Análise x Síntese
Processos Envolvidos

Flávio Márcio

Exemplo – Somas Inteiras Válidas

- Suponha que queremos desenvolver um compilador para analisar expressões de soma entre constantes e variáveis inteiras. Restrições:
  - as constantes são inteiras, sem zeros não significativos e formadas por um ou mais dígitos numéricos;
  - as variáveis são formadas por uma única letra maiúscula ou minúscula;
  - só é aceita a operação de soma entre dois operandos;
  - parêntesis podem ser utilizados para alterar a precedência das somas;

Analisador Léxico

- Alfabeto = { (,) , + , [A-Za-z] , [0-9]}
- id -> [A-Za-z]
- semzero -> [1-9]
- comzero -> [0-9]
- num -> semzero comzero*
- Tokens = { (,) , + , id , num}

Analisador Léxico - Funcionamento

- Exemplo 1: 30 + (idade + 213)
- num + (idade + num) Erro léxico! “id” inválido!
- Exemplo 2: 0030 + (X + 213)
- 0030 + (id + num) Erro léxico! “num” inválido!
- Exemplo 3: 30 + (X + 213)
- num + (id + num) Ok!
- Exemplo 4: 30 +) X + 213 (
- num +) id + num (Lexicamente Ok!
- **Mais adiante veremos que esta sintaticamente errado.**

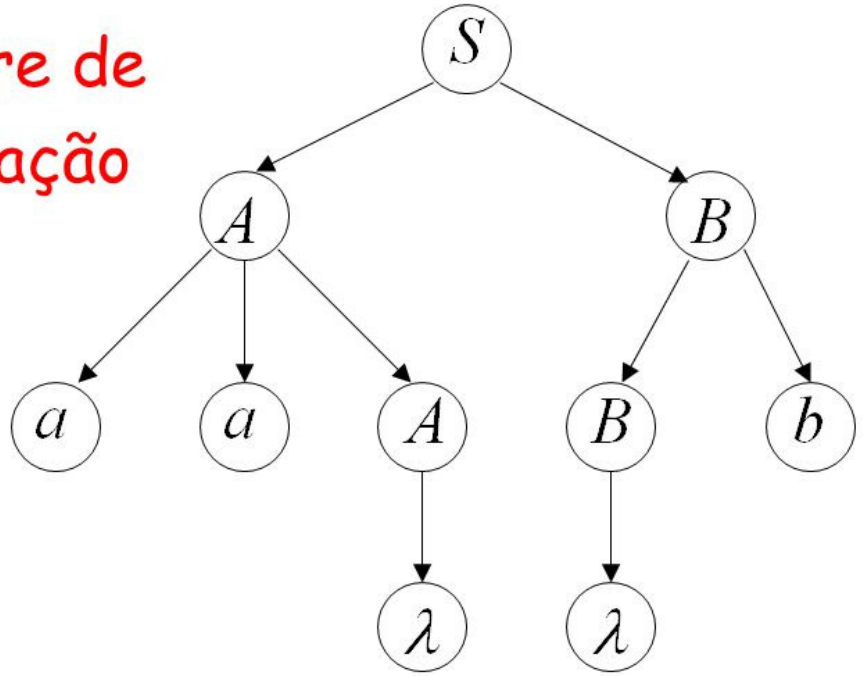
Árvore de derivação

- Notação que exibe uma derivação no formato de árvore.

$$S \to AB \qquad A \to aaA \mid \lambda \qquad B \to Bb \mid \lambda$$

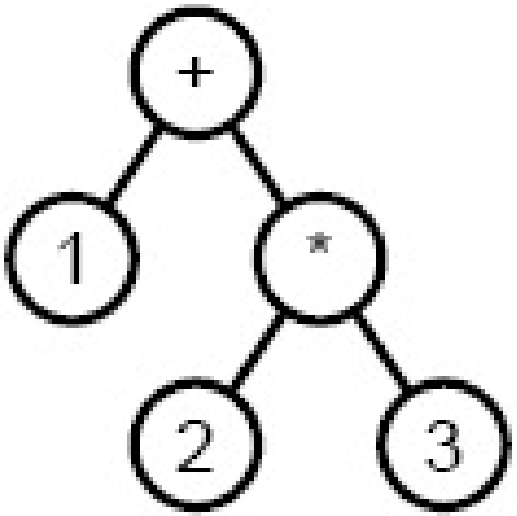
$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb \Rightarrow aab$$

Árvore de derivação



Árvore de Derivação - Outro Exemplo

1 + 2*3 → Parsing



Análise Sintática

- Construir um analisador sintático consiste em criar uma gramática livre de contexto (GLC), capaz de produzir todas as árvores de derivação sintaticamente válidas para o problema, sem ambiguidade.
- Mais adiante veremos que contornar a ambiguidade, significa fazer com que a gramática tenha um caminho ou sequência de derivações única para produzir a árvore.

