Annals of Pure and Applied Logic 166 (2015) 665–700

Contents lists available at ScienceDirect

Annals of Pure and Applied Logic

www.elsevier.com/locate/apal

Inductive theorem proving based on tree grammars

Sebastian Eberhard [1], Stefan Hetzl [*,1]

Institute of Discrete Mathematics and Geometry, Vienna University of Technology, Wiedner Hauptstraße 8-10, AT-1040 Vienna, Austria

ARTICLE INFO

Article history:
Received 14 March 2014
Received in revised form 25 November 2014
Accepted 21 January 2015
Available online 2 March 2015

MSC:
03F03
03F07
03F30
68Q42

Keywords:
Proof theory
Herbrand's theorem
Inductive theorem proving
Automated deduction

ABSTRACT

Induction plays a key role in reasoning in many areas of mathematics and computer science. A central problem in the automation of proof by induction is the non-analytic nature of induction invariants. In this paper we present an algorithm for proving universal statements by induction that separates this problem into two phases. The first phase consists of a structural analysis of witness terms of instances of the universal statement. The result of such an analysis is a tree grammar which induces a quantifier-free unification problem which is solved in the second phase. Each solution to this problem is an induction invariant. The arguments and techniques used in this paper heavily exploit a correspondence between tree grammars and proofs already applied successfully to the generation of non-analytic cuts in the setting of pure first-order logic.

1. Introduction

Induction is a reasoning principle of fundamental importance in many areas of mathematics and computer science. It is indispensable already for the definition of basic data structures such as lists, trees, programs, etc. and for reasoning about these structures. Induction has also a prominent place in proof theory since, at least, Gentzen's proof of the consistency of Peano Arithmetic [16].

Due to the ubiquity of induction in computer science there is a large interest in devising algorithms which find proofs by induction automatically. This field, inductive theorem proving, has produced a remarkable range of systems and techniques such as ACL2 [29,28], rippling [8,9], or implicit induction (e.g. the Spike theorem prover [1]). More recent techniques include cyclic proofs [7] (and the cyclist prover [6]), the Zeno prover [31] and approaches based on forward reasoning like inductive theory formation (IsaCoSy [27] and HipSpec [30]).

* Corresponding author.
E-mail addresses: eberhard@iam.unibe.ch (S. Eberhard), stefan.hetzl@tuwien.ac.at (S. Hetzl).
[1] Research supported by the WWTF-project VRG12-04.

http://dx.doi.org/10.1016/j.apal.2015.01.002
0168-0072/

A fundamental problem of inductive theorem proving is the non-analytic nature of invariants. Very often, already for quite simple practical cases, the induction invariant needed for proving a theorem is not that theorem itself, nor a subformula nor an instance of it. Instead, what is needed is a generalisation of the theorem. In which direction or what way the theorem needs to be generalised however strongly depends on the theorem and it is hence difficult to develop uniform methods capable of introducing the right generalisation.

In this paper we present a new approach to inductive theorem proving which attacks this problem by separating it into two phases. When searching for a proof of a universal statement we first compute proofs of a few small instances. In a first phase we then decompose the witness terms of these proofs in a way that is induced by an induction proof. This first phase can be described and solved without any reference to logic as a problem of finite sets of terms and structures describing such sets: tree grammars. A tree grammar then determines the instantiation-structure of an induction-proof and induces a quantifier-free second-order unification problem. Each solution to this unification problem is an induction invariant. In the second, independent, phase we compute a solution to this unification problem. Such solutions are typically not analytic.

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 [3] on the constructive omega-rule. Similar ideas also exist in other contexts, for example bounded model checking [4]. What our work 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. This leads to new methods for computing invariants which succeed in typical test cases.

This overall two-phased strategy has already proved very effective in the related context of generating non-analytic lemmas for abbreviating and structuring proofs in pure first-order logic, i.e. without induction. The basic techniques have been introduced in [19], the cut-introduction method for a single Π_1-cut has first been defined in [22] and extended to the introduction of an arbitrary number of Π_1-cuts in [21]. The paper [20] describes how to extend the method to proofs modulo equality and cuts using blocks of quantifiers. It also contains a comprehensive empirical evaluation of an implementation.

Let us give an outline of the paper.

In Section 2, we present the main results of [21]. Also, many notations of [21] will be introduced since they are reused in this paper.

In Section 3, we describe the form of the induction proof our algorithm tries to find. We call proofs of this form simple induction proofs. We will allow just one application of induction over ν for a formula of the form $\forall y F[\nu, y]$ with F quantifier-free followed immediately by a cut. While this is quite restricted from a proof-theoretic point of view, it covers a large class of practically relevant cases including many classical examples of inductive theorem proving.

In Section 3, we also make connections between various notions of provability which are of relevance to this paper: provability by simple induction proofs, by induction-free proofs using iterated cuts with a uniform cut formula, and by cut- and induction-free proofs. In addition a close connection between tree grammars and proofs is established and explained.

In Section 4, our algorithm IndProof is sketched. As input it takes a finite set of Herbrand sequents of instances of a universal statement and, if it terminates, it returns a simple induction proof of this universal statement. The algorithm is composed of two steps, FindGram and FindFml. FindGram searches for regularities between the Herbrand sequents which are the input of IndProof. For this aim Herbrand sequents are interpreted as tree languages, and then techniques of language compression by grammars are used. Relying on the found regularities, FindFml generates induction invariants in a systematic way. In Sections 5 and 6, IndProof is defined precisely.

Finally, in Section 7, IndProof is applied to test cases. It succeeds to find inductive proofs for two classical examples: the associativity of addition and the equivalence of the two definitions of the factorial by head- and tail-recursion. Remarkably, in both cases very natural analytic proofs of only two instances of the universal statements are sufficient as input of IndProof.

2. Previous results

In order to present the results of [21] some theoretical background has to be given.

Notation 2.1. For variables $x_1, \dots, x_n$, we write $t[x_1, \dots, x_n]$ to denote a term t possibly containing occurrences of $x_1, \dots, x_n$. For terms $u_1, \dots, u_n$, we write $t[x_1 \backslash u_1, \dots, x_n \backslash u_n]$ for the term obtained from t by substituting x_i by u_i for $1 \leq i \leq n$. For simplicity, we normally just write $t[u_1, \dots, u_n]$ instead. We use standard vector notation for variables and terms, e.g. as in $t[\overline{x}]$ or $t[\overline{u}]$. If the length of the vector is irrelevant or clear from the context it is not mentioned explicitly. Analogous notations are used for formulas. We typically use Roman letters $x, y, z, \dots$ for bound variables and Greek letters $\alpha, \beta, \gamma, \dots$ for free variables.

The results of this paper do not depend on the syntactic details of the employed calculus. Indeed, most results will be formulated based on formulas or sequents being tautologies or quasi-tautologies[2] without mention of their actual proofs. The one exception to this negligence of the shape of proofs concerns the use of cut and induction about which we will be very explicit. Nevertheless, for the sake of precision, let us fix a calculus to work in.

Notation 2.2. A sequent is a pair of multisets of formulas, written as $\Gamma \Rightarrow \Delta$. As our logical system, we use a sequent calculus consisting of the following rules:

- Axioms

$$A \Rightarrow A \qquad \Rightarrow t = t$$

  for any atom A and any term t.
- Logical Rules

$$\frac{A, \Gamma \Rightarrow \Delta}{A \wedge B, \Gamma \Rightarrow \Delta} \wedge_{\mathrm{l1}} \qquad \frac{B, \Gamma \Rightarrow \Delta}{A \wedge B, \Gamma \Rightarrow \Delta} \wedge_{\mathrm{l2}} \qquad \frac{\Gamma \Rightarrow \Delta, A \qquad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \wedge B} \wedge_{\mathrm{r}}$$

$$\frac{\Gamma \Rightarrow \Delta, A}{\Gamma \Rightarrow \Delta, A \vee B} \vee_{\mathrm{r1}} \qquad \frac{\Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \vee B} \vee_{\mathrm{r2}} \qquad \frac{A, \Gamma \Rightarrow \Delta \qquad B, \Gamma \Rightarrow \Delta}{A \vee B, \Gamma \Rightarrow \Delta} \vee_{\mathrm{l}}$$

$$\frac{A, \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \to B} \to_{\mathrm{r}} \qquad \frac{\Gamma \Rightarrow \Delta, A \qquad B, \Gamma \Rightarrow \Delta}{A \to B, \Gamma \Rightarrow \Delta} \to_{\mathrm{l}}$$

$$\frac{\Gamma[t], s = t \Rightarrow \Delta[t]}{\Gamma[s], s = t \Rightarrow \Delta[s]} =_1 \qquad \frac{\Gamma[s], s = t \Rightarrow \Delta[s]}{\Gamma[t], s = t \Rightarrow \Delta[t]} =_2$$

$$\frac{A[t], \Gamma \Rightarrow \Delta}{\forall x A[x], \Gamma \Rightarrow \Delta} \forall_{\mathrm{l}} \qquad \frac{\Gamma \Rightarrow \Delta, A[\alpha]}{\Gamma \Rightarrow \Delta, \forall x A[x]} \forall_{\mathrm{r}} \qquad \frac{\Gamma \Rightarrow \Delta, A[t]}{\Gamma \Rightarrow \Delta, \exists x A[x]} \exists_{\mathrm{r}} \qquad \frac{A[\alpha], \Gamma \Rightarrow \Delta}{\exists x A[x], \Gamma \Rightarrow \Delta} \exists_{\mathrm{l}}$$

  where the usual side conditions apply: α is a variable which does not appear in the conclusion of the inference and all substitutions are assumed to be capture-avoiding.

[2] A quasi-tautology is a quantifier-free formula or sequent which is valid in predicate logic with equality.

- Structural Rules

$$\frac{\Gamma \Rightarrow \Delta}{A, \Gamma \Rightarrow \Delta}\ \mathrm{w_l} \qquad \frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, A}\ \mathrm{w_r} \qquad \frac{A, A, \Gamma \Rightarrow \Delta}{A, \Gamma \Rightarrow \Delta}\ \mathrm{c_l} \qquad \frac{\Gamma \Rightarrow \Delta, A, A}{\Gamma \Rightarrow \Delta, A}\ \mathrm{c_r}$$

$$\frac{\Gamma \Rightarrow \Delta, A \qquad A, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}\ \mathrm{cut}$$

We denote the resulting system by **LK**. When we speak of proofs without induction or just of proofs, we always refer to **LK** proofs. Proofs with induction additionally are allowed to contain the following rule for $F[x, y]$ a quantifier-free formula.

$$\frac{\Gamma \Rightarrow \forall y F[0, y], \Delta \quad \Gamma, \forall y F[\nu, y] \Rightarrow \forall y F[\mathsf{s}\nu, y], \Delta}{\Gamma \Rightarrow \forall y F[\alpha, y], \Delta}\ \mathrm{ind}$$

A sequent $\Gamma \Rightarrow \Delta$ of the form

$$\forall \overline{x_1} F_1, \ldots, \forall \overline{x_n} F_n \Rightarrow \exists \overline{y_1} G_1, \ldots, \exists \overline{y_m} G_m$$

with $F_1, \ldots, F_n, G_1, \ldots, G_m$ quantifier-free is called a Σ_1 sequent. Weak and strong quantifiers are defined as usual. For a formula of the form $Q\overline{x}F[\overline{x}]$ with $Q = \forall, \exists$ and terms $\overline{t}$, the formula $F[\overline{t}]$ is called an instance of $Q\overline{x}F[\overline{x}]$.

Definition 2.3 *(Herbrand sequent).* A Herbrand sequent of the Σ_1 sequent $\Gamma \Rightarrow \Delta$ is a quasi-tautology of the form $\Gamma' \Rightarrow \Delta'$ where Γ' consists of instances of Γ and Δ' of instances of Δ, i.e. for every $A' \in \Gamma'$ there is $A \in \Gamma$ s.t. A' is an instance of A and analogously for Δ' and Δ.

In this paper we will often rely on the following version of Herbrand's theorem [18].

Theorem 2.4. *A Σ_1 sequent $\Gamma \Rightarrow \Delta$ is valid in first-order logic with equality iff it has a Herbrand sequent.*

Proof Sketch. Let $\Gamma \Rightarrow \Delta$ be a valid sequent and π be a cut-free proof of $\Gamma \Rightarrow \Delta$. By carrying out rule permutations one can transform π to a proof π' in mid-sequent form, i.e. π' contains a quantifier-free sequent $\Gamma' \Rightarrow \Delta'$ s.t. there are only propositional, structural and equality inferences above $\Gamma' \Rightarrow \Delta'$ and only structural and quantifier inferences below $\Gamma' \Rightarrow \Delta'$. Hence $\Gamma' \Rightarrow \Delta'$ is a Herbrand sequent.

For the other direction, let $\Gamma' \Rightarrow \Delta'$ be a Herbrand sequent. Being a quasi-tautology there is a proof π_0 of $\Gamma' \Rightarrow \Delta'$ (which contains only propositional, structural and equality inferences). As $\Gamma' \Rightarrow \Delta'$ consists of instances of $\Gamma \Rightarrow \Delta$ there is a proof π_1 of $\Gamma \Rightarrow \Delta$ from $\Gamma' \Rightarrow \Delta'$ (which consists only of structural and quantifier inferences). Concatenating π_0 and π_1 gives a proof of $\Gamma \Rightarrow \Delta$. □

Note that the above proof induces straightforward algorithms for reading off a Herbrand sequent from a given cut-free proof and, vice versa, for computing a cut-free proof from a given Herbrand sequent, see e.g. [23] for more information. For the techniques used in this paper and also in [19,21] it is crucial to view Herbrand sequents in addition as sets of terms.

Notation 2.5. For technical reasons, we work with function symbols possibly containing formulas as subindices. For a function symbol r, and distinct formulas C, D the function symbols $\mathsf{r}, \mathsf{r}_C, \mathsf{r}_D$ are assumed to be pairwise different.

In this paper we are only interested in Herbrand sequents of Σ_1 sequents $\Gamma \Rightarrow \Delta$ with Δ containing a single quantifier-free formula. Accordingly, the set of terms T_H of a Herbrand sequent H, which will be defined in the following, only contains terms corresponding to instances of formulas in Γ.

Definition 2.6 *(Set of terms of Herbrand sequent).* Let H be a Herbrand sequent of the Σ_1 sequent $\forall \overline{x_1} F_1[\overline{x_1}], \ldots, \forall \overline{x_n} F_n[\overline{x_n}] \Rightarrow \Delta$. Then the set of terms T_H of H is defined as follows:

- Assume that the instance $F_i[\bar{t}]$ of the formula $\forall \overline{x_i} F_i[\overline{x_i}]$ occurs in H for an $i \in \{1, \ldots, n\}$. Then, the term $\mathsf{r}_{\forall \overline{x_i} F_i[\overline{x_i}]}(\bar{t})$ occurs in T_H. If $\overline{x_i}$ is empty, $\mathsf{r}_{\forall \overline{x_i} F_i[\overline{x_i}]}$ has arity zero.
- T_H does not contain any other elements.

In the following, we often identify the set of terms T_H of a Herbrand sequent with that Herbrand sequent H.

We are now going to define the tree grammars needed in this paper. For a term signature Σ let $\mathcal{T}(\Sigma)$ denote the set of all terms built from symbols in Σ.

Definition 2.7 *(Regular tree grammar).* A regular tree grammar is a tuple $\langle \tau, N, \Sigma, P \rangle$ where N is a finite set of non-terminal symbols with arity 0 s.t. $\tau \in N$. Furthermore, Σ is a finite set of terminal symbols of arbitrary arities, i.e. a term signature, satisfying $N \cap \Sigma = \emptyset$. The productions P are a finite set of rules of the form $\gamma \to t$ where $\gamma \in N$ and $t \in \mathcal{T}(\Sigma \cup N)$.

The non-terminal τ is called the axiom of $G = \langle \tau, N, \Sigma, P \rangle$. A derivation of a term t in G is a list $t_1, \ldots, t_n$ of terms s.t. $t_1 = \tau$, $t_n = t$ and for every $i \in \{1, \ldots, n-1\}$ there is a rule $\gamma \to s \in P$ s.t. t_{i+1} is obtained from t_i by replacing one occurrence of γ by s. The language of G is defined as $\mathcal{L}(G) = \{t \in \mathcal{T}(\Sigma) \mid t \text{ derivable in } G\}$. For more background on tree languages, the interested reader is referred to [14,10].

Definition 2.8 *(Rigid acyclic tree grammar).* Rigid acyclic tree grammars restrict regular tree grammars by imposing the following additional conditions.

- In each derivation, for each non-terminal γ only a single rule of the form $\gamma \to t$ is allowed to occur.
- There is a total order $<$ on the non-terminals N such that for each rule $\gamma \to t$ in P, only non-terminals larger than γ occur in t.

The notion of rigid tree language has been introduced by Jacquemard, Klay, and Vacher in [25,26]. In the following, we often silently assume that grammars are rigid acyclic tree grammars.

Let us give a detailed overview of [19,21] and explain the main notions. First, let us note that the logical setting of [19,21] is predicate logic without equality. Nevertheless, all results can be extended to predicate logic with equality in the style of [20]. This is why we present an extension of [19,21] to predicate logic with equality in this section. In addition, we will adapt some of the notations and definitions used in [19,21] to our setting.

