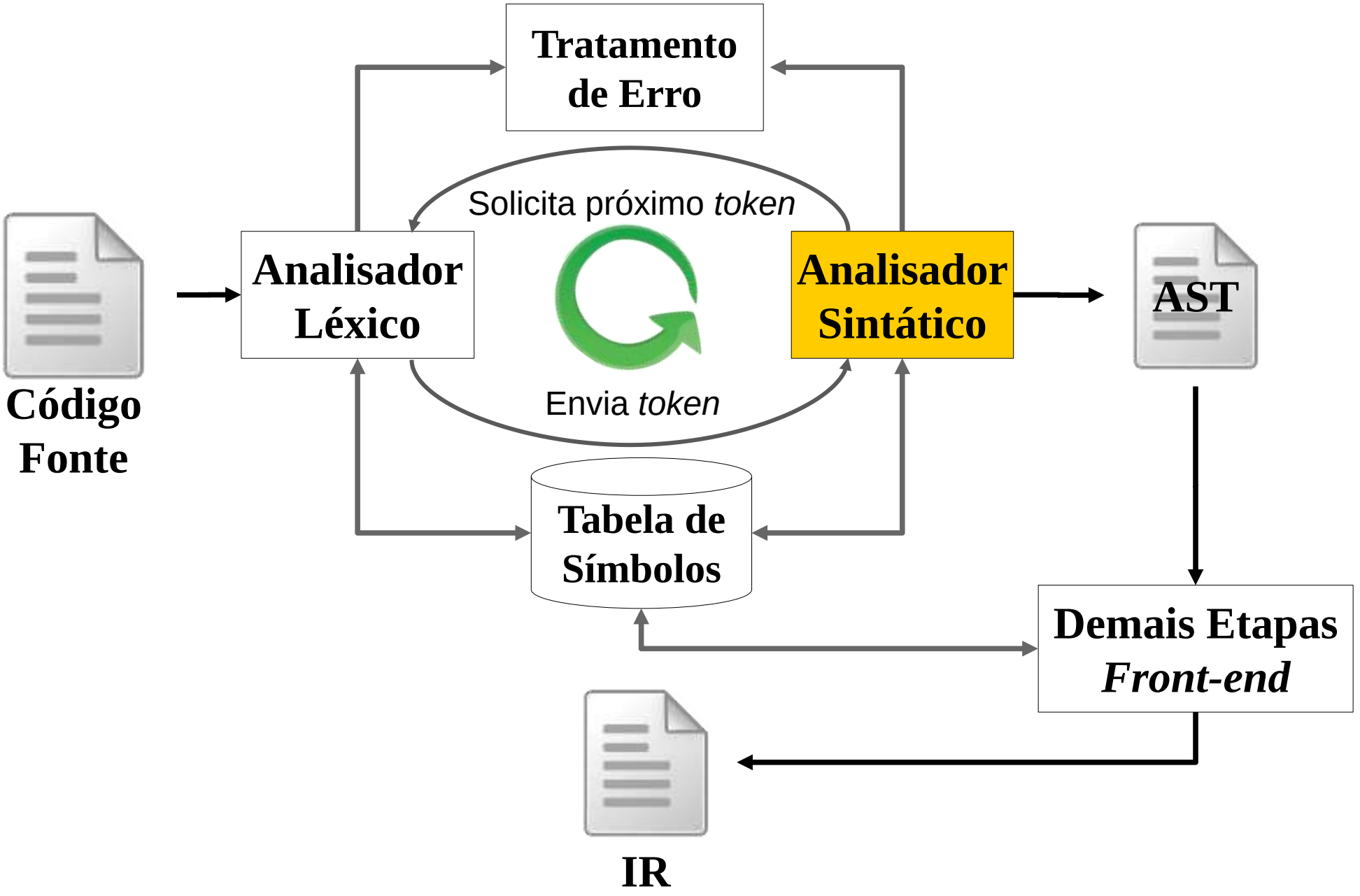
Universidade Federal de Uberlândia
Faculdade de Computação

Análise Sintática (Conceitos Básicos)

