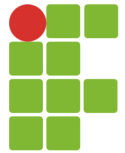
Árvores Sintáticas Abstratas

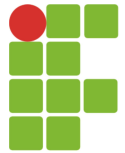
Compiladores

Prof. Daniel Saad Nogueira Nunes

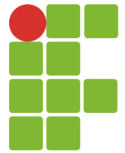
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

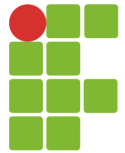


Sumário



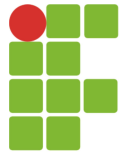
Introdução

- Em linguagens mais simples, compiladores podem ser escritos de modo a fazer a tradução em uma única etapa.
- **Tradução dirigida por sintaxe**.
- Contudo, em linguagens mais complexas, é necessário realizar utilizar mais de uma etapa.
- Para evitar a análise do programa em cada etapa, podemos construir uma estrutura de dados durante a análise sintática, conhecida como **árvore sintática abstrata**, que alimentará outras fases, como: a construção da tabela de símbolos, a análise semântica, e a produção de código.

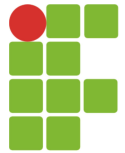


Introdução

- Nesta aula examinaremos como realizar a tradução dirigida pela sintaxe e como construir árvores sintáticas abstratas para dar suporte as outras etapas.

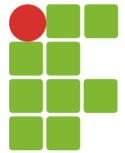


Sumário



Tradução dirigida por sintaxe

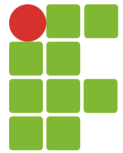
- Para alcançar a tradução dirigida por sintaxe, inserimos código no analisador sintático coadunado com as suas ações sintáticas.
- Dois conceitos importantes: **ações semânticas** e **valores semânticos**.



Tradução dirigida por sintaxe

Ações semânticas

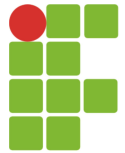
- Cada produção pode ter um código que será executado quando a produção é aplicada.
- Não há imposição sobre o que o código pode ou deve fazer.
- Esses códigos são conhecidos como **ações semânticas**, visto que seu escopo está além da sintaxe da gramática.
- Ações relacionadas ao **significado** de um programa.



Tradução dirigida por sintaxe

Valores semânticos

- Quando uma ação semântica é realizada para uma produção $\mathrm{A} \rightarrow \mathrm{X_1} \ldots \mathrm{X_m}$, valores estão relacionados com cada símbolo.
- O Valor associado a A é baseado nos valores atribuídos a $\mathrm{X_1} \ldots \mathrm{X_m}$.
- Para símbolos terminais, esses valores são obtidos na análise léxica.
- Por exemplo: o valor de um identificador é o seu nome.
- Para símbolos não-terminais, os valores são obtidos após a aplicação das regras.

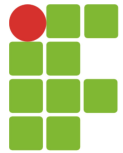


Tradução dirigida por sintaxe.

Considere a seguinte gramática:

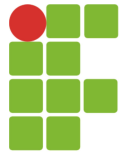
$$\begin{aligned}
&\text{Start} \rightarrow \text{E}\ \$ \\
&\text{E} \rightarrow \text{E}\ +\ \text{T} \\
&\text{E} \rightarrow \text{T} \\
&\text{T} \rightarrow \text{T}\ \cdot\ \text{T} \\
&\text{T} \rightarrow \text{num}
\end{aligned}$$

Em que num é um inteiro.



Tradução dirigida por sintaxe

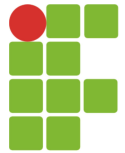
- Podemos escrever um analisador sintático LL(1)que calcula o resultado de uma expressão qualquer, como por exemplo $31 + 8 \cdot 50$, além é claro, de verificar que é uma expressão bem formada.



Tradução dirigida por sintaxe.

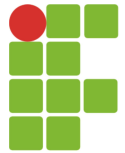
A gramática LL(1) equivalente é:

$$\text{Start} \rightarrow \text{E } \$$$
$$\text{E} \rightarrow \text{T } \text{E}'$$
$$\text{E}' \rightarrow \varepsilon$$
$$\text{E}' \rightarrow + \text{ T } \text{E}'$$
$$\text{T} \rightarrow \text{F } \text{T}'$$
$$\text{T}' \rightarrow \cdot \text{ F } \text{T}'$$
$$\text{T}' \rightarrow \varepsilon$$
$$\text{F} \rightarrow \text{num}$$



Tradução dirigita por sintaxe

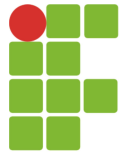
- Ao inserir as ações semânticas no analisador descendente recursivo obtemos o seguinte algoritmo.



