Towards the Compression of First-Order Resolution Proofs by Lowering Unit Clauses

Jan Gorzny[1] * and Bruno Woltzenlogel Paleo[2] **

[1] `jgorzny@uvic.ca`, University of Victoria, Canada
[2] `bruno@logic.at`, Vienna University of Technology, Austria

Abstract. The recently developed `LowerUnits` algorithm compresses propositional resolution proofs generated by SAT- and SMT-solvers by postponing and lowering resolution inferences involving unit clauses, which have exactly one literal. This paper describes a generalization of this algorithm to the case of first-order resolution proofs generated by automated theorem provers. An empirical evaluation of a simplified version of this algorithm on hundreds of proofs shows promising results.

1 Introduction

Most of the effort in automated reasoning so far has been dedicated to the design and implementation of proof systems and efficient theorem proving procedures. As a result, saturation-based first-order automated theorem provers have achieved a high degree of maturity, with resolution and superposition being among the most common underlying proof calculi. Proof production is an essential feature of modern state-of-the-art provers and proofs are crucial for applications where the user requires certification of the answer provided by the prover. Nevertheless, efficient proof production is non-trivial, and it is to be expected that the best, most efficient, provers do not necessarily generate the best, least redundant, proofs. Therefore, it is a timely moment to develop methods that post-process and simplify proofs. While the foundational problem of simplicity of proofs can be traced back at least to Hilbert's 24th Problem, the maturity of automated deduction has made it particularly relevant today.

For proofs generated by SAT- and SMT-solvers, which use propositional resolution as the basis for the DPLL and CDCL decision procedures, there is now a wide variety of proof compression techniques. Algebraic properties of the resolution operation that might be useful for compression were investigated in [5]. The `Reduce&Reconstruct` algorithm [10] searches for locally redundant subproofs that can be rewritten into subproofs of stronger clauses and with fewer resolution steps. A linear time proof compression algorithm based on partial regularization was proposed in [2] and improved in [6]. Furthermore, [6] described a linear time algorithm called `LowerUnits` that delays resolution with unit clauses.

In contrast, for first-order theorem provers, there has been up to now (to the best of our knowledge) no attempt to design and implement an algorithm capable of taking a first-order resolution DAG-proof and efficiently simplifying it, outputting a possibly shorter pure first-order resolution DAG-proof. There are algorithms aimed at simplifying first-order sequent calculus tree-like proofs, based on cut-introduction and Herbrand sequents [9, 7, 8]. There is also an algorithm [12] that looks for terms that occur often in any TSTP [11] proof (including first-order resolution DAG-proofs) and introduces abbreviations for these terms. However, as the definitions of the abbreviations are not part of the output proof, it cannot be checked by a pure first-order resolution proof checker.

In this paper, we initiate the process of lifting propositional proof compression techniques to the first-order case, starting with the simplest known algorithm: `LowerUnits` (described in [6]). As shown in Section 3, even for this simple algorithm, the fact that first-order resolution makes use of unification leads to many challenges that simply do not exist in the propositional case. In Section 4 we describe an easy to implement algorithm with linear time complexity (with respect to the proof length) which partially overcomes these challenges. In Section 5 we present experimental results obtained by applying this algorithm on hundreds of proofs generated with the SPASS theorem prover. The next section introduces the first-order resolution calculus using notations that are more convenient for describing proof transformation operations.

