Induction formulas and term structure [⋆]

Gabriel Ebner and Stefan Hetzl

TU Wien, Vienna, Austria

Abstract. Given a proof with induction of $\forall x\ \varphi(x)$ (where $\varphi(x)$ is quantifier-free) we can compute a cut-free proof $\varphi(n)$ for every $n \in \mathbb{N}$ by implementing the algorithm implicit in Gentzen's proof of the consistency of PA. We show that this induction-elimination is practically reversible using our implementation of an algorithm based on tree grammars, study the theoretical properties of this reversal, and argue that the algorithm's effectiveness is rooted in term-level structural regularity preserved by induction-elimination.

1 Introduction

The concept of analyticity in formal proofs is usually formulated as the subformula property [18]: in first-order logic this means that every formula occurring in the proof is a substitution instance of the formula to be proven. Proofs with induction are typically non-analytic, as they require new induction invariants. This non-analyticity poses a central challenge for automated theorem proving in this practically highly relevant logic.

Cut-elimination is perhaps the most fundamental operation in proof theory, first introduced by Gentzen [8]—it provides an algorithmic means to remove non-analyticity from proofs. By extending cut-elimination to also unfold induction inferences, we can eliminate induction (and cut) inferences from proofs of existential formulas, thus showing the consistency of Peano arithmetic [9].

The quantifier inferences in cut-free proofs directly contain the information necessary to form Herbrand disjunctions [11,1]. In the simplest case, a proof of a purely existential formula $\exists x\ \varphi(x)$ gives rise to the existence of a tautological disjunction of quantifier-free instances $\varphi(t_1) \vee \cdots \vee \varphi(t_n)$, a Herbrand disjunction. Herbrand sequents are the natural generalization where the validity of a prenex sequent is characterized by the existence of a tautological sequent of instances.

Proofs produced by cut-elimination typically possess a certain kind of regularity: e.g. applying cut-elimination to infinitary derivations corresponding to proofs in PA results in proofs with rank bounded by ϵ_0 [19]. In this paper we study the regularity of families of instance proofs obtained from a simple induction proof. Our approach is based on a proof-theoretic analysis using tree grammars [5]. Given a simple induction proof, we can extract a tree grammar that generates essentially the Herbrand sequents of the instance problems. This

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

operation is practically reversible: given a family of Herbrand sequents, we can reconstruct a covering tree grammar, and then use it to construct a proof with induction.

The theoretical foundation for this work was introduced in [5], which describes grammars for a class of proof with induction on natural numbers. We present the following new results here:

- We extend the approach to other inductive data types such as lists and trees.
- We introduce an equational background theory to abstract away from irregularities introduced by automated theorem provers.
- We study the theoretical properties of reconstruction of a class of induction proofs from Herbrand sequents.
- We have implemented this approach and use it to investigate practical reconstructibility in case studies.

In Section 2 we describe the theoretical setting: the class of simple induction proofs that we consider, how their induction grammars are defined, and under what condition induction grammars can be completed to simple induction proofs. Reconstructibility and undecidability questions for this approach are studied in Section 3. Finally, we apply the approach to a number of real-world proofs in a case study in Section 4 and discuss the results.

2 Proofs and grammars

2.1 Simple induction problems

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 $c_1, \dots, c_n$ called *constructors*. Each constructor c_i has the type $\tau_{i,1} \to \dots \to \tau_{i,n_i} \to \rho$, that is, the arguments have the types $\tau_{i,1}, \dots, \tau_{i,n_i}$ 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.

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 consider proofs of the following kind of sequents:

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

Example 1 (Running example). Consider the inductive type ω of natural numbers with the constructor 0^ω and $s^{\omega \to \omega}$, and $\mathit{\Gamma}$ as follows. Then the simple induction problem $\Gamma \vdash \forall x\, d(x) = x + x$ states that the doubling function d can be expressed in terms of addition.

$$\begin{aligned} \Gamma = \{&\forall x\, x + 0 = x, && (f_1) && d(0) = 0, && (f_3) \\ &\forall x \forall y\, x + s(y) = s(x + y), && (f_2) && \forall x\, d(s(x)) = s(s(d(x))) && (f_4)\} \end{aligned}$$

For technical reasons, we only consider the case of a single universal quantifier in the conclusion. We can still treat problems that would naturally be stated using multiple quantifiers by instantiating all but one quantifier with fresh constants: e.g. for commutativity we get the simple induction problem $\Gamma \vdash \forall x\, x + c = c + x$.

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 quantifier-free formula φ is a tautology if it is valid in propositional logic, a quasi-tautology if it is valid in first-order logic with equality, and E-tautology if $\forall E \to \varphi$ is valid.

Example 2 (continuing Example 1). For the running example, we use the equational theory $E = \{d(0) = 0, d(s(x)) = s(s(d(x))), x{+}0 = x\}$ in order to conserve space. This choice effectively ignores the instances of all formulas except (f_3), which contains interesting information about the induction.

Definition 2. *Let ρ be a in inductive type with constructors $c_1, \ldots, c_n$. The set of* constructor terms *of type ρ is the smallest set of terms containing for each constructor c_i all terms $c_i(r_1, \ldots, r_{i_n})$ 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. We denote the set of free constructor terms by $\mathcal{C}$.*

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 induction problem, and t a free constructor term of type ρ. Then $\Gamma \vdash \varphi(t)$ is the* instance problem *for the parameter t.*

2.2 Simple induction proofs

We consider 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.)