In [19,21], **LK** proofs are analysed which contain at most cuts with cut-formulas of the form $\forall x C[x]$ with C quantifier-free and whose conclusion is a Σ_1 sequent Θ. We call such proofs simple proofs. We call cuts of this form Π_1 cuts, and their cut formulas Π_1 cut formulas. Gentzen's cut-elimination procedure [15] applied to a simple proof of a Σ_1 sequent Θ yields a cut-free proof of Θ, giving rise to a Herbrand sequent of Θ as in the proof of Theorem 2.4. In [19,21], the relation between the original proof π of Θ with cuts and its cut-free version is analysed using grammars. In [19] Definition 19, a grammar $\mathcal{G}(\pi)$ is defined for each simple proof π. Let us sketch a version of this definition adapted to our setting.

For simplicity, we assume that the conclusion of π has the form $\forall \bar{x} F[\bar{x}] \Rightarrow$ for F quantifier-free. The general case of arbitrary Σ_1 sequents is treated similarly. Rules of the form $\tau \to \mathsf{r}_{\forall \bar{x} F[\bar{x}]}(\bar{s})$ are added to $\mathcal{G}(\pi)$ for the instances $F[\bar{s}]$ of the weak quantifier block in $\forall \bar{x} F[\bar{x}]$. If the quantifier block $\forall \bar{x}$ is empty, the rule $\tau \to \mathsf{r}_F$ will be added where r_F has arity 0. In addition, for each eigenvariable u of a Π_1 cut formula $\forall x C[x]$, u is introduced as non-terminal of $\mathcal{G}(\pi)$. A rule $u \to t$ is added to $\mathcal{G}(\pi)$ for each instance $C[t]$ of

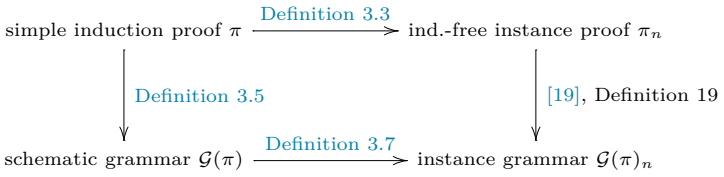

Diagram 1. Dependencies between grammars and proofs.

the occurrence of the cut formula $\forall x C[x]$ containing the weak universal quantifier. The following theorem proved in [19,21] demonstrates the relevance of grammars of simple proofs. It can be easily extended to arbitrary Σ_1 sequents as conclusions of simple proofs.

Theorem 2.9. *Let π be a simple proof of $\forall \bar{x} F[\bar{x}] \Rightarrow$ for F quantifier-free. Then $\mathcal{L}(\mathcal{G}(\pi))$ is a Herbrand-sequent of $\forall \bar{x} F[\bar{x}] \Rightarrow$.*

The previous theorem and the fact that each simple proof has a corresponding grammar allow us to find Herbrand sequents of $\forall \bar{x} F[\bar{x}] \Rightarrow$ directly from simple proofs of the same sequent. Therefore, calculating the language produced by a rigid acyclic tree grammar replaces cut-elimination. However, the main goal of [21] is to allow the opposite transition, i.e. the production of a proof of $\forall \bar{x} F[\bar{x}] \Rightarrow$ containing Π_1 cuts from a Herbrand sequent of $\forall \bar{x} F[\bar{x}] \Rightarrow$ or a cut-free proof of $\forall \bar{x} F[\bar{x}] \Rightarrow$, respectively.

Let us sketch the algorithm *CI* (cut introduction) executing this task, which is presented in [21]. The input of *CI* is a cut-free proof of a Σ_1 sequent Θ. Again, assume for simplicity that Θ has the form $\forall \bar{x} F[\bar{x}] \Rightarrow$.

- From the cut-free proof of Θ, we find a Herbrand sequent T of Θ.
- Considering T as a set of terms, we construct a rigid, acyclic tree grammar G with a minimal number of rules such that $T = \mathcal{L}(G)$.
- From G, which fixes the occurring instances of the cut-formulas, one can easily obtain cut-formulas yielding a simple proof of $\forall \bar{x} F[\bar{x}] \Rightarrow$ with grammar G.
- The cut-formulas are simplified with respect to their length. This yields a shorter simple proof of $\forall \bar{x} F[\bar{x}] \Rightarrow$. The grammar of the original simple proof is not changed by these simplifications since it only depends on the occurring instances of the cut-formulas.

The algorithm *CI* sketched above has been tested in large scale experiments in [20]. The results show that *CI* significantly compresses a high percentage of proofs from the TSTP-library (see [32]). This library collects typical proofs occurring in applications in computer science.

3. Induction proofs and grammars

We are interested in proofs containing induction in a restricted way which we call simple induction proofs. Most importantly, only Π_1 induction formulas are allowed. In addition we only allow a single induction on a formula which implies the conclusion of the proof. An adaptation of our techniques to more complex induction formulas depends on progress in the analysis of proofs with more complex cut formulas by grammars. This is a goal for future research. Nevertheless, already simple induction proofs are highly relevant for applications in computer science because they cover the prototypical situation of a conjecture which needs a step of generalisation in order to find a proof by induction.

In this section, we will give the theoretical foundations for our algorithm, and connect simple induction proofs with tree grammars. This is achieved by defining and explaining the commutative Diagram 1.

Definition 3.1 *(Simple induction proofs).* The following proof is a simple induction proof (in short: s.i.p.) if all of the following conditions hold.

$$\dfrac{\dfrac{\dfrac{\pi_b[\alpha,\beta]}{\Gamma_0[\alpha,\beta] \Rightarrow F[\alpha,0,\beta]}}{\forall\Gamma \Rightarrow \forall y F[\alpha,0,y]}\,* \quad \dfrac{\dfrac{\pi_s[\alpha,\nu,\gamma]}{\Gamma_1[\alpha,\nu,\gamma], \bigwedge_{1\le i\le n} F[\alpha,\nu,t_i[\alpha,\nu,\gamma]] \Rightarrow F[\alpha,\mathsf{s}\nu,\gamma]}}{\forall\Gamma, \forall y F[\alpha,\nu,y] \Rightarrow \forall y F[\alpha,\mathsf{s}\nu,y]}\,*}{\dfrac{\forall\Gamma \Rightarrow \forall y F[\alpha,\alpha,y]}{\forall\Gamma \Rightarrow B[\alpha]}\,\text{cut}}\,\text{ind} \quad \dfrac{\dfrac{\pi_c[\alpha]}{\Gamma_2[\alpha], \bigwedge_{1\le i\le m} F[\alpha,\alpha,u_i[\alpha]] \Rightarrow B[\alpha]}}{\forall\Gamma, \forall y F[\alpha,\alpha,y] \Rightarrow B[\alpha]}\,*$$

- $\alpha, \beta, \nu, \gamma$ are variables occurring only where indicated.
- We have $n, m \in \mathbb{N}$ with $n, m \geq 1$. t_i for $1 \leq i \leq n$ and u_i for $1 \leq i \leq m$ denote terms.
- F and B are quantifier-free formulas.
- Γ_i for $0 \leq i \leq 2$ only contains quantifier-free formulas.
- π_b, π_s, and π_c each are cut-free **LK** proofs of the sequent displayed below them.
- $\forall\Gamma$ only contains formulas of the form $\forall\bar{x}G[\bar{x}]$ for G quantifier-free.
- In each of the displayed proof branches, the inferences labelled with $*$ are a concatenation of several contractions, weakenings and universal quantifier introductions.

In practice, an s.i.p. will often be followed by a $\forall_r$-inference inferring $\forall x B[x]$. We refer to an s.i.p. given as above by the letter π. In the context of a particular s.i.p. π, we will also often use the notations $\pi_b, \pi_s, \pi_c, \Gamma_0, \Gamma_1, \Gamma_2, t_i, u_i, \ldots$ to refer to the respective parts of π. We use the notations π_b^*, π_s^* and π_c^* to refer to the proof π_b, π_s and π_c followed by the respective $*$-inferences.

For practical applications we also cover the case of a quantifier-free induction formula: instead of introducing a vacuously binding $\forall y$, it is technically more convenient to let $n = m = 0$. We will call that case *degenerate* and mention it explicitely whenever it requires special attention.

Example 3.2. In this example, we define an s.i.p. π of the equivalence of the definitions of the factorial by head- and tail-recursion.[3] The first definition of the factorial is the usual one, given by head-recursion for $n \in \mathbb{N}$:

$$f(0) := 1$$
$$f(n+1) := (n+1)f(n)$$

The second definition by tail-recursion uses an auxiliary function g given as follows for all $n, m \in \mathbb{N}$.

$$g(m,0) := m$$
$$g(m,n+1) := g\big(m \cdot (n+1), n\big)$$

Then the statement we would like to show by induction is $f(n) = g(1,n)$ for all $n \in \mathbb{N}$.

Our background theory $\forall\Gamma$ consists of the universal closures of the following formulas which axiomatise the two above definitions of the factorial function and some basic properties of multiplication:

[3] Tail-recursion plays an important role in functional programming as it allows to reduce stack usage to a constant amount. Therefore, proving the equivalence of a tail-recursive definition to a (usually more straightforward) head-recursive definition is a type of verification problem of high practical importance.

$$\begin{array}{ll} f(0) = 1 & (f0) \\ f(\mathsf{s}x) = \mathsf{s}x \cdot f(x) & \big(fST(x)\big) \\ g(x,0) = x & \big(g0(x)\big) \\ g(x,\mathsf{s}y) = g(x \cdot \mathsf{s}y, y) & \big(gST(x,y)\big) \\ x \cdot 1 = x & \big(1R(x)\big) \\ 1 \cdot x = x & \big(1L(x)\big) \\ (x \cdot y) \cdot z = x \cdot (y \cdot z) & \big(ASSO(x,y,z)\big) \end{array}$$

The right column above defines abbreviations of these formulas which we will use throughout the paper. The s.i.p. π is a proof of $\forall\Gamma \Rightarrow g(1,\alpha) = f(\alpha)$ and uses the induction formula $\forall y F(\nu, y) := \forall y\, g(y,\nu) = y \cdot f(\nu)$. Note that the induction formula is more general than the conclusion. The induction premises and the cut-premises of π are derived as follows:

$$\dfrac{\dfrac{\dfrac{}{f0, g0(\beta), 1R(\beta) \Rightarrow g(\beta,0) = \beta \cdot f(0)}\,\pi_b}{\forall\Gamma \Rightarrow \forall y F[0,y]}\,*}{\dfrac{fST(\nu), gST(\gamma,\nu), ASSO\big(\gamma, \mathsf{s}\nu, f(\nu)\big), g(\gamma \cdot \mathsf{s}\nu, \nu) = (\gamma \cdot \mathsf{s}\nu) \cdot f(\nu) \Rightarrow g(\gamma, \mathsf{s}\nu) = \gamma \cdot f(\mathsf{s}\nu)}{\forall\Gamma, \forall y F[\nu,y] \Rightarrow \forall y F[\mathsf{s}\nu, y]}\,\pi_s}\,*$$

$$\dfrac{\dfrac{}{1L\big(f(\alpha)\big), g(1,\alpha) = 1 \cdot f(\alpha) \Rightarrow g(1,\alpha) = f(\alpha)}\,\pi_c}{\forall\Gamma, \forall y F[\alpha,y] \Rightarrow g(1,\alpha) = f(\alpha)}\,*$$

Note that the actual proofs π_b, π_s, and π_c are irrelevant for our method and are therefore not specified.

The next lemma follows immediately from the fact that induction up to a numeral $\overline{n}$ can be unfolded by n iterated cuts.

Definition 3.3. Let π be a simple induction proof as in Definition 3.1. Then for each $n \in \mathbb{N}$ we define the n-th instance proof π_n as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi_b^*[\overline{n},\gamma_0]}{\forall\Gamma \Rightarrow \forall y F[\overline{n},0,y]} \quad \dfrac{\pi_s^*[\overline{n},0,\gamma_1]}{\forall\Gamma, \forall y F[\overline{n},0,y] \Rightarrow \forall y F[\overline{n},\mathsf{s}0,y]}}{\forall\Gamma \Rightarrow \forall y F[\overline{n},\mathsf{s}0,y]}\,\text{cut}}{\vdots}}{\forall\Gamma \Rightarrow \forall y F[\overline{n},\overline{n-1},y]} \quad \dfrac{\pi_s^*[\overline{n},\overline{n-1},\gamma_n]}{\forall\Gamma, \forall y F[\overline{n},\overline{n-1},y] \Rightarrow \forall y F[\overline{n},\overline{n},y]}}{\forall\Gamma \Rightarrow \forall y F[\overline{n},\overline{n},y]}\,\text{cut} \quad \dfrac{\pi_c^*[\overline{n}]}{\forall\Gamma, \forall y F[\overline{n},\overline{n},y] \Rightarrow B[\overline{n}]}}{\forall\Gamma \Rightarrow B[\overline{n}]}\,\text{cut}$$

We refer to the collection of instance proofs as $\pi_{n\geq 0}$.

The previous definition fixes an operation producing from a simple induction proof a simple proof with iterated cuts by eliminating induction. This yields the upper right arrow in Diagram 1. Because of the described elimination of induction, a simple induction proof π can be seen as a finite representation of infinitely many simple instance proofs $\pi_{n\geq 0}$. Schematic grammars which will be defined in the following have a similar role as simple induction proofs on the level of grammars: They come with variables whose suitable replacement yields the grammar $\mathcal{G}(\pi_n)$ of the instance proof π_n for each $n \in \mathbb{N}$.

Definition 3.4 *(Schematic grammar).* Let N be given as $\{\tau, \alpha, \beta, \nu, \gamma, \gamma_{end}\}$. Let Σ be a term signature disjoint from N. Then, the grammar $\langle \tau, N, \Sigma, P\rangle$ is a schematic grammar if each of its production rules is of one of the following forms:

- $\tau \to t[\alpha, \beta]$
- $\tau \to t[\alpha, \nu, \gamma]$
- $\gamma \to t[\alpha, \nu, \gamma]$
- $\gamma_{end} \to t[\alpha]$

The non-terminals α, β, ν will correspond to the respective eigenvariables of the same name in an s.i.p. (see Definition 3.1). The non-terminal τ does not occur on the right-hand side of any production rule – it will take the role of the axiom from which the instances of $\forall\Gamma$ that constitute a Herbrand sequent can be derived. In an s.i.p. the strong quantifier with eigenvariable γ has *two* corresponding weak quantifiers: the one in π_s and the one in π_c. The former will correspond to the non-terminal γ in a schematic grammar and the latter to the non-terminal γ_{end}. The choice of the non-terminals allowed on the right-hand side of the production rules is explained by the position of eigenvariables in π: in π_b the variables α and β occur, in π_s the variables α, ν, and γ occur and in π_c only α occurs. The next definition shows in detail how an s.i.p. induces a schematic grammar.

Definition 3.5 *(From simple induction proofs to schematic grammars).* Let π be an s.i.p. as in Definition 3.1, let $\forall\Gamma = \forall\overline{x_1}F_1[\overline{x_1}], \ldots, \forall\overline{x_\ell}F_\ell[\overline{x_\ell}]$ and let Σ be the signature of π. Furthermore, for all $i \in \{1, \ldots, \ell\}$ let $\mathsf{r}_{\forall\overline{x_i}F_i[\overline{x_i}]}$ be a function symbol whose arity is the length of $\overline{x_i}$ and let $\overline{s_{i,1}}, \ldots, \overline{s_{i,k_i}}$ be the instances of $\forall\overline{x_i}F_i[\overline{x_i}]$ in $\Gamma_0 \cup \Gamma_1 \cup \Gamma_2$. Then the schematic grammar of π is $\mathcal{G}(\pi) = \langle \tau, \{\tau, \alpha, \beta, \nu, \gamma, \gamma_{end}\}, \Sigma \cup \{\mathsf{r}_{\forall\overline{x_1}F_1[\overline{x_1}]}, \ldots, \mathsf{r}_{\forall\overline{x_\ell}F_\ell[\overline{x_\ell}]}\}, P\rangle$ where

$$\begin{aligned} P = &\left\{\tau \to \mathsf{r}_{\forall\overline{x_i}F_i[\overline{x_i}]}(\overline{s_{i,j}}) \mid 1 \le i \le \ell, 1 \le j \le k_i\right\} \\ &\cup \left\{\gamma \to t_i[\alpha, \nu, \gamma] \mid 1 \le i \le n\right\} \\ &\cup \left\{\gamma_{end} \to u_i[\alpha] \mid 1 \le i \le m\right\}. \end{aligned}$$

The previous definition yields the arrow from simple induction proofs to schematic grammars in Diagram 1.

Example 3.6. For π being the s.i.p. defined in Example 3.2 let us calculate $\mathcal{G}(\pi)$ in the following. We introduce constants

$$\mathsf{r}_{f0}, \mathsf{r}_{\forall xfST(x)}, \mathsf{r}_{\forall xg0(x)}, \mathsf{r}_{\forall x\forall ygST(x,y)}, \mathsf{r}_{\forall x1R(x)}, \mathsf{r}_{\forall x1L(x)}, \mathsf{r}_{\forall x\forall y\forall zASSO(x,y,z)}$$

of arity being equal to the number of universal quantifiers occurring in their index. Then, we introduce the following rules depending on the axiom instances occurring in the quasi-tautologies:

$$\begin{aligned} \tau \to\ & \mathsf{r}_{f0} \mid \mathsf{r}_{\forall xg0(x)}(\beta) \mid \mathsf{r}_{\forall x1R(x)}(\beta) \mid \\ & \mathsf{r}_{\forall xfST(x)}(\nu) \mid \mathsf{r}_{\forall x\forall ygST(x,y)}(\gamma, \nu) \mid \mathsf{r}_{\forall x\forall y\forall zASSO(x,y,z)}\big(\gamma, \mathsf{s}\nu, f(\nu)\big) \mid \\ & \mathsf{r}_{\forall x1L(x)}\big(f(\alpha)\big) \end{aligned}$$

To obtain rules with left side γ, we have to check which instance (or instances) of the induction hypothesis is/are used in π_s. We obtain the following rule:

$$\gamma \to \gamma \cdot \mathsf{s}\nu$$

To obtain rules of the form $\gamma \to t[\alpha]$, we have to check which instance (or instances) of the induction formula is/are used in π_c. We obtain the following rule:

$$\gamma_{end} \to 1$$

This completes the description of $\mathcal{G}(\pi)$.