2 The Resolution Calculus

We assume that there are infinitely many variable symbols (e.g. X, Y, Z, X_1, X_2, ...), constant symbols (e.g. a, b, c, a_1, a_2, ...), function symbols of every arity (e.g f, g, f_1, f_2, ...) and predicate symbols of every arity (e.g. p, q, p_1, p_2,...). A *term* is any variable, constant or the application of an n-ary function symbol to n terms. An *atomic formula* (*atom*) is the application of an n-ary predicate symbol to n terms. A *literal* is an atom or the negation of an atom. The *complement* of a literal ℓ is denoted $\bar{\ell}$ (i.e. for any atom p, $\bar{p} = \neg p$ and $\overline{\neg p} = p$). The set of all literals is denoted $\mathcal{L}$. A *clause* is a multiset of literals. $\bot$ denotes the *empty clause*. A *unit clause* is a clause with a single literal. Sequent notation is used for clauses (i.e. $p_1, \ldots, p_n \vdash q_1, \ldots, q_m$ denotes the clause $\{\neg p_1, \ldots, \neg p_n, q_1, \ldots, q_m\}$). $\mathrm{FV}(t)$ (resp. $\mathrm{FV}(\ell)$, $\mathrm{FV}(\Gamma)$) denotes the set of variables in the term t (resp. in the literal ℓ and in the clause Γ). A *substitution* $\{X_1 \backslash t_1, X_2 \backslash t_2, \ldots\}$ is a mapping from variables $\{X_1, X_2, \ldots\}$ to, respectively, terms $\{t_1, t_2, \ldots\}$. The application of a substitution σ to a term t, a literal ℓ or a clause Γ results in, respectively, the term $t\sigma$, the literal $\ell\sigma$ or the clause $\Gamma\sigma$, obtained from t, ℓ and Γ by replacing all occurrences of the variables in σ by the corresponding terms in σ. The set of all substitutions is denoted $\mathcal{S}$. A *unifier* of a set of literals is a substitution that makes all literals in the set equal. A *resolution proof* is a directed acyclic graph of clauses where the edges correspond to the inference rules of resolution and contraction (as explained in detail in Definition 1). A *resolution refutation* is a resolution proof with root $\bot$.

Definition 1 (First-Order Resolution Proof).
A directed acyclic graph $\langle V, E, \Gamma \rangle$, where V is a set of nodes and E is a set of edges labeled by literals and substitutions (i.e. $E \subset V \times 2^{\mathcal{L}} \times \mathcal{S} \times V$ and $v_1 \xrightarrow[\sigma]{\ell} v_2$ denotes an edge from node v_1 to node v_2 labeled by the literal ℓ and the substitution σ), is a proof of a clause Γ iff it is inductively constructible according to the following cases:

- ***Axiom:** If Γ is a clause, $\widehat{\Gamma}$ denotes some proof $\langle \{v\}, \varnothing, \Gamma \rangle$, where v is a new (axiom) node.*
- ***Resolution:** If ψ_L is a proof $\langle V_L, E_L, \Gamma_L \rangle$ with $\ell_L \in \Gamma_L$ and ψ_R is a proof $\langle V_R, E_R, \Gamma_R \rangle$ with $\ell_R \in \Gamma_R$, and σ_L and σ_R are substitutions such that $\ell_L \sigma_L = \overline{\ell_R} \sigma_R$ and $\mathrm{FV}((\Gamma_L \setminus \{\ell_L\}) \sigma_L) \cap \mathrm{FV}((\Gamma_R \setminus \{\ell_R\}) \sigma_R) = \emptyset$, then $\psi_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \psi_R$ denotes a proof $\langle V, E, \Gamma \rangle$ s.t.*

$$V = V_L \cup V_R \cup \{v\} \qquad \Gamma = (\Gamma_L \setminus \{\ell_L\}) \sigma_L \cup (\Gamma_R \setminus \{\ell_R\}) \sigma_R$$

$$E = E_L \cup E_R \cup \left\{ \rho(\psi_L) \xrightarrow[\sigma_L]{\{\ell_L\}} v, \rho(\psi_R) \xrightarrow[\sigma_R]{\{\ell_R\}} v \right\}$$

  where v is a new (resolution) node and $\rho(\varphi)$ denotes the root node of φ. The resolved atom ℓ is such that $\ell = \ell_L \sigma_L = \overline{\ell_R} \sigma_R$ or $\ell = \overline{\ell_L} \sigma_L = \ell_R \sigma_R$.
- ***Contraction:** If ψ' is a proof $\langle V', E', \Gamma' \rangle$ and σ is a unifier of $\{\ell_1, \ldots \ell_n\}$ with $\{\ell_1, \ldots \ell_n\} \subseteq \Gamma'$, then $\lfloor \psi \rfloor_{\{\ell_1, \ldots \ell_n\}}^{\sigma}$ denotes a proof $\langle V, E, \Gamma \rangle$ s.t.*

$$V = V' \cup \{v\} \qquad E = E' \cup \{\rho(\psi') \xrightarrow[\sigma]{\{\ell_1, \ldots \ell_n\}} v\} \qquad \Gamma = (\Gamma' \setminus \{\ell_1, \ldots \ell_n\})\sigma \cup \{\ell\}$$

  where v is a new (contraction) node, $\ell = \ell_k \sigma$ (for any $k \in \{1, \ldots, n\}$) and $\rho(\varphi)$ denotes the root node of φ. □

