EasyChair Preprint

№ 2468

# Induction with Generalization in Superposition Reasoning

Petra Hozzová, Laura Kovács, Johannes Schoisswohl and Andrei Voronkov



January 27, 2020

# Induction with Generalization in Superposition Reasoning

Petra Hozzová[1], Laura Kovács[1,2], Johannes Schoisswohl[1], and Andrei Voronkov[3,4]

[1] TU Wien, Austria
[2] Chalmers University of Technology, Sweden
[3] University of Manchester, UK
[4] EasyChair

**Abstract.** We describe an extension of automating induction in superposition-based reasoning by strengthening inductive properties and generalizing terms over which induction should be applied. We implemented our approach in the first-order theorem prover Vampire and evaluated our work against state-of-the-art reasoners automating induction. We demonstrate the strength of our technique by showing that many interesting mathematical properties of natural numbers and lists can be proved automatically using this extension.

## 1 Introduction

Recent advances related to automating inductive reasoning, such as first-order reasoning with inductively defined data types [6], the AVATAR architecture [13], inductive strengthening of SMT properties [10], structural induction in superposition [5] and general induction rules within saturation [8], make it possible to re-consider the grand challenge of mechanizing mathematical induction [4]. We contribute to these advances in two ways:

(1) We present a new inference rule for first-order superposition reasoning, called *induction with generalization*. Our work extends [8] by proving properties with multiple occurrences of the same induction term and by instantiating induction axioms with logically stronger versions of the property being proved.

(2) We implemented our work in the VAMPIRE theorem prover [7] and compared it to state-of-the-art reasoners automating induction, including CVC4 [2], ZENO [11], ZIPPERPOSITION [5] and IMANDRA [1]. We also provide a set of handcrafted mathematical problems over natural numbers and lists. We show that induction with generalization in VAMPIRE can solve problems that existing systems, including VAMPIRE without this rule, cannot.

## 2 Motivating Example

We assume familiarity with multi-sorted first order logic and the theory of finite term algebras, as well as saturation-based superposition reasoning, for details, we refer to [7,6]. We denote the equality predicate by $=$, the empty clause by $\square$, and use the term algebras of natural numbers $\mathbb{N}$ and lists $\mathbb{L}$ of natural numbers, as defined in Figure 1.

| | Natural numbers $\mathbb{N}$ | Natural lists $\mathbb{L}$ |
|---|---|---|
| Constructors | $0 : \mathbb{N} \qquad s : \mathbb{N} \to \mathbb{N}$ | $\texttt{nil} : \mathbb{L} \qquad \texttt{cons} : \mathbb{N} \times \mathbb{L} \to \mathbb{L}$ |
| Functions | $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$<br>$\leq : \mathbb{N} \times \mathbb{N} \to \text{bool}$ | $+\!+ : \mathbb{L} \times \mathbb{L} \to \mathbb{L}$<br>$\texttt{prefix} : \mathbb{L} \times \mathbb{L} \to \text{bool}$ |
| Axioms | $\forall y.(0 + y = y)$<br>$\forall x, y.(s(x) + y = s(x + y))$<br>$\forall x.0 \leq x$<br>$\forall x.\neg s(x) \leq 0$<br>$\forall x, y.\big(s(x) \leq s(y)$<br>$\qquad \leftrightarrow x \leq y\big)$ | $\forall l.(\texttt{nil} +\!+ l = l)$<br>$\forall x, l, k.(\texttt{cons}(x, l) +\!+ k = \texttt{cons}(x, l +\!+ k))$<br>$\forall l.\texttt{prefix}(\texttt{nil}, l)$<br>$\forall x, l.\neg\texttt{prefix}(\texttt{cons}(x, l), \texttt{nil})$<br>$\forall x, l, y, k.\big(\texttt{prefix}(\texttt{cons}(x, l), \texttt{cons}(y, k))$<br>$\qquad \leftrightarrow (x = y \wedge \texttt{prefix}(l, k))\big)$ |

