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 [7]. Compression algorithms based on rearranging and sharing chains of resolution inferences have been developed in [1] and [12]. Cotton [6] proposed an algorithm that compresses a refutation by repeteadly splitting it into a proof of a heuristically chosen literal ℓ and a proof of $\overline{\ell}$, and then resolving them to form a new refutation. The Reduce&Reconstruct algorithm [11] searches for locally redundant subproofs that can be rewritten into subproofs of stronger clauses and

* Supported by the Google Summer of Code 2014 program.
** Supported by the Austrian Science Fund, project P24300.

with fewer resolution steps. A linear time proof compression algorithm based on partial regularization was proposed in [2] and improved in [8]. Furthermore, [8] also described a new linear time algorithm called `LowerUnits`, which 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 [10, 9], and while in principle resolution DAG-proofs can be translated to sequent-calculus tree-like proofs (and then back), such translations lead to undesirable efficiency overheads. There is also an algorithm [14] that looks for terms that occur often in any TSTP [13] 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 Section 3). As shown in Section 4, 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 5 we describe an algorithm that overcomes these challenges. As this algorithm has quadratic run-time complexity with respect to the input proof length, in Section 6 we describe a variation of this algorithm, which has linear run-time complexity and is easier to implement, at the cost of compressing less. In Section 7 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, $\ldots$), constant symbols (e.g. a, b, c, a_1, a_2, $\ldots$), function symbols of every arity (e.g f, g, f_1, f_2, $\ldots$) and predicate symbols of every arity (e.g. p, q, p_1, p_2,$\ldots$). 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 $\overline{\ell}$ (i.e. for any atom p, $\overline{p} = \neg p$ and $\overline{\neg p} = p$). The set of all literals is denoted $\mathcal{L}$. A *clause* is a multiset of literals. $\perp$ 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. $\mathrm{FV}(\ell)$, $\mathrm{FV}(\mathit{\Gamma})$) denotes the set of variables in the term t (resp. in the literal ℓ and in the clause $\mathit{\Gamma}$). 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 $\varGamma$ results in, respectively, the term $t\sigma$, the literal $\ell\sigma$ or the clause $\varGamma\sigma$, obtained from t, ℓ and $\varGamma$ 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 $\langle V, E, \varGamma \rangle$, 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 $\varGamma$ iff it is inductively constructible according to the following cases:

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

$$
\begin{aligned}
V &= V_L \cup V_R \cup \{v\} \\
E &= E_L \cup E_R \cup \left\{ \rho(\psi_L) \xrightarrow[\sigma_L]{\{\ell_L\}} v, \rho(\psi_R) \xrightarrow[\sigma_R]{\{\ell_R\}} v \right\} \\
\varGamma &= (\varGamma_L \setminus \{\ell_L\})\, \sigma_L \cup (\varGamma_R \setminus \{\ell_R\})\, \sigma_R
\end{aligned}
$$

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 $\langle V', E', \varGamma' \rangle$ and σ is a unifier of $\{\ell_1, \dots \ell_n\}$ with $\{\ell_1, \dots \ell_n\} \subseteq \varGamma'$, then $\lfloor \psi \rceil^{\sigma}_{\{\ell_1, \dots \ell_n\}}$ denotes a proof $\langle V, E, \varGamma \rangle$ s.t.*

$$
\begin{aligned}
V &= V' \cup \{v\} \\
E &= E' \cup \{\rho(\psi') \xrightarrow[\sigma]{\{\ell_1, \dots \ell_n\}} v\} \\
\varGamma &= (\varGamma' \setminus \{\ell_1, \dots \ell_n\})\sigma \cup \{\ell\}
\end{aligned}
$$

where v is a new (contraction) node, $\ell = \ell_k \sigma$ (for any $k \in \{1, \dots, n\}$) 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 usually left implicit in many presentations of the resolution calculus.