The resolution and contraction (factoring) rules described above are the standard rules of the resolution calculus, except for the fact that we do not require resolution to use most general unifiers. The presentation of the resolution rule here uses two substitutions, in order to explicitly handle the necessary renaming of variables, which is often left implicit in other presentations of resolution.

When we write $\psi_L \odot_{\ell_L \ell_R} \psi_R$, we assume that the omitted substitutions are such that the resolved atom is most general. We write $\lfloor \psi \rfloor$ for an arbitrary maximal contraction, and $\lfloor \psi \rfloor^{\sigma}$ for a (pseudo-)contraction that does merge no literals but merely applies the substitution σ. When the literals and substitutions are irrelevant or clear from the context, we may write simply $\psi_L \odot \psi_R$ instead of $\psi_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \psi_R$. The $\odot$ operator is assumed to be left-associative. In the propositional case, we omit contractions (treating clauses as sets instead of multisets) and $\psi_L \odot_{\ell \overline{\ell}}^{\emptyset \emptyset} \psi_R$ is abbreviated by $\psi_L \odot_{\ell} \psi_R$.

If $\psi = \varphi_L \odot \varphi_R$ or $\psi = \lfloor \varphi \rfloor$, then φ, φ_L and φ_R are *direct subproofs* of ψ and ψ is a *child* of both φ_L and φ_R. The transitive closure of the direct subproof relation is the *subproof* relation. A subproof which has no direct subproof is an *axiom* of the proof. V_ψ, E_ψ and Γ_ψ denote, respectively, the nodes, edges and proved clause (conclusion) of ψ. If ψ is a proof ending with a resolution node, then ψ_L and ψ_R denote, respectively, the left and right premises of ψ.

3 First-Order Challenges

In this section, we describe challenges that have to be overcome in order to successfully adapt `LowerUnits` to the first-order case. The first example illustrates the need to take unification into account. The other two examples discuss complex issues that can arise when unification is taken into account in a naively.

Example 1. Consider the following proof ψ, noting that the unit subproof η_2 is used twice. It is resolved once with η_1 (against the literal $p(W)$ and producing the child η_3) and once with η_5 (against the literal $p(X)$ and producing ψ).

$$\frac{\dfrac{\eta_1\colon p(W) \vdash q(Z) \qquad \eta_2\colon\; \vdash p(Y)}{\eta_3\colon\; \vdash q(Z)} \qquad \eta_4\colon p(X), q(Z) \vdash}{\dfrac{\eta_5\colon p(X) \vdash \qquad\qquad\qquad \eta_2}{\psi\colon \bot}}$$

The result of deleting η_2 from ψ is the proof $\psi \setminus \{\eta_2\}$ shown below:

$$\frac{\eta_1'\colon p(W) \vdash q(Z) \qquad \eta_4'\colon p(X), q(Z) \vdash}{\eta_5'\ (\psi')\colon p(W), p(X) \vdash}$$

Unlike in the propositional case, where the literals that had been resolved against the unit are all syntactically equal, in the first-order case, this is not necessarily the case. As illustrated above, $p(W)$ and $p(X)$ are not syntactically equal. Nevertheless, they are unifiable. Therefore, in order to reintroduce η_2', we may first perform a contraction, as shown below:

$$\frac{\dfrac{\dfrac{\eta_1'\colon p(W) \vdash q(Z) \qquad \eta_4'\colon p(X), q(Z) \vdash}{\eta_5'\colon p(X), p(Y) \vdash}}{\lfloor \eta_5' \rfloor\colon p(U) \vdash} \qquad \eta_2'\colon\; \vdash p(Y)}{\psi^\star\colon \bot}$$