Tradução dirigida por sintaxe

Algorithm 1: START()

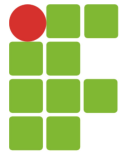
```
1 if( ts.PEEK(∈){num} )
2 |   val ← E()
3 |   ts.MATCH($)
4 |   return val
5 else
6 |   REPORT-SYNTAX-ERROR()
```



Tradução dirigida por sintaxe

Algorithm 2: E()

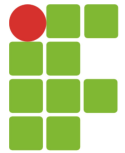
```
1 if( ts.PEEK() ∈ {num} )
2 |   val ← T()
3 |   val ← val + E'()
4 |   return val
5 else
6 |   REPORT-SYNTAX-ERROR()
```



Tradução dirigida por sintaxe

Algorithm 3: E'()

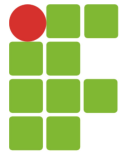
```
1 if( ts.PEEK() ∈ {$} )
2     return 0
3 else if( ts.PEEK() ∈ {+} )
4     ts.MATCH(+)
5     val ← T()
6     val_2 ← E'()
7     return val + val_2
8 else
9     REPORT-SYNTAX-ERROR()
```



Tradução dirigida por sintaxe

Algorithm 4: $\mathrm{T}()$

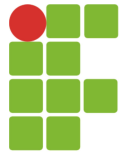
```
1 if( ts.PEEK() ∈ {num} )
2 |   val ← F()
3 |   val_2 ← T'()
4 |   return val · val_2
5 else
6 |   REPORT-SYNTAX-ERROR()
```



Tradução dirigida por sintaxe

Algorithm 5: T'()

```
1 if( ts.PEEK() ∈ {·} )
2     ts.MATCH(·)
3     val ← F()
4     val_2 ← T'()
5     return val · val_2
6 else if( ts.PEEK() ∈ {+, $} )
7     return 1
8 else
9     REPORT-SYNTAX-ERROR()
```

