J Autom Reasoning (2019) 63:95–126
https://doi.org/10.1007/s10817-018-9462-8

On the Generation of Quantified Lemmas

Gabriel Ebner[1] · **Stefan Hetzl**[1] · **Alexander Leitsch**[2] · **Giselle Reis**[3] · **Daniel Weller**[1]

Received: 16 January 2017 / Accepted: 14 March 2018 / Published online: 23 March 2018

Abstract In this paper we present an algorithmic method of lemma introduction. Given a proof in predicate logic with equality the algorithm is capable of introducing several universal lemmas. The method is based on an inversion of Gentzen's cut-elimination method for sequent calculus. The first step consists of the computation of a compact representation (a so-called decomposition) of Herbrand instances in a cut-free proof. Given a decomposition the problem of computing the corresponding lemmas is reduced to the solution of a second-order unification problem (the solution conditions). It is shown that that there is always a solution of the solution conditions, the canonical solution. This solution yields a sequence of lemmas and, finally, a proof based on these lemmas. Various techniques are developed to simplify the canonical solution resulting in a reduction of proof complexity. Moreover, the paper contains a comprehensive empirical evaluation of the implemented method and gives an application to a mathematical proof.

Keywords Cut-introduction · Herbrand's theorem · Proof theory · Lemma generation · The resolution calculus

✉ Stefan Hetzl
stefan.hetzl@tuwien.ac.at

Gabriel Ebner
gebner@gebner.org

Alexander Leitsch
leitsch@logic.at

Giselle Reis
giselle@cmu.edu

Daniel Weller
weller@logic.at

[1] Institute of Discrete Mathematics and Geometry, Vienna University of Technology, Vienna, Austria

[2] Institute of Logic and Computation, Vienna University of Technology, Vienna, Austria

[3] Carnegie Mellon University - Qatar, Al Rayyan, Qatar

1 Introduction

Computer-generated proofs are typically analytic, i.e. they only contain logical material that also appears in the theorem proved. This is due to the fact that analytic proof systems have a much smaller search space which makes proof-search practically feasible. In the case of sequent calculi, proof-search procedures typically work on the cut-free fragment. Resolution is also essentially analytic as resolution proofs do not contain complex lemmas. An important property of non-analytic proofs is their considerably smaller length. The exact difference depends on the logic (or theory) under consideration, but it is typically enormous. In (classical and intuitionistic) first-order logic there are proofs with cut of length n whose theorems have only cut-free proofs of length 2_n (where $2_0 = 1$ and $2_{n+1} = 2^{2_n}$), see [24,27,32]. In contrast, proofs formalized by humans are almost never analytic. Human insight and understanding of a mathematical situation is manifested in the use of concepts, as well as properties of, and relations among, concepts in the form of lemmas. This leads to a high-level structure of a proof. For these two reasons, their length and the insight they (can) contain, we consider the generation of non-analytic proofs an aim of high importance to automated deduction.

There is another, more theoretical, motivation for studying cut-introduction which derives from the foundations of mathematics: most of the central mathematical notions have developed from the observation that many proofs share common structures and steps of reasoning. Encapsulating those leads to a new abstract notion, like that of a group or a vector space. Such a notion then builds the base for a whole new theory whose importance stems from the pervasiveness of its basic notions in mathematics. From a logical point of view this corresponds to the introduction of cuts into an existing proof database. While the introduction of these notions can certainly be justified from a pragmatic point of view since it leads to natural and concise presentations of mathematical theories, the question remains whether they can be justified on more fundamental grounds as well. In particular, the question remains whether the notions at hand provide for an optimal compression of the proofs under consideration. A cut-introduction method based on such quantitative aspects (as the one described in this paper) has the potential to answer such questions, see Sect. 6.1 for a case study.

Work on cut-introduction can be found at a number of different places in the literature. Closest to our work are other approaches which aim to abbreviate or structure *a given input proof*: [35] is an algorithm for the introduction of atomic cuts that is capable of exponential proof compression. The method [15] for propositional logic is shown to never increase the size of proofs more than polynomially. Another approach to the compression of first-order proofs by introduction of definitions for abbreviating terms is [34].

Viewed from a broader perspective, this paper should be considered part of a large body of work on the generation of non-analytic formulas that has been carried out by numerous researchers in various communities. Methods for lemma generation are of crucial importance in inductive theorem proving, which frequently requires generalization [7], see e.g. [20] for a method in the context of rippling [8] that is based on failed proof attempts. In automated theory formation [10,11], an eager approach to lemma generation is adopted. This work has, for example, led to automated classification results of isomorphism classes [30] and isotopy classes [31] in finite algebra. See also [21] for an approach to inductive theory formation. In pure proof theory, an important related topic is Kreisel's conjecture on the generalization of proofs, see [9]. Based on methods developed in this tradition, [5] describes an approach to cut-introduction by filling a proof skeleton, i.e. an abstract proof structure, obtained by an inversion of Gentzen's procedure with formulas in order to obtain a proof with cuts. The use of cuts for structuring and abbreviating proofs is also of relevance in logic programming: [23]

shows how to use focusing in order to avoid proving atomic subgoals twice, resulting in a proof with atomic cuts.

Our previous work in this direction has started with [19] where we presented a basic algorithm for the introduction of a single cut with a single universal quantifier in pure first-order logic. In [17] we have made the method practically applicable by extending it to compute a Π_1-cut with an arbitrary number of quantifiers and by working modulo equality. In [17] we have also presented and evaluated an implementation. The method has been further extended on a proof-theoretic level to the introduction of an arbitrary number of Π_1-cuts with one quantifier each in [18] which already allows for an exponential compression.

In this paper we extend the method to predicate logic with equality and to the introduction of an arbitrary number of Π_1-cuts, each of which has an arbitrary number of quantifiers. We present an implementation based on a new (and efficient) algorithm for computing a decomposition of a Herbrand-disjunction [12]. We carry out a comprehensive empirical evaluation of the implementation and describe a case study demonstrating how our algorithm generates the notion of a partial order from a proof about a lattice. This paper thus completes the theory and implementation of our method for the introduction of Π_1-cuts.

The paper is organized in the same order as the steps of our algorithm. In Sect. 2, we recall basic notions and results about proofs, as well as the extraction of Herbrand sequents and how to encode them as term sets. Sect. 3 is devoted to the computation of decompositions of those term sets. Then in Sect. 4, we describe how to compute canonical cut formulas induced by the decomposition. We present several techniques to simplify those canonical cut formulas in Sect. 5. At the end, we describe our implementation and experiments in Sect. 6.

2 Proofs and Herbrand Sequents

Throughout this paper we consider predicate logic with equality. We typically use the names a, b, c for constants, f, g, h for functions, x, y, z, α for variables, Γ and Δ for sets for formulas, and $\mathcal{S}$ for sequents. We write sequents in the form $\mathcal{S}\colon A_1, \ldots, A_n \to B_1, \ldots, B_m$ where $\mathcal{S}$ is interpreted as the formula $(A_1 \wedge \cdots \wedge A_n) \supset (B_1 \vee \cdots \vee B_m)$. For convenience we write a substitution $[x_1 \backslash t_1, \ldots, x_n \backslash t_n]$ in the form $[\overline{x} \backslash \overline{t}]$ for $\overline{x} = (x_1, \ldots, x_n)$ and $\overline{t} = (t_1, \ldots, t_n)$.

For practical reasons equality will not be axiomatized but handled via substitution rules. We extend the sequent calculus **LK** to the calculus $\mathbf{LK}_=$ by allowing sequents of the form $\to t = t$ as initial sequents and adding the following rules:

$$\frac{s = t, A[s], \Gamma \to \Delta}{s = t, A[t], \Gamma \to \Delta} =^1_{\mathrm{l}} \qquad \frac{s = t, A[t], \Gamma \to \Delta}{s = t, A[s], \Gamma \to \Delta} =^2_{\mathrm{l}}$$

$$\frac{s = t, \Gamma \to \Delta, A[s]}{s = t, \Gamma \to \Delta, A[t]} =^1_{\mathrm{r}} \qquad \frac{s = t, \Gamma \to \Delta, A[t]}{s = t, \Gamma \to \Delta, A[s]} =^2_{\mathrm{r}}$$

$\mathbf{LK}_=$ is sound and complete for predicate logic with equality.

A *strong quantifier* is a $\forall$ ($\exists$) quantifier with positive (negative) polarity. The logical complexity $|\mathcal{S}|_l$ of a sequent $\mathcal{S}$ is the number of propositional connectives, quantifiers and atoms $\mathcal{S}$ contains. We restrict our investigations to end-sequents in prenex form without strong quantifiers.

Definition 1 A Σ_1-*sequent* is a sequent of the form

$$\forall x_1 \cdots \forall x_{k_1} F_1, \ldots, \forall x_1 \cdots \forall x_{k_p} F_p \to \exists x_1 \cdots \exists x_{k_{p+1}} F_{p+1}, \ldots, \exists x_1 \cdots \exists x_{k_q} F_q.$$

for quantifier free F_i.

Note that the restriction to Σ_1-sequents does not constitute a substantial restriction as one can transform every sequent into a validity-equivalent Σ_1-sequent by Skolemization and prenexing.

Definition 2 A sequent $\mathcal{S}$ is called *E-valid* if it is valid in predicate logic with equality; $\mathcal{S}$ is called a *quasi-tautology* [29] if $\mathcal{S}$ is quantifier-free and E-valid.

We use $\models$ for the semantic consequence relation in predicate logic with equality.

Definition 3 The length of a proof φ, denoted by $|\varphi|$, is defined as the number of inferences in φ. The quantifier complexity of φ, written as $|\varphi|_q$, is the number of weak quantifier inferences in φ.

2.1 Extraction of Terms

Herbrand sequents of a sequent $\mathcal{S}$ are sequents consisting of instantiations of $\mathcal{S}$ which are quasi-tautologies. The formal definition is:

Definition 4 Let $\mathcal{S}$ be a Σ_1-sequent as in Definition 1 and let H_i be a finite set of k_i-vectors of terms for every $i \in \{1, \ldots, q\}$. We define a set of quantifier-free formulas $\mathcal{F}_i = \{F_i[\overline{x_i}\backslash\overline{t}] | \overline{t} \in H_i\}$ for each i, and combine them in them in a sequent:

$$\mathcal{S}^* = (\mathcal{F}_1 \cup \cdots \cup \mathcal{F}_p \to \mathcal{F}_{p+1} \cup \cdots \cup \mathcal{F}_q)$$

If $\mathcal{S}^*$ is a quasi-tautology, then $\mathcal{S}^*$ is called a *Herbrand sequent* of $\mathcal{S}$ and the tuple $H = (H_1, \ldots, H_q)$ is called a *Herbrand structure* of $\mathcal{S}$. We define the *instantiation complexity* of $\mathcal{S}^*$ as $|\mathcal{S}^*|_i = \sum_{i=1}^{q} k_i |H_i|$.

Note that, in the instantiation complexity of a Herbrand sequent, we count the formulas weighted by the number of their quantifiers. Formulas in $\mathcal{S}$ without quantifiers are represented by empty tuples in the Herbrand structure (e.g. $H_i = \{()\}$), and do not affect the instantiation complexity as they are weighted by 0.

Example 5 Consider the language containing a constant symbol a, unary function symbols f, s, a binary predicate symbol P, and the sequent $\mathcal{S}$ defined below. We write f^n, s^n for n-fold iterations of f and s and omit parentheses around the argument of a unary symbol when convenient. Let

$$\mathcal{S} = (P(f^4a, a), \forall x.fx = s^2x, \forall xy(P(sx, y) \supset P(x, sy)) \to P(a, f^4a))$$

and $H = (H_1, H_2, H_3, H_4)$ for

$$H_1 = \{()\},\ H_4 = \{()\},\ H_2 = \{a, fa, f^2a, f^3a\},$$
$$H_3 = \{(s^3f^2a, a), (s^2f^2a, sa), (sf^2a, s^2a), (f^2a, s^3a), (s^3a, f^2a), (s^2a, sf^2a),$$
$$(sa, s^2f^2a), (a, s^3f^2a)\}.$$

Then

$$\mathcal{F}_1 = \{P(f^4a, a)\},\ \mathcal{F}_4 = \{P(a, f^4a)\},$$
$$\mathcal{F}_2 = \{fa = s^2a,\ f^2a = s^2fa,\ f^3a = s^2f^2a,\ f^4a = s^2f^3a\}$$
$$\mathcal{F}_3 = \{P(s^4f^2a, a) \supset P(s^3f^2a, sa), P(s^3f^2a, sa) \supset P(s^2f^2a, s^2a),$$

$$P(s^2f^2a, s^2a) \supset P(sf^2a, s^3a),\ P(sf^2a, s^3a) \supset P(f^2a, s^4a),$$
$$P(s^4a, f^2a) \supset P(s^3a, sf^2a),\ P(s^3a, sf^2a) \supset P(s^2a, s^2f^2a),$$
$$P(s^2a, s^2f^2a) \supset P(sa, s^3f^2a),\ P(sa, s^3f^2a) \supset P(a, s^4f^2a)\}.$$

A Herbrand-sequent $\mathcal{S}^*$ corresponding to H is then $\mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3 \rightarrow \mathcal{F}_4$. The instantiation complexity of $\mathcal{S}^*$ is 20. $\mathcal{S}^*$ is a quasi-tautology but not a tautology.

Theorem 6 (Mid-sequent theorem) *Let $\mathcal{S}$ be a Σ_1-sequent and π a cut-free proof of $\mathcal{S}$. Then there is a Herbrand sequent $\mathcal{S}^*$ of $\mathcal{S}$ s.t. $|\mathcal{S}^*|_i \leq |\pi|_q |\mathcal{S}|_l$.*

Proof This result is proven in [16] (section IV, theorem 2.1) for **LK**, but the proof for $\mathbf{LK}_=$ is basically the same. By permuting the inference rules, one obtains a proof π' from π which has an upper part containing only propositional inferences and the equality rules (which can be shifted upwards until they are applied to atoms only) and a lower part containing only quantifier inferences. The sequent between these parts is called *mid-sequent* and has the desired properties. □

$\mathcal{S}^*$ can be obtained by tracing the introduction of quantifiers in the proof, which for every formula $Q\bar{x}_i.F_i$ in the sequent (where $Q \in \{\forall, \exists\}$) yields a set of term tuples H_i, and then computing the sets of formulas $\mathcal{F}_i$.

The algorithm for introducing cuts described here relies on computing a compressed representation of a Herbrand structure, which is explained in Sect. 3. Note, though, that the Herbrand structure $(H_1, \ldots, H_q)$ is a list of sets of term tuples (i.e. each H_i is a set of tuples $\bar{t}$ used to instantiate the formula F_i). In order to facilitate computation and representation, we will add to the language fresh function symbols $f_1, \ldots, f_q$. Each f_i will be applied to the tuples of the set H_i, therefore encoding a list of sets of tuples into a set of terms. In this new set, each term will have some f_i as its head symbol that indicates to which formula the arguments of f_i belong.

Definition 7 Let $\mathcal{S}$ be a Σ_1-sequent as in Definition 1 and let $f_1, \ldots, f_q$ be fresh function symbols. We then say that the term $f_i(\bar{t})$ *encodes* the instance $F_i[\bar{x}\backslash\bar{t}]$. Terms of the form $f_i(\bar{t})$ for some i are called *decodable*.

