Tree grammars for induction on inductive data types modulo equational theories*

Gabriel Ebner and Stefan Hetzl

TU Wien

Abstract. Inductive theorem proving based on tree grammars was introduced in [9]. In this approach, proofs with induction on natural numbers are found by generalizing automatically generated proofs of finite instances on the level of Herbrand disjunctions. We extend this method to support general inductive data types, and reasoning modulo a background theory to abstract from irregularities in automatically generated proofs. We present an experimental implementation of the method and show that it automatically produces non-analytic induction formulas for several examples.

1 Introduction

The reasoning principle of induction is essential in mathematics and computer science, for example to prove properties of recursively defined data types such as lists or natural numbers. Proofs by induction are typically *non-analytic*: the induction formula does not necessarily occur in the problem to be proven. This non-analyticity makes automated inductive theorem proving a challenging endeavor: it requires the prover to synthesize an unknown predicate, the induction formula. See [18, Section 2] for a more thorough discussion of this aspect of induction.

Many approaches have been devised to address this problem, including rippling [4], theory exploration [6], term orders on formulas [22,20], heuristic special-purpose provers [21], in-processing in superposition provers [23,7], and cyclic proofs [3,19]. These approaches typically synthesize induction formulas by generalization or exhaustive enumeration.

In this paper, we present an extension of the recently developed approach using tree grammars [9] and an implementation of this extension. This approach divides the search for a proof of a universal formula into two distinct phases: the first phase finds a *grammar* that describes the quantifier inferences in the proof with induction based on proofs of small instances. The second phase then finds the actual induction formula. Non-analyticity is only required in the second phase, and then only in a limited form since the required formula is quantifier-free, even if the produced induction formula is quantified. Thus a problem with quantifiers is reduced to one without.

* Supported by the Vienna Science and Technology Fund (WWTF) project VRG12-004

The idea of first considering instances of a universal statement in order to prove this statement is not new. In the context of inductive theorem proving, this can for example be found in the work [1] on the constructive omega-rule. Similar ideas also exist in other contexts, for example bounded model checking [2]. What our approach adds to this basic idea is a thorough proof-theoretic analysis of the relationship between the proofs of the instances and the proof of the universal statement based on techniques like Herbrand's theorem, tree grammars and certain second-order unification problems.

Concretely, we present the following new results in this paper:

- We show how to support inductive data types other than natural numbers, and work in many-sorted logic.
- We reason modulo an equational background theory, in order to abstract away from irregularities present in automatically generated proofs and increase the robustness of the prover.
- We implement the algorithm and evaluate this implementation. We demonstrate that it finds non-analytic induction formulas on real-world examples.

In Sections 2 to 4 we first present the theoretical notions that underpin the prover: the connection between Herbrand sequents and tree languages as well as the class of problems that we consider is presented in Section 2, then Section 3 defines the class of proofs with induction that we want to find, and Section 4 introduces the notion of grammar that describes the quantifier inferences in such a proof with induction.

The rest of the paper is structured in the same order that the prover runs: Section 5 describes the algorithm used to compute grammars, Section 6 explains how we find the solution to the grammar—which will become the actual induction formula—and Section 7 constructs the proof with induction. Finally, in Section 8 we describe an experimental implementation of this algorithm and another example.

2 Herbrand sequents and tree languages

We consider a many-sorted first-order logic where some sorts are structurally inductive data types. A *structurally inductive data type* is a sort ρ with distinguished functions $c_1, \ldots, c_n$ called *constructors*. Each constructor c_i has the type $\tau_{i,1} \to \cdots \to \tau_{i,n_i} \to \rho$, that is, the arguments have the types $\tau_{i,1}, \ldots, \tau_{i,n_i}$ and the return type of the constructor is ρ. It may be the case that $\tau_{i,j} = \rho$, then the index j of such an argument is called a *recursive occurrence* in the constructor c_i. For simplicity, we do not consider mutually inductive types or other extensions.

We only require that ρ is generated by its constructors: there is no proper definable subset X of ρ such that we have $c_i(r_1, \ldots, r_{n_i}) \in X$ for all i whenever $r_j \in X$ for all recursive occurrences j. This property will be implemented by the inference rule for induction. Other properties such as the injectivity of the constructors are explicitly added as assumptions.

The intended semantics is that ρ is the set of finite terms freely generated by the constructors and values of the other argument types. For example, let the sort ω be a structurally inductive data type with the two constructors c_1 and c_2 of type ω and $\omega \to \omega$, resp. Then in the intended semantics, ω is interpreted as the set $\{c_1, c_2(c_1), c_2(c_2(c_1)), \dots\}$—a structure isomorphic to the natural numbers.

We aim to prove problems of the following kind:

Definition 1. *A* simple induction problem *is a sequent $\Gamma \vdash \forall x\, \varphi(x)$ where Γ is a list of universally quantified prenex formulas, $\varphi(x)$ quantifier-free, and the quantifier x ranges over an inductive sort ρ.*

Example 1 (Running example). Consider the inductive type of natural numbers with the constructor 0^ω and $s^{\omega\to\omega}$. Let $\mathrm{fact}^{\omega\to\omega}$ be a function symbol defined as usual for the factorial function, and $\mathrm{qfact}^{\omega\to\omega\to\omega}$ a "quick" tail-recursive implementation satisfying the following definitions:

$$\begin{aligned}
\Gamma = \{&\forall x\, (s(0)\cdot x = x \wedge x\cdot s(0) = x), && (f_1)\\
&\forall x\forall y\forall z\, x\cdot(y\cdot z) = (x\cdot y)\cdot z, && (f_2)\\
&\mathrm{fact}(0) = s(0), && (f_3)\\
&\forall x\, \mathrm{fact}(s(x)) = s(x)\cdot \mathrm{fact}(x), && (f_4)\\
&\forall y\, \mathrm{qfact}(y, 0) = y, && (f_5)\\
&\forall x\forall y\, \mathrm{qfact}(y, s(x)) = \mathrm{qfact}(y\cdot s(x), x)\} && (f_6)
\end{aligned}$$

Then the simple induction problem $\Gamma \vdash \forall x\, \mathrm{qfact}(s(0), x) = \mathrm{fact}(x)$ states the correctness of the quick implementation.

For simplicity, we only consider the case of a single universal quantifier in the conclusion. If there is more than one quantifier, we consider the generalized problem where all but one quantifier are instantiated with fresh Skolem constants.

When proving a simple induction problem, we assume an equational background theory E, consisting of a set of implicitly universally quantified equations that are contained in Γ. We say that a formula is a tautology if it is valid in propositional logic, a quasi-tautology if it is valid in propositional logic with equality (sometimes referred to as `QF_UF`), and E-tautology if it is valid in propositional logic with equality modulo E.