A schematic grammar G has to be thought of as inducing not a single language but instead a sequence of instance grammars which in turn induces a sequence of languages. If G is produced from a simple induction proof π these instance grammars will be the grammars of the instance proofs of π (in the sense of the definition described on p. 669). Note that while schematic grammars are cyclic, due to the production $\gamma \to t[\alpha, \nu, \gamma]$ the grammars of instance proofs are acyclic. This breaking of cycles in the grammars corresponds to the replacement of the induction by cuts in the proof. Also note that instance grammars only give information about the used instances of cut formulas but not about the cut formulas themselves. Accordingly, the schematic grammar G does not determine the induction formula F of its simple induction proof π.

We define the operation from schematic grammars to instance grammars given as the lower right arrow in Diagram 1.

Definition 3.7 *(From schematic grammars to instance grammars).* Let $G = \langle \tau, N, \Sigma, P \rangle$ be a schematic grammar given as in Definition 3.4. Then for any $n \in \mathbb{N}$ we define the n-th instance grammar as the rigid acyclic tree grammar $G_n = \langle \tau, N_n, \Sigma, P_n \rangle$ where $N_n = \{\tau, \gamma_0, \dots, \gamma_n\}$ and

$$\begin{aligned} P_n = &\left\{\tau \to t[\overline{n}, \gamma_0] \mid \tau \to t[\alpha, \beta] \in P\right\} \\ &\cup \left\{\tau \to t[\overline{n}, \dot{i}, \gamma_{i+1}] \mid \tau \to t[\alpha, \nu, \gamma] \in P, 0 \leq i \leq n-1\right\} \\ &\cup \left\{\gamma_i \to t[\overline{n}, \dot{i}, \gamma_{i+1}] \mid \gamma \to t[\alpha, \nu, \gamma] \in P, 0 \leq i \leq n-1\right\} \\ &\cup \left\{\gamma_n \to t[\overline{n}] \mid \gamma_{end} \to t[\alpha] \in P\right\} \end{aligned}$$

We call the collection $G_{n \geq 0}$ instance grammars of G.

Example 3.8. Let π be the s.i.p. defined in Example 3.2. Its grammar $\mathcal{G}(\pi)$ has been described in Example 3.6. Its instance grammar $\mathcal{G}(\pi)_2 = \langle \tau, N_2, \Sigma, P_2 \rangle$ where $N_2 = \{\tau, \gamma_0, \gamma_1, \gamma_2\}$ and $P_2 =$

$$\begin{aligned} \tau \to\ & \mathsf{r}_{f0} \mid \mathsf{r}_{\forall x g 0(x)}(\gamma_0) \mid \mathsf{r}_{\forall x 1 R(x)}(\gamma_0) \mid \\ & \mathsf{r}_{\forall x f S T(x)}(0) \mid \mathsf{r}_{\forall x \forall y g S T(x,y)}(\gamma_1, 0) \mid \mathsf{r}_{\forall x \forall y \forall z A S S O(x,y,z)}\big(\gamma_1, \mathsf{s}0, f(0)\big) \mid \\ & \mathsf{r}_{\forall x f S T(x)}(\mathsf{s}0) \mid \mathsf{r}_{\forall x \forall y g S T(x,y)}(\gamma_2, \mathsf{s}0) \mid \mathsf{r}_{\forall x \forall y \forall z A S S O(x,y,z)}\big(\gamma_2, \mathsf{ss}0, f(\mathsf{s}0)\big) \mid \\ & \mathsf{r}_{\forall x 1 L(x)}\big(f(\mathsf{ss}0)\big) \\ \gamma_0 \to\ & \gamma_1 \cdot \mathsf{s}0 \\ \gamma_1 \to\ & \gamma_2 \cdot \mathsf{ss}0 \\ \gamma_2 \to\ & \mathsf{s}0 \end{aligned}$$

Let π_2 be the 2nd instance proof of π as in Definition 3.3. Then we have $\mathcal{G}(\pi)_2 = \mathcal{G}(\pi_2)$ as the interested reader is invited to verify. Moreover, by Theorem 2.9, we know that $\mathcal{L}(\mathcal{G}(\pi)_2) = \mathcal{L}(\mathcal{G}(\pi_2))$ is a Herbrand sequent of $\forall \Gamma \Rightarrow g(\overline{1}, \overline{2}) = f(\overline{2})$.

The algorithm sketched on p. 669 maps each simple proof π to a grammar $\mathcal{G}(\pi)$ whose language corresponds to a Herbrand sequent of the conclusion of π, which is stated as Theorem 2.9 in this paper. This yields the missing down arrow on the right in Diagram 1. The commutativity of Diagram 1 is now easy to see.

Proposition 3.9. *Diagram 1 commutes.*

Consider the extension of Diagram 1 for a simple induction proof π of $\Gamma \Rightarrow B[\alpha]$ to Diagram 2. The additional upper horizontal arrow denotes non-erasing Gentzen cut elimination which produces a normal

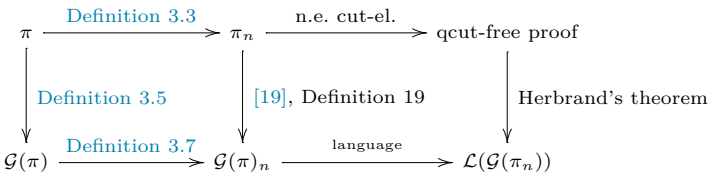

Diagram 2. Dependencies between grammars and proofs.

form which is free of cuts with quantifier inferences (qcut-free). From such proofs the standard technique (see Theorem 2.4) can be used to read off a Herbrand sequent of linear size. Moreover, all normal forms of this non-erasing reduction have the same Herbrand sequent, see [24] for details. The additional lower horizontal arrow denotes language production in the sense of rigid, acyclic tree grammars. The down arrow denotes the usual operation to obtain a Herbrand sequent from a cut-free proof of a Σ_1 sequent as in the proof of Theorem 2.4. Then, the next proposition follows immediately from the results of [19,21].

Proposition 3.10. *Diagram 2 commutes.*

Let π be a simple induction proof. By eliminating induction according to Definition 3.3 we obtain the simple proofs $\pi_{n\geq 0}$. Applying cut-elimination to them yields a Herbrand sequent $\mathcal{L}(\mathcal{G}(\pi_n))$ of $\Gamma \Rightarrow B[\overline{n}]$ for each $n \in \mathbb{N}$. In this paper, we aim at inverting this operation, i.e. at producing a simple induction proof from a finite sub-sequence of a sequence of Herbrand sequents of instances of a Σ_1 sequent. It is an immediate corollary of Gödel's second incompleteness theorem that this is impossible in full generality:

Proposition 3.11. *There is a Σ_1 sequent $\Gamma \Rightarrow B[\alpha]$ with B quantifier-free such that:*

- *There is a collection $HS(n)_{n\geq 0}$ of Herbrand sequents of $\Gamma \Rightarrow B[\overline{n}]$.*
- *There is no simple induction proof of $\Gamma \Rightarrow B[\alpha]$.*

Proof Sketch. An obvious example for such a Σ_1 sequent contains $B[x]$ expressing that x is not a proof of $\overline{0} = \overline{1}$ in a consistent logical system comprising Peano arithmetic with Γ containing the recursion equations for several primitive recursive functions which together allow a coding of metamathematics. For this sequent the statement follows immediately from Gödel's second incompleteness theorem. □

On the other hand, if there is a simple induction proof then there is a corresponding sequence of Herbrand sequents. Now, what is the difference between such sequences of Herbrand sequents which do and such which do not correspond to simple induction proofs? On a high level the answer is clear: the former have a uniformity which the latter lack. Of which nature exactly this uniformity is and how to characterise it combinatorially is much less clear. One obvious consequence of the considerations described in this section is that a sequence generated by a simple induction proof is also generated by a schematic grammar. And indeed, for constructing a situation as in Proposition 3.11 it is enough to consider a sequent which cannot be proved by Herbrand sequents producible by a schematic grammar. In the below Proposition 3.13 we strengthen Proposition 3.11 in the sense that not even the two right arrows at the bottom of Diagram 2 can be inverted. To that aim it will be helpful to first make a simple observation on the growth of the instance languages of a schematic grammar.

Lemma 3.12. *Let $G = \langle \tau, N, \Sigma, P\rangle$ be a schematic grammar. Then there is a $k \in \mathbb{N}$ such that $|\mathcal{L}(G_n)| \leq k^{n+3}$.*

Proof. Let $k_1 = |\{\tau \to t[\alpha, \beta] \in P\}|$, $k_2 = |\{\tau \to t[\alpha, \nu, \gamma] \in P\}|$, $k_3 = |\{\gamma \to t[\alpha, \nu, \gamma] \in P\}|$, and $k_4 = |\{\gamma_{end} \to t[\alpha] \in P\}|$. For a non-terminal δ let $\mathcal{L}(G_n, \delta) = \{t \in \mathcal{T}(\Sigma) \mid t \text{ derivable from } \delta \text{ in } G_n\}$. We obtain

$$\left|\mathcal{L}(G_n, \gamma_l)\right| \leq k_3^{n-\ell} k_4 \quad \text{for all } \ell \in \{0, \dots, n\}$$

directly from the definition of G_n. Furthermore we have

$$\left|\mathcal{L}(G_n, \tau)\right| \leq k_1 k_3^n k_4 + \sum_{i=0}^{n-1} k_2 k_3^{n-(i+1)} k_4 \leq (k_1 + k_2) k_4 \sum_{i=0}^{n} k_3^i$$

from which, by letting $k = \min\{2, \max\{k_1 + k_2, k_3, k_4\}\}$, we obtain $|\mathcal{L}(G_n)| \leq k^{n+3}$. □

Proposition 3.13. *There is a Σ_1 sequent $\Gamma \Rightarrow B[\alpha]$ with B quantifier-free such that:*

- *There is a collection $HS(n)_{n\geq 0}$ of Herbrand sequents of $\Gamma \Rightarrow B[\overline{n}]$.*
- *There is no schematic grammar G such that its instance grammars G_n produce Herbrand sequents of $\Gamma \Rightarrow B[\overline{n}]$ for all $n \in \mathbb{N}$.*

Proof. Let F represent the usual axioms for the factorial f defined by head-recursion given in Example 3.2, and let M and A represent usual axioms for multiplication and addition. Let P be a predicate variable. We define the sequent $\Gamma \Rightarrow B[\alpha]$ as

$$F, M, A, P(0), \forall x\big(P(x) \to P(\mathsf{s}x)\big) \Rightarrow P\big(f(\alpha)\big).$$

For all $n \in \mathbb{N}$, a Herbrand sequent of $\Gamma \Rightarrow B[\overline{n}]$ is given by

$$F', M', A', P(0), P(0) \to P(\overline{1}), \dots, P(\overline{n! - 1}) \to P(\overline{n!}) \Rightarrow P\big(f(\overline{n})\big),$$

where F', M', and A' consist of the instances of F, M, and A respectively which are necessary to derive the equation $f(\overline{n}) = \overline{n!}$. Note that this Herbrand sequent is minimal w.r.t. the number of instances of $\forall x(P(x) \to P(\mathsf{s}x))$ as the following lemma shows.

Lemma 3.14. *If $\Gamma' \Rightarrow P(f(\overline{n}))$ is a Herbrand sequent of $\Gamma \Rightarrow P(f(\overline{n}))$ then Γ' contains at least $n!$ instances of $\forall x(P(x) \to P(\mathsf{s}x))$.*

Proof. W.l.o.g. we can assume that Γ' contains only ground formulas. Hence every term t occurring in Γ' has a value $|t| \in \mathbb{N}$ given by the standard interpretation of 0, s, $+$, $\cdot$, and f. Let

$$P(t_1) \to P(\mathsf{s}t_1), \dots, P(t_k) \to P(\mathsf{s}t_k)$$

be the instances of $\forall x(P(x) \to P(\mathsf{s}x))$ in Γ'. Suppose that $k < n!$, then there is a $c \in \{0, 1, \dots, n! - 1\}$ s.t. there is no $i \in \{1, \dots k\}$ with $|t_i| = c$. Define the structure $\mathcal{M} = (\mathbb{N}, I)$ where $I(0)$, $I(\mathsf{s})$, $I(+)$, $I(\cdot)$ and $I(f)$ are the respective standard interpretations and $I(P) = \{0, \dots, c\}$. Then $\mathcal{M} \vDash F$, $\mathcal{M} \vDash M$, $\mathcal{M} \vDash A$, and $M \vDash P(0)$. Furthermore $\mathcal{M} \vDash P(t_i) \to P(\mathsf{s}t_i)$ for $|t_i| < c$ because $\mathcal{M} \vDash P(t_i)$ and $\mathcal{M} \vDash P(\mathsf{s}t_i)$. But we also have $\mathcal{M} \vDash P(t_i) \to P(\mathsf{s}t_i)$ for $|t_i| > c$ because $\mathcal{M} \nvDash P(t_i)$ and $\mathcal{M} \nvDash P(\mathsf{s}t_i)$. Hence $\mathcal{M} \vDash \Gamma'$ but $\mathcal{M} \nvDash P(f(\overline{n}))$ and thus $\Gamma' \Rightarrow P(f(\overline{n}))$ is not a Herbrand sequent of $\Gamma \Rightarrow P(f(\overline{n}))$. □

This immediately implies that there is no schematic grammar with instance grammars producing the necessary sequence of Herbrand sequents since, by Lemma 3.12, the size of the languages produced by the instance grammars G_n grows only exponentially in n. □

At this point it is not yet clear whether a situation as in Proposition 3.11 can arise in a way different from Proposition 3.13. Or, phrased differently, while the existence of a schematic grammar is clearly necessary

for a sequence of Herbrand sequents to be generated by an s.i.p. it is not yet clear whether it is sufficient. We will see later, in Proposition 6.7, that this is not the case.

4. Searching for a simple induction proof

Despite the negative results about the general case shown above, a computation of a simple induction proof from a sequence of Herbrand sequents *is possible* provided this sequence is *sufficiently uniform*.

In this short section, we sketch our algorithm IndProof for constructing an inductive proof of a universal statement based on given proofs of instances of this statement. The assumption that proofs of instances of a universal statement are easily available is practically justified since typically the instances are provable without induction and hence powerful methods from automated deduction like term rewriting [2] or SMT-solving [11] can be used.

Let us explain the behaviour of our algorithm using Diagram 2: suppose we want to prove a statement of the form $\Gamma \Rightarrow B[\alpha]$ for a free variable α only occurring in B. We first obtain proofs of $\Gamma \Rightarrow B[\overline{n}]$ for $n \in M$ where M is a finite test-set of natural numbers. From these proofs, we obtain Herbrand sequents $HS(n)$ of $\Gamma \Rightarrow B[\overline{n}]$ for $n \in M$. In the diagram they are located at the right side at the bottom. IndProof consists of two major steps, FindGram and FindFml. FindGram produces from the given Herbrand sequents $HS(n)$ for $n \in M$ a schematic grammar G with $HS(n) \subseteq \mathcal{L}(G_n)$ for all $n \in M$. The algorithm FindGram thus reverses the two lower right arrows in the diagram. FindFml produces from a schematic grammar G with $HS(n) \subseteq \mathcal{L}(G_n)$ for all $n \in M$ a simple induction proof of $\Gamma \Rightarrow B[\alpha]$ with schematic grammar G, thus reversing the down arrow on the left.

We will describe FindGram in Section 5 and FindFml in Section 6.

5. Finding a schematic grammar

In this section, we define the algorithm FindGram. It relies on the algorithms ConstKeys, ConstFml, SolveSAT, and EvalToFml which will be given in detail in the following.

Notation 5.1. Let $\Theta[\alpha] := \forall \Gamma \Rightarrow B[\alpha]$ be a Σ_1 sequent with α only occurring in the quantifier-free formula B. Let $M \subseteq \mathbb{N}$ and let $HS(n)$ be a Herbrand sequent of $\Theta[\overline{n}]$ for all $n \in M$. Let G be a schematic grammar. Then, G is called a schematic grammar for $(HS(n))_{n \in M}$ if $HS(n) \subseteq \mathcal{L}(G_n)$ for all $n \in M$.

Algorithm 1. FindGram

Input: List of Herbrand sequents $HS(i_1), \ldots, HS(i_q)$ of $\Theta[i_1], \ldots, \Theta[i_q]$ for $i_1, \ldots, i_q \in \mathbb{N}$ where Θ is a Σ_1 sequent given as in Notation 5.1.

Output: A schematic grammar G for $HS(i_1), \ldots, HS(i_q)$ with a minimal number of rules among all schematic grammars for $HS(i_1), \ldots, HS(i_q)$.

Execution: Apply the following four algorithms consecutively: ConstKeys, ConstFml, SolveSAT, and EvalToFml.

end

We sketch the steps of FindGram which will be detailed later:

1. ConstKeys produces from the input of FindGram a decomposition list of the terms of $\bigcup_{1 \leq j \leq q} HS(i_j)$. Its entries will contain all terms possibly occurring as right sides of a minimal schematic grammar for $HS(i_1), \ldots, HS(i_q)$.