Example 2. There are cases, as shown below, when the literals that had been resolved away are not unifiable, and then a contraction is not possible.

$$\frac{\eta_4\colon r(X), p(b) \vdash s(Y) \quad \dfrac{\eta_1\colon p(a) \vdash q(Y), r(Z) \qquad \eta_2\colon\; \vdash p(X)}{\eta_3\colon\; \vdash q(Y), r(Z)}}{\dfrac{\dfrac{\eta_5\colon p(b) \vdash s(Y), q(Y) \qquad\qquad \eta_6\colon s(Y) \vdash}{\eta_7\colon p(b) \vdash q(Y) \qquad\qquad \eta_8\colon q(Y) \vdash}}{\dfrac{\eta_2 \qquad\qquad\qquad \eta_9\colon p(b) \vdash}{\psi\colon \bot}}}$$

If we attempted to postpone the resolution inferences involving the unit η_2 (i.e. by deleting η_2 and reintroducing it with a single resolution inference in the bottom of the proof), a contraction of the literals $p(a)$ and $p(b)$ would be needed. Since these literals are not unifiable, the contraction is not possible. Note that, in principle, we could still lower η_2 if we resolved it not only once but twice when reintroducing it in the bottom of the proof. However, this would lead to no compression of the proof's length.

The observations above lead to the idea of requiring units to satisfy the following property before collecting them to be lowered.

Definition 2. *Let η be a unit with literal ℓ and let $\eta_1, \ldots, \eta_n$ be subproofs that are resolved with η in a proof ψ, respectively, with resolved literals $\ell_1, \ldots, \ell_n$. η is said to satisfy the* pre-deletion unifiability property *in ψ if $\ell_1, \ldots, \ell_n$, and $\overline{\ell}$ are unifiable.*

Example 3. Satisfaction of the pre-deletion unifiability property is not enough. Deletion of the units from a proof ψ may actually change the literals that had been resolved away by the units, because fewer substitutions are applied to them. This is exemplified below:

$$\cfrac{\cfrac{\eta_1\colon r(Y), p(X, q(Y,b)), p(X,Y) \vdash \qquad \eta_2\colon\ \vdash p(U,V)}{\eta_3\colon r(V), p(U, q(V,b)) \vdash} \qquad \eta_4\colon\ \vdash r(W)}{\cfrac{\eta_5\colon p(U, q(W,b)) \vdash \qquad\qquad\qquad\qquad\qquad\qquad \eta_2}{\psi\colon \bot}}$$

If η_2 is collected for lowering and deleted from ψ, we obtain the proof $\psi \setminus \{\eta_2\}$:

$$\cfrac{\eta_1'\colon r(Y), p(X, q(Y,b)), p(X,Y) \vdash \qquad \eta_4'\colon\ \vdash r(W)}{\eta_5'(\psi')\colon p(X, q(W,b)), p(X,W) \vdash}$$

Note that, even though η_2 satisfies the pre-deletion unifiability property (since $p(X, q(Y,b))$ and $p(U, q(W,b))$ are unifiable), η_2 still cannot be lowered and reintroduced by a single resolution inference, because the corresponding modified post-deletion literals $p(X, q(W,b))$ and $p(X, W)$ are actually not unifiable.

The observation above leads to the following stronger property:

Definition 3. *Let η be a unit with literal ℓ_η and let $\eta_1, \ldots, \eta_n$ be subproofs that are resolved with η in a proof ψ, respectively, with resolved literals $\ell_1, \ldots, \ell_m$. η is said to satisfy the* post-deletion unifiability property *in ψ if $\ell_1^{\dagger\downarrow}, \ldots, \ell_m^{\dagger\downarrow}$, and $\overline{\ell_\eta^{\dagger}}$ are unifiable, where $\ell^{\dagger}$ is the literal in $\psi \setminus \{\eta\}$ corresponding to ℓ in ψ and $\ell_k^{\dagger\downarrow}$ is the descendant of $\ell_k^{\dagger}$ in the root of $\psi \setminus \{\eta\}$.*

4   A Linear Greedy Variant of First-Order LowerUnits

The examples shown in the previous section indicate that there are two main challenges that need to be overcome in order to generalize `LowerUnits` to the first-order case:

1. The deletion of a node changes literals. Since substitutions associated with the deleted node are not applied anymore, some literals become more general. Therefore, the reconstruction of the proof during deletion needs to take such changes into account.

