Judson Santos Santiago

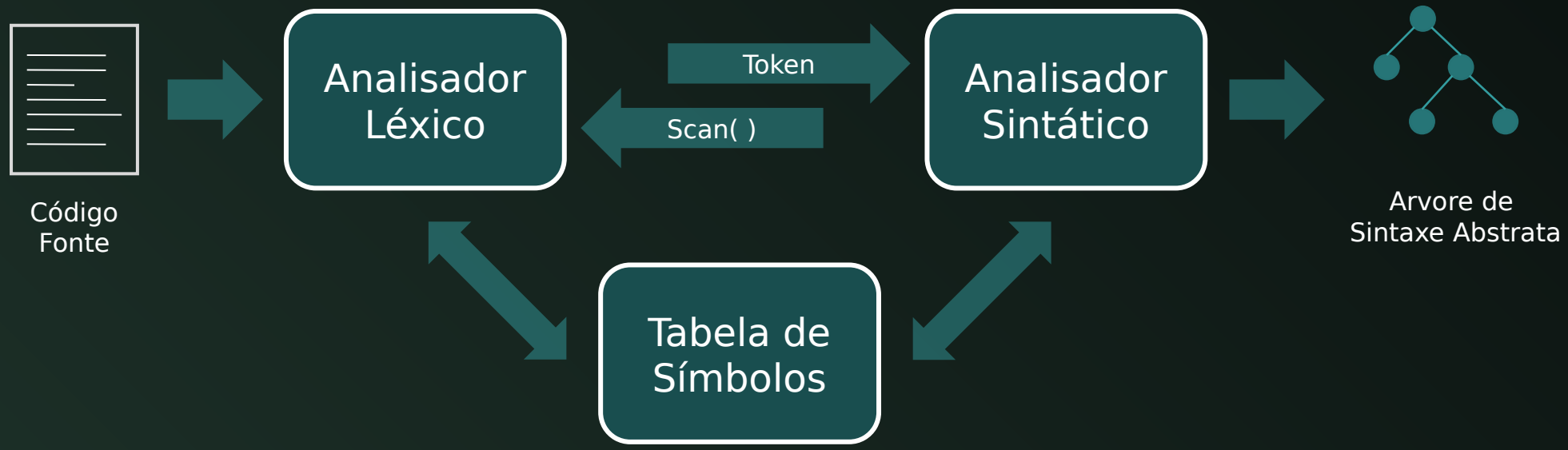
Expressões Regulares

Compiladores

Introdução

- As principais tarefas de um analisador léxico são:
  - Ler os caracteres da entrada
  - Agrupá-los em lexemas
  - Produzir uma sequência de **tokens**
- É comum ele interagir com a tabela de símbolos
  - Quando o lexema é um identificador, ele precisa ir para a tabela
- Ele também interage com o analisador sintático
  - O sintático chama o léxico através de uma função `Scan()`

Introdução



Interações entre
Analisador Léxico e Sintático

Introdução

- É comum o analisador léxico realizar também outras tarefas:
  - Remover caracteres sem significado para o compilador
    - Espaço, tabulação e salto de linha
    - Comentários do programador
  - Correlacionar as mensagens de erro com o programa fonte
    - Manter controle sobre o número da linha e coluna
    - Guardar uma cópia do programa fonte para exibição
  - Fazer a expansão de macros
    - Se a linguagem suportar um pré-processador de macros
    - Ex.: #include, #define ,etc.

Introdução

- As fases de análises léxica e sintática poderiam ser combinadas, mas normalmente elas são feitas separadas devido:
  - Simplicidade de projeto
    Um analisador sintático que tivesse que lidar com espaços em branco e comentários seria muito mais complexo
  - Eficiência do compilador
    Técnicas de *buffering* podem acelerar significativamente a leitura dos caracteres da entrada
  - Portabilidade do compilador
    As peculiaridades do dispositivo de entrada ficam isoladas das demais fases do compilador

Terminologia

- Ao discutir análise léxica, se faz necessário conhecer os termos:
  - Token
    É um par consistindo em um nome e um atributo (opcional)
    Ex.: `<id, "total">, <num, 60>, <+>`
  - Padrão
    É uma descrição da forma que as sequências de caracteres podem assumir
    Ex.: uma letra inicial seguida de letras, números ou sublinhado
  - Lexema
    É uma sequência de caracteres no programa fonte que casa
    com o padrão para um token

Terminologia

Token	Padrão	Lexema
if	caracteres i, f	`if`
else	caracteres e, l, s, e	`else`
rel	caracteres > ou < ou >= ou <= ou == ou !=	`<=, !=`
+	caractere +	`+`
*****	caractere *	`*`
id	letra seguida por letras e dígitos	`pi, total, c2`
num	sequência de dígitos com “.” e “E” opcional	`3.14159, 50, 6.2E5`
literal	Um ou mais caracteres cercados por "	`"falha grave", "erro"`