Curso de Bacharelado em Ciência da Computação
GBC071 - Construção de Compiladores
Prof. Luiz Gustavo Almeida Martins

Etapa de Análise do Compilador



Análise Sintática

- Determina se a **estrutura sintática do programa** (**cadeia de *tokens***) é aceita na linguagem
  - Especificação usa **gramáticas livre de contexto (GLC)**
    - *Tokens* são os **símbolos terminais** da gramática
  - GLC descrevem **maior parte da sintaxe** das linguagens
  - Fases subsequentes devem analisar a AST para garantir a compatibilidade com as **regras que não foram contempladas**
- Reconhecimento baseado em **autômato de pilha (AP)**

Autômato de Pilha (AP)

- **Máquina de estados** usada no reconhecimento das sentenças (**estrutura sintática**) das linguagens livres de contexto

- Autômato consiste em uma 6-tupla **(*S, Σ, Γ, δ, s , F*)**:
  - **Conjunto finito de estados *S***
  - **Conjunto de símbolos *Σ*** (**alfabeto de entrada**)
  - **Conjunto de símbolos *Γ*** (**alfabeto da pilha ou auxiliar**)
  - **Função de transição *δ*** que determina os estados alcançáveis e a operação na pilha a partir do estado atual, do topo da pilha e do símbolo lido
  - Um **estado inicial** ou de partida *s*
  - Um **conjunto de estados finais** ou de aceitação $F \in S$

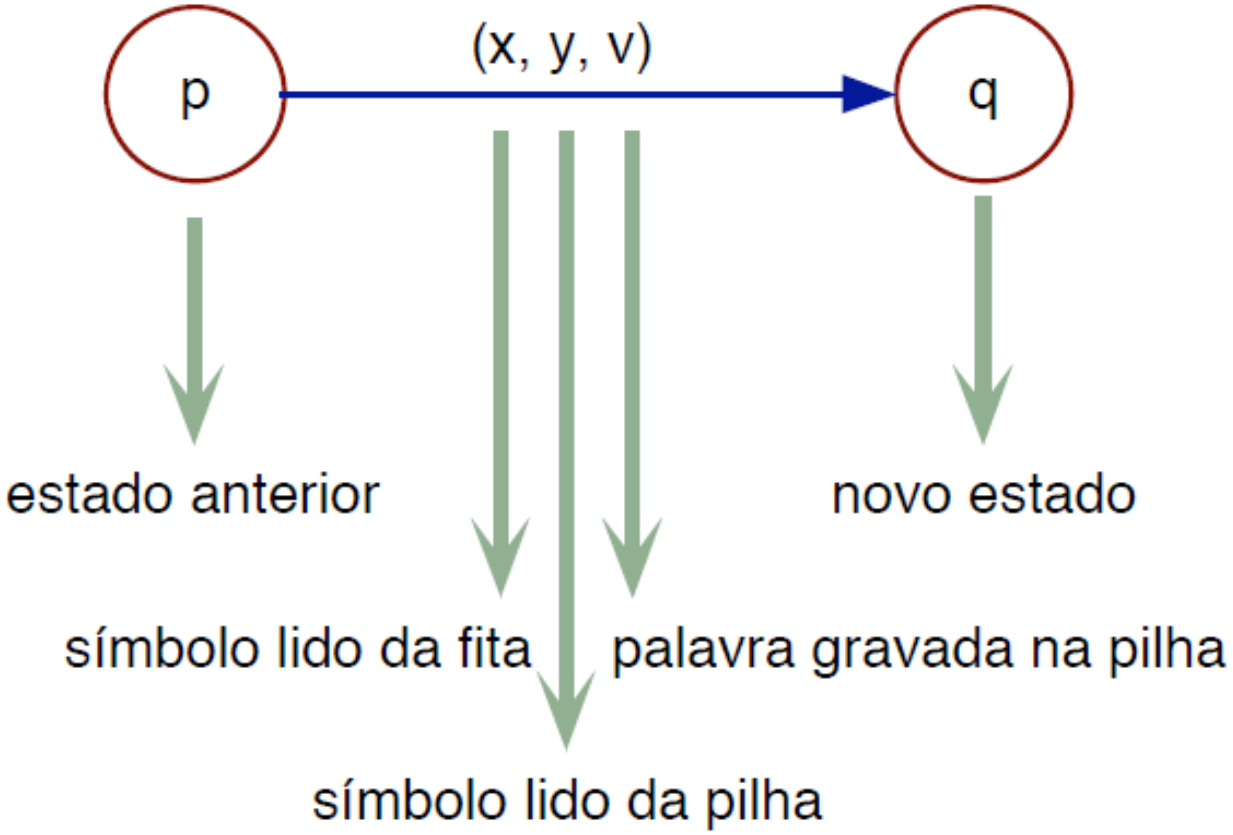
Autômato de Pilha (AP)

