Learning Theorem Proving by Example - Implementing JavaRes

Adam Pease[1] and Stephan Schulz[2]

[1] Articulate Software, USA, apease@articulatesoftware.com
[2] DHBW Stuttgart, Germany, schulz@eprover.org

Abstract. Did PyRes [15] achieve its goal of being a sufficient model for learning about how to implement a first-order ATP system? JavaRes is a demonstration prover patterned after PyRes. In this paper we discuss the architecture and data structures of this prover and the experience of one of us implementing the prover, without prior expertise in writing an automated theorem prover prover. We present performance measurements relative to PyRes and other systems. To illustrate the value of JavaRes for learning about theorem proving we also mention the implementation of several features beyond the original PyRes concept.

1 Introduction

Automated theorem proving is a fascinating and useful discipline, but can be mystifying for someone not deeply acquainted with the field. It is fair to say that most computer science professionals do not understand the power of inference in first-order logic (FOL), and how it provides distinct capabilities relative to simpler representations such as graphs or description logics. Part of the reason for this lack of general familiarity may be because the barrier to entry for the field remains high, despite many decades of work and publication. Most publications require a degree of mathematical sophistication to understand, and even with such capability, a reader will not know which data structures to use, or which of the many algorithms will be simplest or best to implement.

We initially began with just an attempt for one of us (Pease) to learn about automated Theorem Proving (ATP), motivated by decades of work in formal ontology [6,8], and finding that among many excellent books, including [3], the first steps to understand ATP were too difficult. Fortunately, the creator of the prover **E** [13,14] (Schulz), provided the explanations needed to understand how and where to begin. This grew into the creation of PyRes.

PyRes is a simple, resolution-based theorem prover. It implements the basic calculus from Robinson's seminal paper [11], extended with negative literal selection and some redundancy elimination as described by Bachmair and Ganziner [1]. The core is a given-clause based clausal saturation algorithm. The system also supports full first-order input via clausification, and equality handling via automatic addition of equality axioms.

To see whether PyRes provided enough structure to learn how to write a theorem prover, we developed JavaRes, a prover modelled on PyRes, but written

in Java. We also used JavaRes as a basis to show implementation of several extensions to a baisc prover. We believe that the lessons learned in the overall process, now incorporated into the two provers, provide a suitable platform for ATP education.

JavaRes is open source and all code and data is available at `https://github.com/ontologyportal/JavaRes`.

2 Programming Language and Software Engineering Considerations

One lesson learned is that with code as complex as theorem proving code, unless there is full understanding of the algorithm and expected results, catching problems can be very challenging. Minor coding errors can be magnified since it is hard to know where to look in system output for problems, and system output can be very large in a combinatorially explosive search space. Several such issues occurred in the construction of JavaRes. In one case literals were not initialized as to whether literal selection had considered them suitable for inference. Copies were created of literals that included their literal selection flag, rather than reinitializing them by default as true. This problem didn't cause any errors, just failure to prove certain theorems (but not others) and therefore wasn't obvious in simple examples and unit tests. Another problem resulted from a typo in the subsumption code, where the variable `'subsumed'` was mistyped as `'subsumer'`. Even meaningful variable names can be a problem if they are very similar to others. A more serious case of a variable naming issue that was present in a test case was having variables named l1, ll and ll1 which could be easily confused in most fonts.

We started work on JavaRes at CADE in 2011 and mostly kept the Java and Python versions in sync, working one day a week together for a period of a year. Then there was a gap of 8 years and a final push to bring JavaRes up to date with PyRes, consisting of two months of half-time effort for one of us, plus occasional coaching and some intensive debugging at the end. This experience indicates that implementing a basic prover patterned after PyRes should be feasible, at least in terms of the time commitment involved, as a graduate semester project after an introductory course in logic.

The code base of JavaRes is significantly larger (measured in lines of code) than that of PyRes.Java is more verbose than Python. Verbosity is a double-edged sword. Short code is easier to read and to understand, as long as it is not cryptic. On the other hand more verbose code may be more self documenting. Efforts to teach ATP from these two systems may tell us which is better for students.

While PyRes is intended as a fairly minimal example of an ATP system, we use JavaRes as a springboard for implementing additional features and alternative algorithms.

As a result of these two issues, JavaRes is 19,334 total lines of code, versus 8553 lines for PyRes (including comments, docstrings, and unit tests). If we

only count actual production code, there are 7508 lines of effective code for JavaRes and only 3681 lines of effective code in PyRes. It is likely that after this experience, refactoring and reimplementation could reduce this discrepancy, which points simply to inexperience being at least partly the cause of more verbose and less elegant code.