We refer to the encoded Herbrand structure as the *term set* of a proof. Conversely, such a term set defines a Herbrand structure and thus a Herbrand sequent.

Example 8 Using this new notation, the Herbrand structure H of the previous example is now represented as the set of terms:

$$T = \{f_1, f_4, f_2(a), f_2(fa), f_2(f^2a), f_2(f^3a),$$
$$f_3(s^3f^2a, a), f_3(s^2f^2a, sa), \ldots, f_3(a, s^3f^2a)\}$$

3 Computing Decompositions

Computing a compact representation of the Herbrand structure of a cut-free proof is the first step in our lemma introduction algorithm. This is accomplished by computing decompositions of a proof's term set.

Definition 9 (Decomposition) We define a *decomposition D* as:

$$U \circ_{\overline{\alpha_1}} S_1 \circ_{\overline{\alpha_2}} \ldots \circ_{\overline{\alpha_k}} S_k$$

- $\overline{\alpha_i}$ is a vector of variables of size n_i;
- all of the variables $\overline{\alpha_i}_j$ are pairwise different;
- U is a finite set of terms which can contain all variables from $\overline{\alpha_1}$, ..., $\overline{\alpha_k}$
- S_i is a finite set of term vectors of size n_i;
- the terms in S_i's vectors may only contain the variables from $\overline{\alpha_{i+1}}$, ..., $\overline{\alpha_k}$ (consequently, S_k contains only ground terms).

The *language* of D is $L(D) = \{u[\overline{\alpha_1}\backslash s_1]...[\overline{\alpha_k}\backslash s_k] \mid u \in U \text{ and } s_i \in S_i\}$. For a finite set of ground terms T, we say that D *covers* T, or equivalently that D is a decomposition of T, iff $L(D) \supseteq T$. Finally, the *size* $|D|$ of a decomposition is defined as $|U| + \sum_{i=1}^{k} n_i |S_i|$.

Note that in the definition of covering above, we only require that $L(D)$ is a superset of T and not that it is equal to T. This requirement is motivated by a property of Herbrand sequents: every supersequent of a Herbrand sequent is a Herbrand sequent as well. This relaxed requirement allows us to consider more decompositions for a given term set, and hence obtain a stronger compression. We aim to find a decomposition of minimal size that covers a given term set T.

The notion of decomposition in Definition 9 is stated purely on the level of formal languages, without any references to proofs or formulas or Herbrand sequents. The algorithms we present in this section will likewise not be concerned with proofs, and compute decompositions based purely on the set of terms they get as input. Unfortunately not all decompositions can be decoded into quantifier instances of a proof with cut—however a very slight restriction on the decomposition suffices to ensure that this is nevertheless possible:

Definition 10 Let $\mathcal{S}$ be a Σ_1-sequent. Then a decomposition $D = U \circ_{\overline{\alpha_1}} S_1 \circ_{\overline{\alpha_2}} \cdots \circ_{\overline{\alpha_k}} S_k$ is called *decodable* (for $\mathcal{S}$) iff every term $u \in U$ is decodable.

Recall that a term u is decodable iff it is of the form $f_i(\overline{t})$ for some i. Regarding only the size of the decomposition, this is no restriction at all. We can always transform a non-decodable decomposition into a decodable one, without increasing its size. The crucial property here is that the function symbols f_i only appear as the root symbols of terms in T, and not inside the terms.

Lemma 11 *Let T be a finite set of ground terms, and D a decomposition of T. Then there exists a decomposition D' of T such that $|D'| \leq |D|$ and $n_i = 1$ for all i.*

Proof We replace every $\circ_{\overline{\alpha_i}} S_i$ in the decomposition with $\circ_{\alpha_{i,1}} \pi_1(S_i) \cdots \circ_{\alpha_{i,n_i}} \pi_{n_i}(S_i)$, where π_m is the m-th projection. That is, instead of substituting all the variables in $\overline{\alpha_i}$ at once, we substitute them one by one. The size of decomposition does not increase since we multiplied the number of elements in S_i by n_i. □

Lemma 12 *Let π be a cut-free proof, T its term set, and D a decomposition of T. Then there exists a decodable decomposition D' of T such that $|D'| \leq |D|$.*

Proof Without loss of generality assume $n_i = 1$ for all i, and let $U \circ_{\alpha_1} S_1 \circ_{\alpha_2} \cdots \circ_{\alpha_k} S_k = D$. Define $U' = \{u \in U \cup S_1 \cup \cdots \cup S_k \mid \exists i\ u = f_i(\ldots)\}$, $S'_i = S_i \setminus U'$ for $1 \leq i \leq k$, and set $D' = U' \circ_{\alpha_1} S'_1 \circ_{\alpha_2} \cdots \circ_{\alpha_k} S'_k$, leaving out any S'_i where $S'_i = \emptyset$. The size of the decomposition does not increase with this transformation. We need to show that $L(D) \supseteq T$, so let $t = u[\alpha_1\backslash s_1, \ldots, \alpha_k\backslash s_k] \in T$ where $u \in U$, and $s_i \in S_i$ for all i. If u is of the form $f_i(\ldots)$, then any s_i such that $s_i \in U'$ is irrelevant since it does not contribute to t. If $S_i \setminus U' = \emptyset$, then we leave them out, otherwise we replace them by an arbitrary other $s'_i \in S'_i$. On the other hand, if $u = \alpha_j$ for some j, then we change u to $u' = s_j \in U'$ and leave out or replace the remaining s_i as before. □

We can also formulate the problem of finding a minimal decomposition as a decision problem: given a finite set of ground terms T and $m \geq 0$, is there a decomposition D of T such that $|D| \leq m$? This problem is in NP: given a decomposition D, and for every term $t \in T$ the necessary substitutions, we can check in polynomial time whether the language covers T. We conjecture that the problem is NP-hard as well.

Definition 13 An algorithm to produce decompositions takes as input a finite set of ground terms T, and returns a decomposition $D = U \circ_{\overline{\alpha_1}} S_1 \circ_{\overline{\alpha_2}} \cdots \circ_{\overline{\alpha_k}} S_k$ of T. Such an algorithm is called *complete* iff it always returns a decomposition of minimal possible size.

We will now present an incomplete but practically feasible solution to find small decompositions for a term set. Our algorithm is based on an operation called Δ-vector. Intuitively, it computes "greedy" decompositions $U \circ S$ with only one element in the set U. We will call those *simple* decompositions. They are stored in a data structure called Δ-table, which is later processed for combining simple decompositions into more complex ones.

A previous version of this algorithm was presented in [17]. Since then, we have identified its source of incompleteness and implemented the so-called *row merging* heuristic for finding more decompositions. Additionally, many bugs in the implementation were fixed.

3.1 The Δ-vector

Definition 14 Let T be a finite, non-empty set of terms, u a term and S a set of substitutions. Then (u, S) is a *simple decomposition* of T iff $uS = \{u\sigma \mid \sigma \in S\} = T$. Additionally, (u, S) is called *trivial* iff u is a variable.

Example 15 Let $T = \{f(c, c), f(d, d)\}$. Then $(f(\alpha, \alpha), \{[\alpha\backslash c], [\alpha\backslash d]\})$ is a simple decomposition of T. Another decomposition of T is $(\alpha, \{[\alpha\backslash f(c, c)], [\alpha\backslash f(d, d)]\})$, which is simple and trivial.

Given a non-empty subset $T' \subseteq T$, the Δ-vector for T' produces a simple decomposition of T'; we write $\Delta(T') = (u, S)$. This term u is computed via *least general generalization*, a concept introduced independently in [25,26] and [28]. The least general generalization of two terms is computed recursively:

Definition 16 Let $\alpha_{t,s}$ be a different variable for each pair of terms (t, s).

$$\begin{aligned} \mathrm{lgg}(f(t_1, \ldots, t_n), f(s_1, \ldots, s_n)) &= f(\mathrm{lgg}(t_1, s_1), \ldots, \mathrm{lgg}(t_n, s_n)) \\ \mathrm{lgg}(t, s) &= \alpha_{t,s} \quad \text{otherwise} \end{aligned}$$

Example 17 Let f, a, and b be constants, and g a binary function symbol, then $\mathrm{lgg}(f, g(a, b)) = \alpha_1$, $\mathrm{lgg}(g(a, b), g(b, a)) = g(\alpha_1, \alpha_2)$, $\mathrm{lgg}(g(a, b), g(a, a)) = g(a, \alpha_1)$, and $\mathrm{lgg}(g(a, a), g(b, b)) = g(\alpha_1, \alpha_1)$.

To have a canonical result term, we use the names $\alpha_1, \ldots, \alpha_n$ for the variables in $\mathrm{lgg}(t, s)$, read left-to-right. The lgg subsumes each of the arguments: given terms t and s, there always exist substitutions σ and τ such that $\mathrm{lgg}(t, s)\sigma = t$ and $\mathrm{lgg}(t, s)\tau = s$. The lgg operation is associative and commutative as well, and we can naturally extend it to finite non-empty sets of terms:

Definition 18 Let $T = \{t_1, \ldots, t_n\}$ be a non-empty finite set of terms. We define its least general generalization $\mathrm{lgg}(T)$ using the binary lgg operation:

$$\begin{aligned} \mathrm{lgg}\{t_1\} &= t_1 \\ \mathrm{lgg}\{t_1, \ldots, t_n\} &= \mathrm{lgg}(t_1, \mathrm{lgg}(t_2, \ldots \mathrm{lgg}(t_{n-1}, t_n))) \quad \text{if } n \geq 2 \end{aligned}$$

Example 19 Let a and b be constants, f a unary and g a binary function symbol, then $\mathrm{lgg}\{f(a)\} = f(a)$, $\mathrm{lgg}\{f(a), f(b)\} = f(\alpha_1)$, and $\mathrm{lgg}\{f(a), f(b), g\} = \alpha_1$. Additionally let $l = \mathrm{lgg}\{g(a,a), g(b,b), g(f(b), f(b))\} = g(\alpha_1, \alpha_1)\}$, then l subsumes each of the three arguments: $l[\alpha_1 \backslash a] = g(a,a)$, $l[\alpha_1 \backslash b] = g(b,b)$, $l[\alpha_1 \backslash f(b)] = g(f(b), f(b))$,

Just as in the binary case, the lgg always subsumes it arguments: for each $t \in T'$, there exists a substitution σ_t such that $\mathrm{lgg}(T')\sigma_t = t$. We can now define the Δ-vectors in terms of the lgg:

Definition 20 Let T' be a finite, non-empty, set of ground terms. Then we define its Δ-vector as $\Delta(T') = (\mathrm{lgg}(T'), \{\sigma_t \mid t \in T'\})$, where for every $t \in T'$ the substitution σ_t satisfies $\mathrm{lgg}(T')\sigma_t = t$.

Example 21 Let $T' = \{f(c,c), f(d,d)\}$, then $\Delta(T') = (f(\alpha_1, \alpha_1), \{[\alpha_1 \backslash c], [\alpha_1 \backslash d]\})$.

Algebraically, we can consider the set of terms where we identify terms up to variable renaming. This set is partially ordered by subsumption and the lgg computes the meet operation, making it a meet-semilattice. In this semilattice, terms have a least upper bound iff they are unifiable; the join operation is given by most general unification.

The subset of terms with at most one variable is such a semilattice as well: every pair of two terms has a greatest lower bound. From this point of view, we can also define a function lgg_1 as the meet operation in the subsemilattice of terms with at most one variable. We may then define a variant $\Delta_1(T) = (u, S)$ of the Δ-vector, where u may contain only a single variable. We will compare both variants of the Δ-vector in the large-scale experiments in Sect. 6.2.

3.2 The Δ-table

The Δ-table is a data-structure that stores all *non-trivial* Δ-vectors of subsets of T, indexed by their sets of substitutions. Some of these simple decompositions are later combined into a decomposition of T.

Definition 22 A Δ-row is a pair $S \to U$ where S is a set of substitutions, and U is a set of pairs (u, T) such that $uS = T$. A Δ-table is a map where every key-value pair is a Δ-row.

Algorithm 1 computes a Δ-table containing the Δ-vectors for all subsets of T. As an optimization, we do not iterate over all subsets. Instead we incrementally add terms to the subset, stopping as soon as the Δ-vector is trivial. This optimization is justified by the following lemma:

Theorem 23 *Let T be a set of terms. If $\Delta(T)$ is trivial, then so is $\Delta(T')$ for every $T' \supseteq T$.*

Proof Let $\Delta(T) = (u, S)$ and $\Delta(T') = (u', S')$. By the subsumption property of the lgg, there is a substitution σ such that $u'\sigma = u$. So if u is a variable, then u' is necessarily a variable as well. □

Whenever a subset T' of T has a *trivial* decomposition, i.e. $\Delta(T') = (\alpha_1, T')$, it is not added to the Δ-table. Moreover, no superset of T' is considered from this point on, since we know that these will also have only trivial decompositions.

After having computed the Δ-table, we need to combine the simple decompositions to find a suitable one, i.e., generating the full set T. Since we did not add trivial decompositions, each

Algorithm 1 Δ-table algorithm

```
function POPULATE(M: Δ-table, L: list of terms, T: set of terms)
  if L is non-empty then
    T' ← T ∪ {HEAD(L)}
    (u, S) ← Δ- VECTOR(T')
    if u is not a variable then
      M[S] ← M[S] + (u, T')
      POPULATE(M, TAIL(L), T')
    end if
    POPULATE(M, TAIL(L), T)
  end if
end function
function COMPUTEDECOMPOSITION(T: list of terms)
  M ← new Δ-table
  POPULATE(M, T, ∅)
  if row-merging enabled then
    MERGESUBSUMEDROWS(M)
  end if
  COMPLETEROWS(M, T)
  return FINDMINIMALDECOMPOSITION(M)
end function
```

row of the Δ-table is completed with the pairs $(t, \{t\})$ for every $t \in T$ as a post-processing step.

Let $S \to [(u_1, T_1), ..., (u_r, T_r)]$ be one entry of T's Δ-table. We know that $T_i \subseteq T$ and that $\{u_i\} \circ S$ is a decomposition of T_i for each $i \in \{1...r\}$. Take $\{T_{i_1}, ..., T_{i_s}\} \subseteq \{T_1, ..., T_r\}$ such that $T_{i_1} \cup ... \cup T_{i_s} = T$. Then, since combining each u_{i_j} with S yields T_{i_j}, and the union of these terms is T, the decomposition $\{u_{i_1}, ..., u_{i_s}\} \circ S$ will generate all terms from T. Observe that the vector of variables $\overline{\alpha}$ used will be the same for all combined decompositions, since they share the same set S.

There might be several subsets of $\{T_1, ..., T_r\}$ that cover T, so different decompositions can be found. For our purposes, only the minimal ones are considered. In the end, the Δ-table algorithm produces a decomposition D of T. If T was the term set of a proof, then D is even decodable:

Lemma 24 *Let π be a cut-free proof, T its term set, and $D = U \circ S$ the decomposition produced by the Δ-table algorithm. Then D is decodable.*

Proof The Δ-table only contains non-trivial simple decompositions (u, S') where u is the lgg of a subset of T. Such a u is necessarily of the form $f_i(\dots)$, and hence all $u \in U$ are as well. □

Decompositions with $k > 1$ The algorithm shown (and implemented in GAPT, see Sect. 6) computes only decompositions of the shape $U \circ_{\overline{\alpha}} S$, i.e., with $k = 1$ (see Definition 9). In order to generate more general decompositions, we would have to run it again on the set U, treating all variables in $\overline{\alpha}$ as constants.