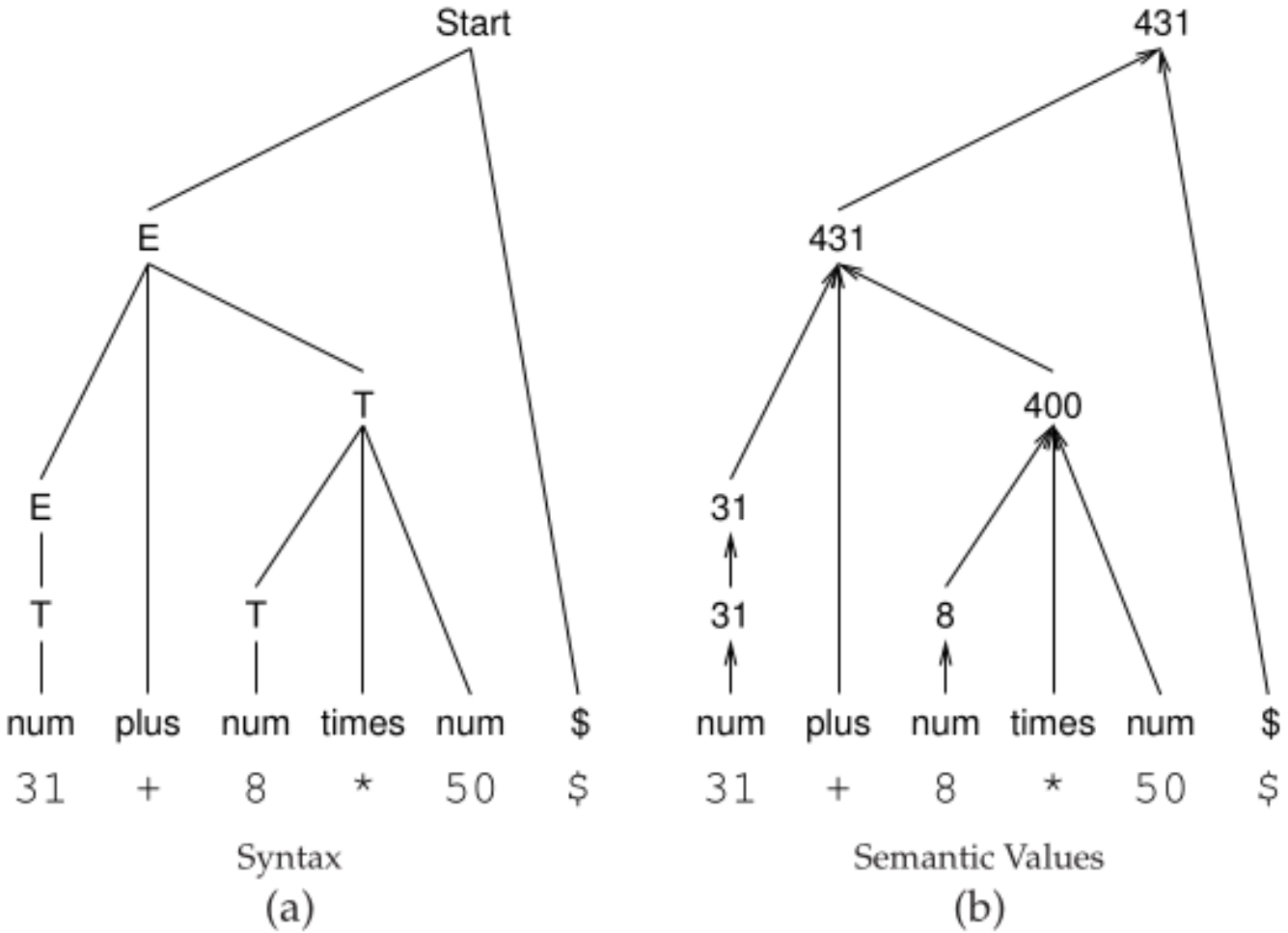


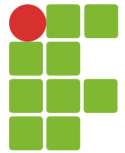
Tradução dirigida por sintaxe

Algorithm 6: F()

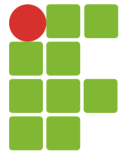
```
1 if( ts.PEEK() ∈ {num} )
2     val ← ts.VALUE()
3     ts.MATCH(num)
4     return val
5 else
6     REPORT-SYNTAX-ERROR()
```

Tradução dirigida por sintaxe



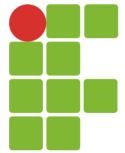


Sumário



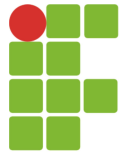
Árvores sintáticas abstratas

- Muitas tarefas de um compilador podem ser realizadas em uma única etapa através da tradução dirigida por sintaxe.
- Contudo, isso é desencorajado, visto que muitas funcionalidades acabam sendo incorporadas em um único componente do compilador, no caso, o analisador sintático.
- Tarefas como construção da tabela de símbolos, otimização de código, e análise semântica, merecem ter um tratamento separado.
- Consolidar todos esses componentes durante a análise sintática faz com que o código gerado seja difícil de ser mantido e estendido.



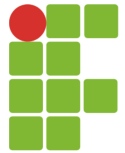
Árvores sintáticas abstratas

- A árvore sintática abstrata (AST) é uma estrutura de dados que pode ajudar nesse processo de separação de componentes.
- Ela é uma estrutura que é construída durante a análise sintática e utilizada pelos demais componentes.

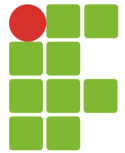


Árvores sintáticas abstratas

- ASTs devem ser, ao mesmo tempo, concisas e flexíveis.
- Devem ser simples, mas capazes de descrever o programa do seu ponto de vista sintático sem a perda de informações importantes.

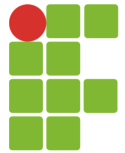


Sumário



Árvores sintáticas concretas e abstratas

- Árvores sintáticas **concretas** representam a entrada de acordo com a análise sintática.
- Contudo, elas possuem informações que muitas das vezes são desnecessárias para que a entrada seja realizada sem a ocorrência de ambiguidades, com por exemplo: delimitadores, palavras chaves e parênteses para garantir a ordem de avaliação de expressões.