Exemplos de Token, Padrão e Lexema

Tokens

- Para a maior parte das linguagens de programação é suficiente:
  - Um token para cada palavra-chave
    Ex.: `int`, `main`, `if`, `return`, etc.
  - Tokens para os operadores (individuais ou agrupados)
    Ex.: `+`, `-`, `>` , `<`, `rel`, `&&`, etc.
  - Um token para os identificadores
    Ex.: `id`
  - Um ou mais tokens para as constantes, como números e strings
    Ex.: `num`, `integer`, `floating`, `literal`, etc.
  - Um token para cada símbolo de pontuação
    Ex.: `;`, `(`, `)`, `{`, `}`, `LP`, `RP`, `LB`, `RB`, etc.

Tokens

- Quando um **token** deve ter atributos?
  - Quando mais de um lexema casar com o padrão de um token
    - Os lexemas `3`, `52` e `829` casam com o padrão de um **num**
    - Os lexemas `total`, `cont` e `val` casam com o padrão de um **id**
  - As próximas fases precisam saber qual lexema foi casado
  - O analisador léxico precisa oferecer informações adicionais
    - Nestes casos utilizam-se atributos para os **tokens**:
      - `<num,3>`, `<num,52>`, `<num,829>`
      - `<id,"total">`, `<id,"cont">`, `<id,"func">`

Tokens

- Normalmente, os **tokens** possuem apenas um atributo associado
  - Mas esse atributo pode ser um registro que agrupa diversas informações
  - Ou um ponteiro para a tabela de símbolos
    - Um **id** pode guardar seu lexema e tipo na tabela de símbolos
    - <**id**, ponteiro para tabela de símbolos do identificador>

```
total = soma * 2
```

```
<id, ponteiro para total na tabela >
<=>
<id, ponteiro para soma na tabela >
<*>
<num, ponteiro para 2 na tabela >
```

Tabela de Símbolos

total	int	id
soma	float	id
"2"	int	num

Exercício

1. Divida o seguinte programa C++ em tokens:

```
float limitedSquare(x) {
    // retorna x ao quadrado, limitado ao valor 100
    return (x <= -10.0 || x >= 10.0) ? 100 : x*x;
}
```

2. As linguagens de marcação, como HTML e XML, são diferentes das linguagens convencionais porque as marcas de pontuação são muito numerosas. Sugira como dividir o documento HTML a seguir:

```
Aqui está uma foto da <b>minha casa</b>:
<img src="casa.jpg"><br>
Veja <a href="maisfotos.html">mais fotos</a> se você gostar dessa.
```

Especificação de Tokens

- O reconhecimento de padrões presentes em cadeias de caracteres e a criação de **tokens** pode ser automatizada
  - Geradores de analisadores léxicos
    Ex.: `Lex`, `Flex`, etc.
- As expressões regulares são uma importante notação para especificar os padrões dos tokens
  - Não expressam todos os padrões possíveis
  - Eficientes para os padrões normalmente usados nas linguagens

$$letra\_\ (letra\_ \mid digito)^*$$

Cadeias e Linguagens

- Um alfabeto é qualquer conjunto finito de símbolos
  - { 0, 1} é o alfabeto binário
  - { A, C, G, T } é o alfabeto genético
  - { a, e, i, o, u } é um alfabeto formado pelas vogais
- A tabela ASCII é um exemplo importante de alfabeto
  - Usado em muitos sistemas computacionais
- O Unicode é um alfabeto que inclui aproximadamente 100.000 caracteres de línguas do mundo inteiro

Cadeias e Linguagens

- Uma cadeia[†] é uma sequência de símbolos de um alfabeto
  - O tamanho de uma cadeia s é dado por |s|
  - A cadeia vazia, indicada por ϵ, tem tamanho zero