The experiments with the simpler algorithm have given satisfying results so far, even when compared to another approach which finds more general decompositions (see Sect. 3.4). We have thus decided to postpone the analysis and implementation of an iterated Δ-table method.

3.3 Incompleteness and Row-Merging

The proposed algorithm is incomplete because it only combines simple decompositions from the same line of the Δ-table (i.e., with the same set S). Completeness could be achieved by combining decompositions regardless of where they occur in the table. As an example, consider the set $T = T_r \cup T_s$, $|T| = 18$, where:

$$T_r = \{rc, rfc, rf^2c, ..., rf^8c\}$$
$$T_s = \{sd, sgd, sg^2d, ..., sg^8d\}$$

Considered separately, sets T_r and T_s have concise decompositions of size 6:

$$\{r\alpha_r, rf^3\alpha_r, rf^6\alpha_r\} \circ_{\alpha_r} \{c, fc, f^2c\}$$
$$\{s\alpha_s, sg^3\alpha_s, sg^6\alpha_s\} \circ_{\alpha_s} \{d, gd, g^2d\}$$

The Δ-table algorithm will find the elements to assemble both decompositions, but since it only combines those that have a common right-hand side, these will never be combined to obtain the following decomposition of size 12:

$$\{r\alpha_r, rf^3\alpha_r, rf^6\alpha_r, s\alpha_s, sg^3\alpha_s, sg^6\alpha_s\} \circ_{\alpha_r,\alpha_s} \{c, fc, f^2c, d, gd, g^2d\}$$

Nevertheless, the complexity of a complete algorithm makes it unfeasible. We are investigating ways to make it "more complete" by operations that would not compromise its efficiency so much.

One approach to improve the completeness of the Δ-table algorithm is to merge rows in the table. Consider the following term set T and its decomposition D:

$$\begin{aligned} T &= T_1 \cup T_2 \cup T_3 \\ T_1 &= \{q(a,b,c), q(b,c,a), q(c,a,b)\} \\ T_2 &= \{r(a,b,c), r(b,c,a), r(c,a,b)\} \\ T_3 &= \{s(a,b), s(b,c), s(c,a)\} \\ D &= \{q(\alpha_1,\alpha_2,\alpha_3), r(\alpha_1,\alpha_2,\alpha_3), s(\alpha_1,\alpha_2)\} \\ &\quad \circ \{(a,b,c), (b,c,a), (c,a,b)\} \end{aligned}$$

This decomposition will never be found as the substitutions in the Δ-vector do not match — in one case we have a substitution of two variables, in the other three variables. In particular the Δ-table will contain the following two rows (for space reasons, we abbreviate $[\alpha_1\backslash a, \alpha_2\backslash b, \alpha_3\backslash c]$ as $[a, b, c]$):

$$\{[a,b,c], [b,c,a], [c,a,b]\} \rightarrow \{(q(\alpha_1,\alpha_2,\alpha_3), T_1), (r(\alpha_1,\alpha_2,\alpha_3), T_2)\}$$
$$\{[a,b], [b,c]\} \rightarrow \{(s(\alpha_1,\alpha_2), T_3)\}$$

If we could just put the contents of the second row into the first one, then we would find the desired decomposition immediately. Intuitively, the reason we can merge the rows without violating the invariant of the Δ-table algorithm is because the substitutions of the second row are in a sense contained in the substitutions of the first row. The following definition makes this intuition precise:

Definition 25 (Substitution-set subsumption) Let S_1, S_2 be sets of substitutions, and D_1, D_2 be sets of variables such that $\mathrm{dom}(\tau) \subseteq D_i$ for all $\tau \in S_i$ and $i \in \{1, 2\}$. Then S_1 subsumes

S_2, written $S_1 \preceq S_2$, if and only if there exists an injective substitution $\sigma : D_1 \to D_2$ with the following property:

$$\forall \tau_1 \in S_1 \quad \exists \tau_2 \in S_2 \quad \forall x \in D_1 \quad x\tau_1 = x\sigma\tau_2$$

Lemma 26 *(row-merging) Let $S_1 \to R_1$ and $S_2 \to R_2$ be Δ-rows, and $S_1 \preceq S_2$ with the substitution σ witnessing this subsumption. Then $S_2 \to (R_2 \cup R_1\sigma)$ is a Δ-row as well.*

Proof Let $(u, T') \in R_1$. We need to show that $u\sigma S_2 = T'$. But this follows from $uS_1 = T'$ since $S_1 \preceq S_2$ via σ. □

After the initial computation of the Δ-table, we use this lemma to merge all pairs of rows where one set of substitutions subsumes the other. Whenever we have rows $S_1 \to R_1$ and $S_2 \to R_2$ such that $S_1 \preceq S_2$, we replace $S_2 \to R_2$ by $S_2 \to R_2 \cup R_1\sigma$ and keep the S_1 row as it is. This increases the set of possible decompositions that we can find, since we did not remove any elements of the rows. This allows to find the desired decomposition in the example. We have $\{[a, b], [b, c]\} \preceq \{[a, b, c], [b, c, a], [c, a, b]\}$ via the identity substitution $[\alpha_1 \backslash \alpha_1, \alpha_2 \backslash \alpha_2]$, and generate the following new row:

$$\{[a, b, c], [b, c, a], [c, a, b]\} \to \{(q(\alpha_1, \alpha_2, \alpha_3), T_1), (r(\alpha_1, \alpha_2, \alpha_3), T_2), (s(\alpha_1, \alpha_2), T_3)\}$$

3.4 The MaxSAT-Algorithm

In [12], the authors propose an algorithm for the compression of a finite set of terms by reducing the problem (in polynomial time) to Max-SAT. This is another method for finding a decomposition. The difference to the Δ-table algorithm is that one must provide the numbers k and $|\overline{\alpha_1}|$, ..., $|\overline{\alpha_k}|$ in advance.

Using the reduction to Max-SAT to find decompositions is, in principle, a complete algorithm, meaning that it finds all decompositions in the shape specified by the parameters. But this requires finding all possible solutions for the generated Max-SAT problems. In addition, due to the number of variables in the generated problem, it is hardly feasible to find decompositions for $k > 2$.

Given the limitations of both algorithms, their practical performance in terms of compressing proofs is comparable. Having both implementations is justified since the methods find different decompositions and therefore generate different cut formulas.

4 Computing Cut Formulas

After having computed a decomposition $U \circ S_1 \circ ... \circ S_n$ as described in Sect. 3, the next step is computing cut formulas based on that decomposition. A decomposition D specifies the instances of quantifier blocks in a proof with $\forall$-cuts (both for end-sequent and cut formulas), but does not contain information about the propositional structure of the cut formulas to be constructed.

The set U in the decomposition corresponds to the instances of formulas in the end-sequent, the sequent $\mathcal{S}_U$ in the following Definition 27 consists precisely of these instances. The sequents $\mathcal{S}^i_U$ will simplify the definition of the proof with cut—the definition of $\mathcal{S}^i_U$ is motivated by the eigenvariable condition: the instances in $\mathcal{S}^i_U$ are precisely those which may occur at a point where the eigenvariables $\overline{\alpha_1}, \ldots, \overline{\alpha_i}$ have been introduced below.

Definition 27 Let $\mathcal{S}$ be a Σ_1-sequent and F_i, k_i as in Definition 1, and $D = U \circ_{\overline{\alpha_1}} S_1 \circ_{\overline{\alpha_2}} \cdots \circ_{\overline{\alpha_n}} S_n$ a decodable decomposition. We define the sequent $\mathcal{S}_U = \mathcal{F}_{U,1}, \ldots, \mathcal{F}_{U,p} \to \mathcal{F}_{U,p+1}, \ldots, \mathcal{F}_{U,q}$, where $\mathcal{F}_{U,i} = \{F_i[\overline{x_i}\backslash\overline{t}] \mid f_i(\overline{t}) \in U\}$.

In addition, we define for every $0 \leq j \leq n$ the sequent $\mathcal{S}^j_U$ as follows: $\mathcal{S}^j_U$ consists of all formulas $F \in \mathcal{S}_U$ such that the free variables of F are included in $\overline{\alpha_j}, \ldots, \overline{\alpha_n}$.

Example 28 Consider the sequent $\mathcal{S} = P(c), \forall x.(P(x) \supset P(s(x)) \to P(s^6c)$, its Herbrand sequent $H = P(c), P(c) \supset P(sc), \ldots, P(s^5c) \supset P(s^6c) \to P(s^6c)$, and the decomposition $D = U \circ S$:

$$U = \{f_1, f_2(\alpha), f_2(s\alpha), f_2(s^2\alpha), f_3\}$$
$$S = \{c, s^3c\}$$

Now $\mathcal{S}_U$ contains the instances of $\mathcal{S}$ as specified by U, without the function symbols f_1, f_2, f_3, and the sequents $\mathcal{S}^1_U$ and $\mathcal{S}^2_U$ contain the formulas with the appropriate free variables:

$$\mathcal{S}_U = P(c), P(\alpha) \supset P(s\alpha), P(s\alpha) \supset P(s^2\alpha), P(s^2\alpha) \supset P(s^3\alpha) \to P(s^6c)$$
$$\mathcal{S}^1_U = \mathcal{S}_U$$
$$\mathcal{S}^2_U = P(c) \to P(s^6c)$$

Given a decomposition, it may be impossible to incorporate some formulas as cut formulas in a proof with the quantifier inferences indicated by the decomposition. For example, in most cases we will not be able to use $\forall\alpha_1.\bot$ as a cut formula. Definition 30 states the precise conditions under which given formulas are usable as cut formulas. These conditions are also precisely the necessary conditions that will later on allow us to build a proof with these formulas as cuts.

Definition 29 Let $\mathcal{S} = \Gamma \to \Delta$ and $\mathcal{T} = \Sigma \to \Pi$ be sequents. Then the sequent $\mathcal{S} \circ \mathcal{T} = \Gamma, \Sigma \to \Delta, \Pi$ is called the *composition* of $\mathcal{S}$ and $\mathcal{T}$.

Definition 30 Let $\mathcal{S}$ be a Σ_1-sequent, and $D = U \circ_{\overline{\alpha_1}} S_1 \circ_{\overline{\alpha_2}} \cdots \circ_{\overline{\alpha_n}} S_n$ a decodable decomposition. Moreover, let $S_i = \{\bar{w}^i_1, \ldots, \bar{w}^i_{k_i}\}$, and X_i be a fresh $|\overline{\alpha_i}|$-ary predicate variable for $1 \leq i \leq n$. Then the following sequents are called *solution conditions*:

$$I_0 = \mathcal{S}^1_U \circ (\to X_1(\overline{\alpha_1}), \ldots, X_n(\overline{\alpha_n}))$$
$$I_i = \mathcal{S}^{i+1}_U \circ (X_i(\bar{w}^i_1), \ldots, X_i(\bar{w}^i_{k_i}) \to X_{i+1}(\overline{\alpha_{i+1}}), \ldots, X_n(\overline{\alpha_n})) \quad \text{if } i > 0$$

If additionally $F_1, \ldots, F_n$ are formulas such that the free variables of F_i are contained in $\overline{\alpha_i}, \ldots, \overline{\alpha_n}$, then the second-order substitution $\sigma = [X_i\backslash\lambda\overline{\alpha_i}.F_i]^n_{i=1}$ is called a *solution* if $I_i\sigma$ is a quasi-tautology for all $0 \leq i \leq n$.

Example 31 Let $\mathcal{S}$ and D be as in Example 28. Then the solution conditions are as follows:

$$I_0 = \mathcal{S}^1_U \circ (\to X(\alpha))$$
$$= P(c), P(\alpha) \supset P(s\alpha), P(s\alpha) \supset P(s^2\alpha), P(s^2\alpha) \supset P(s^3\alpha) \to X(\alpha), P(s^6c)$$
$$I_1 = P(c), X(c), X(s^3c) \to P(s^6c)$$

The formula $F = P(\alpha) \supset P(s^3\alpha)$ forms the solution $\sigma = [X\backslash\lambda\alpha.F]$, since the following sequents are quasi-tautological (in this case, they are even tautological):

$$
\begin{aligned}
I_0\sigma = {} & P(c), P(\alpha) \supset P(s\alpha), P(s\alpha) \supset P(s^2\alpha), P(s^2\alpha) \supset P(s^3\alpha) \\
& \rightarrow P(\alpha) \supset P(s^3\alpha), P(s^6c) \\
I_1\sigma = {} & P(c), P(c) \supset P(s^3c), P(s^3c) \supset P(s^6c) \rightarrow P(s^6c)
\end{aligned}
$$

We can now proceed to give a definition of the proof with cut induced by a decomposition and a solution.

Definition 32 Let $\mathcal{S}$ be a Σ_1-sequent, $D = U \circ_{\overline{\alpha_1}} S_1 \circ_{\overline{\alpha_2}} \cdots \circ_{\overline{\alpha_n}} S_n$ a decodable decomposition, and $F_1, \ldots, F_n$ be formulas that form a solution. Let the elements of each S_i in D be $\{\bar{w}_1^i, \ldots, \bar{w}_{k_i}^i\}$. Then the *proof with cut* $\pi_{D,F}$ using the decomposition D and solution F is constructed recursively as follows:

$$
\pi^0_{D,F} = \dfrac{(\psi_0)}{\mathcal{S}^1_U \circ (\rightarrow F_1, \ldots, F_n)}{\mathcal{S}^1_U \circ \mathcal{S} \circ (\rightarrow F_1, \ldots, F_n)}
$$

$$
\pi^i_{D,F} = \dfrac{\dfrac{\dfrac{(\pi^{i-1}_{D,F})}{\mathcal{S}^i_U \circ \mathcal{S} \circ (\rightarrow F_i, \ldots, F_n)}}{\mathcal{S}^{i+1}_U \circ \mathcal{S} \circ (\rightarrow F_i, \ldots, F_n)} \quad \dfrac{(\psi_i)}{\mathcal{S}^{i+1}_U \circ (F_i[\bar{\alpha}_i \backslash \bar{w}^i_1], \ldots, F_i[\bar{\alpha}_i \backslash \bar{w}^i_{k_i}] \rightarrow F_{i+1}, \ldots, F_n)}}{\mathcal{S}^{i+1}_U \circ (\forall \bar{\alpha}_i F_i \rightarrow F_{i+1}, \ldots, F_n)}}{\mathcal{S}^{i+1}_U \circ \mathcal{S} \circ (\rightarrow F_{i+1}, \ldots, F_n)}\ \text{cut}
$$

$$
\pi_{D,F} = \dfrac{(\pi^n_{D,F})}{\mathcal{S}^{n+1}_U \circ \mathcal{S}}{\mathcal{S}}
$$

The sub-proofs $\psi_0, \ldots, \psi_n$ are cut-free proofs of the indicated sequents—these exist since F is a solution.

The construction in Definition 32 is clearly a proof in LK ending in $\mathcal{S}$. The quantifier complexity $|\pi_{D,F}|_q$ is bounded by $|\mathcal{S}|_l |U| + \sum_{i=1}^n a_i |S_i|$, where a_i is the length of the vector $\overline{\alpha_i}$.

