Towards Automatic Verification of Textbook Proof Steps

Adam Dingle

Charles University, Prague

Introduction In a style of interactive proving pioneered by Naproche [3] and also used by our own prover Natty [4], the user enters proof text in a controlled natural language [10] that looks much like ordinary mathematical English. For this form of proving to be practical, we would like the system to be able to verify steps of a size typically found in textbook mathematics.

We have developed an automatic prover based on higher-order superposition [2] that is an integral part of Natty. The prover is intended to be able to prove conjectures that represent typical proof steps reliably and quickly. Natty's prover is unusual in some ways: its term ordering, unification, given clause selection, heuristics for pruning inferences, and rewriting mechanism work somewhat differently from other automatic provers such as E and Vampire.

We first described Natty and its automatic prover in a workshop paper two years ago [4]. The prover's architecture has evolved since then and its performance has improved dramatically. In this work we describe the prover's current architecture and compare its performance with other higher-order provers on two suites of test problems of textbook mathematics.

The Natty proof assistant Natty [6] implements a controlled natural language called N for writing mathematics. In N an input file may include axioms, definitions, and theorems with or without proofs. Inside a proof, any step may be annotated with *reasons*, which are references to one or more axioms or theorems to be used in proving that step.

Natty can translate an N source file into a series of formulas of higher-order logic [1]. Each generated formula is paired with a set of *premises* that may be used in proving it. Each premise is classified as an **axiom**, **definition**, or **theorem** that appeared earlier in the input file, or a **hypothesis** indicating an assumption or a known fact that appeared earlier in a proof.

By default, Natty will try to prove all the generated conjectures. Alternatively, Natty can write each generated conjecture to a separate output file in the THF format [7].

Proof calculus, term ordering, and unification Natty's proof calculus is based on a pragmatic subset of the higher-order superposition calculus [2].

Unusually, Natty uses the lexicographic path order [8] (LPO) for term ordering. Most other systems use the Knuth-Bendix order [9] by default. The LPO requires an ordering on constant symbols. Natty assigns precedences to constants according to their declaration order in the input. We have chosen the LPO and this particular term ordering so that rewrites will expand definitions and algebraic expressions.

Natty's prover performs commutative unification [12] of terms, which is unusual among automatic provers. To achieve this, Natty notices any axioms or theorems that state that a binary operation is commutative. Because Natty performs commutative unification, it does not need to perform any superposition or rewrite between a commutative axiom or theorem and another clause.

Initial inferences Natty considers certain superposition steps at the beginning of a clause's derivation to be *initial inferences*. A *reason application* is an initial inference between a premise marked as a reason and the goal or the last hypothesis. A *definition expansion* is an initial inference between a premise marked as a definition and the goal or any hypothesis. A *hypothesis substitution* is an initial inference between any hypothesis and the goal or the last hypothesis.

Superposition restrictions Natty requires that in all superpositions at least one premise must be derived from a hypothesis or the goal. This is essentially a set of support strategy [13]. Natty classifies each superposition inference as a *resolution* or a *paramodulation*. Natty allows paramodulations only in certain circumstances, such as when the inference is a definition expansion or hypothesis substitution, or the premises are the last hypothesis and the goal. Natty performs some superpositions (such as definition expansions and hypothesis substitutions) *leniently*, meaning that side conditions indicating term ordering restrictions are disregarded.

Cost function Natty uses only a single priority queue of clauses, ordered by *cost*. The cost of a clause is the total cost of all inferences in its derivation, where an inference's cost is defined as follows. A superposition inference is *increasing* if the resulting clause (after rewriting) has a weight that is larger than the weight of any parent clause that is derived from a hypothesis or the goal. If any non-initial superposition is increasing, it is discarded. Otherwise, every superposition inference has a cost of 0.01, except for a non-increasing reason application which has a cost of 0. The effect of these rules is that every proof must consist of a series of possibly increasing initial inferences, followed by a series of superpositions that do not increase.

Non-destructive rewriting Suppose that a clause C may be rewritten with either of two identities A or B. It may be that a rewrite with A will yield a quick refutation, whereas a rewrite with B will destroy that opportunity and lead to a significantly longer proof. To prevent an unfortunate rewrite from spoiling an easy refutation attempt, Natty implements a technique that we call *non-destructive rewriting*. To minimize its performance cost, it is used only for clauses derived from the goal or the last hypothesis. When such a clause is created, Natty creates a copy of the clause called a *rewrite child*, which may be destructively rewritten with any identity. Natty also preserves the original clause, which may be rewritten only with certain identities deemed to be *safe*, such as the identity $\forall x, y \,.\, x > y \leftrightarrow y < x$.

