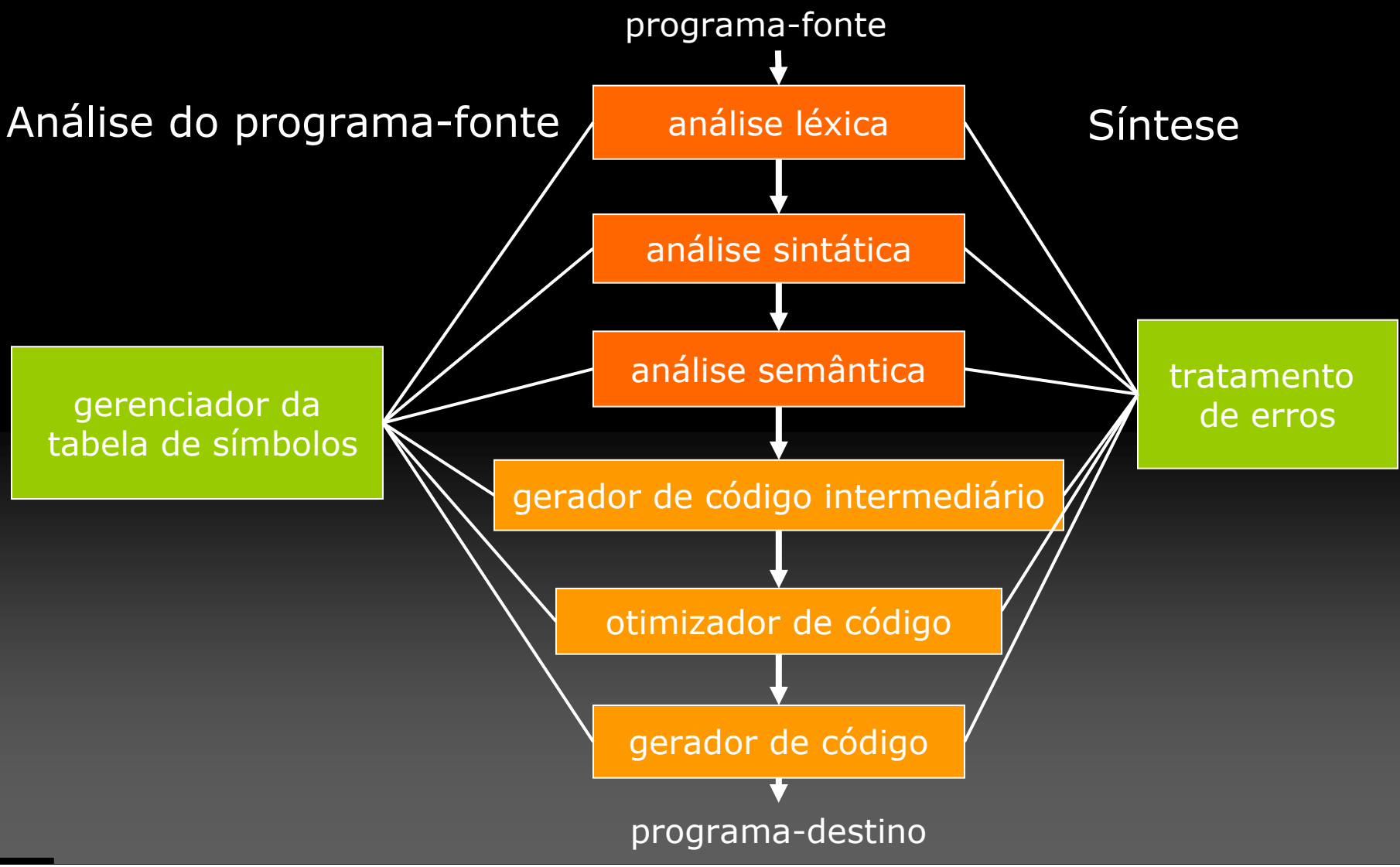
Projeto de Compiladores

FIR – Faculdade Integrada do Recife
João Ferreira
joaoferreira@fir.br
5 e 6 de março de 2007

Agenda da Aula

- Resposta exercício da aula passada
- Revisão
  - As Fases de Um Compilador
- Analisador Léxico
  - Scanner e Parser
- Especificação de *Tokens*
- Reconhecimento de *Tokens*

