Curso de Engenharia de Computação
ECM253 – Linguagens Formais, Autômatos e Compiladores

Análise Léxica

Slides da disciplina ECM253 – Linguagens Formais, Autômatos e Compiladores
Curso de Engenharia de Computação
Instituto Mauá de Tecnologia – Escola de Engenharia Mauá
Prof. Marco Antonio Furlan de Souza
<marco.furlan@maua.br>

Conceitos de Análise Léxica

- **Tarefas do Analisador Léxico**
  - A tarefa do **analisador léxico** ou **scanner** é transformar um fluxo de caracteres em um fluxo de palavras na linguagem de entrada;
  - Cada **palavra** precisa ser **classificada** em uma **categoria sintática**, ou “classe gramatical” da linguagem de entrada;
  - O **scanner** é o **único passo** do compilador a ter **contato** com cada **caractere** do **programa de entrada**;
  - O **scanner aplica** um **conjunto** de **regras** que **descrevem** a **estrutura léxica** da linguagem de programação de **entrada**, às vezes chamada de sua **microssintaxe**;
  - É importante que o scanner seja projetado para ter um alto desempenho – prefere-se, atualmente, recorrer a programas **geradores de scanner** do que implementá-los manualmente (a menos que se tenha uma boa justificativa).

Conceitos de Análise Léxica

- **Como um scanner funciona?**
  - A **base para teórica** para a implementação de um scanner é o **autômato finito determinístico** (DFA);
  - O que um scanner **reconhece** e **classifica** são os símbolos terminais de uma linguagem – símbolos fixos que compõem o texto de um programa;
  - Esses símbolos são **categorizados** pelo scanner. Por exemplo, em Java tem-se **palavras reservadas** (`if`, `class`, `package ...`), palavras que representam **identificadores** (`temp`, `getName`, `...`), **constantes** ou **literais** que serão utilizadas em expressões (“`Mensagem`”, `10`, `true`, `...`) , **sinais de pontuação** (`{`, `;`, `)`, `...`), **operadores** (`+`, `%`, `instanceof`, `...`);
  - Um programa de scanner possui um **laço de repetição**, **governado** pela **chegada do símbolo de fim de arquivo** ou pela **presença de erro**, em que tentará, a cada entrada de símbolo, **classificar** uma palavra dentre as **categorias sintáticas** como as do item anterior;
  - Assim, o scanner é um programa que **normalmente contém diversos DFA’s**: um para cada categoria sintática da linguagem, e sua implementação mais eficiente é por meio de **tabelas** de estado.

Conceitos de Análise Léxica

- **Exemplo de scanner simples em Java**

```java
package lexer;
import java.io.IOException;
import java.util.Hashtable;
public class Lexer {
        public int line = 0;
        private char peek = ' ';
        private Hashtable<String, Word> words = new Hashtable<>();
        void reserve(Word t) {
                words.put(t.lexeme, t);
        }
        public Lexer() {
                reserve(new Word(Tag.TRUE, "true"));
                reserve(new Word(Tag.FALSE, "false"));
        }
        public Token scan() throws IOException {
                for (;; peek = (char) System.in.read()) {
                        if (peek == ' ' || peek == '\t')
                                continue;
                        else if (peek == '\n')
                                line = line + 1;
                        else
                                break;
                }
                if (Character.isDigit(peek)) {
                        int v = 0;
                        do {
                                v = 10 * v + Character.digit(peek, 10);
                                peek = (char) System.in.read();
                        } while (Character.isDigit(peek));
                        return new Num(v);
                }
                if (Character.isLetter(peek)) {
                        StringBuffer b = new StringBuffer();
                        do {
                                b.append(peek);
                                peek = (char) System.in.read();
                        } while (Character.isLetterOrDigit(peek));
                        String s = b.toString();
                        Word w = words.get(s);
                        if (w != null)
                                return w;
                        w = new Word(Tag.ID, s);
                        words.put(s, w);
                        return w;
                }
                Token t = new Token(peek);
                peek = ' ';
                return t;
        }
}
```