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

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 $\alpha_1, \ldots, \alpha_{m_i}$ (for each constructor c_i) are eigenvariables of the inference, that is, they may not occur in $\varGamma$ or $\varDelta$.

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

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

Definition 4. *Let ρ be an inductive type, $\varGamma \vdash \forall x\, \varphi(x)$ a simple induction problem, $\psi(x, w, \overline{y})$ a quantifier-free formula, $\varGamma_1, \ldots, \varGamma_n, \varGamma_c$ quantifier-free instances of $\varGamma$, 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:*

$$\cfrac{\cfrac{\cfrac{\cfrac{(\pi_i)}{\varGamma_i, \psi(\alpha, \nu_{i,i_l}, \overline{t_{i,i_l,k}}), \cdots \vdash \psi(\alpha, c_i(\overline{\nu_i}), \overline{\gamma})}}{\varGamma, \forall \overline{y}\, \psi(\alpha, \nu_{i,i_l}, \overline{y}), \cdots \vdash \forall \overline{y}\, \psi(\alpha, c_i(\overline{\nu_i}), \overline{y})} \qquad \cdots}{\varGamma \vdash \forall \overline{y}\, \psi(\alpha, \alpha, \overline{y})}\,ind_\rho \qquad \cfrac{\cfrac{(\pi_c)}{\varGamma_c, \psi(\alpha, \alpha, \overline{u_k}), \cdots \vdash \varphi(\alpha)}}{\varGamma, \forall \overline{y}\, \psi(\alpha, \alpha, \overline{y}) \vdash \varphi(\alpha)}}{\cfrac{\varGamma \vdash \varphi(\alpha)}{\varGamma \vdash \forall x\, \varphi(x)}}\,cut$$

Example 4. We consider a simple induction proof of Example 1 with the induction formula $\forall y\, \psi(\alpha, \nu, y)$ where $\psi(\alpha, \nu, y) = (s(y) + \nu = s(y + \nu) \land d(\nu) = \nu + \nu)$. Formally the proof contains the following instances and terms:

$$\varGamma_2 = \{s(\gamma) + s(\nu) = s(s(\gamma) + \nu), \gamma + s(\nu) = s(\gamma + \nu), s(\nu) + s(\nu) = s(s(\nu) + \nu)\}$$
$$\varGamma_1 = \emptyset \qquad \varGamma_c = \emptyset \qquad t_{2,1,1} = \gamma \qquad t_{2,1,2} = \nu \qquad u_1 = 0$$

Lemma 1. *Let $\varGamma \vdash \forall x\, \varphi(x)$ be a simple induction problem. If there exists a simple induction proof π for the simple induction problem, then for every t there exists a first-order proof π_t for the instance problem with parameter t.*

Proof (sketch). By unrolling the induction in a similar way as in Gentzen's proof of the consistency of Peano Arithmetic [9,10]. For natural numbers see [5]. □

2.3 Herbrand sequents and tree languages

The relevant part of the proofs that we focus on are the quantifier instances of the formulas in $\varGamma$—these are given by (a special case of) Herbrand's theorem [11]:

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 $1 \leq i \leq n$ and $1 \leq j \leq k_i$ 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, the instance $\theta_i[\overline{t_{i,j}}]$ is encoded as the term $f_i(\overline{t_{i,j}})$, and conversely, the term $f_i(\overline{t_{i,j}})$ decodes to $\theta_i[\overline{t_{i,j}}]$. We say that a set of terms L is (quasi-/E-)tautological iff the decoded sequent of instances is.

Cut- and induction-free proofs directly contain the terms $t_{i,j}$ in the quantifier inferences [1]. Given such a proof π, we write $L(\pi)$ for the encoded Herbrand sequent extracted from π. In particular, given a simple induction proof π and an instance term t, the set $L(\pi_t^*)$ decodes to a Herbrand sequent for the corresponding instance problem.

Example 5. In our running example, the instance problem for $s(s(0))$ is $\Gamma \vdash d(s(s(0))) = s(s(0))+s(s(0))$. The following set of terms L decodes to a Herbrand sequent of the instance problem for $s(s(0))$: $L = \{f_2(s^2(0), s(0)),\ f_2(s^3(0), 0)\}$

Here, $f_2(s^2(0), s(0))$ decodes to the formula $s^2(0) + s^2(0) = s(s^2(0) + s(0))$, where f_2 refers to the formula with that label in Example 1. Decoding L gives a Herbrand sequent; we invite the reader to check for themselves that it is indeed E-tautological.

2.4 Grammars

Just as sets of terms describe the quantifier inferences in proofs of the sequents 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 encoded instances $f_i(\bar{t})$ have type o, the type of Booleans.

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

1. *the start 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 $\tau \to t[\alpha, \overline{\nu_i}, \overline{\gamma}]$ or $\overline{\gamma} \to \bar{t}[\alpha, \overline{\nu_i}, \overline{\gamma}]$ for some i.*

Example 6. Let $\overline{\nu_0} = ()$, $\overline{\nu_s} = (\nu)$, and $\overline{\gamma} = (\gamma)$ where γ has the type ω. Then the induction grammar $G = (\tau, \alpha, (\overline{\nu_c})_c, \overline{\gamma}, P)$ describes the quantifier instances in the simple induction proof of Example 4 (with the following productions P):

$$\tau \to f_2(s(\gamma), \nu) \mid f_2(\gamma, \nu) \mid f_2(s(\nu), \nu) \qquad \gamma \to \gamma \mid \nu \mid 0$$

An induction grammar generates a family of languages: each constructor term induces a language. We omit the definition here for space reasons, for details see Appendix A, or [5] for the special case of natural numbers.

