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

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

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

Abstract. The recently developed `LowerUnits` algorithm compresses propositional resolution proofs generated by SAT- and SMT-solvers by lowering (i.e. postponing) resolution inferences involving unit clauses (i.e. clauses having 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. And 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. Therefore, it is a timely moment to develop methods that post-process and simplify proofs.

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]. Compression algorithms based on rearranging and sharing chains of resolution inferences have been developed in [1] and [8]. Cotton [4] 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

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

new refutation. The `Reduce&Reconstruct` algorithm [7] 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] 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 [**?**,**?**], 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 [**?**] that looks for terms that occur often in any TSTP [**?**] 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 a sophisticated algorithm that overcomes these challenges. Furthermore, in Section 6 we describe a simpler version of this algorithm, which is easier to implement and possibly more efficient, at the cost of compressing less. In Section 7 we present experimental results obtained by applying the simpler 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. $\bot$ 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 $\perp$.

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.*

$$V = V_L \cup V_R \cup \{v\}$$
$$E = E_L \cup E_R \cup \left\{\rho(\psi_L) \xrightarrow[\sigma_L]{\{\ell_L\}} v, \rho(\psi_R) \xrightarrow[\sigma_R]{\{\ell_R\}} v\right\}$$
$$\Gamma = (\Gamma_L \setminus \{\ell_L\})\,\sigma_L \cup (\Gamma_R \setminus \{\ell_R\})\,\sigma_R$$

  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.*

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

  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 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 $\lfloor \psi \rfloor^{\sigma}_{\{\ell_1, \dots \ell_n\}}$. 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 [6] 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 $(\psi \setminus \{\mu_1, \dots, \mu_n\}) \odot \mu_1 \odot \dots \odot \mu_n$, where $\mu_1, \dots, \mu_n$ 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 [**?**] 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. **??**).

Input: a proof φ
Input: D a set of subproofs
Output: a proof φ' obtained by deleting the subproofs in D from φ
Data: a map $.'$, initially empty, eventually mapping any ξ to `delete(`ξ, D`)`

1 **if** $\varphi \in D$ *or* $\rho(\varphi)$ *has no premises* **then return** φ
2 **else**
3   **let** $\varphi_L \odot_\ell \varphi_R = \varphi$;
4     $\varphi'_L \leftarrow$ `delete(`φ_L,D`)` ;
5     $\varphi'_R \leftarrow$ `delete(`φ_R,D`)` ;
6   **if** $\varphi'_L \in D$ **then return** φ'_R **else if** $\varphi'_R \in D$ **then return** φ'_L
7   **else if** $\ell \notin \Gamma_{\varphi'_L}$ **then return** φ'_L **else if** $\overline{\ell} \notin \Gamma_{\varphi'_R}$ **then return** φ'_R
8   **else return** $\varphi'_L \odot_\ell \varphi'_R$

Algorithm 1: `delete`

Input: a proof ψ
Output: a compressed proof $\psi^\star$
Data: a map $.'$: after line 4, it maps any φ to `delete(`φ, D`)`

1 Units $\leftarrow \varnothing$; `// queue to store collected units`
2 **for** *every subproof* φ*, in a bottom-up traversal of* ψ **do**
3   **if** φ *is a unit with more than one child* **then** enqueue φ in Units
4 $\psi' \leftarrow$ `delete(`ψ,Units`)` ;
`// Reintroduce units`
5 $\psi^\star \leftarrow \psi'$;
6 **for** *every unit* φ *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 discuss the challenges introduced by adapting `LowerUnits` to the first-order case. The first example illustrates how to extend `LowerUnits` to first-order logic in the obvious way. Examples 2 and on illustrate concerns that are introduced by the unification process that must be over come in order to successfully postpone resolution with a unit clause as a result of this extension.