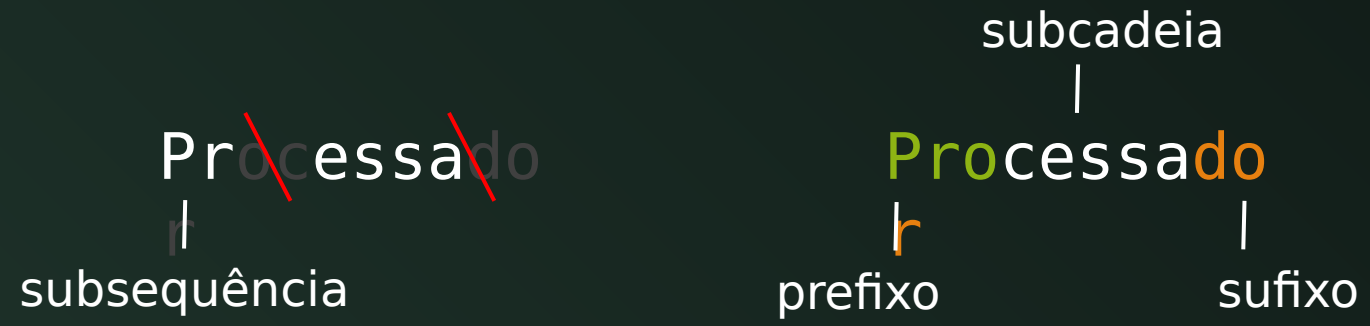
Ex.:

```
01101110110101110101O   - binária
AGTCCTAGAGTCAGTGGTTAC   - genética
aeiouiouiouaeieiouae    - vogais
Processador de Linguagem - ASCII
Expressões regulares    - Unicode
```

[†] Também chamada de palavra, sentença ou frase

Cadeias e Linguagens

- Algumas partes de cadeias recebem nomes:
  - Prefixo: obtido pela remoção de zero ou mais símbolos do final
  - Sufixo: obtido pela remoção de zero ou mais símbolos do início
  - Subcadeia: obtido pela remoção de qualquer prefixo e sufixo
  - Subsequência: obtido pela remoção de zero ou mais posições não necessariamente consecutivas



Cadeias e Linguagens

- Uma linguagem é um conjunto de cadeias[†] de algum alfabeto
  - O conjunto de todas as frases corretas em português é uma linguagem
  - O conjunto de todos os programas C++ bem formados também
  - O conjunto vazio { ▦ } também é uma linguagem
- As cadeias podem ser combinadas através de operações
  - As operações básicas são:
    - União
    - Concatenação

Cadeias e Linguagens

- Se x e y são cadeias:
  - A união é indicada por x ∪ y
    Ex.: se x = ”sol” e y = ”dado”, então x ∪ y = { ”sol”, ”dado” }
  - A concatenação é indicada por xy
    Ex.: se x = ”sol” e y = ”dado”, então xy = ”soldado”
  - A cadeia vazia é a identidade da concatenação: ϵ s = s ϵ = s
  - A concatenação pode ser vista como um produto
    - A partir daí define-se a exponenciação:
      $s^0 = \epsilon$, e para todo $i > 0$, $s^i = s^{i-1}s$
      ou seja: $s^1 = s$, $s^2 = ss$, $s^3 = sss$, etc.

Operações sobre Linguagens

- As linguagens também podem ser combinadas:
  - União – conjunto obtido pela união das cadeias das linguagens L e M
  - Concatenação – conjunto obtido concatenando cadeias de L e M
  - Fechamento – conjunto obtido concatenando L zero ou mais vezes

Operação	Definição e Notação
União de L e M	L M = { s | s está em L ou está em M }
Concatenação de L e M	LM = { st | s está em L e t está em M }
Fecho Kleene de L	

Operações sobre Linguagens

- Considerando os conjuntos

```
L = { A, B, ..., Z, a, b, ..., z }
D = { 0, 1, ..., 9 }
```

  - Podemos pensar em L e D como:
    - Os alfabetos de letras e dígitos, respectivamente
    - Linguagens em que as cadeias tem tamanho um

Operações sobre Linguagens

- Se L e D forem linguagens, podemos construir outras linguagens usando as operações:
  - L ∪ D é um conjunto de letras e dígitos – 62 cadeias de tamanho um
  - LD é o conjunto de 520 cadeias de tamanho dois – letra seguida por dígito
  - L^4 é o conjunto de todas as cadeias de 4 letras
  - L* é o conjunto de todas as cadeias de letras, incluindo a cadeia vazia ∅
  - D* é o conjunto de todas as cadeias de dígitos, incluindo a cadeia vazia ∅
  - L(L ∪ D)* é o conjunto de todas as cadeias de letras e dígitos que iniciam com uma letra

Expressões Regulares

- Usando as operações sobre linguagens é possível, por exemplo, descrever os identificadores da linguagem C++
  - *letra_* significa qualquer letra ou sublinhado
  - *dígito* significa qualquer dígito
  - *letra_* (*letra_* | *dígito*)*
- Esse processo é tão importante que uma notação chamada expressões regulares foi criada para descrever todas as linguagens formadas a partir desses operadores

Expressões Regulares

- Cada expressão regular *r* denota uma linguagem *L(r)*
  - As regras que definem as expressões regulares para algum alfabeto Σ:
    - Base:
      1. ϵ é uma expressão regular, e L(ϵ) é { ϵ }
      2. Se *a* é um símbolo de Σ então **a** é uma expressão regular e L(**a**) = { a }
    - Indução:
      1. (r) é uma expressão regular denotando L(r)
      2. (r)|(s) é uma expressão regular denotando a linguagem L(r)L(s)
      3. (r)(s) é uma expressão regular denotando a linguagem L(r)L(s)
      4. (r)* é uma expressão regular denotando L(r)*