Definition 6. *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 $L(G, t)$.*

Example 7. The induction grammar in Example 6 produces the following language, which decodes to an E-tautology:

$$\begin{aligned}L(G, s(s(0))) = \{&f_2(s(0), 0), f_2(s(s(0)), 0), f_2(s(0), s(0)),\\ &f_2(0, 0), f_2(s(0), 0), f_2(0, s(0))\}\end{aligned}$$

Recall that $L(G(\pi), t)$ is a set of terms that encodes a Herbrand sequent for the instance problem with parameter t. The computation of the Herbrand sequent on the term-level using grammars closely mirrors the Herbrand sequents obtained via induction-unfolding and cut-elimination on the level of proofs (that is, $L(\pi_t^*)$): the language generated by the grammar is a superset of the language extracted from the proof. That $L(G(\pi), t)$ is (in general) a strict superset of $L(\pi_t^*)$ (and not exactly equal) is a subtlety introduced by weakening inferences in π: quantifier inferences may be deleted when reducing weakening inferences during cut-elimination. But the grammar still generates these delete terms, since it intentionally abstracts away from the propositional reasoning of the proof.

Theorem 2. *Let π be a simple induction proof and t an instance term. Then $L(G(\pi), t) \supseteq L(\pi_t^*)$.*

$$\begin{tikzcd}
\pi \arrow[r, mapsto] \arrow[d, mapsto] & \pi_t \arrow[r, mapsto] \arrow[d, mapsto] & \pi_t^* \arrow[d, mapsto] \\
G(\pi) \arrow[r, mapsto] & G(\pi_t) \arrow[r, mapsto] & L(G(\pi), t) \subseteq L(\pi_t^*)
\end{tikzcd}$$

Proof (sketch). The theorem follows from a similar result for proofs ψ with universally quantified prenex cuts [12], where we also have $L(G(\psi)) \supseteq L(\psi^*)$. Unfolding the simple induction proof π results in a proof π_t with universally quantified prenex cuts. The language $L(G(\pi), t)$ is defined exactly in such a way that $L(G(\pi), t) = L(G(\pi_t))$, see Appendix A and [5] for details. □

In particular, Theorem 2 tells us that induction grammars extracted from proofs produce Herbrand languages for every instance:

Definition 7. *A family of languages $(L_t)_{t\in I}$ is called (E-)*tautological *if L_t is E-tautological for every t. An induction grammar G is called* tautological *if $(L(G, t))_{t\in\mathcal{C}}$ is tautological, i.e., the language $L(G, t)$ is E-tautological for every constructor term t.*

Corollary 1. *Let π be a simple induction proof. Then the induction grammar G is tautological.*

2.5 Solvability and formula equations

Not all induction grammars correspond to simple induction proofs—that is, there might be no induction formula such that we can construct a simple induction proof. We can collect the necessary conditions into a formula equation (FE), defined in full generality as follows:

Definition 8. *A* formula equation Ψ *is a formula with a free second-order predicate variable* X. *A* solution *modulo the theory* E *is a formula* φ *such that* $E \vdash \Psi[X \backslash \varphi]$.

Example 8. $\Psi = (P(a) \to X(a)) \wedge (X(b) \to P(b))$ has the solution $\varphi(x) := P(x)$.

The concept of formula equations can be traced back to the 19th century, see [20] for an overview. Finding a quantifier-free solution for a prenex universally quantified formula equation is the UBUP problem defined in [6], and is in general undecidable.

Definition 9. *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:*

- $P^i_{\overline{\kappa}} = \{\overline{t} \mid \overline{\kappa} \to \overline{t} \in P \wedge \mathrm{FV}(\overline{t}) \subseteq \{\alpha\} \cup \overline{\nu}_{c_i} \cup \overline{\gamma}\}$
- $P^c_{\overline{\kappa}} = \{\overline{t} \mid \overline{\kappa} \to \overline{t} \in P \wedge \mathrm{FV}(\overline{t}) \subseteq \{\alpha\}\}$
- $\Gamma_j = P^j_\tau$
- $T_j = P^j_{\overline{\gamma}}$ *if* $P^j_{\overline{\gamma}} \neq \emptyset$, *otherwise* $T_j = \{\overline{\gamma}\}$

Definition 10. *Let G be an induction grammar for a simple induction problem* $\Gamma \vdash \forall x\, \varphi(x)$, *then the corresponding FE* Φ_G *is the conjunction of the following formulas:*

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

Theorem 3. *The formula equation* Φ_G *is solvable iff there exists a simple induction proof* π *such that* $G(\pi) = G$.

Proof. The formulas in Definition 10 are equivalent to the initial sequents in the simple induction proof in Definition 4. Hence whenever the FE Φ_G has a quantifier-free solution modulo the background theory E, we can construct a simple induction proof π with the solution as induction formula such that $G(\pi) = G$. Vice versa, if we have a simple induction proof π then its induction formula is a solution for Φ_G. □

Example 9. The induction grammar G in Example 6 induces the following FE:

$$X(\alpha, 0, \gamma) \wedge (X(\alpha, \alpha, 0) \to d(\alpha) = \alpha + \alpha) \wedge$$
$$\forall \nu \forall \gamma\ \big(X(\alpha, \nu, \gamma) \wedge X(\alpha, \nu, \nu) \wedge X(\alpha, \nu, 0) \wedge s(\gamma) + s(\nu) = s(s(\gamma) + \nu) \wedge$$
$$\gamma + s(\nu) = s(\gamma + \nu) \wedge s(\nu) + s(\nu) = s(s(\nu) + \nu) \to X(\alpha, s(\nu), \gamma)\big)$$

This FE has the solution $\varphi(\alpha, \nu, \gamma) := (s(\gamma) + \nu = s(\gamma + \nu) \wedge d(\nu) = \nu + \nu)$.

