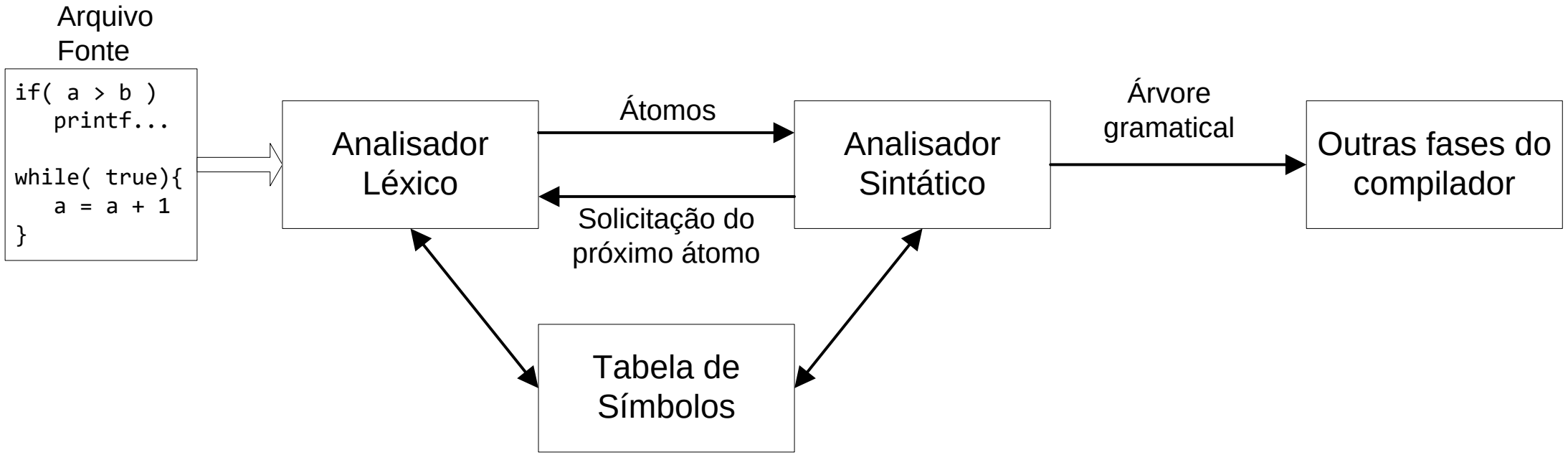
Análise Sintática

Fabio Lubacheski
fabio.lubacheski@mackenzie.br

Introdução a Análise Sintática

- A análise sintática ou gramatical constitui a segunda fase do compilador, sua função é verificar se o programa fonte está gramaticalmente correto, ou seja, se uma sequência de tokens (átomos) pode ser gerada por uma gramática.
- Em nosso modelo de compilador, o analisador sintático obtém sequências de tokens (átomos) proveniente do analisador léxico, e verifica se a mesma pode ser gerada (construída), seguindo a gramática da linguagem-fonte.



Introdução a Análise Sintática

- Na análise sintática é construída **árvore de análise sintática**, também denominada de **árvore gramatical** ou **árvore de derivação**.

- Caso o analisador sintático identifique quaisquer erros de sintaxe, este deve se recuperar dos erros que ocorram, a fim de continuar processando o resto de sua entrada.

- Para especificar a sintaxe de uma linguagem de programação para a análise sintática usamos as **gramáticas livre contexto** (GLC), pois as **definições regulares** (ou gramáticas regulares) não permitem especificar construções aninhados, por exemplo **(())**,ou seja, parênteses balanceados.

Analisador sintático

- Um **analisador sintático** (**parsing**) para uma gramática é um **programa** que aceita como entrada uma sentença (=palavra), caso a palavra pertença a linguagem gerada pela gramática o analisador sintático constrói uma **árvore derivação implicitamente na memória do computador**, caso contrário, ou seja, caso a sentença não pertença à linguagem descrita pela gramática, uma indicação de erro deve ser gerada.
- Existem duas técnicas básicas para a construção de analisadores sintáticos: **ascendente (bottom-up)** ou a **descendente (top-down)**.
  - **bottom-up:** Começam pelas folhas e trabalham a árvore até a raiz. (análise ascendente)
  - **top-down:** Constroem a árvore de derivação do topo da raiz para as folhas. (análise descendente)

Analisador sintático descendente (top-down).