Árvores sintáticas concretas e abstratas

Considere a seguinte gramática:

1 Start → Stmt $

2 Stmt → id assign E

3 | if lparen E rparen Stmt else fi

4 | if lparen E rparen Stmt fi

5 | while lparen E rparen do Stmt od

6 | begin Stmts end

7 Stmts → Stmts semi Stmt

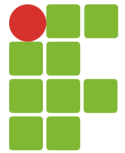
8 | Stmt

9 E → E plus T

10 | T

11 T → id

12 | num

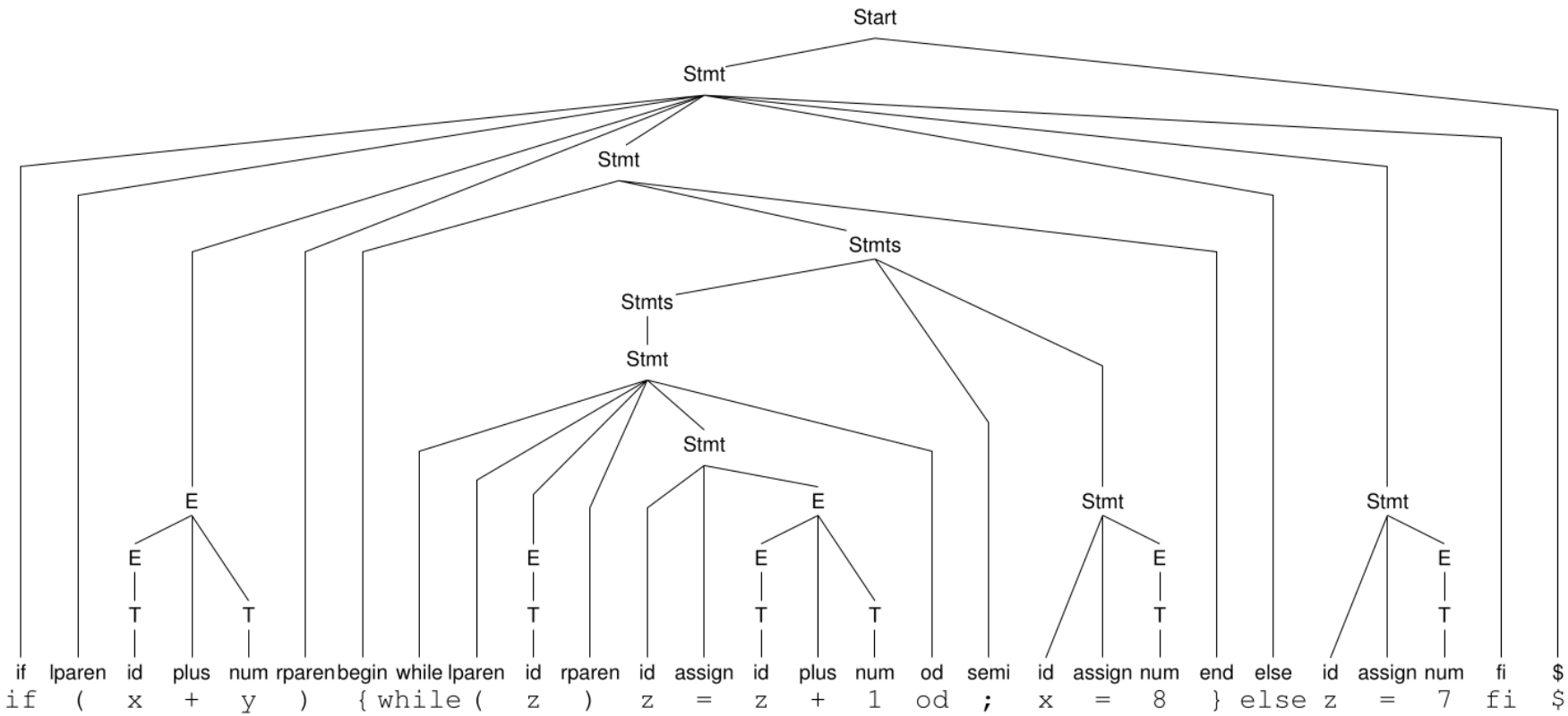


Árvores sintáticas concretas

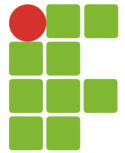
- A árvore sintática para a sequência de tokens

  ```
  if ( x + y ) { while ( z ) z = z+1
  od ; x = 8} else z = 7 fi $
  ```

  vem a seguir.