3 Reconstruction

In Section 2, we started out with a simple induction proof π and described its induction grammar and induced formula equation. Via induction-elimination we obtained a family $(L_t)_{t \in \mathcal{C}}$ of instance languages, where $L_t := L(\pi^*_t)$ for each free constructor term $t \in \mathcal{C}$. By Corollary 1, L_t is tautological for each t, that is, it decodes to a Herbrand sequent for the instance problem with parameter t.

Definition 11. *A* family $(L_t)_{t \in I}$ of Herbrand languages *is a function from a set of free constructor terms $I \subseteq \mathcal{C}$ to languages, such that L_t is tautological for each $t \in I$.*

We now aim to invert this construction: starting from a finite family $(L_t)_{t \in I}$, we aim to reconstruct a simple induction proof π' such that $L({\pi'}^*_t) \supseteq L_t$ for all $t \in I$. For this section we fix a simple induction problem and $\overline{\gamma}$.

Definition 12. *Let $(L_t)_{t \in I}$ be a family of Herbrand languages. Then $(L_t)_{t \in I}$ is* regular *iff $\exists \pi\, \forall t\, L_t \subseteq L(\pi^*_t)$, and* grammatically regular *iff $\exists G\, \forall t\, L_t \subseteq L(G, t)$.*

Every regular family $(L_t)_{t \in I}$ is also grammatically regular by Theorem 2. Grammatical regularity is a statement purely on the level of formal languages, it does not imply the existence of an induction formula. However, if there is any simple induction proof for the problem at all, then grammatical regularity and regularity are equivalent:

Theorem 4. *Let $(L_t)_{t \in I}$ be a grammatically regular family. Then $(L_t)_{t \in I}$ is regular iff there exists a simple induction proof for the problem.*

Proof. If the family is regular then there exists a simple induction proof by definition. For the other direction, assume that G is an induction grammar covering the family and π is a simple induction proof with induction invariant φ. Consider the induction grammar $G' = G \cup G(\pi)$ (taking the union of the productions). Since φ is a solution for Φ_G, it is easy to see that it is also a solution for $\Phi_{G'}$. Hence we can construct a simple induction proof π' with $G(\pi') = G'$ by Theorem 3, which witnesses the regularity of $(L_t)_{t \in I}$. $\square$

Theorem 5 ([5]). *The set of simple induction problems that have a simple induction proof is computably enumerable (but not decidable).*

Theorem 6. *The set of grammatically regular families $(L_t)_{t \in I}$ with finite I is decidable. The set of regular families $(L_t)_{t \in I}$ with finite I is computably enumerable (but not decidable).*

Proof. First we show that grammatical regularity is decidable. Note that whenever there is a induction grammar G such that $L(G, t) \supseteq L_t$ for all $t \in I$, there is also one that contains only generalizations of subterms in $(L_t)_{t \in I}$—hence there is a straightforward upper bound on the symbolic complexity of G and we can simply iterate through all induction grammars of smaller symbolic complexity to find it. If we can find a covering induction grammar, then the family is grammatically regular, otherwise not.

Let us now consider regularity. For computable enumerability, we just enumerate all possible simple induction proofs and check whether $L(\pi_t^*) \supseteq L_t$ for all $t \in I$, all of which are computable operations. For undecidability, set $I = \emptyset$ and use Theorem 5. $\square$

Let us now turn to algorithms on infinite families $(L_t)_{t\in\mathcal{C}}$. Formally, we assume that the family is given as an oracle that returns L_t when given $t \in \mathcal{C}$.

Theorem 7. *For families $(L_t)_{t\in\mathcal{C}}$, grammatical regularity and regularity are both undecidable.*

Proof. Fix a regular family $(L_t)_{t\in\mathcal{C}}$. Any algorithm that decides (grammatical) regularity returns a result in finite time and hence only accesses a finite subfamily $(L_t)_{t\in I}$ for $I \subseteq \mathcal{C}$ finite. Hence it necessarily returns the same result for any other family that extends $(L_t)_{t\in I}$.

An easy example for a family that is not regular is one that grows too fast—by straightforward counting we see that $|L(G,t)| \leq |P|^{|t|}$ where $|P|$ is the number of productions in G, thus any family that grows faster cannot be (grammatically) regular. Hence we can extend $(L_t)_{t\in I}$ to a family $(L'_t)_{t\in\mathcal{C}}$ which is not grammatically regular, but which the algorithm claims to be (grammatically) regular. $\square$

Theorem 8 (learnability in the limit). *There is an algorithm that takes a family $(L_t)_{t\in\mathcal{C}}$ as input, and produces a sequence of induction grammars $(G_n)_{n\geq 0}$ with the following property: if $(L_t)_{t\in\mathcal{C}}$ is grammatically regular, then there exists an $N \geq 0$ such that $G_n = G_N$ for all $n \geq N$, and $L(G_N, t) \supseteq L_t$ for all t.*

(The sequence $(G_n)_{n\geq 0}$ is eventually const. iff $(L_t)_{t\in\mathcal{C}}$ is grammatically regular.)