Example 33 Continuing Examples 28 and 31, we obtain the following proof with cut $\pi_{D,F}$, where ψ_0 and ψ_1 are cut-free proofs of the indicated sequents:

$$
\dfrac{\dfrac{\dfrac{(\psi_0)}{\mathcal{S}^1_U \circ (\rightarrow P(\alpha) \supset P(s^3\alpha))}}{\mathcal{S} \circ (\rightarrow P(\alpha) \supset P(s^3\alpha))}}{\mathcal{S} \circ (\rightarrow \forall\alpha.(P(\alpha) \supset P(s^3\alpha)))} \quad \dfrac{(\psi_1)}{P(c), P(c) \supset P(s^3c), P(s^3c) \supset P(s^6c) \rightarrow P(s^6c)}{P(c), \forall\alpha.(P(\alpha) \supset P(s^3\alpha)) \rightarrow P(s^6c)}}{\mathcal{S}}
$$

The question remains whether every decomposition has a solution; we show below that this is indeed the case if the sequent defined by the term set of the decomposition is quasi-tautological. The main ingredient in this proof is the definition of a canonical substitution, which will turn out to be a solution in Theorem 35: the canonical substitution consists of formulas C_i, such that C_i captures the maximum amount of logical information from the axioms that is available above the i-th cut.

Definition 34 Let D be a decodable decomposition for a Herbrand sequent $\mathcal{S}^*$, and $\overline{\alpha_i}$, $\bar{w}_j^i$, X_i, and $\mathcal{S}_U$ be as in Definition 30. The formulas C_i are defined recursively as follows:

$$C_1 = \neg \mathcal{S}_U$$
$$C_{i+1} = \bigwedge_{j=1}^{k_i} C_i[\overline{\alpha_i} \backslash \bar{w}_j^i]$$

Then $\sigma = [X_i \backslash \lambda \overline{\alpha_i}.C_i]_{i=1}^n$ is called the *canonical substitution*.

We will now show that the canonical substitution is, in fact, a solution.

Theorem 35 *Let $\mathcal{S}$ be a valid Σ_1-sequent and D be a decodable decomposition for some Herbrand sequent $\mathcal{S}^*$ of $\mathcal{S}$. Then the canonical substitution σ is a solution.*

Proof First note that the variable condition is fulfilled as the free variables of C_i are included in $\{\overline{\alpha_i}, \ldots, \overline{\alpha_n}\}$. We now need to check that each of the sequents $I_i\sigma$ is quasi-tautological. Consider first $I_0\sigma = \mathcal{S}_U^1 \circ (\rightarrow C_1, \ldots, C_n)$. Since $\mathcal{S}_U^1 = \mathcal{S}_U$ and $C_1 = \neg\mathcal{S}_U$, we only need to observe that $\mathcal{S}_U \circ (\rightarrow \neg\mathcal{S}_U)$ is quasi-tautological.

For $0 < i \leq n$ and $I_i\sigma = \mathcal{S}_U^i \circ (\{C_i\bar{w}_j^i, 1 \leq j \leq k_i\} \rightarrow C_{i+1}, \ldots, C_n)$, we see that $\{C_i\bar{w}_j^i, 1 \leq j \leq k_i\}$ is equivalent to C_{i+1}, and only need to show that $\mathcal{S}_U^i \circ (C_{i+1} \rightarrow C_{i+1}, \ldots, C_n)$ is quasi-tautological, which is clear in the case $i < n$. For $i = n$ it suffices to show that $C_{n+1} \rightarrow$ is quasi-tautology: this is true since the sequent defined by the term set of D is quasi-tautological, and $C_{n+1} \rightarrow$ is logically equivalent to the Herbrand sequent represented by the term set. □

The cut formulas corresponding to the canonical solution are

$$\forall \overline{\alpha_1}\, C_1, \ldots, \forall \overline{\alpha_n}\, C_n$$

Example 36 Applying Theorem 35 to our running example, we obtain the canonical substitution $\sigma = [X \backslash \lambda\alpha C_1]$:

$$C_1 = P(c) \wedge (P(\alpha) \supset P(s\alpha)) \wedge (P(s\alpha) \supset P(s^2\alpha)) \wedge (P(s^2\alpha) \supset P(s^3\alpha)) \wedge \neg P(s^6c)$$

The cut formula corresponding to the canonical substitution is $\forall\alpha.C_1$.

5 Improving the Solution

After completing the first phase of cut-introduction, namely the computation of a decomposition, the next step is to find a solution to the schematic extended Herbrand sequent induced by the decomposition. Such a solution is guaranteed to exist by Theorem 35, and its construction is described in Definition 34. But is this solution optimal? The canonical solution as defined in Sect. 4 is relatively large, in general even exponential in the size of the decomposition. As a first step towards a smaller solution, we consider a slightly less elegant version of the canonical solution with lower logical complexity:

Definition 37 Let D be a decodable decomposition for a sequent $\mathcal{S}$, and let $\overline{\alpha_i}$, $\bar{w}_j^i$, X_i, and $\mathcal{S}_U$ be as in Definition 30. Furthermore let the formulas C_i' be defined recursively as follows, where $(\Gamma \rightarrow \Delta) \setminus (\Pi \rightarrow \Lambda) = (\Gamma \setminus \Pi) \rightarrow (\Delta \setminus \Lambda)$ denotes the difference operation on sequents:

$$C'_1 = \neg(\mathcal{S}^1_U \setminus \mathcal{S}^2_U)$$

$$C'_{i+1} = \neg(\mathcal{S}^{i+1}_U \setminus \mathcal{S}^{i+2}_U) \wedge \bigwedge_{j=1}^{k_i} C_i[\overline{\alpha_i} \backslash \vec{w}^i_j]$$

Then $\sigma' = [X_i \backslash \lambda \vec{\alpha_i}.C'_i]_{i=1}^n$ is called the *modified canonical substitution*.

The "regular" canonical solution introduces all instances immediately in C_1. By contrast, the modified canonical solution introduces instances as late as possible. Purely propositional instances are never included.

Theorem 38 *Let $\mathcal{S}$ be a Σ_1-sequent and D be a decodable decomposition for some Herbrand sequent $\mathcal{S}^*$ of $\mathcal{S}$. Then the modified canonical substitution σ' is a solution.*

Proof Similar to the proof of Theorem 35. □

If we approach the question of optimality from the point of view of the $|\cdot|_q$ measure, then all solutions can be considered equivalent. From the point of view of symbolic complexity or logical complexity, things may be different: there are cases where the canonical solution is large, but small solutions exist. The following example exhibits such a case. In this example, a smaller solution not only exists, but is also more natural than (and hence in many applications preferable to) the canonical solution.

Example 39 Consider the sequent

$$\mathcal{S}\colon Pa, \forall x\,(Px \supset Pfx) \to Pf^9a.$$

Then $\mathcal{S}$ has a (minimal) Herbrand-sequent

$$H\colon Pa, Pa \supset Pfa, \ldots, Pf^8a \supset Pf^9a \to Pf^9a.$$

The terms of this Herbrand-sequent are represented by the decomposition

$$D = U \circ W = \{\alpha, f\alpha, f^2\alpha\} \circ \{a, f^3a, f^6a\}$$

which gives rise to the solution conditions:

$$I_0 = Pa, P\alpha \supset Pf\alpha, Pf\alpha \supset Pf^2\alpha, Pf^2\alpha \supset Pf^3\alpha \to X\alpha, Pf^9a.$$
$$I_1 = Pa, Xa, Xf^3a, Xf^6 \to Pf^9a.$$

The corresponding canonical solution is $\sigma = [X \backslash \lambda\alpha.C]$ with

$$C = Pa \wedge (P\alpha \supset Pf\alpha) \wedge (Pf\alpha \supset Pf^2\alpha) \wedge (Pf^2\alpha \supset Pf^3\alpha) \wedge \neg Pf^9a.$$

But there also exists a much simpler solution; we just take $\theta = [X \backslash \lambda\alpha.A]$ with

$$A = P\alpha \supset Pf^3\alpha.$$

Since the solution for the schematic extended Herbrand sequent is interpreted as the lemmata that give rise to the proof with cuts, and these lemmata will be read and interpreted by humans in applications, it is important to consider the problem of improving the logical and symbolic complexity of the canonical solution. Furthermore, a decrease in the logical complexity of a lemma often yields a decrease in the length of the proof that is constructed from it.

In the following sections, we describe a method which computes small solutions for schematic Herbrand sequents induced by decompositions. The method is incomplete (in the

sense that a solution of minimal complexity is missed) but efficient. It is based on resolution and paramodulation.

We start by investigating the case of a single Π_1-cut (Sect. 5.1). Similar results have been presented already in [19]. For simplicity of presentation we consider a fixed sequent:

$$\mathcal{S} = \forall \bar{x}\, F(\bar{x}) \rightarrow$$

although the results can be extended to more general end-sequents as in Sect. 4. The problem of improving the canonical solution concerns a quantifier-free formula, hence, in the sequel, the variable vector $\bar{\alpha}$ is to be interpreted as a vector of constant symbols. All formulas are quantifier-free unless otherwise noted.

5.1 On the Solutions for a Single Π_1-Cut

We start to study the problem of simplifying the canonical solution by looking at the case of *1-decompositions* $U \circ W$, for

$$U = \{f_1(\overline{u_1}), \ldots, f_1(\overline{u_m})\},\ W = \{\overline{s_1}, \ldots, \overline{s_k}\},$$

which gives rise to proofs with a single Π_1-cut. In the setting of 1-decompositions, an arbitrary solution is of the form $[X\backslash\lambda\bar{\alpha}.A]$. Throughout this section, we consider a fixed 1-decomposition $U \circ V$, along with the solution conditions $\mathcal{I}$

$$\begin{array}{l} \Gamma \rightarrow X\bar{\alpha}, \\ \Gamma', X\overline{s_1}, \ldots, X\overline{s_k} \rightarrow, \end{array}$$

for $\Gamma = F[\bar{x}\backslash\overline{u_1}], \ldots, F[\bar{x}\backslash\overline{u_m}]$ and Γ' being a subset of Γ. We also consider the canonical solution

$$\sigma = [X\backslash\lambda\bar{\alpha}.C] = [X\backslash\lambda\bar{\alpha}.\bigwedge_{i=1}^{m} F[\bar{x}\backslash\overline{u_i}]].$$

If $[X\backslash\lambda\bar{\alpha}.A]$ is a solution for $\mathcal{I}$, we will say simply that A is a solution.

The first basic observation is that solvability is a semantic property. The following is an immediate consequence of Definition 30.

Lemma 40 *Let A be a solution, B a formula and $\models A \Leftrightarrow B$. Then B is a solution.*

Hence we may restrict our attention to solutions which are in *conjunctive normal form* (CNF). Formulas in CNF can be represented as sets of *clauses*, which in turn are sets of *literals*, i.e. possibly negated atoms. It is this representation that we will use throughout this section, along with the following properties: for sets of clauses A, B, $A \subseteq B$ implies $B \models A$, and for clauses C, D, $C \subseteq D$ implies $C \models D$.

Note that the converse of the Lemma above does not hold: given a solution A there may be solutions B such that $\not\models A \Leftrightarrow B$. We now turn to the problem of finding such solutions. In Example 39, we observe that that $C \models A$ (but $A \not\models C$). We can generalize this observation to show that the canonical solution is most general.

Lemma 41 *Let C be the canonical solution and A an arbitrary solution. Then $C \models A$.*

Proof Since $\vartheta = [X\backslash\lambda\bar{\alpha}.A]$ is a solution for $\mathcal{I}$, the sequent $F[\bar{x}\backslash\overline{u_1}], \ldots, F[\bar{x}\backslash\overline{u_m}], A \supset \bigwedge_{j=1}^{k} A[\bar{\alpha}\backslash\overline{s_j}] \rightarrow$ is E-valid. By definition, $C = \bigwedge_{i=1}^{m} F[\bar{x}\backslash\overline{u_i}]$, and therefore $C, A \supset \bigwedge_{j=1}^{k} A[\bar{\alpha}\backslash\overline{s_j}] \rightarrow$ is E-valid, hence $C \rightarrow A$ is E-valid. □

This result states that any search for simple solutions can be restricted to consequences of the canonical solution. Unfortunately, due to equality in our language, there are infinitely many consequences. Even enumerating all consequences bounded by a fixed bound on symbol size would be computationally infeasible. Towards a more efficient iterative method, we give a criterion that allows us to disregard some of those consequences.

Lemma 42 *If $A \models B$ then*

(1) If $\Gamma', A[\bar{\alpha}\backslash\overline{s_1}], \ldots, A[\bar{\alpha}\backslash\overline{s_k}] \rightarrow$ is not E-valid, then B is not a solution.
(2) If A is a solution then $\Gamma \rightarrow B$ is E-valid.
(3) If A is a solution, then $\Gamma', B[\bar{\alpha}\backslash\overline{s_1}], \ldots, B[\bar{\alpha}\backslash\overline{s_k}] \rightarrow$ is E-valid iff $[X \backslash \lambda\bar{\alpha}.B]$ is a solution of $\mathcal{I}$.

Proof For (1), we will show the contrapositive. By assumption, we have that $\Gamma', B[\bar{\alpha}\backslash\overline{s_1}], \ldots, B[\bar{\alpha}\backslash\overline{s_k}] \rightarrow$ is E-valid. Since $A \models B$, we find that furthermore $\Gamma', A[\bar{\alpha}\backslash\overline{s_1}], \ldots, A[\bar{\alpha}\backslash\overline{s_k}] \rightarrow$ is E-valid. For (2) it suffices to observe that since A is a solution $\Gamma \rightarrow A$ is E-valid, and to conclude by $A \models B$. (3) is then immediate by definition. □

Lemma 43 (Sandwich Lemma) *Let A, B be solutions and $A \models D \models B$. Then D is a solution.*

Proof By Lemma 42 (2), the first solution condition $\Gamma \rightarrow D$ is E-valid. The second solution condition $D[\bar{\alpha}\backslash\overline{s_1}], \ldots, D[\bar{\alpha}\backslash\overline{s_k}], \Gamma \rightarrow$ is E-valid by Lemma 42 (1). □

5.2 Simplification by forgetful inference

In this section we define a method to simplify solutions which is based on resolution and paramodulation. The idea behind it is to generate solutions of smaller size by *forgetful inference*, i.e. if we derive F from F_1, F_2 we replace F_1, F_2 by F. This principle of inference is sound but obviously incomplete. The method is also incomplete in the sense that it might fail to produce the shortest solution; however it proved very useful in practice and is part of our implementation. From now on we assume that the formulas are in clause form, i.e. they are represented as finite sets of clauses (and clauses are considered as finite sets of literals). We may also assume that the clauses are ground (in particular we consider variables from $\bar{\alpha}$ as constants). Therefore the principles of resolution and paramodulation used below do not require unification.

Definition 44 (simplification) Let $\mathcal{C}$ be a set of ground clauses. We define

- $\mathcal{C} \rhd_r \mathcal{C}'$ if $\mathcal{C}' = (\mathcal{C} \setminus \{C_1, C_2\}) \cup \{R\}$, where $C_1, C_2 \in \mathcal{C}$, $C_1 \neq C_2$ and R is a resolvent of C_1 and C_2 which is not a tautology.
- $\mathcal{C} \rhd_p \mathcal{C}'$ if $\mathcal{C}' = (\mathcal{C} \setminus \{C_1, C_2\}) \cup \{R\}$, where $C_1, C_2 \in \mathcal{C}$, $C_1 \neq C_2$ and R is a paramodulant of C_1 and C_2 which is not a tautology.
- $\mathcal{C} \rhd_s \mathcal{C}'$ if either $\mathcal{C} \rhd_r \mathcal{C}'$ or $\mathcal{C} \rhd_p \mathcal{C}'$.