Example 1. Resolution with a unit clause u, with literal ℓ, may be performed with a clause v provided v contains $\overline{\ell}$ and there is some unifier σ such that

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

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

for *every subproof* φ*, in a top-down traversal of* ψ **do**
  if φ *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** φ 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** $\bar{\ell} \notin \Gamma_{\varphi'_R}$ **then** $\varphi' \leftarrow \varphi'_R$
    else $\varphi' \leftarrow \varphi'_L \odot_\ell \varphi'_R$
  if φ *is a unit with more than one child* **then**
    push φ' onto Units;
    add φ to D ;

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

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

$FV(\ell\sigma) = FV(\bar{\ell})$ (or $FV(\ell) = FV(\bar{\ell}\sigma)$). After applying σ to the premises, the literals match syntactically, and so this behaves like the propositional case. Thus the notion of looking for unifiable formulas to postpone resolution with u is natural. Consider the following proof of ψ, where resolution with η_2 will be postponed:

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

After postponing resolution with η_2, the formulas $p(Y)$ (in η_1) and $p(X)$ (in η_4) will both remain after unifying these two nodes together. Unlike in the propositional case, where we could drop the repeated literal, in order to compress the proof soundly, we must first contract these literal (η'_3) into a single literal (η'_4). Then we can finish the proof by reintroducing our postponed unit, as in below.

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

Example 2. When attempting to lower a unit clause in the first order case, additional properties must be satisfied. In addition to requiring that each resolved literal is unifiable with the unit literal, we require that all such unifiers between u behave similarly in some sense. Consider the example below, where we consider postponing η_2 in the proof of ψ.

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

The literals resolved with $u = \eta_2$ are $p(a)$ (in η_1) and $p(b)$ (in η_7). If we attempt to postpone resolution, at the contraction step the clause would be $p(a), p(b)$. This clause cannot be contracted, as there is no unifier between these terms that would make their variable sets equal (in fact, there are no variables at all). Thus it would be a waste of time to attempt to postpone resolution with u, and so we require any unit we wish to lower to satisfy the following property.

Property 1. Let u be a unit clause with literal ℓ, let $\overline{\ell}_1, \ldots, \overline{\ell}_n$ contained in clauses $v_1, \ldots, v_n$ be the literals that are unified with ℓ during resolution between v_i and u in the original proof. Then for every $\overline{\ell}_i$ and $\overline{\ell}_j$ for $i \neq j$, there should be a unifier $\sigma_{i,j}$ such that $FV(\overline{\ell_i}\sigma_{i,j}) = FV(\overline{\ell_j})$ or $FV(\overline{\ell_i}) = FV(\overline{\ell_j}\sigma_{i,j})$

Example 3. Although pair-wise unifiability of literals is necessary in order to achieve some compression, it may not be enough. In the last example, the literals were checked as they appeared when they were to be resolved against a unit u which was to be postponed. However, it may be the case that they appeared this way (had a particular set of variables) because of a series of unifiers $\sigma_1, \ldots, \sigma_n$ were applied to their original form ℓ' so that $\ell = \ell'\sigma_1 \ldots \sigma_n$. For example,

$$\dfrac{\dfrac{\dfrac{\eta_1\colon \overline{\ell}', t_1, t_2 \ \vdash \qquad \eta_2\colon \ \vdash u = \ell}{\eta_3\colon (\overline{\ell}'\sigma_1), (t_1\sigma_1) \ \vdash}\sigma_1 \qquad \eta_2}{\eta_4\colon \ell = ((\overline{\ell}'\sigma_1)\sigma_2) \ \vdash}\sigma_2 \qquad \eta_2}{\psi\colon \bot}\sigma_3$$

Even though ℓ satisfied the last property, these unifiers σ_i might not be applied in the case of postponing resolution with u (or more generally, because another unit clause has been postponed). The following proof illustrates this concrete, where resolution $u = \eta_2$ is to be postponed.

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

Note that the literals resolved against u are $p(X\ Y)$ (in η_1) and $p(U\ q(V\ b))$ (in η_3) and further that the latter is $\sigma_1 = \{X \setminus U, Y \setminus V\}$ applied to $p(X\ q(Y\ b))$. These two formulas are unifiable via the substitution $\sigma_2 = \{X \setminus U, Y \setminus q(V\ b)\}$. However, if resolution with u is postponed, we will not apply the unification σ_1 that is applied to η_1, and thus the the original sources of these two formulas, $p(X\ Y)$ and $p(X\ q(Y\ b))$ can no longer be unified and contracted. Thus we require roots of resolved formulas to be pair-wise unifiable, and we call this property 2 below.

