Universidade Federal de Pelotas

Análise Léxica

André Rauber Du Bois
dubois@inf.ufpel.edu.br

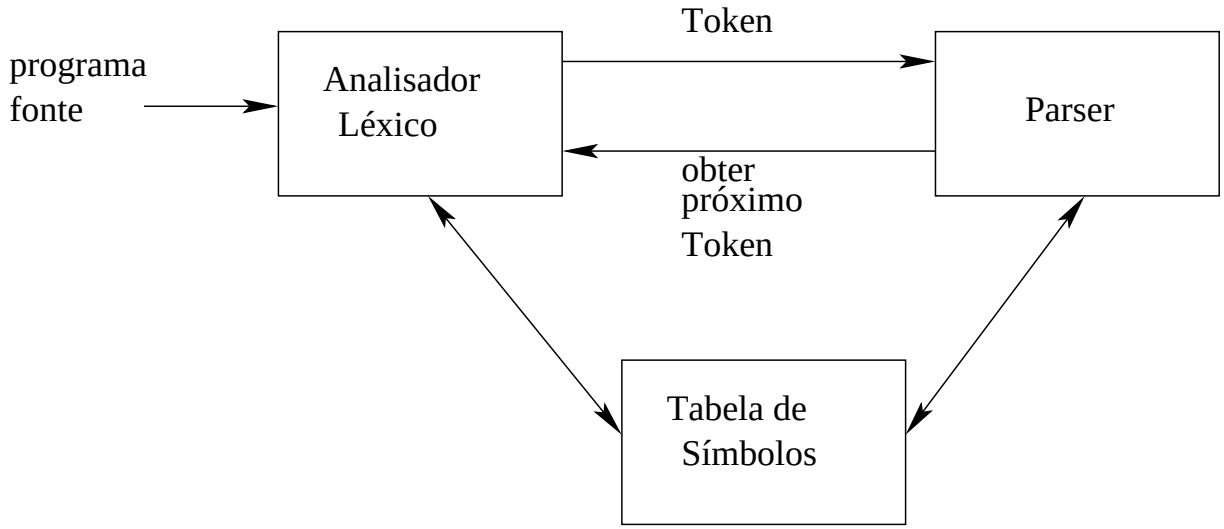
MOTIVAÇÃO

- Técnica de implementação de analisadores léxicos podem ser usadas para qualquer tipo de programa que tenha que reconhecer padrões de caracteres
- Processamento de arquivos, recuperação de informações
- Iremos discutir a implementação de analisadores léxicos e o uso de programas que geram analisadores léxicos. Ex: LEX
- Uso de expressões regulares para reconhecer padrões
- Perl, awk, shell do linux

ANALISADOR LÉXICO

- Primeira fase do compilador. Lê os caracteres de entrada e produz uma sequência de *tokens* que o *parser* utiliza para a análise sintática
- Comumente implementado como uma subrotina do parser (Análise Sintática), oferecendo um comando do tipo *obter próximo token* (ex: o compilador simples apresentado em aula)
- Lê o texto fonte e performa tarefas secundárias do tipo remover espaços, comentários, tabulações (i.e., layout)

Interação do Analisador Léxico com o Parser



Separando Análise Léxica da Sintática

- A separação da análise léxica da sintática é importante pois
  - Permite simplificar as duas análises. Um parser que deve lidar com espaços em branco e comentários seria muito mais complicado
  - Eficiência. Tendo um analisador léxico separado permite construir um processador especializado e talvez mais eficiente
  - Maior portabilidade. Peculiaridades do alfabeto de entrada e leitura dos arquivos podem ser restringidas ao analisador léxico.