Conceitos de Análise Léxica

- **Exemplo de scanner simples em Java**
  - É tarefa do analisador sintático recuperar as **marcas** (**tokens**) que são produzidas pelo scanner (aqui realizada pelo programa principal para simplificar):

```java
package lexer;
import java.io.IOException;
public class Main {
    public static void main(String[] args) {
        try {
            Lexer lexer = new Lexer();
            Token t;
            while ((t = lexer.scan()).tag != (char)(-1)) {
                System.out.print(t);
            }
            System.out.println();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

Conceitos de Análise Léxica

- **Lexema, marca e atributos**
  - **Lexema**: é uma **sequência de caracteres** no programa fonte que **corresponde** ao **padrão** de uma **marca** e é **identificada** pelo **analisador léxico** como uma **instância** dessa marca;
  - **Marca** (ou **token**): é uma cadeia de símbolos com um significado atribuído. Ela pode ser um número simples ou um tipo estruturado contendo algum tipo de identificador único (número, enumeração) e possivelmente um ou mais atributos;
  - **Atributo de marca**: adiciona informações necessárias para se utilizar a marca de modo conveniente. Por exemplo, pode-se adicionar o lexema de um identificador a uma marca do tipo identificador, para que, no analisador sintático, adicione ou recupere valores deste identificador em uma tabela de símbolos.

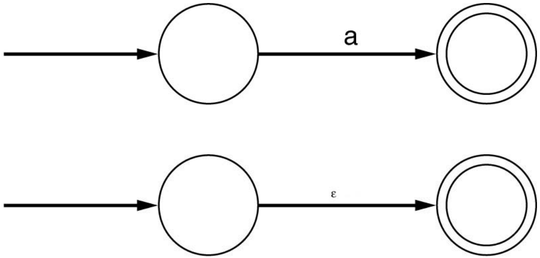
De expressões regulares a scanners

- **Conceitos**
  - **Expressões regulares são compactas** – preferíveis para **descrever marcas**;
  - O algoritmo de conversão **converte a expressão regular em NFA** e, depois, **converte este último em um DFA**. Por **fim**, tal **DFA é minimizado** quanto a seus estados:

  - **Construção de Thompson**: utiliza transições-ϵ para juntar máquinas de cada pedaço de uma expressão regular e formar uma máquina correspondente à expressão toda

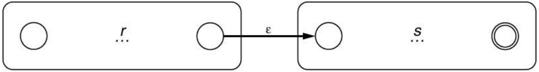
De expressões regulares a scanners

- **Expressões básicas**
  - As expressões regulares na forma $\mathbf{a}$, ϵ e ϕ são traduzidas em NFAs assim:



De expressões regulares a scanners

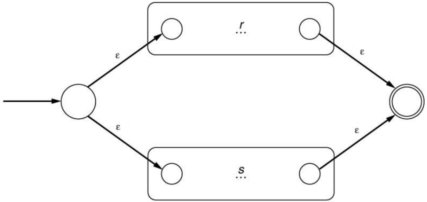
- **Concatenação**
  - A expressão regular na forma rs, onde r e s são expressões regulares, é assim traduzida em um NFA:



  - A NFA resultante é obtida pela ligação das NFAs de r e s por meio de uma transição ϵ
  - O estado inicial de r será o estado inicial da nova máquina, enquanto que o(s) estado(s) final(ais) de s será(ão) o(s) estado(s) final(ais) da nova máquina
  - Linguagem aceita: $L(rs) = L(r)L(s)$

De expressões regulares a scanners

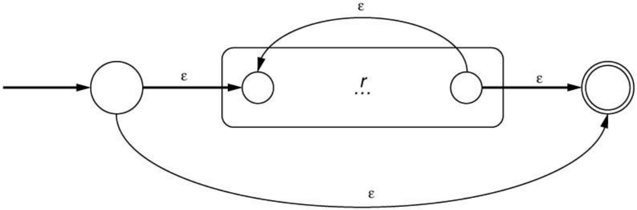
- **Escolha entre alternativas**
  - A expressão regular na forma $r|s$, onde r e s são expressões regulares, é assim traduzida em um NFA:



  - A NFA resultante é obtida pela ligação em paralelo das NFAs de r e s por meio de uma transição ϵ
  - O estado inicial da nova máquina é ligado aos estados iniciais das NFAs de r e s por meio de transições ϵ e os estados finais das NFAs de r e s são ligados ao estado final da nova máquina

De expressões regulares a scanners

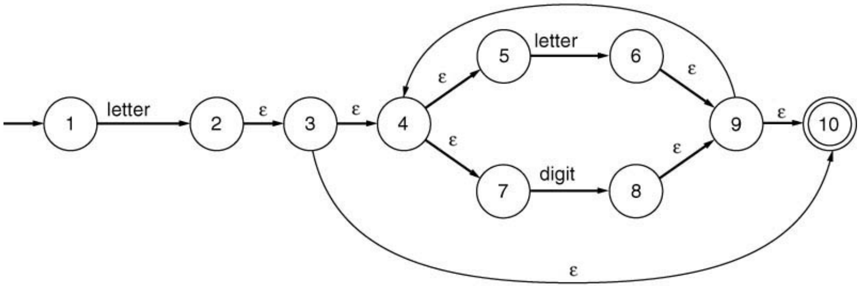
- **Repetição**
  - A expressão regular na forma r^* é assim traduzida em um NFA:



  - A NFA resultante é obtida pela adição de transições ϵ do estado inicial ao estado inicial de r, do(s) estado(s) final(ais) de r ao estado final da nova máquina e também da transição ϵ do estado inicial ao estado final da nova máquina

De expressões regulares a scanners

- **Exemplo**
  - O NFA da expressão regular $\mathbf{letra(letra|digito)}^*$ é (**verificar**):



De expressões regulares a scanners

- **Para converter um NFA em DFA é necessário**

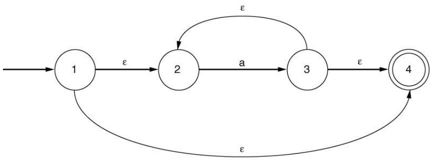
  1. **Eliminar** as **transições** ϵ
  2. **Eliminar** as **transições múltiplas** a **partir** de um **estado**, para um **mesmo símbolo** de entrada

- A **eliminação** das **transições** ϵ **requer a construção de fechos** ϵ. Um fecho ϵ é o conjunto de **estados alcançáveis por transições** ϵ a partir de um ou mais estados;

- A **eliminação** as **transições múltiplas** a partir de um estado, para um mesmo símbolo de entrada **requer o acompanhamento do conjunto de estados atingíveis pelo casamento com um único caractere**.

De expressões regulares a scanners

- **Fechamento ϵ de um conjunto de estados**