As Fases de Um Compilador



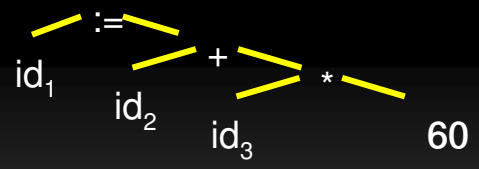
Exemplo Completo

position := initial + rate * 60

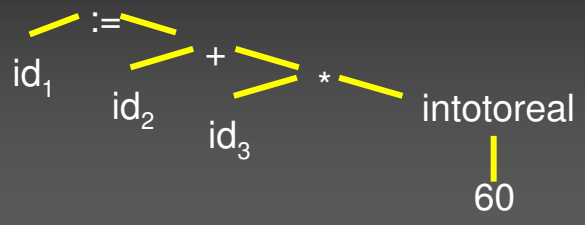
análise léxica

id1 := id2 + id3 * 60

análise sintática



análise semântica



gerador de código intermediário

temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3

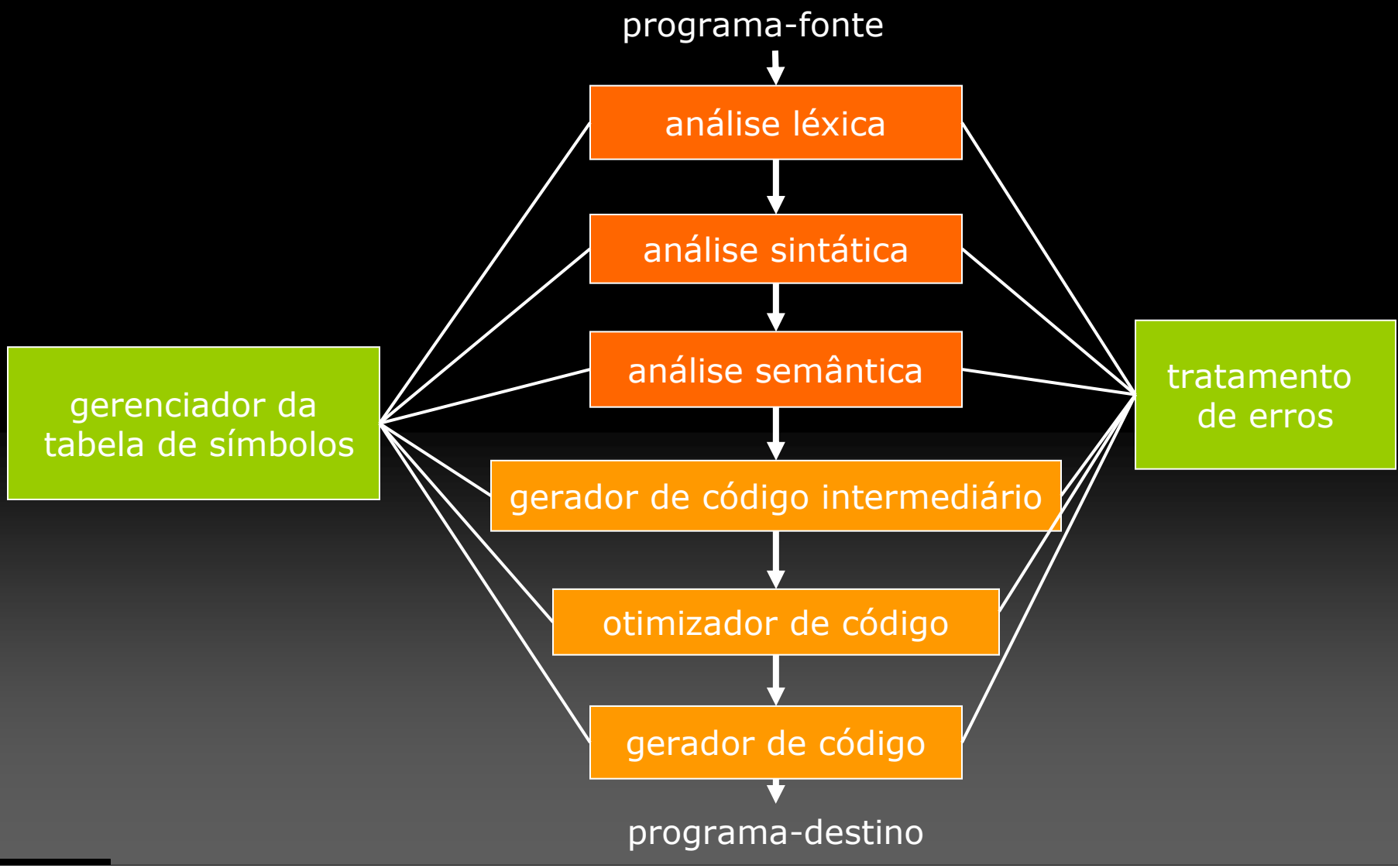
otimizador de código

temp1 := id3 * 60.0
id1 := id2 + temp1

gerador de código

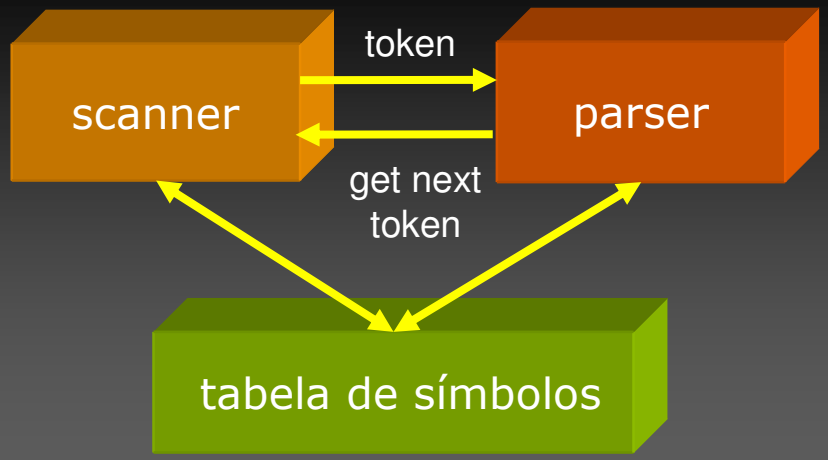
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1

Onde Estamos?



Análise Léxica

- Leitura de caracteres para formação de *tokens* que serão utilizados pela Análise Sintática
- Geralmente é implementado como uma subrotina (“get next token”) do *parser*



Por que separar *Scanner* e *Parser*?

- Torna o projeto e implementação mais simples
  - Ex: Remoção de espaços e comentários pelo Analisador Léxico, torna mais simples o *Parser*
- Aumento de performance através da especialização de atividades
  - *Buffering*
