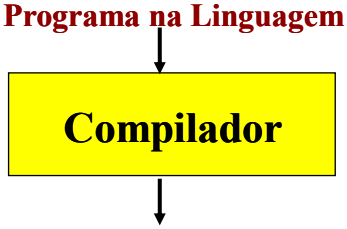
O Modelo Análise-Síntese

Rafael Dueire Lins

Será apresentado um modelo amplamente utilizado para explicar compiladores

Compilador

Programa na Linguagem Fonte

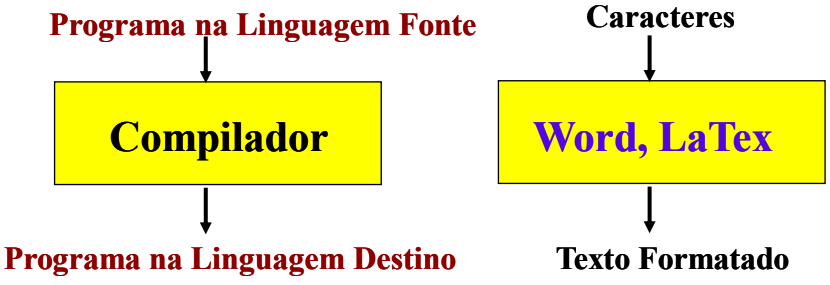


Programa na Linguagem Destino

Compilador

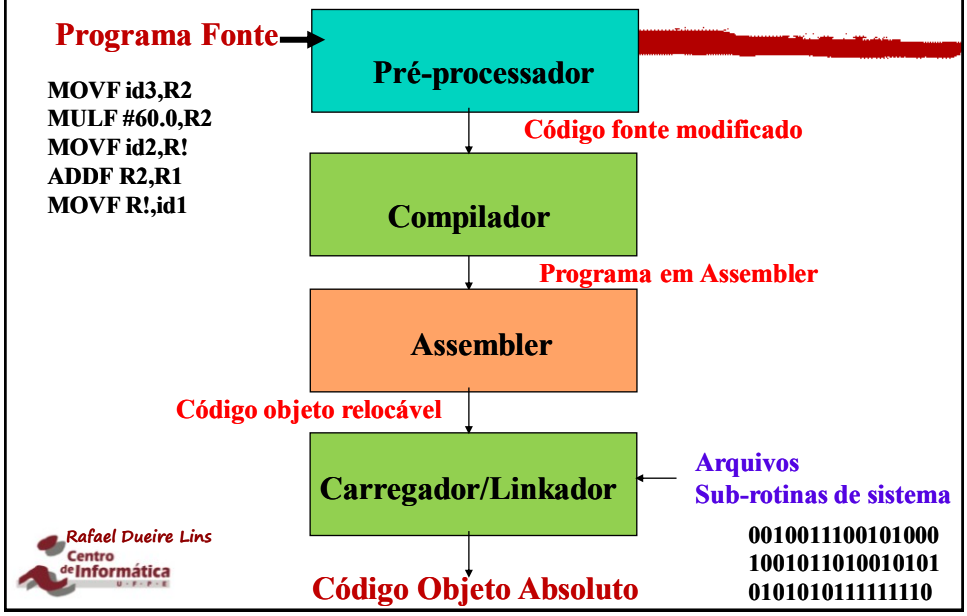
Rafael Dueire Lins
Centro de Informática
UFPE

