

– INF01147 –  
Compiladores

Análise Léxica  
AFND  $\rightarrow$  AFD  
F?LEX

Prof. Lucas M. Schnorr  
– Universidade Federal do Rio Grande do Sul –



# Revisão

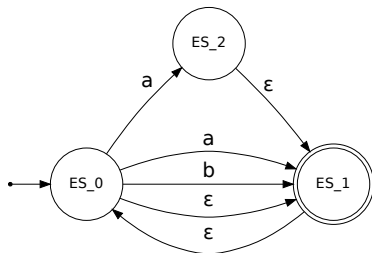
- ▶ Qual a principal função do analisador léxico?
- ▶ O que é um lexema, um padrão e um token?
- ▶ O que utilizamos para definir padrões?
- ▶ O que são AFNDs e AFDs? O que é uma transição vazia?
- ▶ Qual a relação de ambos com expressões regulares?
- ▶ Qual tipo de autômato é mais fácil implementar?
- ▶ Como saber se um lexema foi gerado por uma dada ER?

# Plano da Aula de Hoje

- ▶ Algoritmo para transformar um AFND em um AFD
- ▶ Descrição do Flex
- ▶ Especificação da Etapa 1 do Trabalho

# Problema dos AFNDs e Solução

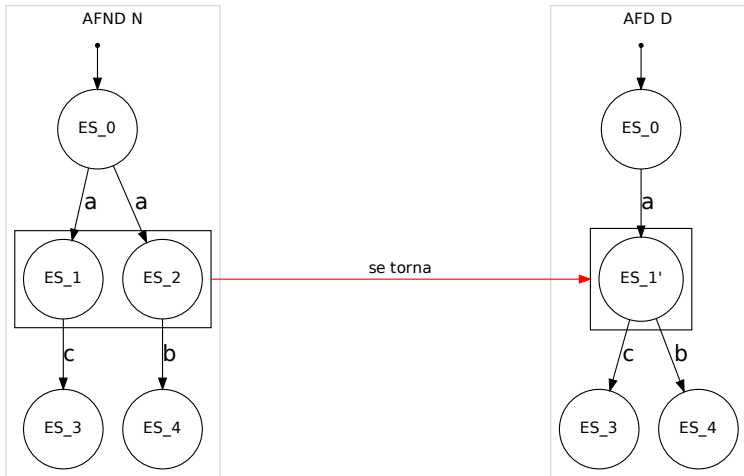
- ▶ Autômato Finito Não-Determinístico (AFND)
  - ▶ Bastante poderoso para implementar ERs
  - ▶ Trivial aplicação
    - ▶ Um AFND para cada definição regular
    - ▶ Combinação de todos os AFNDs com  $\epsilon$ -transições (Estados inicial e final únicos)
- ▶ Problema
  - ▶  $\epsilon$ -transições + múltiplas saídas com mesmo símbolo



- ▶ Fácil para a fase de projeto, difícil de implementar
- ▶ Método de transformação de um AFND em um AFD

# Construção de Subconjuntos – Visão Geral

- ▶ Considerando um AFND **N** e um AFD **D**
- ▶ **Idéia Geral**
  - ▶ Cada estado de D equivale a um conjunto de estados de N

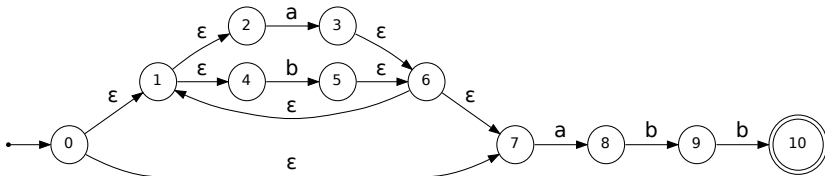


# Operações Fundamentais

- **Fechamento- $\epsilon$ (s)**

Conjunto de estados alcançados a partir do estado **s** utilizando somente transições  $\epsilon$