- Aumento da portabilidade através do isolamento da camada de entrada de dados

Tokens, Padrões e Lexemas

- *Token* é uma unidade léxica (ex. identificador, palavra reservada, operador)
- Padrão é um conjunto de regras para formação de um *token*
- Lexema é uma seqüência de caracteres que "casa" com um padrão

Exemplo

Token	Lexema	Padrão
if	if	if
relacao	<, <=, =, <>, >, >=	< **ou** <= **ou** = **ou** <> **ou** > **ou** >=
id	pi,count, D2	letra seguida por letra ou dígitos
num	3.1416, 0, 15	constante numérica

Slide 9

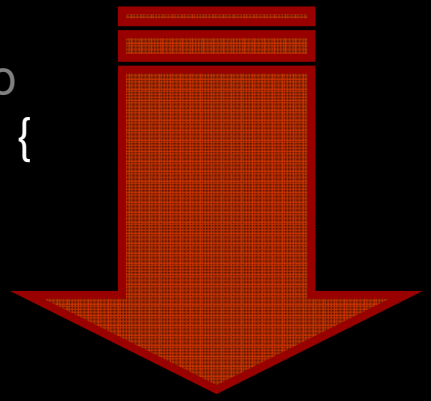
IFS2 Verificar se é necessário falar de linguagens como FORTRAN que dificultam a definição de padrões justamente por exigir que certos comandos sejam aplicados em determinadas colunas

Verificar se é importante colocar o exemplo da página 86 do dragon book que explica o problema de não se ignorar espaços em branco

In Forma Software; 23/2/2007

Exemplo

```
class Programa {
        //comentario
        void teste()  {
        }
}
```



