PARADIGMAS E LINGUAGENS DE PROGRAMAÇÃO

ENGENHARIA DA COMPUTAÇÃO – UFC/SOBRAL

Prof. Danilo Alves

danilo.alves@alu.ufc.br

ANÁLISE LÉXICA

UNIVERSIDADE FEDERAL DO CEARÁ

- Análise léxica: A análise léxica (AL) é responsável por ler o código fonte e separá-lo em partes significativas agrupando os caracteres em **lexemas** e produzir uma sequência de símbolos léxicos conhecidos como **tokens**.

- Classes de lexemas mais comuns:
  - Palavras reservadas;
  - Números inteiros sem sinal;
  - Números reais;
  - Cadeias de caracteres;

ANÁLISE LÉXICA

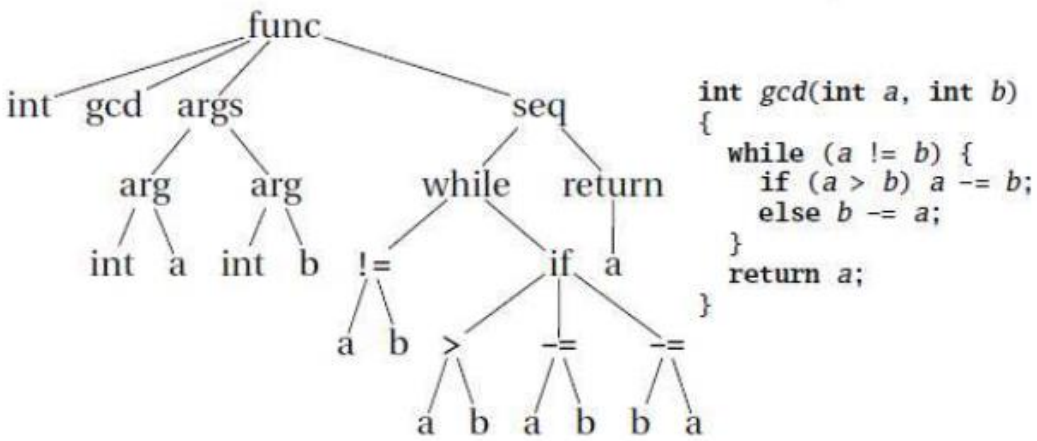
UNIVERSIDADE FEDERAL DO CEARÁ

- Exemplos de tokens que podem ser reconhecidos em uma linguagem de programação como C
  - palavras reservadas if else while do
  - operadores relacionais < > <= >= == !=
  - operadores aritméticos + * / -
  - operadores lógicos && || & | !
  - operador de atribuição =
  - delimitadores ; ,
  - caracteres especiais ( ) [ ] { }

ÁRVORE SINTÁTICA

- A análise sintática fornece uma árvore abstrata construída a partir das regras da gramática

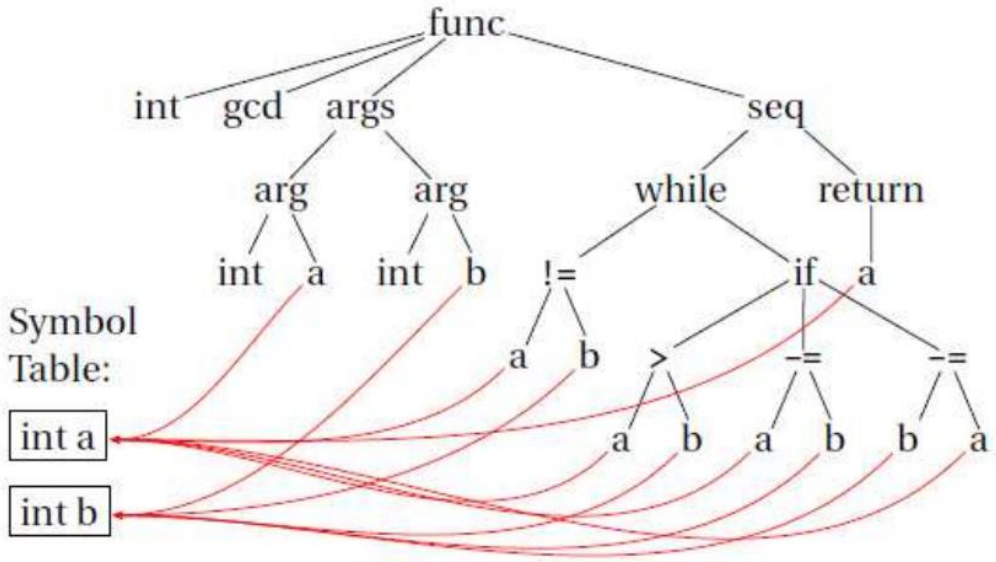
```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```



ANÁLISE SEMÂNTICA

- A análise semântica resolve símbolos, com tipos checados