**Fig. 1.** Term algebras of $\mathbb{N}$ and $\mathbb{L}$, together with additional functions and axioms.

We motivate our approach to induction with generalization, by considering the following formula expressing the associativity of addition over $\mathbb{N}$:

$$\forall x, y, z.(x + (y + z) = (x + y) + z), \qquad \text{with } x, y, z \in \mathbb{N}. \tag{1}$$

The induction approach introduced in [8] is able to prove this problem. The main steps of such a proof are shown in Figure 2 and discussed next. First, the negation of formula (1) is skolemized, yielding the (unit) clause $C_1$ of Figure 2. Throughout this paper, we use $\sigma, \sigma_i$ to denote fresh Skolem constants introduced during clausification. Next, the structural induction rule of $\mathbb{N}$ is instantiated so that its conclusion can resolve against $C_1$ using the constant $\sigma_1$ as the induction term, resulting in the formula:

$$\begin{array}{l}\big(0 + (\sigma_2 + \sigma_3) = (0 + \sigma_2) + \sigma_3 \wedge \\ \forall x.(x + (\sigma_2 + \sigma_3) = (x + \sigma_2) + \sigma_3) \to (s(x) + (\sigma_2 + \sigma_3)) = (s(x) + \sigma_2) + \sigma_3)\big) \\ \to \forall y.(y + (\sigma_2 + \sigma_3) = (y + \sigma_2) + \sigma_3).\end{array} \tag{2}$$

This formula is clausified and resolved against $C_1$, yielding clauses $C_2, C_3$ of Figure 2. Clause $C_4$ originates by repeated demodulation into $C_3$ using the second axiom of Figure 1 over $\mathbb{N}$. Further, $C_5$ is derived from $C_4$ by using the injectivity property of term algebras and $C_6$ is a resolvent of $C_2$ and $C_5$. Clause $C_7$ is then derived by repeated demodulation into $C_6$, using the the first axiom of Figure 1 over $\mathbb{N}$. By removing the trivial inequality from $C_7$, we finally derive the empty clause.

Consider now the following instance of associativity property (1):

$$\forall x.(x + (x + x) = (x + x) + x). \tag{3}$$

While (3) is an instance of (1), we cannot prove it using the same approach since we would add a different instance of structural induction, shown below, insufficient to prove either associativity or (3):

$$\begin{array}{l}\big(0 + (0 + 0) = (0 + 0) + 0 \wedge \\ \forall x.(x + (x + x) = (x + x) + x) \to (s(x) + (x + x)) = (s(x) + x) + x)\big) \\ \to \forall y.(y + (y + y) = (y + y) + y).\end{array} \tag{4}$$

The solution we consider in this paper is based on the idea of adding in addition to this instance of the induction schema, also instance (2).

| | |
|---|---|
| $(C_1)\ \sigma_1 + (\sigma_2 + \sigma_3) \not\approx (\sigma_1 + \sigma_2) + \sigma_3$ | [input] |
| $(C_2)\ 0 + (\sigma_2 + \sigma_3) \not\approx (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) = (\sigma + \sigma_2) + \sigma_3$ | [induct. $C_1$] |
| $(C_3)\ 0 + (\sigma_2 + \sigma_3) \not\approx (0 + \sigma_2) + \sigma_3 \vee s(\sigma) + (\sigma_2 + \sigma_3) \not\approx (s(\sigma) + \sigma_2) + \sigma_3$ | [induct. $C_1$] |
| $(C_4)\ 0 + (\sigma_2 + \sigma_3) \not\approx (0 + \sigma_2) + \sigma_3 \vee s(\sigma + (\sigma_2 + \sigma_3)) \not\approx s((\sigma + \sigma_2) + \sigma_3)$ | [$C_3$ + axiom] |
| $(C_5)\ 0 + (\sigma_2 + \sigma_3) \not\approx (0 + \sigma_2) + \sigma_3 \vee \sigma + (\sigma_2 + \sigma_3) \not\approx (\sigma + \sigma_2) + \sigma_3$ | [Injective $C_4$] |
| $(C_6)\ 0 + (\sigma_2 + \sigma_3) \not\approx (0 + \sigma_2) + \sigma_3$ | [res. $C_2$, $C_5$] |
| $(C_7)\ \sigma_2 + \sigma_3 \not\approx \sigma_2 + \sigma_3$ | [$C_6$ + axiom] |
| $(C_8)\ \square$ | [trivial ineq. $C_7$] |