2. Whether a unit should be collected for lowering must depend on whether the literals that were resolved with the unit's single literal are unifiable after they are propagated down to the bottom of the proof by the process of unit deletion. Only if this is the case, they can be contracted and the unit can be reintroduced in the bottom of the proof.

The first challenge can be overcome by keeping an additional map from old literals in the input proof to the corresponding more general changed literals in the output proof under construction. The second challenge is harder to overcome. In the propositional case, collecting units and deleting units can be done in two distinct and independent phases (as in `LowerUnits`). In the first-order case, on the other hand, these two phases seem to be so interlaced, that they appear to be in a deadlock: the decision to collect a unit to be lowered depends on what will happen with the proof after deletion, while deletion depends on knowing which units will be lowered. In a naive approach, the deletion algorithm may have to be executed once for every collected unit, and since the number of collected units is in the worst case linear in the length of the proof, the overall runtime complexity is quadratic with respect to the length of the proof.

This section presents `GreedyLinearFirstOrderLowerUnits` (Algorithm 1), a single traversal first-order adaptation of `LowerUnits`, which avoids the quadratic complexity and the implementation difficulties by: 1) ignoring the stricter post-deletion unifiability property and focusing instead on the pre-deletion unifiability property, which is easier to check (lines 9-11); and 2) employing a greedy contraction approach (lines 16-18) together with substitutions (lines 5-7), in order not to care about bookkeeping. By doing so, compression may not always succeed on all proofs (e.g. Example 3). When compression succeeds, the root clause of the generated proof will be the empty clause (line 20) and the generated proof may be returned. Otherwise, the original proof must be returned (line 20).

5   Experiments

A prototype of a (two-traversal) version of `GreedyLinearFirstOrderLowerUnits` has been implemented in the functional programming language Scala as part of the Skeptik library (`https://github.com/Paradoxika/Skeptik`) [3]. Before evaluating this algorithm, we first generated several benchmark proofs. This was done by executing the SPASS (`http://www.spass-prover.org/`) theorem prover on 2280 real first-order problems without equality of the TPTP Problem Library (among them, 1032 problems are known to be unsatisfiable). In order to generate pure resolution proofs, the advanced inference rules of SPASS were disabled. The Euler Cluster at the University of Victoria was used and the time limit was 300 seconds per problem. Under these conditions, SPASS generated 308 proofs.

The evaluation of `GreedyLinearFirstOrderLowerUnits` was performed on a laptop (2.8GHz Intel Core i7 processor with 4 GB of RAM (1333MHz DDR3) available to the Java Virtual Machine). For each benchmark proof ψ, we measured the time needed to compress the proof $(t(\psi))$ and the compression ratio

---

Input: a proof ψ
Output: a compressed proof $\psi^\star$
Data: a map $.'$, eventually mapping any φ to delete(φ, Units)

1 $D \leftarrow \varnothing$; // set for storing subproofs that need to be deleted
2 Units $\leftarrow \varnothing$; // stack for storing collected units

3 for *every subproof φ, in a top-down traversal of ψ* **do**
4   **if** *φ is an axiom* **then** $\varphi' \leftarrow \varphi$;
5   **else if** $\varphi = \varphi_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \varphi_R$ **then**
6   **if** $\varphi_L \in D$ *and* $\varphi_R \in D$ **then add** φ to D ;
7   **else if** $\varphi_L \in D$ **then** $\varphi' \leftarrow \lfloor \varphi'_R \rfloor^{\sigma_R}$;
8   **else if** $\varphi_R \in D$ **then** $\varphi' \leftarrow \lfloor \varphi'_L \rfloor^{\sigma_L}$;
9   **else if** $\ell \notin \Gamma_{\varphi'_L}$ **then** $\varphi' \leftarrow \lfloor \varphi'_L \rfloor^{\sigma_L}$;
10   **else if** $\bar{\ell} \notin \Gamma_{\varphi'_R}$ **then** $\varphi' \leftarrow \lfloor \varphi'_R \rfloor^{\sigma_R}$;
11   **else** $\varphi' \leftarrow \varphi'_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \varphi'_R$;
12   **else if** $\varphi = \lfloor \varphi_c \rfloor_{\{\ell_1, \ldots, \ell_n\}}^{\sigma}$ **then** $\varphi' \leftarrow \lfloor \varphi'_c \rfloor_{\{\ell_1, \ldots, \ell_n\}}^{\sigma}$;