- Na disciplina iremos começar implementando os analisadores **top-down**, que são mais simples de se construir, mas manipulam uma classe mais **restrita de gramáticas**.

- Uma restrição importante para a implementação de analisadores sintáticos descendente (**top-down**) é que a gramática **não seja ambígua**.

- Uma gramática livre de contexto é dita **ambígua** se existe pelo menos **uma palavra,** pertencente a linguagem gerada pela gramática, que possui **duas ou mais árvores de derivações distintas**.

Analisador Sintático descendente (top-down)

- A construção da árvore deve ser feita de maneira sistemática, usando sempre que possível a sentença da entrada para auxiliar nas decisões do algoritmo. Para implementar a análise sintática **top-down** (**descendente**) temos duas opções:

  - **Descendentes recursivos**: é bastante versátil e o método mais adequado para analisador sintático escrito manualmente.

  - **LL(1)**: não é utilizada com frequência, mas é útil como estudo de utilização de uma pilha explícita e também serve como introdução aos analisadores ascendentes. A análise LL(1) tem esse nome porque a sentença é analisada da esquerda para direita (**L**eft-to-right), com a verificação de apenas do símbolo inicial da cadeia (**1** símbolo) de tal forma que tenhamos derivações mais à esquerda (**L**eft most).

Analisador Sintático descendente recursivos

- Há duas formas de analisadores sintáticos descentes recursivos:
  - **Recursiva com Retrocesso (backtraking)**
    - A cada passo, escolhe uma regra e aplica;
    - Se falhar em algum ponto, retrocede e escolhe uma outra regra;
    - O processo termina quando a cadeia é reconhecida ou quando as regras se esgotaram e a cadeia não foi reconhecida.
  - **Recursiva preditiva.**
    - **Não pode ocorrer dúvida** na escolha de qual produção será aplicada.
    - A gramática **não pode conter recursividade à esquerda**.
    - A gramática deve estar **fatorada**.

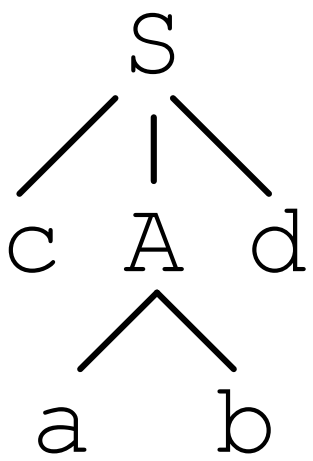
Análise Sintática Recursiva com Retrocesso (backtraking)

- Considere a gramática: cadeia de entrada cad onde S é a produção inicial da gramática

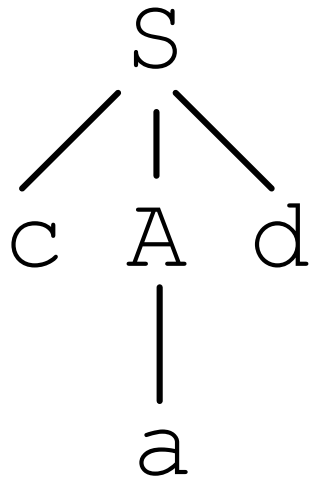
  S ::= c A d

  A ::= ab | a

primeira tentativa



segunda tentativa



Notação BNF (*Backus-Naur Form*)

- A gramática do exemplo anterior está na notação **BNF** (*Backus-Naur Form*) as regras de uma gramática seguem o seguinte padrão:

  <símbolo> ::= <expressão com símbolos>

  Onde o termo <símbolo> (ou em **maiúsculo**) é um **não terminal**, a <expressão com símbolos> consiste em sequências de símbolos separados pela barra vertical (‘|‘), indicando uma escolha e λ representa a palavra vazia.