- Considerando o AFND para  $(a|b)^*abb$

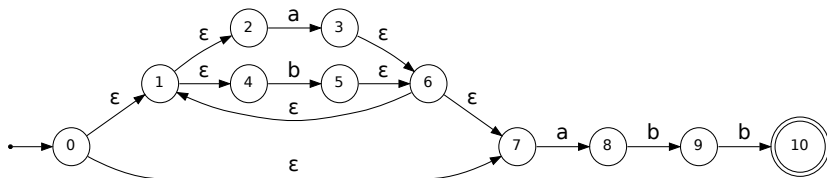


# Operações Fundamentais

## ► Fechamento- $\epsilon$ (T)

Conjunto de estados alcançados a partir de um conjunto de estados **T** utilizando somente transições  $\epsilon$

## ► Considerando o AFND para $(a|b)^*abb$



# Operações Fundamentais

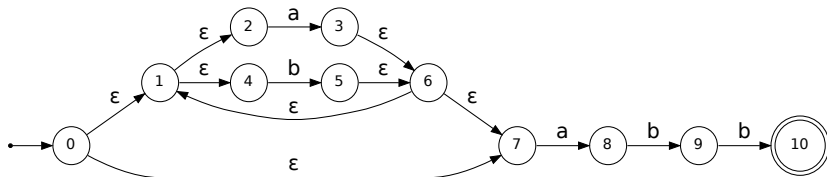
- **Movimento( $T, a$ )**

Conjunto de estados alcançados a partir de um conjunto de estados  $T$  utilizando o símbolo  $a$  na entrada

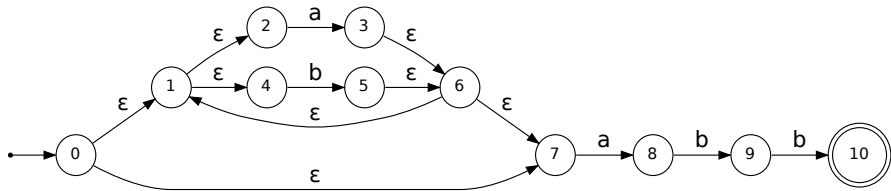
- **Fechamento- $\epsilon$ ( Movimento( $T, a$ ) )**

Depois de reconhecer o símbolo  $a$  na entrada, ainda é possível fazer transições consumindo o  $\epsilon$

- Considerando o AFND para  $(a|b)^*abb$

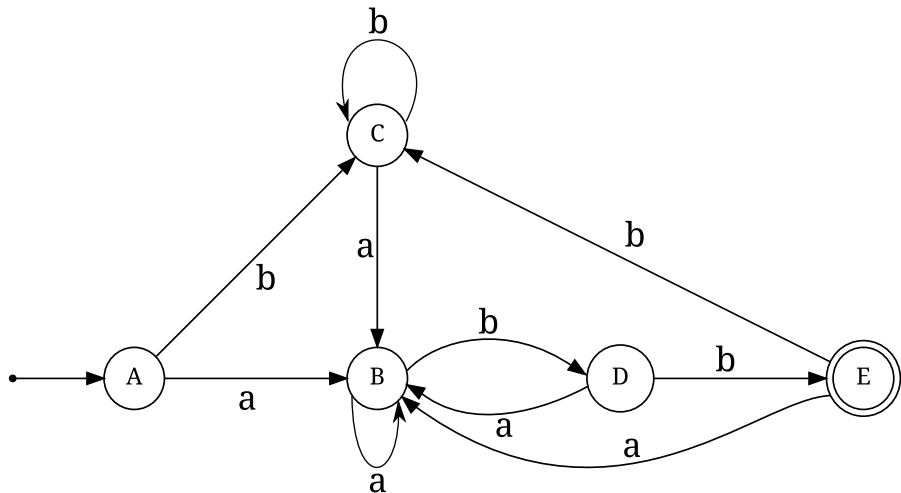


Exemplo a partir do AFND para  $(a|b)^*abb$



Estados do AFND	Estado AFD	a	b

AFD a partir do AFND para  $(a|b)^*abb$



F?LEX

# (F)LEX

- ▶ Lex: Ferramenta Unix para gerar analisadores léxicos
- ▶ **Flex**: Versão GNU (Fast Lex)  
<http://www.gnu.org/software/flex/>
- ▶ Funcionamento Geral
  - ▶ Especificação LEX (arquivo fonte com extensão .l)  
→ Declarações, Padrões, Regras para ações
  - ▶ Compilação LEX  
→ Obtenção de um código em C (lex.yy.c)
  - ▶ Compilação do programa C  
→ Obtenção de um analisador léxico