If there exists no $\mathcal{C}'$ s.t. $\mathcal{C} \rhd_s \mathcal{C}'$ we say that $\mathcal{C}$ is in normal form.

The principle defined above simply consists in generating a resolvent or a paramodulant and afterwards deleting the parent clauses. By solving a set of solution conditions with variables $X_1, \ldots, X_n$ we obtain a canonical solution of the form

$$[X_1 \setminus \lambda\overline{\alpha_1}.C_1, \ldots, X_n \setminus \lambda\overline{\alpha_n}.C_n].$$

Then the clause forms $\mathcal{C}_i$ of the formulas C_i represent the i-th cut formula. We represent the cut formulas obtained so far as a tuple $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$. To simplify all the cut formulas we have thus to extend the relation $\rhd_s$ to tuples of clause sets.

Definition 45 Let $(\mathcal{C}_1, \ldots, \mathcal{C}_n)$, $(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ be tuples of clause sets for $n \geq 1$. We define $(\mathcal{C}_1, \ldots, \mathcal{C}_n) \rhd_s (\mathcal{D}_1, \ldots, \mathcal{D}_n)$ if there exists an $i \in \{1, \ldots, n\}$ s.t. $\mathcal{C}_i \rhd_s \mathcal{D}_i$ and for all $j \leq n$ and $j \neq i$ we have $\mathcal{D}_j = \mathcal{C}_j$.

Proposition 46 *$\rhd_s$ is sound, i.e. if $(\mathcal{C}_1, \ldots, \mathcal{C}_n) \rhd_s (\mathcal{D}_1, \ldots, \mathcal{D}_n)$ then, for all $i \in \{1, \ldots, n\}$, $\mathcal{C}_i \models \mathcal{D}_i$.*

Proof By the soundness of resolution and paramodulation over equality interpretations we have that $\mathcal{C} \rhd_r \mathcal{C}'$ ($\mathcal{C} \rhd_p \mathcal{C}'$) implies $\mathcal{C} \models \mathcal{C}'$. □

Proposition 47 *$\rhd_s$ is terminating.*

Proof Assume that $(\mathcal{C}_1, \ldots, \mathcal{C}_n) \rhd_s (\mathcal{D}_1, \ldots, \mathcal{D}_n)$. Then there exists an i such that $\mathcal{C}_i \rhd_s \mathcal{D}_i$ and, by definition of $\rhd_s$, $|\mathcal{C}_i| > |\mathcal{D}_i|$; for $j \neq i$ we have $\mathcal{C}_j = \mathcal{D}_j$. So if we define

$$\|(\mathcal{C}_1, \ldots, \mathcal{C}_n)\| = \sum_{i=1}^{n} |\mathcal{C}_i|$$

we obtain $\|(\mathcal{C}_1, \ldots, \mathcal{C}_n)\| > \|(\mathcal{D}_1, \ldots, \mathcal{D}_n)\|$ and thus $\rhd_s$ is terminating.

Remark 48 $\rhd_s$ is not confluent: consider e.g.

$$\mathcal{C} = \{\{P(\alpha_1), Q(\alpha_1)\},\ \{\neg P(\alpha_1), Q(\alpha_1)\},\ \{\neg Q(\alpha_1)\}\}.$$

Then, clearly, $\mathcal{C} \rhd_s \{\{Q(\alpha_1)\},\ \{\neg Q(\alpha_1)\}\}$ and $\mathcal{C} \rhd_s \{\{P(\alpha_1), Q(\alpha_1)\},\ \{\neg P(\alpha_1)\}\}$. But there exists no $\mathcal{C}'$ s.t.

$$\{\{Q(\alpha_1)\},\ \{\neg Q(\alpha_1)\}\} \rhd_s^* \mathcal{C}' \text{ and } \{\{P(\alpha_1), Q(\alpha_1)\},\ \{\neg P(\alpha_1)\} \rhd_s^* \mathcal{C}'.$$

$\mathcal{C}$ has in fact the two different normal forms $\{\{\}\}$ and $\{\{Q(\alpha_1)\}\}$.

Example 49 Let

$$\mathcal{C}_1 = \{\{\alpha_1 = \alpha_2\},\ \{P(\alpha_1), Q(\alpha_1)\}, \{\neg P(\alpha_1)\}\},$$
$$\mathcal{C}_2 = \{\{f(\alpha_2) = \alpha_2\},\ \{\neg P(f(\alpha_2)), R(\alpha_2)\},\ \{P(\alpha_2)\}\}.$$

Then

$$\mathcal{C}_1 \rhd_p \{\{P(\alpha_2), Q(\alpha_1)\},\ \{\neg P(\alpha_1)\}\},$$
$$\mathcal{C}_1 \rhd_r \{\{\alpha_1 = \alpha_2\}, \{Q(\alpha_1)\}\} \rhd_p \{\{Q(\alpha_2)\}\} \text{ and therefore}$$
$$\mathcal{C}_1 \rhd_s \{\{P(\alpha_2), Q(\alpha_1)\},\ \{\neg P(\alpha_1)\}\},\ \mathcal{C}_1 \rhd_s^* \{\{Q(\alpha_2)\}\}.$$

For $\mathcal{C}_2$ we obtain

$$\mathcal{C}_2 \rhd_p \{\{\neg P(\alpha_2), R(\alpha_2)\},\ \{P(\alpha_2)\}\} \rhd_r \{\{R(\alpha_2)\}\} \text{ and}$$
$$\mathcal{C}_2 \rhd_p \{\{\neg P(f(\alpha_2)), R(\alpha_2)\},\ \{P(f(\alpha_2))\}\} \rhd_r \{\{R(\alpha_2)\}\}.$$

We define a normal form computation on the tuple $(\mathcal{C}_1, \mathcal{C}_2)$:

$$\begin{aligned}(\mathcal{C}_1, \mathcal{C}_2) \rhd_s\ & (\{\{\alpha_1 = \alpha_2\}, \{Q(\alpha_1)\}\}, \mathcal{C}_2) \rhd_s \\ & (\{\{\alpha_1 = \alpha_2\}, \{Q(\alpha_1)\}\}, \{\{\neg P(\alpha_2), R(\alpha_2)\},\ \{P(\alpha_2)\}\}) \rhd_s \\ & (\{\{\alpha_1 = \alpha_2\}, \{Q(\alpha_1)\}\}, \{\{R(\alpha_2)\}\}) \rhd_s (\{\{Q(\alpha_2)\}\}, \{\{R(\alpha_2)\}\}).\end{aligned}$$

We thus get the normal form $(\{\{Q(\alpha_2)\}\}, \{\{R(\alpha_2)\}\})$ of $(\mathcal{C}_1, \mathcal{C}_2)$ under $\rhd_s$. Note that $(\{\{P(\alpha_2), Q(\alpha_1)\},\ \{\neg P(\alpha_1)\}\}, \{\{R(\alpha_2)\}\})$ is another normal form.

Below we define the set of simplified solution tuples for a set of solution conditions.

Definition 50 (Solution tuple) Let $\mathcal{I}$ be a set of solution conditions with variables $X_1, \ldots, X_n$ and let

$$\Theta\colon\ [X_1 \setminus \lambda\overline{\alpha_1}.D_1, \ldots, X_n \setminus \lambda\overline{\alpha_n}.D_n]$$

be a solution of $\mathcal{I}$. Let $\mathcal{D}_i$ be clause forms of D_i for $i = 1, \ldots, n$. Then we call $(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ a *solution tuple* of $\mathcal{I}$. If Θ is the canonical solution we call $(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ the *canonical solution tuple* of $\mathcal{I}$.

Definition 51 (Set of simplified solutions) Let $\mathcal{I}$ be a set of solution conditions. Then we define the set of simplified solutions $Sol_s(\mathcal{I})$ by:

- the canonical solution tuple of $\mathcal{I}$ is in $Sol_s(\mathcal{I})$,
- if $\Psi \in Sol_s(\mathcal{I})$, $\Psi \rhd_s \Psi'$ and Ψ' is a solution tuple of $\mathcal{I}$ then $\Psi' \in Sol_s(\mathcal{I})$.

Proposition 52 *Let $\mathcal{I}$ be a set of solution conditions. Then $Sol_s(\mathcal{I})$ is a finite set of solution tuples of $\mathcal{I}$ and $Sol_s(\mathcal{I})$ is computable.*

Proof $Sol_s(\mathcal{I})$ is finite as, for the canonical solution tuple Ψ_0, there are only finitely many Ψ s.t. $\Psi_0 \rhd_s^* \Psi$ (note that, by Proposition 47, $\rhd_s$ is terminating). It is computable because it is decidable whether a given tuple of clause sets Ψ is a solution tuple of $\mathcal{I}$. □

There are various ways to extract solution tuples from the set $Sol_s(\mathcal{I})$. We can either compute a minimal Ψ, i.e. a $\Psi \in Sol_s(\mathcal{I})$ s.t. either all components of Ψ are in normal form or $\Psi \rhd_s \Psi'$ implies that Ψ' is not a solution anymore. Or we can compute all minimal solution tuples $\Psi \in Sol_s(\mathcal{I})$ and select those of minimal logical complexity.

Our implementation iteratively finds one minimal solution in $\Psi \in Sol_s(\mathcal{I})$: we start from the canonical solution $\Psi = (D_1, \ldots, D_n) \in Sol_s(\mathcal{I})$. We process the components of the tuple from right to left, starting at D_n. In each step we minimize one component of the solution tuple, computing all $\rhd_s$-simplifications, picking one minimal simplification, and replacing that component by the simplification. Performing a simplification at one component preserves the minimality of the components to the right, so we produce a minimal solution after one loop.

There are several heuristics which may further improve the algorithm. One straightforward (but expensive) strategy is to delete a single clause in the clause form and to check whether the formula is still a solution; this feature is built in but is not used in the tests. Another (better) one is to eliminate clauses in the CNF-form which do not contain variables from $\bar{\alpha}$. The example below illustrates advantages and potential problems with this heuristic.

Example 53 Let A be a solution in CNF and construct A' from A by removing all clauses that do not contain variables from $\bar{\alpha}$. Then we have to check whether A' is a solution.

Let $\mathcal{S} = (\forall x.F(x) \to) = \big(\forall x((Pa \wedge (Px \supset Pf(x))) \wedge \neg Pf^3(a)) \to\big)$ and $U \circ W$ a 1-decomposition (of $\{f_1(a), f_1(f(a)), f_1(f^2(a))\}$) for

$$U = \{f_1(\alpha), f_1(f(\alpha))\},\ W = \{a, f(a)\}.$$

and let $\Gamma = F(\alpha), F(f(\alpha))$. Then the corresponding solution system is

$$\begin{array}{l} \Gamma \to X\alpha, \\ Xa, Xf(a) \to . \end{array}$$

The canonical solution is $F(\alpha) \wedge F(f(\alpha))$, its CNF being

$$Pa \wedge (\neg P\alpha \vee Pf(\alpha)) \wedge (\neg Pf\alpha \vee Pf^2(\alpha)) \wedge \neg Pf^3(a).$$

Note that the formula

$$G(\alpha)\colon (\neg P\alpha \vee Pf(\alpha)) \wedge (\neg Pf\alpha \vee Pf^2(\alpha))$$

obtained after removing the α-free clauses is not a solution of $X\alpha, Xf(\alpha) \to !$
However if we choose the logically equivalent version

$$\mathcal{S}'\colon P(a), \forall x(P(x) \supset P(f(x))) \to P(f^3(a))$$

and the same decomposition $U \circ W$ we obtain the solution system

$$P(a), P(\alpha) \supset P(f(\alpha)), P(f(\alpha) \supset P(f^2(\alpha)) \to P(f^3(a)), X\alpha$$
$$P(a), Xa, Xf(a) \to P(f^3(a)).$$

For the system above $G(\alpha)$ is indeed a solution. In the last system α-parts and α-free parts are cleanly separated while in the first one this is not the case. We see that the efficiency of the strategy to eliminate α-free clauses depends on the syntactic form of the problem.

The example below illustrates the procedure of computing a minimal solution from a canonical solution tuple.

Example 54 Let S be the sequent

$$Pa, \forall x.fx = s^3x,\ \forall x(Px \supset P(sx)) \to P(f^3a).$$

S has a Herbrand sequent H for

$$H = \Gamma,\ P(a),\ P(a) \supset P(sa), \ldots, P(s^8a) \supset P(s^9a) \to P(f^3a)$$

where $\Gamma = \{fa = s^3a, f^2a = s^3fa, f^3a = s^3f^2a\}$. The instantiation term set T corresponding to S and H is

$$T\colon \{f_1(a), f_1(fa), f_1(f^2a), f_2(a), \ldots, f_2(f^8a)\}.$$

We define a decomposition D of T by

$$\{f_1(a), f_1(fa), f_1(f^2a),\ f_2(\alpha), f_2(s\alpha), f_2(s^2\alpha)\} \circ \{a, s^3a, s^6a\}.$$

The solution conditions corresponding to D are $\mathcal{I} = \{\mathcal{I}_0, \mathcal{I}_1\}$ for

$$\mathcal{I}_0 = \Gamma, P(a), P(\alpha) \supset P(s\alpha), P(s\alpha) \supset P(s^2\alpha), P(s^2\alpha) \supset P(s^3\alpha) \to X\alpha, P(f^3a),$$
$$\mathcal{I}_1 = \Gamma, P(a), Xs^3a, Xs^6a \to P(f^3a).$$

We have

$$S_U = \Gamma, P(a), P(\alpha) \supset P(s\alpha), P(s\alpha) \supset P(s^2\alpha), P(s^2\alpha) \supset P(s^3\alpha) \to Pf^3a.$$

$\neg S_U$ is the canonical solution; its clause form is $\mathcal{C}$ for

$$\mathcal{C} = \{\{fa = s^3a\}, \{f^2a = s^3fa\}, \{f^3a = s^3f^2a\},$$
$$\{\neg P(\alpha), P(s\alpha)\},\ \{\neg P(s\alpha), P(s^2\alpha)\},\ \{\neg P(s^2\alpha), P(s^3\alpha)\}, \{\neg P(f^3a)\}\}.$$

We are now simplifying the solution via $\rhd_s$:

$$\begin{aligned}
\mathcal{C} = {} & \{\{P(a)\},\ \{fa = s^3a\}, \{f^2a = s^3fa\}, \{f^3a = s^3f^2a\},\\
& \{\neg P(\alpha),\ P(s\alpha)\},\ \{\neg P(s\alpha),\ P(s^2\alpha)\},\ \{\neg P(s^2\alpha),\ P(s^3\alpha)\}, \{\neg P(f^3a)\}\}\rhd_r\\
& \{\{P(a)\},\ \{fa = s^3a\}, \{f^2a = s^3fa\}, \{f^3a = s^3f^2a\},\\
& \{\neg P(\alpha),\ P(s^2\alpha)\},\ \{\neg P(s^2\alpha),\ P(s^3\alpha)\}, \{\neg P(f^3a)\}\}\rhd_r\\
& \{\{P(a)\},\ \{fa = s^3a\},\ \{f^2a = s^3fa\},\ \{f^3a = s^3f^2a\},\\
& \{\neg P(\alpha),\ P(s^3\alpha)\},\ \{\neg P(f^3a)\}\}\rhd_p\\
& \{\{P(a)\},\ \{fa = s^3a\},\ \{f^2a = s^3fa\},\ \{\neg P(\alpha),\ P(s^3\alpha)\},\ \{\neg P(s^3f^2a)\}\}\rhd_p\\
& \{\{P(a)\},\ \{fa = s^3a\},\ \{\neg P(\alpha),\ P(s^3\alpha)\},\ \{\neg P(s^6fa)\}\}\rhd_p\\
& \{\{P(a)\},\ \{\neg P(\alpha),\ P(s^3\alpha)\},\ \{\neg P(s^9a)\}\}.
\end{aligned}$$

$\{\{P(a)\},\ \{\neg P(\alpha),\ P(s^3\alpha)\},\ \{\neg P(s^9a)\}\}$ is a normal form under $\rhd_s$ and yields the cut formula

$$\forall x.(P(a) \wedge (\neg P(x) \vee P(s^3x)) \wedge \neg P(s^9a)).$$

By deleting α-free clauses we obtain the set of clauses $\{\{\neg P(\alpha),\ P(s^3\alpha)\}\}$ which yields the simplified cut formula $\forall x(\neg P(x) \vee P(s^3x))$.