**Fig. 2.** Proof of associativity of $+$ in a saturation-based theorem prover with induction

## 3 Induction with Generalization

Following [8], we consider an *induction schema/axiom* to be any valid (in the underlying theory, such as the theory of term algebras) formula of the form $premise \rightarrow (\forall y)(L[y])$. The work [8] introduces a rule of induction where a ground literal $\neg L[t]$ appearing in the proof search triggers addition of the corresponding induction axioms $premise \rightarrow (\forall y)(L[y])$ to the search space:

$$\frac{\neg L[t] \vee C}{\text{CNF}(premise \rightarrow (\forall y)(L[y]))}\ (\texttt{induction}), \tag{5}$$

where $L[y]$ is obtained from $L[t]$ by replacing *all* occurrences of $t$ by $y$.

While addition of a large number of such formulas may seem to blow up the search space, in practice VAMPIRE handles such addition with little overhead, resulting in finding proofs containing nearly 150 induction inferences [8]. The reason why the overhead of adding structural induction axioms is small comes from the fact that the added axioms are ground and contain at most one positive equality, as detailed in [9]. They are especially easy for the AVATAR architecture.

**Induction with generalization.** It is common in inductive theorem proving, giving a goal, to try to prove a more general goal instead. For example, given formula (3) to be proved we can try to prove (1) instead.

This makes no sense in saturation-based theorem proving. What we do instead is, given the same goal, add an induction axiom corresponding to a more general one. The rule can be formulated in the same way as (5), yet with a different condition:

$$\frac{\neg L[t] \vee C}{\text{CNF}(premise \rightarrow (\forall y)(L[y]))}\ (\texttt{IndGen}) \tag{6}$$

where $L[y]$ is obtained from $L[t]$ by replacing *some* occurrences of $t$ by $y$.

Both induction rules are obviously sound because their conclusions are valid in the underlying theory.

To implement `IndGen`, if a clause selected for inferences contains a ground literal $\neg L[t]$ having more than one occurrence of $t$, we should select a non-empty subset of occurrences of $t$ in $L[t]$, select an induction axiom corresponding to this subset, and then apply the rule.

**Motivating example, continued.** Suppose that $t$ is $\sigma_1$ and $\neg L[t]$ is $\sigma_1 + (\sigma_1 + \sigma_1) \not\simeq (\sigma_1 + \sigma_1) + \sigma_1$, which is obtained by negating and skolemizing (3). Then by applying `IndGen` we can add the following induction axiom:

$$\begin{array}{l}\big(0 + (\sigma_1 + \sigma_1) = (0 + \sigma_1) + \sigma_1 \wedge \\ \forall x.(x + (\sigma_1 + \sigma_1) = (x + \sigma_1) + \sigma_1) \rightarrow (s(x) + (\sigma_1 + \sigma_1)) = (s(x) + \sigma_1) + \sigma_1)\big) \\ \rightarrow \forall y.(y + (\sigma_1 + \sigma_1) = (y + \sigma_1) + \sigma_1),\end{array} \quad (7)$$

which is different from (4). When we add this formula, we can derive the empty clause in the same way as in Figure 2.

**Heuristics for induction with generalization.** The main questions to answer when applying induction with generalization is how many and which occurrences of the induction term in the induction literal to choose. An obvious heuristics is to try to use *all* non-empty subsets. However, this can result in too many formulas added even for one clause $\neg L[t] \vee C$, when the number of occurrences of $t$ is large. For example, $\sigma_1 + (\sigma_1 + \sigma_1) \not\simeq (\sigma_1 + \sigma_1) + \sigma_1$ would result in adding 63 induction formula. Note that addition of an induction formula may result in clauses containing many ground terms too, to which induction can again be applied.