When the literals and substitutions involved in a resolution or contraction inference are irrelevant or clear from the context, we may write simply $\psi_L \odot \psi_R$ instead of $\psi_L \odot^{\sigma_L \sigma_R}_{\ell_L \ell_R} \psi_R$ and $\lfloor \psi \rfloor$ instead of $\lfloor \psi \rfloor^{\sigma}_{\{\ell_1, \ldots \ell_n\}}$. 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. When parenthesis are omitted, $\odot$ is assumed to be left-associative. In the propositional case, we omit contractions (treating clauses essentially as sets instead of multisets) and $\psi_L \odot^{\emptyset\emptyset}_{\ell\overline{\ell}} \psi_R$ 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 The Propositional LowerUnits Algorithm

We denote by $\psi \setminus \{\varphi_1, \varphi_2\}$ the result of deleting the subproofs φ_1 and φ_2 from the proof ψ and fixing it according to Algorithm 1[1]. We say that a subproof φ in a proof ψ can be lowered if there exists a proof ψ' such that $\psi' = \psi \setminus \{\varphi\} \odot \varphi$ and $\Gamma_{\psi'} \subseteq \Gamma_\psi$. If φ originally participated in many resolution inferences within ψ (i.e. if φ had many children in ψ) then lowering φ compresses the proof (in number of resolution inferences), because $\psi \setminus \{\varphi\} \odot \varphi$ contains a single resolution inference involving φ.

It has been noted in [8] that, in the propositional case, φ can always be lowered if it is a *unit* (i.e. its conclusion clause is unit). This led to the invention of `LowerUnits` (Algorithm 2), which aims at transforming a proof ψ into $(\psi \setminus \{\mu_1, \ldots, \mu_n\}) \odot \mu_1 \odot \ldots \odot \mu_n$, where μ_1, ..., μ_n are all units with more than one child. Units with only one child are ignored because no compression is gained by lowering them. The order in which the units are reintroduced is important: if a unit φ_2 is a subproof of a unit φ_1 then φ_2 has to be reintroduced later than (i.e. below) φ_1.

In Algorithm 2, units are collected in a queue during a bottom-up traversal (lines 2-3), then they are deleted from the proof (line 4) and finally reintroduced in the bottom of the proof (lines 5-7). In [4] it has been observed that the two traversals (one for collection and one for deletion) could be merged into a single traversal, if we collect units during deletion. As deletion is a top-down traversal, it is then necesary to collect the units in a stack. This improvement leads to

[1] The deletion algorithm is a variant of the Reconstruct-Proof algorithm presented in [3]. The basic idea is to traverse the proof in a top-down manner, replacing each subproof having one of its premises marked for deletion (i.e. in D) by its other premise (cf. [4]).

```
  Input: a proof $\varphi$
  Input: $D$ a set of subproofs
  Output: a proof $\varphi'$ obtained by deleting the subproofs in $D$ from $\varphi$
  Data: a map $.'$, initially empty, eventually mapping any $\xi$ to delete($\xi$, $D$)
1 if $\varphi \in D$ or $\rho(\varphi)$ has no premises then return $\varphi$
2 else
3     let $\varphi_L \odot_\ell \varphi_R = \varphi$ ;
4     $\varphi'_L \leftarrow$ delete($\varphi_L$,$D$) ;
5     $\varphi'_R \leftarrow$ delete($\varphi_R$,$D$) ;
6     if $\varphi'_L \in D$ then return $\varphi'_R$ else if $\varphi'_R \in D$ then return $\varphi'_L$
7     else if $\ell \notin \Gamma_{\varphi'_L}$ then return $\varphi'_L$ else if $\overline{\ell} \notin \Gamma_{\varphi'_R}$ then return $\varphi'_R$
8     else return $\varphi'_L \odot_\ell \varphi'_R$
```

Algorithm 1: `delete`

```
  Input: a proof $\psi$
  Output: a compressed proof $\psi^\star$
  Data: a map $.'$: after line 4, it maps any $\varphi$ to delete($\varphi$, $D$)
1 Units $\leftarrow \varnothing$;   // queue to store collected units
2 for every subproof $\varphi$, in a bottom-up traversal of $\psi$ do
3     if $\varphi$ is a unit with more than one child then enqueue $\varphi$ in Units
4 $\psi' \leftarrow$ delete($\psi$,Units) ;
  // Reintroduce units
5 $\psi^\star \leftarrow \psi'$ ;
6 for every unit $\varphi$ in Units do
7     let $\{\ell\} = \Gamma_\varphi$ ;
8     if $\overline{\ell} \in \Gamma_{\psi'}$ then $\psi^\star \leftarrow \psi^\star \odot_\ell \varphi'$
```

Algorithm 2: `LowerUnits`

Algorithm 3. Both algorithms have a linear run-time complexity with respect to the length of the proof, because they perform a contant number of traversals.