Compilador

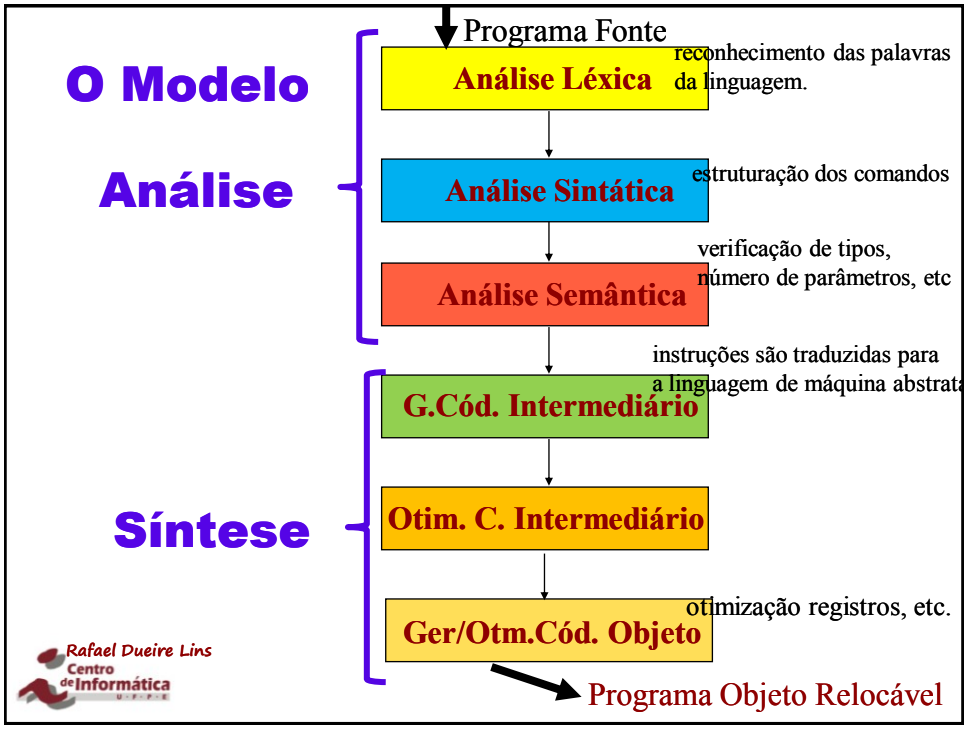


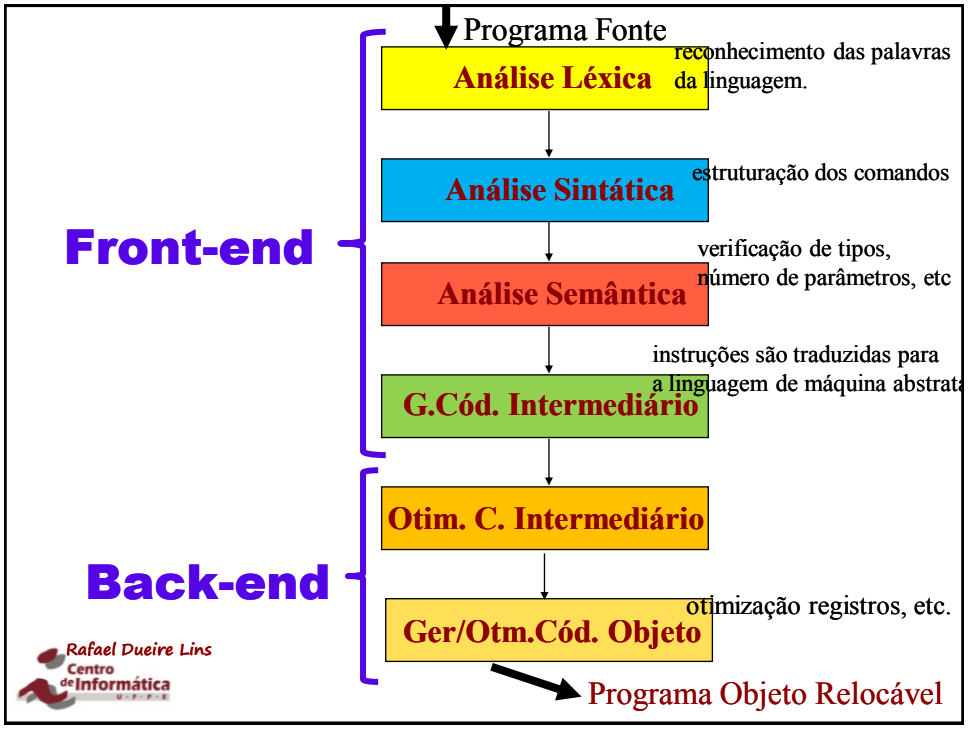
Rafael Dueire Lins
Centro de Informática
UFPE