- Novidade é o uso da **pilha** como **memória auxiliar**
  - Pilha tem **tamanho ilimitado** e possui **alfabeto próprio**
    - Alfabeto independe dos símbolos de entrada
  - Transições consideram operações de **escrita e leitura no topo da pilha**
  - Garante **propriedade de auto-incorporação** da linguagem
    - **Ex:** balanceamento de escopo, aninhamentos de comandos, etc.

Autômato de Pilha (AP)

- **Representação gráfica da função de transição:**

$$\delta(p,x,y) = \{(q,v)\}$$

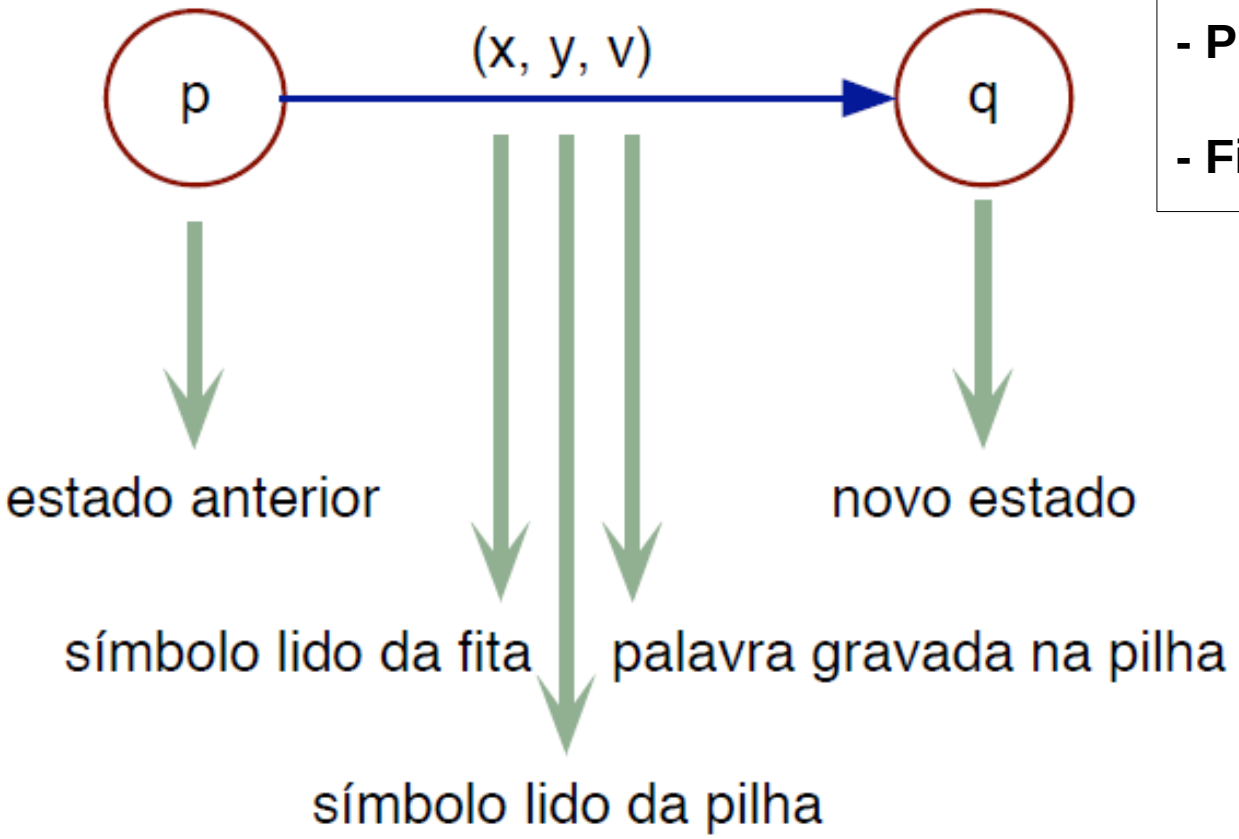