4 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 take into account in a naive way.

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

```
Input: a proof $\psi$
Output: a compressed proof $\psi^\star$
Data: a map $.'$, eventually mapping any $\varphi$ to delete($\varphi$, Units)
$D \leftarrow \varnothing$;   // set for storing subproofs that need to be deleted
Units $\leftarrow \varnothing$;   // stack for storing collected units
for every subproof $\varphi$, in a top-down traversal of $\psi$ do
    if $\varphi$ is an axiom then $\varphi' \leftarrow \varphi$ else
        let $\varphi_L \odot_\ell \varphi_R = \varphi$ ;
        if $\varphi_L \in D$ and $\varphi_R \in D$ then add $\varphi$ to $D$ else if $\varphi_L \in D$ then
        $\varphi' \leftarrow \varphi'_R$ else if $\varphi_R \in D$ then $\varphi' \leftarrow \varphi'_L$
        else if $\ell \notin \Gamma_{\varphi'_L}$ then $\varphi' \leftarrow \varphi'_L$ else if $\overline{\ell} \notin \Gamma_{\varphi'_R}$ then $\varphi' \leftarrow \varphi'_R$
        else $\varphi' \leftarrow \varphi'_L \odot_\ell \varphi'_R$

    if $\varphi$ is a unit with more than one child then
        push $\varphi'$ onto Units;
        add $\varphi$ to $D$ ;

// Reintroduce units
$\psi^\star \leftarrow \psi'$ ;
while Units $\neq \varnothing$ do
    $\varphi' \leftarrow$ pop from Units;
    let $\{\overline{\ell}\} = \Gamma_\varphi$ ;
    if $\ell \in \Gamma_{\psi^\star}$ then $\psi^\star \leftarrow \psi^\star \odot_\ell \varphi'$
```

Algorithm 3: Improved `LowerUnits` (with a single traversal)

the child η_3) and once with η_5 (against the literal $p(X)$ and producing the root ψ).

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

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

$$\cfrac{\eta'_1\colon p(W) \vdash q(Z) \qquad \eta'_4\colon p(X), q(Z) \vdash}{\eta'_5\ (\psi')\colon p(W), p(X) \vdash}$$

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:

$$\cfrac{\cfrac{\cfrac{\eta'_1\colon p(W) \vdash q(Z) \qquad \eta'_4\colon p(X), q(Z) \vdash}{\eta'_5\colon p(X), p(Y) \vdash}}{\lfloor \eta'_5 \rfloor\colon p(U) \vdash} \qquad \eta'_2\colon \vdash p(Y)}{\psi^\star\colon \bot}$$

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.

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

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 η_1, ..., η_n be subproofs that are resolved with η in a proof ψ, respectively, with resolved literals ℓ_1, ..., ℓ_n. η is said to satisfy the* pre-deletion unifiability property *in ψ if ℓ_1,...,ℓ_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:

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

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

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

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 η_1, ..., η_n be subproofs that are resolved with η in a proof ψ, respectively, with resolved literals ℓ_1, ..., ℓ_m. η is said to satisfy the* post-deletion unifiability property *in ψ if $\ell_1^{\dagger\downarrow}$,...,$\ell_m^{\dagger\downarrow}$, 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 $\psi \setminus \{\eta\}$.*

5 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.

Algorithm 4 overcomes the first challenge by keeping an additional map from old literals in the input proof to the corresponding more general changed literals in the output proof unders construction. This is done in lines 6 to 7. The correspondence can be computed by proper bookkeeping during deletion (e.g. by having data structures that preserve the positions of literals or by annotating literals with ids). In cases where, due to previous deletions above in the proof, no corresponding literal is available anymore, the special constant `none` is used.

Not only the literals, but also the substitutions must change during deletion. While it would be in principle possible to keep track of such changes as well, it is simpler to search for new substitutions that result in a most general resolved atom. This is why substitutions are omitted in line 12. As a beneficial side-effect, we may obtain more general literals in the root clause of the output proof.

The second challenge is much harder to overcome. In the propositional case, collecting units and deleting units can be done in two distinct and independent phases (as shown in Algorithm 2). 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.