Example 2 (continuing Example 1). We use associativity of $\cdot$ and the unit laws as the equational theory $E = \{x\cdot(y\cdot z) = (x\cdot y)\cdot z, x\cdot s(0) = x, s(0)\cdot x = x\}$.

Finite instances $\Gamma \vdash \varphi(t)$ of the simple induction problem can typically be proven without the use of induction, and we can easily obtain proofs using automated theorem provers. We will now define the class of instances that we consider.

Definition 2. *Let ρ be a in inductive type with constructors $c_1, \dots, c_n$. The set of* constructor terms *of type ρ is the smallest set of terms containing $c_i(r_1, \dots, r_{i_n})$*

whenever it contains all r_j where j is a recursive occurrence in c_i. A free constructor term *is a constructor term where all subterms of a type other than ρ are pairwise distinct fresh constants.*

Example 3. For natural numbers, the terms $0, s(0), s(s(0))$ are free constructor terms, but $s(x)$ is not; all constructor terms are already free constructor terms. If we consider lists of natural numbers with the constructors nil and cons, then $\mathrm{nil}, \mathrm{cons}(a_1, \mathrm{nil}), \mathrm{cons}(a_1, \mathrm{cons}(a_2, \mathrm{nil}))$ are free constructor terms, but $\mathrm{cons}(x + x, \mathrm{cons}(x, \mathrm{nil}))$ is a constructor term that is not a free constructor term.

Definition 3. *Let $\Gamma \vdash \forall x\, \varphi(x)$ be a simple inductive problem, and t a free constructor term of type ρ. Then $\Gamma \vdash \varphi(t)$ is the* instance problem for t.

The relevant part of the proofs that we focus on are the quantifier instances of the formulas in Γ—these are given by (a special case of) Herbrand's theorem:

Theorem 1. *Let $\forall \overline{x}\, \theta_1[\overline{x}], \ldots, \forall \overline{x}\, \theta_n[\overline{x}] \vdash \varphi$ be a sequent where $\theta_1, \ldots, \theta_n, \varphi$ are quantifier-free formulas. Then the sequent is valid in first-order logic modulo the equational theory E if and only if there exist terms $\overline{t_{i,j}}$ for $1 \le i \le n$ and $1 \le j \le k_i$ such that $\theta_1[\overline{t_{1,1}}], \ldots, \theta_1[\overline{t_{1,k_1}}], \cdots, \theta_n[\overline{t_{n,1}}], \ldots, \theta_n[\overline{t_{n,k_n}}] \vdash \varphi$ is an E-tautology. This sequent of instances is called a* Herbrand sequent.

We encode Herbrand sequents as sets of terms by adding a new function symbol f_i for every formula θ_i (with the same type), the instance $\theta_i[\overline{t_{i,j}}]$ is encoded as the term $f_i(\overline{t_{i,j}})$. Terms of the form $f_i(\overline{t})$ for some i and $\overline{t}$ are called *decodable* if $\overline{t}$ does not contain any of the function symbols f_j. A set of terms L is called decodable if all of its terms are. We say that a decodable set of terms L is (quasi-/E-)tautological if the corresponding sequent of instances is.

Example 4. In our running example, the instance problem for $s(s(0))$ is $\Gamma \vdash \mathrm{qfact}(s(0), s(s(0))) = \mathrm{fact}(s(s(0)))$. The following set of terms L decodes to a Herbrand sequent of the instance problem for $s(s(0))$:

$$L = \{f_3, f_4(0), f_4(s(0)), f_5(s(0) \cdot s(s(0))), f_6(0, s(0) \cdot s(s(0))), f_6(s(0), s(0))\}$$

Here, $f_5(s(0) \cdot s(s(0)))$ decodes to the formula $\mathrm{qfact}(s(0) \cdot s(s(0)), 0) = s(0) \cdot s(s(0))$, where f_5 refers to the formula with that label in Example 1. Decoding L gives the following Herbrand sequent; we invite the reader to check for themselves that it is indeed E-tautological:

$$\begin{aligned}
&\mathrm{fact}(0) = s(0), \mathrm{fact}(s(0)) = s(0) \cdot \mathrm{fact}(0), \mathrm{fact}(s(s(0))) = s(s(0)) \cdot \mathrm{fact}(s(0)),\\
&\mathrm{qfact}(s(0) \cdot s(s(0)), 0) = s(0) \cdot s(s(0)),\\
&\mathrm{qfact}(s(0) \cdot s(s(0)), s(0)) = \mathrm{qfact}((s(0) \cdot s(s(0))) \cdot s(0), 0),\\
&\mathrm{qfact}(s(0), s(s(0)) = \mathrm{qfact}(s(0) \cdot s(s(0)), s(0))\\
&\vdash \mathrm{qfact}(s(0), s(s(0))) = \mathrm{fact}(s(s(0)))
\end{aligned}$$

3 Simple induction proofs

To describe the generated class of proofs, we use the sequent calculus LK with additional rules for the equational background theory and induction. For the background theory we add E-tautological atomic sequents as axioms. (This implicitly includes reasoning about equality as well.)

$$\frac{}{\Gamma \vdash \Delta}\,\mathrm{E} \quad \text{if } \Gamma \vdash \Delta \text{ is atomic and } E \models \bigwedge \Gamma \to \bigvee \Delta$$

For every inductive sort ρ there is a corresponding structural induction rule. This rule has one premise for each constructor c_i of the inductive type, and for every recursive argument α_{j_l} of the constructor there is an inductive hypothesis. The variables α_j are eigenvariables of the inference, that is, they may not occur in Γ or Δ.

$$\frac{\Gamma, \varphi(\alpha_{j_1}), \ldots, \varphi(\alpha_{j_{k_i}}) \vdash \Delta, \varphi(c_i(\alpha_1, \ldots, \alpha_{m_i})) \qquad \text{(for each } c_i\text{)}}{\Gamma \vdash \Delta, \varphi(t)}\,\mathrm{ind}_\rho$$

We can now define the class of simple induction proofs, these consist of a single induction followed by a cut. Note that the sequents below (π_i) and (π_c) are E-tautologies.

