Saturation-Guided Inductive Synthesis

Laura Kovács
TU Wien, Vienna, Austria

Abstract

Proof by induction is common-place in mathematics [17, 4], formal verification [3, 1, 5], cybersecurity [12, 14], and many more areas. This talk overviews recent progress in automating inductive reasoning in quantified logic, with applications to code synthesis. Key to our work is saturation-based first-order theorem proving [13], using variants of the superposition calculus [15]. We show that induction and synthesis are *better together in saturation*, allowing us not only to prove quantified properties F, but also generate a functional implementation of F during proof search[1]. We showcase our results using the first-order theorem prover Vampire [2], a completely automatic push-button theorem prover for first-order logic with theories, including arithmetic, inductively defined datatypes, induction, and higher-order logic.

We structure our talk within three inter-connected parts.

First, we overview the main ingredients behind *saturation* provers [2, 16, 18] using superposition. Such provers work by negating an input conjecture F, transforming $\neg F$ into a clausal normal form, and using superposition inferences to derive new clauses from existing ones until a contradiction is reached; when a contradiction is derived, validity of F is established. Many years of development in saturation-based theorem proving have gone into making this process as efficient as possible, while deriving new clauses only when needed in order to tame growth of the search space. Doing so, highly-efficient superposition calculi parametrized by so-called clause selection functions have been proposed, in order to make as few inferences between clauses as possible. Redundancy elimination techniques further prune the search space.

Next, we show how to formalize applications of *induction* in the saturation process [7], without bringing drastic changes into the overall framework of first-order proving. A natural choice for implementing induction would be by reducing goals to subgoals, in particular by proving a base case and an inductive step case of a valid induction principle. For example, a goal $\forall x.F(x)$ over natural numbers x can be proven using structural induction: we prove $F[0]$ (base case) and $\forall x.F(x) \implies F(x+1)$ (step case). However, saturation theorem proving is not about reducing goals to subgoals: in principle, each clause in the search space can be chosen during any step of saturation. We therefore automate induction in saturation as follows. When a clause $F(x)$ is chosen and inductive reasoning over F should be applied (for example, because F uses inductively defined data types x, such as natural numbers), we combine the application of a valid induction schema over $F(x)$ with resolution. Put it simply, induction and resolution are combined in one step of saturation, allowing us to use parts of $F(x)$ as subgoals of $F(x)$. Interestingly with this approach is that clauses generated during saturation may be stronger than the induction schema and, most importantly, are friendly to saturation provers: they are mostly quantifier-free Horn clauses and their (at most one) positive equality cannot be used in many inferences during saturation. Thus, applying many induction inferences during proof search would hardly affect the performance of a saturation prover. Figure 1 lists a property over natural numbers: every natural number x is the half of another natural number y. Proving this property in saturation, and in particular using Vampire, can be achieved by (structural) induction over x.

Finally, we extend saturation proof search with code *synthesis* [10]. While proving formula F, we track the constructive parts of the proof of F using so-called answer literals [6]. We use these parts to synthesize a program satisfying F and use the applications of induction in saturation to construct recursive programs satisfying F. In a nutshell, the base case and inductive case steps of induction in saturation express how to construct the desired program for the next recursive step using the program for the previous recursive step; we capture this information via answer literals. When we apply induction in saturation, we introduce a special term into the answer literal and record

[1] for simplicity, we may write F instead of $\forall x.F(x)$

11th International Conference on Formal Structures for Computation and Deduction (FSCD 2026).
Editor: Frank Pfenning; Article No. 2; pp. 2:1–2:3

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$\begin{aligned}
\text{axioms: } & \mathsf{half}(0) \simeq 0 && \text{(H1)}\\
& \mathsf{half}(\mathsf{s}(0)) \simeq 0 && \text{(H2)}\\
& \forall x.\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \simeq \mathsf{s}(\mathsf{half}(x)) && \text{(H3)}\\
& \mathsf{even}(0) && \text{(E1)}\\
& \forall x.\ \mathsf{even}(\mathsf{s}(\mathsf{s}(x))) \simeq \mathsf{even}(x) && \text{(E2)}\\
\text{property: } & \forall x \exists y.\ \mathsf{half}(y) \simeq x \ \wedge\ \mathsf{even}(y) && \text{(F)}
\end{aligned}$$

Figure 1 Axioms of half and even and the conjectured property to be proven over natural numbers. Here, 0 and s respectively denote the 0 and `successor` constructors of natural numbers; we denote the equality predicate by $\simeq$.

the program corresponding to the induction step. As we prove induction steps, we capture their corresponding programs in the answer literal. Finally, we convert the special tracker terms from the answer literals into recursive functions, and obtain a program satisfying property F. For example, from the proof of property of Figure 1, our approach implemented in Vampire infers the following functional implementation of a recursive function r, while using only the signature of Figure 1:

$$\begin{aligned}
\mathsf{r}(0) &:= 0\\
\mathsf{r}(\mathsf{s}(x)) &:= \mathsf{s}(\mathsf{s}(\mathsf{r}(x)))
\end{aligned}$$

The above inferred function r satisfies the property of Figure 1 and, for each input natural number x, computes a natural number $\mathsf{r}(x)$ such that x is half of $\mathsf{r}(x)$.