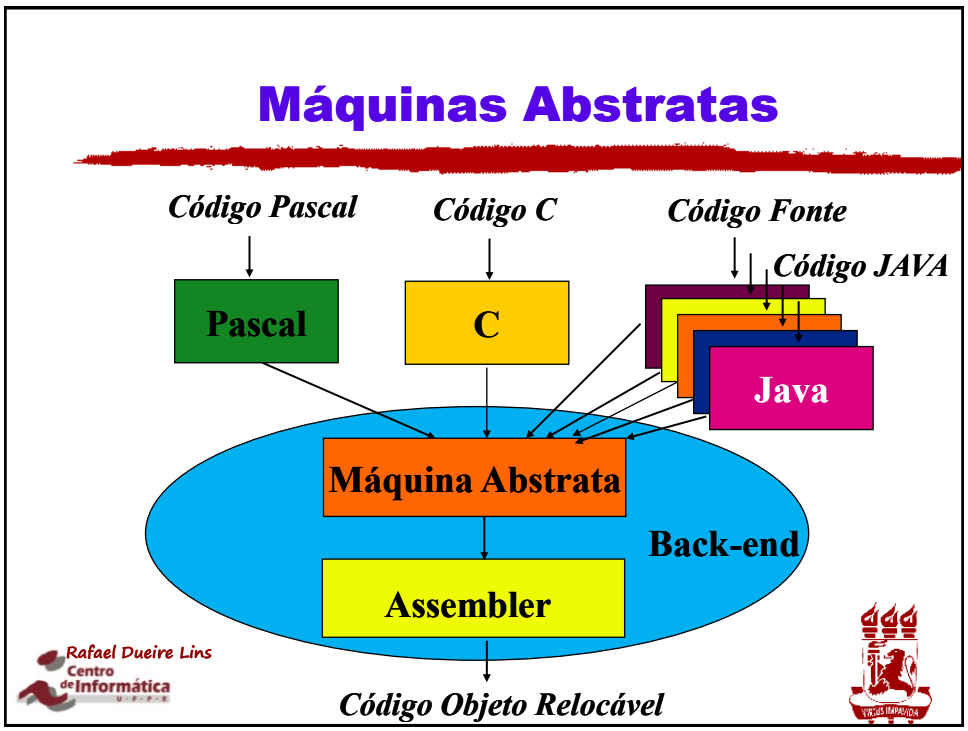
Ambiente Processador de Linguagem



O Modelo



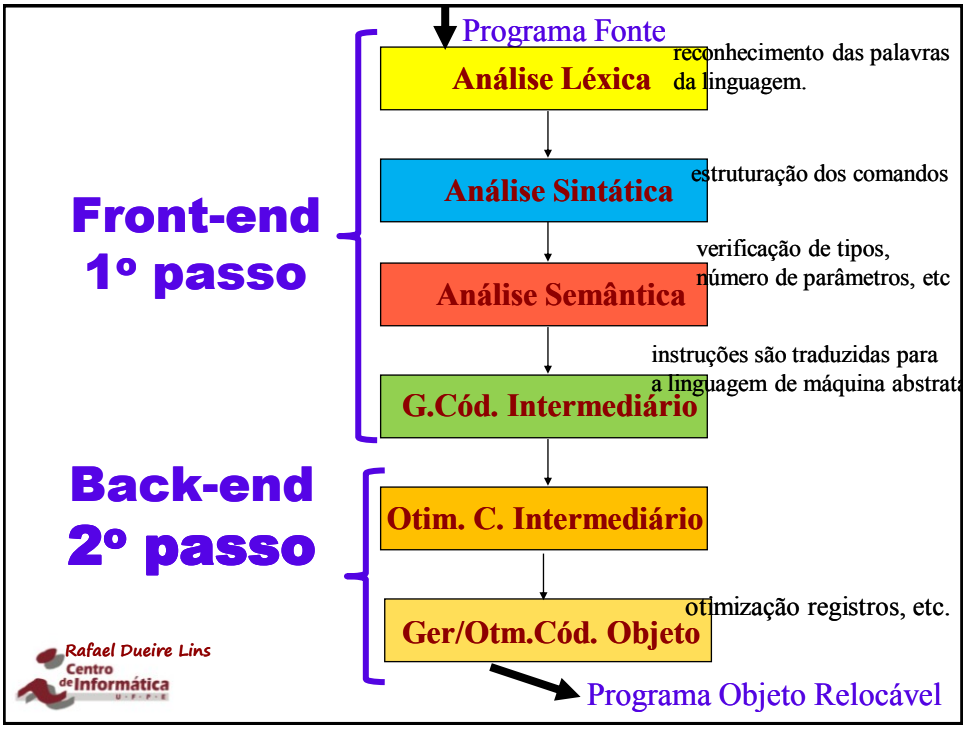

Programa Fonte
Análise Léxica
reconhecimento das palavras da linguagem.
Análise Sintática
estruturação dos comandos
Front-end
Análise Semântica
verificação de tipos, número de parâmetros, etc
G.Cód. Intermediário
instruções são traduzidas para a linguagem de máquina abstrata
Otim. C. Intermediário
Back-end
Ger/Otm.Cód. Objeto
otimização registros, etc.
Programa Objeto Relocável
Rafael Dueire Lins
Centro de Informática UFPE


Máquinas Abstratas
Código Pascal
Código C
Código Fonte
Código JAVA
Pascal
C
Java
Máquina Abstrata
Back-end
Assembler
Código Objeto Relocável
Rafael Dueire Lins
Centro de Informática UFPE

Passo de Compilação

- Cada saída de um compilador é dito um passo de compilação.
- Quanto mais passos de compilação mais fácil manter e alterar o compilador.
- Quanto menos passos, geralmente mais eficiente o compilador em termos de tempo e espaço de compilação.
- Não necessariamente melhor código!!