Proof. Enumerate the free constructor terms as $\mathcal{C} = \{t_1, t_2, \dots\}$. Now for $i \geq 0$ compute an induction grammar H_i such that $L(H_i, t_j) \supseteq L_{t_j}$ for $j \leq i$ and H_i is of minimal symbolic complexity. As in Theorem 6, we can iterate through all induction grammars of a certain size to find H_i. The output of the algorithm is then $G_i = \bigcup_{j\leq i} H_i$ where the union operation on grammars is defined as the union of the sets of productions.

If $(L_t)_{t\in\mathcal{C}}$ is grammatically regular, i.e., there exists a covering induction grammar G, then the symbolic complexity of H_i is bounded by the symbolic complexity of G for all i. Hence there are only finitely many possibilities for H_i, and also for G_i. Since $(G_i)_i$ is monotonically increasing, it must be eventually constant with value G_N. We have $L(G_N, t_j) = L(G_{N+j}, t_j) \supseteq L(H_{N+j}, t_j) \supseteq L_{t_j}$ by construction. $\square$

Theorem 9 ([5]). *There exists a regular family $(L_t)_{t\in\mathcal{C}}$ and an induction grammar G such that $L(G,t) \supseteq L_t$ for all t, but Φ_G is not solvable.*

Theorem 10 (reconstructability in the limit). *There is an algorithm that takes a family $(L_t)_{t\in\mathcal{C}}$ as input, and produces a sequence $(\pi_n)_{n\geq 0}$, where each π_n*

is either a simple induction proof or $\emptyset$. The output has the following property: if $(L_t)_{t\in\mathcal{C}}$ is regular, then there exists an $N \geq 0$ such that $\pi_n = \pi_N \neq \emptyset$ for all $n \geq N$, and $L((\pi_N)^*_t) \supseteq L_t$ for all t.

(The sequence $(\pi_n)_{n\geq 0}$ is eventually constant iff $(L_t)_{t\in\mathcal{C}}$ is regular.)

Proof. Enumerate all simple induction proofs as $\psi_0, \psi_1, \ldots$. Define $\tilde{\pi}_n$ as the first ψ_j which proves the current simple induction problem with $j \leq n$, or $\emptyset$ if none exists. By Theorem 9 it is possible that $L((\tilde{\pi}_n)^*_t) \not\supseteq L_t$, we will fix this as in Theorem 4. Let G_n be as in Theorem 8; define π_n as a simple induction proof with $G(\pi_n) = G_n \cup G(\tilde{\pi}_n)$ if $\tilde{\pi}(n) \neq \emptyset$, and $\emptyset$ otherwise. If $(L_t)_{t\in\mathcal{C}}$ is regular, then $(\tilde{\pi}_n)_{n\geq 0}$ and $(G_n)_{n\geq 0}$ are the desired eventually constant sequences. □

Theorem 11 ([6]). *The set of induction grammars G such that Φ_G is solvable is computably enumerable but not decidable (even if $E = \emptyset$, and also if we consider the subset of induction grammars that are E-tautological).*

The simple induction proof π obtained by reconstruction is not unique. It can be different from the original proof in two important aspects: first, the induction grammar $G(\pi)$ can be different—e.g., it can contain extra productions. Second, the induction formula may be different—e.g., it could be any propositionally equivalent formula.

4 Implementation and case studies

Algorithmically, the reconstruction proceeds in two phases: first we find an induction grammar, and then we find a solution to the induced formula equation. We find the grammar using a simple refinement loop: we start with a family of Herbrand languages $(L_t)_{t\in I_0}$ and find a covering induction grammar G_0 such that $L(G_0, t) \supseteq L_t$ for all $t \in I_0$. Guided by Corollary 1, we know that if we want to complete G_0 to a simple induction proof, then G_0 must be E-tautological. Hence we check that $L(G_0, t)$ is indeed E-tautological using an automated theorem prover for randomly picked terms t. If this check succeeds for 10 random terms, then we have found a grammar and proceed to find a solution. We have two algorithms to solve the induced formula equation. One is based on applying forgetful inferences [5]; the other applies techniques from constrained Horn clause solvers and is based on interpolation [14]. This algorithm has been implemented in the open-source GAPT [7] system for proof transformations, version 2.14[1].

We consider 73 simple induction proofs concerning basic properties of operations on natural numbers and lists. Of these, 26 are proofs of problems in the TIP benchmark suite for automated inductive theorem provers (tons of inductive problems [2]); GAPT contains 92 formal proofs of TIP problems as example data to test induction-elimination, 26 of these happen to be simple induction proofs. The other 48 proofs are lemmas from a formalization of the fundamental theorem of arithmetic in GAPT, again we took all the simple induction proofs.

[1] available at https://logic.at/gapt

For each proof, we performed two experiments. First, we used induction-elimination to compute the instance proofs and perform the reconstruction. As the second experiment, we then used an automated theorem prover[2] to generate the instance proofs. As we see in the following summary, the proofs obtained by induction-elimination are clearly different from the proofs produced by the automated theorem prover: the algorithm always finds a grammar for induction-elimination, but only for about two thirds of the ATP proofs. There is a smaller difference in the second solution-finding phase; interestingly the set of problems where we cannot find a solution is a strict subset of the corresponding problems from induction-elimination.