  Os símbolos que nunca aparecem no lado esquerdo são ditos **terminais**, e normalmente estão em letra minúscula.

ASDR – Analisador Sintático Descente Recursivo Preditivo

- Para que o **ASDR preditivo** tome a decisão correta quanto a produção a ser aplicada, algumas limitações para a gramática devem ser impostas:

  1. Toda produção é da forma $A ::= y\alpha$, onde y é um terminal.
  2. Se $A ::= y_1\alpha_1 | y_2\alpha_2 | .... | y_n\alpha_n$ são todas as alternativas para o não-terminal A, então os terminais y_i são todos diferentes entre si.

- Com restrição faz com que o ASDR só terá uma alternativa a ser escolhida, caso não haja nenhuma escolha, então a sentença entrada não faz parte da linguagem que será reconhecida pelo analisador sintático.

ASDR

- Considere a gramática:
  $E ::= a \mid b \mid +EE \mid *EE$

- A gramática gera palavras na **notação prefixa**, para saber mais acesse:
  http://www.cs.man.ac.uk/~pjj/cs212/fix.html
  https://raj457036.github.io/Simple-Tools/prefixAndPostfixConvertor.html

- Uma maneira simples e comumente utilizada para implementar o ASDR preditivo é usar um conjunto de **funções mutuamente recursivas**, cada função corresponde a um não-terminal da gramática, e dentro da função é analisada a parte inicial da cadeia (`*buffer`).
- Primeiramente iremos implementar uma versão simples do **ASDR**.

Implementação do ASDR

- Para facilitar a integração entre o analisador léxico e a analisador sintático teremos uma **única função** que fará a solicitação de átomos ao analisador léxico (**`obter_átomo()`**). Nesse exemplo os **átomos são caracteres** armazenados no ponteiro `*buffer`.

- Além disso o símbolo inicial para ser avaliado pelo analisador sintático estará armazenado em variável denominada **`lookahead`**.

```
char *buffer; // manipulado pelo analisador léxico
char lookahead; // posteriormente será do tipo TAtomo, declarado no ASDR
```

- Antes de começar a análise sintática, temos que inicializar a variável `lookahead`.

```
lookahead = *buffer++; // obter_átomo()
```

ASDR

- A variável **lookahead** será atualizada pela função **void consome(char atomo)** recebe como parâmetro o símbolo que deveria estar no início do buffer avaliado, caso o símbolo esperado não seja igual ao símbolo que está no início podemos emitir uma mensagem de erro sintático.

```
void consome( char atomo ){
    if( lookahead == atomo )
        lookahead =*buffer++; // obter_atomo();
    else{
        printf("erro sintatico: esperado [%c] encontrado
                [%c]\n",atomo,lookahead);
        exit(1);
    }
}
```

ASDR – Exercício

1) Considere a gramática com símbolo inicial A:

$A ::= bBb$
$B ::= cC \mid eD$
$D ::= da$
$C ::= aA \mid \lambda$

a) Apresente a árvore de derivação para a palavra $bcabcbb$.

b) Construa um ASDR para essa gramática.

ASDR – DESAFIO

2) Considere a gramática com símbolo inicial da gramática igual a S.

$S ::= aAB \mid aBA$

$A ::= b \mid cS$

$B ::= d \mid eS$

a) Apresente a árvore de derivação para a palavra $acadbeadb$.

b) Construa um ASDR para essa gramática.

Aula EAD: Integração do analisador léxico e analisador sintático

- Considere agora a gramática onde os terminais são átomos apresentado na aula que descreve o Mini analisador léxico (Aula3).