13   **if** *φ is a unit with more than one child satisfying the pre-deletion unifiability property* **then**
14   **push** φ' onto Units;
15   **add** φ to D ;

   // Reintroduce units
16 $\psi^\star \leftarrow \psi'$;
17 while Units $\neq \varnothing$ **do**
18 $\varphi' \leftarrow$ **pop** from Units;
19 $\psi_{\text{next}}^\star \leftarrow \lfloor \psi^\star \rfloor$;
20   **while** $\Gamma_{\psi_{\text{next}}^\star} \neq \psi^\star$ **do**
21 $\psi^\star \leftarrow \psi_{\text{next}}^\star$;
22 $\psi_{\text{next}}^\star \leftarrow \lfloor \psi^\star \rfloor$;
23   **if** *$\psi^\star \odot \varphi'$ is well-defined* **then** $\psi^\star \leftarrow \psi^\star \odot \varphi'$;
24 if $\Gamma_{\psi^\star} = \bot$ **then return** $\psi^\star$;
25 else return ψ;

Algorithm 1: GreedyLinearFirstOrderLowerUnits (single traversal)

$((|\psi| - |\alpha(\psi)|)/|\psi|)$, where $|\psi|$ is the length of ψ (i.e. the number of axioms, resolution and contractions (ignoring substitutions)) and $\alpha(\psi)$ is the result of applying GreedyLinearFirstOrderLowerUnits to ψ. The raw data is available at: http://www.math.uvic.ca/~jgorzny/data/.

The proofs generated by SPASS were small (with lengths from 3 to 49). These proofs are specially small in comparison with the typical proofs generated by SAT- and SMT-solvers, which usually have from a few hundred to a few million nodes. The number of proofs (compressed and uncompressed) per length is shown in Figure 1 (b). Uncompressed proofs are those which had either no lowerable units to lower or for which GreedyLinearFirstOrderLowerUnits failed and returned the original proof. Such failures occurred on only 14 benchmark proofs.

Among the smallest of the 308 proofs, very few proofs were compressed. This is to be expected, since the likelihood that a very short proof contain a lowerable unit (or even merely a unit with more than one child) is low. The proportion of compressed proofs among longer proofs is, as expected, larger, since they have more nodes and it is more likely that some of these nodes are lowerable units. 13 out of 18 proofs with length greater than or equal to 30 were compressed.

Figure 1 (a) shows a box-whisker plot of compression ratio with proofs grouped by length and whiskers indicating minimum and maximum compression ratio achieved within the group. Besides the median compression ratio (the horizontal thick black line), the chart also shows the mean compression ratios for all proofs of that length and for all compressed proofs (the red cross and the blue circle). In the longer proofs (length greater than 34), the median and the means are in the range from 5% to 15%, which is satisfactory in comparison with the total compression ratio of 7.5% that has been measured for the propositional `LowerUnits` algorithm on much longer propositional proofs [4].

Figure 1 (c) shows a scatter plot comparing the length of the input proof against the length of the compressed proof. For the longer proofs (circles in the right half of the plot), it is often the case that the length of the compressed proof is significantly lesser than the length of the input proof.

Figure 1 (d) plots the cumulative original and compressed lengths of all benchmark proofs (for an x-axis value of k, the cumulative curves show the sum of the lengths of the shortest k input proofs). The total cumulative length of all original proofs is 4429 while the cumulative length of all proofs after compression is 3929. This results in a total compression ratio of 11.3%, which is impressive, considering that the inclusion of all the short proofs (in which the presence of lowerable units is a priori unlikely) tends to decrease the total compression ratio. For comparison, the total compression ratio considering only the 100 longest input proofs is 18.4%.