	#problems	#grammars	#solutions
ind.-elim.	73	73	61
ATP	60	38	29

In many cases the computed grammar does not match the grammar of the original simple induction proof exactly. One relatively common situation occurring in 10 reconstructed proofs is that we can replace a quantified induction formula by a quantifier-free one. That is, the original proof had e.g. $\forall y\,(x-(x+y)=0)$ as the induction formula[3], and we replace it by the quantifier-free formula $x-(x+z)=0$ where z is an eigenvariable of the end-sequent. On the level of the induction grammar, this corresponds to the elimination of the productions $\gamma \to \gamma \mid z$ (which are not very useful since the nonterminal γ will always expand to z).

In our running example[4], the computed grammar differs in a more interesting way. Here the algorithm again finds a grammar with fewer productions (compared to Example 6): $P = \{\tau \to f_2(\gamma,\nu), \gamma \to \gamma \mid \nu \mid \alpha\}$. We do not know if the corresponding FE has a solution or not (modulo $E = \{d(0)=0, d(s(x)) = s(s(d(x))), x+0=x\}$):

$$\forall\gamma\, X(\alpha,0,\gamma) \qquad \wedge \qquad (X(\alpha,\alpha,\alpha) \to d(\alpha) = \alpha+\alpha) \qquad \wedge$$
$$\forall\nu\forall\gamma\, (X(\alpha,\nu,\gamma) \wedge X(\alpha,\nu,\nu) \wedge X(\alpha,\nu,\alpha) \wedge \gamma + s(\nu) = s(\gamma+\nu) \to X(\alpha,s(\nu),\gamma))$$

If we look at the proofs produced by the ATP, then we get yet another grammar inducing the following formula equation; we do not know whether this FE is solvable either.

$$X(\alpha,0) \wedge (X(\alpha,\alpha) \to d(\alpha) = \alpha+\alpha) \wedge \forall\nu (X(\alpha,\nu) \wedge \alpha+s(\nu) = s(\alpha+\nu) \to X(\alpha,s(\nu)))$$

It is instructive to compare the instance proofs from induction-elimination of Example 4 with the proofs produced by the ATP. The straightforward way for an ATP to prove the instance problems is to treat the assumptions (f_1)–(f_4) as a term rewriting system and to rewrite the conjectured equation $d(s^n(0)) = s^{2n}(0)$ into normal form:

[2] We used the superposition prover built into GAPT, called Escargot. Other superposition provers such as Vampire, E, SPASS, etc., produce similar results.

[3] The binary operation $-$ denotes *truncating* subtraction, i.e. $x - y = \max\{x-y, 0\}$.

[4] `prod/prop_01` from the TIP library, slightly adjusted to conserve space

$$d(s^n(0)) \overset{(f_4)}{\twoheadrightarrow} s^{2n}(d(0)) \overset{(f_3)}{\twoheadrightarrow} s^{2n}(0)$$
$$s^n(0) + s^n(0) \overset{(f_2)}{\twoheadrightarrow} s^n(s^n(0) + 0) \overset{(f_1)}{\twoheadrightarrow} s^{2n}(0)$$

Looking closely, this proof uses the instances $f_2(s^n(0), s^i(0))$ for $0 \leq i < n$. This set of instances corresponds to the formula $\alpha{+}s(\nu) = s(\alpha{+}\nu)$ in the formula equation. Note also that there are no "repeating" equations in this sequence that could give rise to an induction formula. By contrast, the proof obtained via induction-elimination contains instances for a much more roundabout rewriting of the right-hand side, iterating the following snippet:

$$\begin{aligned} s^{i+1}(0) + s^{i+1}(0) \overset{(f_2)}{\twoheadrightarrow} s^{i+1}(s^{i+1}(0) + 0) &\overset{(f_1)}{\twoheadrightarrow} s^{2i+2}(0) \\ &\overset{(f_1)}{\twoheadleftarrow} s^{i+2}(s^i(0) + 0) \overset{(f_2)}{\twoheadleftarrow} s^2(s^i(0) + s^i(0)) \end{aligned}$$

There are 12 problems where the algorithm cannot find a solution for the grammar. Even in the special case under consideration here, it is surprisingly difficult to solve formula equations, that is, to determine whether a given formula equation has a solution or not. Appendix B lists some of these formula equations. We were unable to solve them by hand either.

5 Conclusion

Analytic proofs produced by induction-elimination carry a wealth of structure and regularity on the term level. We have shown how to exploit this regularity to practically reconstruct proofs with induction. This success is strong empirical evidence for our claim that information is shifted from the non-analytic induction invariant in the formulas, and retained as recognizable regularity in the terms.

We can observe a related phenomenon in mathematical logic with a more direct reflection of formulas as term-level structures: for example the set theory ZFC is not finitely axiomatizable due to the axiom scheme of replacement, which is parameterized by a formula. By encoding this formula parameter as a term we get the finitely axiomatized conservative extension NBG [17]. These phenomena differ from our approach as the induction formula is explicitly encoded as a term, whereas we investigate the incidental regularity in the quantifier instance terms, which implicitly indicates the provenance of induction-elimination.

Proofs found by common automated theorem provers do not typically possess this particular kind of regularity, as we have seen in Section 4—these proofs are not even grammatically regular. However if we had automated theorem provers which reliably produced regular families of proofs, then we could use our approach more effectively as a method for automated inductive theorem proving.