Rafael Dueire Lins
Centro de Informática UFPE



Rafael Dueire Lins
Centro de Informática UFPE

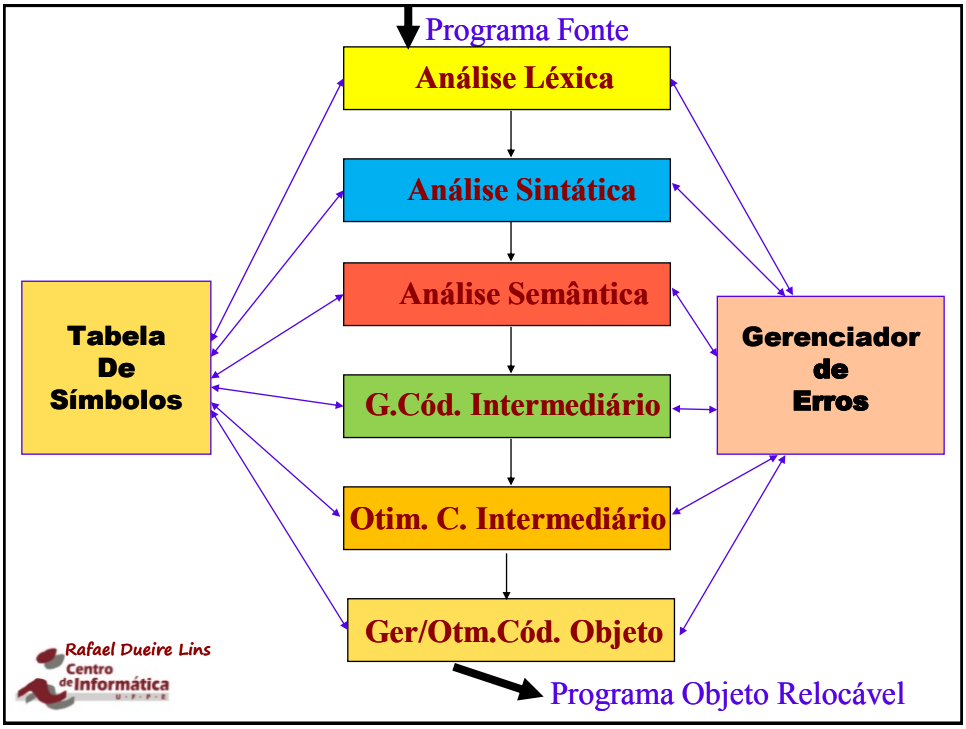
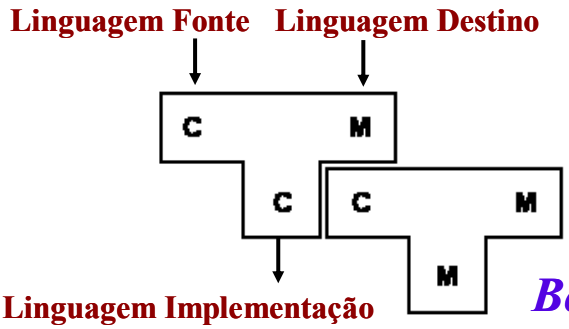


Diagrama T



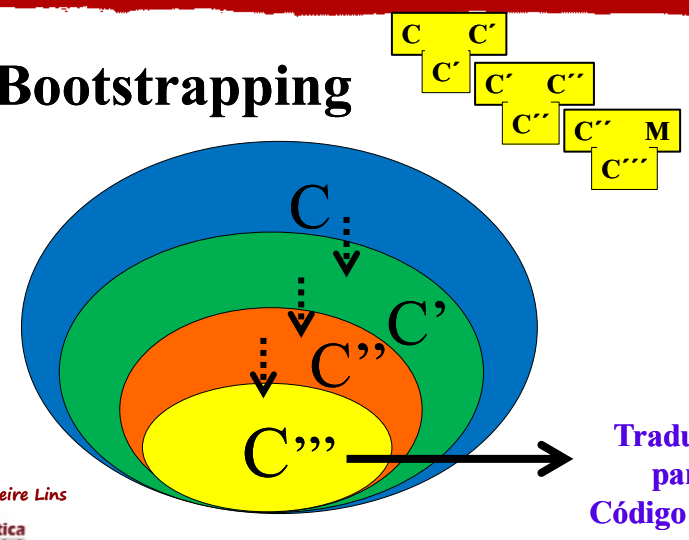
Bootstrapping

O código gerado por *bootstrapping* pode ser mais eficiente que o manual!!!

Rafael Dueire Lins
Centro de Informática

A Linguagem C

Bootstrapping



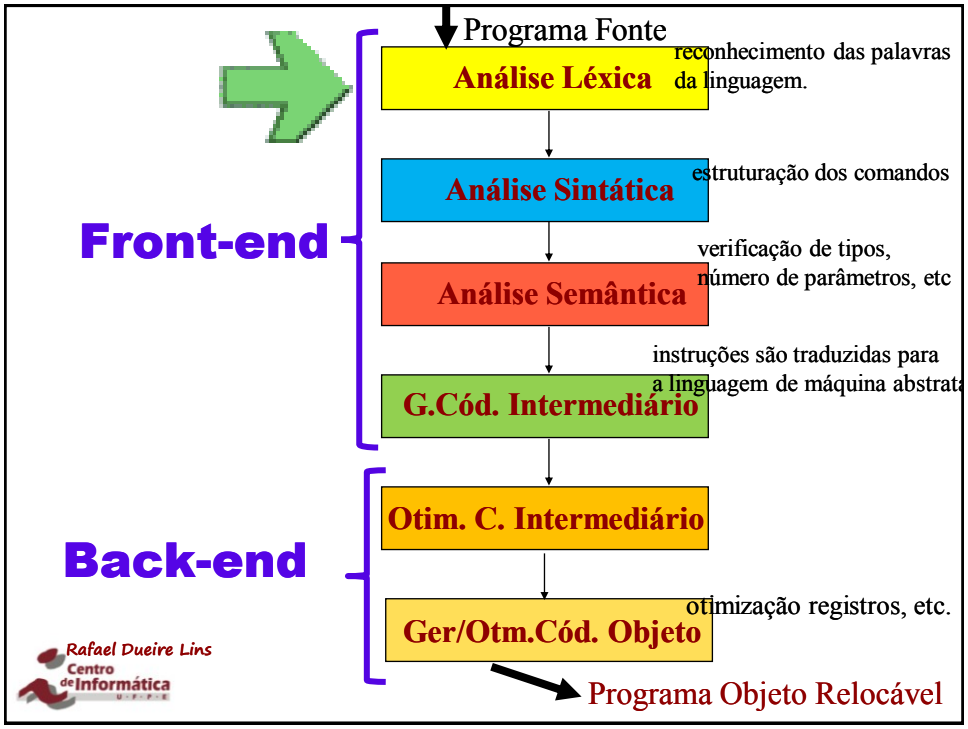
Rafael Dueire Lins
Centro de Informática UFPE