```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```



ANÁLISE LÉXICA

UNIVERSIDADE FEDERAL DO CEARÁ

- Análise léxica: pode ser feita através de autômatos finitos (AF) ou expressões regulares
- AF é uma máquina de estados finitos. Formada por um conjunto de estados (um estado inicial e um ou mais estados finais)

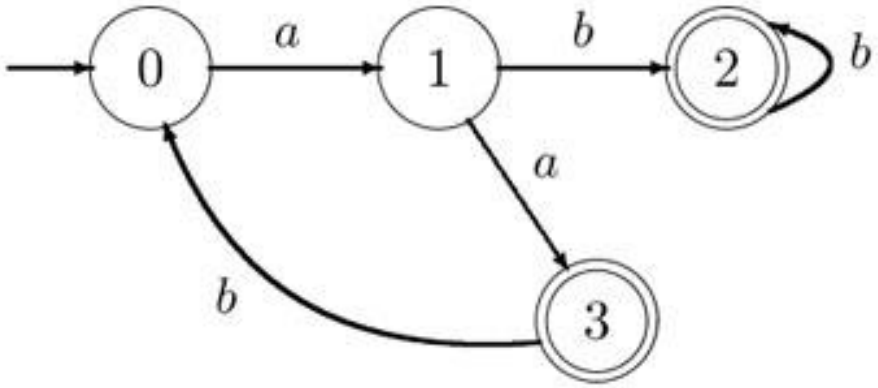
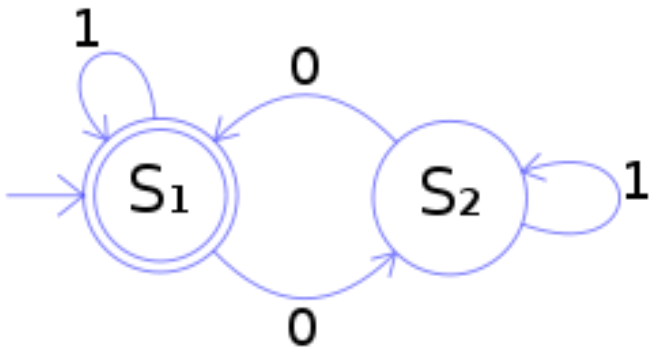


Figura 1: Autômato simples.



ANÁLISE LÉXICA

UNIVERSIDADE FEDERAL DO CEARÁ

- AF: exemplos...
  - 1. Cadeia de caracteres a,b,c
  - 2. Números inteiros (com ou sem sinal)
  - 3. Números reais (com ou sem sinais)
  - 4. Identificador
  - 5. Comparação (>, >=, <, <=, !=) entre dois identificadores
  - 6. Atribuição (=) entre dois identificadores
  - 7. Comando IF if(condicao){ comandos;} else { comandos;}
  - 8. Comando While

INTRODUÇÃO AOS AUTÔMATOS

UNIVERSIDADE FEDERAL DO CEARÁ

- Alfabeto: Conjunto finito e não vazio de símbolos (ou caracteres), que são denominados elementos do alfabeto.
- Símbolos: são representações gráficas indivisíveis.
  - Exemplos: b, abc, begin, if , 7045, 2.017e4.
- Palavra (ou cadeia de caracteres): uma palavra ou cadeia de caracteres sobre um alfabeto é uma sequência finita de símbolos do alfabeto justapostos.
  - As palavras sobre um alfabeto são denotadas por letras gregas minúsculas: α; β; γ; ...
- Comprimento da cadeia: quantidade de símbolos que a compõem.
  - Representada por |letra_grega|

- Σ representa um alfabeto

INTRODUÇÃO AOS AUTÔMATOS

UNIVERSIDADE FEDERAL DO CEARÁ

- O conjunto dos dígitos hexadecimais
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f\}$.
- 12ab, 88541aaf, e4, cdfee03, 8, são exemplos de palavras formadas
- Exemplo:
- Considerando as cadeias α = ab56d; β = 2; e γ = 4e sobre Σ, então, temos que $|\alpha|$ = ?; $|\beta|$ = ? e $|\gamma|$ = ?
- Cadeia Vazia: Uma cadeia sem símbolos.
  - Símbolo ε, formalmente por $|\varepsilon|$
  - Σ^* é o conjunto de todas as palavras possíveis
  - $\Sigma^+ = \Sigma^* - \{\varepsilon\}$

AUTÔMATOS FINITOS

UNIVERSIDADE FEDERAL DO CEARÁ

- Modelo matemático de um sistema com entradas e saídas discretas que pode assumir um número finito e pré-definido de estados.
- O primeiro modelo computacional de definição de linguagens
- Verificar se uma palavra w pertence a uma linguagem L, ou seja, o autômato verifica se
  - $w \in L$ ou $w \notin L$.
- A linguagem reconhecida pelo autômato finito é constituída por todas as palavras que passem no teste.