Definition 4. *Let ρ be an inductive type, $\Gamma \vdash \forall x\, \varphi(x)$ a simple induction problem, $\psi(x, w, \overline{y})$ a quantifier-free formula, $\Gamma_1, \ldots, \Gamma_n, \Gamma_c$ quantifier-free instances of Γ, and $\overline{t_{i,j,k}}, \overline{u_k}$ term vectors. Then a* simple induction proof *is a proof π of the following form, where $\pi_1, \ldots, \pi_n, \pi_c$ are cut-free proofs:*

$$\dfrac{\dfrac{\dfrac{(\pi_i)}{\Gamma_i, \psi(\alpha, \nu_{i,i_l}, \overline{t_{i,i_l,k}}), \cdots \vdash \psi(\alpha, c_i(\overline{\nu_i}), \overline{\gamma})}}{\Gamma, \forall \overline{y}\, \psi(\alpha, \nu_{i,i_l}, \overline{y}), \cdots \vdash \forall \overline{y}\, \psi(\alpha, c_i(\overline{\nu_i}), \overline{y})} \quad \cdots}{\Gamma \vdash \forall \overline{y}\, \psi(\alpha, \alpha, \overline{y})}\, ind_\rho \qquad \dfrac{\dfrac{(\pi_c)}{\Gamma_c, \psi(\alpha, \alpha, \overline{u_k}), \cdots \vdash \varphi(\alpha)}}{\Gamma, \forall \overline{y}\, \psi(\alpha, \alpha, \overline{y}) \vdash \varphi(\alpha)}}{\dfrac{\Gamma \vdash \varphi(\alpha)}{\Gamma \vdash \forall x\, \varphi(x)}}\, cut$$

Lemma 1. *Let $\Gamma \vdash \forall x\, \varphi(x)$ be a simple induction problem. If there exists a simple induction proof π of this simple induction problem, then there exist first-order proofs of all instance problems.*

Proof (sketch). By unrolling the induction in a similar way as in Gentzen's proof of the consistency of Peano Arithmetic [13,14].

Example 5. A natural simple induction proof of the problem in Example 1 uses the induction formula $\forall y\, \psi(\alpha, \nu, y)$ where $\psi(\alpha, \nu, y) = (\mathrm{qfact}(y, w) = y \cdot \mathrm{fact}(w))$. Formally the proof uses the instances and terms listed below. The following sections will then recover this data from instance proofs: we get the formula instances and terms from the grammar computed in Section 4, and the induction formula in Example 11 in Section 6.

$$\Gamma_1 = \{\mathrm{fact}(0) = s(0), \mathrm{qfact}(0, \gamma) = \gamma\}$$
$$\Gamma_2 = \{\mathrm{qfact}(\gamma, s(\nu)) = \mathrm{qfact}(\gamma \cdot s(\nu), \nu), \mathrm{fact}(s(\nu)) = s(\nu) \cdot \mathrm{fact}(\nu)\}$$
$$\Gamma_c = \emptyset \qquad t_{2,1,1} = \gamma \cdot s(\nu) \qquad u_1 = s(0)$$

4 Grammars

Just as sets of terms describe the quantifier inferences in proofs of the instance problems (via their decoding to Herbrand sequents), we use *induction grammars* to describe the quantifier inferences in the simple induction proof. The type o is the type of Booleans, the encoded instances $f_i(\overline{t})$ also have type o.

Definition 5. *An* induction grammar $G = (\tau, \alpha, (\overline{\nu_c})_c, \overline{\gamma}, P)$ *is a quintuple consisting of:*

1. *the start nonterminal, a nonterminal τ of type o,*
2. *a nonterminal α whose type ρ is an inductive sort,*
3. *a family of nonterminal vectors $(\overline{\nu_c})_c$, such that for each constructor c of the inductive sort ρ the term $c(\overline{\nu_c})$ is well-typed,*
4. *a nonterminal vector $\overline{\gamma}$, and*
5. *a set of vectorial productions P, where each production is of the form:*
   - *$\tau \to t[\alpha, \overline{\nu_i}, \overline{\gamma}]$ for some i, or*
   - *$\overline{\gamma} \to \overline{t}[\alpha, \overline{\nu_i}, \overline{\gamma}]$ for some i.*

Example 6. Let $\overline{\nu_0} = ()$, $\overline{\nu_s} = (\nu)$, and $\overline{\gamma} = \gamma$ where γ has the type ω. Then $G = (\tau, \alpha, (\overline{\nu_c})_c, \overline{\gamma}, P)$ is an induction grammar where the set P contains the following productions:

$$\tau \;\to\; f_3 \;\mid\; f_4(\nu) \;\mid\; f_5(\gamma) \;\mid\; f_6(\nu, \gamma) \qquad \gamma \;\to\; \gamma \cdot s(\nu) \;\mid\; s(0)$$

This induction grammar describes the quantifier instances in the simple induction proof of Example 5.

We will not directly define derivations and the generated language for inductions grammars. Instead we will define an instantiation operation that results in *vectorial totally rigid acyclic tree grammars* (VTRATG [8,16]), and define the language of the induction grammar as the language of the VTRATG obtained via instantiation.

Definition 6. *A VTRATG $G = (\tau, N, P)$ is a triple consisting of:*

- *the start nonterminal τ,*
- *a finite sequence $N = (\tau, \overline{\alpha_1}, \ldots, \overline{\alpha_n})$ of nonterminal vectors such that the nonterminals are pairwise distinct,*
- *and a finite set P of vectorial productions. A vectorial production is a pair $\overline{\alpha_i} \to \overline{t}$, where $\overline{\alpha_i} \in N$ is a nonterminal vector and $\overline{t}$ is a vector of terms of the same types as $\overline{\alpha}$ containing only nonterminals from $\overline{\alpha_{i+1}}, \ldots, \overline{\alpha_n}$.*

Similar to induction grammars, VTRATGs have a close connection to proofs: they describe quantifier inferences in proofs with universally quantified cuts but without induction, and also generate Herbrand sequents. For notational convenience, we may write α_0 or $\overline{\alpha_0}$ instead of τ. The language of a VTRATG is now defined as the set of all terms that we obtain by treating the productions as substitutions and applying them to τ:

Definition 7. *Let $G = (\tau, N, P)$ be a VTRATG. Its* language *is the following set of terms:* $L(G) = \{\tau[\overline{\alpha_0}\backslash\overline{t_0}]\cdots[\overline{\alpha_n}\backslash\overline{t_n}] \mid \overline{\alpha_0} \to \overline{t_0} \in P, \ldots, \overline{\alpha_n} \to \overline{t_n} \in P\}$

For terms t, s, we write $t \trianglelefteq s$ if t occurs as a subterm of s. The instantiation operation depends on a constructor term r as parameter, in the same way as the instance problem $\Gamma \vdash \varphi(r)$ uses a constructor term.