CLASS IDEN(“Programa”) LBRACE VOID IDEN(“teste”)
LPAREN RPAREN LBRACE RBRACE RBRACE

Atributos dos *Tokens*

- É importante saber que lexema “casou” com o padrão
- O analisador deve coletar informações influenciarão a tradução do *token*
  - <id, ponteiro para “rate” na tabela de símbolos>
  - <op_atribuicao, >
  - <num, valor real 60.0>
- Nem todos os atributos são obrigatórios

Erros Léxicos

- Poucos erros podem ser verificados pelo analisador devido a sua visão localizada do código
  - Ex: em fi (a == f(x)) O analisador léxico entende “fi” como um identificador e não como um erro

Erros Léxicos

- Técnicas de recuperação de erros
  - "Panic Mode" – Remoção de caracteres até que um *token* válido seja encontrado
  - Remoção do caractere estranho
  - Inserção do caractere ausente
  - Substituição do caractere incorreto
  - Transposição dos caracteres adjacentes
- A maioria das técnicas pressupõem que apenas um caractere está errado

Buferização de Entrada

- Análise Léxica é a única fase do compilador que lê o código-fonte, por isso deve-se investir tempo no seu projeto

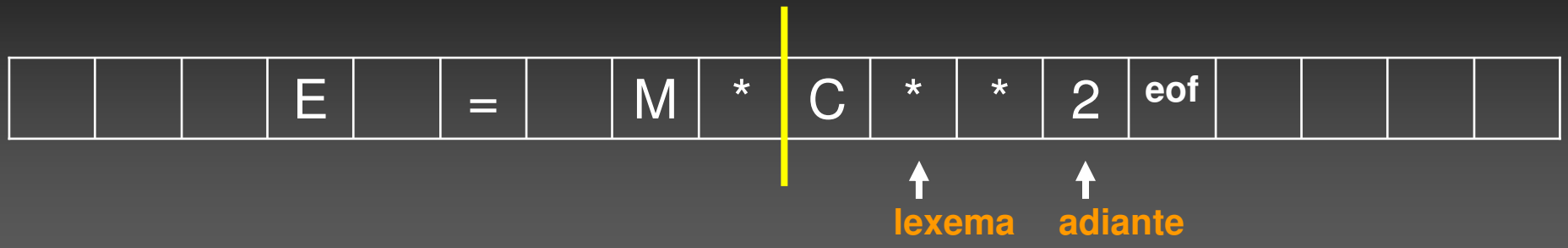
- Abordagens (Dificuldade X Eficiência)
  - Utilizar um Gerador de Analisadores Léxicos
    - Lex
    - GALS
  - Escrever um Analisador Léxico usando I/O
  - Escrever um Analisador em Assembly gerenciando explicitamente a leitura

Pares de *Buffers*

- Funcionamento
  - Divide o buffer em 2 metades de N-caracteres
  - Realiza a leitura e armazenamento em cada uma das metades
  - Se um número menor que N for lido na segunda metade adiciona um **eof**
  - Dois apontadores são utilizados
    - lexema
    - adiante

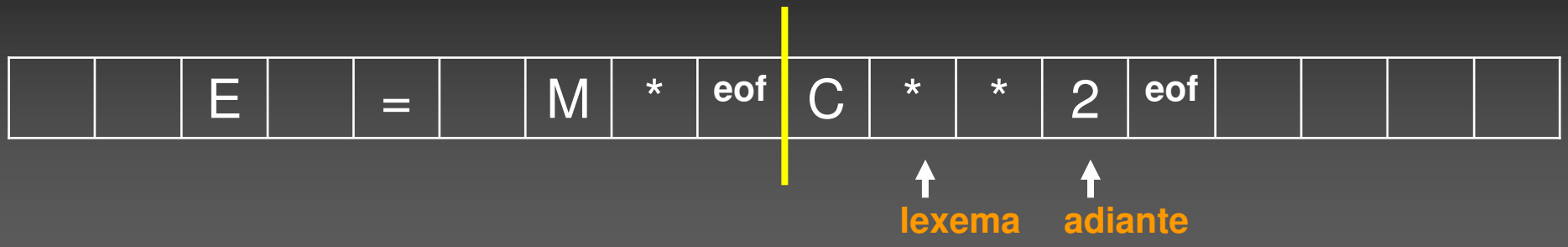
Pares de *Buffers*

- Funcionamento (cont.)
  - A *string* de caracteres entre os dois apontadores é o lexema
  - Comentários e espaços em branco são padrões que não produzem *tokens*
  - Se o apontador adiante chega à metade, a metade direita recebe um novo conjunto de N caracteres
  - Se o apontador adiante chega ao final do buffer, a metade esquerda recebe um novo conjunto de N caracteres