AUTÔMATO FINITO DETERMINÍSTICO (AFD)

UNIVERSIDADE FEDERAL DO CEARÁ

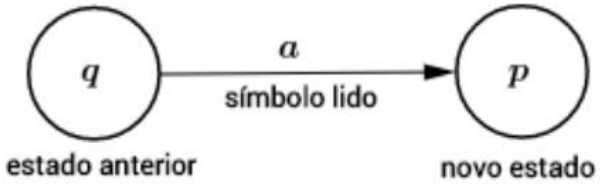
- Um autômato finito determinístico (AFD), ou simplesmente autômato finito (M) é uma quíntupla:
  - $M = (\Sigma, Q, \delta, q_o, F)$

- Q - Conjunto finito de estados possíveis do autômato
- Σ - Alfabeto de símbolos de entrada
- δ - Função de Transição ou Função Programa
  - $\delta: Q \times \Sigma \rightarrow Q$
  - $\delta(q, a) = p$
- F - Conjunto de estados finais, tais que $F \subseteq Q$.

REPRESENTAÇÃO DE UM AUTÔMATO

UNIVERSIDADE FEDERAL DO CEARÁ

- O estados do autômato é representado por nós
- Arcos representam transições
- q0 é o nodo destino de uma seta de origem
- qf (estado final) é representado de forma diferenciada, sendo o traço da circunferência do nó mais forte ou duplo.



REPRESENTAÇÃO DE UM AUTÔMATO

UNIVERSIDADE FEDERAL DO CEARÁ

- O processamento de um autômato finito M para uma palavra de entrada w consiste na sucessiva aplicação da Função de Transição para cada símbolo de w, da esquerda para direita, até ocorrer uma condição de parada.
- Ex: autômato finito M = ({a, b}, {q0, q1, q2, qf}, δ, q0, {qf})
- Função de transição δ dada por:

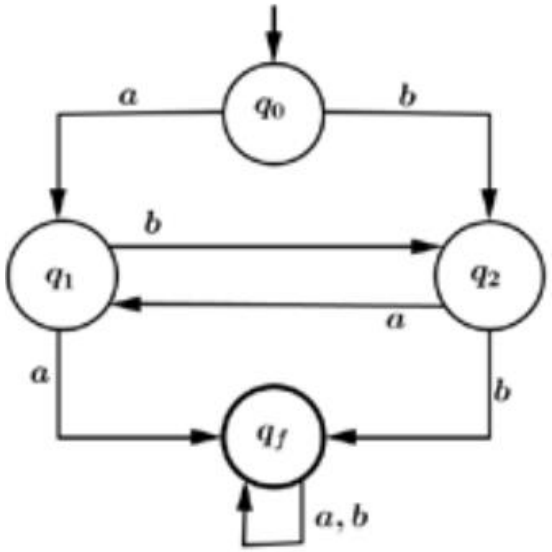
δ	a	b
q_0	q_1	q_2
q_1	q_f	q_2
q_2	q_1	q_f
q_f	q_f	q_f

REPRESENTAÇÃO DE UM AUTÔMATO

UNIVERSIDADE FEDERAL DO CEARÁ

- Pode-se deduzir que:
- Σ = {a, b}
- Q = {q0, q1, q2, qf} são os estados possíveis
- δ : (Q × Σ) → Q é a função de transição
- Q = {qf} é o estado final
- Diagrama de estados:
  - δ(q0, a) = q1

δ	a	b
q_0	q_1	q_2
q_1	q_f	q_2
q_2	q_1	q_f
q_f	q_f	q_f



LINGUAGEM DEFINIDA POR UM AF

UNIVERSIDADE FEDERAL DO CEARÁ

- A linguagem L definida por um autômato finito M é o conjunto de todas as cadeias w sobre o alfabeto Σ que levam M da sua configuração inicial para alguma configuração final por meio da aplicação sucessiva de transições definidas pela função δ.

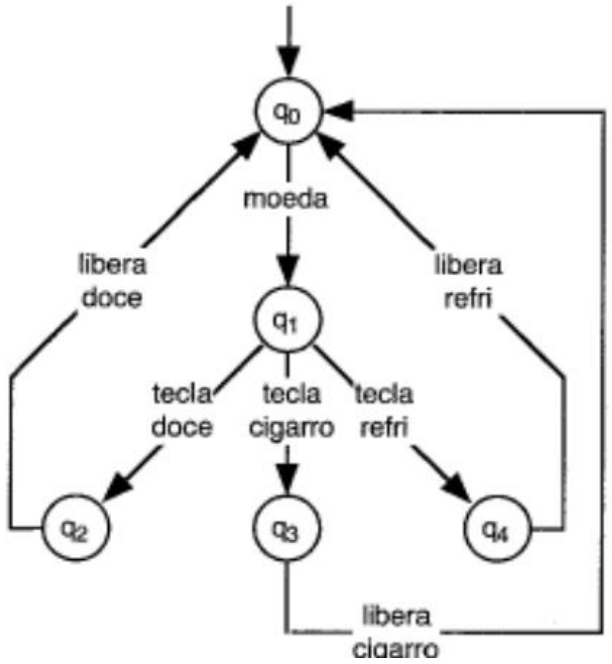
$$L(M) = \{w \in \Sigma^* \mid \delta(q0, w) \in F \}$$