Test suites We have built a test suite called TextbookMath derived from a textbook by Mendelson [11] that develops the number systems $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$. We translated several sections of the Mendelson text as literally as possible into a set of N source files. The suite contains 151 theorems, from which Natty generates 962 conjectures. Additionally, Natty includes a library of mathematics called Natlib [5] that is more loosely based on the Mendelson text, and in which we have added intermediate proof steps so that Natty can verify all theorems in the library with a 5-second time limit per step. Natlib contains 235 theorems with 1597 conjectures.

Performance comparison Table 1 shows the performance of Natty 0.0.3, E 3.3.1, Vampire 4.8, and Zipperposition 2.1 in proving the conjectures from the TextbookMath suite and the Natlib library. We used a 5-second time limit per conjecture, reflecting our desire for fast interactive verification of proof steps. Natlib uses polymorphic types, so the Natlib comparison does not include E, which does not support polymorphic THF. In the table, the column entitled "Natty + by" shows Natty's performance when it is given the reasons that were indicated for each proof step. The column "Natty" shows the performance when those reasons are hidden.

Table 1: Prover performance (5-second timeout)

suite		**Natty + by**	**Natty**	**E**	**Vampire**	**Zipperpos.**
TM	proved (of 962)	894 (93%)	877 (91%)	809 (84%)	818 (85%)	782 (81%)
	avg time	0.05 sec	0.05 sec	0.12 sec	0.36 sec	0.35 sec
Natlib	proved (of 1597)	1597 (100%)	1558 (98%)	—	1224 (77%)	1133 (71%)
	avg time	0.06 sec	0.06 sec	—	0.59 sec	0.58 sec

Acknowledgements This research was supported by grant 128524 from Charles University.

References

[1] Andrews, P.B.: An introduction to mathematical logic and type theory: to truth through proof, vol. 27. Springer Science & Business Media (2013)

[2] Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P.: Superposition for higher-order logic. Journal of Automated Reasoning **67**(1) (2023)

[3] Cramer, M., Fisseni, B., Koepke, P., Kühlwein, D., Schröder, B., Veldman, J.: The Naproche project: controlled natural language proof checking of mathematical texts. In: International Workshop on Controlled Natural Language. pp. 170–186. Springer (2009)

[4] Dingle, A.: A natural-language proof assistant for higher-order logic (work in progress). In: Brown, C.W., Kaufmann, D., Nalon, C., Steen, A., Suda, M. (eds.) Proceedings of the 9th Workshop on Practical Aspects of Automated Reasoning (PAAR), 2024 co-located with the 12th International Joint Conference on Automated Reasoning (IJCAR 2024), Nancy, France. CEUR Workshop Proceedings, vol. 3717, pp. 57–73. CEUR-WS.org (2024)

[5] Dingle, A.: Natlib: Natty standard library (2026), https://github.com/medovina/natty/tree/main/math

[6] Dingle, A.: Natty code repository (2026), https://github.com/medovina/natty

[7] Kaliszyk, C., Sutcliffe, G., Rabe, F.: TH1: The TPTP typed higher-order form with rank-1 polymorphism. In: PAAR@ IJCAR. pp. 41–55 (2016)

[8] Kamin, S.: Two generalizations of the recursive path ordering. Unpublished manuscript (1980)

[9] Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras. In: Computational problems in abstract algebra. pp. 263–297. Elsevier (1970)

[10] Kuhn, T.: A survey and classification of controlled natural languages. Computational Linguistics **40**(1), 121–170 (2014)

[11] Mendelson, E.: Number Systems and the Foundations of Analysis. Academic Press, Inc. (1973)

[12] Siekmann, J.H.: Unification of commutative terms. In: Proceedings of the International Symposiumon on Symbolic and Algebraic Computation. p. 22 (1979)

[13] Wos, L., Robinson, G.A., Carson, D.F.: Efficiency and completeness of the set of support strategy in theorem proving. Journal of the ACM (JACM) **12**(4), 536–541 (1965)