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, XXXXX; TINF17ITA
Advisors	Prof. Dr. Stephan Schulz and Geoff Sutcliffe

Contents

Acronyms

BNF	Backus-Naur form
CNF	First order clause normal form
FOF	Full first order logic
TFF	Typed first order logic
THF	Typed high order logic
TPTP	Thousands of Problems for Theorem Provers

List of Figures

List of Tables

Listings

1 Introduction

1.1 Problem Statement and Goals

Programming languages and other formal languages (e.g. data representation languages like JSON) are often described via a context-free formal grammar, e.g. in the Extended Backus–Naur form that can be automatically processed by tools like Yacc and Bison. These languages tend to grow over time, as more features are added and the language covers more application cases. On the one hand, this makes the language more powerful. But on the other hand, this often makes the language harder to understand and harder to implement, increasing the barrier to entry for new developers.

This problem can, of course, be addressed by manually maintaining simplified or partial grammars. However, this creates not only a maintenance overhead, but may also lead to undesired and even unnoticed divergences of the full and the simplified grammar. Instead, we propose to use automatically extraction of self-contained parts of a formal grammar to create simpler sub-languages.

Our particular use case is the TPTP syntax for automated theorem proving tools. TPTP defines a family of languages, from pure first-order clause normal form (CNF) and full first order (FOF) via typed first-order (TFF), eventually to typed higher-order logic (THF). The language can be used to write both input problems (i.e. lists of logical formulae), but also derivations (where some of the formulas are input formulas, and the others are justified by reference to existing formulas and some mechanism for deriving the former from the latter. Most of the extensions are modular and even conservative (i.e. FOF is mostly a superset of CNF, and normally FOF is used as shorthand for CNF+FOF), and we are interested in e.g. extracting just a grammar for CNF or CNF+FOF. We are also interested extracting languages that do not allow certain features, e.g. a language in which only the pure input format in specified, not logical derivations. The task of this thesis is the design of a tool that performs this extraction and presents the result in a format that is both machine readable, but also friendly to use for human users.

One approach to this would be to a) select a given non-terminal node as the start symbol for the new grammar and b) block the undesired productions starting at certain non-terminal symbols. The tool can then start at the selected new start symbol and follow all legal transition to collect the remaining reachable grammar. It can also clean-up the resulting grammar by inlining rules for non-terminals with only a single production. An outstanding solution would also heuristically associate comments with grammar productions, and keep the comments referring to the remaining productions in place.

!!! The approach is to first build a parser that parses the BNF of the Thousands of Problems for Theorem Provers (TPTP) language. The parser builds a data structure of the TPTP language that shows all existing grammar rules. This data structure is presented to the user and the user is able to choose which specific grammar rules should be included in his sub language. After the user specified the sub language, the developed application is able to substract the defined sub language out of the TPTP language. Finally, the application presents the resulting language FOR EXAMPLE in the BNF. !!!

There are several possible tools with different levels of convenience for the user:

A basic command line tool which accepts a grammar file and a control file listing blocked productions, and that then produces the reduced grammar in the form of a new grammar file. In agile terms, this would be the minimal viable product. The same, with maintaining of comments. The same, but with (optional) inlining of single-alternative rules. A visualization tool (e.g. based on Graphviz) visualizing the full and the reduced grammar. A GUI tool allowing the user to interactively select which productions should be blocked and dynamically visualizing the resulting new grammar. Preferably, this would use the command line version in the background and be able to generate and export not only the reduced grammar, but also the control file.

1.2 Structure of the Report

2 Background and Theory

2.1 TPTP Language

2.2 Backus-Naur form (BNF)

2.3 Parser

2.3.1 Lex

Lexing/lexical analysis: Division of input into units so called tokens [1][1]

Input: description of tokens - lex specification, regular expressions [1][1] Output: routine that identifies those tokens [1][1]

2.3.2 Yacc

Parsing: establish relationship among tokens [1][1] Grammar: list of rules that defines the relationships [1][1]

Input: description of grammar [1][1] Output: parser [1][1]

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