Natural-Language Proofs with Higher-Order Logic

Adam Dingle

Charles University, Prague, Czech Republic

Introduction Natty[1] is a natural-language proof assistant written in OCaml, with an embedded automatic prover based on higher-order superposition. An input file for Natty is written in a controlled natural language [3, 2] that looks like mathematical English with a restricted grammar and vocabulary. It may contain axioms, definitions, and theorems, which may optionally include proofs written in natural language. Natty translates this input to a series of formulas of higher-order logic, each representing an entire theorem or a proof step. Natty then attempts to prove these formulas using its embedded prover.

Natty has now been in development for approximately one year. It can read an natural-language input file that defines the natural numbers and integers and asserts a series of theorems about them. Natty can prove most of these theorems in less than 1 second per theorem. Many of these proofs require a higher-order induction step. Additionally, some theorems in this input file contain hand-written proofs. Natty can verify most of the steps in these proofs quickly. Natty can also export theorems and proof steps to files in the THF [6] format. The resulting THF files with theorems about natural numbers may form an interesting test set for other provers too.

Natural-language input Natty reads source files written in the Natty language, a controlled natural language for writing mathematics. The language is intended to resemble textbook mathematics as closely as possible, within the limitations of plain text and the Unicode character set. For example, Natty will implicitly multiply variables that appear in adjacent succession.

Logic Natty is founded on classical higher-order logic, which is useful as a foundation even in this early stage of development. For example, an input file can express Peano induction without using an axiom schema. Natty represents sets as functions with codomain $\mathbb{B}$, an elegant formulation that does not require any set theory axioms. The strong type system allows Natty to perform useful type checking on the user's behalf, and may even make inference more efficient.

Translation into logic Natty parses the input file, following a mostly context-free grammar for its controlled natural language. The parser outputs a list of *statements*, each of which is a type declaration, a constant declaration, an axiom, a definition, or a theorem. A theorem that includes a natural-language proof will contain parsed proof steps, which Natty arranges into a tree that indicates the scope of each variable and assumption in the proof. Natty infers the proof structure heuristically; generally speaking, it makes the scope of each introduced variable as small as possible while still enclosing all references to the variable. Natty then generates a series of formulas representing individual proof steps.

Proof calculus Natty's proof calculus is a pragmatic, incomplete variant of the higher-order superposition calculus recently developed by Bentkamp et al. [1] Natty has superposition and equality resolution rules adapted from that calculus, plus an outer clausification rule and a clause splitting rule that are variants of the OuterClaus rule from [4]. It also has typical contracting rules that rewrite, simplify or eliminate clauses. To compare two higher-order terms, Natty first maps them to first-order terms using a translation suggested in [1]. It then applies the

[1] https://github.com/medovina/natty

Table 1: Theorems and proof steps ($\mathbb{N}$)

	Natty	E	Vampire	Zipperposition
proved (of 214)	214	180	177	185
average time	0.20	0.08	0.27	0.23
PAR-2 score	0.20	1.65	1.95	1.55

Table 2: Theorems and proof steps ($\mathbb{Z}$)

	Natty	E	Vampire	Zipperposition
proved (of 117)	85	101	86	74
average time	0.49	0.12	0.19	0.62
PAR-2 score	3.09	1.47	2.79	4.06

lexicographic path order, using a symbol ordering in which terms defined later have higher precedence.

Unification A complete higher-order prover must perform full higher-order unification, which is challenging and expensive. Natty is a pragmatic, incomplete prover and its unification procedure is almost entirely first-order. It can also unify two lambda terms in some situations. Despite these limitations, Natty can perform useful unifications in which a variable becomes bound to a higher-order term. In this way Natty can often find inductive proofs.

Main loop and given clause selection Natty's main loop is modeled after the main loop in E [5] and searches for a contradiction by saturating the set of input formulas. Natty uses a given clause selection mechanism with only a single priority queue. The cost of a clause is the total cost of all the proof steps that were used to build it. The cost of a rewrite step is 0; the cost of a superposition step is assigned heuristically depending on how it affects the number of literals and symbols in the clause.

Clausification Natty attempts to preserve formula structure as much as possible when generating inferences via a process of *dynamic clausification*. Whenever Natty computes possible superpositions between two clauses C and D, it expands them temporarily into two sequences $C_1, \ldots, C_m$ and $D_1, \ldots, D_n$ by performing all possible incremental clausification steps that do not split the clauses. It then looks for superposition inferences between all pairs C_i and D_j, considering each literal or subterm only in the first C_i or D_j in which it because exposed at a position valid for superposition. In this way, each processed clause serves as a representative of all the clauses that can be generated from it by non-splitting clausification steps.

Performance We evaluated Natty's current performance versus that of E 3.2.5, Vampire 4.8, and Zipperposition 2.1 in proving theorems and proof steps exported from a file that asserts various elementary theorems about the natural numbers and integers. The time limit per proof was 5 seconds. The results are in Tables 1 and 2. We think Natty's performance at this early stage is promising.

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

References

[1] Alexander Bentkamp, Jasmin Blanchette, Sophie Tourret, and Petar Vukmirović. Superposition for higher-order logic. *Journal of Automated Reasoning*, 67(1), 2023.

[2] Thomas Hales. An argument for controlled natural languages in mathematics, 2019.

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

[4] Visa Nummelin, Alexander Bentkamp, Sophie Tourret, and Petar Vukmirovic. Superposition with first-class booleans and inprocessing clausification. In *CADE*, pages 378–395, 2021.

[5] Stephan Schulz. E–a brainiac theorem prover. *AI Communications*, 15(2-3):111–126, 2002.

[6] Geoff Sutcliffe. The logic languages of the TPTP world. *Logic Journal of the IGPL*, 31(6):1153–1169, 2023.