Learning Abstractions for Guiding Equational Proof Search

Guy Axelrod[1,2], Moa Johansson[1,2], and Nicholas Smallbone[1,2]

[1] Chalmers University of Technology, Gothenburg, Sweden
[2] University of Gothenburg, Gothenburg, Sweden

Abstract

Completion-based equational theorem provers rely on lightweight heuristics for selecting critical pairs. We shall report on Twitch, a system that symbolically learns recurring term patterns from existing proofs and uses them to guide the equational prover Twee. These abstractions are discovered by proof-term compression using the library-learning tool Stitch and are used to influence the critical-pair weighting heuristic. Experiments on TPTP unit equality problems show that learned abstractions can speed up proof search and help solve hard problems. Details can be found in the preprint [1] – accepted for IJCAR 2026, forthcoming. We plan to present and outline future work on abstraction-aware critical-pair selection, better abstraction sources, neural abstraction generation, and equational hints. We will also mention ongoing work on applications of our abstraction approach in the context of proof refactoring and shortening.

Motivation

Completion-based provers such as Twee [5] derive rewrite rules from critical pairs. The number of possible inferences is enormous, so success depends heavily on which critical pairs are selected. Classical systems such as Otter and Prover9 address this problem through weighting [4], hints [7, 8], or resonators [9]: patterns describing terms or clauses that should be treated as promising.

Our aim is to discover such guidance automatically from proofs of related problems, in the context of equational reasoning. If a term shape occurs repeatedly in proofs from a domain, then future critical pairs containing that shape may be worth prioritising. We call these recurring term patterns *abstractions*. For instance, in the Sheffer-stroke presentations of lattice problems, the pattern $f(x, x)$ is often rediscovered; unsurprising since, semantically, this corresponds to negation.

Learning and Using Abstractions

Twitch learns abstractions using Stitch a library-learning tool that finds compressive function definitions in collections of Lisp-like programs [2]. Given a Twee proof of an equation, we extract the terms occurring in the axiomatic equality chain and translate them into the input language expected by Stitch, which produces abstractions that compress this term collection. Those abstractions that can be read back as first-order term patterns are translated back and used to guide Twee.

Twitch uses two sources of proof data. First, from a failed proof attempt, it extracts derived equations that look like promising lemmas, preferring simple statements with comparatively long derivations, and runs Stitch on the proof terms of the highest-ranked lemmas. Second, from successful proofs of related problems, it extracts local abstractions, keeps those that empirically speed up reproving their source problems, and compresses them again to obtain *domain abstractions*.

A learned abstraction could be added as a definitional axiom, but this changes the search space. Instead, we accommodate abstractions in Twee via the critical pair ranking heuristic, analogous to weighting in Otter. Specifically, when a critical pair contains a subterm matching

an abstraction, Twee reduces its weight, making that critical pair more likely to be selected. Thus abstractions guide the order of search without adding new equations.

Results

We evaluated Twitch on 1041 unsatisfiable unit equality problems from TPTP [6]. Domain abstractions are particularly effective when combined with goal flattening: on problems already solvable within a 300 second cutoff, this combination roughly halves runtime, and solves around 25 additional problems within that cutoff.

We consider a problem *hard* if it has TPTP rating at least 0.9 and default Twee cannot solve it within 1000 seconds; there are 70 such problems in our benchmark. Domain abstractions solve 18 of these within 1000 seconds, including 12 rating-1 problems. Since our evaluation uses a 1000 second timeout, we also ran Vampire 5.0.1 [3] on these 12 rating-1 problems. Vampire solves only 3 of the 12 within the same timeout, and in each case slower than Twitch.

Ablations also support treating abstractions as heuristic guidance rather than as definitional axioms. Adding definitions can give large speedups in individual cases, but becomes brittle as more definitions are added. Heuristic abstractions are more robust to larger abstraction sets.

Future Work

There are various avenues for further exploration.

The first question regards how exactly abstraction-based guidance should be implemented in Twee. At present, when checking for abstraction matches, Twee considers a critical pair *after* it has been simplified by existing rewrite rules. This can destroy the syntactic occurrence of the abstraction. We are experimenting with a critical pair-like mechanism for generating extra *derived abstractions* when an abstraction overlaps with a rewrite rule in such a way that rewriting destroys the abstraction. We are able to generate many extra abstractions this way, but the generated abstractions can be very different from the original abstractions and it is not clear if they respect the user's intent.

The second question is what terms should be abstracted. Current abstractions come from considering all terms occurring in the proof's final human-readable equality chain. But completion offers other sources: overlap terms, pre- and post-simplification critical pairs, and repeatedly reused rewrite rules. Some abstractions may correspond to meaningful algebraic operations; others may mark useful local rewrite contexts. Future work should compare these sources using criteria such as speedup, compression, proof-length reduction, transfer across problems, and interpretability.

The third question is whether abstractions can be generated predictively. Our experiments produce natural training data: problems, traces, proofs, learned abstractions, and measured speedups. Neural models could use this data to propose candidate abstractions from a problem statement, an early saturation trace, or related proofs. Such generation should be grammar-constrained so that proposed abstractions are well-formed first-order term patterns. We also wish to present currently ongoing related work on finding abstractions with the aim of producing shorter, more understandable proofs.

Finally, it would be interesting to adapt Veroff's hint strategy to the equational setting. Veroff obtained strong results by guiding a prover towards clauses occurring in proofs of related problems. Our abstraction-based weighting approach is, in a sense, orthogonal: rather than favouring particular proof steps that may occur once, it favours recurring term shapes that appear many times. For Twee, a hint might correspond not to a clause, but to a derived rewrite rule, a critical pair, or a peak that the search should eventually find or subsume.

References

[1] Guy Axelrod, Moa Johansson, and Nicholas Smallbone. Twitch: Learning abstractions for equational theorem proving. *arXiv preprint arXiv:2603.06849*, 2026.

[2] Matthew Bowers, Theo X. Olausson, Lionel Wong, Gabriel Grand, Joshua B. Tenenbaum, Kevin Ellis, and Armando Solar-Lezama. Top-down synthesis for library learning. *Proceedings of the ACM on Programming Languages*, 7(POPL):1182–1213, January 2023.

[3] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 1–35, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[4] R. Overbeek, J. McCharen, and L. Wos. Complexity and related enhancements for automated theorem-proving programs. *Computers & Mathematics with Applications*, 2(1):1–16, 1976.

[5] Nicholas Smallbone. Twee: An equational theorem prover. In *CADE*, pages 602–613, 2021.

[6] Geoff Sutcliffe and Christian Suttner. The TPTP problem library. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[7] Robert Veroff. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *Journal of Automated Reasoning*, 16(3):223–239, 1996.

[8] Robert Veroff. Solving open questions and other challenge problems using proof sketches. *Journal of Automated Reasoning*, 27(2):157–174, 2001.

[9] Larry Wos. The resonance strategy. *Computers & Mathematics with Applications*, 29(2):133–178, 1995.