A simple way of unlocking this apparent deadlock is depicted in Algorithm 5. It optimistically assumes that all units with more than one child are lowerable (lines 2-3). Then it deletes the units (line 6) and tries to reintroduce them in the bottom (lines 8-19). If the reintroduction of a unit φ fails because the descendants of the literals that had been resolved with φ's literal are not unifiable, then φ is removed from the queue of collected units (lines 14-16) and the whole process is repeated, inside the *while* loop (lines 5-19), now without φ among the collected units. Since in the worst case the deletion algorithm may have to be executed once for every collected unit, and the number of collected units is in the worst case linear in the length of the proof, the overall runtime complexity is in the worst case quadratic with respect to the length of the proof. This is the price

```
Input: a proof $\varphi$
Input: $D$ a set of subproofs
Output: a proof $\varphi'$ obtained by deleting the subproofs in $D$ from $\varphi$
Data: a map $.'$, initially empty, eventually mapping any $\xi$ to delete($\xi$, $D$)
Data: a map $.^\dagger$, initially empty, eventually mapping literals to changed literals
if $\varphi \in D$ or $\rho(\varphi)$ has no premises then return $\varphi$
else if $\varphi = \varphi_L \odot^{\sigma_L \sigma_R}_{\ell_L \ell_R} \varphi_R$ then
    $\varphi'_L \leftarrow$ delete($\varphi_L$,$D$) ;
    $\varphi'_R \leftarrow$ delete($\varphi_R$,$D$) ;
    for every $\ell$ in $\Gamma_{\varphi_L}$ or $\Gamma_{\varphi_R}$ do
        $\ell^\dagger \leftarrow$ the literal in $\Gamma_{\varphi'_L}$ or $\Gamma_{\varphi'_R}$ corresponding to $\ell$, otherwise none ;
    if $\varphi'_L \in D$ then return $\varphi'_R$ else if $\varphi'_R \in D$ then return $\varphi'_L$
    else if $\ell^\dagger_L = none$ then return $\varphi'_L$ else if $\ell^\dagger_R = none$ then return $\varphi'_R$
    else return $\varphi'_L \odot_{\ell^\dagger_L \ell^\dagger_R} \varphi'_R$
else if $\varphi = \lfloor \varphi_c \rfloor^{\sigma}_{\{\ell_1,\ldots,\ell_n\}}$ then
    $\varphi'_c \leftarrow$ delete($\varphi_c$,$D$) ;
    for every $\ell$ in $\Gamma_{\varphi_c}$ do
        $\ell^\dagger \leftarrow$ the literal in $\Gamma_{\varphi'_c}$ corresponding to $\ell$, otherwise none ;
    return $\lfloor \varphi_c \rfloor_{\{\ell^\dagger_1,\ldots,\ell^\dagger_n\}\setminus\{none\}}$;
```

Algorithm 4: `fo-delete`

paid to disentangle the dependency between unit collection and deletion in a simple way.

Alternatively, we could try to lower units incrementally, one at a time, always eagerly deleting the unit and reconstructing the proof immediately after it is collected. The optimistic approach of Algorithm 5, however, has the potential to save some deletion cycles.

6 A Linear Greedy Variant of First-Order LowerUnits

The `FirstOrderLowerUnits` described in the previous section is not only complex (worst-case quadratic run-time complexity in the length of the input proof) but also difficult to implement. The necessity to ensure the post-deletion unifiability property would require a lot of bookkeeping, to track changes in literals and their descendants, and to know which literals have to be contracted in the bottom of the proof before reintroduction of the units.