2. ConstFml translates the output of ConstKeys into a propositional formula F s.t. every evaluation of F to true will give rise to a specific selection of rules, and will completely determine a schematic grammar. The purpose of this translation is to allow the application of optimised algorithms from the SAT community.
3. SolveSAT searches an evaluation E of F to true evaluating a minimal number of so-called counting atoms to true. Such an evaluation corresponds to a schematic grammar with a minimal number of production rules.
4. Finally, EvalToFml transforms E into the schematic grammar which is given as output.

The algorithm FindGram relies on algorithms and techniques presented in [12]. Let us briefly cite some important results of [12]: An efficient algorithm Decomp is presented there for the decomposition of an input language into a minimal rigid, acyclic tree grammar. The computational difficulty in finding a small grammar G producing an input language L is that a priori an enormous number of different terms k might occur as right side of rules of G. In fact, the number of terms k possibly occurring in a derivation of a term $t \in L$ is exponential in the size of t. In [12], we proved that to obtain a grammar G of minimal size producing an input language L, we can restrict ourselves to terms k in a certain normal form as right side of rules of G. The number of such terms is only polynomial in the size of L. The problem of finding an optimal set of right sides of rules is then translated into a weighted SAT problem for which efficient algorithms are known.

Let us discuss how these results and techniques can be applied to our setting. The input of FindGram is a list of Herbrand sequents $HS(i_1), \ldots, HS(i_q)$ for $i_1, \ldots, i_q \in \mathbb{N}$. Note that the instance grammars of the searched schematic grammar are rigid acyclic tree grammars. Nevertheless, in the case of induction proofs, the additional difficulty is that these instance grammars depend on each other since they are instance grammars of the same schematic grammar. Therefore, it does not make sense to minimise their size individually.

Nevertheless, also in the case of induction proofs the restriction of the terms possibly occurring as right side of rules of the schematic grammar is crucial. To this aim, we will introduce an adapted notion of normal form, the so-called restricted normal form. Then, analogously as in [12], it can be shown that we can limit ourselves to terms in restricted normal form as right sides of rules of the searched schematic grammar. Then, again as in [12], the problem of choosing an optimal set of terms in restricted normal form as right sides of rules is transformed into a weighted SAT problem.

5.1. Keys in restricted normal form

We now introduce the concepts of [12] necessary for the definition of FindGram. To present the crucial but involved definition of a key in normal form, we have to introduce some notation and auxiliary definitions.

Notation 5.2. We always use τ to denote the axiom and $\alpha_1, \alpha_2, \ldots$ to denote the remaining non-terminals of a grammar. Terms not containing non-terminals are called closed. Finite sets of closed terms are called languages.

For a term k containing at most non-terminals $\alpha_1, \ldots, \alpha_n$, for closed terms $t_1, \ldots, t_n$ and a closed term q we write

$$k \circ_{\alpha_1,\ldots,\alpha_n} \begin{pmatrix} t_1 \\ \vdots \\ t_n \end{pmatrix} = q$$

for $k[\alpha_1 \backslash t_1, \ldots, \alpha_n \backslash t_n] = q$. We call $t_1, \ldots, t_n$ the rests of the decomposition of q by k. More specifically, t_i is the rest of α_i. Note that since $t_1, \ldots, t_n$ are closed the above displayed substitutions can be applied in any order. Note that the terms $t_1, \ldots, t_n$ are uniquely determined by the pair k, q.

For a term k containing at most non-terminals $\alpha_1, \ldots, \alpha_n$ and closed terms $t_{1,1}, \ldots, t_{1,n}, \ldots, t_{m,1}, \ldots, t_{m,n}$ we write

$$k \circ_{\alpha_1,\ldots,\alpha_n} \left\{ \begin{pmatrix} t_{1,1} \\ \vdots \\ t_{1,n} \end{pmatrix}, \ldots, \begin{pmatrix} t_{m,1} \\ \vdots \\ t_{m,n} \end{pmatrix} \right\}$$

for

$$\left\{ k \circ_{\alpha_1,\ldots,\alpha_n} \begin{pmatrix} t_{i,1} \\ \vdots \\ t_{i,n} \end{pmatrix} : 1 \leq i \leq m \right\}$$

Definition 5.3 *(Key of a language).* A term k containing at most non-terminals $\alpha_1, \ldots, \alpha_n$ is called key of a language M if there is a set R of vectors of closed terms such that $k \circ_{\alpha_1,\ldots,\alpha_n} R = M$.

Notation 5.4. A term k is called key of a grammar/schematic grammar G exactly if G contains a rule with right side k.

Notation 5.5. For a set of terms M, let $st(M)$ denote the set of subterms of terms in M.

Proposition 5.6. *Let G be a grammar. Every key k of G is key of a language $L_k \subseteq st(\mathcal{L}(G))$.*

Proof. Let $\gamma \to k$ be a production of G and $\tau = t_1, t_2, \ldots, t_n = t$ a derivation of a closed term t in G which uses $\gamma \to k$. Let l be the index of the first application of $\gamma \to k$, then $t_l = r[\gamma]_p$ and $t_{l+1} = r[k]_p$ for some position p. Then k is key of the singleton set $\{t|_p\} \subseteq st(\mathcal{L}(G))$ as the derivation $t_{l+1}|_p, \ldots, t_n|_p$ of G shows. □

The above proof shows that every key k of a grammar G is even a key of a singleton language contained in $st(\mathcal{L}(G))$. When trying to find small grammars the challenge consists in finding keys of languages with as many elements as possible. A central result of [12], and the basis for the decomposition algorithm presented there, is that for a minimal decomposition of an input language L only a small number of all keys of subsets of $st(L)$ suffices.

Definition 5.7 *(Induced substitution).* Let k be a key with non-terminals $\alpha_1, \ldots, \alpha_n$ for a language M. Let $q \in M$ and

$$k \circ_{\alpha_1,\ldots,\alpha_n} \begin{pmatrix} t_1 \\ \vdots \\ t_n \end{pmatrix} = q.$$

Then we write $\sigma_{k,q}$ for the following substitution:

$$[\alpha_1 \backslash t_1, \ldots, \alpha_n \backslash t_n]$$

We often write σ_q if the key k is clear from the context.

Definition 5.8. Let k be a key with non-terminals $\alpha_1, \ldots, \alpha_n$ for the language M. Let t_0, t_1 be terms containing at most the non-terminals $\alpha_1, \ldots, \alpha_n$. We say that M satisfies the equation $t_0 = t_1$ for k if for all $t \in M$:

$$t_0\sigma_{k,t} = t_1\sigma_{k,t}.$$

If M does not satisfy an equation $t_0 = t_1$ for k, then there is a term $t \in M$ s.t. $t_0\sigma_{k,t} \neq t_1\sigma_{k,t}$. In that case we say that t falsifies the equation $t_0 = t_1$.

Example 5.9. Let $M = \{f(g(c), g(c)), f(g(d), g(d))\}$ and $k = f(\alpha_1, g(\alpha_2))$ be a key of M. We have

- $\sigma_{k,f(g(c),g(c))} = [\alpha_1 \backslash g(c), \alpha_2 \backslash c]$ and
- $\sigma_{k,f(g(d),g(d))} = [\alpha_1 \backslash g(d), \alpha_2 \backslash d]$.

Accordingly, M fulfils the equation $\alpha_1 = g(\alpha_2)$.

Notation 5.10. We call an equation trivial if it is of the form $s = s$ for a term s.

Definition 5.11 *(Normal form of decomposition key).* Assume that k is a key with non-terminals $\alpha_1, \ldots, \alpha_n$ for the language M. Then k is in normal form relative to M exactly if the following condition holds:

For any non-trivial equation of the form $\alpha_i = q$ for $1 \leq i \leq n$ for q being a subterm of k or a closed term there is a term $t \in M$ falsifying it.

Intuitively, keys in normal form do not contain idle non-terminals: If M satisfies an equation of the form $\alpha_i = q$ for a key k given as above, it is easy to see that the key $k[\alpha_i \backslash q]$ decomposes M as well and that $k[\alpha_i \backslash q]$ contains one non-terminal less than k.

Example 5.12. Let k and M be given as in Example 5.9. Then k is not in normal form relative to M because M satisfies the equation $\alpha_1 = g(\alpha_2)$ for k and $g(\alpha_2)$ is a subterm of k. However, the key $k[\alpha_1 \backslash g(\alpha_2)] = f(g(\alpha_2), g(\alpha_2))$ is in normal form relative to M. Let $M' = M \cup \{f(g(c), g(d))\}$. Then k is in normal form relative to M' as, in particular, M' no longer satisfies the equation $\alpha_1 = g(\alpha_2)$ for k.

Notation 5.13. Let G be a grammar. The size of G, written as $|G|$, is defined as the number of rules of G.

Definition 5.14 *(Marked term).* An n-marked term t is a pair $\langle n, t' \rangle$ where $n \in \mathbb{N}$ and t is a term.

Definition 5.15 *(Marked language).* An n-marked language is a finite set of n-marked closed terms.

Remark 5.16. FindGram will interpret its input language $HS(n)$ as an n-marked language.

Notation 5.17.

- For an n-marked term $t = \langle n, t' \rangle$ in notations and concepts referring to terms, we often write t instead of t'. E.g. we write that k is in normal form relative to $\{t\}$ meaning that k is in normal form relative to $\{t'\}$.
- We use L_n for $n \in \mathbb{N}$ to refer to an n-marked language.

Notation 5.18. Let $T = \{\langle n_1, t_1 \rangle, \ldots, \langle n_m, t_m \rangle\}$ be a set of marked terms. Then, extending Notation 5.5, we write $st(T)$ for the following set of marked terms:

$$\big\{\langle n, u \rangle \mid \exists \langle n, t \rangle \in T \text{ such that } u \in st\big(\{t\}\big)\big\}.$$

Recall that we are in the process of defining the algorithm FindGram whose input is a list $HS(i_1), \ldots, HS(i_q)$ of Herbrand sequents and whose output is a schematic grammar G with $HS(i_j) \subseteq \mathcal{L}(G_{i_j})$ for all $j \in \{1 \ldots, q\}$. Let us give some motivation for the following definition of the restricted normal form.

We will be interested in sets $M \subseteq st(T)$ for the set of marked terms $T := \bigcup_{1\leq j\leq q} HS(i_j)$. Since in a schematic grammar, the non-terminals α, ν, γ have a special interpretation, we can restrict the terms that can be possibly substituted for them. E.g. for a term $t \in HS(i_j)$ for $1 \leq j \leq q$ in each derivation of t in an instance grammar G_{i_j} of a schematic grammar G, α will be replaced by $\overline{i_j}$ where $\overline{n}$ is the numeral of $n \in \mathbb{N}$. Similar restrictions apply to ν. The restriction of the normal form makes sure that only keys respecting this special interpretation of the non-terminals are considered.

Definition 5.19 *(Restricted normal form of decomposition key).* A key k containing at most non-terminals α, ν, γ of a set of closed marked terms M is in restricted normal form relative to M exactly if the following conditions hold.

- k is in normal form relative to M.
- For all $\langle j, t\rangle \in M$ with $j > 0$ we have $\alpha\sigma_{k,t} = \overline{j}$ and $\nu\sigma_{k,t} = \overline{i}$ for some $i \in \{0, \ldots, j-1\}$.
- For all $\langle 0, t\rangle \in M$ we have $\alpha\sigma_{k,t} = 0$ and ν does not occur in k.

The next lemma is proved analogously to the previously mentioned result in [12].

Lemma 5.20. *Let $L_{i_1}, \ldots, L_{i_q}$ be marked languages for $i_1, \ldots, i_q \in \mathbb{N}$. Let G be a schematic grammar with $L_{i_j} \subseteq \mathcal{L}(G_{i_j})$ for all $1 \leq j \leq q$. Then there is a schematic grammar G' with $|G'| \leq |G|$ such that*

1. *$L_{i_j} \subseteq \mathcal{L}(G'_{i_j})$ for all $1 \leq j \leq q$ and*
2. *For each key $k[\alpha, \nu, \zeta]$ of G' with $\zeta \in \{\beta, \gamma\}$ there is an $M \subseteq st(\bigcup_{1\leq j\leq q} L_{i_j})$ such that $k[\alpha, \nu, \gamma]$ is in restricted normal form relative to M.*

The previous lemma implies that we only have to consider schematic grammars containing exclusively keys in restricted normal form. FindGram will collect such keys into a decomposition list.

Definition 5.21 *(Decomposition list for 3 nonterminals).* Let T be a set of closed marked terms. D is a decomposition list for 3 non-terminals of T exactly if the following property holds.

For all $M \subseteq st(T)$ with $M = \{t_1, \ldots, t_m\}$ and u with non-terminals α, ν, γ in restricted normal form relative to M with $u \circ_{\alpha,\nu,\gamma} S = M$ and S of the form

$$\left\{ \begin{pmatrix} s_{1,1} \\ s_{1,2} \\ s_{1,3} \end{pmatrix}, \ldots, \begin{pmatrix} s_{m,1} \\ s_{m,2} \\ s_{m,3} \end{pmatrix} \right\}$$

D contains an entry $\left(t_i, u, \begin{pmatrix} s_{i,1} \\ s_{i,2} \\ s_{i,3} \end{pmatrix}\right)$ for each $1 \leq i \leq m$.

In addition, D does not contain other entries.

Lemma 5.22. *Let $L_{i_1}, \ldots, L_{i_q}$ be marked languages for $i_1, \ldots, i_q \in \mathbb{N}$. Let G be a schematic grammar with $L_{i_j} \subseteq \mathcal{L}(G_{i_j})$ for all $1 \leq j \leq q$. Then there is a schematic grammar G' with $|G'| \leq |G|$ such that*

1. *$L_{i_j} \subseteq \mathcal{L}(G'_{i_j})$ for all $1 \leq j \leq q$ and*
2. *G' contains only keys $k[\alpha, \nu, \zeta]$ with $\zeta \in \{\beta, \gamma\}$ such that $k[\alpha, \nu, \gamma]$ occurs as second component of an entry in the decomposition list for 3 non-terminals of $\bigcup_{1\leq j\leq q} L_{i_j}$.*

Proof. By Lemma 5.20 there is a schematic grammar G' with 1 s.t. for every key $k[\alpha, \nu, \zeta]$ of G' with $\zeta \in \{\beta, \gamma\}$ there is an $M \subseteq st(\bigcup_{1\leq j\leq q} L_{i_j})$ s.t. $k[\alpha, \nu, \gamma]$ is in restricted normal form relative to M. Thus by Definition 5.21 the decomposition list D contains an entry with $k[\alpha, \nu, \gamma]$ as second component hence 2. □

We are now able to define the first step ConstKeys of FindGram in detail.

Algorithm 2. ConstKeys

Input: Marked languages $L_{i_1}, \cdots, L_{i_q}$ with $i_1, \ldots, i_q \in \mathbb{N}$ (intended input: Herbrand sequents).
Output: Decomposition list for 3 non-terminals of $\bigcup_{1 \leq j \leq q} L_{i_j}$.
Execution:
Clearly, a decomposition list of $\bigcup_{1 \leq j \leq q} L_{i_j}$ for 3 non-terminals can be produced in exponential time in $\bigcup_{1 \leq j \leq q} L_{i_j}$. Nevertheless, a slight modification and adaptation of the algorithm ConstNF presented in [12] allows the production of the searched decomposition list in polynomial time in our case. Due to space constraints we refrain from describing this adaptation here.

end

We call the output decomposition list of ConstKeys D. The non-terminals of the keys in D are α, ν, γ.

5.2. Decomposition list as propositional formula

Next, we present the second step ConstFml of FindGram which converts D into a propositional formula carrying equal information. The rationale of this translation is to allow the application of optimised algorithms from the SAT community for finding a minimal schematic grammar.

Algorithm 3. ConstFml

Input: Marked languages $L_{\ell_1}, \ldots, L_{\ell_q}$ with $\ell_1, \ldots, \ell_q \in \mathbb{N}$, decomposition list D of 3 non-terminals of $\bigcup_{1 \leq k \leq q} L_{\ell_k}$
Output: Propositional formula $\mathsf{ConstFml}(L_{\ell_1}, \ldots, L_{\ell_q}, D)$ defined in Definition 5.27.

end

The formula $F := \mathsf{ConstFml}(L_{\ell_1}, \ldots, L_{\ell_q}, D)$ will be constructed such that each assignment of truth values to the atoms of F satisfying F will correspond to a schematic grammar whose instance grammars derive all terms in the marked languages $L_{\ell_1}, \ldots, L_{\ell_q}$. Moreover, the size of this schematic grammar is the number of certain so-called counting atoms of F which are set to true by the evaluation. First, we will sketch F and describe the strategy to build it. Then, we will define F precisely in Definition 5.27.

Notation 5.23. In the following, we denote propositional atoms by $x_{s,t}$ for strings s and t. If strings s', t' are syntactically different from s, t, we assume that the atom $x_{s,t}$ is different from $x_{s',t'}$.

F will be composed of two sorts of atoms. The first sort of atoms are of the form x_{t,γ_i} where t is a subterm of a term in one of the input marked languages and $i \in \mathbb{N}$. The second sort of atoms (the counting atoms) are of the form $x_{premise,k}$ where *premise* is one of the strings *start–*β, *start–*γ, *step*, or *end* and k is a key which occurs as second component of an entry of the input decomposition list D.

As mentioned above, each evaluation E of the atoms of F satisfying F corresponds to a schematic grammar G_E whose instance grammars derive the input marked languages. Accordingly, each assignment of the truth value true to an atom $x_{s,t}$ of F by E fixes some properties of G_E. We will say that $x_{s,t}$ expresses this property in the following.