Definition 8. *Let $G = (\tau, \alpha, (\overline{\nu_c})_c, \overline{\gamma}, P)$ be an induction grammar, and r a constructor term of the same type as α. The* instance grammar $I(G, r) = (\tau, N, P')$ *is a VTRATG with nonterminal vectors $N = \{\tau\} \cup \{\overline{\gamma_s} \mid s \trianglelefteq r\}$ and productions $P' = \{p' \mid \exists p \in P\ (p \rightsquigarrow p')\}$. The instantiation relation $p \rightsquigarrow p'$ is defined as follows:*

- $\tau \to t[\alpha, \overline{\nu_i}, \overline{\gamma}] \rightsquigarrow \tau \to t[r, \overline{s}, \overline{\gamma_{c_i(\overline{s})}}]$ *for* $c_i(\overline{s}) \trianglelefteq r$
- $\overline{\gamma} \to \overline{t}[\alpha] \rightsquigarrow \overline{\gamma_s} \to \overline{t}[r]$ *for* $s \trianglelefteq r$
- $\overline{\gamma} \to \overline{t}[\alpha, \overline{\nu_i}, \overline{\gamma}] \rightsquigarrow \overline{\gamma_{s_j}} \to \overline{t}[r, \overline{s}, \gamma_{c_i(\overline{s})}]$ *for* $c_i(\overline{s}) \trianglelefteq r$, *where j is a recursive occurrence in c_i*

Example 7. Let us instantiate the induction grammar in Example 6 with the parameter $s(s(0))$. The instance grammar will have the nonterminals τ, γ_0, $\gamma_{s(0)}$, and $\gamma_{s(s(0))}$. We use the abbreviation $p \rightsquigarrow p'_1 \mid p'_2$ for $p \rightsquigarrow p'_1 \wedge p \rightsquigarrow p'_2$. The productions on the right side are all the productions in the instance grammar $I(G, s(s(0)))$.

$$\begin{aligned}
\tau \to f_3 &\rightsquigarrow \tau \to f_3 \\
\tau \to f_4(\nu) &\rightsquigarrow \tau \to f_4(0) \mid \tau \to f_4(s(0)) \\
\tau \to f_5(\gamma) &\rightsquigarrow \tau \to f_5(\gamma_0) \mid \tau \to f_5(\gamma_{s(0)}) \mid \tau \to f_5(\gamma_{s(s(0))}) \\
\tau \to f_6(\nu, \gamma) &\rightsquigarrow \tau \to f_6(0, \gamma_{s(0)}) \mid \tau \to f_6(s(0), \gamma_{s(s(0))}) \\
\gamma \to \gamma \cdot s(\nu) &\rightsquigarrow \gamma_0 \to \gamma_{s(0)} \cdot s(0) \mid \gamma_{s(0)} \to \gamma_{s(s(0))} \cdot s(s(0)) \\
\gamma \to s(0) &\rightsquigarrow \gamma_0 \to s(0) \mid \gamma_{s(0)} \to s(0) \mid \gamma_{s(s(0))} \to s(0)
\end{aligned}$$

We can now define the language in terms of the instance grammar. There is a different language for each constructor term.

Definition 9. *Let $G = (\tau, \alpha, (\overline{\nu_c})_c, \overline{\gamma}, P)$ be an induction grammar, and t a constructor term of the same type as α. Then we define the language at the term t as $L(G, t) = L(I(G, t))$.*

Example 8. The instance grammar in Example 7 produces the following language, which is a strict superset of the language in Example 4, and hence decodes to an E-tautology as well:

$$\begin{aligned}
L(G, s(s(0))) = \{&f_3, f_4(0), f_4(s(0)), f_5(s(0)), f_5(s(0) \cdot s(0)), \\
&f_5((s(0) \cdot s(s(0))) \cdot s(0)), f_5(s(0)), f_5(s(0) \cdot s(s(0))), \\
&f_6(0, s(0)), f_6(0, s(0) \cdot s(s(0))), f_6(s(0), s(0))\}
\end{aligned}$$

5 A refinement loop to find induction grammars

So far, the definition of induction grammar describes merely a family of tree languages without any reference to logic. In Section 2, we introduced the function symbols f_i to encode formulas instances in Herbrand sequents. Grammars that produce such terms whose root function symbol is f_i for some i are called decodable, since they produce decodable languages:

Definition 10. *Let G be an induction grammar for the inductive sort ρ, and $\Gamma \vdash \forall x\, \varphi(x)$ a simple induction problem. Then G is* decodable *iff for every production $\tau \to t \in G$, the right-hand side t is decodable.*

Lemma 2. *Let G be a decodable induction grammar. Then $L(G,t)$ is decodable for every constructor term t.*

Proof. Every τ-production in the instance grammar $I(G,t)$ has the form $\tau \to f_i(\dots)$ for some i. Hence all terms in the language of $I(G,t)$ are of the form $f_i(\dots)$ as well.

We implicitly identify terms of the form $f_i(\dots)$ with their corresponding formulas when there is no confusion. For example in the sequent $L(G) \vdash \varphi(t)$, the antecedent is the set of all formulas that are encoded as some term in $L(G)$.

The following lemma motivates our search for grammars. If there exists a simple induction proof, then there exists an induction grammar that generates Herbrand sequents for the instance problems. In the rest of this section, we will then reconstruct this grammar from automatically generated Herbrand sequents.

Lemma 3. *Let π be a simple induction proof of the simple induction problem $\Gamma \vdash \forall x \varphi(x)$. Then there exists an induction grammar G such that $L(G,t)$ decodes to an E-tautology for every free constructor term t.*

Proof (sketch). If we unroll the induction as in Lemma 1, then we get a proof whose quantifier inferences are described by the VTRATG $I(G,t)$. Since $L(I(G,t))$ decodes to a tautology, $L(G,t) = L(I(G,t))$ does as well. See also Proposition 3.10 in [9].

We want the languages of the induction grammar to describe Herbrand sequents for the instance problems. Hence we take automatically generated Herbrand sequents for a finite set of terms, and find a grammar that generates a superset. Note that a superset of a Herbrand sequent is still a Herbrand sequent. We could also require that the grammar generates the input Herbrand sequents exactly, but this requirement would make it much harder to find grammars.

Grammars for simple induction problems are generated using a refinement loop. We first obtain Herbrand sequents for a few random instances and find a covering grammar for the corresponding term sets. We then compute the language of the grammar on a different set of random instances, and check if it is E-tautological since we want the languages to describe Herbrand sequents. When

Algorithm 1 Refinement loop for grammar generation

```
procedure RefinementLoop(Γ, φ, σ, τ̄)
    G ← ∅
    L ← ∅
    while i ← MinimalCounterexample(G, φ) do
        π ← proof of Γ ⊢ φ(i)
        L ← L ∪ {i ↦ T(π)}
        G ← FindGrammar(σ, τ̄, L)
    end while
    return G
end procedure
procedure MinimalCounterexample(G, φ)
    for n ← 1 ... 10 do
        i ← random free constructor term
        if L(G, i) ⊢ φ(i) is not E-tautological then
            i ← minimal subterm of i such that L(G, i) ⊢ φ(i) is not E-tautological
            return i
        end if
    end for
end procedure
```

the language is not E-tautological, we repeat the process, adding Herbrand sequents for new random instances.

