Even Better Induction in Superposition Reasoning

Márton Hajdú[1], Petra Hozzová[1], Laura Kovács[1,2], and Andrei Voronkov[3,4]

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

Abstract.

1 Introduction

This paper aims to contribute to the automation of proving problems with inductive types in the area of saturation-based theorem proving. Specifically, it introduces (1) new techniques to generate induction formulas and (2) new inference rules in saturation-based proof search to tackle some of the issues inherent in this topic.

While induction itself has been a well-known mathematical tool for centuries, its automation in inductive proofs started only in the 70s with a sequence of theorem provers like ACL2, CLAM/OYSTER or IsaPlanner just to name a few. In the nowadays more and more prominent field of saturation-based theorem proving, induction has been relatively out of focus with some work done in e.g. ZipperPosition, Vampire or CVC4. These initial inductive techniques mostly utilize basic induction formulas for term algebra datatypes and integers and during saturation very little help is provided for the prover to be able to actually solve the problems. We address several of these issues in this paper.

2 Preliminaries

3 Motivating Example

Finding an induction formula that leads to a successful proof for an inductive problem is known to be undecidable. Some provers like ACL2 or IsaPlanner take into account the structure of any function present in an inductive goal when creating induction formulas and provide heuristics to select the most suitable one as the next proof step. Such heuristics are however still relatively uncommon in saturation-based theorem provers.

Example 1.

$$\forall x, y.(\texttt{even}(x) \wedge \texttt{even}(y)) \rightarrow \texttt{even}(x + y)$$

The function definitions for $\mathtt{even}$ and $+$ can be found in Figure ??. Negating and clausifying this conjecture yields:

$$\big\{\{\mathtt{even}(\sigma_0)\}, \{\mathtt{even}(\sigma_1)\}, \{\neg\mathtt{even}(\sigma_0 + \sigma_1)\}\big\}$$

One could select the clause $\{\neg\mathtt{even}(\sigma_0 + \sigma_1)\}$ for induction and use the simplest induction formula with either σ_0 or σ_1 as the induction term. E.g. in the former case, we get:

$$\begin{pmatrix} \mathtt{even}(0 + \sigma_1) \\ \forall z. \begin{pmatrix} \mathtt{even}(z + \sigma_1) \rightarrow \\ \mathtt{even}(\mathtt{s}(z) + \sigma_1) \end{pmatrix} \end{pmatrix} \rightarrow \forall x.\mathtt{even}(x + \sigma_1)$$

After clausification and binary resolution with given clause, we get:

$$\left\{ \begin{matrix} \{\neg\mathtt{even}(0 + \sigma_1) \vee \mathtt{even}(\sigma_2 + \sigma_1)\}, \\ \{\neg\mathtt{even}(0 + \sigma_1) \vee \neg\mathtt{even}(\mathtt{s}(\sigma_2) + \sigma_1)\} \end{matrix} \right\}$$

The base case can be solved by applying the base case of $+$ and binary resolving with $\{\mathtt{even}(\sigma_1)\}$ but the literal $\neg\mathtt{even}(\mathtt{s}(\sigma_2) + \sigma_1)$ can be only simplified once to get $\neg\mathtt{even}(\mathtt{s}(\sigma_2 + \sigma_1))$ because there is no matching case in the definition of $\mathtt{even}$. Inducting once again could solve the problem but in this context, each induction inference gives new Skolem constants, so after the second induction we are left with two induction hypotheses, none of which matches the current induction step conclusion literal:

$$\big\{\{\mathtt{even}(\sigma_2 + \sigma_1)\}, \{\mathtt{even}(\mathtt{s}(\sigma_3 + \sigma_1))\}, \{\neg\mathtt{even}(\sigma_3 + \sigma_1)\}\big\}$$

Using σ_1 as induction term does not help either – we cannot get rid of any constructor terms in the second argument position of $+$.

In this paper, we introduce some basic preprocessing of function definitions to get "matching" induction formulas and then – if different induction formulas are generated for a goal – combine these in a way that all function terms can be simplified in the case distinction.

In a saturation-based theorem prover, even the "correct" induction formula is useless if the inference rules prohibit rewriting function terms in inductive step and base cases, either because a function term cannot be expanded into its definition or a necessary induction hypothesis cannot be used, ultimately leading to a stuck proof.

Example 2. Given a unit-clause:

$$\{\mathtt{fltn}(\sigma_0, \mathtt{nil}) \neq \mathtt{fltn}_2(\sigma_0)$$

Proving the literal in this clause inductively leads to a simple case distinction in the induction formula with a base case $\mathtt{leaf}$ and a step case $\mathtt{node}(u, v, w)$ with induction hypotheses u and v. However, due to the large terms on the right-hand side of function definitions $\mathtt{fltn}$ and $\mathtt{fltn}_2$ for case $\mathtt{node}(u, v, w)$, the induction step conclusion cannot be simplified:

$$\mathtt{fltn}(\mathtt{node}(\sigma_0, \sigma_1, \sigma_2)) +\!\!+ \sigma_3 \neq \mathtt{fltn}_2(\mathtt{node}(\sigma_0, \sigma_1, \sigma_2), \sigma_3)$$

4 Function definitions and Structural Induction Improvements

One important ingredient in inductive reasoning is coming up with the right induction formulas. However, an induction formula is useless if the inference rules prohibit rewriting function terms in inductive step and base cases, either because a function term cannot be expanded into its definition or a necessary induction hypothesis cannot be used, ultimately leading to a stuck proof.

In particular, superposition and demodulation rules only apply when an equality obeys term, literal and clause orderings s.t. the resulting superposed or demodulated clause is smaller than the rewritten premise. For instance, the following function definition for flattening a binary tree into a list contains two branches, one of which cannot be oriented in the right way to be able to

5 Function definition rewriting

One important ingredient in inductive reasoning is coming up with the right induction formulas. However, an induction formula is useless if the inference rules prohibit rewriting function terms in inductive step and base cases, either because a function term cannot be expanded into its definition or a necessary induction hypothesis cannot be used, ultimately leading to a stuck proof.

In particular, superposition and demodulation rules only apply when an equality obeys term, literal and clause orderings s.t. the resulting superposed or demodulated clause is smaller than the rewritten premise. For instance, the following function definition for flattening a binary tree into a list contains two branches, one of which cannot be oriented in the right way to be able to

6 Experiments

7 Related Work

8 Conclusions

Acknowledgements