Sentinelas

- A técnica anterior precisa verificar a cada deslocamento dos apontadores se os limites do *buffer* foram ultrapassados
- O uso se sentinelas (**eof**) ao final de cada metade do *buffer* facilita a identificação dos limites



Especificação de *Tokens*

- Cadeias e Linguagens
- Operações em Linguagens
- Expressões Regulares
- Definições Regulares
- Simplificações Notacionais
- Conjuntos Não-regulares

jz10

Cadeias e Linguagens

- Alfabeto é um conjunto finito de símbolos (letras, números, ...)
  - Alfabeto binário = {0,1}
  - Vogais = {a,e,i,o,u}
- Palavra é uma seqüência/cadeia finita de caracteres de um alfabeto
  - Palavra do “alfabeto” binário ex: 101
  - Palavra do “alfabeto” vogais ex: oi, ei, ao
- O comprimento da cadeia é denotado por |s|, onde s é o número de ocorrências

Slide 19

jz10 Adicionar a figura 3.7 e seu conteúdo teórico
jz; 28/2/2007

Cadeias e Linguagens

- A cadeia vazia é representada por ε
- Linguagem é um conjunto de cadeias sobre um alfabeto e pode ser {ε} ou ø
- O significado das palavras será discutido mais adiante
- Concatenação é a união em seqüência de duas ou mais palavras
  - X = laranja, Y = cravo, XY = laranjacravo
- Se a concatenação pode ser tratada como um produto, temos exponenciação em
  - $s^0 = \varepsilon$ e $\varepsilon s = s$
  - Para $i>0$, $s^i = s^{i-1}s$
    - $s^1 = s$
    - $s^2 = ss$

jz11

Operações em Linguagens

- Assim como podemos realizar operações com palavras podemos realizar operações com linguagens

- União
  - L U M = { s | s está em L ou s está em M }
- Concatenação
  - LM = { st | s está em L e t está em M }
- Fechamento *Kleene*
  - L^* (zero ou mais)
- Fechamento Positivo
  - L^+ (um ou mais)

Slide 21

jz11 Juntar as definições e exemplos em apenas um slide
Melhorar isso. Talvez deixar mais claro antes de entrar em detalhes qual a real necessidade de se aprender esse conteúdo
jz; 28/2/2007

Operações em Linguagens

- Exemplo: Sejam L = {A..Z, a..z} e D = {0..9} linguagens finitas. Podem ser derivadas
  - L U D = conjunto de letras e dígitos
  - LD = conjunto de cadeias de uma letra seguida por um dígito
  - L^4 = conjunto de cadeias com quatro letras
  - L^* = conjunto de todas as cadeias de letras, ε inclusive
  - $L(L \cup D)^*$ = conjunto de cadeias de letras e dígitos que começam com uma letra
  - D^+ = conjunto de cadeias de um ou mais dígitos

jz12

Expressões Regulares

- Permite definir precisamente conjuntos/padrões
- Exemplo: Pascal identificador é um letra seguida por zero ou mais letras ou dígitos
  - letra (letra | dígito)*
- Uma expressão regular é formada por expressões regulares mais simples usando um conjunto de regras de definição
- A expressão r representa um linguagem L(r)

Slide 23

jz12 Aqui é bom fazer um gancho e mostrar outras aplicações e expressões regulares, se possível levar um exemplo (escrito em java) de como utilizar expressões regulares
jz; 28/2/2007

jz13

Expressões Regulares: Algumas Regras

1. ε denota um conjunto com uma cadeia vazia
2. Se a é um símbolo em ∑, a denota {a}
3. Se r e s são expressões regulares de L(r) e L(s)
   a. (r) | (s) = L(r) U L(s)
   b. (r)(s) = L(r)L(s)
   c. $(r)^* = (L(r))^*$
   d. (r) = L(r)

Slide 24

jz13 Aqui é bom fazer um gancho e mostrar outras aplicações e expressões regulares, se possível levar um exemplo (escrito em java) de como utilizar expressões regulares
jz; 28/2/2007