The procedure shown in Algorithm 1 calls automated provers in two places: first, when checking whether the quantifier-free sequent $L(G, i) \vdash \varphi(i)$ is E-tautological. For this problem, we pass the quantifier-free problem together with the quantified equations for the background theory to an SMT solver with a resource limit to guarantee termination. This approach is clearly not a complete decision procedure for E-tautology, but an incomplete solver is acceptable here since it just causes us to consider more instance proofs.

Secondly, we want to obtain a proof, or more accurately the set of instance terms of the first-order sequent $\Gamma \vdash \varphi(i)$. Here, we pass the sequent together with the universally quantified equations to a resolution-based first-order prover. The resulting proof is a proof in pure first-order logic, and will contain instances that are unnecessary when assuming the background theory E. To remedy this, we take a minimal subset of the instances that is still E-tautological (as checked by the SMT-based procedure above).

The rest of this section describes the function FindGrammar, which actually generates the grammar. Fix an inductive sort ρ and a nonterminal vector $\overline{\gamma}$.

Definition 11. *A* ρ-indexed term set *is a family* $(L_i)_{i \in I}$*, where* I *is a finite set of constructor terms of type* ρ*, and* L_i *is a finite set of terms for every* $i \in I$*.*

Definition 12. *Let* $(L_i)_{i \in I}$ *be an* ρ*-indexed term set, and* G *an inductive grammar with inductive type* ρ*. We say that* G covers $(L_i)_{i \in I}$ *if and only if* $L(G, i) \supseteq L_i$ *for every* $i \in I$*.*

Computational Problem 1 (Parameterized Indexed Termset Cover, PITCP*).*
INPUT: Indexed term set $(L_i)_{i\in I}$, set of function symbols Σ, nonterminals α, $\overline{\gamma}$.
OUTPUT: Induction grammar $G = (\tau, \alpha, (\overline{\nu_c})_c, \overline{\gamma}, P)$ such that G covers $(L_i)_{i\in I}$ and all productions use only function symbols in Σ.

The restriction on the function symbols is necessary because of the fresh constants we introduce in the instance terms. Recall that for lists, we use instance terms such as $\mathrm{cons}(a_0, \mathrm{nil})$ where a_0 is a fresh constant. This constant can not appear in the proof with induction, and hence should not occur in the grammar either.

The function FINDGRAMMAR then solves Computational Problem 1: it returns a covering induction grammar with the specified nonterminals if it exists, or fails otherwise. It is implemented by a reduction to the MaxSAT optimization problem based on the reduction for VTRATGs in [8] and similar to [9].[1]

For types such as the natural numbers, FINDGRAMMAR will never fail since there is always the trivial grammar that simply contains all productions $\tau \to t$ where $t \in \bigcup_i L_i$. Due to the restriction on the function symbols, it can fail to cover an indexed term set with other types, such as lists. Consider $\overline{\gamma} = ()$ and $L_{\mathrm{cons}(a_0,\mathrm{cons}(a_1,\mathrm{nil}))} = \{f_1(a_0, a_1)\}$, then there exists no covering grammar since every term in the generated instance language will contain only either a_0 or a_1, but not both.

6 Boolean Unification Problem and its solution

After we have computed a grammar G using Algorithm 1, we want to compute a induction formula for it that will allow us to construct a proof with induction that contains the quantifier inferences indicated by G. We collect the conditions for a formula to be a solution in a Boolean unification problem.

Computational Problem 2 (Boolean Unification Problem, BUP*).*
INPUT: Formula $\varphi(X(\overline{y}), \overline{y})$ with at most the free second-order variable X
OUTPUT: Is there a quantifier-free formula ψ such that $\varphi(\psi(\overline{y}), \overline{y})$ is an E-tautology?

Such a formula ψ is called a solution to the BUP. This problem is a generalization of the UBUP problem defined in [10], where it is shown to be undecidable in general.

Definition 13. *Let $G = (\tau, \alpha, (\overline{\nu}_c)_c, \overline{\gamma}, P)$ be an induction grammar. We define the following sets where $\overline{\kappa} \in \{\tau, \overline{\gamma}\}$, i is the index of a constructor, and j is c or an index of a constructor:*

- $P^i_{\overline{\kappa}} = \{\overline{t} \mid \overline{\kappa} \to \overline{t} \in P \wedge \mathrm{FV}(\overline{t}) \subseteq \{\alpha\} \cup \overline{\nu}_{c_i} \cup \overline{\gamma}\}$
- $P^c_{\overline{\kappa}} = \{\overline{t} \mid \overline{\kappa} \to \overline{t} \in P \wedge \mathrm{FV}(\overline{t}) \subseteq \{\alpha\}\}$

[1] See the appendix available at `https://gebner.org/pdfs/2018-01-29_indmodth_supplemental.pdf` for a detailed description.

- $\Gamma_j = P^j_\tau$
- $T_j = P^j_{\overline{\gamma}}$ *if* $P^j_{\overline{\gamma}} \neq \emptyset$, *otherwise* $T_j = \{\overline{\gamma}\}$

Definition 14. *Let G be an induction grammar for a simple induction problem $\Gamma \vdash \forall x\, \varphi(x)$, then the corresponding BUP consists of the conjunction of the following sequents:*

- $\Gamma_i, \bigwedge_l \bigwedge_{\overline{t} \in T_i} X(\alpha, \nu_{c_i,l}, \overline{t}) \vdash X(\alpha, c_i(\overline{\nu}_{c_i}), \overline{\gamma})$ *where i is the index of a constructor*
- $\Gamma_c, \bigwedge_{\overline{t} \in T_c} X(\alpha, \alpha, \overline{t}) \vdash \varphi(\alpha)$

A solution *is a formula $\psi(\alpha, \nu, \overline{\gamma})$ such that all sequents are E-tautological after substituting X by ψ.*

The sequents in Definition 14 are propositionally equivalent to the initial sequents in the simple induction proof in Definition 4. In the next section, we will use this fact to construct the proof with induction in Theorem 2.