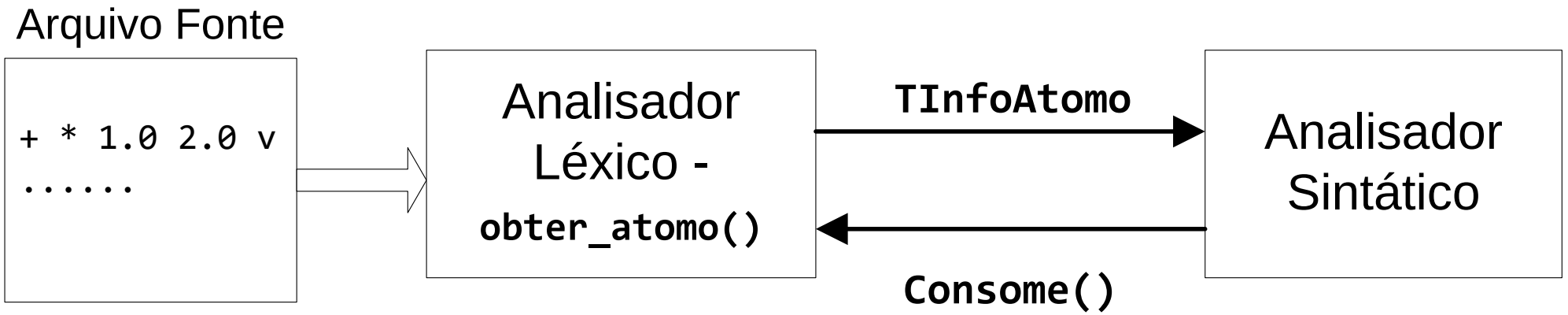
  E ::= numero | identificador | +EE | *EE

  Por questões de legibilidade os tokens estarão separados por espaço em branco. O analisador léxico implementado anteriormente, deverá ser adaptado para atender as necessidade do analisador sintático.

Aula EAD: Integração do analisador léxico e analisador sintático

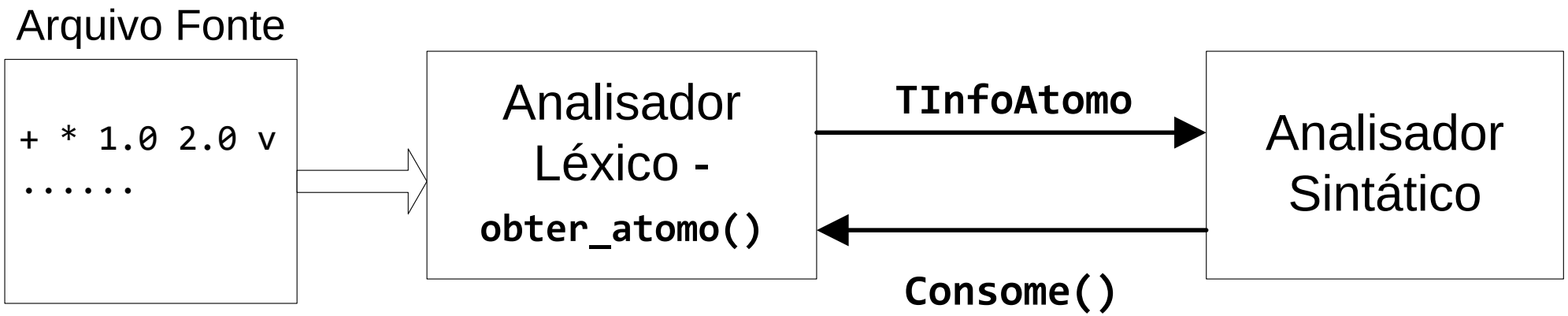
Uma das adaptações é a introdução de mais dois átomos ao mini analisador léxico, os átomos `OP_SOMA` (‘**+**’) e `OP_MULT`(‘*****’).

A interação entre o analisador léxico e o analisador sintático se dará por meio da função **`consome()`**, somente ela fará chamadas à função **`obter_atomo()`** do analisador léxico.



Aula EAD: Integração do analisador léxico e analisador sintático

- No analisador sintático o átomo que será avaliado sempre estará armazenado em variável **`lookahead`** e antes de começar a análise sintática a variável **`lookahead`** deve ser inicializa.
- Como somente a função **`consome()`** chamará a função **`obter_atomo()`** a função **`consome()`** também será responsável em atualizar a variável **`lookahead`**.



Agora vamos a implementação ....

Obrigado !!

Dúvidas ??