Expressões Regulares: Precedência

- Regras de precedência podem ser adotadas para reduzir o número de parênteses
  - O * tem maior precedência e é associativo à esquerda
  - A Concatenação tem a segunda maior precedência e é associativa à esquerda
  - O | possui a menor precedência e é associativo à esquerda
  - (a) | ((b)*(c)) = a | b*c

Expressões Regulares: Exemplos

- Seja $\sum = \{a,b\}$
  - $a \mid b = \{a,b\}$
  - $a^* = ?$
  - $(a \mid b)^* = ?$
  - $a \mid a^*b = ?$
  - $(a \mid b)(a \mid b) = ?$

Propriedades Algébricas das Expressões Regulares

Axioma	Descrição

Definições Regulares

- Podemos nomear as expressões regulares para simplificar referências
  - $d_1 \rightarrow r_1$
  - $d_n \rightarrow r_n$
- Exemplo, Identificador em Pascal
  - letra → A | .. | Z | a | .. | z
  - dígito → 0 | .. | 9
  - id → letra (letra | dígito)*

Definições Regulares

- Exemplo Números (sem sinal) em Pascal
  - dígito → 0 | .. | 9
  - dígitos → dígito dígito*
  - op_fração → . dígitos | ε
  - op_exp → (E | (+ | - | ε) dígitos) | ε
  - num → dígitos op_fração op_exp
- A expressão regular “num” é capaz de representar
  - 5.37 ? sim
  - 3.06E-2 ? sim
  - 1.0 ? sim
  - 1. ? não
  - 1? sim

IFS6 procurar uma definição melhor
Como mostrar exemplos da utilização ou padrões que casam ex: 1.0, 5.35 E-4
In Forma Software; 1/3/2007

Simplificações Notacionais

- Servem como atalhos para simplificar a escrita de uma expressão regular
- + - Uma ou mais instâncias
- ? - Zero ou uma instância
- * - Zero ou mais instâncias
- [ ] - Classes de caracteres
- Cada ferramenta tem sua própria notação para expressões regulares

Conjuntos Não-regulares

- Algumas regras não podem ser descritas com expressões regulares. Em alguns casos são utilizadas Gramáticas Livres de Contexto para expressar tais regras

IFS8 Será que é necessário mesmo?
In Forma Software; 1/3/2007

Exemplo em Java?

Reconhecimento de *Tokens*

- Já sabemos como especificar *tokens*. E agora, como encontrá-los?

- A forma mais recomendada no aprendizado é o Diagrama de Transições seguido da implementação de uma máquina de estados
  - AFN – Autômatos Finitos Não-determinísticos
  - AFD – Autômatos Finitos Determinísticos

jz15

Diagrama de Transições

- É um tipo de fluxograma que delimita as ações de um analisador léxico para obter *tokens*
- Controla informações sobre caracteres através de "posições" (ou estados)
- Estados são conectados por setas (ou lados)
- Os lados possuem rótulos que indicam o caractere que motivou a transição
- O rótulo outro representa um caractere não identificado
- Um estado inicial indica o ponto de partida

Slide 34

jz15 Acho que se deixar apenas o diagrama fica melhor para explicar
É melhor definir logo o que é uma máquina de estados assim corta o bla bla bla
jz; 1/3/2007

Diagrama de Transições

- Exemplo, seja a linguagem

expr → *termo* **relop** *termo*
| *termo*

termo → **id** | **num**

relop → < | <= | = | <> | > | >=

id → **letra** (**letra** | **dígito**)*

letra → [A..Za..z]

dígito → [0..9]

num → **dígito**$^+$ (. **dígito**$^+$)? (E(+ | -)? **dígito**$^+$)?