- Linguagens reconhecidas pelo autômato anterior
- L1 = {aa, aaa, aaaa, ...}
- L2 = {bb, bba, bbab, bbaaba, bbabb}
- L3 = {abaa, abb, abba, abbb, aab}

EXEMPLO PRÁTICO

UNIVERSIDADE FEDERAL DO CEARÁ

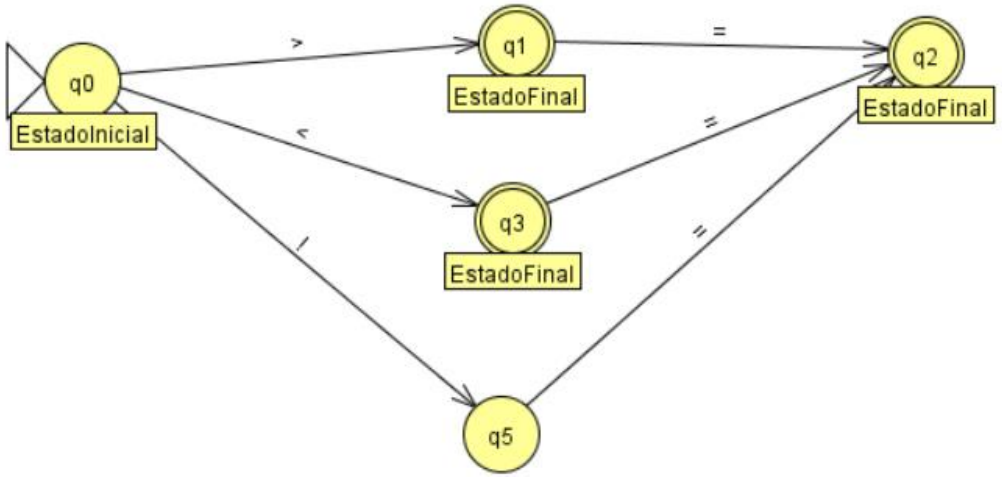
- Considere M1 o autômato finito ilustrado na figura abaixo, o qual representa a interface "homem × máquina" de uma máquina de vendas de refrigerante, cigarro e doce.
- Configuração desse autômato?

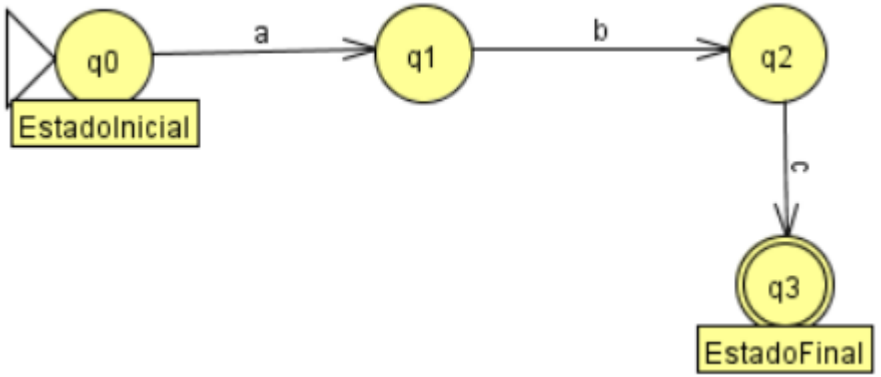


APLICAÇÃO NA LÉXICA

UNIVERSIDADE FEDERAL DO CEARÁ

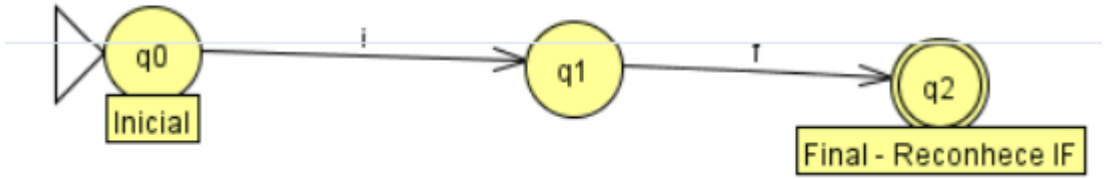
- Reconhecer a cadeia de caracteres a,b,c
- Reconhecer as cadeias de comparação (>, >=, <, <=, !=)





APLICAÇÃO NA LÉXICA

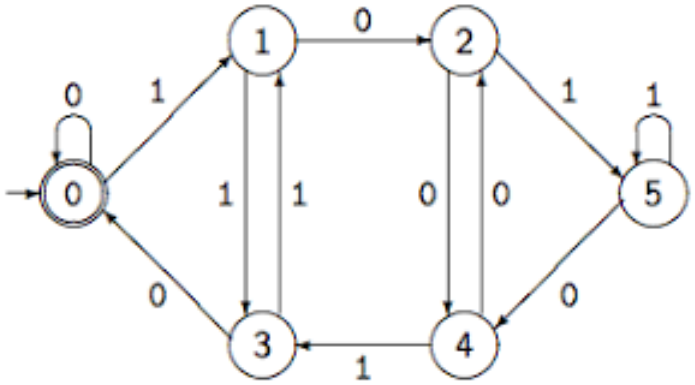
- Reconhecimento da palavra reservada IF



- Façam os demais AF para reconhecimento dos tokens
  - palavras reservadas (if else while do)
  - operadores lógicos (&& || & | !)