TOKEN, LEXEMA E PADRÃO

- Token é a classe em que se encaixa o símbolo léxico. Ex: identificador, constante, palavra-reservada, operadores, etc. São os símbolos terminais em uma gramática
- Lexema é um conjunto de caracteres no programa-fonte que é reconhecido pelo padrão do Token
- O padrão é descreve como reconhecer uma cadeia de entrada como um token. Expressão regular.
  - Ex: `int pi = 3.14`, a subcadeia `pi` é um lexema para o token *identificador*

Token	Lexema	Descrição Informal do Padrão
if	`if`	`if`
relação	`<`,`<=`,`>`, `>=`,`=`,`!=`	< ou <= ou > ou >= ou = ou !=
id	`pi`, `contador`, `x`	letra seguida por letras e/ou dígitos
num	`3.1416`, `0`, `6.02`	qualquer constante numérica

ATRIBUTOS DO TOKEN

- Um token reconhecido pode possuir atributos. Ex: no compilador didático tinhamos o símbolo ASCII para os operadores
- Na prática, geralmente o atributo é um ponteiro para a tabela de símbolos onde são guardadas as informações importantes. Ex: lexema, número da linha em que foi encontrado, etc.
  - Ex: `x = 3 * 2`
    tokens e seus atributos:
    <**id**, ponteiro para a entrada na tabela de símbolos de x>
    <**operador_atribuicao**>
    <**num**, valor inteiro 3>
    <**operador_de_multiplicacao**>
    <**num**, valor inteiro 2>

Erros Léxicos

- São poucos os erros que podem ser detectados pela análise léxica
- Por exemplo, quando encontra a cadeia `fi`

```
fi ( a== true )
```

- O analisador léxico geralmente nao consegue detectar se `fi` é um identificador, ou se é a grafia errada do `if`
- O analisador prossegue, e o erro é identificado usando a gramática presente no analisador sintático

Implementando Analisadores Léxicos

- Existem basicamente, duas maneiras de se implementar um analisador Léxico:
  - Usar uma linguagem de programação para escrever o analisador léxico(Ex: o compilador simples apresentado na outra aula)
  - Usar um gerador de analisadores léxicos. (Ex: O analisador Lex, que será apresentado mais tarde). Esse tipo de analisador recebe a especificação dos tokens, baseada em expressões regulares, e gera o analisador léxico

LINGUAGENS

- O símbolos de uma linguagem poderiam ser descritos de forma informal no manual da linguagem. Ex: Um identificador é uma sequência de letras ou dígitos, sempre começando com uma letra.
- Para a construção de compiladores, geralmente se usa *expressões regulares* para descrever os símbolos. Evita-se assim ambiguidades, e pode-se usar técnicas de implementação já conhecidas para se criar o analisador léxico
- Uma expressão regular é uma fórmula que descreve um conjunto de strings, possivelmente infinito

LINGUAGENS

- Alfabeto: Conjunto de Símbolos. Ex: $\{0,1\}$ (alfabeto binário)
- Cadeia/Sentença/Palavra: sequência finita tirada de um alfabeto
- Comprimento de uma cadeia *s*: $|s|$
- Cadeia vazia: ϵ
- Linguagem: qualquer conjunto de cadeias sobre um alfabeto
- Concatenação: se *x* e *y* são cadeias, sua concatenação é descrita por *xy*
- Exponenciação: $s^0 = \epsilon$, e para $i > 0$, s^i significa $s^{i-1}s$. Dessa forma $s^2 = ss$, $s^3 = sss$, e assim por diante

Operações em Linguagens

- União: $L \cup M$ = $\{s \mid s$ está em L ou s está em $M\}$
- Concatenação: LM = $\{st \mid s$ está em L e t está em $M\}$
- L^* = zero ou mais concatenações de L
- L^+ = uma ou mais concatenações de L

Operações em Linguagens

Sendo L o cojunto/linguagem com os símbolos sendo todas as letras maiúsculas e minúsculas e D a linguagem que contém os dez dígitos decimais, temos

- $L \cup D$ é o conjunto de letras e dígitos
- LD é o conjunto com as palavras consistindo em uma letra seguida por um dígito
- L^4 é o conjunto de todas as cadeias com quatro letras
- L^* é o conjunto de todas as cadeias de letras, incluindo ϵ
- $L(L \cup D)^*$ é o conjunto de todas as cadeias de letras e dígitos, que começam por uma letra
- D^+ é o conjunto de todas as cadeias de um ou mais dígitos

EXPRESSÕES REGULARES

Uma *expressão regular* r, sobre um conjunto de símbolos T, representa uma linguagem L(r), a qual pode ser definida indutivamente