Example 9. The induction grammar G in Example 6 has the following BUP:

$$\begin{gathered}
\mathrm{qfact}(\gamma, 0) = \gamma, \mathrm{fact}(0) = s(0), \top \vdash x(\alpha, 0, \gamma) \\
\mathrm{fact}(0) = s(0), \mathrm{fact}(s(\nu)) = s(\nu) \cdot \mathrm{fact}(\nu), \\
\mathrm{qfact}(\gamma, 0) = \gamma, \mathrm{qfact}(\gamma, s(\nu)) = \mathrm{qfact}(\gamma \cdot s(\nu), \nu), \\
X(\alpha, \nu, s(0)) \wedge X(\alpha, \nu, \gamma \cdot s(\nu)) \vdash X(\alpha, s(\nu), \gamma) \\
\mathrm{fact}(0) = s(0), X(\alpha, \alpha, s(0)) \vdash \mathrm{qfact}(s(0), \alpha) = \mathrm{fact}(\alpha)
\end{gathered}$$

The formula $\psi(\alpha, \nu, \gamma) = (\mathrm{qfact}(\gamma, \nu) = \gamma \cdot \mathrm{fact}(\nu))$ is a solution for this BUP.

In general it is undecidable whether such a BUP has a solution or not, see [10]. Moreover there exist grammars G where $L(G, t)$ is tautological for every t, but the corresponding BUP does not have a solution. However, we will now present a heuristic algorithm that is often successful. It is based on the corresponding search algorithm for cut-introduction introduced in [17] that has shown to be practically effective [15,11]. First, let us give an important restrictions on solutions. All solutions are consequences of a general "canonical formula" that is implied by the assumptions:

Definition 15. *Let G be an induction grammar and s a constructor term, then the* canonical formula *$C(s, \alpha, \overline{\gamma})$ is defined by recursion on s, where RO_i is the set of recursive occurrences in c_i:*

$$C(\alpha, c_i(\overline{s}), \overline{\gamma}) = \bigwedge \Gamma_i \wedge \bigwedge_{l \in \mathrm{RO}_i} \bigwedge_{\overline{\gamma} \to \overline{t} \in G} C(\alpha, s_l, \overline{t})$$

Example 10. Let G be the induction grammar from Example 6, let us compute the canonical formula $C(\alpha, s(0), \gamma)$:

$$\begin{aligned}
C(\alpha, 0, \gamma) &= \mathrm{qfact}(\gamma, 0) = \gamma \wedge \mathrm{fact}(0) = s(0) \\
C(\alpha, s(0), \gamma) &= \mathrm{fact}(0) = s(0) \wedge \mathrm{fact}(s(\nu)) = s(\nu) \cdot \mathrm{fact}(\nu) \wedge \mathrm{qfact}(\gamma, 0) = \gamma \\
&\quad \wedge \mathrm{qfact}(\gamma, s(\nu)) = \mathrm{qfact}(\gamma \cdot s(\nu), \nu) \wedge C(\alpha, 0, s(0)) \wedge C(\alpha, 0, \gamma \cdot s(\nu))
\end{aligned}$$

We can simplify $C(\alpha, 0, s(0))$ a bit to obtain the following equivalent formula, which is incidentally already in conjunctive normal form:

$$\begin{aligned}&\mathrm{fact}(0) = s(0) \wedge \mathrm{fact}(s(\nu)) = s(\nu) \cdot \mathrm{fact}(\nu) \wedge \mathrm{qfact}(\gamma, 0) = \gamma \wedge \\ &\mathrm{qfact}(\gamma, s(\nu)) = \mathrm{qfact}(\gamma \cdot s(\nu), \nu) \wedge \\ &\mathrm{qfact}(s(0), 0) = s(0) \wedge \mathrm{qfact}(\gamma \cdot s(\nu), 0) = \gamma \cdot s(\nu)\end{aligned}$$

Lemma 4. *Let G be an induction grammar and $\psi(\alpha, \nu, \overline{\gamma})$ a solution, then $C(\alpha, t, \overline{\gamma}) \to \psi(\alpha, t, \overline{\gamma})$ is an E-tautology for every constructor term t.*

Proof. By induction on t.

Our strategy to solve the BUP induced by an induction grammar now consists of finding solutions for a fixed instance t, i.e. first find suitable formulas $\psi(\alpha, t, \overline{\gamma})$ and then recover $\psi(\alpha, \nu, \overline{\gamma})$ by replacing (some occurrences of) t by ν. We will hence compute consequences $\chi[\alpha, \overline{\gamma}]$ of the formula $C(\alpha, t, \overline{\gamma})$ as long as $\Gamma_c, \bigwedge_{\overline{u} \in T_c} \chi(t, t, \overline{u}) \vdash \phi(t)$ is E-tautological, and then replace t in the results to generate candidates for the solution.

Definition 16. *Let S be the set of pairs where the first component is a set of equations, and the second component is a set of clauses (and clauses are sets of literals). We define the binary relations $\rhd_e, \rhd_p, \rhd_r$ on S as the smallest relations containing the following, where σ is a substitution:*

$$\begin{aligned}(\mathcal{E} \cup \{l = r\}, \mathcal{C} \cup \{C[l\sigma]\}) &\rhd_e (\mathcal{E}, \mathcal{C} \cup \{C[r\sigma]\}) \\ (\mathcal{E} \cup \{l = r\}, \mathcal{C} \cup \{C[r\sigma]\}) &\rhd_e (\mathcal{E}, \mathcal{C} \cup \{C[l\sigma]\}) \\ (\mathcal{E}, \mathcal{C} \cup \{C \cup \{l = r\}, D[l]\}) &\rhd_p (\mathcal{E}, \mathcal{C} \cup \{C \cup D[r]\}) \\ (\mathcal{E}, \mathcal{C} \cup \{C \cup \{l = r\}, D[r]\}) &\rhd_p (\mathcal{E}, \mathcal{C} \cup \{C \cup D[l]\}) \\ (\mathcal{E}, \mathcal{C} \cup \{C \cup \{l\}, \{\neg l\} \cup D\}) &\rhd_r (\mathcal{E}, \mathcal{C} \cup \{C \cup D\})\end{aligned}$$

The relations $\rhd_e, \rhd_p, \rhd_r$ are called simplification by forgetful equational rewriting, resolution, and paramodulation, respectively. We define the relation $\rhd = \rhd_e \cup \rhd_r \cup \rhd_p$ as their union.

Starting from $C(\alpha, t, \overline{\gamma})$, we compute consequences by first transforming the formula into conjunctive normal form $\mathcal{C}$, and compute all consequences $\mathcal{C}'$ such that $(E, \mathcal{C}) \rhd^* (\ldots, \mathcal{C}')$. We skip all branches where $\Gamma_c, \bigwedge_{\overline{u} \in T_c} \mathcal{C}[\alpha \backslash t, \overline{\gamma} \backslash \overline{u}] \vdash \phi(t)$ is not E-tautological, since they cannot lead to solutions. For each resulting $\mathcal{C}'$ we compute all possible generalizations of t by ν, and check whether the generalization solves the BUP.