Atoms of the form x_{t,γ_i} with t being an ℓ_j-marked term for $1 \leq j \leq q$ express that t occurs as the γ_{i-1} rest (see Notation 5.2) of a derivation of a term in L_{ℓ_j} in the instance grammar G_{E,ℓ_j} of the schematic grammar G_E for $1 \leq i \leq \ell_j + 1$. For $i = 0$ it expresses $t \in L_{i_j}$, i.e. it has to be assigned true by all evaluations E satisfying F for $t \in L_{i_j}$.

The meaning of the other sort of atoms becomes clearer if we describe the form of F: F will be a large conjunction of subformulas which are roughly of the form

$$x_{t,\gamma_i} \to (x_{premise_1,key_1} \wedge x_{t'_1,\gamma_{i+1}}) \vee \cdots \vee (x_{premise_\kappa,key_\kappa} \wedge x_{t'_\kappa,\gamma_{i+1}}).$$

They have to be interpreted as follows for $i > 0$: If t occurs as the γ_{i-1} rest of a derivation, then for some $1 \leq \zeta \leq \kappa$ the schematic grammar G_E must contain a rule which is completely determined by $premise_\zeta$ and key_ζ, and as the result of applying this rule to decompose t, we receive as rest a new term t'_ζ to decompose. Let us detail how pairs of *premise* and $k[\alpha,\gamma,\nu]$ fix the mentioned rules of G_E:

- For $premise = start$–β, $x_{premise,k}$ expresses that G_E contains the rule $\tau \to k[\alpha,\beta,\nu]$.
- For $premise = start$–γ, $x_{premise,k}$ expresses that G_E contains the rule $\tau \to k[\alpha,\gamma,\nu]$.
- For $premise = step$, $x_{premise,k}$ expresses that G_E contains the rule $\gamma \to k[\alpha,\gamma,\nu]$.
- For $premise = end$, $x_{premise,k}$ expresses that G_E contains the rule $\gamma_{end} \to k[\alpha]$ ($\equiv k[\alpha,\gamma,\nu]$ in this case).

Note how these four possibilities correspond to the four types of productions in a schematic grammar (Definition 3.4). Keeping this motivation in mind, the definition of the formula F is an easy but tedious exercise in bookkeeping. We prepare its definition by some technically motivated auxiliary definitions.

Notation 5.24. Let a decomposition list D be given. We define $\overline{D}$ as $\{k : \exists x, y \langle x,k,y\rangle \in D\}$.

The following definition will allow to treat keys of $\overline{D}$ containing different subsets of the non-terminal α,γ,ν separately. This will make the definition of F more comprehensive.

Definition 5.25. Let a decomposition list D be given. Then the disjoint subsets D_1, D_2, D_3, D_4 of D are defined as follows.

$$\begin{aligned}
D_1 &:= \{k \in \overline{D} : k\text{contains at most the non-terminal } \alpha\}\\
D_2 &:= \{k \in \overline{D} : k\text{contains exactly the non-terminals } \alpha,\nu\}\\
&\quad \cup \{k \in \overline{D} : k \text{ contains exactly the non-terminal } \nu\}\\
D_3 &:= \{k \in \overline{D} : k \text{ contains exactly the non-terminals } \alpha,\gamma\}\\
&\quad \cup \{k \in \overline{D} : k \text{ contains exactly the non-terminal } \gamma\}\\
D_4 &:= \{k \in \overline{D} : k \text{ contains at least the non-terminals } \nu,\gamma\}
\end{aligned}$$

For an input n-marked term t, the decomposition list D gives rise to various decompositions. The collection of these decompositions will be described by disjunctions of the form $\mathsf{Dis}_{D,t,\cdot,\cdot}$ which will be defined in the following. The third subscript of $\mathsf{Dis}_{D,t,\cdot,\cdot}$ will determine how the non-terminal ν has to be replaced in the keys used for the mentioned decompositions. The fourth subscript of $\mathsf{Dis}_{D,t,\cdot,\cdot}$ will determine the non-terminals which may appear in the keys of the mentioned decompositions following the above Definition 5.25.

Definition 5.26. Let a decomposition list D be given. Let t be an n-marked term for $n \in \mathbb{N}$ and let $i \in \{1,\dots,n\}$. Then, we define

$$\mathsf{Dis}_{D,t,0,1} := \bigvee_{k \in D_1, t = k[\overline{n}]} x_{start\text{–}\gamma,k}$$

$$\mathsf{Dis}_{D,t,i,1} := \bigvee_{k \in D_1, t = k[\overline{n}]} x_{step,k}$$

$$\mathsf{Dis}_{D,t,n+1,1} := \bigvee_{k\in D_1, t=k[\overline{n}]} x_{end,k}$$

$$\mathsf{Dis}_{D,t,0,2} := \bigvee_{k\in D_2, 0\leq m\leq n-1, t=k[\overline{n},\overline{m}]} x_{start-\gamma,k}$$

$$\mathsf{Dis}_{D,t,i,2} := \bigvee_{k\in D_2, t=k[\overline{n},\overline{i-1}]} x_{step,k}$$

$$\mathsf{Dis}_{D,t,n+1,2} := \bot$$

$$\mathsf{Dis}_{D,t,0,3} := \bigvee_{k\in D_3, t=k[\overline{n},r]} \left((x_{start-\beta,k} \wedge x_{r,\gamma_1}) \vee \bigvee_{j=2}^{n+1} (x_{start-\gamma,k} \wedge x_{r,\gamma_j}) \right)$$

$$\mathsf{Dis}_{D,t,i,3} := \bigvee_{k\in D_3, t=k[\overline{n},r]} x_{step,k} \wedge x_{r,\gamma_{i+1}}$$

$$\mathsf{Dis}_{D,t,n+1,3} := \bot$$

$$\mathsf{Dis}_{D,t,0,4} := \bigvee_{k\in D_4, 0\leq m\leq n-1, t=k[\overline{n},r,\overline{m}]} x_{start-\gamma,k} \wedge x_{r,\gamma_{m+2}}$$

$$\mathsf{Dis}_{D,t,i,4} := \bigvee_{k\in D_4, t=k[\overline{n},r,\overline{i-1}]} x_{step,k} \wedge x_{r,\gamma_{i+1}}$$

$$\mathsf{Dis}_{D,t,n+1,4} := \bot$$

Finally, we are in the position to give the definition of the formula $F := \mathsf{ConstFml}(L_{\ell_1}, \dots, L_{\ell_q}, D)$.

Definition 5.27 *(Formula* $\mathsf{ConstFml}(L_{\ell_1}, \dots, L_{\ell_q}, D)$*).* For a decomposition list D, and a set of marked languages $L_{\ell_1}, \dots, L_{\ell_q}$ the formula $\mathsf{ConstFml}(L_{\ell_1}, \dots, L_{\ell_q}, D)$ is defined as

$$\bigwedge_{1\leq j\leq q} \bigwedge_{t\in L_{\ell_j}} \bigwedge_{0\leq i\leq \ell_j+1} \left(x_{t,\gamma_i} \to \bigvee_{1\leq\kappa\leq 4} \mathsf{Dis}_{D,t,i,\kappa} \right) \wedge \bigwedge_{1\leq j\leq q} \bigwedge_{t\in L_{\ell_j}} x_{t,\gamma_0}$$

It is easy to see that the size of the formula $\mathsf{ConstFml}(L_{\ell_1}, \dots, L_{\ell_q}, D)$ is polynomial in the size of its input and that the algorithm $\mathsf{ConstFml}$ can be executed in polynomial time.

It has to be explained yet, how evaluations of F to true correspond to schematic grammars with instance grammars producing the input languages. The following lemma delivers the missing connection.

Lemma 5.28. *Let* $HS(i_1), \cdots, HS(i_q)$ *be Herbrand sequents of a* Σ_1 *sequent and let*

$$F = \mathsf{ConstFml}\big(HS(i_1), \cdots, HS(i_q), \mathsf{ConstKeys}\big(HS(i_1), \cdots, HS(i_q)\big)\big).$$

Then, the following two assertions hold.

- *For each evaluation* E *with* $E(F) = true$ *which evaluates exactly* n *counting atoms as true there is a schematic grammar* G *for* $HS(i_1), \cdots, HS(i_q)$ *with* $|G| = n$.
- *For each schematic grammar* G *for* $HS(i_1), \cdots, HS(i_q)$ *with* $|G| = n$ *there is an evaluation* E *with* $E(F) = true$ *which evaluates exactly* n *counting atoms as true.*

Proof. Let E be given. Now, the searched schematic grammar G is produced by adding rules corresponding to the counting atoms evaluated true as follows:

- If $E(x_{start\text{–}\beta,k[\alpha,\gamma]}) = \mathit{true}$, the rule $\tau \to k[\alpha,\beta]$ is added to G.
- If $E(x_{start\text{–}\gamma,k[\alpha,\gamma]}) = \mathit{true}$, the rule $\tau \to k[\alpha,\gamma]$ is added to G.
- If $E(x_{step,k[\alpha,\nu,\gamma]}) = \mathit{true}$, the rule $\gamma \to k[\alpha,\nu,\gamma]$ is added to G.
- If $E(x_{end,k[\alpha]}) = \mathit{true}$, the rule $\gamma_{end} \to k[\alpha]$ is added to G.

The second assertion follows from Lemma 5.22 using again the correspondence between rules and atoms. □

Note that the formula $\mathsf{ConstFml}(HS(i_1),\cdots,HS(i_q),\mathsf{ConstKeys}(HS(i_1),\cdots,HS(i_q)))$ is satisfiable due to the existence of a trivial schematic grammar for $HS(i_1),\cdots,HS(i_q)$ using only the non-terminal τ.

Remark 5.29. The proof of the previous lemma establishes a transformation $\mathcal{T}$ of evaluations E with $E(F) = \mathit{true}$ into schematic grammars. $\mathcal{T}$ will be used for the definition of the fourth step EvalToFml of FindGram.

5.3. The complete algorithm for finding a schematic grammar

Let us now define the last two steps SolveSAT and EvalToFml of FindGram.

Algorithm 4. SolveSAT

Input: Satisfiable propositional formula F composed of atoms indexed like outputs of ConstFml.
Output: Evaluation E of F to true, evaluating a minimal number of counting atoms as true.
Execution: Apply an efficient weighted max-sat algorithm to F with suitably weighted clauses (see [17] for an example of such an algorithm).

end

Algorithm 5. EvalToFml

Input: Satisfiable propositional formula F composed of atoms indexed like outputs of ConstFml, evaluation E with $E(F) = \mathit{true}$.
Output: Schematic grammar.
Execution: Translate E into a schematic grammar according to transformation $\mathcal{T}$ described in the proof of Lemma 5.28.

end

The precise definition of all steps of FindGram allows to prove its correctness.

Lemma 5.30. *The algorithm* FindGram *satisfies its specification given on p. 677.*

Proof. The lemma immediately from Lemma 5.22, Lemma 5.28, and the proof of Lemma 5.28 assuming that the sketched ConstKeys fulfils its specification. □

Our aim is to produce a schematic grammar G such that its instance grammar G_n produces a Herbrand sequent of a Σ_1 sequent $\forall\Gamma \Rightarrow B[\overline{n}]$ for all $n \in \mathbb{N}$, e.g. a schematic grammar of $\mathbb{N}$ for $\forall\Gamma \Rightarrow B[\alpha]$. However, from our finite input we do not know whether a Herbrand sequent of $\forall\Gamma \Rightarrow B[\overline{n}]$ exists for each $n \in \mathbb{N}$. In addition even if this is the case, because of Proposition 3.13 a schematic grammar with the above mentioned property might not exist. Nevertheless, FindGram always produces an output which has to be seen as a *candidate* for a schematic grammar G of the entire $\mathbb{N}$.

In [13], we proved that it is undecidable for an input schematic grammar with sequent $\Theta[\alpha]$ whether for all $n \in \mathbb{N}$ the instance grammar G_n produces a Herbrand sequent of $\Theta[\overline{n}]$. This immediately implies that it is undecidable whether FindGram from a certain input in fact produced a schematic grammar of $\mathbb{N}$.

Therefore, the algorithm IndProof will just use the output of FindGram, implicitly assuming that it is a schematic grammar of $\mathbb{N}$ for Σ without checking this property.

Let us explain how to choose the input collection of Herbrand sequents, and why we do not just search a schematic grammar for a single Herbrand sequent $HS(n)$ for $n \in \mathbb{N}$.

The flexibility in choosing the input collection allows us to search for minimal schematic grammars first for a small test-collection of Herbrand sequents. Nevertheless, for small test-collections $HS(n_1), \ldots, HS(n_m)$ with small $n_i \in \mathbb{N}$ there is the problem that these might be special cases, and therefore the minimal schematic grammar of $\{n_1, \ldots, n_m\}$ might not be a schematic grammar of larger subsets of natural numbers. Taking a larger test-collection instead might solve this problem. This explains why it is crucial for FindGram to be applicable to an arbitrary (finite) test-collection of marked languages.

Example 5.31. Recall the two definitions of the factorial given in Example 3.2. We sketch a natural sequence of Herbrand sequents with right side $g(1, n) = f(n)$ for $n \in \mathbb{N}$. They give rise to a schematic grammar using the algorithm FindGram.

A natural way to prove the equation $g(1, n) = f(n)$ for $n \in \mathbb{N}$ is to start an equation chain from $g(1, n)$ which successively decreases the second argument of g, and multiplies the first one. One arrives at $g(1 \cdot n \cdot (n-1) \cdot \cdots \cdot 1, 0)$ and continues with $1 \cdot n \cdot (n-1) \cdot \cdots \cdot 1$. From there, calculation rules for 1 and $f(0)$ allow us to reach $(1 \cdot n \cdot (n-1) \cdot \cdots \cdot 1) \cdot f(0)$. Rules for the factorial and associativity allow us to reach $(1 \cdot n \cdot (n-1) \cdot \cdots \cdot 2) \cdot f(1)$. By continuing analogously, we reach $1 \cdot f(n)$. Calculation rules for 1 deliver $f(n)$ which finishes the equation chain.

The described strategy determines a collection of proofs only containing atomic cuts for $g(\overline{1}, \overline{n}) = f(\overline{n})$. They give rise to a Herbrand sequent $HS(n)$ of the form $I_n \Rightarrow g(\overline{1}, \overline{n}) = f(\overline{n})$ for each $n \in \mathbb{N}$ where I_n contains instances of arithmetic axioms and the axioms for f and g as given in Example 3.2. Let us have a closer look at I_n:

- gST is instantiated by the following pairs of terms:

$$\langle \overline{1}, \overline{n-1} \rangle, \langle \overline{1} \cdot \overline{n}, \overline{n-2} \rangle, \langle \overline{1} \cdot \overline{n} \cdot \overline{n-1}, \overline{n-3} \rangle, \ldots, \langle \overline{1} \cdot \overline{n} \cdot \overline{n-1} \cdot \cdots \cdot \overline{2}, 0 \rangle$$

- fST is instantiated by the following terms:

$$0, \overline{1}, \overline{2}, \ldots, \overline{n-1}$$

- $ASSO$ is instantiated by the following triples of terms:

$$\langle \overline{1} \cdot \overline{n} \cdot \overline{n-1} \cdot \cdots \cdot \overline{2}, \overline{1}, f(0) \rangle, \langle \overline{1} \cdot \overline{n} \cdot \overline{n-1} \cdot \cdots \cdot \overline{3}, \overline{2}, f(\overline{1}) \rangle \cdots \langle \overline{1}, \overline{n}, f(\overline{n}-1) \rangle$$

Also all other instances can be obtained from the above sketched proofs easily.

Let us now present a schematic grammar G only using keys in restricted normal form. We will argue that G is the minimal schematic grammar with $HS(2) \subseteq \mathcal{L}(G_2)$ and $HS(3) \subseteq \mathcal{L}(G_3)$. It is easy to see that we even have $HS(n) = \mathcal{L}(G_n)$ for all $n \in \mathbb{N}$. The grammar G contains exactly the following rules:

- $\tau \to \mathsf{r}_{f0} \mid \mathsf{r}_{g0}(\beta) \mid \mathsf{r}_{1R}(\beta)$
- $\tau \to \mathsf{r}_{fST}(\nu) \mid \mathsf{r}_{gST}(\nu, \gamma)$
- $\tau \to \mathsf{r}_{1L}(f(\alpha)) \mid \mathsf{r}_{ASSO}(\gamma, \mathsf{s}\nu, f(\nu))$
- $\gamma \to \gamma \cdot \mathsf{s}\nu$
- $\gamma_{end} \to 1$

Lemma 5.32. *The grammar G defined above is the only schematic grammar G' with $HS(2) \subseteq \mathcal{L}(G'_2)$ and $HS(3) \subseteq \mathcal{L}(G'_3)$ and $|G'| \leq 9$.*

Proof. Assume that there is a schematic grammar G' containing 9 or less rules fulfilling the above mentioned condition. Instances of all the mentioned axioms occur in $HS(n)$ for $n > 0$, therefore G' contains one rule of the form $\tau \to \mathsf{r}_{Ax}(\cdots)$ for $Ax = f0, g0, \ldots, ASSO$. $HS(2)$ and $HS(3)$ contain 2, respectively 3 pairwise different terms of the form $\mathsf{r}_{gST}(\cdots)$. Let us call these terms s_1, s_2 and t_1, t_2, t_3, respectively. The rule of the form $\tau \to \mathsf{r}_{gST}(\cdots)$ contained in G' must contain β or γ since otherwise t_1, t_2, t_3 cannot be produced with only two additional rules. In addition, the structure of the terms t_1, t_2, t_3 implies that a rule of the form $\tau \to \mathsf{r}_{gST}(\beta, \cdot)$ or $\tau \to \mathsf{r}_{gST}(\gamma, \cdot)$ must occur in G'. Since t_3 equals $\mathsf{r}_{gST}(1, \cdot)$, the rule $\gamma_{end} \to 1$ has to be contained in G'. But now the only possible way to produce t_1, t_2, t_3 by adding a single additional rule is to add the rule $\gamma \to \gamma \cdot \mathsf{s}\nu$.