1. $\{\epsilon\}$ representa a linguagem cuja única palavra é a vazia
2. $\{x \mid x \in T\}$, ou seja $\{x\}$, x representa a linguagem cuja única palavra é x
3. Se r_1 e r_2 são expressões regulares definindo as linguagens $L(r_1)$ e $L(r_2)$, tem-se que:
   - $r_1 \mid r_2$ é uma expressão regular denotando $L(r_1) \cup L(r_2)$
   - $r_1 r_2$ é uma expressão regular denotando $L(r_1)L(r_2)$
   - $(r)^*$ é uma expressão regular denotando $L(r)^*$
   - (r) é uma expressão regular denotando $L(r)$

Expressões Regulares

Exemplo: seja T = $\{a, b\}$

1. $a \mid b$ denota o conjunto $\{a, b\}$
2. $(a \mid b)(a \mid b)$ denota o conjunto $\{\ aa, ab, ba, bb\ \}$
3. a^* conjunto de zero ou mais a's, $\{\ \epsilon, a, aa, aaa, ...\}$
4. $(a \mid b)^*$ conjunto de todas as cadeias contendo zero ou mais instâncias de a ou b. Outra expressão: $(a^*b^*)^*$
5. $a \mid a^*b$ conjunto contendo a cadeia a e todas as cadeias com zero ou mais a's seguidos por um b

DEFINIÇÕES REGULARES

Pode-se definir nomes para expressões regulares:

1. Exemplo: identificadores em Pascal
   letra $\rightarrow A \mid B \mid ... \mid Z \mid a \mid b \mid ... \mid z$
   digito$\rightarrow 0 \mid 1 \mid ... \mid 9$
   identificador $\rightarrow$ **letra** $\mid$ (**letra** $\mid$ **digito**)*

DEFINIÇÕES REGULARES

Números em Pascal: (Ex: `432, 39.37, 6.78E4, 6.555E+5, 8.4E-55`)

digito $\rightarrow 0 \mid 1 \mid ... \mid 9$
digitos $\rightarrow$ **digito digito***
fracao $\rightarrow$ **.digitos** $\mid \epsilon$
expoente $\rightarrow$ (E (+ $\mid$ - $\mid \epsilon$) **digitos**) $\mid \epsilon$
num $\rightarrow$ **digitos fracao expoente**

SIMPLIFICAÇÕES EM DEFINIÇÕES REGULARES

1. *Uma ou mais ocorrências*: r^+. Ex: r^* = $r^+ \mid \epsilon$, r^+ = rr^*
2. *Zero ou uma ocorrência*: $r?$. Ex: $r?$ = $r \mid \epsilon$

digito $\rightarrow 0 \mid 1 \mid ... \mid 9$
digitos $\rightarrow$ **digito**$^+$
fracao $\rightarrow$ (**.digitos**) $?$
expoente $\rightarrow$ (E (+ | -)? **digitos**)?
num $\rightarrow$ **digitos fracao expoente**

OUTRO EXEMPLO PARA IDENTIFICADORES

Um *identificador* é uma sequência de letras, dígitos e sublinhados que se inicia com uma letra, dois sublinhados consecutivos não são permitidos, e nem sublinhado no final.

letra $\rightarrow$ [a-zA-Z]
digitos $\rightarrow$ [0-9]
sublinha $\rightarrow$ _
letraoudigito $\rightarrow$ **letra** $|$ **digito**
finalsublinhado $\rightarrow$ **sublinha letraoudigito**$^{+}$
identificador $\rightarrow$ **letra letraoudigito*** **finalsublinhado***

UM ANALISADOR LÉXICO SIMPLES

- Analisador que reconhece os símbolos: Identificador (conforme slide anterior), inteiros, símbolos `+`, `-`, `*` e `/` e os separadores (**,**(vírgula), **;**, **(**, **)**, {, }).
- Comentários são ignorados: # comentario #
- Arquivo `lex.h` possui as definições de classes dos tokens e do tipo `Token`
- O arquivo `lex_cmp.c` contém o an. léxico