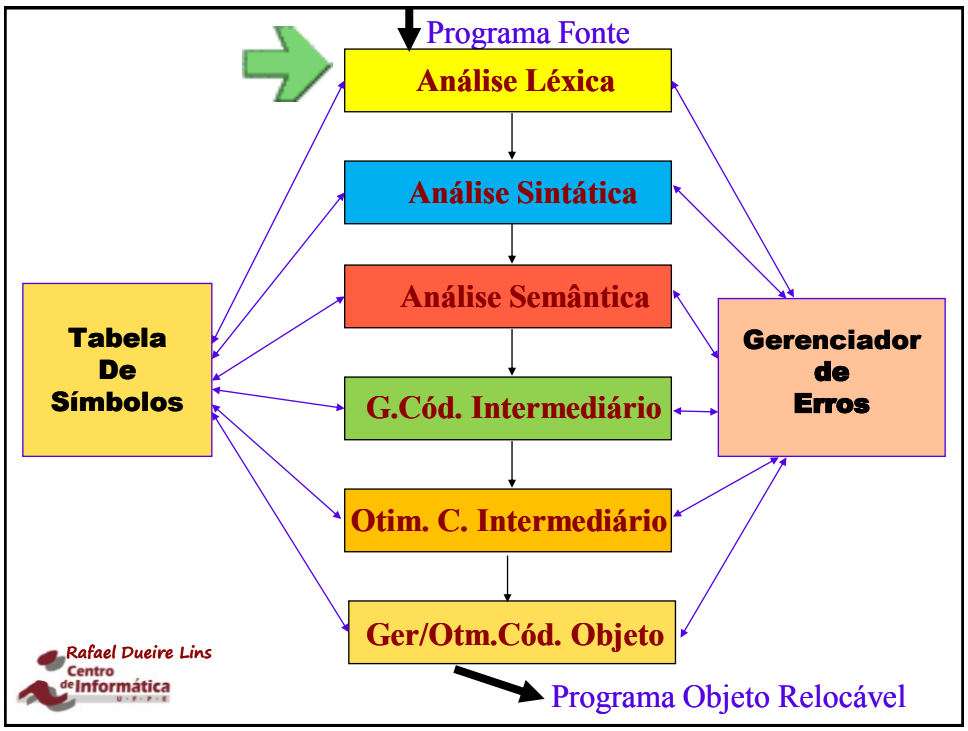
Análise Léxica

Rafael Dueire Lins

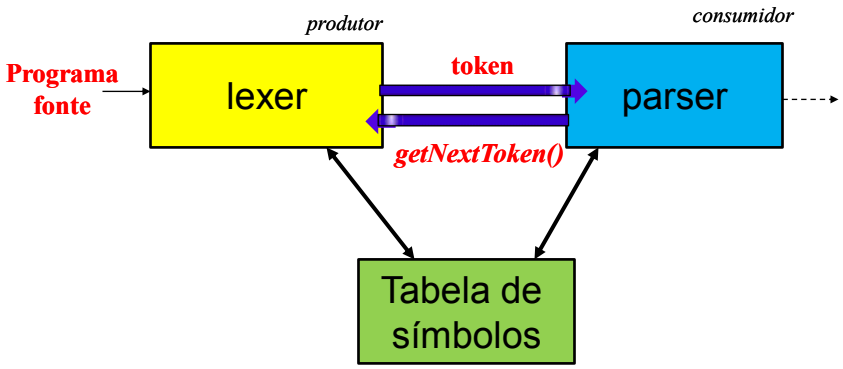
A análise léxica ou scanning é a primeira fase da compilação.