Fonte: [Menezes, 2011]

Autômato de Pilha (AP)

- **Representação gráfica da função de transição:**

$$\delta(p,x,y) = \{(q,v)\}$$



Símbolo ε (movimento vazio)

- **Pilha:** não lê ou grava no topo

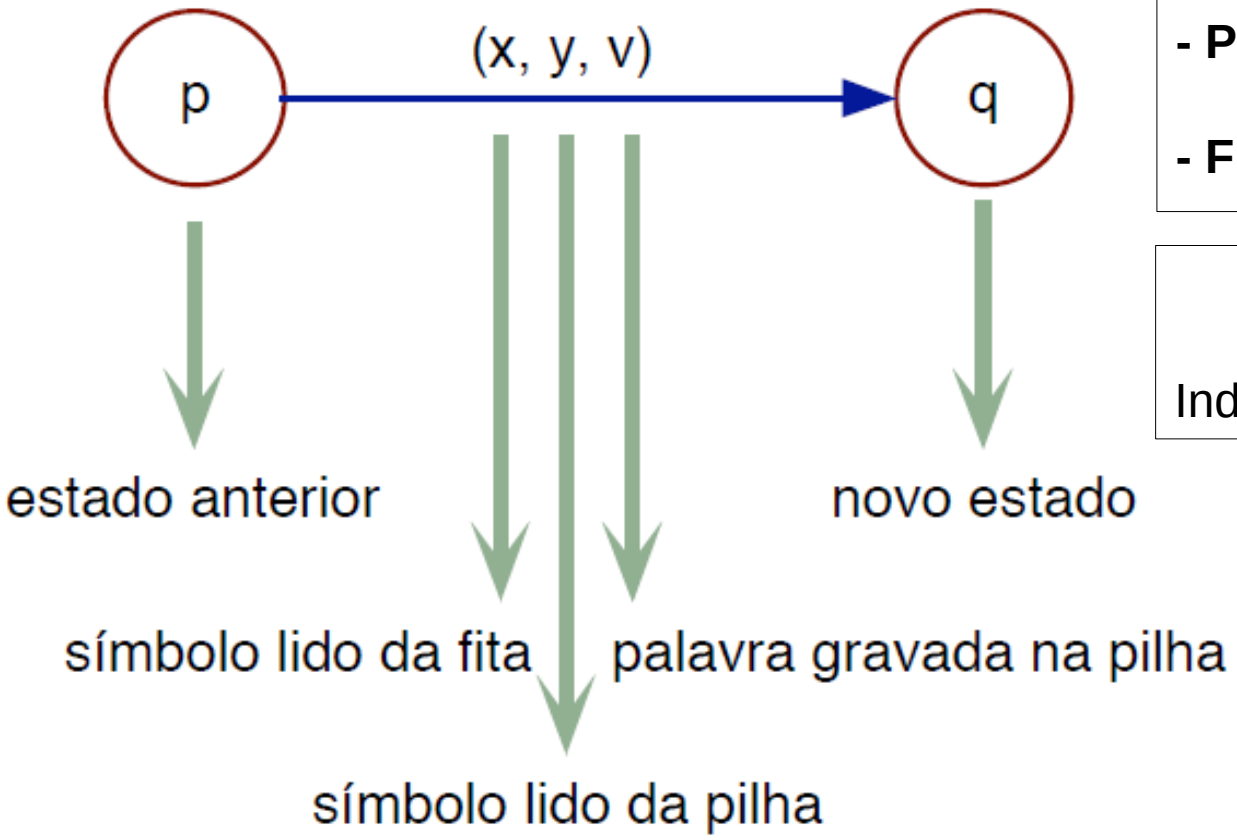
- **Fita:** não move a cabeça de leitura

Fonte: [Menezes, 2011]

Autômato de Pilha (AP)

- **Representação gráfica da função de transição:**

$$\delta(p,x,y) = \{(q,v)\}$$



Símbolo ε (movimento vazio)

- **Pilha:** não lê ou grava no topo

- **Fita:** não move a cabeça de leitura

Símbolo ?

