Wrocław University
of Science and Technology

Faculty of Computer Science and Management
Field of study: Computer Science
Specialty: —

Engineering Thesis

FORMAL GRAMMAR PRODUCTION RULE PARSING TOOL

Karol Belina

keywords:
Parser combinators, context-free grammars, Extended Backus-Naur Form

short summary:

Supervisor	dr inż. Zdzisław Spławski	.....	.....
	Title/degree/name and surname	grade	signature

The final evaluation of the thesis

Head of the examination commission	.....	.....	.....
	Title/degree/name and surname	grade	signature

*For the purposes of archival thesis qualified to:**

a) category A (perpetual files)

b) category BE 50 (subject to expertise after 50 years)

** delete as appropriate*

stamp of the faculty

Wrocław 2020

Abstract

The thesis presents the design and implementation of an EBNF-based context-free grammar parsing tool with real-time explanations and error detection. For this purpose, the official specification of the Extended Backus-Naur Form from the ISO/IEC 14977 standard has been examined and transformed to an unambiguous and ready for implementation form. The thesis proposes a definition of a grammar in the form of an abstract syntax tree. It describes the process of tokenization — the act of dividing the grammar in a textual form into a sequence of tokens — while taking into account proper interpretation of Unicode graphemes. The whitespace-agnostic tokens are then being combined together to form a previously-defined AST with a technique called *parser combination*. A number of smaller helper parsers are defined, all of which are then combined into more sophisticated parsers capable of parsing entire terms, productions and grammars. [**TODO** *coś o regexach w specjalnych sekwencjach?*] The paper defines an algorithm for handling left recursion in the resulting grammar defined by an AST, as well as a dependency graph reduction algorithm for determining the starting rule of a grammar. Up to this stage, any errors encountered in the textual form of a grammar are reported to the user in a user-friendly format with exact locations of the errors in the input. The paper thus compares several techniques of storing the locations of individual tokens and AST nodes for the purposes of error reporting. Further, the thesis describes a method of testing an arbitrary input against the constructed grammar to determine if it belongs to the language generated by that grammar. [**TODO** *tutaj prawdopodobnie coś o wyjaśnieniach zwracanych przez checker*] The thesis describes the process of creating a simple command line REPL program to act as a basic tool for interfacing with the grammar parser and checker, but in order to efficiently use the library, a web-based application is designed on top of that to serve as a more visual, user-friendly and easily accessible tool. [**TODO** *tutaj coś o wizualizacjach, edytorze tesktowym i highlightowaniu*] The paper describes the deployment of the application on a static site hosting service, as well as a cross-platform desktop application with the use of Electron. The designed and implemented system gives the opportunity to extend it with other grammar specifications. [**TODO** *poparafrazować "The thesis describes..."*]

Contents

Introduction

1. Problem analysis

1.1. Description

1.2. Motivation

1.3. Goal

1.4. Scope

2. Analysis of similar solutions

3. Theoretical preliminaries

3.1. Context-free grammars

3.2. Specification

3.3. Grammar definition

3.4. Tokenization

3.5. Parsing

3.5.1. Methods

3.5.2. Parser combination

3.5.3. Parser definitions

3.6. Grammar preprocessing

3.6.1. Left recursion handling

3.6.2. Dependency graph reduction

3.7. Grammar processing

4. Design

4.1. Requirements

4.1.1. Functional requirements

4.1.2. Non-functional requirements

4.2. Use cases

4.3. The architecture

4.4. Interface prototype

5. Implementation

5.1. Environment

5.2. Business logic

5.2.1. Lexer

5.2.2. Parser

5.2.3. Preprocessor

5.2.4. Checker

5.3. Command line application

5.4. Web-based application

5.4.1. Linking the business logic

5.4.2. Text editor

5.4.3. Visualizations

6. Testing

6.1. Automated testing

6.1.1. Business logic testing

6.1.2. UI testing

6.2. Manual testing

7. Deployment

7.1. GitHub Pages

7.2. Electron

Summary

Bibliography

[1] *Information technology, syntactic metalanguage, extended BNF*. ISO/IEC, 1996.

List of Figures

List of Tables

List of Listings

A. Modified specification

```
character
  = ? any Unicode non-control character ?;
letter
  = ? any Unicode alphabetic character ?;
digit
  = ? any Unicode numeric character ?;
whitespace
  = ? any Unicode whitespace character ?;
comment
  = '(*', {comment | character}, '*)';
gap
  = (whitespace | comment), {whitespace}, {{comment}, {whitespace}};
identifier
  = letter, {{whitespace}, letter | digit};
factor
  = [[gap], digit, {{whitespace}, digit}, [gap], '*'],
    [gap], [(identifier
      | ('[' | '(/'), alternative, (']' | '/)')
      | ('{' | '(:'), alternative, ('}' | ':)')
      | '(', alternative, ')'
      | "'", character - "'", {character - "'"}, "'"
      | '"', character - '"', {character - '"'}, '"'
      | '?', {{whitespace}, character - '?'}, '?'), [gap]];
term
  = factor,
    ['-', ? a factor that could be replaced
      by a factor containing no identifiers ?];
sequence
  = term, {',', term};
alternative
  = sequence, {('|' | '/' | '!'), sequence};
production
  = [gap], identifier, [gap], '=', alternative, (';' | '.'), [gap];
grammar
  = production, {production};
```

Listing A.1: Modified version of the EBNF language specification defined in [1]