Example 11. In order to find a solution for the BUP in Example 9, we start with $C(\alpha, 0, \gamma)$ and perform the simplifications as shown in Definition 16, where we abbreviate $A = (x \cdot (y \cdot z) = (x \cdot y) \cdot z)$:

$$\begin{aligned}(E, \mathcal{C}) = (\{A, s(0) \cdot x = x, x \cdot s(0) = x\}, \{\{\mathrm{qfact}(\gamma, 0) = \gamma\}, \{\mathrm{fact}(0) = s(0)\}\}) \\ \rhd_e (\{A, s(0) \cdot x = x\}, \{\{\mathrm{qfact}(\gamma, 0) = \gamma \cdot s(0)\}, \{\mathrm{fact}(0) = s(0)\}\}) \\ \rhd_p (\{A, s(0) \cdot x = x\}, \{\{\mathrm{qfact}(\gamma, 0) = \gamma \cdot \mathrm{fact}(0)\}\})\end{aligned}$$

We now obtain the solution $\mathrm{qfact}(\gamma, \nu) = \gamma \cdot \mathrm{fact}(\nu)$ by replacing 0 with ν in the last CNF $\mathrm{qfact}(\gamma, 0) = \gamma \cdot \mathrm{fact}(0)$.

7 Generated proof with induction

To conclude, we show that we can obtain a simple induction proof from a solution to the BUP. This implies the correctness of the prover.

Theorem 2. *Let G be an induction grammar, and $\psi(\alpha, \nu, \overline{\gamma})$ a solution for G. Then there exists a simple induction proof of the corresponding induction problem (with ψ as the induction formula).*

Proof. The simple induction proof uses the instances Γ_i and Γ_c as defined in Definition 13, the terms $\overline{u_k}$ are taken from the set T_c, and the terms $\overline{t_{i,j,k}}$ are taken from the set T_i. We now need to find the cut-free proofs π_c and π_i for every i. But the sequents we need to prove are almost exactly the sequents in the BUP (they are propositionally equivalent) and hence E-tautological. By completeness, we can then find cut-free proofs for all of these E-tautological sequents.

8 Discussion

The algorithm presented here has been implemented in the open-source GAPT [12] system for proof transformations, version 2.9[2]. Table 1 shows some of the non-analytic induction formulas that are automatically generated using this method.

The addition of a background theory was motivated by a prototypical examples: the tail-recursive factorial function shown in Example 1—we cannot even find a grammar without a background theory. The running example throughout this paper shows how the algorithm (as it is implemented) can now automatically solve it and find a non-analytic induction formula.

Another unexpectedly challenging problem involves a function d that doubles its argument:

$$\forall x\, 0 + x = x, \quad \forall x \forall y\, s(x) + y = s(x + y), \quad d(0) = 0, \quad \forall x\, d(s(x)) = s(s(d(x)))$$
$$\vdash \forall x\, d(x) = x + x$$

This problem has natural instance proofs where we normalize the instance problem using the equations on the left-hand side of the sequent: that is, the instance proof performs the calculation $d(s^n(0)) = \cdots = s^{2n}(d(0)) = s^{2n}(0) = s^n(0 + s^n(0)) = \cdots = s(s^{n-1}(0) + s^n(0)) = s^n(0) + s^n(0)$. The instances required for this computational proof are captured by the induction grammar with the following productions:

$$\tau \;\to\; f_1(\alpha) \;\mid\; f_2(\nu, \alpha) \;\mid\; f_3 \;\mid\; f_4(\nu)$$

[2] available at `https://logic.at/gapt`

Problem	Solution	Equational theory
$x + (x + x) = (x + x) + x$	$\nu + (\alpha + \alpha) = (\alpha + \alpha) + \nu$	$\{0 + x = x, x + 0 = x\}$
$\mathrm{fact}(x) = \mathrm{qfact}(s(0), x)$	$\mathrm{qfact}(\gamma, \nu) = \gamma \cdot \mathrm{fact}(\nu)$	associativity + units
$\mathrm{qrev}(\mathrm{qrev}(x, \mathrm{nil}), \mathrm{nil}) = x$	$\mathrm{qrev}(\mathrm{qrev}(\nu, \gamma), \mathrm{nil}) = \mathrm{qrev}(\gamma, \nu)$	$\emptyset$
$\mathrm{S}(\mathrm{cnt}(x, xs)) = \mathrm{cnt}(x, \mathrm{cons}(x, xs))$	$\mathrm{equals}(\nu, \nu)$	$\emptyset$
$\mathrm{last}(xs{+}{+}\mathrm{nil}) = \mathrm{last}(xs)$	$\nu{+}{+}\mathrm{nil} = \mathrm{nil}$	$\emptyset$

Table 1: Examples of automatically generated non-analytic lemmas

This induction grammar induces the following BUP Φ. With the empty equational theory $E = \emptyset$, the formula $d(\nu) = \nu + \nu$ does not solve this BUP and we conjecture Φ to be unsolvable:

$$\Phi = \begin{cases} 0 + \alpha = \alpha, d(0) = 0 \vdash X(\alpha, 0) \\ 0 + \alpha = \alpha, s(\nu) + \alpha = s(\nu + \alpha), \\ d(0) = 0, d(s(\nu)) = s(s(d(\nu))), X(\alpha, \nu) \vdash X(\alpha, s(\nu)) \\ 0 + \alpha = \alpha, d(0) = 0, X(\alpha, \alpha) \vdash d(\alpha) = \alpha + \alpha \end{cases}$$

Conjecture 1. The BUP Φ is unsolvable with the empty equational theory. That is, there is no quantifier-free formula ψ such that all sequents in $\Phi[X \backslash \psi]$ are quasi-tautologies.

However, with a non-empty equational theory this BUP can be solved by a quantified formula: if we set $E = \{0 + x = x, s(x) + y = s(x + y)\}$, then $\forall y\ \nu + s(y) = s(\nu + y) \wedge d(\nu) = \nu + \nu$ is a (quantified) solution for Φ. This observation suggests that we can effectively extend the class of solvable BUPs by also considering quantified solution. The algorithm presented in Section 6 only finds quantifier-free solutions, so we need to extend this algorithm as well.