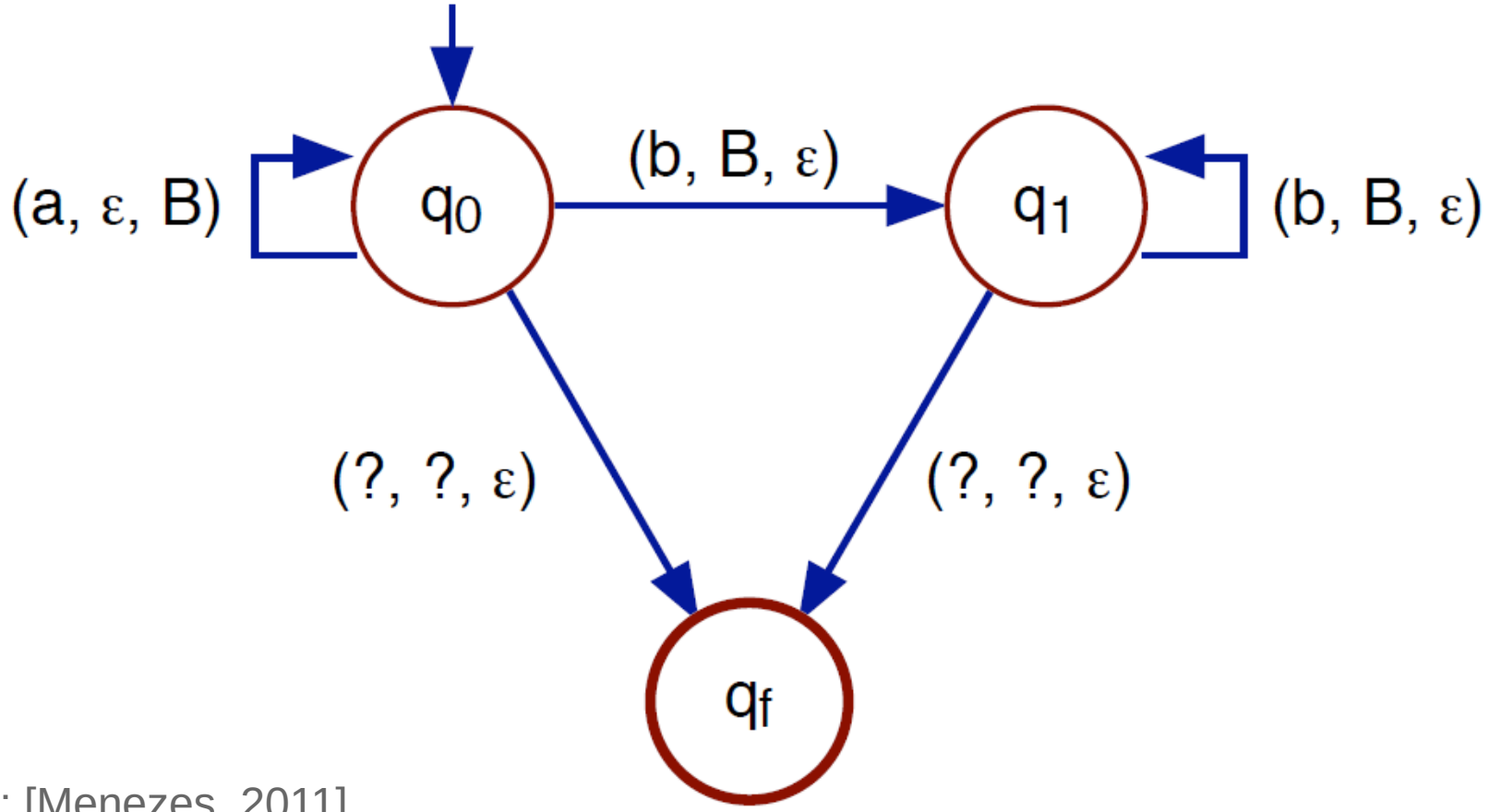
Indica **pilha vazia ou fim da entrada**