2.1 Clause Selection, Indexing, Heuristics, SInE and other Features

JavaRes includes all the optimization strategies in PyRes. For clause selection it implements two methods, which can be combined. The most basic is a first-in-first-out (FIFO) strategy that will eventually try every clause. However, this is rarely optimal. A symbol-counting strategy picks the clause with the fewest symbols. This is often a good strategy but in many cases it will fail because larger clauses may never be considered. The most successful simple strategy turns out to be a combination of the two, where the smallest symbol count is tried five times and then FIFO is tried once. This results in a strong bias to smaller clauses while ensuring that all clauses will eventually be tried. JavaRes also supports a more conservative stregy of two tries with the smallest symbol for every one try of the FIFO stack. Other combinations are possible with a small change to the code.

JavaRes supports indexing for subsumption and resolution. Subsumption removes clauses from the set of clauses to be processed (called "forward subsumption") and from the set already processed ("backwards subsumption") thereby decreasing the problem search space. More general clauses subsume more specific ones.

Indexing employs records with signs and predicate symbols only, so that potential clauses can be accepted or rejected more rapidly than attempting unification. In resolution, literals of opposite sign may resolve, as opposed to subsumption in which those of the same sign potentially subsume. Resolution and subsumption therefore have separate indexes.

JavaRes also implements PyRes' approach to literal selection. A naive approach to resolution has to compare every literal in a resolver to every literal in the proposed resolvant to see if they unify but have opposite signs. Optimizations other than this exhaustive approach can have an impact. We implement five strategies: choose simply the first literal, the largest literal, the smallest number of constants and variables, the smallest number of variables and finally, a combined strategy that picks an equation of two variables, if present, or the smallest number of variables or if those are equal then the largest number of symbols. Largest literal selection is the default strategy.

For large theories, JavaRes has implemented the SInE algorithm[4] that has proven to be the dominant approach in the LTB division of the CASC competition.

JavaRes can also parse the first order portion of SUO-KIF syntax used with the SUMO knowledge base that is arguably a much more friendly syntax for theory authoring, especially for complex or nested axioms, than TPTP. The

JavaRes data structures used for ATP with the different surface syntax of SUMO are identical with TPTP.

3 Testing and Examples

It is very valuable to have many examples. When examples are implemented as unit tests they catch bugs as well as explain how the algorithms are supposed to work at each stage. The example provide a sense of purpose for each class, and are as valuable as the algorithms themselves. Often, just having a clear set of examples is sufficient to code at least some version of the algorithm needed. Adding more tests and examples has been a key benefit of the implementation of JavaRes, as it showed what obscure bugs might appear, or what non-obvious errors might exist. For example, in an early version of unification, the implementation failed to consider all possible options, just returning a list of one set of substitutions, rather than all possible substitution. This problem didn't surface in testing until much later, but was easily explained with an example and test that should ensure that no future implementer will move forward with an implementation while being unaware of this issue, should it arise. Java tests are implemented in the jUnit framework. Whereas Python coders add tests at the end of each class, in Java tests are separated into their own classes.

One of the lessons from implementing JavaRes is that one can never have too many tests. A number of bugs, mostly from typographical errors, were not evident in the unit tests and only appeared once tested on some of the larger problems in the TPTP. Even smaller TPTP problems could lead to false confidence, since several successful paths to a contradiction often exist and only on problems that are challenging will the absence of a particular component of proving, such as forward subsumption, be a critical issue.

We present the results in Table 1 for different problem classes within TPTP: UEQ (unit problems with equality), CNE (clausal problems without equality), CEQ (clausal problem with equality, but excluding UEQ), FNE (FOF problems without equality) and FEQ (FOF problems with equality). Readers interested in performance of the systems in other than their best configuration can refer to our previous PyRes paper and we only present the systems here with their performance including indexing, subsumption and best clause selection strategy.

As with our earlier experiments with PyRes, testing was on StarExec Miami, a spin-off of the original StarExec project [16]. StarExec proved to be an essential resource for scaling up testing as it is not practical to run all ˜24,000 available tests for 300 seconds each on a laptop, or to load it so heavily with theorem proving that no other work can be done. Even so, running an experiment of this sort can take 24 hours of clock time, unless there is no other load on the system. It was tempting, too early in the development process, to run the full set of TPTP tests. After several abortive attempts, a much smaller set of problem tests was used to create an integration test suite that was more comprehensive that unit tests, nearly as fast, and certainly much faster than an indiscriminate run on all available TPTP problems.

The StarExec machines were equipped with 256 GB of RAM and Intel Xeon CPUs running at 3.20 GHz. The per-problem time-limit was set to 300 seconds. TPTP 7.4 was used but only on problems also present in 7.2, to allow for continuity with the measurements in our earlier paper on PyRes.

JavaRes begins to approach the performance of Prover9 on problems without equality. It is markedly inferior to E and especially so on problems with equality. Implementing the JavaRes/PyRes approach to ATP in C or C++ would likely result in further speed improvements.