Rafael Dueire Lins
Centro de Informática UFPE


Programa Fonte
Análise Léxica
reconhecimento das palavras da linguagem.
Análise Sintática
estruturação dos comandos
Front-end
Análise Semântica
verificação de tipos, número de parâmetros, etc
G.Cód. Intermediário
instruções são traduzidas para a linguagem de máquina abstrata
Otim. C. Intermediário
Back-end
Ger/Otm.Cód. Objeto
otimização registros, etc.
Programa Objeto Relocável
Rafael Dueire Lins
Centro de Informática


Programa Fonte
Análise Léxica
Análise Sintática
Análise Semântica
Tabela De Símbolos
Gerenciador de Erros
G.Cód. Intermediário
Otim. C. Intermediário
Ger/Otm.Cód. Objeto
Programa Objeto Relocável
Rafael Dueire Lins
Centro de Informática

Analisador Léxico ou *Scanner*



Separação *lexer - parser*

- **Simplicidade:** misturar análise léxica com sintática torna o *parser* mais complicado
- **Eficiência:** separação possibilita construir processadores léxicos e sintáticos mais eficientes

Tokens, Padrões e Lexemas

- ***Tokens***: par com o nome do token e atributos opcionais.
- ***Padrão***: descrição dos possíveis lexemas associados a um tipo de token.
- ***Lexeme***: sequência de caracteres que casam com o padrão de um tipo de token.

Observações

- Existem conjuntos de strings na entrada que geram o mesmo tipo de token:
  aluno8, aluno51, var1, xyz2 são tokens do mesmo tipo ID
- Símbolos terminais de uma gramática correspondem a tokens:
  - Palavras reservadas,
  - operadores,
  - identificadores,
  - constantes,
  - parênteses, etc.

Análise Léxica

Look-ahead token

E = M * C ** 2 → = M * C ** 2

Tabela de Símbolos		
E	identificador	

Rafael Dueire Lins
Centro de Informática UFPE

Análise Léxica

Look-ahead token

E = M * C ** 2 → M * C ** 2

Tabela de Símbolos		
E	identificador	
=	Símbolo de atribuição	

Rafael Dueire Lins
Centro de Informática UFPE

Análise Léxica

Look-ahead token

E = M * C ** 2 → * C ** 2

Tabela de Símbolos		
E	identificador	
=	Símbolo de atribuição	
M	identificador	

Análise Léxica

Look-ahead token

E = M * C ** 2 → C ** 2

Tabela de Símbolos		
E	identificador	
=	Símbolo de atribuição	
M	identificador	
*****	mult_op	

Análise Léxica

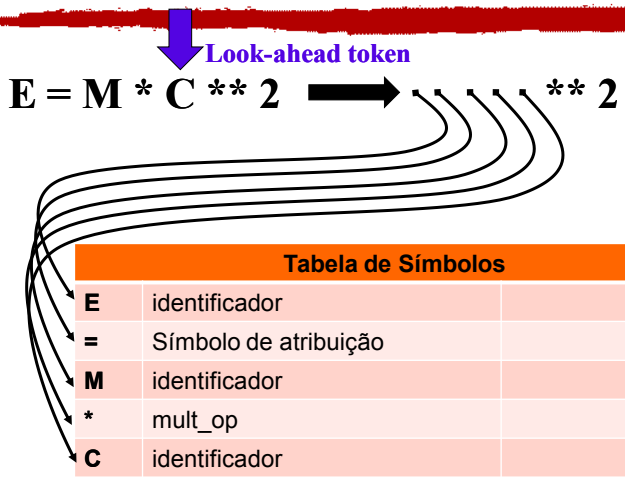


Tabela de Símbolos		
E	identificador	
=	Símbolo de atribuição	
M	identificador	
*****	mult_op	
C	identificador	

Rafael Dueire Lins
Centro de Informática UFPE

Análise Léxica

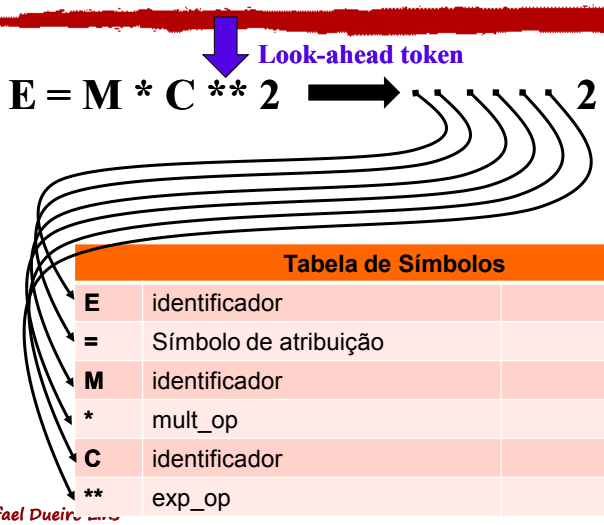


Tabela de Símbolos		
E	identificador	
=	Símbolo de atribuição	
M	identificador	
*****	mult_op	
C	identificador	
**	exp_op	

Rafael Dueire Lins
Centro de Informática UFPE

Análise Léxica

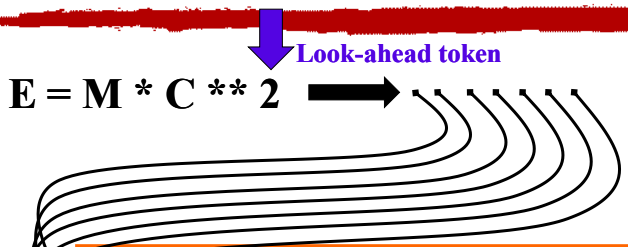


Tabela de Símbolos		
E	identificador	
=	Símbolo de atribuição	
M	identificador	
*****	mult_op	
C	identificador	
******	exp_op	
2	integer	