Fonte: [Menezes, 2011]

Autômato de Pilha (AP)

- **Exemplo:** balanceamento duplo (***a b***)

M1 = ({ q , q , q }, { a, b }, { B }, δ , q , { q })



Fonte: [Menezes, 2011]

Análise Sintática

- Analisador sintático deve ser capaz de construir a **árvore de derivação da cadeia de *tokens* (árvore sintática)**
  - **Determinar a sequência de produções** para a cadeia de entrada (**árvore de derivação**)
  - Construção da árvore garante a **tradução correta**
  - Na prática, a construção efetiva **não é necessária**

Estratégias de Análise Sintática

- Existem 2 estratégias para verificar se uma cadeia de *tokens* pertencente a GLC:
  - **Descendente (*top-down*): expansão** da cadeia de *tokens*
    - Inicia pelo símbolo inicial da GLC (**nó raiz**)
    - Busca forma sentencial que case com a cadeia de *tokens* (**nós-folha**)
  - **Ascendente (*bottom-up*):** adota o **processo inverso da expansão**
    - Inicia pela cadeia de *tokens* (**nós-folha**)
    - Busca "reduzir" a forma sentencial até alcançar o símbolo inicial (**nó raiz**)
- Diferença entre as estratégias está na **forma como a árvore de derivação é construída** (**derivações canônicas**):
  - **Descendente: derivação mais a esquerda**
  - **Ascendente: derivação mais a direita**

Derivações Canônicas

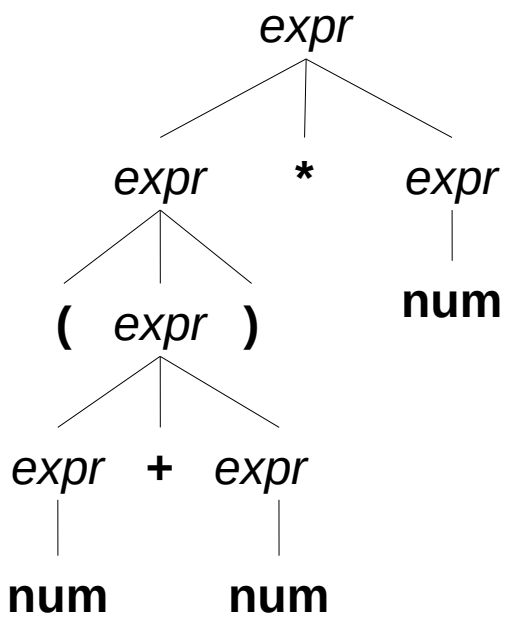
- Mesma árvore pode ter **diferentes derivações** (***1 : n***)
  - Geradas por formas sentenciais com + de um símbolo não-terminal
  - **Problema:** ***Qual o próximo símbolo não-terminal a derivar?***

- **Derivações canônicas** estabelecem uma **forma sistemática** de selecionar o próximo símbolo a ser substituído (***1 : 1***)
  - Uma **única derivação mais a esquerda (*leftmost derivation*):**
    - Aplica uma regra de derivação ao **símbolo não-terminal mais à esquerda**
    - Ordem das derivações resulta na **sequência de reconhecimento dos métodos descendentes** (***leftmost parse***)
  - Uma **única derivação mais a direita (*rightmost derivation*):**
    - **Símbolo não-terminal mais à direita** é selecionado para a derivação
    - Ordem **inversa** das derivações resulta na **sequência de reconhecimento dos métodos ascendentes** (***rightmost parse***)

Derivações Canônicas

- **Ex:** considere a sentença **(num + num) * num** e a gramática:
  - *expr* → *expr* **+** *expr* ① | *expr* ***** *expr* ② | **(** *expr* **)** ③ | **num** ④