We now illustrate the use of forgetful inference in simplifying a solution for two cut formulas.

Example 55 Consider the sequent

$$S\colon\ Pa, \forall x\,(Px \supset Pfx) \to Pf^8a$$

which has a Herbrand sequent H generated by the instantiating the quantifier of the second formula with $\{a,\ fa,\ f^2a,\ \ldots,\ f^7a\}$. A corresponding decomposition is

$$U \circ_{\alpha_1} S_1 \circ_{\alpha_2} S_2 = \{f_2(\alpha_1),\ f_2(f\alpha_1)\} \circ \{\alpha_2,\ f^2\alpha_2\} \circ \{a,\ f^4a\}.$$

For S and this decomposition we obtain

$$\begin{aligned}
S_U &= Pa,\ P\alpha_1 \supset Pf\alpha_1,\ Pf\alpha_1 \supset Pf^2\alpha_1 \to Pf^8a,\\
S^1_U &= S_U,\\
S^2_U &= Pa \to Pf^8a.
\end{aligned}$$

The set of solution conditions is $\mathcal{I}\colon\ \{\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3\}$ for

$$\begin{aligned}
\mathcal{I}_0 &= Pa,\ P\alpha_1 \supset Pf\alpha_1,\ Pf\alpha_1 \supset Pf^2\alpha_1 \to Pf^8a,\ X_1\alpha_1,\ X_2\alpha_2,\\
\mathcal{I}_1 &= Pa,\ X_1\alpha_2,\ X_1f^2\alpha_2 \to Pf^8a,\ X_2\alpha_2,\\
\mathcal{I}_3 &= Pa,\ X_2a,\ X_2f^4a \to Pf^8a.
\end{aligned}$$

The canonical solution of $\mathcal{I}$ is

$$[X_1 \setminus \lambda\alpha_1.C_1,\ X_2 \setminus \lambda\alpha_2.C_1[\alpha_1\backslash\alpha_2] \wedge C_1[\alpha_1\backslash f^2\alpha_2]]$$

for $C_1 = \neg(Pa \wedge (P\alpha_1 \supset Pf\alpha_1) \wedge (Pf\alpha_1 \supset Pf^2\alpha_1) \supset Pf^8a)$.
We construct the clause forms $\mathcal{C}_1$ for C_1 and $\mathcal{C}_2$ for $C_1[\alpha_1\backslash\alpha_2] \wedge C_1[\alpha_1\backslash f^2\alpha_2]$:

$$\mathcal{C}_1 = \{\{\neg P\alpha_1, Pf\alpha_1\},\ \{\neg Pf\alpha_1, Pf^2\alpha_1\},\ \{Pa\},\ \{\neg Pf^8a\}\},$$
$$\mathcal{C}_2 = \{\{\neg P\alpha_2, Pf\alpha_2\},\ \{\neg Pf\alpha_2, Pf^2\alpha_2\},\ \{Pa\},\ \{\neg Pf^8a\},\ \{\neg Pf^2\alpha_2, Pf^3\alpha_2\},\ \{\neg Pf^3\alpha_2, Pf^4\alpha_2\}\}.$$

Therefore the corresponding canonical solution tuple is $(\mathcal{C}_1, \mathcal{C}_2)$ and $(\mathcal{C}_1, \mathcal{C}_2) \in Sol_s(\mathcal{I})$. Now we get $(\mathcal{C}_1, \mathcal{C}_2) \rhd_r (\mathcal{C}'_1, \mathcal{C}_2)$ for

$$\mathcal{C}'_1 = \{\{\neg P\alpha_1, Pf^2\alpha_1\},\ \{Pa\},\ \{\neg Pf^8a\}\}.$$

But $(\mathcal{C}'_1, \mathcal{C}_2)$ is not a solution tuple for $\mathcal{I}$ as $\mathcal{I}_1$ is not valid under the corresponding substitution.

The right way to proceed is to simplify the solution for X_2 first and then that for X_1. So we compute $\mathcal{C}_2 \rhd_r \mathcal{C}'_2$ for

$$\mathcal{C}'_2 = \{\{\neg P\alpha_2, Pf^2\alpha_2\},\ \{\neg Pf^2\alpha_2, Pf^3\alpha_2\},\ \{\neg Pf^3\alpha_2, Pf^4\alpha_2\},\ \{Pa\},\ \{\neg Pf^8a\}\}.$$

It is easy to check that $(\mathcal{C}_1, \mathcal{C}'_2) \in Sol_s(\mathcal{I})$. Now we define

$$\mathcal{C}''_2 = \{\{\neg P\alpha_2, Pf^2\alpha_2\},\ \{\neg Pf^2\alpha_2, Pf^4\alpha_2\},\ \{Pa\},\ \{\neg Pf^8a\}\},$$
$$\mathcal{C}^3_2 = \{\{\neg P\alpha_2, Pf^4\alpha_2\},\ \{Pa\},\ \{\neg Pf^8a\}\}.$$

Then $(\mathcal{C}_1, \mathcal{C}'_2) \rhd_s (\mathcal{C}_1, \mathcal{C}''_2) \rhd_s (\mathcal{C}_1, \mathcal{C}^3_2)$. Moreover, $(\mathcal{C}_1, \mathcal{C}''_2) \in Sol_s(\mathcal{I})$ and $(\mathcal{C}_1, \mathcal{C}^3_2) \in Sol_s(\mathcal{I})$. Indeed we can easily check that

$$[X_1 \setminus \lambda\alpha_1.\mathcal{C}_1,\ X_2 \setminus \lambda\alpha_2.\mathcal{C}^3_2]$$

is a solution of $\mathcal{I}$.

Now we already know that $\mathcal{C}_1 \rhd_r \mathcal{C}'_1$ and we compute $(\mathcal{C}_1, \mathcal{C}^3_2) \rhd_s (\mathcal{C}'_1, \mathcal{C}^3_2)$, and (this time) $(\mathcal{C}'_1, \mathcal{C}^3_2) \in Sol_s(\mathcal{I})$. Neither $\mathcal{C}'_1$ nor $\mathcal{C}^3_2$ can be simplified further and we obtain a minimal solution (a normal form under $\rhd_s$). This solution yields the cut formulas

$$\forall x((\neg Px \vee Pf^2x) \wedge Pa \wedge \neg Pf^8a) \text{ and}$$
$$\forall x((\neg Px \vee Pf^4x) \wedge Pa \wedge \neg Pf^8a)$$

for the proof with cut. A further simplification via elimination of α-free clauses would result in the cut formulas $\forall x(\neg Px \vee Pf^2x)$ and $\forall x(\neg Px \vee Pf^4x)$.

Remark 56 Example 55 shows that the simplification must start from the "rear", i.e. we must first simplify the solution for X_n, then that for X_{n-1} and so on. The reason is that the simplified formula may be logically weaker and the best place to insert a weaker cut is at the lowermost cut; here only the right-hand side of the lowermost cut (that means the last solution condition) has to be checked accordingly. This order of simplification is also implemented and used for te tests.

5.3 Beautifying the Solution

The minimization procedure defined above takes a solution in conjunctive normal form and combines some of the clauses into new clauses. These new clauses form the actual non-analytic content of the lemma that we generate. However, there can be parts of the CNF that the minimization procedure did not modify—these unmodified parts are then just instances of formulas in the end-sequent. In addition, some clauses of the minimized solution may contain literals that already occur in the end-sequent and are hence always true.

Example 57 Let us look again at Example 28 from Sect. 4. We had a proof of the following sequent:

$$Pc, \forall x(Px \supset Psx) \to Ps^6c$$

And we obtained the following decomposition D together with the canonical substitution $[X \backslash \lambda\alpha\ C_1]$, from which we got the minimized solution C_1':

$$D = U \circ S = \{f_1, f_2\alpha, f_2s\alpha, f_2s^2\alpha, f_3\} \circ \{c, s^3c\}$$
$$C_1 = Pc \wedge (P\alpha \supset Ps\alpha) \wedge (Ps\alpha \supset Ps^2\alpha) \wedge (Ps^2\alpha \supset Ps^3\alpha) \wedge \neg Ps^6c$$
$$C_1' = Pc \wedge (P\alpha \supset Ps^3\alpha) \wedge \neg Ps^6c$$

However, while minimization managed to simplify the part of the solution that contains the implications, the two literals Pc and $\neg Ps^6c$ still remain unmodified.

In contrast to solution minimization, we will not only modify the solution, but the decomposition as well. We will first define the operations on the solution, and then show their effect on the decomposition.

Definition 58 *(Beautification)* Let $\mathcal{S}$ be a Σ_1-sequent. We define $\rhd_{as}^{\mathcal{S}}$ and $\rhd_{ur}^{\mathcal{S}}$ as the smallest relations on sets of clauses satisfying the following:

- $\mathcal{C} \cup \{C\} \rhd_{as}^{\mathcal{S}} \mathcal{C}$ if C is subsumed by a clause in the CNF of $\neg\mathcal{S}$ ("axiom subsumption")
- $\mathcal{C} \cup \{C \cup \{l\}\} \rhd_{ur}^{\mathcal{S}} \mathcal{C} \cup \{C\}$ if $\neg l$ is subsumed by a clause in the CNF of $\neg\mathcal{S}$ ("unit resolution")

We then extend these relations to beautification of solutions, defining $\rhd_b^{\mathcal{S}}$ to be the smallest relation such that:

- $(\mathcal{C}_1, \ldots, \mathcal{C}_i, \ldots, \mathcal{C}_n) \rhd_b^{\mathcal{S}} (\mathcal{C}_1, \ldots, \mathcal{C}_i', \ldots, \mathcal{C}_n)$ if $\mathcal{C}_i \rhd_{as}^{\mathcal{S}} \mathcal{C}_i'$ or $\mathcal{C}_i \rhd_{ur}^{\mathcal{S}} \mathcal{C}_i'$,
- $(\mathcal{C}_1, \ldots, \mathcal{C}_{i-1}, \{\}, \mathcal{C}_{i+1}, \ldots, \mathcal{C}_n) \rhd_b^{\mathcal{S}} (\mathcal{C}_{i+1}, \ldots, \mathcal{C}_n)$, and
- $(\mathcal{C}_1, \ldots, \mathcal{C}_{i-1}, \{\{\}, \ldots\}, \mathcal{C}_{i+1}, \ldots, \mathcal{C}_n) \rhd_b^{\mathcal{S}} (\mathcal{C}_1, \ldots, \mathcal{C}_{i-1}, \mathcal{C}_{i+1}, \ldots, \mathcal{C}_n)$.

Example 59 We have the solution in CNF $F = (\mathcal{C}_1)$ where

$$\mathcal{C}_1 = \{\{Pc\}, \{\neg P\alpha, Ps^3\alpha\}, \{\neg Ps^6c\}\}$$

Here we can apply axiom subsumption twice:

$$\mathcal{C}_1 \rhd_{as}^{\mathcal{S}} \{\{\neg P\alpha, Ps^3\alpha\}, \{\neg Ps^6c\}\} \rhd_{as}^{\mathcal{S}} \{\{\neg P\alpha, Ps^3\alpha\}\}$$

On the level of the solution we have $F \rhd_b^{\mathcal{S}} (\{\{\neg P\alpha, Ps^3\alpha\}\})$.

Lemma 60 *Let $\mathcal{S}$ be a Σ_1-sequent. Let F be a solution in CNF for a decomposition D of a term set corresponding to a Herbrand sequent of $\mathcal{S}$. If $F \rhd_b^{\mathcal{S}} F'$, then there exists a decomposition D' corresponding to a potentially different Herbrand sequent of $\mathcal{S}$ such that F' is a solution for D'.*

Proof Let $F = (\mathcal{C}_1, \ldots, \mathcal{C}_i, \ldots, \mathcal{C}_n)$, $F' = (\mathcal{C}_1, \ldots, \mathcal{C}_i', \ldots, \mathcal{C}_n)$, and $D = U \circ S_1 \circ \cdots \circ S_n$. Depending on the operation, we will add new elements to U. That is, we construct a decomposition $D' = U' \circ S_1 \circ \cdots \circ S_n$ such that all the solution conditions are satisfied for D' and F'.

First, consider the case that $\mathcal{C}_i = \mathcal{C} \cup \{C \cup \{l\}\} \rhd_{ur}^{\mathcal{S}} \mathcal{C} \cup \{C\} = \mathcal{C}_i'$. Let u be a term that describes an instance I of a formula in $\mathcal{S}$ such that I implies $\neg l$, and set $U' = U \cup \{u\}$. Assume without loss of generality that this instance is in the antecedent. Now we have $I \models \mathcal{C}_i \supset \mathcal{C}_i'$

and $\models \mathcal{C}_i'\sigma \supset \mathcal{C}_i\sigma$ for any substitution σ. This implies that the solution conditions are still satisfied.

Now consider the case that $\mathcal{C}_i = \mathcal{C}_i' \cup \{C\} \rhd_{as}^{\mathcal{S}} \mathcal{C}_i'$. Let u be a term that describes an instance I of a formula in $\mathcal{S}$ such that I implies C, and set $U' = U \cup (u \circ S_i)$. Again assume without loss of generality that the instance is in the antecedent. Here we have $\models \mathcal{C}_i \supset \mathcal{C}_i'$, and $I\sigma \models \mathcal{C}_i'\sigma \supset \mathcal{C}_i\sigma$ for every $\sigma \in S_i$, hence the solution conditions are satisfied as well. □

Example 61 After applying axiom subsumption on the clauses $\{Pc\}$ and $\{\neg Ps^6 c\}$, we need to add the instances f_1 and f_3 to U. Since these are already present, the decomposition does not change.

Starting from the minimized solution C_0, we obtain the beautified solution by computing a C_b such that $C_0 (\rhd_b^{\mathcal{S}})^* C_b$, and C_b cannot be further beautified. We achieve this by exhaustively reducing the solution using $\rhd_b^{\mathcal{S}}$.

