Cutting Languages Down to Size

Student Project

at the Cooperative State University Baden-Württemberg Stuttgart

by

Nahku Saidy and Hanna Siegfried

08.06.2020

Time of Project	nothing
Student ID; Course	8540946, 6430174; TINF17ITA
Advisors	Prof. Dr. Stephan Schulz and Geoff Sutcliffe

Contents

Acronyms

BNF	Backus-Naur Form
CNF	First-Order Clause Normal Dorm
FOF	Full First-Order Logic
TFF	Typed First-Order Logic
THF	Typed Higher-Order Logic
TPTP	Thousands of Problems for Theorem Provers

List of Figures

List of Tables

Listings

1 Introduction

1.1 Problem Statement and Goals

Formal languages are likely to grow over time as they are getting more complex when their functionality is extended and more application cases are covered. On the one hand that leads to a more powerful language. However, on the other hand it becomes harder to understand the language and to implement it. Thus, it becomes harder for new users to use the language.
This problem can be addressed by dividing languages into smaller sub-languages that cover everything relevant to the specific use case. This could be done manually, but using this method is likely to raise errors or divergences from the original grammar.
Therefore, the approach considered in this report is to develop an application that is able to automatically extract sub-languages from a language. A sub-language should be specified by the user using the application.
This report focusses on the Thousands of Problems for Theorem Provers (TPTP) language for automated theorem proving. Sub-languages of interest are for example a grammar just for First-Order Clause Normal Dorm (CNF) or Full First-Order Logic (FOF) The grammar of the language is provided in an extended Backus-Naur Form (BNF).
The first step to divide the TPTP language in smaller sub-languages is to build a parser that parses the grammar of the TPTP language. The parser should build a parse tree that represents the grammar rules of the TPTP language. This parse tree should be visually presented to the user and the user can then choose which grammar rules should not be included in the desired sub-language. After the user specified the sub-language, the developed application should extract the sub-language from the TPTP language and present the sub-language in the same format as the original TPTP syntax. Also, comments present in the TPTP syntax should be maintained and associated with the corresponing rules in the reduced syntax.

1.2 Structure of the Report

2 Background and Theory

2.1 TPTP Language

[1]

2.2 Backus-Naur Form (BNF)

2.3 Parser

2.3.1 Lex

Lexing/lexical analysis: Division of input into units so called tokens [2]

Input: description of tokens - lex specification, regular expressions][2] Output: routine that identifies those tokens [2]

2.3.2 Yacc

Parsing: establish relationship among tokens [2] Grammar: list of rules that defines the relationships [2]

Input: description of grammar [2] Output: parser [2]

2.3.3 PLY

Python implementation of lex and yacc [LALR-parsing] consists of lex.py and yacc.py

lex.py tokenizes an input string

http://www.dabeaz.com/ply/ply.html

3 Concept

4 Implementation

5 Validation

6 Conclusion

Outlook

Bibliography

Publikationen

[1] G. Sutcliffe. “The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0”. In: *Journal of Automated Reasoning* 59.4 (2017), pp. 483–502.

[2] John Levine, Tony Mason, and Doug Brown. *Lex & Yacc*. O’Reilly Media Inc., 1992. ISBN: 9781565920002.