Árvore Derivação



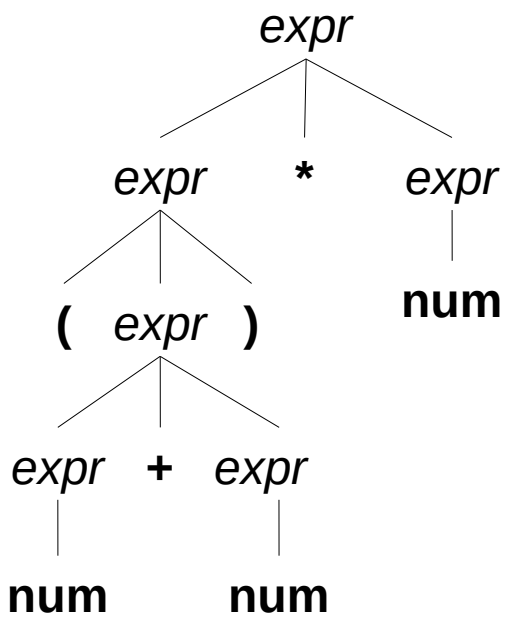
Derivação mais a esquerda (*leftmost derivation*)

expr
② ⇒ *expr* ***** *expr*
③ ⇒ **(** *expr* **)** ***** *expr*
① ⇒ **(** *expr* **+** *expr* **)** ***** *expr*
④ ⇒ **(** **num** **+** *expr* **)** ***** *expr*
④ ⇒ **(** **num** **+** **num** **)** ***** *expr*
④ ⇒ **(** **num** **+** **num** **)** ***** **num**

Derivações Canônicas

- **Ex:** considere a sentença **(num + num) * num** e a gramática:
  - *expr* → *expr* **+** *expr* ① | *expr* ***** *expr* ② | **(** *expr* **)** ③ | **num** ④

Árvore Derivação



Percorrimento em pré-ordem da AST

Derivação mais a esquerda (*leftmost derivation*)

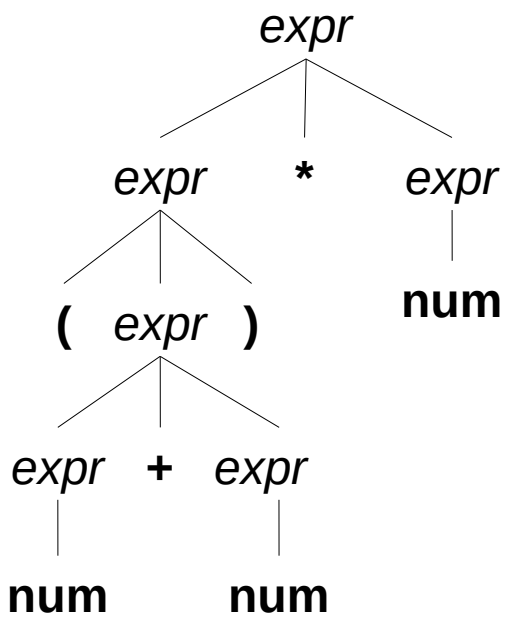
expr
② ⇒ *expr* ***** *expr*
③ ⇒ **(** *expr* **)** ***** *expr*
① ⇒ **(** *expr* **+** *expr* **)** ***** *expr*
④ ⇒ **(** **num** **+** *expr* **)** ***** *expr*
④ ⇒ **(** **num** **+** **num** **)** ***** *expr*
④ ⇒ **(** **num** **+** **num** **)** ***** **num**

Sequência de reconhecimento mais a esquerda (*leftmost parse*): ②③①④④④

Derivações Canônicas

- **Ex:** considere a sentença **(num + num) * num** e a gramática:
  - *expr* → *expr* **+** *expr* ① | *expr* ***** *expr* ② | **(** *expr* **)** ③ | **num** ④

