

Texto auxiliar para Atividade Avaliativa #2: Analisador Léxico e Analisador Sintático

Pedro Santi Binotto [20200634]^{*1}, Eduardo Caigar Dudel [18206076]^{†2},
Arthur Alexandre Nascimento [18200410]^{‡3}, and Eduardo Sousa
Szczepaniak [20202478]^{§4}

¹Departamento de Informática e Estatística, Universidade Federal de
Santa Catarina

22 de julho de 2024

Resumo

Relatório do projeto prático da disciplina de *Introdução a Compiladores*.

^{*}pedro.binotto@grad.ufsc.br

[†]dudeleduardo@gmail.com

[‡]arthuralexnascimento@hotmail.com

[§]szcz.eduardo@gmail.com

Contents

Elaboração do trabalho	2
Normalização da gramática	3
Gramática original da linguagem	3
Gramática ajustada	4
Referências Bibliográficas	5

Elaboração do trabalho

O trabalho foi elaborado utilizando as ferramentas *Flex*[3] e *Bison*[1] para gerar os analisadores léxico e sintático, respectivamente. Além dos manuais das ferramentas, a apostila “*Lex & Yacc*”[2] foi outro recurso extensivamente consultado para utilização das tecnologias empregadas.

Para gerar o analisador sintático, uma descrição idêntica à especificação encontrada no documento da proposta do trabalho foi fornecida ao *Bison* para conversão.

Caso desejássemos construir uma *parse-table* LL(1), seria necessário adequar a escrita da sintaxe para possibilitar a análise *top-down sem backtracking*; entretanto, não é possível remover todos os casos de ambiguidades na gramática sem fazer alterações na especificação da linguagem, como a forma das cláusulas “*if...else*” é inerentemente ambígua.

Uma versão da gramática ajustada para resolver os casos de ambiguidade possíveis e livre de recursão à esquerda pode ser encontrada na seção “*Normalização da gramática*”.

A implementação final dos analisadores adiciona uma propriedade além do que descreve a especificação da gramática para possibilitar a escrita de comentários nos arquivos de código fonte:

```
/*
 * Texto dentro destes delimitadores não será interpretado.
 *
 * Palavras-chave e símbolos da linguagem também não produzem
 * efeito nenhum:
 *
 * def int print return () {} ;
 */

def main() {
    int A = calc_a();
    return;
}
```

Os arquivos de teste utilizam destes comentários como uma forma de registrar metadados sobre o caso de teste e saída esperada da execução da análise.

Normalização da gramática

Gramática original da linguagem

A seguir está a gramática original, como fornecida na proposta do trabalho, em notação BNF:

$\langle main \rangle ::= \langle stmt \rangle$ $\langle flist \rangle$ $\langle empty \rangle$	$\langle printst \rangle ::= \text{'print'} \langle expr \rangle$
$\langle flist \rangle ::= \langle fdef \rangle \langle flist \rangle$ $\langle fdef \rangle$	$\langle returnst \rangle ::= \text{'return'}$
$\langle fdef \rangle ::= \text{'def'} \text{'id'} \text{'('} \langle parlist \rangle \text{'')} \text{'{'}$ $\langle stmtlist \rangle \text{'}'}$	$\langle ifstmt \rangle ::= \text{'if'} \text{'('} \langle expr \rangle \text{'')} \langle stmt \rangle$ $\text{'else'} \langle stmt \rangle$ $\text{'if'} \text{'('} \langle expr \rangle \text{'')} \langle stmt \rangle$
$\langle parlist \rangle ::= \text{'int'} \text{'id'} \text{' ,' } \langle parlist \rangle$ $\text{'int'} \text{'id'}$ $\langle empty \rangle$	$\langle stmtlist \rangle ::= \langle stmt \rangle \langle stmtlist \rangle$ $\langle stmt \rangle$
$\langle stmt \rangle ::= \text{'int'} \text{'id'} \text{' ;'}$ $\langle atribst \rangle \text{' ;'}$ $\langle printst \rangle \text{' ;'}$ $\langle returnst \rangle \text{' ;'}$ $\langle ifstmt \rangle$ $\text{'{' } \langle stmtlist \rangle \text{'}'}$ ' ;'	$\langle expr \rangle ::= \langle numexpr \rangle \text{'<'} \langle numexpr \rangle$ $\langle numexpr \rangle \text{'>'} \langle numexpr \rangle$ $\langle numexpr \rangle \text{'==' } \langle numexpr \rangle$ $\langle numexpr \rangle$
$\langle atribst \rangle ::= \text{'id'} = \langle expr \rangle$ $\text{'id'} = \langle fcall \rangle$	$\langle numexpr \rangle ::= \langle numexpr \rangle \text{'+' } \langle term \rangle$ $\langle numexpr \rangle \text{'-' } \langle term \rangle$ $\langle term \rangle$
$\langle fcall \rangle ::= \text{'id'} \text{'('} \langle parlistcall \rangle \text{'')}$	$\langle term \rangle ::= \langle term \rangle \text{'*'} \langle factor \rangle$ $\langle factor \rangle$
$\langle parlistcall \rangle ::= \text{'id'} \text{' ,' } \langle parlistcall \rangle$ 'id' $\langle empty \rangle$	$\langle factor \rangle ::= \text{'num'}$ $\text{'('} \langle numexpr \rangle \text{'')}$ 'id'