Figure 1 also indicates an interesting potential trend. The gap between the two cumulative curves seems to grow superlinearly. If this trend is extrapolated, progressively larger compression ratios can be expected for longer proofs. This is compatible with Theorem 10 in [6], which shows that, for proofs generated by eagerly resolving units against all clauses, the propositional `LowerUnits` algorithm can achieve quadratic assymptotic compression. SAT- and SMT-solvers based on CDCL (Conflict-Driven Clause Learning) avoid eagerly resolving unit clauses by dealing with unit clauses via boolean propagation on a conflict graph and extracting subproofs from the conflict graph with every unit being used at most once per subproof (even when it was used multiple times in the conflict graph). Saturation-based automated theorem provers, on the other hand, might be susceptible to the eager unit resolution redundancy described in Theorem 10 [6]. This potential trend would need to be confirmed by further experiments with more data (more proofs and longer proofs).

The total time needed by SPASS to solve the 308 problems for which proofs were generated was 2403 seconds, or approximately 40 minutes (running on the Euler Cluster and including parsing time and proof generation time for

(a) Compression ratio

(b) Number of (non-)compressed proofs

(c) Compressed length against input length
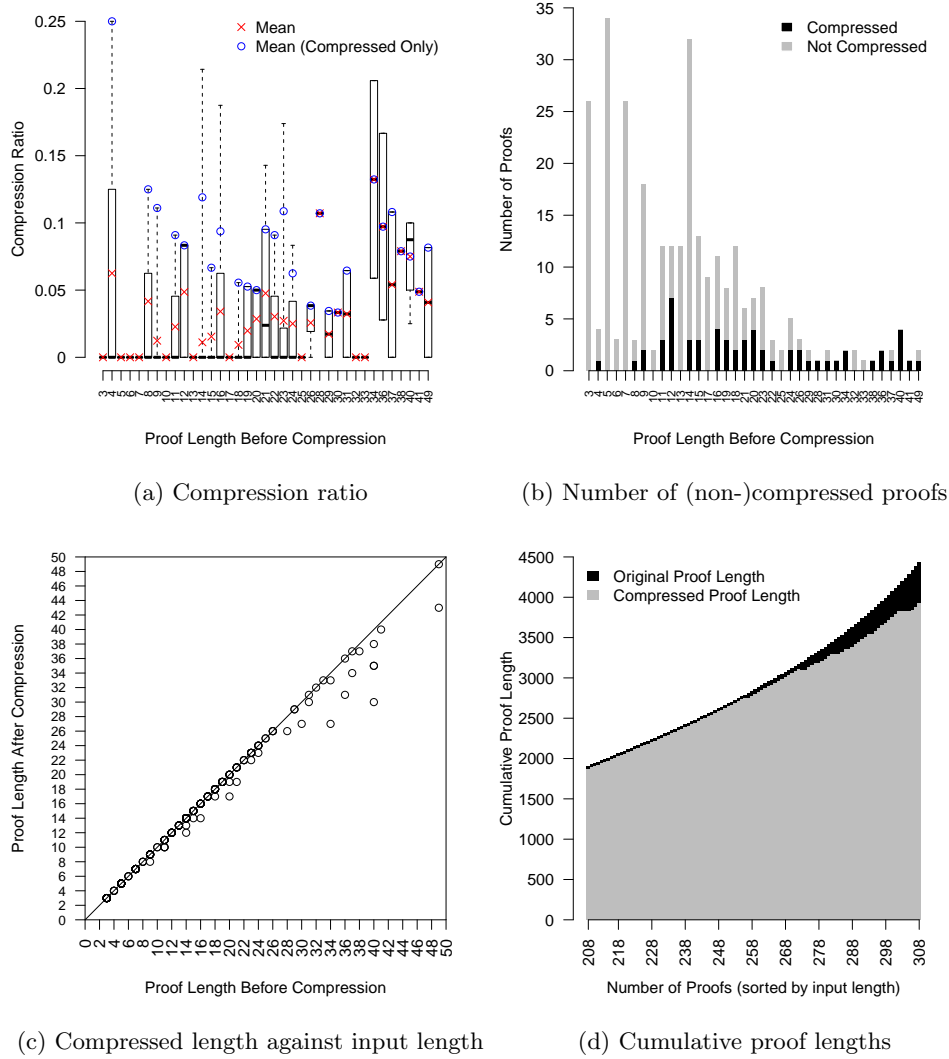