This section presents `GreedyLinearFirstOrderLowerUnits` (Algorithm 6), an alterantive (single traversal) variant of `FirstOrderLowerUnits`, 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 (line 13); and 2) employing

```
   **Input**: a proof $\psi$
   **Output**: a compressed proof $\psi^\star$
   **Data**: a map $.'$: after line 4, it maps any $\varphi$ to delete($\varphi$, $D$)
   **Data**: a map $.^\dagger$,mapping literals to changed literals,updated after every deletion
 1 Units $\leftarrow \varnothing$;   // queue to store collected units
 2 for every subproof $\varphi$, in a bottom-up traversal of $\psi$ do
 3     if $\varphi$ is a unit with more than one child then enqueue $\varphi$ in Units
 4 $s \leftarrow$ false;   // indicator of successful reintroduction of all units
 5 while $\neg s$ do
 6     $\psi' \leftarrow$ delete($\psi$,Units) ;
       // Reintroduce units
 7     $s \leftarrow$ true ;
 8     $\psi^\star \leftarrow \psi'$ ;
 9     for every unit $\varphi$ in Units do
10         let $\{\ell\} = \Gamma_\varphi$ ;
11         let $\{\ell_1, \ldots, \ell_n\}$ be the literals resolved against $\ell$ in $\psi$ ;
12         let $c = \{\ell_1^\dagger, \ldots, \ell_n^\dagger\} \setminus \{none\}$ ;
13         let $c^\downarrow$ be the descendants of $c$'s literals in $\Gamma_{\psi'}$ ;
14         if $c^\downarrow$'s literals are not unifiable then
15             $s \leftarrow$ false ;
16             remove $\varphi$ from Units ;
               // interrupt the for-loop
17             break;
18         else if $c^\downarrow \neq \emptyset$ then
19             let $\sigma$ be the unifier of $c^\downarrow$'s literals and $\ell^c$ the unified literal ;
20             $\psi^\star \leftarrow \lfloor \psi^\star \rfloor^{\sigma}_{c^\downarrow} \odot_{\ell^c \ell^\dagger} \varphi'$ ;
```

Algorithm 5: FirstOrderLowerUnits

a greedy contraction approach (lines 19-22) together with substitutions (lines 7-10), 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 24) and the generated proof may be returned. Otherwise, the original proof must be returned (line 25).

```
Input: a proof $\psi$
Output: a compressed proof $\psi^\star$
Data: a map $.'$, eventually mapping any $\varphi$ to delete($\varphi$, Units)
$D \leftarrow \varnothing$;   // set for storing subproofs that need to be deleted
Units $\leftarrow \varnothing$;   // stack for storing collected units
for every subproof $\varphi$, in a top-down traversal of $\psi$ do
    if $\varphi$ is an axiom then $\varphi' \leftarrow \varphi$ else if $\varphi = \varphi_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \varphi_R$ then
        if $\varphi_L \in D$ and $\varphi_R \in D$ then add $\varphi$ to $D$ else if $\varphi_L \in D$ then
        $\varphi' \leftarrow \lfloor \varphi'_R \rfloor^{\sigma_R}$ else if $\varphi_R \in D$ then $\varphi' \leftarrow \lfloor \varphi'_L \rfloor^{\sigma_L}$
        else if $\ell \notin \Gamma_{\varphi'_L}$ then $\varphi' \leftarrow \lfloor \varphi'_L \rfloor^{\sigma_L}$ else if $\overline{\ell} \notin \Gamma_{\varphi'_R}$ then
        $\varphi' \leftarrow \lfloor \varphi'_R \rfloor^{\sigma_R}$
        else $\varphi' \leftarrow \varphi'_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \varphi'_R$
    else if $\varphi = \lfloor \varphi_c \rfloor^{\sigma}_{\{\ell_1,\ldots,\ell_n\}}$ then $\varphi' \leftarrow \lfloor \varphi'_c \rfloor^{\sigma}_{\{\ell_1,\ldots,\ell_n\}}$
    if $\varphi$ is a unit with more than one child satisfying the pre-deletion unifiability
    property then
        push $\varphi'$ onto Units;
        add $\varphi$ to $D$ ;

// Reintroduce units
$\psi^\star \leftarrow \psi'$ ;
while Units $\neq \varnothing$ do
    $\varphi' \leftarrow$ pop from Units;
    $\psi^\star_{\text{next}} \leftarrow \lfloor \psi^\star \rfloor$ ;
    while $\Gamma_{\psi^\star_{\text{next}}} \neq \psi^\star$ do
        $\psi^\star \leftarrow \psi^\star_{\text{next}}$ ;
        $\psi^\star_{\text{next}} \leftarrow \lfloor \psi^\star \rfloor$ ;
    if $\psi^\star \odot \varphi'$ is well-defined then $\psi^\star \leftarrow \psi^\star \odot \varphi'$
if $\Gamma_{\psi^\star} = \bot$ then return $\psi^\star$ else return $\psi$
```

Algorithm 6: GreedyLinearFirstOrderLowerUnits (in a single traversal)

7 Experiments

A prototype[2] of a (two-traversal) version of GreedyLinearFirstOrderLowerUnits has been implemented in the functional programming language Scala[3] as part of the Skeptik library[4].