INTRODUÇÃO AOS AUTÔMATOS

- Exemplos



	0	1
→ q_0	q_2	q_0
* q_1	q_1	q_1
q_2	q_2	q_1

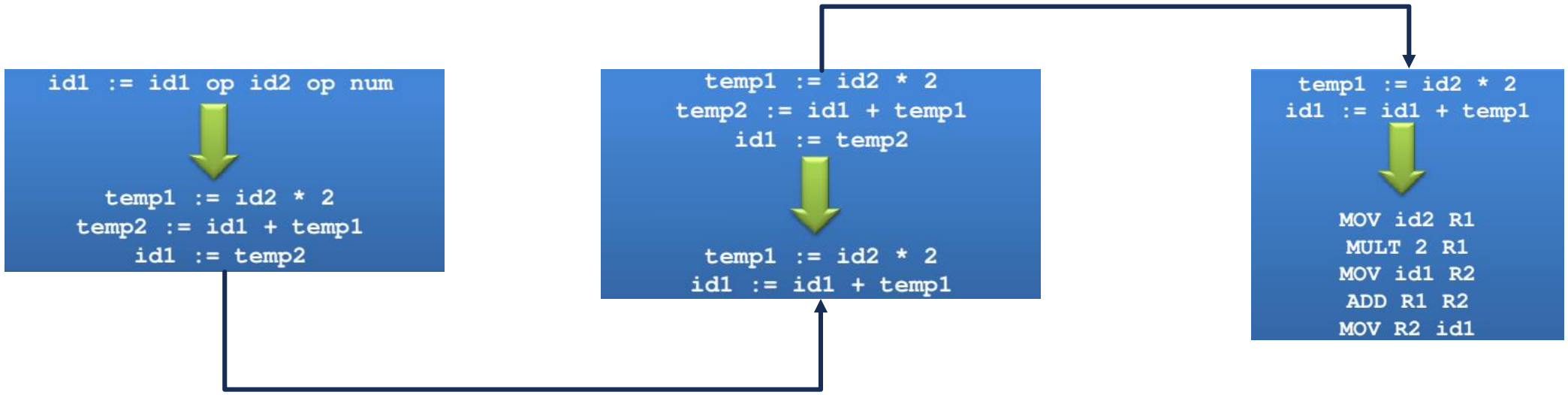
MÁQUINA DE TURING

UNIVERSIDADE FEDERAL DO CEARÁ

- Dispositivo teórico conhecido como máquina universal
- Alan Turing em 1936, muito antes dos computadores digitais
- Modelo abstrato de um computador, que se restringe apenas aos aspectos lógicos
- Modelar qualquer computador digital
- Invenção automática capaz de manipular símbolos em uma fita de acordo com uma série de regras para guardar informação

CÓDIGO INTERMEDIÁRIO

- Geração de código intermediário



SINTAXE DE UMA LINGUAGEM

UNIVERSIDADE FEDERAL DO CEARÁ

- O conhecimento da sintaxe da linguagem de programação usada, bem como do significado de suas construções – semântica – está diretamente relacionado com a **capacidade que o programador terá de se expressar** nessa linguagem

- O que o seguinte código faz?

```c
#include <stdio.h>

#define resp(x) x>17? 1: 0

void f1() { /* ... */ }
void f2() { /* ... */ }

int main(void) {
  int x;
  scanf("%d", &x);
  if (resp(x)) f1(); else f2();

  return 0;
}
```

TERMINOLOGIA DA SINTAXE

UNIVERSIDADE FEDERAL DO CEARÁ

- Uma sentença é uma cadeia de caracteres formada a partir do alfabeto da linguagem
- Uma linguagem é uma cadeia de sentenças
- Um **lexema** é a unidade sintática de mais baixo nível de uma linguagem
  - Identificadores, literais, operadores, palavras especiais

    Ex.: *, sum, begin
- Um (símbolo) token é uma categoria de lexemas
  - Ex.: identificador, literal inteiro, operador soma, operador de multiplicação

TERMINOLOGIA DA SINTAXE

UNIVERSIDADE FEDERAL DO CEARÁ

- Instrução em C: index = 2 * cont + 17;

Lexemas	Símbolos (*Token*)
index	identificador
=	sinal_igualdade
2	literal_inteiro
*	op_mult
cont	identificador
+	op_soma
17	literal_inteiro
;	ponto_e_virgula