By similar arguments, it can be seen that all rules of the form $\tau \to \cdots$ in G' are also contained in G and vice versa. Especially, we have to argue with $HS(2)$ and $HS(3)$ since otherwise the use of the variable α would not be necessary. □

From the correctness of FindGram and the previous lemma, we deduce that the output of FindGram applied to $HS(2), HS(3)$ is the grammar G defined above.

6. Computing an induction formula

In this section we will invert the left down arrow of Diagram 2. Note that the arrow inversion process that the algorithm IndProof executes starts with Herbrand sequents of a given Σ_1 sequent Θ. FindGram produces a schematic grammar G which will be assumed to be a schematic grammar of $\mathbb{N}$ for Θ, as we explained on p. 685. The algorithm FindFml detailed in this section will make use of this assumption and the sequent Θ which we call sequent of G in the following. Accordingly, in this section we assume that each schematic grammar has its sequent which will often not be mentioned explicitly if it is clear from the context.

Note that a schematic grammar G gives us much information about the conclusions of π_b, π_s, π_c of the simple induction proofs π having G as grammar. In particular, the instances of the left side formulas, the step terms, and the cut terms are fixed by G. The right side of the conclusion of π is fixed by the right side of the sequent of G. Using this, we can produce a schematic form of the simple induction proof S having as only unknown the quantifier-free part of the induction formula. Then, candidates for induction formulas are generated systematically and checked. If a correct candidate C is found, a simple induction proof with induction formula C is built completing S. This proof is returned.

In this section we will define two algorithms: $\mathsf{FindFml}^{\mathsf{c}}$ and $\mathsf{FindFml}^{\mathsf{h}}$. $\mathsf{FindFml}^{\mathsf{c}}$ is complete in the sense that if S can be completed to a simple induction proof, $\mathsf{FindFml}^{c}$ will find the missing induction formula, and return the completed simple induction proof. However, $\mathsf{FindFml}^{\mathsf{c}}$ is not feasible. $\mathsf{FindFml}^{\mathsf{h}}$ is a heuristic which checks a limited amount of systematically produced induction formulas but is much more efficient and succeeds to find induction formulas in typical examples. Note that it is undecidable whether a schematic s.i.p. can be completed to a simple induction proof, see [13]. Therefore, if $\mathsf{FindFml}^{\mathsf{c}}$ is given a schematic s.i.p. which cannot be completed to an s.i.p. it will not terminate.

Remember that FindFml produces from an input schematic grammar a schematic s.i.p. S. Then FindFml searches for induction formulas completing S to a simple induction proof. In the first subsection, we will define schematic s.i.p.s in a rigorous way. In the second subsection, we discuss their solvability, which is the possibility of completing the scheme to a simple induction proof. In the rest of the section, we precisely define $\mathsf{FindFml}^{\mathsf{c}}$ and $\mathsf{FindFml}^{\mathsf{h}}$.

6.1. Schematic simple induction proofs

We will see that each schematic grammar induces a schematic s.i.p. defined as follows.

Definition 6.1 *(Schematic s.i.p.).* Let $\alpha, \beta, \nu, \gamma$ be variables only occurring where indicated. Let $\Gamma_0[\alpha, \beta]$, $\Gamma_1[\alpha, \nu, \gamma], \Gamma_2[\alpha]$ only contain quantifier-free formulas. Let $B[\alpha]$ be a quantifier-free formula. Let $t_i[\alpha, \nu, \gamma]$ for $1 \leq i \leq n$, and $u_i[\alpha]$ for $1 \leq i \leq m$ be terms. Assume $n, m \geq 1$. Let X be a ternary predicate variable. Then, the list of the following three sequents is a schematic s.i.p. where it is assumed that formulas not containing any of β, ν, γ either occur in all or no left side formulas:

- $\Gamma_0[\alpha, \beta] \Rightarrow X[\alpha, 0, \beta]$
- $\Gamma_1[\alpha, \nu, \gamma], \bigwedge_{1 \leq i \leq n} X[\alpha, \nu, t_i[\alpha, \nu, \gamma]] \Rightarrow X[\alpha, \mathsf{s}\nu, \gamma]$
- $\Gamma_2[\alpha], \bigwedge_{1 \leq i \leq m} X[\alpha, \alpha, u_i[\alpha]] \Rightarrow B[\alpha]$

Note that a schematic s.i.p. is determined completely by the side formulas $\Gamma_0, \Gamma_1, \Gamma_2$, B, the so-called step-terms t_i for $1 \leq i \leq n$, and the so-called cut-terms u_i for $1 \leq i \leq m$.

There is a natural way to obtain a schematic s.i.p. from a schematic grammar.

Definition 6.2 *(From a schematic grammar to a schematic s.i.p.).* Let G be a schematic grammar and Θ its sequent. Then, its schematic s.i.p. S has the following side formulas $\Gamma_0, \Gamma_1, \Gamma_2, B$, induction step terms, and cut terms. We assume that $\alpha, \beta, \nu, \gamma$ are displayed whenever they occur.

- For each rule of the form $\tau \to \mathsf{r}_{\forall \bar{x} F[\bar{x}]}(\bar{t}[\alpha, \beta])$, we add the side formula $F[\bar{t}[\alpha, \beta]]$ to Γ_0.
- For each rule of the form $\tau \to \mathsf{r}_{\forall \bar{x} F[\bar{x}]}(\bar{t}[\alpha, \nu, \gamma])$, we add the side formula $F[\bar{t}[\alpha, \nu, \gamma]]$ to Γ_1.
- For each rule of the form $\tau \to \mathsf{r}_{\forall \bar{x} F[\bar{x}]}(\bar{t}[\alpha])$, we add the side formula $F[\bar{t}[\alpha]]$ to $\Gamma_0, \Gamma_1, \Gamma_2$.
- B is the right side of Θ.
- For each rule of the form $\gamma \to t[\alpha, \nu, \gamma]$ we include the term t into the set of step terms.
- For each rule of the form $\gamma_{end} \to t[\alpha]$ we include the term t into the set of cut terms.
- If the grammar G does not contain a rule of the form $\gamma \to t$ we include the term 0 into the set of step terms. If the grammar G does not contain a rule of the form $\gamma_{end} \to t$ we include the term 0 into the set of cut terms.

Note that for rules of the form $\tau \to c$ for c not containing any of β, ν, γ, corresponding side formulas are added to Γ_0, Γ_1 and Γ_2. The addition of 0 to the step-, respectively cut-terms guarantees that the schematic s.i.p. contains at least one step- and at least one cut-term. This gives them the right shape to be completable to simple induction proofs by $\mathsf{FindFml}$ which typically will not contain occurrences of z in their induction formula $F[x, y, z]$. Schematic grammars without rules of the form $\gamma \to t$ and $\gamma_{end} \to t$ are relevant since they correspond to quantifier-free induction formulas.

Definition 6.3 *(Solution of schematic s.i.p.).* Assume that a schematic s.i.p. S is given. Assume that there is a quantifier-free formula $F[x, y, z]$ such that the three sequents of S with X replaced by F are quasi-tautological. Then, we call F a solution of S.

Notation 6.4. Let G be a schematic grammar and let S be its associated schematic s.i.p. given in Definition 6.1. Then each formula F occurring in S is an instance of a formula occurring as a subindex of a constant of G. We call this formula $\forall F$. For $0 \leq i \leq 2$, assume $\Gamma_i := A_1, \ldots, A_n$. Then we denote $\forall A_1, \ldots, \forall A_n$ by $\forall \Gamma_i$.

Lemma 6.5. *Let S be a schematic s.i.p. consisting of sequents $S_1[X], S_2[X], S_3[X]$. Let F be a solution of S. Then, there is an s.i.p. π of*

$$\forall\Gamma_0, \forall\Gamma_1, \forall\Gamma_2 \Rightarrow B[\alpha]$$

such that the conclusion of π_b is $S_1[X\backslash F]$, the conclusion of π_s is $S_2[X\backslash F]$, and that the conclusion of π_c is $S_3[X\backslash F]$. We arbitrarily fix one of these proofs and denote it by $S[X\backslash F]$.

Proof. Since the premises of S with X replaced by F are quasi-tautological, they have cut-free proofs. Next, we introduce universal quantifiers which yields the required induction premises and the cut-premise. Finally, by applying induction and executing a cut, we obtain the searched simple induction proof. □

Lemma 6.6. *Let G be a schematic grammar containing rules of the form $\gamma \to \cdots$ and $\gamma_{end} \to \cdots$. Let S be its schematic s.i.p. Assume that F is a solution of S. Then G is the schematic grammar of the simple induction proof $S[X\backslash F]$.*

Proof. By the construction of Lemma 6.5 the s.i.p. $S[X\backslash F]$ consists of $\pi_b\colon S_1[X\backslash F]$, $\pi_s\colon S_1[X\backslash F]$, and $\pi_c\colon S_3[X\backslash F]$. These three sequents are the instances of the schematic s.i.p. $S_1[X], S_2[X], S_3[X]$ of G which in turn determine G. □

The previous lemma reduces the problem of finding a simple induction proof yielding a given schematic grammar G, which is the inverse of the left down arrow in Diagram 2, to solving a schematic s.i.p.

6.2. Solvability of schematic simple induction proofs

Not every schematic s.i.p. S is solvable, which follows immediately from the fact that the formula B in S is not restricted in any way. However, for our paper only the case is relevant where S is produced from a schematic grammar G whose instance grammars G_n produce Herbrand sequents of its sequent for all $n \in \mathbb{N}$ because this is a necessary condition for the solvability of S. However, as shown in Proposition 6.7 below, it is not a sufficient condition. The below Proposition 6.7 together with Proposition 3.13 is best thought of as an elaboration of Proposition 3.11 as follows: while Proposition 3.11 shows that, in general, the path in Diagram 2 from π to $\mathcal{L}(\mathcal{G}(\pi_n))$ cannot be inverted, Proposition 3.13 gives one reason for that: un-invertibility of the path from $\mathcal{G}(\pi)$ to $\mathcal{L}(\mathcal{G}(\pi_n))$ while Proposition 6.7 will give another reason: un-invertibility of the arrow from π to $\mathcal{G}(\pi)$.

In [13] we have shown that it is undecidable for an input schematic s.i.p. whether it is solvable even if it is produced from a grammar satisfying the condition stated in Proposition 6.7 of this paper. This result immediately implies Proposition 6.7. Here we give a simpler direct proof of Proposition 6.7.

Proposition 6.7. *There is a Σ_1 sequent $\forall\Gamma \Rightarrow B[\alpha]$ with B quantifier-free and a schematic grammar G with sequent $\forall\Gamma \Rightarrow B[\alpha]$ such that*

- *$\mathcal{L}(G_n)$ is a Herbrand sequent of $\forall\Gamma \Rightarrow B[\overline{n}]$ for all $n \in \mathbb{N}$, but*
- *the schematic s.i.p. of G does not have a solution.*

Proof. Consider the following schematic grammar G.

- $\tau \to \mathsf{r}_{P(0)}$
- $\tau \to \mathsf{r}_{\forall x(P(x)\to P(\mathsf{s}x))}(\gamma)$

- $\gamma \to \mathsf{s}\gamma$
- $\gamma_{end} \to 0$

It is easy to see that for each $n \in \mathbb{N}$ its instance grammar G_n produces the Herbrand sequent

$$P(0), P(0) \to P(\bar{1}), \dots, P(\overline{n-1}) \to P(\bar{n}) \Rightarrow P(\bar{n})$$

of

$$P(0), \forall x\big(P(x) \to P(\mathsf{s}x)\big) \Rightarrow P(\bar{n}).$$

Its schematic s.i.p. S is given as follows.

- $P(0) \Rightarrow X[\alpha, 0, \beta]$
- $P(0), P(\gamma) \to P(\mathsf{s}\gamma), X[\alpha, \nu, \mathsf{s}\gamma] \Rightarrow X[\alpha, \mathsf{s}\nu, \gamma]$
- $P(0), X[\alpha, \alpha, 0] \Rightarrow P(\alpha)$

In the following, we prove that S is not solvable. Towards a contradiction, assume that a solution $F[x, y, z]$ of S exists which means that the premises of S with X replaced by F are quasi-tautological.

We substitute ν and γ successively as follows in the second sequent of $S[X\backslash F]$ for $n \in \mathbb{N}$ where we write $\mathsf{s}^n(\cdot)$ for $\underbrace{\mathsf{s}\cdots\mathsf{s}}_{n \text{ times}}(\cdot)$ for $n \in \mathbb{N}$:

$$\big[\nu\backslash\nu, \gamma\backslash\mathsf{s}^{n-1}\gamma\big], \big[\nu\backslash\mathsf{s}\nu, \gamma\backslash\mathsf{s}^{n-2}\gamma\big], \dots, \big[\nu\backslash\mathsf{s}^{n-1}\nu, \gamma\backslash\gamma\big]$$

From the quasi-tautologies resulting by the substitutions, we easily derive the following sequent $\Theta(n)$ for any $n \in \mathbb{N}$.

$$P(0), P(\gamma) \to P(\mathsf{s}\gamma), P(\mathsf{s}\gamma) \to P(\mathsf{ss}\gamma), \dots, P\big(\mathsf{s}^{n-1}\gamma\big) \to P\big(\mathsf{s}^n\gamma\big), F\big[\alpha, \nu, \mathsf{s}^n\gamma\big] \Rightarrow F\big[\alpha, \mathsf{s}^n\nu, \gamma\big]$$

For the further argumentation, we will w.l.o.g. assume that $F[x, y, z]$ is given as conjunction of the formulas

$$EQ_1 \to F_1, \cdots, EQ_n \to F_n$$

where EQ_i is a conjunction only containing (possibly negated) atoms of the form $s = t$, and F_i is a disjunction only containing (possibly negated) atoms of the form $P(\cdot)$.

From now on, we argue in the models of signature $\{0, \mathsf{s}, P\}$ with universe $\mathbb{N}$ and zero and the successor interpreted as usual. We say that a formula F is true in $\mathcal{M}$ if it holds in all of these models. Note that the models only diverge in their interpretation of the predicate P.

Let us analyse the structure of the true equations in $\mathcal{M}$. First, note that all terms of $\mathcal{M}$ have the form $\mathsf{s}^n(u)$ for $u = 0$ or u being a variable. Equations of the form $\mathsf{s}^n(\bar{p}) = \mathsf{s}^m(\bar{q})$ for $n < m$ are true exactly if $\bar{p} = \mathsf{s}^{m-n}(\bar{q})$. This implies the following property:

(A): For any $d \in \mathbb{N}$, and terms $\mathsf{s}^n(x), \mathsf{s}^m(x)$ with $n, m \in \mathbb{N}$ and a variable x, we have $\mathsf{s}^n(\bar{k}) = \mathsf{s}^m(\bar{\ell})$ exactly if $\mathsf{s}^n(\overline{k+d}) = \mathsf{s}^m(\overline{\ell+d})$.

Let $m \in \mathbb{N}$ be an upper bound for the depth of terms occurring in F. We will prove the following auxiliary lemma.

Lemma 6.8. *For all $q_1, q_2, n \in \mathbb{N}$ with $q_1, q_2 > 2n$ and $n > m$ the following properties hold.*

- *$EQ_i[\overline{q_1}, \overline{q_1}, 0] \leftrightarrow EQ_i[\overline{q_2}, \overline{q_2}, 0]$ is true in $\mathcal{M}$ for all $1 \le i \le n$.*
- *$EQ_i[\overline{q_1}, \overline{q_1 - n}, \overline{n}] \leftrightarrow EQ_i[\overline{q_2}, \overline{q_1 - n}, \overline{n}]$ is true in $\mathcal{M}$ for all $1 \le i \le n$.*

Proof. Assume $k \in \mathbb{N}$. For $i = 1, 2$, no equations of the form $\mathsf{s}^k(q_i) = c$ are true in $\mathcal{M}$ where c is a closed term of $F[x, y, z]$ since the depth of c is bounded by n. Assume $k, \ell \in \mathbb{N}$. For equations of the form $\mathsf{s}^k(u) = \mathsf{s}^\ell(v)$ of $F[x, y, z]$ where $u = x, y$ and $v = x, y$ $\mathsf{s}^k(\overline{q_1}) = \mathsf{s}^\ell(\overline{q_1})$ is true in $\mathcal{M}$ exactly if $\mathsf{s}^k(\overline{q_2}) = \mathsf{s}^\ell(\overline{q_2})$ is true in $\mathcal{M}$ because of property (A). For equations of the form $\mathsf{s}^k(u) = \mathsf{s}^\ell(z)$ of $F[x, y, z]$ where $u = x, y$ the equation $\mathsf{s}^k(\overline{q_i}) = \mathsf{s}^\ell(0)$ is not true in $\mathcal{M}$ since the depth of the term at the right side is bounded by n. In total this implies the first property. For the second property, we argue similarly. □

We choose arbitrary $q_1, q_2, n \in \mathbb{N}$ with $q_2 > q_1 > 2n$, and $n > m$, and consider $\Theta(n)[\alpha \backslash \overline{q_1}][\nu \backslash \overline{q_1 - n}][\gamma \backslash 0]$. Both, $F[\overline{q_1}, \overline{q_1 - n}, \overline{n}]$ and $F[\overline{q_1}, \overline{q_1}, 0]$ can be replaced by conjunctions of the form $F_{i_1} \wedge \cdots \wedge F_{i_k}$ which yields the formula Θ', true in $\mathcal{M}$ only containing atoms of the form $P(\cdot)$. We replace all occurrences of $\overline{N}$ in Θ' with $q_1 \leq N$ by $\overline{N + q_2 - q_1}$, and obtain a formula which is still true in $\mathcal{M}$ since the models evaluate P on different numerals independently. Note that on the left side of Θ' exactly atoms are replaced which have been produced by replacing α. On the right side of Θ' exactly atoms are replaced which have been produced by replacing α or ν. Because of the special choice of q_1, q_2, n, and the previous lemma we deduce that the formula Θ'' given as

$$P(0), P(0) \to P(\overline{1}), P(\overline{1}) \to P(\overline{2}), \ldots, P(\overline{n-1}) \to P(\overline{n}), F[\overline{q_2}, \overline{q_1 - n}, \overline{n}] \Rightarrow F[\overline{q_2}, \overline{q_2}, 0]$$

is true in $\mathcal{M}$. For $n, m \in \mathbb{N}$, we abbreviate

$$P(\overline{n}) \to P(\overline{n+1}), P(\overline{n+1}) \to P(\overline{n+2}), \ldots, P(\overline{m-1}) \to P(\overline{m})$$

as $propag_P(n, m-1)$. A similar unfolding of the induction step premise as we used before to obtain Θ yields

$$P(0), propag_P(n, q_1 - 1) \Rightarrow F[\overline{q_2}, \overline{q_1 - n}, \overline{n}].$$

Together with Θ'' we deduce

$$P(0), propag_P(0, q_1 - 1) \Rightarrow F[\overline{q_2}, \overline{q_2}, 0]$$

which finally yields using the cut premise

$$P(0), propag_P(0, q_1 - 1) \Rightarrow P(\overline{q_2})$$

which is clearly not true in $\mathcal{M}$. This is a contradiction which rejects the assumption that the schematic s.i.p. S obtained from G can be solved. □

Let us conclude, however, with the following positive result.