Diagrama de Transições

- Diagrama de transições que reconhece >=

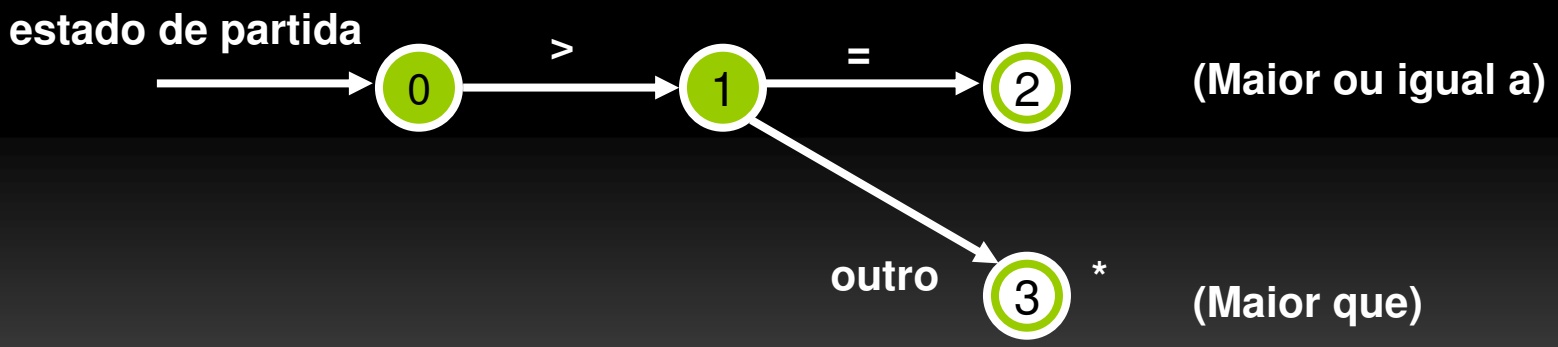


Diagrama de Transições

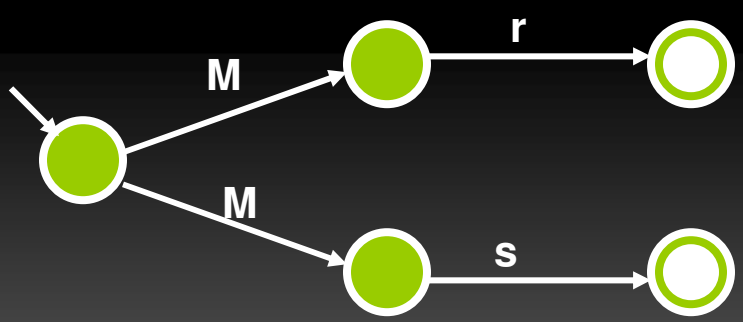
- Diagrama de transições que reconhece identificadores e palavras reservadas

- Como diferenciar **id** de **palavra-reservada**?

jz16

Máquina de Estados Finita

- Uma FSM (Finite State Machine) que reconhece M r | M s

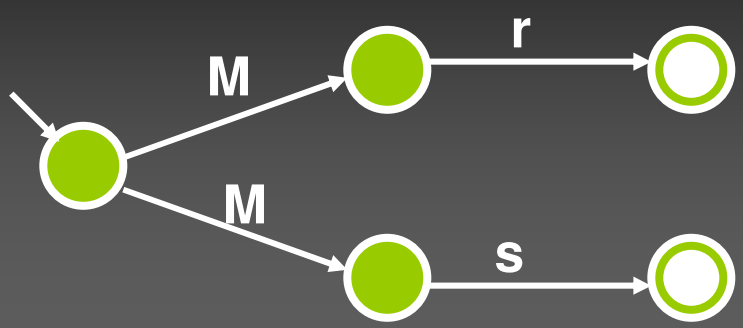


Slide 38

jz16 Talvez fique redundante falar de Máquina de Estados Finit e Diagrama de Transições
Talvez fosse o certo mesmo, e em seguida falar brevemente sobre automatos
jz; 1/3/2007

Autômatos Finitos Não-Determinísticos

- **Definição:** Um AFN é um modelo matemático que consiste em (S, Σ, δ, S_0, F)
  - S = Conjunto finito de estados
  - Σ = "Alfabeto" de símbolos de entrada (ex: ASCII)
  - δ = Função de transição
  - S_0 = *Estado inicial*
  - F = *Estado final*



