Proof By Abduction in Isabelle/HOL

Yutaka Nagashima[1] and Daniel Sebastian Goc[2]

[1] Independent
[2] University of Cambridge

Abstract

When proving an inductive problem, we often prove auxiliary lemmas that are useful for proving the original problem. If these auxiliary lemmas themselves are challenging, we must introduce more lemmas to prove these lemmas. To automate such multi-step conjecturing, we developed Abduction Prover. Given a proof goal, Abduction Prover conjectures a series of lemmas and attempts to prove the original goal using these lemmas. Our working prototype of Abduction Prover for Isabelle/HOL is publicly available on GitHub.

All major theorem provers for higher-order logics offer tools called tactics. Tactics are designed to transform proof goals into easier formats. Users apply suitable tactics with certain arguments to proof goals until the tactics solve their proof goals completely.

When proving challenging statements, experienced users often introduce auxiliary lemmas explicitly and use these lemmas to prove the final goals. Many believe that this approach is superior to developing long sequences of tactics: these lemmas make the resulting proof scripts more declarative and readable, and they can be utilized to tackle other challenging problems. If the auxiliary lemmas themselves are challenging, more conjectures should be introduced that are useful for proving the challenging lemmas.

In this abstract, we introduce Abduction Prover, a framework designed to search for useful conjectures recursively to prove the goal until the proof is completed using these conjectures. The overall workflow of Abduction Prover is shown in Algorithm 1, although some definitions are omitted due to space restrictions. Given a proof goal, this algorithm builds its proof and auxiliary lemmas by expanding a rooted directed graph.

Algorithm 1 Abduction Prover

```
1: graph ← set_root goal
2: depth ← 1
3: while depth ≤ Limit ∧ ¬ proved graph do
4:     depth ← depth + 1
5:     nodes ← get_active_nodes graph
6:     fold expand_node nodes graph
7: end while
8: show graph
```

We call this graph an abduction graph, as abductive reasoning is executed on this graph to identify conjectures useful for proving the original goal, which appears as the root node. The graph consists of two kinds of nodes (and-nodes and or-nodes) and two kinds of edges (labeled edges and unlabeled edges). Each node represents a proof goal.

Intuitively, or-nodes represent choices, while and-nodes represent obligations. When an or-node points to multiple and-nodes, it means the or-node can be proven if one of the and-nodes directly pointed to by the or-node can be proven. On the other hand, when an and-node points to multiple or-nodes, it means the and-node can be proven if all the or-nodes directly pointed to by the and-node are proven.

Figure 1 displays an example of an abduction graph. This graph demonstrates that Algorithm 1 has already worked on the root node twice to conjecture auxiliary lemmas (conjecturing-A and conjecturing-B) and once to apply a tactic (tactic-B). Nodes encircled by single lines represent or-nodes, while nodes enclosed by double lines signify and-nodes. For example, the root node points to two and-nodes via solid lines, which means that to prove the goal, only one of the two and-nodes must be proven.

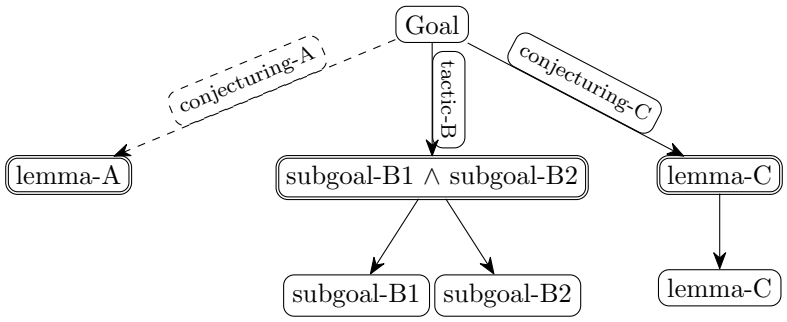

Figure 1: An Example Abduction Graph.

This figure shows that Algorithm 1 has produced two conjectures by explicit conjecturing: lemma-A and lemma-C, while it has also produced another conjecture (subgoal-B1 ∧ subgoal-B2) by applying a tactic. The solid edge connecting the root to lemma-C should be interpreted as follows: Algorithm 1 confirmed that we can prove the root node using lemma-C as assumption. On the other hand, the dashed edge from the root to lemma-A indicates that Algorithm 1 conjectured lemma-A but Algorithm 1 failed to prove the root using lemma-A as assumption. Since we do not know if lemma-A is useful to prove the root, Algorithm 1 does not connect lemma-A to the root to keep the size of the graph small.

In addition to explicit conjecturing, Algorithm 1 integrates tactic applications as implicit conjecturing. For example, Figure 1 indicates that the root node can be reduced into two subgoals (subgoal-B1 and subgoal-B2) by applying tactic-B, and Algorithm 1 treats the application of tactic-B as equivalent to conjecturing two lemmas, subgoal-B1 and subgoal-B2. In general, one tactic application can return multiple subgoals, which Algorithm 1 groups into a single and-node. Then, from such an and-node, Algorithm 1 produces or-nodes, each of which corresponds to a subgoal in the parent and-node.

We call Algorithm 1 Abduction Prover because it continually conjectures new auxiliary lemmas for the leaf or-nodes recursively in each iteration until enough lemmas have been proven to establish the root node. Furthermore, we call the underlying data structure an abduction graph rather than an abduction tree because, in general, Algorithm 1 may produce the same conjectures multiple times from different nodes.