CONCEITO DE LINGUAGEM

UNIVERSIDADE FEDERAL DO CEARÁ

- Linguagem é uma coleção de cadeias de símbolos, de comprimento finito. Estas cadeias são denominadas sentenças da linguagem, e são formadas pela justaposição de elementos individuais, os símbolos ou átomos da linguagem.
- Alfabeto = Conjunto de símbolos
- Cadeia = União de símbolos
  - Cadeia vazia ε
- A sintaxe de uma linguagem é descrita por uma **gramática**

DESCRIÇÃO DA SINTAXE DA LINGUAGEM

UNIVERSIDADE FEDERAL DO CEARÁ

- Gramáticas livres de contexto

  – Desenvolvido por Noam Chomsky no meio dos anos 1950

  – Geradores de linguagem, feitos para descrever a sintaxe de **linguagens naturais**

  – Define classes de linguagens, das quais uma se chamadas de **livres de contexto** e uma segunda se chama **linguagens regulares**

- Forma de Backus-Naur (1959)

  - BNF é uma notação natural para descrever sintaxe.

  – Inventada por **John Backus** para descrever **Algol 58** e modificada por Peter Naur para descrever o Algol 60

  – Forma de Backus-Naur (BNF) é equivalente às gramáticas livres de contexto (gramáticas)

DESCRIÇÃO DA SINTAXE DA LINGUAGEM

UNIVERSIDADE FEDERAL DO CEARÁ

- **Metalinguagem** é uma linguagem usada para descrever outra. Ex: BNF.
- Símbolos terminais: cadeias que estão no programa
  - while, do, for, id
- Símbolos não-terminais: não aparecem no programa
  - <cmd_while>, <programa>
- Produções: como produzir cadeias que formam o programa
  - <cmd_while> => while (<expressão>) <comandos>
- Símbolo inicial: não-terminal a partir do qual se inicia a produção do programa
  - <programa>

FORMA DE BACKUS-NAUR

UNIVERSIDADE FEDERAL DO CEARÁ

- Metalinguagem
  - A BNF é uma metalinguagem para descrever as LP
  - Uma descrição BNF, ou gramática, é uma coleção de regras.
- Abstrações
  - Símbolos Não-terminais
- Lexemas
  - Símbolos Terminais
- Produção
  - É uma definição de uma abstração;
  - O lado esquerdo corresponde à abstração;
  - O lado direito pode ser uma definição ou um conjunto de definições.

FUNDAMENTOS DE BNF

- Uma regra tem um lado esquerdo (LHS), que é um **não terminal**, e um lado direito (RHS), que é uma cadeia de **terminais** e **não terminais**
- O LHS é a abstração que está sendo definida
- O RHS é a definição de LHS, sendo composto por um misto de tokens, lexemas e referências a outras abstrações
- A definição completa, com LHS e RHS é chamada de **regra ou produção**
  - Exemplos das regras de BNF:

```
<assign> → <var> = <expression>
```

Regra ou produção

"abstração <assign> é definida como uma instância da abstração <var>, seguida pelo lexema =, seguido por uma instância da abstração <expression>"

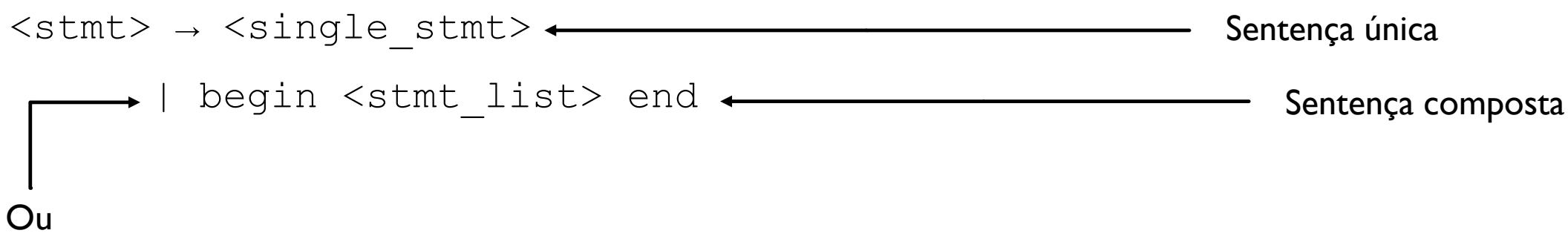
FUNDAMENTOS DE BNF

UNIVERSIDADE FEDERAL DO CEARÁ

- `<assign>` representa uma atribuição
- < > indica um não-terminal – termo que precisa ser expandido, por
- exemplo <variavel>
- Símbolos não cercados por < > são terminais;
  - Eles são representativos por si
  - Exemplo: if, while, (, =
- Os símbolos => significam é definido como

REGRAS DE BNF

- Uma abstração (ou símbolo não terminal) pode ter mais de uma definição ao lado direito




```
<if_stmt> → if (<logic_expr>) <stmt>
          | if (<logic_expr>) <stmt> else <stmt>
```

DESCREVENDO LISTAS E GRAMÁTICA

UNIVERSIDADE FEDERAL DO CEARÁ

- Utiliza recursão
- Uma regra é recursiva se seu lado esquerdo aparecer em seu lado direito

```
<ident_list> → identifier
             | identifier , <ident_list>
```

- Gramática: é um dispositivo de geração para definir linguagens
- Sequencias de aplicações de regras iniciando com um não terminal especial da gramática chamado de **símbolo inicial**
- O símbolo inicial representa um programa completo e é chamado de `<program>`
- A geração de uma sentença é chamada de **derivação**.