Category	**UEQ**	**CNE**	**CEQ**	**FNE**	**FEQ**	**All**
Class size	(1193)	(2383)	(4442)	(1771)	(6305)	(16094)
PyRes	113	945	499	632	725	2914
JavaRes	148	1107	644	737	1617	4253
E 2.4	813	1939	2648	1484	4054	10938
Prover9-1109a	728	1316	1678	709	2001	6432
LeanCoP 2.2	6	0	0	969	1826	2801

Table 1. JavaRes and PyRes problem correctness (E, Prover9 and leanCoP for comparison)

4 Conclusion and Future Work

The experience of JavaRes has shown that it is possible to use PyRes as the basis for a new theorem prover implementation in about three person months of work, albeit with some coaching, and for someone already familiar with TPTP, as a user of TPTP and with FOL generally. The experience has shown us how to improve the clarity of the code, and expand code comments and unit tests.

With an implementation in Java, it should be easier to integrate the prover with existing tools to support special cases that apply to SUMO development, since its existing Sigma tool set [10] and SUMOjEdit editor [9] are also written in Java.

Acknowledgement: Thanks to Geoff Sutcliffe for his work on, and advice about, StarExec and TPTP, which allowed us to test our work.

References

1. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of automated reasoning, vol. I, chap. 2, pp. 19–99. Elsevier (2001)
2. Chalupsky, H., Russ, T.A.: Whynot: Debugging failed queries in large knowledge bases. In: Proceedings of the 14th Conference on Innovative Applications of Artificial Intelligence - Volume 1. p. 870–877. IAAI'02, AAAI Press (2002)
3. Harrison, J.: Handbook of Practical Logic and Automated Reasoning. Cambridge University Press, New York, NY, USA, 1st edn. (2009)

4. Hoder, K., Voronkov, A.: Sine Qua Non for Large Theory Reasoning. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) Proc. of the 23rd CADE, Wroclav, Poland. LNAI, vol. 6803, pp. 299–314. Springer (2011)
5. Löchner, B., Schulz, S.: An Evaluation of Shared Rewriting. In: de Nivelle, H., Schulz, S. (eds.) Proc. of the 2nd International Workshop on the Implementation of Logics. pp. 33–48. MPI Preprint, Max-Planck-Institut für Informatik, Saarbrücken (2001)
6. Niles, I., Pease, A.: Toward a Standard Upper Ontology. In: Welty, C., Smith, B. (eds.) Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001). pp. 2–9 (2001)
7. Nonnengart, A., Weidenbach, C.: Computing Small Clause Normal Forms. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, chap. 5, pp. 335–367. Elsevier Science and MIT Press (2001)
8. Pease, A.: Ontology: A Practical Guide. Articulate Software Press, Angwin, CA (2011)
9. Pease, A.: A programmer's text editor for a logical theory: The sumojedit editor (system description). In: International Joint Conference on Automated Reasoning. pp. 472–479. Springer, Cham (2020)
10. Pease, A., Benzmüller, C.: Sigma: An integrated development environment for formal ontology. AI Communications **26**(1), 79–97 (2013)
11. Robinson, J.A.: A Machine-Oriented Logic Based on the Resolution Principle. Journal of the ACM **12**(1), 23–41 (1965)
12. Russel, S.J., Norvig, P.: Artificial Intelligence – A Modern Approach. Prentice Hall Series in Artificial Intelligence, Prentice Hall International (1995)
13. Schulz, S.: E – A Brainiac Theorem Prover. Journal of AI Communications **15**(2/3), 111–126 (2002)
14. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, higher, stronger: E 2.3. In: Fontaine, P. (ed.) Proc. of the 27th CADE, Natal, Brasil. pp. 495–507. No. 11716 in LNAI, Springer (2019)
15. Schulz, S., Pease, A.: Teaching automated theorem proving by example: PyRes 1.2 (system description). In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Proc. of the 10th IJCAR, Paris. LNCS, vol. 12167, pp. 158–166. Springer (2020)
16. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: A Cross-Community Infrastructure for Logic Solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Proc. of the 7th IJCAR, Vienna. LNCS, vol. 8562, pp. 367–373. Springer (2014)
17. Suda, M., Sutcliffe, G., Wischnewski, P., Lamotte-Schubert, M., de Melo, G.: External sources of axioms in automated theorem proving. In: Mertsching, B., Hund, M., Aziz, M.Z. (eds.) KI 2009: Advances in Artificial Intelligence, 32nd Annual German Conference on AI, Paderborn, Germany, September 15-18, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5803, pp. 281–288. Springer (2009). https://doi.org/10.1007/978-3-642-04617-9_36, `https://doi.org/10.1007/978-3-642-04617-9_36`