DHBW
Duale Hochschule
Baden-Württemberg

Implementing a graph-based clause-selection strategy for Automatic Theorem Proving in Python

from the course of studies Computer Science

at the Cooperative State University Baden-Württemberg Stuttgart

by

Jannis Gehring

03/13/2025

Time frame:	09/30/2024 - 06/12/2025
Student ID, Course:	6732014, TINF22B
Supervisor at DHBW:	Prof. Dr. Stephan Schulz

Declaration of Authorship

Gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden- Württemberg vom 29.09.2017. Ich versichere hiermit, dass ich meine Arbeit mit dem Thema:

Implementing a graph-based clause-selection strategy for Automatic Theorem Proving in Python

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass alle eingereichten Fassungen übereinstimmen.

Stuttgart, 03/13/2025

Jannis Gehring

Table of Contents

List of Acronyms

FOL	First order Logic
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer

Glossary

Exploit An exploit is a method or piece of code that takes advantage of vulnerabilities in software, applications, networks, operating systems, or hardware, typically for malicious purposes.

Patch A patch is data that is intended to be used to modify an existing software resource such as a program or a file, often to fix bugs and security vulnerabilities.

Vulnerability A Vulnerability is a flaw in a computer system that weakens the overall security of the system.

1 Introduction

2 Theory

2.1 First order Logic (FOL)

This chapter contains the underlying first-order predicate logic definitions and a short introduction to binary resolution. We start from the basic context of predicate logic.

Term constitute elements of the corresponding domain and consist of variables, functions and constants. *Variables* are denoted with the letters $x, y, z, x_1, y_2, \ldots$. *Functions* are denoted with the letters $f, g, h, \ldots$. A function of the form $f(t_1, t_2, ..., t_n)$ (with t being terms) has an associated arity of n . *Constants* are denoted with the letters $a, b, c, \ldots$ and represent a special case of functions with arity 0.

Predicates are denoted with $P, Q, R, \ldots$. They map (tuples of) terms to boolean truth values. The concept of function-arity is extended to predicates accordingly. An *atom* consists of a predicate P and the corresponding terms. A *formula* is either an atom or the combination of atoms with logical operators ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$) or quantors ($\forall, \exists$).

A *literal* is either an atom or a negated atom. A *clause* is a set of literals interpreted as the conjunction of those.

2.2 Current clause selection strategies

3 PyRes

PyRes is an open-source theorem prover for first-order logic. Its name originates from **Py**thon, the programming language it is built with, and the **Res**olution calculus, which it implements for solving FOL problems.

3.1 Python

Python is an interpreted high-level programming language. It supports multiple programming paradigms like functional programming and object orientation. Python was created by Guido van Rossum in the early 1990s [1]. Not only is Python easy to learn and read, it also has a lot of packages like Numpy for efficient numerical computations, Pandas for manipulating big datasets, Matplotlib for plotting or TensorFlow and PyTorch for machine learning. This is why it is still among the most used programming languages.

3.2 PyRes in constrast to other theorem provers

A lot of modern theorem provers, i.e. E, Vampire and SPASS, are built with low-level languages like C and C++ ([2], [3], [4]). They employ optimized data structures and complex algorithms to increase their performance. Other provers like iprover [5] are implemented in lesser-known languages like OCamL. Both approaches make it harder for new developers to understand the codebase and functionality, hence hindering further developement. This also leaves the didactic potential of theorem provers unused.

PyRes, on the other hand, is explicitly built for readability. Extensive documentation and the choice of an interpreted language enable a step-by-step understanding of the functionality. Its architecture and calculus is similar to other theorem provers, making it a suitable entry for understanding a multitude of provers. [6]

Whilst PyRes doesn't have as much extensions for optimization than other provers, this simplicity makes it a good candidate for implementing and evaluating new approaches (like alternating path theory in this case).

3.3 PyRes architecture

PyRes heavily builds upon Python's support for object orientation

- Lexer
  - Token
- Term
- Literal
- Clause
- Formula

References

[1] “Python - History and License.”

[2] S. Schulz, “eprover.” GitHub, 2024.

[3] “vampire.” [Online]. Available: https://github.com/vprover/vampire

[4] “tspass.” [Online]. Available: https://github.com/michel-ludwig/tspass

[5] “iprover.” [Online]. Available: https://github.com/edechter/iprover

[6] S. Schulz and A. Pease, “Teaching Automated Theorem Proving by Example: PyRes 1.2,” in *Automated Reasoning*, N. Peltier and V. Sofronie-Stokkermans, Eds., Cham: Springer International Publishing, 2020, pp. 158–166.