(d) Cumulative proof lengths

Fig. 1: Experimental results

each problem). The total time for `GreedyLinearFirstOrderLowerUnits` to be executed on all 308 proofs was just under 5 seconds on a simple laptop (including parsing each proof). Therefore, `GreedyLinearFirstOrderLowerUnits` is a fast algorithm. For a very small overhead in time (in comparison to proving time), it may simplify the proof considerably.

6 Conclusions and Future Work

`GreedyLinearFirstOrderLowerUnits` is our first attempt to lift a propositional proof compression algorithm to the first-order case. We consider this algorithm

a prototype, useful to evaluate whether this approach is promising. The experimental results discussed in the previous section are encouraging, especially in comparison with existing results for the propositional case. In the near future, we shall seek improvements of this algorithm as well as other ways to overcome the difficulties related to the post-deletion unifiability property.

The difficulties related to unit reintroduction suggest that other propositional proof compression algorithms that do not require reintroduction (e.g. `RecyclePivotsWithIntersection` [6]) might need less sophisticated bookkeeping when lifted to first-order.

The efficiency and versatility of contemporary automated theorem provers depend on inference rules (e.g. equality rules) and techniques (e.g. splitting) that go beyond the pure resolution calculus. The eventual generalization of the compression algorithms to support such extended calculi will be essential for their usability on a wider range of problems.

References

1. *Logic for Programming, Artificial Intelligence, and Reasoning - 16th Intl. Conf., LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, LNCS. Springer, 2010.
2. O. Bar-Ilan, O. Fuhrmann, S. Hoory, O. Shacham, and O. Strichman. Linear-time reductions of resolution proofs. In *Haifa Verification Conf.*, LNCS, pages 114–128. Springer, 2008.
3. J. Boudou, A. Fellner, and B. Woltzenlogel Paleo. Skeptik: A proof compression system. In *Automated Reasoning - 7th Intl. Joint Conf., IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings*, LNCS, pages 374–380. Springer, 2014.
4. J. Boudou and B. Woltzenlogel Paleo. Compression of propositional resolution proofs by lowering subproofs. In *Automated Reasoning with Analytic Tableaux and Related Methods - 22th Intl. Conf., TABLEAUX*, LNCS, pages 59–73. Springer, 2013.
5. P. Fontaine, S. Merz, and B. Woltzenlogel Paleo. Exploring and exploiting algebraic and graphical properties of resolution. In *8th Intl. Workshop on Satisfiability Modulo Theories*, Edinburgh, Royaume-Uni, July 2010.
6. P. Fontaine, S. Merz, and B. Woltzenlogel Paleo. Compression of propositional resolution proofs via partial regularization. In *CADE*, LNCS, pages 237–251. Springer, 2011.
7. S. Hetzl, A. Leitsch, G. Reis, and D. Weller. Algorithmic introduction of quantified cuts. *Theor. Comput. Sci.*, 549:1–16, 2014.
8. S. Hetzl, A. Leitsch, D. Weller, and B. Woltzenlogel Paleo. Herbrand sequent extraction. In *9th Intl. Conf. on Intelligent Computer Mathematics*, LNCS, pages 462–477. Springer, 2008.
9. B. Woltzenlogel Paleo. Atomic cut introduction by resolution: Proof structuring and compression. In *LPAR-16* [1], pages 463–480.
10. S. F. Rollini, R. Bruttomesso, and N. Sharygina. An efficient and flexible approach to resolution proof reduction. In *Hardware and Software: Verification and Testing*, LNCS, pages 182–196. Springer, 2011.
11. G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

12. J. Vyskocil, D. Stanovský, and J. Urban. Automated proof compression by invention of new definitions. In *LPAR-16* [1], pages 447–462.