9 Conclusion

We have presented an extension of the method in [9] to support general inductive data types and reason modulo an equational background theory. The equational background theory makes it practically much more effective since we can ignore many irregularities in the automatically generated instance proofs: in Example 1, the associativity axiom typically has instances that are difficult to generalize. The method in [9] can not find a grammar that covers these instances, but working modulo a background theory we can easily find a grammar because the irregular instances are filtered out.

In general, this does not solve the problem of finding regular families of instance proofs. We get irregular instance proofs for more than half of the problems in the TIP benchmark suite (tons of inductive problems, see [5])—in the sense that we cannot find a covering grammar. One potential way to obtain more regular families of proofs is to modify the search procedure of the automated theorem prover, for example to use an outermost-first strategy for rewriting,

which sometimes seems beneficial in our experience. Another option would be to perform a heuristic post-processing to regularize the instance proofs.

Instead of finding more regular families of instance proofs, we could also extend the grammars so that they can cover more irregular families. The induction grammars in this paper need to cover the term sets for the instance proofs exactly, without taking the background theory into account. We plan to extend the grammar generation algorithm so that it generates grammars that cover the input term sets modulo the background theory—that is, where every term in the input term set is E-equivalent to a term in the generated language.

We have not spent much effort yet in developing the algorithm to solve the BUP and there is still much room for improvement. Even without changing the algorithm drastically, a clever organization of the search that avoids an exhaustive computation of all $\triangleright$-simplifications and generalizations could improve the performance and success rate. Another important challenge is to generate quantified solutions as explained in Section 8. We also plan to investigate and integrate successful methods from model checking as well as loop invariant generation.

Finally, there are many configuration options in this method that are currently set manually, such as the number of quantifiers in the induction formula or the equational theory. For quantitative options such as the number of quantifiers, we intend to implement heuristics and portfolio modes.

References

1. Baker, S., Ireland, A., Smaill, A.: On the Use of the Constructive Omega-Rule within Automated Deduction. In: Voronkov, A. (ed.) Logic Programming and Automated Reasoning (LPAR) 1992. Lecture Notes in Computer Science, vol. 624, pp. 214–225 (1992)
2. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic Model Checking without BDDs. In: Cleaveland, R. (ed.) Tools and Algorithms for Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 1579, pp. 193–207. Springer (1999)
3. Brotherston, J., Gorogiannis, N., Petersen, R.L.: A generic cyclic theorem prover. In: Jhala, R., Igarashi, A. (eds.) Programming Languages and Systems: 10th Asian Symposium, APLAS. pp. 350–367. Springer (2012)
4. Bundy, A., Basin, D., Hutter, D., Ireland, A.: Rippling: Meta-level Guidance for Mathematical Reasoning. Cambridge Tracts in Theoretical Computer Science (No. 56) (2005)
5. Claessen, K., Johansson, M., Rosén, D., Smallbone, N.: TIP: Tons of inductive problems. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) Conferences on Intelligent Computer Mathematics. pp. 333–337 (2015)
6. Claessen, K., Rosén, D., Johansson, M., Smallbone, N.: Automating inductive proofs using theory exploration. In: 24th International Conference on Automated Deduction. pp. 392–406 (2013), 15
7. Cruanes, S.: Superposition with structural induction. In: Dixon, C., Finger, M. (eds.) 11th International Symposium on Frontiers of Combining Systems, FroCoS. Lecture Notes in Computer Science, vol. 10483, pp. 172–188 (2017)

8. Eberhard, S., Ebner, G., Hetzl, S.: Algorithmic compression of finite tree languages by rigid acyclic grammars. ACM Transactions on Computational Logic 18(4), 26:1–26:20 (Sep 2017)
9. Eberhard, S., Hetzl, S.: Inductive theorem proving based on tree grammars. Annals of Pure and Applied Logic 166(6), 665–700 (2015)
10. Eberhard, S., Hetzl, S., Weller, D.: Boolean unification with predicates. Journal of Logic and Computation 27(1), 109–128 (2017)
11. Ebner, G., Hetzl, S., Leitsch, A., Reis, G., Weller, D.: On the generation of quantified lemmas (2017), submitted
12. Ebner, G., Hetzl, S., Reis, G., Riener, M., Wolfsteiner, S., Zivota, S.: System description: GAPT 2.0. In: Olivetti, N., Tiwari, A. (eds.) International Joint Conference on Automated Reasoning (IJCAR). Lecture Notes in Computer Science, vol. 9706, pp. 293–301. Springer (2016)
13. Gentzen, G.: Die Widerspruchsfreiheit der reinen Zahlentheorie. Mathematische Annalen 112, 493–565 (1936)
14. Gentzen, G.: Neue Fassung des Widerspruchsfreiheitsbeweises für die reine Zahlentheorie. Forschungen zur Logik und zur Grundlegung der exakten Wissenschaften 4, 19–44 (1938)
15. Hetzl, S., Leitsch, A., Reis, G., Tapolczai, J., Weller, D.: Introducing quantified cuts in logic with equality. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) International Joint Conference on Automated Reasoning (IJCAR). Lecture Notes in Computer Science, vol. 8562, pp. 240–254. Springer (2014)
16. Hetzl, S., Leitsch, A., Reis, G., Weller, D.: Algorithmic introduction of quantified cuts. Theoretical Computer Science 549, 1–16 (2014)
17. Hetzl, S., Leitsch, A., Weller, D.: Towards Algorithmic Cut-Introduction. In: Logic for Programming, Artificial Intelligence and Reasoning (LPAR-18). Lecture Notes in Computer Science, vol. 7180, pp. 228–242. Springer (2012)
18. Hetzl, S., Wong, T.L.: Some observations on the logical foundations of inductive theorem proving. Logical Methods in Computer Science 13(4) (2017)
19. Kersani, A., Peltier, N.: Combining superposition and induction: A practical realization. In: Fontaine, P., Ringeissen, C., Schmidt, R.A. (eds.) Frontiers of Combining Systems - 9th International Symposium (FroCoS). Lecture Notes in Computer Science, vol. 8152, pp. 7–22. Springer (2013)
20. Reddy, U.S.: Term rewriting induction. In: 10th International Conference on Automated Deduction (CADE). pp. 162–177 (1990)
21. Sonnex, W., Drossopoulou, S., Eisenbach, S.: Zeno: An automated prover for properties of recursive data structures. In: Flanagan, C., König, B. (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 7214, pp. 407–421. Springer (2012)
22. Stratulat, S.: A Unified View of Induction Reasoning for First-Order Logic. In: Turing-100, The Alan Turing Centenary Conference. Manchester, United Kingdom (Jun 2012)
23. Wand, D., Weidenbach, C.: Automatic induction inside superposition, `http://people.mpi-inf.mpg.de/~dwand/datasup/d.pdf`, unpublished