Gramática ajustada

Gramática ajustada, livre de recursão à esquerda e corrigida para retificar todos os casos de ambiguidade que são possíveis; em notação BNF:

$\langle main \rangle ::= \langle stmt \rangle$	$\langle printst \rangle ::= \text{'print'} \langle expr \rangle$
$\langle flist \rangle$	$\langle returnst \rangle ::= \text{'return'}$
$\langle empty \rangle$	$\langle ifstmt \rangle ::= \text{'if' '(' } \langle expr \rangle \text{' ')} \langle stmt \rangle$
$\langle flist \rangle ::= \langle fdef \rangle \langle flist' \rangle$	$\langle ifstmt' \rangle ::= \text{'else' } \langle stmt \rangle$
$\langle flist' \rangle ::= \langle flist \rangle$	$\langle empty \rangle$
$\langle empty \rangle$	$\langle stmtlist \rangle ::= \langle stmt \rangle \langle stmtlist' \rangle$
$\langle fdef \rangle ::= \text{'def' 'id' '(' } \langle parlist \rangle \text{' ')} \text{'{'}$	$\langle stmtlist' \rangle ::= \langle stmtlist \rangle$
$\langle parlist \rangle ::= \text{'int' 'id' } \langle parlist' \rangle$	$\langle empty \rangle$
$\langle parlist' \rangle ::= \text{' , ' } \langle parlist \rangle$	$\langle expr \rangle ::= \langle numexpr \rangle \langle expr' \rangle$
$\langle empty \rangle$	$\langle expr' \rangle ::= \text{'<' } \langle numexpr \rangle$
$\langle stmt \rangle ::= \text{'int' 'id' ';'}$	$\text{'>' } \langle numexpr \rangle$
$\langle atribst \rangle \text{' ; '}$	$\text{'==' } \langle numexpr \rangle$
$\langle printst \rangle \text{' ; '}$	$\langle empty \rangle$
$\langle returnst \rangle \text{' ; '}$	$\langle numexpr \rangle ::= \langle term \rangle \langle numexpr' \rangle$
$\langle ifstmt \rangle$	$\langle numexpr' \rangle ::= \text{'+' } \langle term \rangle \langle numexpr' \rangle$
$\text{'{' } \langle stmtlist \rangle \text{'}'}$	$\text{'-' } \langle term \rangle \langle numexpr' \rangle$
' ; '	$\langle empty \rangle$
$\langle atribst \rangle ::= \text{'id' '=' } \langle atribst' \rangle$	$\langle term \rangle ::= \langle factor \rangle \langle term' \rangle$
$\langle atribst' \rangle ::= \langle expr \rangle$	$\langle term' \rangle ::= \text{'*'} \langle factor \rangle \langle term' \rangle$
$\langle fcall \rangle$	$\langle empty \rangle$
$\langle fcall \rangle ::= \text{'id' '(' } \langle parlistcall \rangle \text{' ')} \text{' ; '}$	$\langle factor \rangle ::= \text{'num'}$
$\langle parlistcall \rangle ::= \text{'id' } \langle parlistcall' \rangle$	$\text{'(' } \langle numexpr \rangle \text{' ')} \text{' ; '}$
$\langle parlistcall' \rangle ::= \text{' , ' } \langle parlistcall \rangle$	'id'
$\langle empty \rangle$	

Referências Bibliográficas

References

- [1] Charles Donnelly and Richard Stallman. Bison - the yacc-compatible parser generator. *Free Software Foundation*, 1992. Accessed: 2024-07-14.
- [2] Tom Niemann. Lex & yacc. <https://epaperpress.com/lexandyacc/>. Accessed: 2024-07-14.
- [3] Vern Paxson. Using flex - a fast lexical analyzer generator. *The Regents of the University of California*, 1990. Accessed: 2024-07-14.