Another heuristics can restrict the number of occurrences selected as induction term to a fixed number. However, possible heuristics for selecting subsets common cases of literals are out of scope of this paper.

## 4 Experiments

**Implementation.** We implemented induction with generalization in VAMPIRE[5], with two new options `indgen` and `indgenss`. Option `indgen`, with values `on`/`off` controls the application of induction with generalization, using (i) all subsets of occurrences of the induction term and (ii) subsets of size at most `indgenss`. In what follows, VAMPIRE refers to the (default) version of VAMPIRE with induction rule (5), and VAMPIRE* is its extended version with the `IndGen` rule of induction with generalization.

**SMT-LIB experiments.** We evaluated our work using the UFDT and UFDTLIA problem sets from SMT-LIB [3], yielding all together 4854 problems. Induction (5) in VAMPIRE was already evaluated in [8] against other solvers on these examples. Hence, we were only interested how VAMPIRE* performs against VAMPIRE. We ran our experiments on the StarExec cluster [12]. Compared to VAMPIRE, VAMPIRE* solved 4 new problems, using `IndGen` at most 3 times and having the depth of induction at most 4. We conclude that SMT-LIB does not yet contain sufficiently challenging benchmarks for which generalizations are needed. However, such examples would typically arise in mathematical properties over naturals/lists, as next discussed.

**Experiments with new challenges.** We handcrafted 32 mathematical properties to test inductive reasoning with generalization; Table 1 lists 15 such representative examples. By increasing the number of occurrences of induction terms in these examples, we also generated a set of 3637 examples, containing for example variations of (3) with 20 occurrences of $x$. We compared existing reasoners supporting induction on all these

[5] Our implementation is avialable at `https://github.com/vprover/vampire`, in the branch `hzzv-induction1`.

**Table 1.** Experiments on 15 handcrafted benchmarks. "✓" denotes success, "–" denotes failure.

| | Theory | VAMPIRE* | VAMPIRE | CVC4 | ZIPPERPOSITION | ZENO | IMANDRA | CVC4-GEN | ZIPREWRITE |
|---|---|---|---|---|---|---|---|---|---|
| $\forall x, y.(x + y = y + x)$ | $\mathbb{N}$ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| $\forall x.(s(x) + x = x + s(x))$ | | ✓ | – | – | – | – | – | ✓ | ✓ |
| $\forall x, y, z.(x + (y + z) = (x + y) + z)$ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| $\forall x.(x + (x + x) = (x + x) + x)$ | | ✓ | – | – | – | ✓ | – | ✓ | ✓ |
| $\forall x.(x+x)+((x+x)+x) = x+(x+((x+x)+x))$ | | ✓ | – | – | – | ✓ | – | ✓ | ✓ |
| $\forall x, y.(y + (x + x)) = ((x + y) + x)$ | | ✓ | – | – | – | – | – | – | ✓ |
| $\forall x, y.(x \leq x + y)$ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| $\forall x.(x \leq x + x)$ | | ✓ | – | – | – | – | – | – | – |
| $\forall x.(x + x \leq (x + x) + x)$ | | ✓ | – | – | – | ✓ | – | – | – |
| $\forall l, k, j.(l +\!\!+ (k +\!\!+ j) = (l +\!\!+ k) +\!\!+ j)$ | $\mathbb{L}$ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| $\forall l.(l +\!\!+ (l +\!\!+ l) = (l +\!\!+ l) +\!\!+ l)$ | | ✓ | – | – | – | – | – | – | ✓ |
| $\forall l, k.l +\!\!+ (k +\!\!+ (l +\!\!+ l)) = (l +\!\!+ k) +\!\!+ (l +\!\!+ l)$ | | ✓ | – | – | – | – | – | – | ✓ |
| $\forall l, k.\texttt{prefix}(l, l +\!\!+ k)$ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| $\forall l.\texttt{prefix}(l, l +\!\!+ l)$ | | ✓ | – | – | – | – | – | – | – |
| $\forall l : \mathbb{L}, x : \mathbb{N}.\texttt{cons}(x + s(x), l) +\!\!+ (l +\!\!+ l)$ $= (\texttt{cons}(s(x) + x, l) +\!\!+ l) +\!\!+ l$ | $\mathbb{N}, \mathbb{L}$ | ✓ | – | – | – | – | – | – | – |