Before evaluating this algorithm, we first generated several benchmark proofs. This was done by executing the SPASS[5] theorem prover on 2280 real first-order problems without equality of the TPTP Problem Library [6]. In order to generate

[2] Source code available at https://github.com/jgorzny/Skeptik

[3] http://www.scala-lang.org/

[4] https://github.com/Paradoxika/Skeptik

[5] http://www.spass-prover.org/

[6] http://www.cs.miami.edu/˜tptp/

pure resolution proofs, most advanced inference rules used by SPASS were disabled. The Euler Cluster at the University of Victoria[7] was used and the time limit was 300 seconds per problem. Under these conditions, SPASS was able to generate 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[8] the time needed to compress the proof ($t(\psi)$) and the compression ratio $((|\psi| - |\alpha(\psi)|)/|\psi|)$, 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 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

[7] https://rcf.uvic.ca/euler.php

[8] The raw data is available at ToDo (this link is not working) https://docs.google.com/spreadsheets/d/1F1-t2OuhypmTQhLU6yTj42aiZ5CqqaZvhVvOzeFgn0k/edit#gid=1182923972

is 3929. This results in a total compression ratio of 11.3%, which is impressive, considering 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 ToDo:(compute this percentage).

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 [8], 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 [8]. This potential trend would need to be confirmed by further experiments with more data (more proofs and longer proofs).

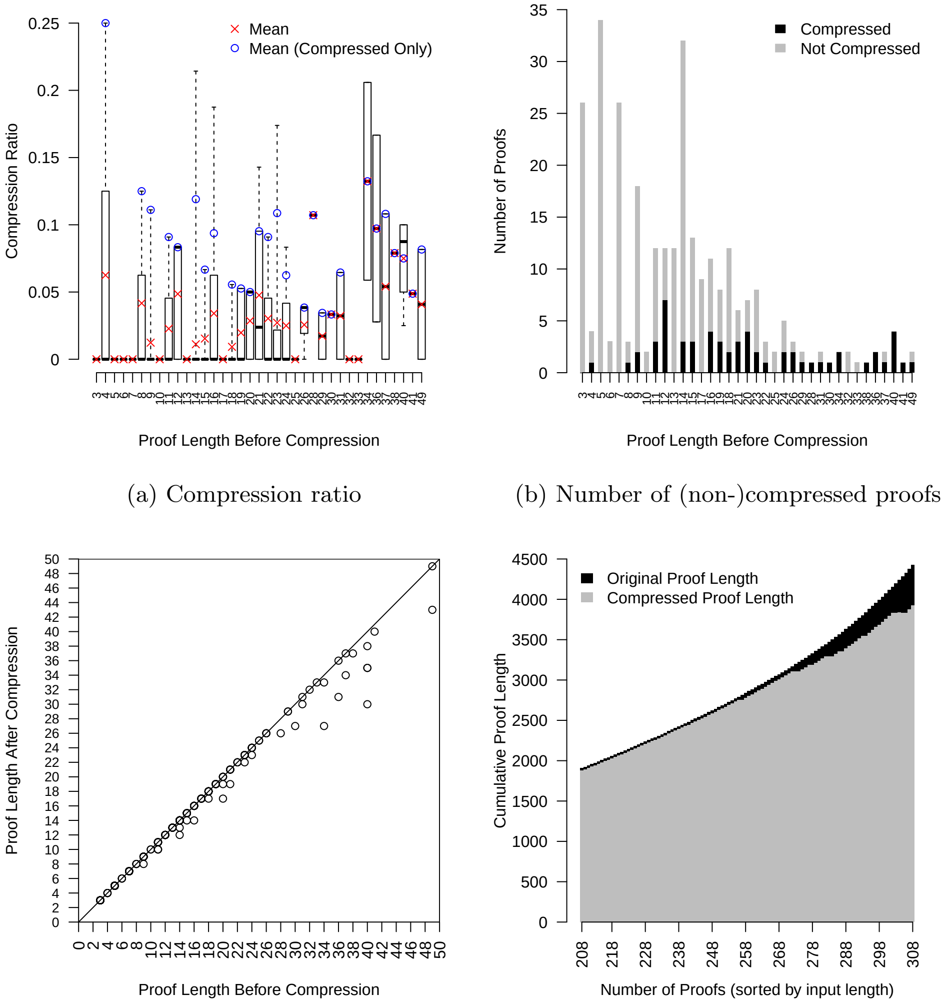
The total time needed by SPASS to generate all 308 proofs on the Euler Cluster was ToDo. The total time for `GreedyLinearFirstOrderLowerUnits` to be executed on all 308 proofs was ToDo on a simple laptop. (ToDo: make sure the total time calculation either includes or excludes parsing times for both Skeptik and SPASS. otherwise the comparison would be biased and unfair). Therefore, `GreedyLinearFirstOrderLowerUnits` is a fast algorithm. For a small overhead in time (in comparison to proving time), it may simplify the proof considerably.