Expressões Regulares

- Podemos remover muitos parênteses definindo:
  a) A precedência dos operadores (da maior para a menor):
     - O fechamento *
     - A concatenação
     - A união |
  b) Todos os operadores são associativos à esquerda

Exemplo: considerando Σ = { a, b }, temos que

a|a*b representa zero ou mais a's concatenados com b, ou a
= {a, b, ab, aab, aaab, ...}

Expressões Regulares

- Mais alguns exemplos:

1. **a|b** denota a linguagem {a,b}
2. **(a|b)(a|b)** denota a linguagem {aa,ab,ba,bb}
   (aa|ab|ba|bb) é outra expressão regular para a mesma linguagem
3. **a*** denota todas as cadeias de zero ou mais a's, ou seja, {ϵ,a,aa,aaa,...}
4. **(a|b)*** denota a linguagem {ϵ,a,b,aa,ab,ba,bb,aaa,...}
   (a*b*)* é outra expressão regular para a mesma linguagem

Expressões Regulares

- Uma linguagem que pode ser definida por uma expressão regular é chamada de conjunto regular
  - Duas expressões r e s são equivalentes se denotam o mesmo conjunto regular

Lei	Descrição
`r|s = s|r`	A união é comutativa
`r|(s|t) = (r|s)|t`	A união é associativa
`r(st) = (rs)t`	A concatenação é associativa
`r(s|t) = rs|rt; (s|t)r = sr|tr`	A concatenação é distributiva com a união
`ϵr = rϵ = r`	Vazio é a identidade da concatenação
`r* = (r|ϵ)*`	Vazio é garantido em um fechamento
`r** = r*`	O fechamento é idempotente

Definições Regulares

- Por conveniência de notação, podemos dar nomes a certas expressões regulares e usá-los em outras expressões
  - Uma definição regular é uma sequência de definições da forma:

    $d_1 \rightarrow r_1$
    $d_2 \rightarrow r_2$
    ...
    $d_n \rightarrow r_n$

  - Onde:
    - Cada d_i é um novo símbolo, não em Σ e diferente de quaisquer outros d's
    - Cada r_i é uma expressão regular envolvendo símbolos de $\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$

Definições Regulares

- **Exemplo 1**: os identificadores da linguagem C++

$$
\begin{aligned}
\textit{letra\_} &\rightarrow A \mid B \mid \ldots \mid Z \mid a \mid b \mid \ldots \mid z \mid \_ \\
\textit{dígito} &\rightarrow 0 \mid 1 \mid \ldots \mid 9 \\
\textit{id} &\rightarrow \textit{letra\_}(\textit{letra\_} \mid \textit{dígito})^*
\end{aligned}
$$

- **Exemplo 2**: números sem sinal (inteiros ou ponto-flutuantes)

$$
\begin{aligned}
\textit{dígito} &\rightarrow 0 \mid 1 \mid \ldots \mid 9 \\
\textit{dígitos} &\rightarrow \textit{dígito dígito}^* \\
\textit{optFrac} &\rightarrow .\textit{dígitos} \mid \epsilon \\
\textit{optExp} &\rightarrow (E(+ \mid - \mid \epsilon)\textit{dígitos}) \mid \epsilon \\
\textit{número} &\rightarrow \textit{dígitos optFrac optExp}
\end{aligned}
$$

Exercícios

1. Descreva as linguagens denotadas pelas seguintes expressões regulares:
   a) $a(a|b)^*a$
   b) $((\epsilon|a)b^*)^*$
   c) $(a|b)^*a(a|b)(a|b)$
   d) $a^*ba^*ba^*ba^*$

2. Escreva definições regulares para as seguintes linguagens:
   a) Todas as cadeias de letras minúsculas que contém as cinco vogais em ordem
   b) Horários no formato de 12 (AM e PM) ou 24 horas

Resumo

- **Expressões regulares** podem ser usadas para representar os padrões dos tokens válidos de uma linguagem de programação
  - União, concatenação e fechamento são as operações básicas

$$a|a^*b = \{a,b,ab,aab,aaab,...\}$$

- **Definições regulares** permitem dar nomes às expressões regulares

$$\begin{aligned} \textit{letra\_} &\rightarrow A \mid B \mid \ldots \mid Z \mid a \mid b \mid \ldots \mid z \mid \_ \\ \textit{dígito} &\rightarrow 0 \mid 1 \mid \ldots \mid 9 \\ \textit{id} &\rightarrow \textit{letra\_}(\textit{letra\_} \mid \textit{dígito})^* \end{aligned}$$