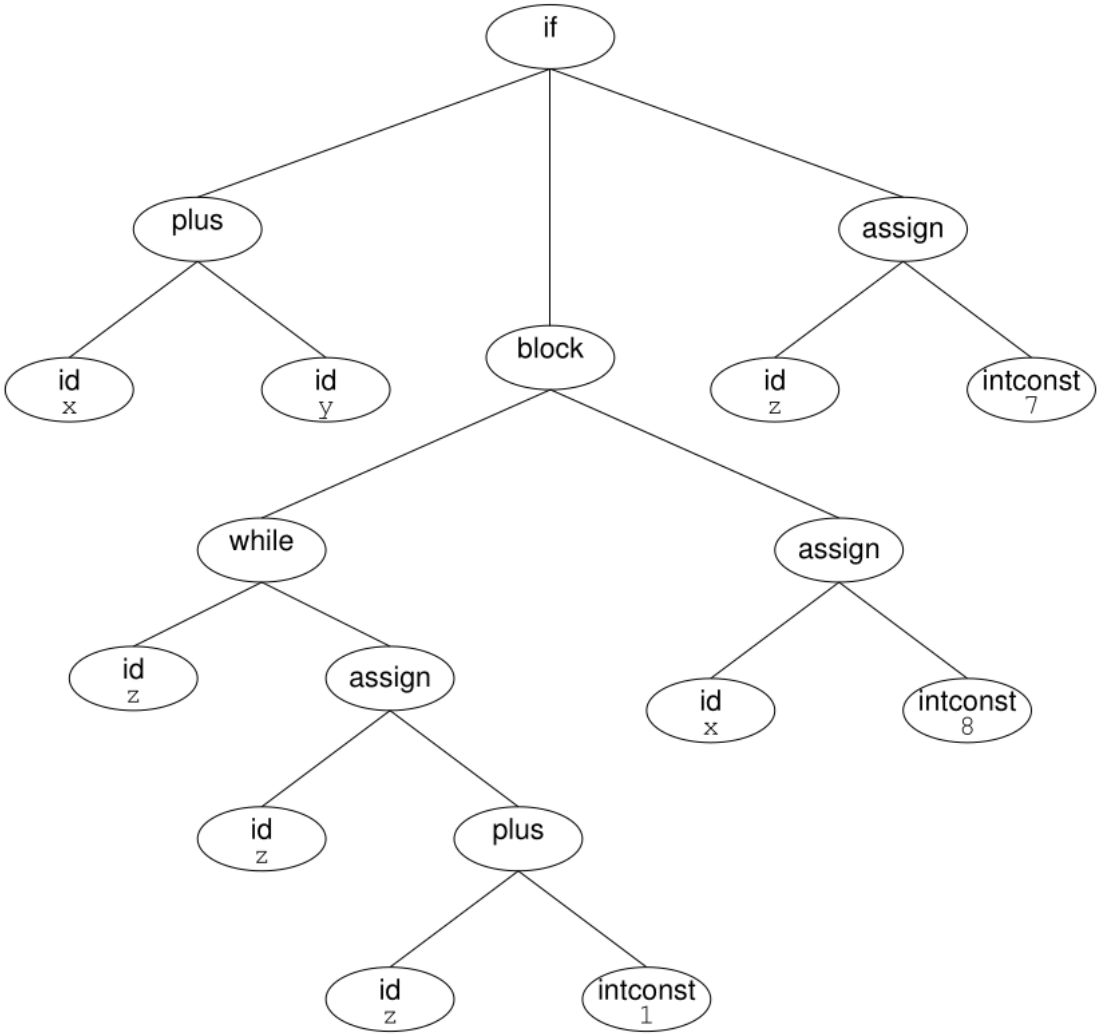
Árvores sintáticas concretas

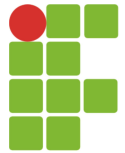


Árvores sintáticas abstras

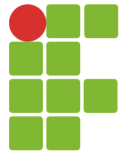
- A árvore sintática abstrata gerada durante o processo de análise sintática poderia eliminar algumas informações desnecessárias para as próximas etapas da compilação.
- A AST resultante vem a seguir.

Árvores sintáticas abstratas





Sumário



EDs para ASTs

Para projetar uma ED eficiente para ASTs, devemos considerar que:

- As ASTs são construídas tipicamente de baixo para cima. Uma lista de irmãos é gerada que só depois serão adotados por um nó pai.
- A lista de irmãos é gerada tipicamente por regras recursivas. A ED deve possibilitar a adoção de irmãos de maneira simples através das extremidades da lista.
- Alguns nós da AST possui um número fixo de nós, como por exemplo um nó que representa uma soma, que tem dois filhos. Contudo, Algumas linguagens de programação podem requerer uma AST que suporte nós com vários filhos. A ED deve suportar uma aridade arbitrária das ASTs.