Analisador Sintático

- Projetar uma gramática livre de contexto que gere todas as "Somas Inteiras Válidas"
- **E -> E + E | F**
- **F -> id | num | (E)**
- Exemplo de derivação para (id + id) + num:
- E => E + E => F + E => (E) + E => (E + E) + E =>
- (F + E) + E => (id + E) + E => (id + F) + E =>
- (id + id) + E => (id + id) + F => (id + id) + num
- Obs: o que esta em vermelho foi trocado pelo que esta sublinhando. Derivação mais a esquerda, trocar sempre o não terminal mais a esquerda.

De volta ao exemplo com erro sintático

- Gramática proposta:
- E -> E + E | F
- F -> id | num | (E)

- Note que a sequência de tokens:
- **num +) id + num (**
- Não pode ser derivada a partir da gramática proposta. Pois está sintaticamente incorreta.

Problema da Ambiguidade

- E -> E + E | F
- F -> id | num | (E)
- Mais de uma derivação mais a esquerda para **id + id + num**:
- E => E + E => E + E + E => F + E + E => id + E + E =>
- id + F + E => id + id + E => id + id + F => id + id + num

- E => E + E => F + E => id + E => id + E + E => id + F + E
- => id + id + E => id + id + F => id + id + num

Como eliminar a ambiguidade?

- Impor restrições às regras da GLC sem afetar a linguagem produzida pela mesma.
  - Através de alterações feitas pelo projetista da gramática e da linguagem. Utilizando sua experiência e insight.
  - Transformação para uma “Forma Normal” adequada. Através de algoritmos próprios.
- Vamos aprender a segunda que irá nos ajudar a desenvolver a experiência e insight; exigidos na primeira opção.

Gramática não ambígua

- E -> FM
- M -> +FM | λ
- F -> (E) | id | num
- Única derivação mais a esquerda para **id + id + num**:
- E => FM => id M => id +FM => id + id M => id + id +FM => id + id + num M => **id + id + num**

