Scaling Naproche

Adrian De Lon and Peter Koepke

University of Bonn, Germany, https://www.math.uni-bonn.de/ag/logik/

The Naproche [1] (Natural proof checking) theorem prover demonstrates that it is possible to write non-trivial fully natural machine-verified proofs in a controlled natural language for mathematics: every statement is a statement of the common language of mathematics, and the argument uses familiar declarative proof structures. Naproche formalizations can be written in a LaTeX format which allows immediate mathematical typesetting. The system is available within the Isabelle prover platform [3], including some example formalizations which partly resemble undergraduate mathematical material.

Natural mathematical texts assume that the reader could in principle fill in lots of technical detail and implicit proof steps. Likewise Naproche's aspiration to process similar texts requires a high degree of "machine intelligence" which is achieved through careful organization of proof tasks which are given to strong automated provers. Modeling the softly typed natural language is carried out with first-order defined types so that subtypes and partial functions with guards on definedness are available. Unfortunately type-checking of complex terms presently spawns a number of general first-order proof tasks which in general can only be discharged at run-time, i.e., during the proof process. "Human-sized" proof steps also stretch the abilities of ATPs. At the moment basically all previous statements are made into premisses of proof tasks. Often reformulations of proofs are necessary to help the ATP whilst keeping the naturalness and readability of a text.

As improvements to the Naproche system are allowing longer and interlinked formalizations, the above-mentioned difficulties are mounting up. Naproche's predecessor SAD was only able to process and check mathematical "miniatures" which each came along with purpose-built ontological preliminaries. Small text sizes resulted in small proof tasks which did not overwhelm the ATPs. With the incorporation of SAD into Naproche we have gone from miniatures to chapter sized texts and recently to small libraries to be re-used in other formalizations. Obviously this led to a steep increase in the number of possible premisses which resulted in several types of prover problems:

Check times. Many Naproche formalizations require several minutes checking time on standard laptops but some texts need even some hours. Usually more time is spent on type checking (which is called ontological checking) than on the logical checking of proof steps. These problems got worse after replacing a crude and potentially contradictory underlying set theory by a Kelley–Morse-style ontology which distinguishes between sets and classes. This, however, leads to many additional proof in ontological checking: when unions $A \cup B$ have been originally defined for classes, then the union $a \cup b$ of two sets requires proofs that a and b are also classes.

Erratic prover behaviour. Extensive texts usually contain many function symbols which an ATP can use for unification-based proof searches. Automatic proof steps can be confused by adding symbols which in principle are not connected with the actual prover task.

Ontological checking. Although type-checking should normally be a mild proving task, the use of general purpose ATPs in a large number of checks increases the probability that an ATP will get on a wrong track and miss obvious arguments. Sometimes we could only get ontological correctness by putting some type information into a statement immediately before the statement with the problematic term - something one would hardly find in natural mathematical texts.

Stability of formalizations. We use E [7, 8] as the main external prover. Many Naproche formalizations are written against a specific version of E, with proof steps chosen accordingly.

Replacing that version of E with a different version (or a different ATP entirely) typically results in some proof tasks failing, even when the different version performs better overall. This results in a significant maintenance burden, particularly for larger formalizations.

In our talk we shall report on ongoing work aiming to avoid or mitigate these scaling issues.

1. Tracing prover behaviour: when do provers latch on to obviously hopeless search path? What can we learn from concrete examples?

2. Experimenting with multiple ATPs: where lie the relative strengths of individual ATPs in the context of Naproche? For example we have seen proof tasks where the addition of a single irrelevant hypothesis makes the task impossible for some provers, while others can solve the problem within roughly one second. So far we have used E, iProver [4, 2], SPASS [9, 11], and Vampire [10, 5] for our experiments.

3. Adding premise selection (e.g. starting with a MePo-like [6] filter) to Naproche.

4. Simplifying or reducing ontological checking. Most ontological checks should not amount to full first-order proof tasks.

5. Experiments with different ontologies (Kelley–Morse vs. Zermelo–Fraenkel) show that there are trade-offs between richer ontologies (adding classes and/or urelements) on the one hand and burdening users with proof obligations as well as cluttering exported proof tasks with type guards on the other.

6. Reducing checking times through improvements to the architecture of Naproche: increasing parallelism, caching, and more.

We shall also talk more generally about the potential of the Naproche approach with respect to article-sized and textbook-sized formalizations.

References

[1] De Lon, A., Koepke, P., Lorenzen, A., Marti, A., Schütz, M., Wenzel, M.: The Isabelle/Naproche natural language proof assistant. In: Platzer, A., Sutcliffe, G. (eds.) Automated Deduction – CADE 28. pp. 614–624. Springer International Publishing, Cham (2021)

[2] Duarte, A., Korovin, K.: Implementing superposition in iProver (system description). In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12167, pp. 388–397. Springer (2020). https://doi.org/10.1007/978-3-030-51054-1_24, https://doi.org/10.1007/978-3-030-51054-1_24

[3] Isabelle contributors: The Isabelle2021 release (February 2021), https://isabelle.in.tum.de

[4] Korovin, K.: iProver – a theorem prover for first-order logic with support for arithmetic reasoning, http://www.cs.man.ac.uk/~korovink/iprover/

[5] Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: CAV 2013. pp. 1–35 (2013)

[6] Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. J. Appl. Log. **7**, 41–57 (2009)

[7] Schulz, S.: The E theorem prover, https://eprover.org

[8] 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)

[9] SPASS contributors: SPASS workbench, https://www.mpi-inf.mpg.de/departments/automation-of-logic/software/spass-workbench

[10] Voronkov, A., Kovács, L., Reger, G., Suda, M., Rawson, M., Bhayat, A., Schoisswohl, J., Rath, J., Hozzova, P.: The Vampire prover, https://vprover.github.io/

[11] Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: Schmidt, R.A. (ed.) Automated Deduction – CADE-22. pp. 140–145 (2009)