Lemma 6.9. *Let G be a schematic grammar such that there is a simple induction proof π with $\mathcal{G}(\pi) = G$. Then the schematic s.i.p. of G has a solution.*

Proof. Clearly, the non-quantified induction formula of π is a solution of the schematic s.i.p. of G. □

6.3. The complete algorithm $\mathsf{FindFml}^{\mathsf{c}}$

In this section we will define the algorithm $\mathsf{FindFml}^{\mathsf{c}}$ which is a complete algorithm in the sense that it succeeds in solving each schematic s.i.p. which has a solution.

Definition 6.10. Let S be a schematic s.i.p. with premises given as in Definition 6.1. Then, the collection of formulas $C_{S,q}[x,z]_{q\geq 0}$ is defined by recursion as follows.

$$C_{S,0}[x,z] := \bigwedge \Gamma_0[x,z]$$

$$C_{S,q+1}[x,z] := \bigwedge \Gamma_1[x,\bar{q},z] \wedge \bigwedge_{1\leq i\leq n} C_{S,q}\big[x,t_i[x,\bar{q},z]\big]$$

$C_{S,q}[x,z]$ is called the q-th canonical solution of S.

Remark 6.11. Note that the canonical solutions of this paper are similar to the canonical solutions of [21, Definition 8]. Both can be seen as being solutions for a sequence of predicate variables in a proof containing iterated Π_1 cuts. In the case of this paper, this proof is obtained by unfolding induction by iterated Π_1 cuts. Nevertheless, some important differences occur. The form of the proof with iterated Π_1 cuts is much more restricted in our case. This has of course also an effect on the variable conditions of the mentioned proof, and causes the q-th canonical solution to contain at most two variables for all $q \in \mathbb{N}$. A similar restriction cannot be made for the canonical solutions in [21]. In addition, they cannot be defined in such a uniform way as the canonical solutions of this paper.

Lemma 6.12. *Let S be a schematic s.i.p. Then, for any solution F of S and any $q \in \mathbb{N}$, the q-th canonical solution $C_{S,q}[x,z]$ logically implies $F[x,\bar{q},z]$.*

Proof. The lemma is proved by induction on q. It holds for $q=0$ since we have $\Gamma_0[x,z] \Rightarrow F[x,0,z]$ for the solution F of S. Let us prove the claim for $q+1$. We have

$$\Gamma_1[x,\bar{q},z], \bigwedge_{1\leq i\leq n} F\big[x,\bar{q},t_i[x,\bar{q},z]\big] \Rightarrow F[x,\overline{q+1},z]$$

since F is a solution of S. By induction hypothesis, we have that $C_{S,q}[x,t_i[x,\bar{q},z]]$ implies $F[x,\bar{q},t_i[x,\bar{q},z]]$ for each $1 \leq i \leq n$. Then the claim for $q+1$ immediately follows from the definition of $C_{S,q+1}$. □

The previous lemma justifies the following definition of $\mathsf{FindFml}^{\mathsf{c}}$.

Algorithm 6. $\mathsf{FindFml}^{\mathsf{c}}$

Input: Schematic grammar G with sequent Θ.
Output: The algorithm is partial. If an output is given, it is a simple induction proof of Θ.
Execution:

- Construct the schematic s.i.p. S of G.
- Calculate $C_{S,0}[x,z]$.
- For all logical consequences $C'[x,z]$ of $C_{S,0}[x,z]$ having the same signature as $C_{S,0}[x,z]$ plus possibly equations:
  - For each subset Q of occurrences of 0 in $C'[x,z]$:
    - Replace all occurrences in Q by y, yielding a formula $\tilde{C}[x,y,z]$. If $\tilde{C}[x,y,z]$ is a solution of S (which is check-able since S is quantifier-free), break the for-loops.
- Find proofs π_b, π_s, and π_c of the quasi-tautologies of $S[X\backslash\tilde{C}]$ and complete them in the obvious way to a simple induction proof of Θ.

end

The proofs π_b, π_s, and π_c of the quasi-tautologies in $S[X\backslash\tilde{C}]$ might be large. Therefore, for implementations of the algorithm, we will content ourselves with guarantees of quasi-tautology checkers that they are in fact quasi-tautologies.

Theorem 6.13. *Let G be a schematic grammar such that there is an s.i.p. π with $\mathcal{G}(\pi) = G$. Then* $\mathsf{FindFml}^{\mathsf{c}}(G)$ *is a simple induction proof and if G is non-degenerate then $\mathcal{G}(\mathsf{FindFml}^{\mathsf{c}}(G)) = G$.*

Proof. Let π be an s.i.p. and $G = \mathcal{G}(\pi)$. Then by Lemma 6.9 the schematic s.i.p. S of G has a solution, say F. By Lemma 6.12 we know that $C_{S,0}[x, z]$ logically implies $F[x, 0, z]$. As $\mathsf{FindFml}^{\mathsf{c}}$ enumerates all logical consequences of $C_{S,0}[x, z]$ and, in each of them, abstracts all subsets of occurrences of 0 by y it eventually finds $\tilde{C}[x, y, z] = F[x, y, z]$ which is a solution and from which, by Lemma 6.5, an s.i.p. π' is obtained. Since $S[X\backslash F]$ and hence $\mathsf{FindFml}^{\mathsf{c}}(G)$ induce the same schematic grammar as S via Definition 3.5 we have $\mathcal{G}(\mathsf{FindFml}^{\mathsf{c}}(G)) = G$. □

Note that this is a completeness result in the sense that an s.i.p. can be found once its schematic grammar has been found. Referring to Diagram 2 this means that there is a partial computable function inverting the left down arrow. However, note that because of Proposition 6.7 $\mathsf{FindFml}^{\mathsf{c}}$ does not terminate on some input schematic grammars even if their instance grammars all produce Herbrand sequents.

6.4. The heuristic algorithm $\mathsf{FindFml}^{\mathsf{h}}$

The algorithm $\mathsf{FindFml}^{\mathsf{c}}$ defined in the previous section, although complete, is quite inefficient. Therefore, in this section we define a heuristic algorithm $\mathsf{FindFml}^{\mathsf{h}}$ which is more efficient but still succeeds in typical examples. The algorithm $\mathsf{FindFml}^{\mathsf{h}}$ uses a similar strategy as the previously presented $\mathsf{FindFml}^{\mathsf{c}}$. Nevertheless, it is much more efficient because instead of producing all logical consequences of canonical solutions it only produces all consequences obtained by applying forgetful resolution and forgetful paramodulation. This heuristic is a generalisation of a similar approach used in the context of cut-introduction [22] that has proved useful in large-scale experiments, see [20].

Definition 6.14. Propositional resolution is the following inference rules on clauses:

$$\frac{C \vee p \qquad C \vee \neg p}{C \vee D}$$

Propositional paramodulation is the following inference rule on clauses

$$\frac{C \vee t_0 = t_1 \qquad D[x\backslash t_i]}{C \vee D[x\backslash t_j]}$$

for $0 \leq i, j \leq 1$ and $i \neq j$.

Let C_1, C_2 be two clauses, then we denote the set of clauses that can be obtained from C_1, C_2 by resolution by $res(C_1, C_2)$ and the set of clauses that can be obtained from C_1, C_2 by paramodulation by $para(C_1, C_2)$. Letting F be a formula with conjunctive normal form (CNF) $\{C_i\}_{i\in I}$ we define

$$\mathcal{F}(F) = \left\{ C \wedge \bigwedge_{i\in I\backslash\{j,k\}} C_i \;\middle|\; C \in res(C_j, C_k) \cup para(C_j, C_k) \right\}.$$

We can now present an auxiliary algorithm $\mathsf{FindConseq}$. Using the operation $\mathcal{F}$, it produces consequences of a canonical solution $C_{S,n}$, then recurses upon those consequences which pass a certain quasi-tautology

check, finally returning a set of formulas from which candidates for induction formulas will be produced. FindConseq depends on a schematic s.i.p. S given as in Definition 6.1 and a natural number n. The intended input of $\mathsf{FindConseq}_{S,n}$ is the canonical solution $C_{S,n}$ in CNF.

Algorithm 7. $\mathsf{FindConseq}_{S,n}$

Input: Formula A in CNF.
Output: A set of logical consequences of A.

function $\mathsf{FindConseq}_{S,n}(A$: formula in CNF)
  $M \leftarrow \{A\}$
  for $F \in \mathcal{F}(A)$ **do**
    if $\Gamma_2[\overline{n}], \bigwedge_{1\leq i\leq m} F[\overline{n}, u_i[\overline{n}]] \Rightarrow B[\overline{n}]$ is a quasi-tautology **then**
      $M \leftarrow M \cup \mathsf{FindConseq}_{S,n}(F)$
    end if
  end for
  return M
end function

end

For the intended input, the quasi-tautology test sorts out logical consequences C of $C_{S,n}$ which are too weak to make the third sequent of S tautological. No further logical consequences of C have to be produced in this case.

Note that in general FindConseq is not feasible since it uses a forking recursion. Nevertheless, many branches of the recursion might not be treated because of the quasi-tautology criterion. Large scale experiments in [20] suggest good running times for a very similar algorithm defined in [20].

Let us define $\mathsf{FindFml}^{\mathsf{h}}$ in detail.

Algorithm 8. $\mathsf{FindFml}^{\mathsf{h}}$

Input: Schematic grammar G with sequent Θ, $n \in \mathbb{N}$.
Output: Simple induction proof π of Θ, or error output.
Execution:

- Produce the schematic s.i.p. S of G.
- Calculate $C_{S,n}$.
- Calculate $\mathsf{FindConseq}_{S,n}(C_{S,n})$.
- Starting from the formulas in $\mathsf{FindConseq}(C_{S,n})$ consisting only of a single clause progressing to the ones consisting of several clauses, a certain (user-defined) number of candidate induction formulas are produced by inserting a second variable y for subsets of occurrences[4] of $\overline{n}$. It is checked for each candidate induction formula $\tilde{C}$ whether it solves S using an efficient quasi-tautology checker.
  - If this is the case for a candidate $\tilde{C}$ find proofs π_b, π_s, and π_c of the quasi-tautologies of $S[X\backslash\tilde{C}]$. Output the corresponding simple induction proof, and break the algorithm.
  - If this is not the case for all of the produced candidates, we continue with the next formula in $\mathsf{FindConseq}_{S,n}(C_{S,n})$.
- Output error.

end

[4] There are systematic ways of reducing the number of subsets of occurrences of $\overline{n}$ for which it makes sense to replace them by y. Put informally, we recommend only to replace occurrences of $\overline{n}$ by y if they are obtained by replacing occurrences of ν of the schematic s.i.p. when calculating $C_{S,n}$.

As for $\mathsf{FindFml}^{\mathsf{c}}$, the equation $\mathcal{G}(\pi) = G$ does not hold for a degenerate grammar. Again, we will content ourselves with guarantees of quasi-tautology checkers that the sequents of $S[X\backslash\tilde{C}]$ are in fact quasi-tautologies instead of asking for proofs of them.

7. Examples

We test the algorithm $\mathsf{IndProof}^{\mathsf{h}}$ on two typical examples from the literature. Remember that $\mathsf{IndProof}^{\mathsf{h}}$ uses the version $\mathsf{FindFml}^{\mathsf{h}}$ of $\mathsf{FindFml}$. $\mathsf{IndProof}^{\mathsf{h}}$ will succeed to find simple induction proofs for the associativity of addition and the equivalence of the two definitions of the factorial by head- and tail-recursion.

7.1. Example: two definitions of the factorial

Recall the two definitions of the factorial used in Example 3.2. To test the performance of $\mathsf{IndProof}^{\mathsf{h}}$ on the Herbrand sequents displayed there, we only have to apply $\mathsf{FindFml}^{\mathsf{h}}$ to the output G of $\mathsf{FindGram}$ described in Example 5.31. We will execute $\mathsf{FindFml}^{\mathsf{h}}$ on input $G, 0$. First, $\mathsf{FindFml}^{\mathsf{h}}$ produces the schematic s.i.p. S from G which is given as follows:

- $f(0) = 1, g(\beta, 0) = \beta, \beta \cdot 1 = \beta \Rightarrow X[0, \beta]$
- $\mathsf{s}\nu \cdot f(\nu) = f(\mathsf{s}\nu), g(\gamma, \mathsf{s}\nu) = g(\gamma \cdot \mathsf{s}\nu, \nu), (\gamma \cdot \mathsf{s}\nu) \cdot f(\nu) = \gamma \cdot (\mathsf{s}\nu \cdot f(\nu)), X[\nu, \gamma \cdot \mathsf{s}\nu] \Rightarrow X[\mathsf{s}\nu, \gamma]$
- $1 \cdot f(\alpha) = f(\alpha), X[\alpha, 1] \Rightarrow g(1, \alpha) = f(\alpha)$

$C_{S,0}[z]$ is given as

$$f(0) = 1 \wedge g(\beta, 0) = \beta \wedge \beta \cdot 1 = \beta.$$

$\mathsf{FindFml}^{\mathsf{h}}$ carries out paramodulations between $g(\beta, 0) = \beta$ and $\beta = \beta \cdot 1$. There are 5 resulting formulas. 4 of them do not pass the quasi-tautology test. The formula which passes is

$$g(\beta, 0) = \beta \cdot 1 \wedge 1 = f(0).$$

Between $f(0) = 1$ and $g(\beta, 0) = \beta$ paramodulation cannot be applied. Paramodulation between $f(0) = 1$ and $\beta \cdot 1 = \beta$ delivers as unique resulting formula

$$g(\beta, 0) = \beta \wedge \beta \cdot f(0) = \beta.$$

The only possible further paramodulation delivers $g(\beta, 0) = \beta \cdot f(0)$ as unique formula passing the quasi-tautology test. This is the only produced formula containing a single clause.

Next, $\mathsf{FindFml}^{\mathsf{h}}$ substitutes subsets of occurrences of 0 by y yielding 4 candidates for induction formulas. For the candidate $C[y, z] := g(z, y) = z \cdot f(y)$ it can be easily checked (by a quasi-tautology check) that it is a solution of S. No other candidate is a solution of S. $\mathsf{FindFml}^{\mathsf{h}}$ therefore outputs the natural simple induction proof for the equivalence of the two definitions of the factorial given in Example 3.2.

It can be seen easily that $\mathsf{FindFml}^{\mathsf{h}}$ yields the same output independent of its second input, so it is not necessary to fix it as zero as we did before. Let us remark that this section together with Example 5.31 shows that $\mathsf{IndProof}^{\mathsf{h}}$ applied to only two of the natural Herbrand sequents of $g(\overline{1}, \overline{n}) = f(\overline{n})$ from Example 5.31 yields a simple induction proof of the universal statement.

7.2. Example: associativity of addition

Let us sketch how our algorithm deals with the following special case of associativity of addition given as

$$(\forall x)\big(x + (x + x) = (x + x) + x\big).$$

This formula itself is not inductive but a slight generalisation of it is. We write $\mathsf{s}^n(\cdot)$ for $\underbrace{\mathsf{s}\cdots\mathsf{s}}_{n \text{ times}}(\cdot)$ for $n \in \mathbb{N}$. Let us describe a natural proof strategy for

$$\overline{n} + (\overline{n} + \overline{n}) = (\overline{n} + \overline{n}) + \overline{n}$$

for $n \in \mathbb{N}$ to fix the input Herbrand sequents of $\mathsf{IndProof}$:

Starting with $\overline{n} + (\overline{n} + \overline{n})$, we repeatedly decrease the first summand which yields

$$\mathsf{s}\big(\overline{n-1} + (\overline{n} + \overline{n})\big), \mathsf{ss}\big(\overline{n-2} + (\overline{n} + \overline{n})\big), \ldots, \mathsf{s}^n\big(0 + (\overline{n} + \overline{n})\big).$$

We obtain $\mathsf{s}^n(\overline{n} + \overline{n})$, and from this $\mathsf{s}^n((0 + \overline{n}) + \overline{n})$.