  - O fechamento ϵ de um único estado s, representado como $\overline{s}$, é o conjunto de todos os estados alcançáveis por uma série de zero ou mais transições ϵ a partir desse estado:



  - No exemplo acima: $\overline{1} = \{1,2,4\}$, $\overline{2} = \{2\}$, $\overline{3} = \{2,3,4\}$, $\overline{4} = \{4\}$
  - O fechamento ϵ de um conjunto de estados, $\overline{S}$, é a união dos fechamentos ϵ de cada estado individual

$$\overline{S} = \bigcup_{s \in S} \overline{s}$$

  - No exemplo: $\overline{\{1,3\}} = \overline{1} \cup \overline{3} = \{1,2,4\} \cup \{2,3,4\} = \{1,2,3,4\}$

De expressões regulares a scanners

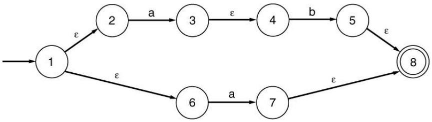
- **Algoritmo para construir o DFA $\overline{M}$, correspondente a um NFA M**

  1. O fecho ϵ do estado inicial de M torna-se o estado inicial de $\overline{M}$
  2. Para esse conjunto e para cada conjunto subseqüente, computam-se as transições com rótulos a da seguinte maneira: dado um conjunto S de estados e um caractere a do alfabeto, computa-se o conjunto $S'_a = \{t |$ para algum $s \in S$ e $((s,a),t) \in T_M\}$
  3. Computa-se $\overline{S'_a}$, o fecho ϵ de S'_a
  4. Acrescenta-se a transição $S \xrightarrow{a} \overline{S'_a}$ à nova máquina $\overline{M}$
  5. Repetir 2–4 até que não seja possível a criação de novos estados e transições

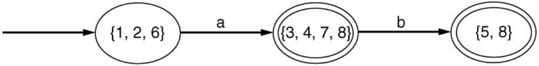
- $\overline{M}$ é o DFA procurado

De expressões regulares a scanners

- **Exemplo**



- Estado inicial $\overline{\{1\}} = \{1, 2, 6\}$
- De 2 e 6 partem transições com o rótulo $\mathbf{a}$. Então $\overline{\{1,2,6\}}_a = \{\overline{3}, \overline{7}\} = \{3,4,7,8\}$. Assim, a transição a ser adicionada é $\{1,2,6\} \xrightarrow{a} \{3,4,7,8\}$. Considerando $\{3,4,7,8\}$, há uma transição de rótulo $\mathbf{b}$ de 4 para 5 e $\overline{\{3,4,7,8\}}_b = \{\overline{5}\} = \{5,8\}$, adicionando a transição $\{3,4,7,8\} \xrightarrow{b} \{5,8\}$. Assim, o DFA equivalente será:



De expressões regulares a scanners

- **Algoritmo para calcular o fechamento ϵ** (COOPER; TORCZON, 2014)

for cada estado $n \in N$ **do** ▷ N é o conjunto de estados
    $E(n) \leftarrow \{n\}$ ▷ $E(n)$ é o fechamento-ϵ de n – a resposta desejada
end for
$WorkList \leftarrow N$
while $WorkList \neq \emptyset$ **do**
    remover n de $WorkList$
    $t \leftarrow \{n\} \cup \bigcup_{n \xrightarrow{\epsilon} p \,\in\, \delta_N} E(p)$ ▷ formar em t um conjunto de todos
▷ os estados alcançáveis por transições ϵ
    if $t \neq E(n)$ **then**
        $E(n) \leftarrow t$
        $WorkList \leftarrow WorkList \cup \{m \mid m \xrightarrow{\epsilon} n \in \delta_N\}$ ▷ adiciona os estados m que possuem
▷ transição ϵ para o estado n em questão
    end if
end while

De expressões regulares a scanners

- **Algoritmo de construção por subconjunto** (COOPER; TORCZON, 2014)

$q_0 \leftarrow E(n_0)$ ▷ $E(n)$ é calculado pelo algoritmo anterior
$Q \leftarrow \{q_0\}$
$WorkList \leftarrow \{q_0\}$
while $WorkList \neq \emptyset$ **do**
    remover q de $WorkList$
    for cada caractere $c \in \Sigma$ **do**
        $t \leftarrow E(\delta(q, c))$ ▷ próximo estado
        $T[q, c] \leftarrow t$ ▷ atualiza a tabela de estados final
        if $t \notin Q$ **then**
            adicione t a Q e também a $WorkList$
        end if
    end for
end while

De expressões regulares a scanners

- **Minimização do número de estados do DFA**
  - Para a minimização de estados do DFA pode-se utilizar o **algoritmo de Hopcroft**;
  - O princípio é simples: detecta-se quando dois estados são equivalentes – quando **ambos produzem o mesmo comportamento sobre qualquer string de entrada**. Desse modo, criam-se classes de equivalência de estados do DFA com base no seu comportamento;
  - **Inicialmente**, **todo DFA possui duas classes** de **equivalência**: uma formada pelos **estados de aceitação** e outra formada pelos **estados de não aceitação**, que certamente devem estar em partições distintas;
  - Depois, **o algoritmo examina cada partição separadamente** procurando **encontrar** se existe a **necessidade** de criar **novas partições** pela verificação que existem estados que não são equivalentes em seu interior;
  - Para isso, se utiliza uma **sub-rotina auxiliar denominada `Split(p)`**, que retorna uma nova partição a partir de `p` ou ele próprio, caso contrário;
  - Esse processo é **repetido** até que **não haja mais necessidade de se particionar** – as **partições resultantes** serão os **estados minimizados** do **DFA**.

De expressões regulares a scanners

- **Algoritmo Split** (COOPER; TORCZON, 2014)

