecida
- ▶ Equivale a construir a árvore de derivação para a sentença a partir da raiz, e obter nas folhas os símbolos da sentença
- ▶ Quando as folhas representam os símbolos terminais relativos à cadeia de entrada, a mesma foi reconhecida

Análise Sintática Descendente

- ▶ No curso:
 - Análise Descendente com Retrocesso
 - Análise Descendente Recursiva
 - Análise LL(k)

Análise Descendente com Retrocesso

- ▶ Um dos primeiros métodos para Análise Sintática
- ▶ Apresenta grandes ineficiências quanto a utilização de memória e tempo

Análise Descendente com Retrocesso

- ▶ O processo é tipicamente exploratório:
 - Tenta-se a derivação mais a esquerda para obter uma dada cadeia de entrada, através da exploração
 - No caso de haver várias regras com o mesmo lado esquerdo e diferentes lados direitos, ele escolhe uma e continua a análise
 - Caso tenha escolhido mal, ele seleciona outra possibilidade e continua o processo
- ▶ O processo termina quando não existem mais alternativas ou quando a cadeia foi reconhecida

Exemplo

- ▶ Suponhamos a gramática dada por $G=(N, T, P, S)$, onde $N=\{S,A,B\}$, $T=\{a,b,c,d\}$ e P é dado por:
 - $S \rightarrow A$
 - $A \rightarrow a$
 - $A \rightarrow aB$
 - $B \rightarrow bB$
 - $B \rightarrow cB$
 - $B \rightarrow d$
- ▶ Quais seriam as tentativas executadas pelo *parser* para reconhecer a cadeia **abcd**?

Exemplo

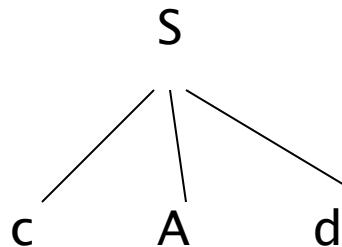
- ▶ A partir do exemplo, observa-se que o problema passa a ser: como fazer para evitar retrocessos na análise.
- ▶ Ou seja, sabendo qual é o próximo símbolo da cadeia e qual o não-terminal a ser derivado, determinar qual regra deve ser aplicada.

Análise Descendente com Retrocesso

- ▶ Suponhamos a gramática dada por $G=(N,T,P,S)$, onde $N=\{S,A\}$, $T=\{a,b,c,d\}$ e P é dado por:
 - $S \rightarrow cAd$
 - $A \rightarrow ab$
 - $A \rightarrow a$

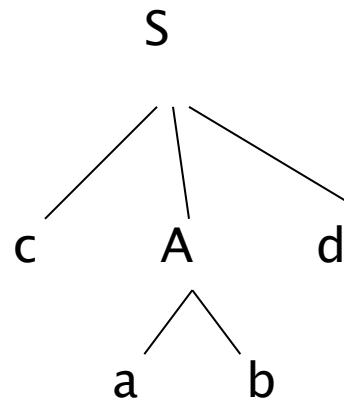
Análise Descendente com Retrocesso

- ▶ O processo de construção top-down para a cadeia **cad** seria:
 - Fazemos S como nossa árvore inicial, c é o primeiro símbolo de α
 - Usando a primeira produção possível para S obtém-se a árvore:



Análise Descendente com Retrocesso

- Como **c** corresponde ao primeiro símbolo de α , então uma posição é avançada, ficando o ponteiro apontando para **a**, e o não-terminal **A**. A tentativa abaixo pode ser efetuada:



Análise Descendente com Retrocesso

- ▶ Agora, como na leitura do próximo símbolo d não há reconhecimento, deve-se voltar ao símbolo A e tentar nova regra de produção que permita o reconhecimento, e assim sucessivamente

Análise Descendente com Retrocesso – Observações

- ▶ Possibilidade de *loop* infinito para produções com recursão à esquerda
- ▶ *Backtracking*: uma sequência de expressões errôneas, capaz de gerar uma má cadeia, leva a nova tentativa e à perda do efeito semântico da tentativa errada
- ▶ A ordem das substituições podem afetar a aceitação da cadeia. Por exemplo, no caso anterior, se tivéssemos tentado expandir A para **a** e reconhecido A, o *parser* falharia na tentativa de reconhecimento