8 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 complexity and the bookkeeping difficulties of `FirstOrderLowerUnits`.

The difficulties related to unit reintroduction suggest that other propositional proof compression algorithms that do not require reintroduction (e.g. `RecyclePivotsWithIntersection` [8]) 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.



(a) Compression ratio

(b) Number of (non-)compressed proofs

(c) Compressed length against input length

(d) Cumulative proof lengths

Fig. 1: Empirical evaluation results

References

1. Hasan Amjad. Compressing propositional refutations. *Electr. Notes Theor. Comput. Sci.*, 185:3–15, 2007.
2. Omer Bar-Ilan, Oded Fuhrmann, Shlomo Hoory, Ohad Shacham, and Ofer Strichman. Linear-time reductions of resolution proofs. In Hana Chockler and Alan J. Hu, editors, *Haifa Verification Conference*, volume 5394 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2008.
3. Omer Bar-Ilan, Oded Fuhrmann, Shlomo Hoory, Ohad Shacham, and Ofer Strichman. Reducing the size of resolution proofs in linear time. *STTT*, 13(3):263–272, 2011.
4. Joseph Boudou and Bruno Woltzenlogel Paleo. Compression of propositional resolution proofs by lowering subproofs. In Didier Galmiche and Dominique Larchey-Wendling, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 22th International Conference, TABLEAUX 2013, Nancy, France, September 16-19, 2013. Proceedings*, volume 8123 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2013.
5. Edmund M. Clarke and Andrei Voronkov, editors. *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, volume 6355 of *Lecture Notes in Computer Science.* Springer, 2010.
6. Scott Cotton. Two techniques for minimizing resolution proofs. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing SAT 2010*, volume 6175 of *Lecture Notes in Computer Science*, pages 306–312. Springer, 2010.
7. Pascal Fontaine, Stephan Merz, and Bruno Woltzenlogel Paleo. Exploring and exploiting algebraic and graphical properties of resolution. In *8th International Workshop on Satisfiability Modulo Theories - SMT 2010*, Edinburgh, Royaume-Uni, July 2010.
8. Pascal Fontaine, Stephan Merz, and Bruno Woltzenlogel Paleo. Compression of propositional resolution proofs via partial regularization. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2011.
9. Stefan Hetzl, Alexander Leitsch, Giselle Reis, and Daniel Weller. Algorithmic introduction of quantified cuts. *Theor. Comput. Sci.*, 549:1–16, 2014.
10. Bruno Woltzenlogel Paleo. Atomic cut introduction by resolution: Proof structuring and compression. In Clarke and Voronkov [5], pages 463–480.
11. Simone Fulvio Rollini, Roberto Bruttomesso, and Natasha Sharygina. An efficient and flexible approach to resolution proof reduction. In Sharon Barner, Ian Harris, Daniel Kroening, and Orna Raz, editors, *Hardware and Software: Verification and Testing*, volume 6504 of *Lecture Notes in Computer Science*, pages 182–196. Springer, 2011.
12. Carsten Sinz. Compressing propositional proofs by common subproof extraction. In Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *EUROCAST*, volume 4739 of *Lecture Notes in Computer Science*, pages 547–555. Springer, 2007.
13. 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.
14. Jirí Vyskocil, David Stanovský, and Josef Urban. Automated proof compression by invention of new definitions. In Clarke and Voronkov [5], pages 447–462.