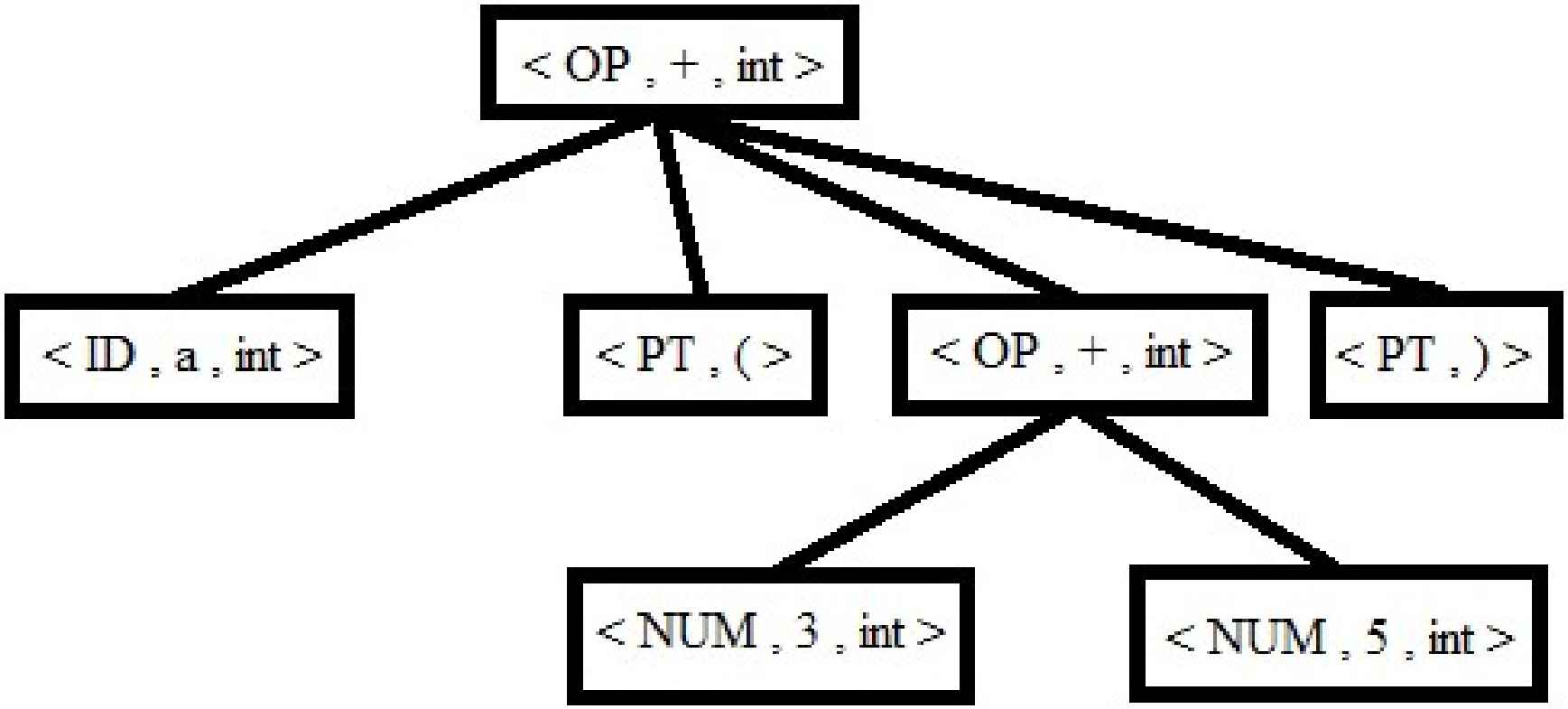
Análise Semântica

- Exemplo: **a + (3 + 5)**
- id + (num + num) Lexicamente correto!
- E -> FM
- M -> +FM | λ
- F -> (E) | id | num
- E => FM => idM => id+FM => id+(E)M => id+(FM)M =>
- id+(numM)M => id+(num+FM)M => id+(num+numM)M =>
- id+(num+num)M => id + (num + num)
- Sintaticamente correto!
- **Análise semântica:** verificar se o valor atribuído a variável "a" é inteiro; verificar se as operações "(3+5)" e "a+(3+5)" produzem resultados inteiros.

Análise Semântica

- Para realizar as suas análises, o analisador semântico cruza informações da árvore sintática, com restrições/pré-requisitos da linguagem e informações presentes na tabela de símbolos. Criando a **“árvore semântica”** (próximo slide).
- É pré-requisito que as constantes sejam inteiras.
- É pré-requisito que uma soma entre operandos inteiros possua resultado inteiro.
- É necessário saber qual é o tipo do identificador “a”, esta informação esta na tabela de símbolos.

Árvore Semântica



Síntese para código intermediário

- **1) Seleção de instruções:** utilização de um algoritmo com abordagem bottom up de cobertura máxima, para gerar um conjunto de instruções mínimo em linguagem de montagem, que seja equivalente as operações presentes na árvore semântica.
- ADD Temp1 , 3 , 5
- ADD Temp1 , a, Temp1

Síntese para código intermediário

- **2) Alocação de registradores:** o algoritmo que atua aqui, tem que conhecer os detalhes de baixo nível desta arquitetura. Irá gerar um código em assembly executável no processador. Exemplo de código no MIPS:
  - variável “a” atribuída ao registrador $s0
- addi $t0 , $zero , 5
- addi $t1 , $zero , 3
- add $t2 , $t0 , $t1
- add $t2 , $s0, $t2

Síntese para código de máquina

- **3) Montagem:** uso do software montador para conversão das pseudo instruções da etapa anterior em suas respectivas representações em binário.
- **4) Link edição**
- **5) Carregamento**

API x ABI

- Como programar em alto nível para uma base de aplicação (Sistema Operacional (SO) + Arquitetura do Computadores)
- Conhecer a API (Application Programming Interface)
  - Conjunto de chamadas de sistema fornecidas pelo SO, em suas bibliotecas, para acesso aos recursos de hardware da máquina.
  - Na maioria das vezes tais funcionalidades são oferecidas, em linguagem de alto nível, pelas bibliotecas prontas do compilador (Ex: #includes).

API x ABI

- Como programar em assembly para uma arquitetura de computadores?
- Conhecer a ABI (Application Binary Interface)
  - Instruções do ISA (Instruction Set Architecture) e seus formatos.
  - Convenção de uso dos registradores (quantidades, tipos e funcionalidades).
  - Modos de endereçamento suportados pelo processador.
- Sugestão: Estudo de caso ABI do MIPS

Próxima aula

- Iremos aprender a eliminar a ambiguidade de uma GLC, transformando esta para uma "forma normal" e fatorando a mesma a esquerda.