Finally, we would like to find more direct characterizations of regularity that give insight into the features of languages causing irregularity, along the lines of Myhill-Nerode [15,16] for regular word languages and regular tree languages [3]. Developing similar characterizations for regularity in TRATGs and induction grammars would help illuminate the phenomenon of regularity in inductive proofs.

References

1. Buss, S.R.: On Herbrand's Theorem. In: Logic and Computational Complexity, Lecture Notes in Computer Science, vol. 960, pp. 195–209. Springer (1995)
2. 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)
3. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2007)
4. 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)
5. Eberhard, S., Hetzl, S.: Inductive theorem proving based on tree grammars. Annals of Pure and Applied Logic **166**(6), 665–700 (2015)
6. Eberhard, S., Hetzl, S., Weller, D.: Boolean unification with predicates. Journal of Logic and Computation **27**(1), 109–128 (2017)
7. 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)
8. Gentzen, G.: Untersuchungen über das logische Schließen I. Mathematische Zeitschrift **39**(1), 176–210 (1935)
9. Gentzen, G.: Die Widerspruchsfreiheit der reinen Zahlentheorie. Mathematische Annalen **112**, 493–565 (1936)
10. Gentzen, G.: Neue Fassung des Widerspruchsfreiheitsbeweises für die reine Zahlentheorie. Forschungen zur Logik und zur Grundlegung der exakten Wissenschaften **4**, 19–44 (1938)
11. Herbrand, J.: Recherches sur la théorie de la démonstration. Ph.D. thesis, Université de Paris (1930)
12. Hetzl, S.: Applying tree languages in proof theory. In: Dediu, A.H., Martín-Vide, C. (eds.) Language and Automata Theory and Applications (LATA). Lecture Notes in Computer Science, vol. 7183, pp. 301–312. Springer (2012)
13. Hetzl, S., Leitsch, A., Reis, G., Weller, D.: Algorithmic introduction of quantified cuts. Theoretical Computer Science **549**, 1–16 (2014)
14. McMillan, K.L., Rybalchenko, A.: Solving constrained horn clauses using interpolation. Tech. Rep. MSR-TR-2013-6, Microsoft Research (2013)
15. Myhill, J.: Finite automata and the representation of events. Tech. Rep. WADD TR-57-624, Wright Patterson AFB, Ohio (1957)
16. Nerode, A.: Linear automaton transformations. Proceedings of the American Mathematical Society **9**(4), 541–544 (1958)
17. Neumann, J.v.: Eine Axiomatisierung der Mengenlehre. Journal für die reine und angewandte Mathematik **154**, 219–240 (1925)
18. Smullyan, R.M.: First-order logic (1968)
19. Tait, W.W.: Normal derivability in classical logic. In: The syntax and semantics of infinitary languages, vol. 72, pp. 204–236. Springer (1968)
20. Wernhard, C.: The boolean solution problem from the perspective of predicate logic. In: Dixon, C., Finger, M. (eds.) Frontiers of Combining Systems - 11th International Symposium, FroCoS 2017, Brasília, Brazil, September 27-29, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10483, pp. 333–350. Springer (2017)

A Instance languages

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 [4,13]), and define the language of the induction grammar as the language of the VTRATG obtained via instantiation.

Definition 13. *A VTRATG $G = (\tau, N, P)$ is a triple consisting of:*

- *the start nonterminal τ,*
- *a finite sequence $N = (\tau, \overline{\alpha_1}, \dots, \overline{\alpha_n})$ 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 $\overline{\alpha_{i+1}}, \dots, \overline{\alpha_n}$.*

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

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 15. *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 10. Let us instantiate the induction grammar in Example 6 with the parameter $s(s(0))$. The instance grammar $I(G, s(s(0)))$ has the nonterminals τ, γ_0, $\gamma_{s(0)}$, and $\gamma_{s(s(0))}$, and contains all productions on the right-hand side (we use the abbreviation $p \rightsquigarrow p'_1 \mid p'_2$ for $p \rightsquigarrow p'_1 \wedge p \rightsquigarrow p'_2$).

$$\begin{aligned}
\tau \to f_2(s(\gamma), \nu) &\quad\rightsquigarrow\quad \tau \to f_2(s(\gamma_{s(0)}), 0) \;\mid\; \tau \to f_2(s(\gamma_{s(s(0))}), s(0)) \\
\tau \to f_2(\gamma, \nu) &\quad\rightsquigarrow\quad \tau \to f_2(\gamma_{s(0)}, 0) \;\mid\; \tau \to f_2(\gamma_{s(s(0))}, s(0)) \\
\tau \to f_2(\nu, \nu) &\quad\rightsquigarrow\quad \tau \to f_2(0, 0) \;\mid\; \tau \to f_2(s(0), s(0)) \\
\gamma \to \gamma &\quad\rightsquigarrow\quad \gamma_0 \to \gamma_{s(0)} \;\mid\; \gamma_{s(0)} \to \gamma_{s(s(0))} \\
\gamma \to \nu &\quad\rightsquigarrow\quad \gamma_0 \to 0 \;\mid\; \gamma_{s(0)} \to s(0) \\
\gamma \to 0 &\quad\rightsquigarrow\quad \gamma_0 \to 0 \;\mid\; \gamma_{s(0)} \to 0 \;\mid\; \gamma_{s(s(0))} \to 0
\end{aligned}$$

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

Definition 16. *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 $L(G, t) = L(I(G, t))$.*

B Formula equations of unknown status

The formula equations in this appendix were generated in the case studies in Section 4, but could not be automatically solved by our implementation. It is surprisingly hard (and we did not manage) to manually find a solution for these FEs, or show that there is no solution. (All FEs have $E = \emptyset$.)