We repeatedly shift the s in front of the first summand which yields

- $\mathsf{s}^{n-1}(\mathsf{s}(0 + \overline{n}) + \overline{n})$, $\mathsf{s}^{n-1}((\overline{1} + \overline{n}) + \overline{n})$
- $\mathsf{s}^{n-2}(\mathsf{s}(\overline{1} + \overline{n}) + \overline{n})$, $\mathsf{s}^{n-2}((\overline{2} + \overline{n}) + \overline{n})$
- $\ldots$
- $\mathsf{s}(\overline{n-1} + \overline{n}) + \overline{n}$, $(\overline{n} + \overline{n}) + \overline{n}$

This concludes the proof sketch for the single instances. As for the factorial this uniform strategy fixes a collection of Herbrand sequents $HS(n)$ of

$$\Gamma \Rightarrow \overline{n} + (\overline{n} + \overline{n}) = (\overline{n} + \overline{n}) + \overline{n}$$

where Γ contains the universal closures of the following formulas.

$$\begin{array}{ll} \mathsf{s}(x + y) = \mathsf{s}x + y & \mathsf{s}{-}Ax, \\ 0 + y = y & 0{-}Ax \end{array}$$

The following instances of $\mathsf{s}{-}Ax$ are used for the sketched proof of $\overline{n} + (\overline{n} + \overline{n}) = (\overline{n} + \overline{n}) + \overline{n}$ for $n \in \mathbb{N}$:

- $\langle \overline{n-1}, \overline{n} + \overline{n}\rangle, \langle \overline{n-2}, \overline{n} + \overline{n}\rangle, \ldots, \langle 0, \overline{n} + \overline{n}\rangle$
- $\langle \overline{n-1} + \overline{n}, \overline{n}\rangle, \langle \overline{n-2} + \overline{n}, \overline{n}\rangle, \ldots, \langle 0 + \overline{n}, \overline{n}\rangle$
- $\langle \overline{n-1}, \overline{n}\rangle, \langle \overline{n-2}, \overline{n}\rangle, \ldots, \langle 0, \overline{n}\rangle$

The used instances of $0{-}Ax$ are $\overline{n} + \overline{n}$ and $\overline{n}$. The following schematic grammar G has the property that its instance grammars G_n produce $HS(n)$ for all $n \in \mathbb{N}$.

- $\tau \to \mathsf{r}_{\mathsf{s}-Ax}(\nu, \alpha + \alpha) \mid \mathsf{r}_{\mathsf{s}-Ax}(\nu + \alpha, \alpha) \mid \mathsf{r}_{\mathsf{s}-Ax}(\nu, \alpha)$
- $\tau \to \mathsf{r}_{0-Ax}(\alpha + \alpha) \mid \mathsf{r}_{0-Ax}(\alpha)$

Note that the schematic grammar is degenerate since it does not contain rules for β and γ. It can be seen easily that G is of minimal size among the schematic grammars G' with $HS(2) \subseteq \mathcal{L}(G'_2)$ and $HS(3) \subseteq \mathcal{L}(G'_3)$. Its schematic s.i.p. S is given as follows.

- $0 + (\alpha + \alpha) = \alpha + \alpha, 0 + \alpha = \alpha \Rightarrow X[\alpha, 0, \beta]$
- $0+(\alpha+\alpha) = \alpha+\alpha, 0+\alpha = \alpha, \mathsf{s}(\nu+(\alpha+\alpha)) = \mathsf{s}\nu+(\alpha+\alpha), \mathsf{s}((\nu+\alpha)+\alpha) = \mathsf{s}(\nu+\alpha)+\alpha, \mathsf{s}(\nu+\alpha) = \mathsf{s}\nu+\alpha, X[\alpha, \nu, 0] \Rightarrow X[\alpha, \mathsf{s}\nu, \gamma]$
- $0 + (\alpha + \alpha) = \alpha + \alpha, 0 + \alpha = \alpha, X[\alpha, \alpha, 0] \Rightarrow \alpha + (\alpha + \alpha) = (\alpha + \alpha) + \alpha$

By using paramodulation on the clauses of $C_{S,0}$ we obtain $0 + (\alpha + \alpha) = (0 + \alpha) + \alpha$. Inserting ν for 0 delivers the following correct candidate induction formula $\tilde{C}$ given as follows.

$$\tilde{C}[x, y, z] := y + (x + x) = (y + x) + x.$$

The example shows that $\mathsf{IndProof}^{\mathsf{h}}$ fully automatically finds a proof of a special case of associativity of addition from natural cut-free proofs of instances.

8. Evaluation

In this short section we discuss merits and drawbacks of our approach to inductive theorem proving.

8.1. Analyticity

What we view as its principal merit is that it breaks the tight relationship between conclusion and induction invariant which is typical for bottom-up proof search techniques. It constructs a schematic grammar from instance proofs using only methods from the combinatorics of finite sets of first-order terms. Once a suitable schematic grammar is found, it is – at least in principle – possible to find an inductive proof (cf. Theorem 6.13). Thus our approach attacks the central problem of inductive theorem proving – the inherent non-analytic nature of induction invariants – by reducing it to a problem in the combinatorics of finite sets of terms and a quantifier-free logical unification problem.

8.2. Completeness

Due to Proposition 3.11, a trivial corollary of Gödel's second incompleteness theorem, it was clear a priori that there are sequences of Herbrand sequents which do not correspond to simple induction proofs. Therefore, only restricted completeness statements make sense. Theorem 6.13 is a restricted completeness theorem which shows that we can compute (by $\mathsf{FindFml}^{\mathsf{c}}$) an s.i.p. from a schematic grammar G s.t. there is an s.i.p. π with $\mathcal{G}(\pi) = G$. It would be desirable to push this restricted completeness result further to the right of Diagram 2 by proving that it is possible to compute an s.i.p. from a finite subsequence $(H_i)_{i=0}^{n}$ of a sequence of Herbrand sequents $(H_i)_{i\geq 0}$ s.t. there is an s.i.p. π with $\mathcal{L}(\mathcal{G}(\pi_i)) \supseteq H_i$ for all $i \geq 0$. Our algorithm $\mathsf{FindGram}$ does not achieve this but we believe it is possible to devise an algorithm which does based on our approach by requiring a more sophisticated search goal than the minimisation of the number of production rules of the schematic grammar.[5]

8.3. Algorithmic complexity

The primary drawback of our approach is that is consists of a number of independent phases, each of which can fail independently. First, the instance proofs can be too irregular.[6] While this seems to be the most fundamental obstacle there are techniques to counter it, for example to work modulo a suitable

[5] Note that this problem has a trivial solution not requiring any input as the set of simple induction proofs is computably enumerable. The challenge is to find a complete and (reasonably) efficient algorithm.

[6] However, not finding instance proofs does not seem to be a danger, at least for the class of examples treated in the literature.

theory (see Section 8.5) which reduces the number of possible instance proofs or to compute more than one instance proof for each n. Secondly, even if the instance proofs are sufficiently regular, the schematic grammar algorithm can fail to produce a solvable grammar (but this seems avoidable, see Section 8.2). Thirdly, even if a solvable grammar is produced it may be infeasible to actually compute the solution. How important an obstacle this is in practice is hard to say without experimental data. In principle, the large body of existing work on the computation of quantifier-free loop invariants (see e.g. [5]) should be adaptable to this context.

8.4. Practical value

Another consequence of the above-mentioned algorithmic complexity is that the implementation requires a relatively large amount of work. However, for assessing the practical value of the presented algorithm its implementation and evaluation on a library of test problems is mandatory. This will be carried out in future research within the gapt system.[7] We are optimistic that $\mathsf{IndProof}^{\mathsf{h}}$ will yield satisfactory results because of the following reasons:

First, we have applied our algorithm to typical examples from the literature with pencil and paper (as described in Section 7) and the small sizes of the search spaces show that a reasonable implementation can solve them automatically.

Secondly, a similar algorithm has already been implemented in gapt and tested quite extensively with good results: in [20] an algorithm *CI* (cut-introduction) which contracts cut-free proofs by introducing Π_1 cuts achieved good results in compressing proofs from the TSTP-library (see [32]). The algorithm $\mathsf{IndProof}^{\mathsf{h}}$ presented in this paper uses very similar techniques as *CI* to guess induction formulas. As *CI*, IndProof decomposes a language, and then solves a similar quantifier-free logical unification problem. The fact that IndProof in contrast to *CI* deals with induction does not essentially increase the complexity of the algorithm but instead causes IndProof to be partial. The test results for *CI* are therefore good evidence that also IndProof should succeed in finding inductive proofs for a high percentage of natural test problems.

8.5. Extendability

This paper has only treated simple induction proofs but the overall approach is more general. Each proof shape (e.g. a nested induction, an induction on another algebraic data type, another induction rule, ...) induces a corresponding notion of schematic grammar.[8]

Independently, it is quite straightforward to extend this approach to work modulo equational theories: the only part which requires some non-trivial insights on the theoretical level is the generalisation of FindGram to compress equivalence classes of terms instead of literal terms. On the practical level, a reasonable implementation of FindFml must assume the existence of efficient solvers for quantifier-free formulas in that equational theory (which puts a certain limit on the theories for which such an extension will be practically useful).

9. Conclusion

The strategy of this paper is comparable to the strategy of [21]: Starting from Herbrand sequents, by inverting productions of grammars, one tries to obtain non-analytic proofs. The results of the paper show

[7] http://www.logic.at/gapt/.

[8] Note that already in this paper the presence/absence of γ-productions determines the presence/absence of the $\forall y$-quantifier in the induction formula. In the associativity example the algorithm determines automatically that this quantifier is not necessary by finding a schematic grammar without γ-productions.

that the correspondence between proofs and grammars which was exploited heavily in [21], can also be fruitfully used in the setting of induction.

Nevertheless, in the setting of induction, new and essential problems occur. Propositions 3.11, 3.13, and 6.7 imply that each algorithm inverting the arrows of Diagram 2 must be partial. This is in contrast to [21] where the reversal of the arrows in the corresponding diagram is always possible, even if the resulting non-analytic proof is not always shorter than the cut-free proof corresponding to the input Herbrand sequent.

Despite of the problems mentioned above, we managed to construct an algorithm IndProof able to automatically compute induction formulas in typical examples. Therefore, we believe that our paper makes a valuable contribution to the difficult field of inductive theorem proving.

Acknowledgements

The authors would like to thank a reviewer for many helpful suggestions which led to a considerable improvement of the legibility of this paper.

References

[1] Alessandro Armando, Michaël Rusinowitch, Sorin Stratulat, Incorporating decision procedures in implicit induction, J. Symbolic Comput. 34 (4) (2002) 241–258.
[2] Franz Baader, Tobias Nipkow, Term Rewriting and All That, Cambridge University Press, 1998.
[3] Siani Baker, Andrew Ireland, Alan Smaill, On the use of the constructive omega-rule within automated deduction, in: Andrei Voronkov (Ed.), Logic Programming and Automated Reasoning, LPAR, 1992, in: Lecture Notes in Computer Science, vol. 624, 1992, pp. 214–225.
[4] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Yunshan Zhu, Symbolic model checking without BDDs, in: Rance Cleaveland (Ed.), Tools and Algorithms for Construction and Analysis of Systems, TACAS, 1999, in: Lecture Notes in Computer Science, vol. 1579, Springer, 1999, pp. 193–207.
[5] Aaron R. Bradley, Zohar Manna, The Calculus of Computation, Springer, 2007.
[6] James Brotherston, Nikos Gorogiannis, Rasmus L. Petersen, A generic cyclic theorem prover, in: Proceedings of APLAS-10, in: Lecture Notes in Computer Science, Springer, 2012, pp. 350–367.
[7] James Brotherston, Alex Simpson, Sequent calculi for induction and infinite descent, J. Logic Comput. 21 (6) (2011) 1177–1216.
[8] Alan Bundy, The automation of proof by mathematical induction, in: Andrei Voronkov, John Alan Robinson (Eds.), Handbook of Automated Reasoning, vol. 1, Elsevier, 2001, pp. 845–911.
[9] Alan Bundy, David Basin, Dieter Hutter, Andrew Ireland, Rippling: Meta-Level Guidance for Mathematical Reasoning, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2005.
[10] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree automata techniques and applications, Available on: http://www.grappa.univ-lille3.fr/tata, 2007, release October, 12th 2007.
[11] Leonardo De Moura, Nikolaj Bjørner, Satisfiability modulo theories: introduction and applications, Commun. ACM 54 (9) (2011) 69–77.
[12] Sebastian Eberhard, Stefan Hetzl, Algorithmic compression of finite tree languages by rigid acyclic grammars, Preprint available at: http://www.logic.at/people/hetzl/research/zerlegung.pdf.
[13] Sebastian Eberhard, Stefan Hetzl, Undecidability results for simple induction proofs, Preprint available at: http://www.logic.at/people/hetzl/research/undecidability.pdf.
[14] Ferenc Gécseg, Magnus Steinby, Tree languages, in: G. Rozenberg, A. Salomaa (Eds.), Beyond Words, in: Handbook of Formal Languages, vol. 3, Springer, 1997, pp. 1–68.
[15] Gerhard Gentzen, Untersuchungen über das logische Schliessen, Math. Z. 39 (1) (1935) 176–210.
[16] Gerhard Gentzen, Die Widerspruchsfreiheit der reinen Zahlentheorie, Math. Ann. 112 (1936) 493–565.
[17] Federico Heras, Javier Larrosa, Albert Oliveras, Minimaxsat: a new weighted max-sat solver, in: International Conference on Theory and Applications of Satisfiability Testing, Addison–Wesley, 2007, pp. 41–55.
[18] Jacques Herbrand, Recherches sur la théorie de la démonstration, PhD thesis, Université de Paris, 1930.
[19] Stefan Hetzl, Applying tree languages in proof theory, in: Adrian-Horia Dediu, Carlos Martín-Vide (Eds.), Language and Automata Theory and Applications, LATA, 2012, in: Lecture Notes in Computer Science, vol. 7183, Springer, 2012.
[20] Stefan Hetzl, Alexander Leitsch, Giselle Reis, Janos Tapolczai, Daniel Weller, Introducing quantified cuts in logic with equality, in: Stéphane Demri, Deepak Kapur, Christoph Weidenbach (Eds.), Automated Reasoning – 7th International Joint Conference, IJCAR, in: Lecture Notes in Computer Science, vol. 8562, Springer, 2014, pp. 240–254.
[21] Stefan Hetzl, Alexander Leitsch, Giselle Reis, Daniel Weller, Algorithmic introduction of quantified cuts, Theoret. Comput. Sci. 549 (2014) 1–16.
[22] Stefan Hetzl, Alexander Leitsch, Daniel Weller, Towards algorithmic cut-introduction, in: Logic for Programming, Artificial Intelligence and Reasoning, LPAR-18, in: Lecture Notes in Computer Science, vol. 7180, Springer, 2012, pp. 228–242.

[23] Stefan Hetzl, Alexander Leitsch, Daniel Weller, Bruno Woltzenlogel Paleo, Herbrand sequent extraction, in: Serge Autexier, John Campbell, Julio Rubio, Volker Sorge, Masakazu Suzuki, Freek Wiedijk (Eds.), Intelligent Computer Mathematics, in: Lecture Notes in Artificial Intelligence, vol. 5144, Springer, 2008, pp. 462–477.
[24] Stefan Hetzl, Lutz Straßburger, Herbrand-confluence for cut-elimination in classical first-order logic, in: Patrick Cégielski, Arnaud Durand (Eds.), Computer Science Logic, CSL, 2012, in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 16, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, pp. 320–334.
[25] Florent Jacquemard, Francis Klay, Camille Vacher, Rigid tree automata, in: Adrian Horia Dediu, Armand-Mihai Ionescu, Carlos Martín-Vide (Eds.), Language and Automata Theory and Applications, LATA, 2009, in: Lecture Notes in Computer Science, vol. 5457, Springer, 2009, pp. 446–457.
[26] Florent Jacquemard, Francis Klay, Camille Vacher, Rigid tree automata and applications, Inform. and Comput. 209 (2011) 486–512.
[27] Moa Johansson, Lucas Dixon, Alan Bundy, Conjecture synthesis for inductive theories, J. Automat. Reason. 47 (3) (2011) 251–289.
[28] Matt Kaufmann, Panagiotis Manolios, J. Strother Moore (Eds.), Computer-Aided Reasoning: ACL2 Case Studies, Kluwer Academic Publishers, 2000.
[29] Matt Kaufmann, Panagiotis Manolios, J. Strother Moore, Computer-Aided Reasoning: An Approach, Kluwer Academic Publishers, 2000.
[30] Koen Claessen, Moa Johansson, Dan Rosén, Nicholas Smallbone, Automating inductive proofs using theory exploration, in: Maria Paola Bonacina (Ed.), Proceedings of the 24th International Conference on Automated Deduction, CADE-24, in: Lecture Notes in Computer Science, vol. 7898, Springer, 2013, pp. 392–406.
[31] William Sonnex, Sophia Drossopoulou, Susan Eisenbach, Zeno: an automated prover for properties of recursive data structures, in: Cormac Flanagan, Barbara König (Eds.), The 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS, 2012, in: Lecture Notes in Computer Science, vol. 7214, Springer, 2012, pp. 407–421.
[32] G. Sutcliffe, The TPTP world – infrastructure for automated reasoning, in: E. Clarke, A. Voronkov (Eds.), Proceedings of the 16th International Conference on Logic for Programming Artificial Intelligence and Reasoning, in: Lecture Notes in Artificial Intelligence, vol. 6355, Springer-Verlag, 2010, pp. 1–12.