Lemma 62 *Let $\mathcal{S}$ be a Σ_1-sequent. We define the complexity of a solution $S = (\mathcal{C}_1, \ldots, \mathcal{C}_n)$ to be the number of literals, clauses, and formulas contained in the solution: $\|S\| = \sum_{i=1}^{n}(1 + \sum_{C \in \mathcal{C}_i}(1 + |C|))$. Then $\rhd_b^{\mathcal{S}}$ strictly decreases the complexity of the solution, and is hence strongly normalizing.*

Proof In each reduction, we either remove a literal, a whole clause, or a formula. □

As a concrete strategy, we first apply axiom subsumption, then unit resolution, and at the end use the rules for {} and {{}}.

6 Implementation and Experiments

Summing up the previous sections, the structure of our lemma generation method is shown in Algorithm 2. We have developed an implementation of the lemma generation method in GAPT, an open-source framework for proof transformations, available at https://logic.at/gapt, see [14] for a system description. We will now present both a concrete example, as well as the results of applying our method to the extensive TSTP library of proofs generated by automated theorem provers.

Algorithm 2 Cut-Introduction

Require: π: cut-free proof
  $T \leftarrow$ extractTermSet(π)
  $D \leftarrow$ computeDecomposition(T)
  $F(\bar{x}) \leftarrow$ canonicalSolution(D)
  $F(\bar{x}) \leftarrow$ minimizeSolution($F(\bar{x})$)
  $F(\bar{x}) \leftarrow$ beautifySolution($F(\bar{x})$)
  return constructProof($F(\bar{x})$)

6.1 Lattices

It is well known that lattices can be defined in two equivalent ways: either as an algebraic structure with the operations meet and join, or as a type of partial order. In this section we will generate a lemma about one direction of this equivalence: starting from a definition of a lattice as an algebraic structure, we will generate the transitivity and anti-symmetry of the order as a lemma. This lemma will be introduced as a cut with the formula into a proof of the following sequent $\mathcal{S}$:

$$\begin{aligned}
&\forall x\ x = x,\\
&\forall x \forall y \forall z\ (x = y \wedge y = z \supset x = z),\\
&\forall x \forall y\ (x = y \supset y = x),\\
&\forall x \forall y \forall u \forall v\ (x = y \wedge u = v \supset f(x,u) = f(y,v)),\\
&\forall x \forall y \forall z\ f(f(x,y),z) = f(x,f(y,z)),\\
&\forall x \forall y\ f(x,y) = f(y,x)\\
&\quad \rightarrow f(a,b) = a \wedge f(b,c) = b \wedge f(c,d) = c \wedge f(d,a) = d \supset a = b \wedge b = c \wedge c = d
\end{aligned}$$

The function symbol f denotes the meet, i.e. the greatest lower bound of two elements. Hence this sequent states that whenever there is a cycle of four elements a, b, c, and d, where each is smaller or equal to the next one, then all must be equal. The standard definition of the partial order of a lattice in terms of its meet operation is $x \leq y$ iff $f(x, y) = x$. Proving the above sequent is the special case $n = 4$ of Exercise 2 in Birkhoff's classic textbook on lattice theory [6]. When expressed in terms of the partial order it is a very natural statement—"the partial order is acyclic"—with a very natural proof: suppose it is not, then transitivity and anti-symmetry lead to a contradiction. In the above sequent we phrase this statement in terms of the meet operation and show how our algorithm expresses the notion of partial order in terms of the algebraic operations.

We start with a manually formalized proof of $\mathcal{S}$.[1] As in the sketched solution to the textbook exercise, this proof first shows transitivity, then anti-symmetry, and finally concludes that there exists no cycle of length 4. We run our algorithm on the Herbrand structure of this proof after cut-elimination. The algorithm will recover the two lemmas from just the information contained in the Herbrand sequent. This case study thus demonstrates how lemmas can be reflected in the (term-)structure of a Herbrand sequent obtained from eliminating these lemmas.

The Herbrand sequent $\mathcal{S}^*$ of this cut-free proof has the instantiation complexity $|\mathcal{S}^*|_i = 144$, and the extracted term set T contains $|T| = 52$ terms. In order to find a decomposition of T, we then apply the Δ-table algorithm described in Sect. 3.2, with row-merging enabled and an additional small modification: for performance reasons, we remove all entries of the Δ-table with more than 3 variables—these entries correspond to cuts with more than 3 quantifiers. The algorithm produces the following decomposition $D = U \circ S$ of size $|D| = 28$:

$$\begin{aligned}
U = \{&f_1(\alpha_2), \quad f_2(f(f(\alpha_1,\alpha_2),\alpha_3), f(\alpha_1, f(\alpha_2,\alpha_3)), \alpha_1),\\
&f_2(f(\alpha_1, f(\alpha_2,\alpha_3)), f(\alpha_1,\alpha_2), \alpha_1), \quad f_2(f(\alpha_1,\alpha_2), f(\alpha_2,\alpha_1), \alpha_2),\\
&f_2(f(\alpha_1,\alpha_3), f(f(\alpha_1,\alpha_2),\alpha_3), \alpha_1), \quad f_2(\alpha_1, f(\alpha_1,\alpha_2), \alpha_2),\\
&f_3(f(f(\alpha_1,\alpha_2),\alpha_3), f(\alpha_1,\alpha_3)), \quad f_3(f(\alpha_3,\alpha_1),\alpha_3),\\
&f_4(f(\alpha_1,\alpha_2),\alpha_1,\alpha_3,\alpha_3), \quad f_4(\alpha_1,\alpha_1, f(\alpha_2,\alpha_3),\alpha_2),\\
&f_5(\alpha_1,\alpha_2,\alpha_3), \quad f_6(\alpha_3,\alpha_1), \quad f_7\}
\end{aligned}$$

$$S = \left\{ \begin{pmatrix} a \\ b \\ c \end{pmatrix}, \begin{pmatrix} b \\ c \\ a \end{pmatrix}, \begin{pmatrix} c \\ a \\ b \end{pmatrix}, \begin{pmatrix} c \\ d \\ a \end{pmatrix}, \begin{pmatrix} d \\ a \\ c \end{pmatrix} \right\}$$

From this decomposition we can already see that we will obtain a lemma with three universal quantifiers. We now compute the canonical substitution and minimize it as in Sects. 4 and 5.

[1] As of GAPT 2.2 this proof is included in `examples/poset/poset-proof.scala`, and `examples/poset/deltatable.scala` contains a script that performs cut-introduction on that proof.

We treat $=$ as an uninterpreted predicate symbol, i.e. we do not apply forgetful paramodulation in the minimization procedure. This results in the following (already minimized) solution for the decomposition D:

$$
\begin{aligned}
&(f(\alpha_3, \alpha_1) = \alpha_3 \supset \alpha_3 = f(\alpha_3, \alpha_1)) \wedge \\
&f(\alpha_3, \alpha_1) = f(\alpha_1, \alpha_3) \wedge \\
&(f(\alpha_1, \alpha_2) = f(\alpha_2, \alpha_1) \wedge f(\alpha_2, \alpha_1) = \alpha_2 \wedge \alpha_1 = f(\alpha_1, \alpha_2) \supset \alpha_1 = \alpha_2) \wedge \\
&(f(\alpha_2, \alpha_3) = \alpha_2 \wedge f(\alpha_1, \alpha_2) = \alpha_1 \wedge \alpha_1 = \alpha_1 \wedge \alpha_3 = \alpha_3 \supset f(\alpha_1, \alpha_3) = \alpha_1) \wedge \\
&\alpha_2 = \alpha_2
\end{aligned}
$$

In this solution we can already vaguely identify transitivity and anti-symmetry of the partial order: line 4 expresses the transitivity and line 3 expresses the anti-symmetry. However there are still superfluous assumptions and direct copies of axioms included in the solution. Applying beautification (Sect. 5.3) removes them, and we obtain the final solution:

$$
\begin{aligned}
&(f(\alpha_2, \alpha_1) = \alpha_2 \wedge \alpha_1 = f(\alpha_1, \alpha_2) \supset \alpha_1 = \alpha_2) \wedge \\
&(f(\alpha_2, \alpha_3) = \alpha_2 \wedge f(\alpha_1, \alpha_2) = \alpha_1 \supset f(\alpha_1, \alpha_3) = \alpha_1)
\end{aligned}
$$

While beautification improved the legibility of the generated lemma, it increased the size of the decomposition from 28 to 44.

6.2 Large-scale Experiments

To demonstrate the wide applicability of our method, we have evaluated Algorithm 2 on a large data set of automatically generated proofs. The TSTP library (Thousands of Solutions from Theorem Provers, see [33]) contains proofs from a variety of automated theorem provers. We selected the first-order proofs (FOF and CNF) as of November 2015, consisting of a total of 138005 proofs. Of these proofs in the TSTP, GAPT can import 68198 proofs (49.41%) as Herbrand structures. The other proofs could not be imported because they use custom proof formats, do not contain any detailed proof at all, contain cyclic inferences, or because they use other unsupported or unsound inference rules. The imported proofs were produced by superposition- and connection-based provers. Of these Herbrand structures, 32714 are trivial: each term has a different root symbol—that is, each formula in the end-sequent is instantiated at most once. Our method cannot generate lemmas for these trivial proofs.

We evaluated our lemma generation method on the remaining 35480 proofs and several different methods to generate decompositions: the Δ-table algorithm for a single variable, and many variables with and without row merging, as well as the so-called MaxSAT-algorithm of [12] for different parameters.

We ran each of the combinations with a timeout of one minute. The computation was performed on a Debian Linux system with an Intel i5-4570 CPU and 8 GiB RAM. Running 4 processes in parallel, the total runtime amounted to 31 days. Of the 35480 non-trivial proofs, we could generate decompositions for 19122 proofs (53.90%), resulting in 12035 lemmas (i.e. beautified solutions, making up 33.92% of the non-trivial proofs).

The first step in Algorithm 2 is to extract the term set of the proof—in the implementation this is part of the proof import. The second step is then the computation of a decomposition. Fig. 1 shows a so-called cactus plot [2] of the performance of the different algorithms that we tested: for each of the algorithms we sorted the CPU runtime of the decomposition

[2] Cactus plots have been popularized by the SAT community to visualize the performance of different solvers on a benchmark set, and have since also been adopted by other competitions.

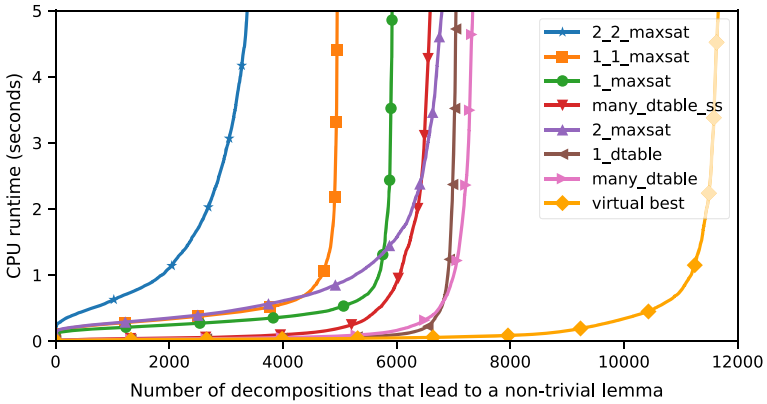

Fig. 1 Comparison of the different algorithms used to generate decompositions. The label `2_2_maxsat` refers to the algorithm of [12] with $|\bar{\alpha_1}| = 2$ and $|\bar{\alpha_2}| = 2$, `many_dtable` is the one from Sect. 3.2, `many_dtable_ss` includes the row-merging modification, and `1_dtable` is the variant of the Δ-table algorithm that uses the Δ_1-vector

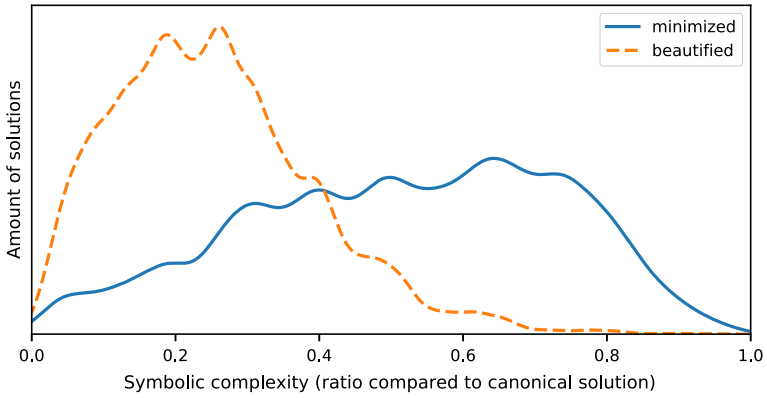

Fig. 2 Symbolic complexity of improved and beautified solutions compared to the canonical solution

generation phase in ascending order, and then plotted the n-th runtime at $x = n$. In short, the lower and righter a line, the better. We only selected those proofs where we could actually generate a non-trivial lemma (and did not fail due to timeouts or beautification detecting a trivial lemma). Judging by the number of decompositions computed that lead to non-trivial lemmas, the best-performing algorithm was the Δ-table algorithm as described in Sect. 3.2. The MaxSAT-algorithm from [12], finding a single cut with 2 quantifiers, came in as a close second, although with a much higher constant overhead. Additional modifications to the Δ-table algorithm (single variable, row merging) did not increase the number of decompositions that could be computed. The orange line on the very right is a "virtual best" algorithm that always picks the fastest one (as in a portfolio). The gap to the other algorithms shows that while they can compute a similar *number* of decompositions, they succeed on classes of proofs with little overlap.

The next big step is the improvement and beautification of the solution. Figure 2 shows the change of symbolic complexity when going from canonical solution to improved solution and finally to the beautified solution. As the size of the canonical solution varies widely

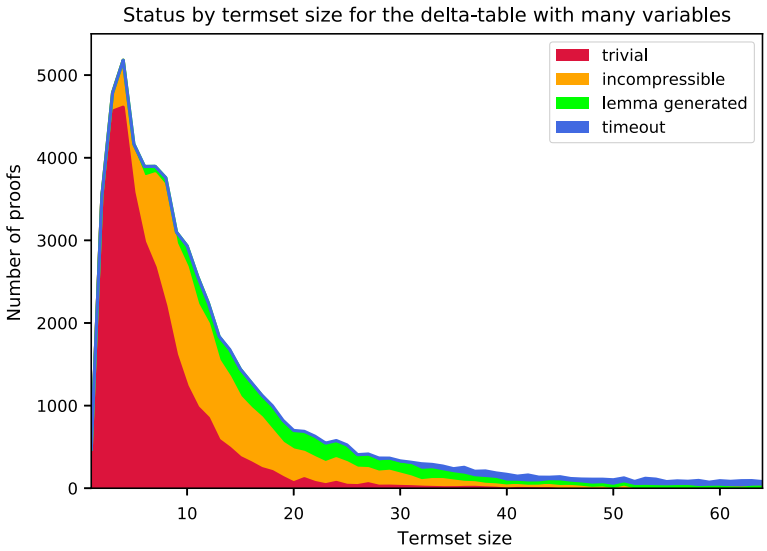

Fig. 3 Result of the algorithm depending on the termset size

depending on the size of the decomposition, we have normalized the symbolic complexity of the improved and beautified solutions by the symbolic complexity of the canonical solution. We also only show data for proofs where we could actually compute a non-trivial beautified solution. Improvement by itself only manages to reduce the size of the canonical solution in some cases, many solutions are irreducible. However improvement plants the seed for beautification to significantly reduce the size of the solution: after beautification, the typical solution is only a third of the size the canonical solution. During beautification, the size of the decomposition increased on average by 10. This is a small increase compared to the size of the decomposition.