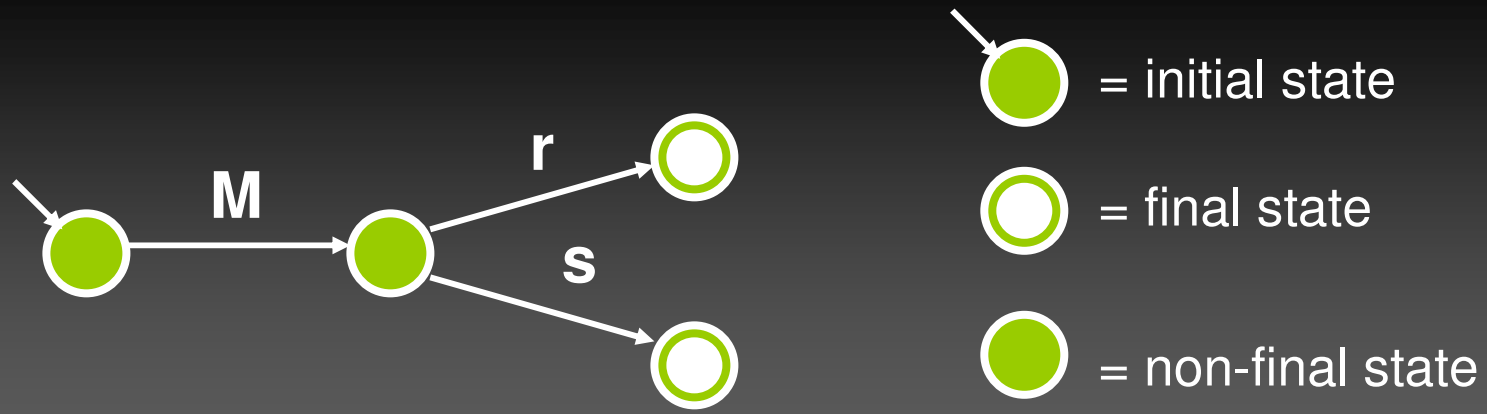
Autômatos Finitos Determinísticos

- **Definição:** Um AFD é um tipo especial de AFN

jz19

  - Nenhum estado possui uma transição-ε

  - Para cada estado *s* de entrada *a* existe no máximo um lado rotulado *a* deixando *s*



Slide 40

jz19 tentar entender melhor este conceito
ou colocar a definicao do slide 19 do sprog
jz; 1/3/2007

Parece divertido???

Lex

- Gerador de analisadores léxicos. Lê uma cadeia de caracteres que especificam o analisador léxico e produz o código fonte do analisador léxico em linguagem C
- Apenas em C?
  - JLex (Java)
  - RunCC (Java)
  - GALS (Java)

Definição de Tokens

Expressões Regulares

Java File: Scanner

Reconhece *Tokens*

Lex - Estrutura

declarações – variáveis, constantes, …
%%
regras de tradução – expr.regulares e ações em C
%%
procedimentos auxiliares

Lex - Exemplo

```
%{ /* definição de constantes LT, LE, EQ,…*/
%}
delim       [ \t\n]
ws          {delim}+
letter      [A-Za-z]
digit       [0-9]
id          {letter}({letter}|{digit})*
number      {digit}+(\.{digit}+)?
                    (E[+\-]?{digit}+)?
%%

…
```

IFS12 Conferir com o exemplo do livro do dragão

In Forma Software; 2/3/2007

Lex - Exemplo

```
…
{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);}
%%
…
```

Lex - Exemplo

```
…

int install_id() {
 Copia o lexema de um "id" para a tabela de
  símbolos.
}

int install_num() {
 Copia o lexema de um número para a tabela
  de símbolos.
}
```

Resumo

- Scanner e Parser
- Tokens, Padrões e Lexemas
  - Especificação de Tokens
  - Expressões Regulares
- Reconhecimento de Tokens
  - AFN
  - AFD
- Lex

Pra Finalizar...

- Onde encontro material para estudar?
  - Livro “UFRGS” (Capítulo 2)
  - Livro do “dragão” (Capítulo 3)
  - WEB!
    - JLex/JFlex
    - RunCC
    - GALS
    - Java RegExp
      java.sun.com/docs/books/tutorial/essential/regex/index.html

Só mais uma coisa...

- Monitoria
  - Data: 13/02 – 20:00
  - Local: Sala 42 ou Lab
- Tarefa de Casa
  1. No contexto de compiladores defina *Bootstrap(ping)*
  2. Como escrever (usando a linguagem X) um compilador para a linguagem X?
  3. Dada uma linguagem binária que só aceite 0 e 1, defina as expressões regulares (em Java) que
     1. representem os números binários (0s e 1s) que são potência de 2
     2. representem os números binários (0s e 1s) que são pares
     3. representem os números binários (0s e 1s) que são ímpares
- Entrega na próxima aula
- Até a próxima semana!