```
function Split(S)            ▷ S é um conjunto de conjuntos que define uma partição de estados
    for cada c ∈ Σ do
        if c separa S em s₁ e s₂ then
            return {s₁, s₂}
        end if
    end for
    return S                 ▷ se não houve separação retorna S original
end function
```

- O que significa “c separa um conjunto de estados S em s_1 e s_2”?
  - Se, para todos os estados em S e para um certo símbolo c, todos os próximos estados alcançáveis a partir de cada estado de S a partir desse símbolo estiverem em uma mesma partição, então o símbolo c **não** separa S;
  - Caso contrário, é criada uma nova partição $s1$ com os estados separados por c e outra partição $s_2 = S - s_1$ com os demais estados.

De expressões regulares a scanners

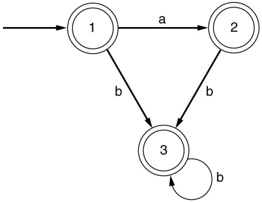
- **Algoritmo de Hopcroft** (COOPER; TORCZON, 2014)

$T \leftarrow \{D_A, D - D_A\}$ $\triangleright$ D_A contém os estados de aceitação; D contém todos os estados
$P \leftarrow \emptyset$
while $P \neq T$ **do**
    $P \leftarrow T$
    $T \leftarrow \emptyset$
    for cada conjunto $p \in P$ **do**
        $T \leftarrow T \cup Split(p)$
    end for
end while

De expressões regulares a scanners

- **Minimização do número de estados do DFA**
  - **Exemplo informal**

DFA original



Primeira divisão

A={1,2,3}

B={}

	1	2	3
a	A	ϕ	ϕ
b	A	A	A

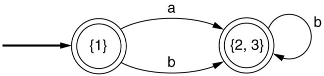
Segunda divisão

A={1}

B={2,3}

	A	B
a	B	ϕ
b	B	B

DFA minimizado



Referências bibliográficas

COOPER, K.; TORCZON, L. **Construindo compiladores**. 2. ed. Rio de Janeiro: Elsevier, 2014.

LOUDEN, K. **Compiladores: princípios e práticas**. [S.l.]: Pioneira Thomson Learning, 2004.