Árvore Derivação



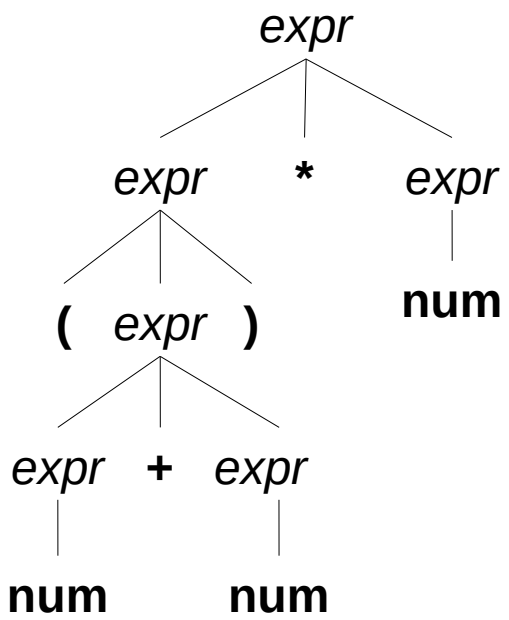
Derivação mais a direita
(*rightmost derivation*)

expr
② ⇒ *expr* ***** *expr*
④ ⇒ *expr* ***** **num**
③ ⇒ **(** *expr* **) * num**
① ⇒ **(** *expr* **+** *expr* **) * num**
④ ⇒ **(** *expr* **+ num) * num**
④ ⇒ **(num + num) * num**

Derivações Canônicas

- **Ex:** considere a sentença **(num + num) * num** e a gramática:
  - *expr* → *expr* **+** *expr* ① | *expr* ***** *expr* ② | **(** *expr* **)** ③ | **num** ④

Árvore Derivação



Percorrimento em pós-ordem da AST

Derivação mais a direita (*rightmost derivation*)

expr
② ⇒ *expr* ***** *expr*
④ ⇒ *expr* ***** **num**
③ ⇒ **(** *expr* **) * num**
① ⇒ **(** *expr* **+** *expr* **) * num**
④ ⇒ **(** *expr* **+ num) * num**
④ ⇒ **(num + num) * num**

Sequência de reconhecimento mais a direita (*rightmost parse*): ④④①③④②

Estratégia Universal

- Possibilita a análise de **qualquer gramática livre de contexto (GLC)**

- **Muito ineficiente** para o desenvolvimento de compiladores de produção
  - **Não-determinismo** dificulta a escolha da produção apropriada
  - Custo para analisar uma cadeia com ***n*** *tokens* é de ***O(n)***

- **Exemplos:**
  - Algoritmo de Earley (***top-down***)
  - Algoritmo de Cocke-Younger-Kasami (***bottom-up***)

Estratégias Usadas na Prática

- Métodos mais eficientes são **determinísticos**
  - Linguagens são analisadas em **tempo linear** (**O(*n*)**)
  - Trabalham com **subclasses das GLC:**
    - **Descendentes: gramáticas LL**
    - **Ascendentes: gramáticas LR**

- Usa **autômato de pilha (AP)** para reconhecer a estrutura sintática do código
  - Pilha guarda as informações dos **nós relevantes da árvore**
  - Relação quase direta entre produções e transições
  - Construção **simples e imediata** por **tabelas de análise sintática**

- Permite a **detecção de erros sintáticos** o mais cedo possível
  - Identifica prefixos da entrada que não pertencem a linguagem (**propriedade de prefixo viável**)

Referências Bibliográficas

- Aho, A.V.; Lam, M.S.; Sethi, R.; Ullman, J.D. Compiladores: Princípios, técnicas e ferramentas, 2 ed., Pearson, 2008
- Alexandre, E.S.M. Livro de Introdução a Compiladores, UFPB, 2014
- Aluisio, S. material da disciplina “Teoria da Computação e Compiladores”, ICMC/USP, 2011
- Dubach, C. material da disciplina “*Compiling Techniques*”, University of Edinburgh, 2018
- Freitas, R. L. notas de aula - Compiladores, PUC Campinas, 2000
- Menezes, P.B. Linguagens Formais e Autômatos, 6 ed., Bookman, 2011
- Ricarte, I. Introdução à Compilação, Elsevier, 2008