It is hard to measure the effect of the algorithm on proof size. For one, we cannot fairly compare the size of the input proofs in the TSTP to the proofs with cut—simply because they are proofs in different calculi. The proofs in the TSTP are typically resolution proofs, while we produce proofs in LK that are cut-free except for the cuts we introduce. When we compare the produced proofs with cut-free proofs in LK, then we actually observe an *increase* in proof size. We used the GAPT tableau prover to generate cut-free proofs of the Herbrand sequents (this is the same prover used to generate the cut-free sub-proofs in the proofs with cut). The proofs with cut are typically 1.5 times longer than the cut-free ones.

To judge the overall results of the algorithm, Fig. 3 then shows for how many proofs from the TSTP data set we successfully generate lemmas, grouped by termset size. Many of the proofs with termsets of size 10 or less are trivial. In a trivial termset, each term has a different root symbol—every quantified formula is instantiated at most once. In this case we cannot find a smaller decomposition, and hence cannot generate lemmas. Incompressible termsets are then those for which the algorithm does not find a compressing decomposition due to other reasons. The algorithm most successfully generates lemmas for proofs with termsets of size between 10 and 50 (Fig. 4).

The theoretical motivation behind our approach is the observation that good lemmas produce small proofs. From the point of view of quantifier complexity, this observation states that good lemmas correspond to small decompositions. Hence it makes sense to evaluate

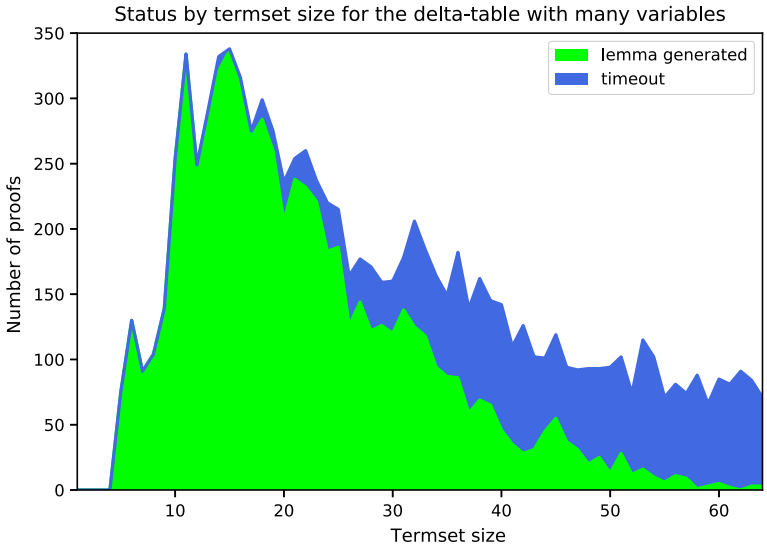

Fig. 4 Result of the algorithm depending on the termset size, ignoring trivial and incompressible term sets

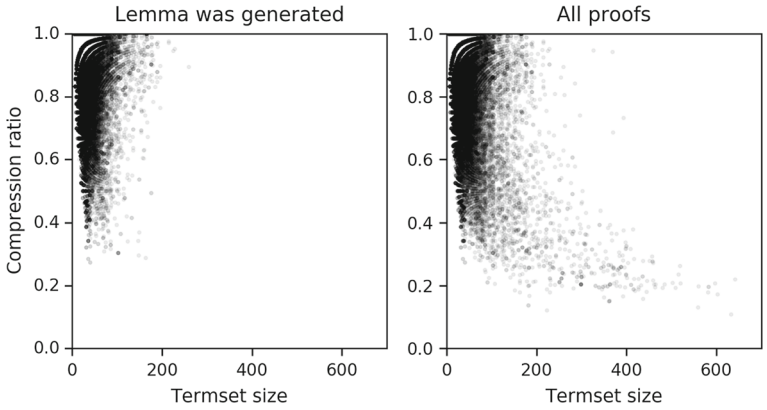

Fig. 5 Compression ratio (size of decomposition divided by size of termset) depending on the termset size

how small the decompositions are that the algorithm produces. Figure 5 shows the achieved compression ratio on the TSTP data set, grouped by termset size; on the right we see just the results from the decomposition phase, on the left we see only those decompositions for which the algorithm did in fact generate a lemma. (The discrete lines are due to the fact that the termset and decomposition size are both small natural numbers.) As observed before, we generate most lemmas for termset sizes between 10 and 50—this is also evident from the left plot. Here we often attain a compression ratio of 0.5, that is, the decomposition is half the size of the termset. Comparing the left and right side, we notice that there is a large number of proofs where we manage to find a decomposition but could not generate a lemma. These are large proofs with termset sizes of more than 100. Nevertheless the decomposition phase results in an even greater compression than for the small proofs. We believe that this gap between found decompositions and generated lemmas is due to the exponential size of the canonical solution that is necessary to generate the lemma.

Table 1 Examples of automatically generated lemmas

Problem	Prover	Generated lemma
SET190-6	E	$complement(complement(x)) = x$
SET047-5	Metis	$set\_equal(x_2, x_1) \supset set\_equal(x_1, x_2)$
SET175+3	SInE	$(x_1 \cap x_2 \subseteq x_3 \vee sk(x_1 \cap x_2, x_3) \in x_2) \wedge$ $(sk(x_2, x_1 \cap x_2) \in x_1 \wedge sk(x_2, x_1 \cap x_2) \in x_2 \supset x_2 \subseteq x_1 \cap x_2)$
PUZ007-1	E	$female(x) \supset from\_mars(x) \vee truthteller(x)$
RNG119+1	E	$aElement Of 0(x_2, x_1) \wedge aIdeal0(x_1) \supset aElement0(x_2)$
GRP040-3	SNARK	$subgroup\_member(identity) \wedge subgroup\_member(x) \supset$ $subgroup\_member(inverse(x))$
SEU154+1	Prover9	$\neg in(x, empty\_set)$

Table 1 shows a few examples of lemmas that were automatically generated from proofs in the TSTP data set. Our method finds purely equational lemmas, as well as propositionally more complex lemmas.

7 Conclusion and Future Work

We have presented an algorithm for the generation of quantified lemmas and evaluated its implementation. The algorithm takes an analytic proof in the form of a Herbrand-sequent as input and creates a sequent calculus proof with Π_1-cuts. It is complete in the sense that it permits a reversal of any cut-elimination sequence [18]. This algorithms shows that, not only does the structure of an analytic proof reflect lemmas of non-analytic proofs of the same theorem, but the latter can be reconstructed from the former.

The evaluation of the implementation in the GAPT-system has demonstrated that it is sufficiently efficient to be applied to proofs generated by automated theorem provers. We have demonstrated it on a case study generating the essential conditions of the definition of a partial order from a proof formulated in the language of lower semilattices.

This algorithm opens up a number of perspectives for future research: it is of proof-theoretic as well as of practical interest to obtain a better understanding of the structural differences between cut-free proofs generated by theorem provers and cut-free proofs generated by cut-elimination, in particular: which strategies of theorem provers are likely to generate proofs which have a structure similar to those obtained by cut-elimination (and hence permit a significant compression by our method)? Can we modify a given cut-free proof in order to adjust the structure to a more regular one, e.g., by factoring out certain background theories?

We also consider it an interesting foundational endeavor to carry out further case studies along the lines of that in Sect. 6.1 motivated by the question mentioned in the introduction: which central mathematical notions can be justified based on grounds of proof-complexity alone (as opposed to human legibility of proofs)?

The algorithm for lemma generation described in this paper has been extended to a method for inductive theorem proving in [13]. In [13], the generated non-analytic formula is the induction invariant. Since the primary goal is to find any inductive proof, concerns about the legibility of proofs as addressed in Sect. 5 become secondary.

Last but not least, we plan to extend the method presented here to cuts with quantifier alternations. There is a satisfactory understanding of the shape of decompositions of more

complex cuts, see [1–4]. The central theoretical problem for an extension in this direction is the question if every such—more complex—decomposition has a canonical solution. In [22], this question has been solved negatively for first-order logic without equality and a partial algorithm for the introduction of a Π_2-cut which is capable of exponential compression has been given. However, for first-order logic with equality, the question remains open.

Acknowledgements Open access funding provided by TU Wien (TUW). This work is supported by the Vienna Science and Technology Fund (WWTF) project VRG12-004.

References

1. Afshari, B., Hetzl, S., Leigh, G.E.: Herbrand disjunctions, cut elimination and context-free tree grammars. In: Altenkirch, T., (ed.) International Conference on Typed Lambda Calculi and Applications (TLCA) 2015, LIPIcs, vol. 38, pp. 1–16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)
2. Afshari, B., Hetzl, S., Leigh, G.E.: Herbrand confluence for first-order proofs with Π_2-cuts. In: Probst, D., Schuster, P. (eds.) Concepts of Proof in Mathematics, Philosophy, and Computer Science, pp. 5–40. De Gruyter, Berlin (2016)
3. Afshari, B., Hetzl, S., Leigh, G.E.: On the Herbrand content of LK. In: Kohlenbach, U., van Bakel, S., Berardi,S., (eds.) 6th International Workshop on Classical Logic and Computation (CL&C 2016), EPTCS, vol. **213**, pp. 1–10 (2016)
4. Afshari, B., Hetzl, S., Leigh G.E.: Herbrand's Theorem as Higher-Order Recursion. Preprint OWP-2018-01, Mathematisches Forschungsinstitut Oberwolfach (2018)
5. Baaz, M., Zach, R.: Algorithmic structuring of cut-free proofs. In: Computer Science Logic (CSL) 1992. Lecture Notes in Computer Science, vol. 702, pp. 29–42. Springer (1993)
6. Birkhoff, G.: Lattice Theory, American Mathematical Society Colloquium Publications, vol. XXV, 3rd edn. American Mathematical Society, Providence (1967)
7. Bundy, A.: The automation of proof by mathematical induction. In: Voronkov, A., Robinson, J.A. (eds.) Handbook of Automated Reasoning, pp. 845–911. Elsevier, Amsterdam (2001)
8. Bundy, A., Basin, D., Hutter, D., Ireland, A.: Rippling: Meta-Level Guidance for Mathematical Reasoning, Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge (2005)
9. Cavagnetto, S.: The lengths of proofs: Kreisel's conjecture and Gödels speed-up theorem. J. Math. Sci. **158**(5), 689–707 (2009)
10. Colton, S.: Automated theory formation in pure mathematics. Ph.D. thesis, University of Edinburgh (2001)
11. Colton, S.: Automated Theory Formation in Pure Mathematics. Springer, Berlin (2002)
12. Eberhard, S., Ebner, G., Hetzl, S.: Algorithmic compression of finite tree languages by rigid acyclic grammars. ACM Trans. Comput. Log. **18**(4), 26:1–26:20 (2017)
13. Eberhard, S., Hetzl, S.: Inductive theorem proving based on tree grammars. Ann. Pure Appl. Log. **166**(6), 665–700 (2015)
14. Ebner, G., Hetzl, S., Reis, G., Riener, M., Wolfsteiner, S., Zivota, S.: System description: GAPT 2.0. In: 8th International Joint Conference on Automated Reasoning, IJCAR (2016)
15. Finger, M., Gabbay, D.: Equal rights for the cut: computable non-analytic cuts in cut-based proofs. Log. J. IGPL **15**(5–6), 553–575 (2007)
16. Gentzen, G.: Untersuchungen über das logische Schließen. Mathematische Zeitschrift **39**, 176–210,405–431 (1934–1935)
17. Hetzl, S., Leitsch, A., Reis, G., Tapolczai, J., Weller, D.: Introducing quantified cuts in logic with equality. In: Demri, S., Kapur, D., Weidenbach, C., (eds.) Automated Reasoning - 7th International Joint Conference, IJCAR. Lecture Notes in Computer Science, vol. 8562, pp. 240–254. Springer (2014)
18. Hetzl, S., Leitsch, A., Reis, G., Weller, D.: Algorithmic introduction of quantified cuts. Theor. Comput. Sci. **549**, 1–16 (2014)

19. Hetzl, S., Leitsch, A., Weller, D.: Towards algorithmic cut-introduction. In: Logic for Programming, Artificial Intelligence and Reasoning (LPAR-18). Lecture Notes in Computer Science, vol. **7180**, pp. 228–242. Springer (2012)
20. Ireland, A., Bundy, A.: Productive use of failure in inductive proof. J. Autom. Reason. **16**(1–2), 79–111 (1996)
21. Johansson, M., Dixon, L., Bundy, A.: Conjecture synthesis for inductive theories. J. Autom. Reason. **47**(3), 251–289 (2011)
22. Leitsch, A., Lettmann, M.P.: The problem of Π_2-cut-introduction. Theor. Comput. Sci. **706**, 83–116 (2018)
23. Miller, D., Nigam, V.: Incorporating tables into proofs. In: 16th Conference on Computer Science and Logic (CSL07). Lecture Notes in Computer Science, vol. **4646**, pp. 466–480. Springer (2007)
24. Orevkov, V.: Lower bounds for increasing complexity of derivations after cut elimination. Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta **88**, 137–161 (1979)
25. Plotkin, G.D.: A note on inductive generalization. Mach. Intell. **5**(1), 153–163 (1970)
26. Plotkin, G.D.: A further note on inductive generalization. Mach. Intell. **6**, 101–124 (1971)
27. Pudlák, P.: The Lengths of Proofs. In: Buss, S. (ed.) Handbook of Proof Theory, pp. 547–637. Elsevier, Amsterdam (1998)
28. Reynolds, J.C.: Transformational systems and the algebraic structure of atomic formulas. Mach. Intell. **5**(1), 135–151 (1970)
29. Shoenfield, J.R.: Mathematical Logic, 2nd edn. Addison Wesley, Boston (1973)
30. Sorge, V., Colton, S., McCasland, R., Meier, A.: Classification results in quasigroup and loop theory via a combination of automated reasoning tools. Comment. Math. Univ. Carol. **49**(2), 319–339 (2008)
31. Sorge, V., Meier, A., McCasland, R., Colton, S.: Automatic construction and verification of isotopy invariants. J. Autom. Reason. **40**(2–3), 221–243 (2008)
32. Statman, R.: Lower bounds on Herbrand's theorem. Proc. Am. Math. Soc. **75**, 104–107 (1979)
33. Sutcliffe, G.: The TPTP problem library and associated infrastructure: the FOF and CNF parts, v3.5.0. J. Autom. Reason. **43**(4), 337–362 (2009)
34. Vyskočil, J., Stanovský, D., Urban, J.: Automated proof compression by invention of new definitions. In: Clark, E.M., Voronkov, A., (eds.) Logic for Programming, Artifical Intelligence and Reasoning (LPAR-16). Lecture Notes in Computer Science, vol. 6355, pp. 447–462. Springer (2010)
35. Woltzenlogel Paleo, B.: Atomic cut introduction by resolution: proof structuring and compression. In: Clark, E.M., Voronkov, A., (eds.) Logic for Programming, Artifical Intelligence and Reasoning (LPAR-16). Lecture Notes in Computer Science, vol. 6355, pp. 463–480. Springer (2010)