In summary, induction and synthesis are better together in saturation-based theorem proving using the superposition calculus. Soundness and practical use of our work has been addressed and experimented using the Vampire theorem prover, both in the case of automating induction [7, 9] and program synthesis [11, 10]. Interesting questions regarding completeness arise: if a program satisfying a given property exists, can we derive it from saturation-based proof search? Our recent results [8] answer this question for recursion-free program using additional assumptions of realizability. A natural direction for future work is to identify realizability assumptions for recursive program synthesis and induction.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Logic and verification; Theory of computation → Equational logic and rewriting; Theory of computation → Program specifications

Keywords and phrases automated reasoning, first-order theorem proving, saturation, induction, program synthesis

Digital Object Identifier 10.4230/LIPIcs.FSCD.2026.2

Category Invited Talk

Funding *Laura Kovács*: This work has been partially funded by the the European Research Council (ERC) Consolidator Grant ARTIST 101002685; the ERC Proof of Concept Grant LEARN 10121341; the Austrian Science Fund (FWF) 10.55776/DOC1345324; the FWF SFB project SpyCoDe F8504; the WWTF ICT22-007 grant ForSmart; and the SBA Research COMET Center SBA-K1 NGC.

Acknowledgements This talk is based on joint works with a number of authors, including Daneshvar Amrollahi, Márton Hajdu, Petra Hozzová, Chase Norman, Andrei Voronkov, and Eva Maria Wagner.

References

1 Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Martin Giese, Elmar Habermalz, Reiner Hähnle, Wolfram Menzel, and Peter H. Schmitt. The KeY Approach: Integrating Object Oriented Design and Formal Verification. In *JELIA*, 2000. doi:10.1007/3-540-40006-0_3.

2 Filip Bártek, Ahmed Bhayat, Robin Coutelier, Márton Hajdú, Matthias Hetzenberger, Petra Hozzová, Laura Kovács, Jakob Rath, Michael Rawson, Giles Reger, Martin Suda, Johannes Schoisswohl, and Andrei Voronkov. The Vampire Diary. In *CAV*, 2025. doi:10.1007/978-3-031-98682-6_4.

3 Raven Beutner and Bernd Finkbeiner. Checking Satisfiability of Hyperproperties Using First-Order Logic. In *ATVA*, 2024. doi:10.1007/978-3-031-78750-8_10.

4 Martin Desharnais, Petar Vukmirovic, Jasmin Blanchette, and Makarius Wenzel. Seventeen Provers Under the Hammer. In *ITP*, 2022. doi:10.4230/LIPIcs.ITP.2022.8.

5 Pamina Georgiou, Bernhard Gleiss, Ahmed Bhayat, Michael Rawson, Laura Kovács, and Giles Reger. The Rapid Software Verification Framework. In *FMCAD*, 2022. doi:10.34727/2022/ISBN.978-3-85448-053-2_32.

6 Cordell Green. Theorem-Proving by Resolution as a Basis for Question-Answering Systems. *Machine Intelligence*, 4:183–205, 1969.

7 Márton Hajdú, Petra Hozzová, Laura Kovács, Giles Reger, and Andrei Voronkov. Getting Saturated with Induction. In *Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, pages 306–322. Springer, 2022. doi:10.1007/978-3-031-22337-2_15.

8 Márton Hajdu, Petra Hozzová, Laura Kovács, and Eva Maria Wagner. Completeness of Synthesis under Realizability Assumptions Using Superposition. In *IJCAR*, 2026. To appear.

9 Márton Hajdú, Laura Kovács, and Michael Rawson. Rewriting and inductive reasoning. In *LPAR 2024*, EPiC Series in Computing, pages 278–294. EasyChair, 2024. doi:10.29007/VBFP.

10 Petra Hozzová, Daneshvar Amrollahi, Márton Hajdú, Laura Kovács, Andrei Voronkov, and Eva Maria Wagner. Synthesis of Recursive Programs in Saturation. In *IJCAR*, pages 154–171. Springer, 2024. doi:10.1007/978-3-031-63498-7_10.

11 Petra Hozzová, Laura Kovács, Chase Norman, and Andrei Voronkov. Program Synthesis in Saturation. In *CADE*, pages 307–324. Springer, 2023. doi:10.1007/978-3-031-38499-8_18.

12 Simon Jeanteur, Laura Kovács, Matteo Maffei, and Michael Rawson. CryptoVampire: Automated Reasoning for the Complete Symbolic Attacker Cryptographic Model. In *IEEE SP*, 2024. doi:10.1109/SP54263.2024.00246.

13 Laura Kovács and Andrei Voronkov. First-Order Theorem Proving and Vampire. In *CAV*, 2013. doi:10.1007/978-3-642-39799-8_1.

14 Evan Laufer, Alex Ozdemir, and Dan Boneh. zkPi: Proving Lean Theorems in Zero-Knowledge. In *CCS*, 2024. doi:10.1145/3658644.3670322.

15 Robert Nieuwenhuis and Albert Rubio. Paramodulation-Based Theorem Proving. In *Handbook of Automated Reasoning (in 2 volumes)*, pages 371–443. Elsevier and MIT Press, 2001. doi:10.1016/B978-044450813-3/50009-6.

16 Stephan Schulz, Simon Cruanes, and Petar Vukmirovic. Faster, Higher, Stronger: E 2.3. In *CADE*, 2019. doi:10.1007/978-3-030-29436-6_29.

17 Josef Urban and Geoff Sutcliffe. Automated Reasoning and Presentation Support for Formalizing Mathematics in Mizar. In *AISC*, 2010. doi:10.1007/978-3-642-14128-7_12.

18 Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In *CADE*, 2009. doi:10.1007/978-3-642-02959-2_10.