Expressões Regulares Notação

Usada na especificação dos Tokens

***** Zero ou mais instâncias

+ Uma ou mais instâncias

? Zero ou uma instância

[…] classes de caracteres [A-Za-z][A-Za-z0-9]*

A | B Instância de **A** ou instância de **B**

AB Instância de A seguida de instância de B

Pascal

- Identificadores: letter (letter | digit) *
- Tokens:

letter = [A- Za-z]
digit = [0-9]
id = **letter** (**letter** | **digit**)*
digits = **digit digit***
opt_fraction = . **digits** | ε
opt_exponent = (E (+|-|ε) **digits**) | ε
num = **digits opt_fraction opt_exponent**

Propriedades Algébricas

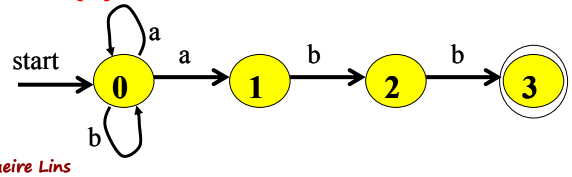
- r|s = s|r comutatividade
- r|(s|t) = (r|s)|t associatividade
- (rs)t = r(st) associatividade
- r(s|t) = rs|rt distribitividade
- (s|t)r= sr|tr distributividade
- εr = rε = r identidade
- r*=(r|ε)* relação entre r e ε
- r** = r* idempotência

Automata Finitos Não-determinísticos

- Modelo Matemático que consiste em:
  1. S – o conjunto de estados
  2. Σ – um conjunto de símbolos de entrada
  3. *Move* : estado X símbolo→S
  4. S_0 – o estado inicial
  5. F – o conjunto de estados finais (ou aceitos)
- Um NFA pode ser representado por um grafo direcionado ou de transição.

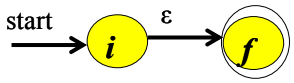
Automata Finitos Não-determinísticos

- (a|b)*abb é reconhecida pelo NFA:
  1. S ={0, 1, 2, 3}
  2. Σ = {a, b}
  3. (0,a)→0, (0,b)→0, (0,a)→1, (1,b)→2, (2,b)→3
  4. S_0 = 0
  5. F = {3}

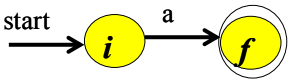


Construção de Thompson
ER → NFA

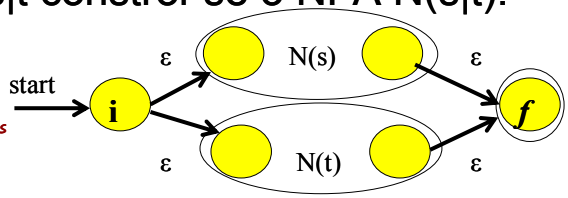
- ε é representada pelo NFA:



- Para a em Σ:



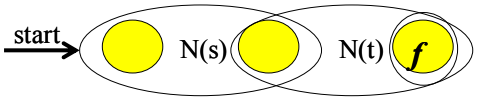
- Para s|t constrói-se o NFA N(s|t):



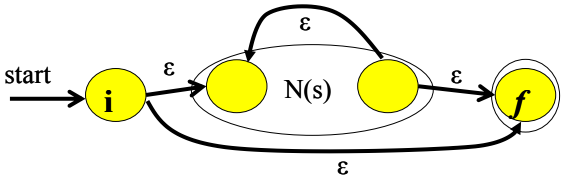
Rafael Dueire Lins
Centro de Informática
UFPE

Construção de Thompson
ER → NFA

- st é representada pelo NFA:



- Para s* temos o NFA:



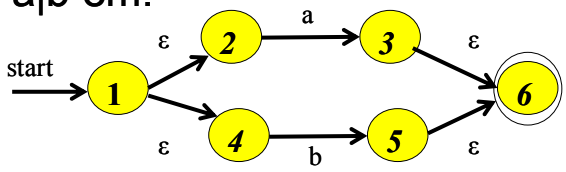
- Para (s), usa-se N(s) como NFA.

Rafael Dueire Lins
Centro de Informática
UFPE

Construção de Thompson
(a|b)*abb → NFA

- a e b são representadas pelo NFAs:

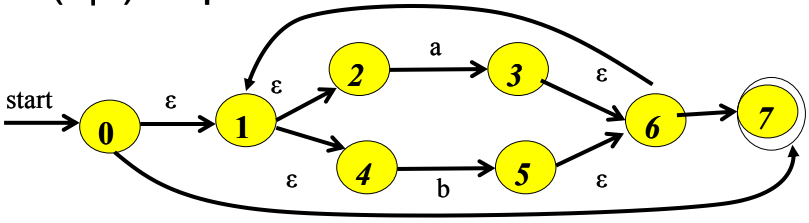
- Para a|b em:



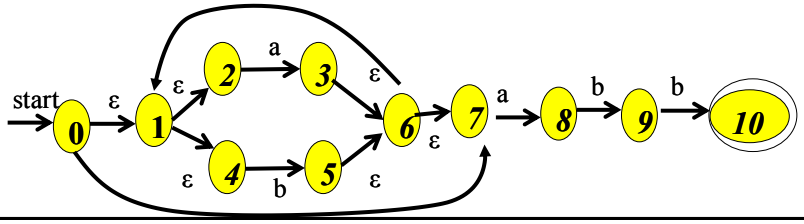
Rafael Dueire Lins
Centro de Informática UFPE

Construção de Thompson
(a|b)*abb → NFA

- (a|b)* equivale ao NFA:

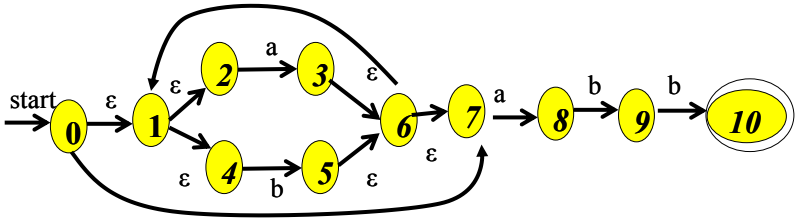


- Para (a|b)*abb em:

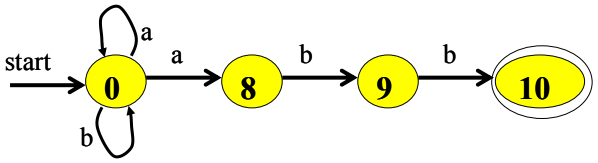


Simplificação do NFA (a|b)*abb

- NFA:



- Pode ser simplificado para:



eliminando as transições vazias.

NFAs e DFAs

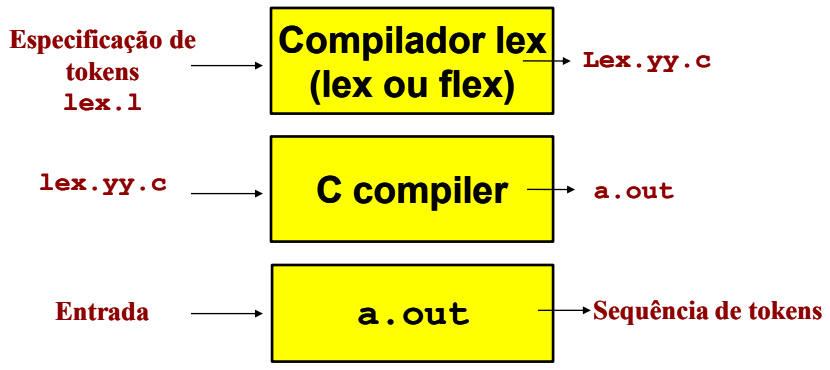
- Todo NFA pode ser transformado em um DFA através da inclusão de novos estados.
- Todo DFA é isomorfo a um Anel.
- Todo Anel pode ser minimizado num Ideal maximal.
- O método de Quine-McCluskey mostra como minimizar um DFA.

Implementação do lexer

- Expressões regulares (ER) como entrada.
- Ferramentas geram autômatos reconhecedores para ERs.
- Exemplos: Lex, Flex, JLex, Alex, etc.

Rafael Dueire Lins
Centro de Informática UFPE

Lex (para C)



Rafael Dueire Lins
Centro de Informática UFPE

Descrição em Lex - estrutura

Declarações – variáveis, constantes, defs.regulares

%%

regras de tradução – expr. regulares e ações em C

Padrão { ação }

%%

procedimentos auxiliares

Rafael Dueire Lins
Centro de Informática UFPE

Descrição em Lex - exemplo

```
%% declarações
%%

delim       [ \t\n]
ws          {delim}+
letter      [A-Za-z]
digit       [0-9]
id          {letter}({letter}|{digit})*
number      {digit}+(\.{digit}+)?
                    (E[+\-]?{digit}+)?
…
```

Rafael Dueire Lins
Centro de Informática UFPE

Descrição em Lex – exemplo

```
%% regras de tradução
%%

{ws}        {/* no action and no return */}
if          {return(IF);}
then        {return(THEN);}
else        {return(ELSE);}
{id}        {yylval=install_id();return(ID);}
{number}    {yylval=install_num();
             return(NUMBER);}
"<"         {yylval = LT; return(RELOP);}
"<="        {yylval = LE; return(RELOP);}
…
```


Descrição em Lex – exemplo

```
%% funções auxiliares
%%

int install_id() {
  Copia lexeme para a tabela de símbolos.
  Primeiro caracter do lexeme é apontado
  pela variável yytext e o comprimento é
  definido pela variável yylength.
}

int install_num() { … }
```


Vantagens do Uso do Lex

- Programação de alto nível.
- Portabilidade.
- Mantainabilidade.
- Geração de autômato finito determinístico mínimo!
- Eficiência!!!!