APLICANDO UMA BNF

UNIVERSIDADE FEDERAL DO CEARÁ

- O processo de tentar "reduzir" as regras de acordo com a entrada e a partir do símbolo inicial é conhecido como derivação.
  - Derivações podem ocorrer mais à esquerda ou mais à direita

- Derivação mais à esquerda: ocorre quando o não-terminal substituído é sempre mais à esquerda.

EXEMPLO DE DERIVAÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ

- Gramática:

```
<program> ::= begin <lista_cmd> end
<lista_cmd> ::= <cmd> | <cmd>; <lista_cmd>
<cmd> ::= <var> = <expr>
<var> ::= A|B|C
<expr> ::= <var> + <var> | <var> * <var> | <var>
```

- Entrada: A=B
- Derivação:
- <programa> ::= begin <lista_cmd> end

  begin <cmd> end

  begin <var> = <expr> end

  begin A= <expr> end

  begin A= <var> end

  begin A= B end

APLICANDO UMA GRAMÁTICA

UNIVERSIDADE FEDERAL DO CEARÁ

- Uma gramática simplificada para uma atribuição com operações aritméticas
  - – Ex: y = (2*x + 5)*x - 7;

```
<assignment> ::= ID = <expression> ;
<expression> ::= <expression> + <term>
                 | <expression> - <term>
                 | <term>
<term>    ::=  <term> * <factor>
                 | <term> / <factor>
                 | <factor>
<factor> ::=      ( <expression> )
                 | ID
                 | NUMBER
<ID> ::= x|y
<NUMBER> ::= 0|1|2|…|9
```