Our working prototype of Abduction Prover for Isabelle2023 [10] is available for public access on GitHub [1] and its demo is available on YouTube [2]. We also added two screenshots from the YouTube video in Appendix. While page limitations prevent us from delving into the technical challenges Abduction Prover addresses, it is worth noting that it offers the following benefits:

- It focuses on useful conjectures out of many produced ones by checking if the conjectures are useful for proving the original goal.
- It integrates explicit conjecturing and tactic applications into one monolithic framework.
- It integrates many existing tools, such as quickcheck [3], sledgehammer [11], PSL [8], smart_induct [5, 6], template-based conjecturing [9], and SeLFiE [4, 7], to keep the size of the abduction graph small.

Appendix

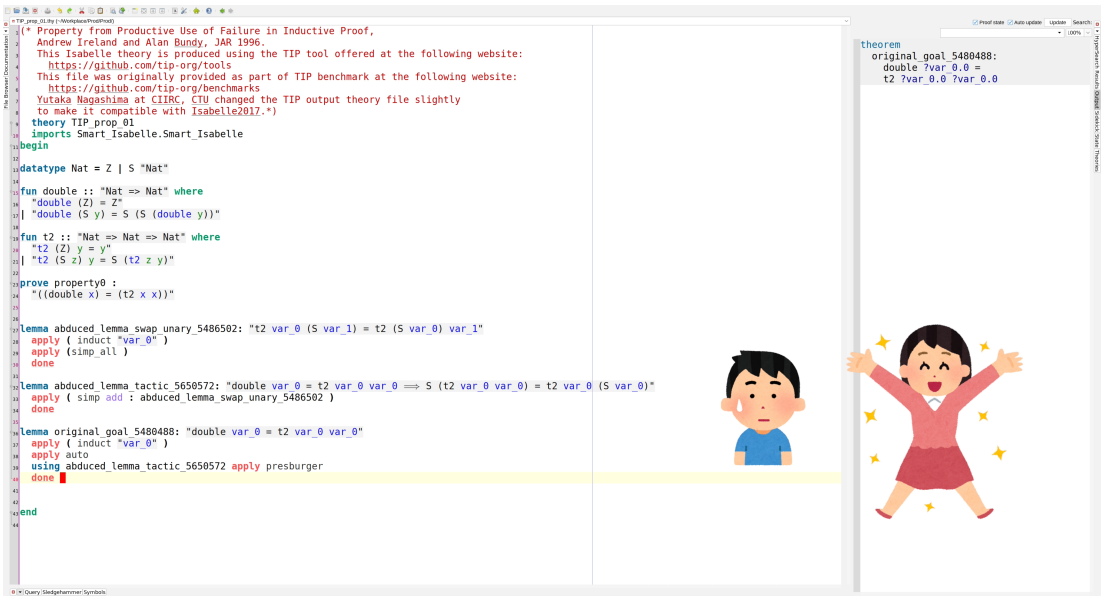

Figure 2: A screenshot from the demo video. The `prove` command invoked the Abduction Prover, which proved the final goal, `((double x) = (t2 x x))`, as `lemma original_goal_5480488` by creating and proving the two auxiliary lemmas.

Figure 3: A screenshot from the demo video. The Abduction Prover proved the given goal by creating and proving the 13 lemmas.

References

[1] https://github.com/data61/PSL/releases/tag/v0.2.7-alpha.

[2] https://youtu.be/rXU-lJxP_GI.

[3] Lukas Bulwahn. The new quickcheck for Isabelle - random, exhaustive and symbolic testing under one roof. In Chris Hawblitzel and Dale Miller, editors, *Certified Programs and Proofs - Second International Conference, CPP 2012, Kyoto, Japan, December 13-15, 2012. Proceedings*, volume 7679 of *Lecture Notes in Computer Science*, pages 92–108. Springer, 2012.

[4] Yutaka Nagashima. LiFtEr: Language to encode induction heuristics for Isabelle/HOL. In *Programming Languages and Systems - 17th Asian Symposium, APLAS, Nusa Dua, Bali, Indonesia*, 2019.

[5] Yutaka Nagashima. Smart induction for Isabelle/HOL (tool paper). In *Proceedings of the 20th Conference on Formal Methods in Computer-Aided Design – FMCAD 2020*, 2020.

[6] Yutaka Nagashima. Faster smarter proof by induction in Isabelle/HOL. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 1981–1988. ijcai.org, 2021.

[7] Yutaka Nagashima. Definitional quantifiers realise semantic reasoning for proof by induction. In Laura Kovács and Karl Meinke, editors, *Tests and Proofs - 16th International Conference, TAP 2022, Held as Part of STAF 2022, Nantes, France, July 5, 2022, Proceedings*, volume 13361 of *Lecture Notes in Computer Science*, pages 48–66. Springer, 2022.

[8] Yutaka Nagashima and Ramana Kumar. A proof strategy language and proof script generation for Isabelle/HOL. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 528–545. Springer, 2017.

[9] Yutaka Nagashima, Zijin Xu, Ningli Wang, Daniel Sebastian Goc, and James Bang. Template-based conjecturing for automated induction in isabelle/hol. In Hossein Hojjat and Erika Ábrahám, editors, *10th IPM International Conference on Fundamentals of Software Engineering, Conference Pre-Proceedings*, pages 111–125. IPM, 2022.

[10] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - a proof assistant for higher-order logic*. Springer, 2002.

[11] Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011*, volume 2 of *EPiC Series in Computing*, pages 1–11. EasyChair, 2010.