B.1 `isaplanner/prop_03`

This proof shows that the number of occurrences of an element in a list increases if we append something to the list. Lists are an inductive data type with the constructors *nil* and *cons*; e.g. $cons(a, cons(b, nil))$ is a two-element list. The function $c(n, l)$ counts how often n occurs in the list l; concatenation of lists is denoted by $a(l_1, l_2)$, $e(m, n)$ is a predicate expressing the equality of elements m and n.

$$\begin{aligned}
&\forall\gamma_0\,\forall\gamma_1\,(le(Z, c(n, ys)) \land c(n, nil) = Z \land\\
&\qquad (\neg e(n, \gamma_1) \to c(n, cons(\gamma_1, \gamma_0)) = c(n, \gamma_0)) \land\\
&\qquad (e(n, \gamma_1) \to c(n, cons(\gamma_1, \gamma_0)) = S(c(n, \gamma_0))) \land\\
&\qquad a(nil, ys) = ys \to X(\alpha, nil, \gamma_0, \gamma_1)) \land\\
&\forall\nu\,\forall\nu_0\,\forall\gamma_0\,\forall\gamma_1\,(X(\alpha, \nu_0, a(\nu_0, ys), \nu) \land X(\alpha, \nu_0, \nu_0, \nu) \land\\
&\qquad le(Z, c(n, ys)) \land c(n, nil) = Z \land\\
&\qquad (\neg e(n, \gamma_1) \to c(n, cons(\gamma_1, \gamma_0)) = c(n, \gamma_0)) \land\\
&\qquad (e(n, \gamma_1) \to c(n, cons(\gamma_1, \gamma_0)) = S(c(n, \gamma_0))) \land\\
&\qquad (le(S(c(n, \nu_0)), S(c(n, a(\nu_0, ys)))) \leftrightarrow le(c(n, \nu_0), c(n, a(\nu_0, ys)))) \land\\
&\qquad a(nil, ys) = ys \land a(cons(\nu, \nu_0), ys) = cons(\nu, a(\nu_0, ys)) \to\\
&\qquad X(\alpha, cons(\nu, \nu_0), \gamma_0, \gamma_1)) \land\\
&\forall\gamma_0\,\forall\gamma_1\,(X(\alpha, \alpha, \gamma_0, \gamma_1) \land le(Z, c(n, ys)) \land c(n, nil) = Z \land\\
&\qquad a(nil, ys) = ys \to le(c(n, \alpha), c(n, a(\alpha, ys))))
\end{aligned}$$

B.2 `prod/prop_13`

This problem states a property of the halving function $h(x) = \lfloor \frac{x}{2} \rfloor$, namely $h(x + x) = x$.

$$\begin{aligned}
&\forall\gamma\,(Z + \gamma = \gamma \land h(Z) = Z \to X(\alpha, Z, \gamma)) \land\\
&\forall\nu\,\forall\gamma\,(X(\alpha, \nu, \alpha) \land X(\alpha, \nu, \gamma) \land X(\alpha, \nu, \nu) \land Z + \gamma = \gamma \land h(Z) = Z \land\\
&\qquad h(S^2(\nu + \nu)) = S(h(\nu + \nu)) \land S(\nu) + \gamma = S(\nu + \gamma) \to X(\alpha, S(\nu), \gamma)) \land\\
&(X(\alpha, \alpha, \alpha) \land h(Z) = Z \to h(\alpha + \alpha) = \alpha)
\end{aligned}$$

B.3 `subpl`

Here we consider a property of the predecessor function p on natural numbers, $p(x) - y = p(x - y)$ ($-$ is the truncating subtraction).

$$\begin{gathered}
\forall\gamma\,(\gamma - 0 = \gamma \rightarrow X(\alpha, 0, \gamma)) \land \\
\forall\nu\,\forall\gamma\,(X(\alpha, \nu, p^2(\gamma)) \land X(\alpha, \nu, x) \land X(\alpha, \nu, p(x)) \land \gamma - 0 = \gamma \land \\
\gamma - s(\nu) = p(\gamma) - \nu \rightarrow X(\alpha, s(\nu), \gamma)) \land \\
(X(\alpha, \alpha, x) \land X(\alpha, \alpha, p(x)) \rightarrow p(x) - \alpha = p(x - \alpha))
\end{gathered}$$

B.4 `subps`

This formula equation comes from a corollary of `subpl`, it is easy to see that every solution for the FE in `subpl` also solves the following FE:

$$\begin{gathered}
\forall\gamma\,(\gamma - 0 = \gamma \land p(s(x)) = x \rightarrow X(\alpha, 0, \gamma)) \land \\
\forall\nu\,\forall\gamma\,(X(\alpha, \nu, p^2(\gamma)) \land X(\alpha, \nu, s(x)) \land X(\alpha, \nu, p(s(x))) \land \gamma - 0 = \gamma \land \\
p(s(x)) = x \land \gamma - s(\nu) = p(\gamma) - \nu \rightarrow X(\alpha, s(\nu), \gamma)) \land \\
(X(\alpha, \alpha, s(x)) \land X(\alpha, \alpha, p(s(x))) \land p(s(x)) = x \rightarrow p(s(x) - \alpha) = x - \alpha)
\end{gathered}$$