UM EXEMPLO DE GRAMÁTICA

UNIVERSIDADE FEDERAL DO CEARÁ

<program> → <stmts>

Ou

<stmts> → <stmt> | <stmt> ; <stmts>

<stmt> → <id> = <expr>

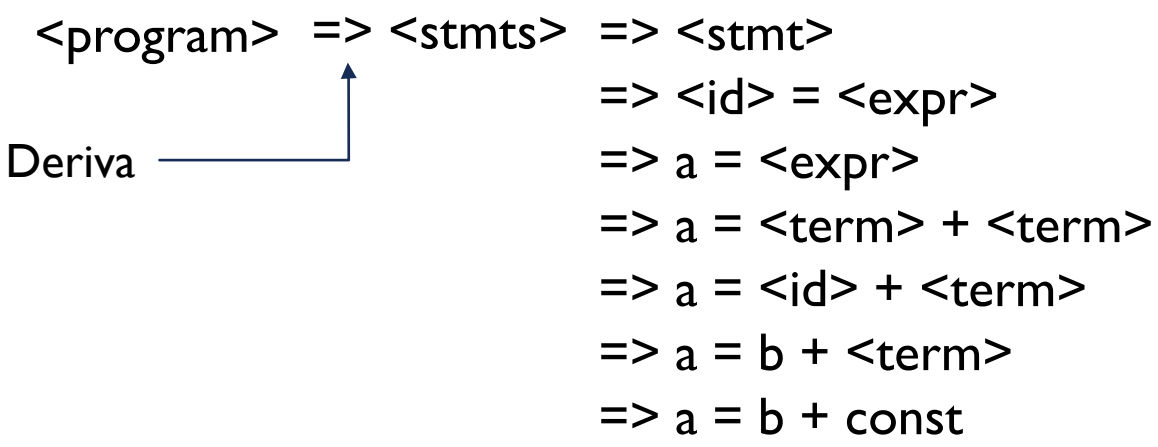
<id> → a | b | c | d

<expr> → <term> + <term> | <term> - <term>

<term> → <id> | const

Símbolo inicial

EXEMPLO DE DERIVAÇÃO



DERIVAÇÕES

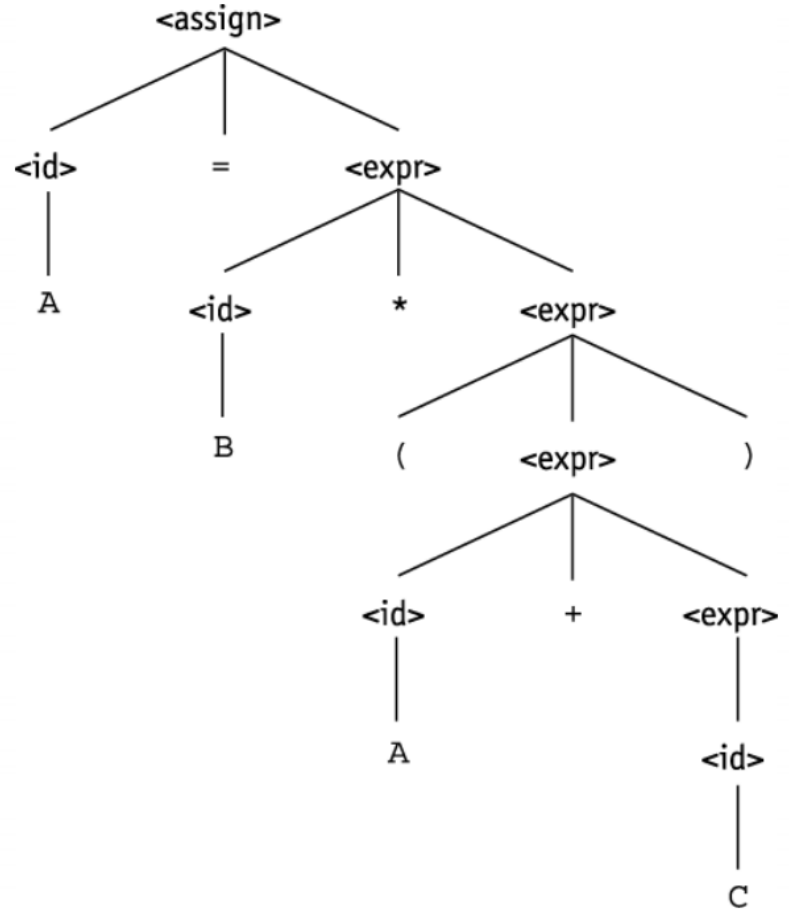
- Uma gramática para uma pequena linguagem e aplicação na expressão: A = B + C ; B = C

<program> → **begin** <stmt_list> **end**
<stmt_list> → <stmt>
| <stmt> ; <stmt_list>
<stmt> → <var> = <expression>
<var> → A | B | C
<expression> → <var> + <var>
| <var> – <var>
| <var>

<program> => **begin** <stmt_list> **end**
=> **begin** <stmt> ; <stmt_list> **end**
=> **begin** <var> = <expression> ; <stmt_list> **end**
=> **begin** A = <expression> ; <stmt_list> **end**
=> **begin** A = <var> + <var> ; <stmt_list> **end**
=> **begin** A = B + <var> ; <stmt_list> **end**
=> **begin** A = B + C ; <stmt_list> **end**
=> **begin** A = B + C ; <stmt> **end**
=> **begin** A = B + C ; <var> = <expression> **end**
=> **begin** A = B + C ; B = <expression> **end**
=> **begin** A = B + C ; B = <var> **end**
=> **begin** A = B + C ; B = C **end**

ÁRVORE DE ANÁLISE SINTÁTICA

- Representação hierárquica de uma derivação
- A = B * (A + C)
- <program> → <assigns>
- <assigns> → <assign> | <assign> ; <assigns>
- <assign> → <id> = <expr>
- <id> → a | b | c | d
- <expr> → <id> + <expr> | <id> * <expr> | (<expr>) | <id>



EXERCÍCIO

UNIVERSIDADE FEDERAL DO CEARÁ

- Usando a gramática seguinte represente as seguintes expressões, pelo método da derivação. Em seguida, construa a árvore de análise sintática

a) A = A + B * C + C

b) B = B * A + C * B + A

c) A = (A + B) * C

d) A = (B * C) + A * B

```
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <expr> + <expr>
       | <expr> * <expr>
       | (<expr>)
       | <id>
```

UMA GRAMÁTICA PARA LÓGICA PROPOSICIONAL

UNIVERSIDADE FEDERAL DO CEARÁ

- Sentença → SentençaAtômica | SentençaComplexa
- SentençaAtômica → Verdadeiro | Falso
- Símbolo → P | Q | R | S

- SentençaComplexa → (Sentença)
  | [Sentença]
  | ¬ Sentença (Negação)
  | Sentença ∧ Sentença (E)
  | Sentença ∨ Sentença (OU)
  | Sentença ⇒ Sentença (Implica)
  | Sentença ⇔ Sentença (Equivalencia)
  | Símbolo
- Precedência: ¬, ∧, ∨, ⇒, ⇔

EXERCÍCIO

UNIVERSIDADE FEDERAL DO CEARÁ

- Usando a gramática seguinte represente as expressões em lógica proposicional, pelo método da derivação. Em seguida, construa a árvore de análise sintática

a) ¬ ¬ ¬P

b) ¬ ¬P

c) ¬P ∨ Q

d) P ∨ Q ∧ R

e) ¬P ∧ ¬Q

f) P ∨ Q => R

Ex: ¬P ∧ R ∨ Q

g) (P => R) ∧ (Q => R)

h) ¬(P => Q)

i) P ⇔ Q => R

j) (¬ P => Q)