# Compilação Típica com Flex

- ▶ Edição da especificação em um arquivo – **regras.l**
- ▶ `flex regras.l`
  - ▶ Código em C no arquivo **lex.yy.c**
  - ▶ Exporta a função **yylex()** que retorna um token
  - ▶ `man flex`
- ▶ `gcc lex.yy.c -o analisador -lfl`
  - ▶ Compila e amarra o analisador com a biblioteca **libfl**
  - ▶ A função **main** deve estar implementada em algum lugar

# Especificação de Entrada Flex

- ▶ Contém três seções
  - ▶ Definições (em C, incluído no início da saída)
  - ▶ Regras (Expressões Regulares) e Ações (em C)
  - ▶ Código (em C, incluído no fim da saída)

- ▶ Sintaxe

Definições

%%

Regras e Ações

%%

Funções Opcionais

- ▶ Variáveis globais definidas por flex
  - ▶ yytext – lexema corrente
  - ▶ yyleng – seu tamanho

# Seção de Definições

- ▶ Opcional
- ▶ Código em C, incluído no início do arquivo C de saída
  - ▶ Declaração de tokens que serão reconhecidos
  - ▶ Diretivas de `include`
  - ▶ Variáveis globais necessárias ao analisador léxico
    - ▶ Contagem de linhas e colunas
  - ▶ Deve estar na forma `%{ ... %}`
- ▶ Contém também a declaração de ERs nomeadas

# Seção de Regras e Ações

- ▶ Obrigatória
- ▶ Cada linha desta seção é da seguinte forma  
`regra { ação }`
- ▶ Onde `regra` é uma expressão regular
  - ▶ Operadores de base (de Kleene)
  - ▶ E expressões regulares estendidas – `man grep`
  - ▶ Usa-se `{xxx}` para distinguir uma variável `xxx` da ER `'xxx'`
- ▶ E onde `ação` é um código escrito em C
  - ▶ `'{ }'` é a ação nula
    - ▶ Ou seja, o lexema é ignorado
    - ▶ Usado para detectar os brancos da entrada
  - ▶ `'{ printf ("oi"); }'` é a ação de imprimir oi na tela
  - ▶ Podem modificar variáveis globais da primeira seção

## Seção de Funções Opcionais

- ▶ Opcional
- ▶ Código em C, incluído no fim da saída
  - ▶ Definir funções para ações complexas da seção anterior
- ▶ Pode conter uma função `main`
- ▶ Impede a modularidade

# Alguns Detalhes

- ▶ Ação padrão → Copia para a saída o texto da entrada
  - ▶ Para evitar, pode-se utilizar uma regra . após todas as ERs
    - ▶ Pode ser visto como um erro léxico
- ▶ No caso de ambiguidade entre regras
  - ▶ Flex usa a regra que provê o maior lexema
- ▶ Regra **a/b**: reconhecer a somente se b aparece após
  - ▶ Se ab é encontrado
    - ▶ Reconhece a
    - ▶ b é mantido no buffer de entrada

# Conclusão

- ▶ Construindo um Analisador Léxico



- ▶ Autômatos intermediários nem sempre são os menores
  - ▶ Mais detalhes na seção 3.9 do livro do dragão
- ▶ Pode-se fazer tudo a mão, várias possibilidades
  - ▶ Gerador de AFND a partir de ER
  - ▶ Implementar um transformador de AFND em AFD
  - ▶ Construir um AFD diretamente

# Especificação da Etapa 1

# Conclusão

- ▶ Leituras Recomendadas

- ▶ Livro do Dragão, seção 3.7.1
- ▶ Livro “Lex & Yacc”
- ▶ <http://dinosaur.compilertools.net>  
Toda a turma: Lex | Yacc | Flex | Bison
- ▶ Git!  
<http://git-scm.com/book/>

- ▶ Próxima Aula

Análise Sintática