problems[6]. We translated these problems, together with the corresponding axioms of Figure 1, in the input syntax of the respective prover: (i) SMT-LIB format for (versions of) VAMPIRE and CVC4; (ii) functional program encodings for ZENO and IMANDRA; (iii) the native input format of ZIPPERPOSITION. Except for IMANDRA (which is a cloud-based service), we ran our experiments on a 2,9 GHz Quad-Core Intel Core i7 machine. We ran each solver as a single-threaded process with a 5 second time limit. Our results are summarized in Table 1, where CVC4-GEN refers to the extension [10] of CVC4 with automatic lemma discovery; and ZIPREWRITE is the ZIPPERPOSITION version supporting heuristics using rewrite rules instead of function axiomatisations [5].

Table 1 shows that VAMPIRE* outperforms all solvers, including VAMPIRE itself. Compared to ZIPPERPOSITION, we note that VAMPIRE* does not necessarily depend on clause splitting performed by AVATAR. When considering solvers without fine-tuned heuristics, such as in ZIPREWRITE and CVC4-GEN, VAMPIRE* solves many more problems. Interestingly, ZIPREWRITE heuristics work well with addition and list concatenation, but not with orders. Further, CVC4-GEN heuristics prove associativity of addition, but not the list counterpart for concatenation. We believe our experiments show the advantage of using induction with generalization as a new inference rule, rather than a heuristic-driven approach.

## 5 Conclusions

We introduced a new rule for induction with generalization in saturation-based reasoning based on using induction axioms for generalizations of the of goals appearing during proof-search. Our experiments show that we solve many problems that existing

[6] all these problems are available on the url of our implementation

systems cannot solve. Future work includes designing heuristics to guide proof search, generalization on more complex terms and performing other kinds of generalization.

## Acknowledgements

We thank Giles Reger for discussions related to the work. We acknowledge funding supporting this work, in particular ERC starting grant 2014 SYMCAR 639270, EPSRC grant EP/P03408X/1, ERC proof of concept grant 2018 SYMELS 842066, the Wallenberg Academy fellowship 2014 TheProSE, and Austrian FWF research project W1255-N23.

## References

1. Imandra, https://docs.imandra.ai/imandra-docs/
2. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: International Conference on Computer Aided Verification. pp. 171–177. Springer (2011)
3. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2016)
4. Boyer, R.S., Moore, J.S.: A Computational Logic Handbook, Perspectives in computing, vol. 23. Academic Press (1979)
5. Cruanes, S.: Superposition with Structural Induction. In: Proc. of FRoCoS. pp. 172–188 (2017)
6. Kovács, L., Robillard, S., Voronkov, A.: Coming to Terms with Quantified Reasoning. In: Proc. of POPL. pp. 260–270 (2017)
7. Kovács, L., Voronkov, A.: First-order theorem proving and vampire. In: International Conference on Computer Aided Verification. pp. 1–35. Springer (2013)
8. Reger, G., Voronkov, A.: Induction in Saturation-Based Proof Search. In: Proc. of CADE. pp. 477–494 (2019)
9. Reger, G., Voronkov, A.: Induction in Saturation-Based Proof Search. EasyChair Smart Slide (2020), `https://easychair.org/smart-slide/slide/hXmP`
10. Reynolds, A., Kuncak, V.: Induction for SMT Solvers. In: Proc. of VMCAI. pp. 80–98 (2015)
11. Sonnex, W., Drossopoulou, S., Eisenbach, S.: Zeno: An Automated Prover for Properties of Recursive Data Structures. In: Proc. of TACAS. pp. 407–421 (2012)
12. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: A Cross-Community Infrastructure for Logic Solving. In: Proc. of IJCAR. pp. 367–373 (2014)
13. Voronkov, A.: AVATAR: The Architecture for First-Order Theorem Provers. In: Proc. of CAV. pp. 696–710. Springer-Verlag (2014)