Property 2. Let u be a unit clause with literal ℓ, let $\overline{\ell}_1, \ldots, \overline{\ell}_n$ contained in clauses $v_1, \ldots, v_n$ be the literals that are unified with ℓ during resolution between v_i and u in the original proof. Let $\overline{\ell}_1^r$ be the original source of the literal $\overline{\ell}_1$, that is $\overline{\ell}_1^r \in v_{1'}$ such that there is a maximal sequence of unifications s applied to $v_{1'}$ and its children in the proof so that eventually $\overline{\ell}_1 = \overline{\ell}_1^r \sigma_1 \ldots \sigma_s$. Then for every $\overline{\ell}_i^r$ and $\overline{\ell}_j^r$ for $i \neq j$, there should be a unifier $\sigma_{i,j}$ such that after applying $\sigma_{i,j}$, $FV(\overline{\ell}_i^r \sigma_{i,j}) = FV(\overline{\ell}_j^r)$ or $FV(\overline{\ell}_i^r) = FV(\overline{\ell}_j^r \sigma_{i,j})$.

Example 4. Lastly, we note that care must be taken when lowering a valid unit in order to ensure that the proof can still be completed. In particular, postponing resolution of a clause v with a unit u may result in a *ambiguous resolution*. A resolution is said to be ambiguous if there are more than one pair of literals (ℓ_l, ℓ_r) with $\ell_l \in \psi_l$ and $\ell_r \in \psi_r$ such that (ℓ_l, ℓ_r) are unifiable. In the case of ambiguous resolution, the compression algorithm must take care to pick the appropriate pair (ℓ_l, ℓ_r) since either ℓ_l or ℓ_r might share variables with other literals in the premise nodes, and attempting to resolve away the wrong pair may ground (or otherwise modify) another literal ℓ' in such a way that ℓ' can no longer be unified with any other literal in the proof, resulting in the inability to complete the proof. Consider the following proof of ψ, where $u = \eta_2$ is the unit we want to postpone resolution with:

$$
\dfrac{\eta_2 \qquad \dfrac{\dfrac{\eta_6\colon \vdash r(W\ V) \qquad \dfrac{\eta_4\colon \vdash r(X\ c) \qquad \dfrac{\eta_1\colon p(U), r(U\ V), r(V\ U), q(V) \vdash \qquad \eta_2\colon \vdash p(c)}{\eta_3\colon r(c\ V), r(V\ c), q(V) \vdash}}{\eta_5\colon r(c\ X), q(X) \vdash}}{\eta_7\colon q(V) \vdash} \qquad \eta_8\colon p(Z) \vdash q(d)}{\eta_9\colon p(Z) \vdash}}{\psi\colon \bot}
$$

If we postpone $\eta_2\colon \vdash p(c)$, the first resolution in the compressed proof would be between the following two clauses:

$$p(U), r(U\ V), r(V\ U), q(V) \vdash \tag{1}$$

$$\vdash r(X\ c) \tag{2}$$

Both $r(U\ V)$ and $r(V\ U)$ are unifiable with the literal in (2), and so this resolution is ambiguous. If we used $(r(U\ V), r(X\ c))$, then we would use the unifier $\sigma = \{U \setminus X, V \setminus c\}$, which would result in the following resolvent clause:

$$p(X), r(V\ X), q(c) \vdash \tag{3}$$

However, the original proof does not have a clause which contains $q(c)$ in the succedent, so it would be impossible to complete the proof. On the other hand, if we chose $(r(V\ U), r(X\ c))$, we would unify with $\sigma = \{V \setminus X, U \setminus c\}$, with which we could complete the proof:

$$\dfrac{\eta'_9\colon \vdash p(c) \qquad \dfrac{\dfrac{\eta'_6\colon p(Z) \vdash q(d) \qquad \dfrac{\dfrac{\eta'_1\colon p(U), r(U\ V), r(V\ U), q(V) \vdash \qquad \eta'_2\colon \vdash r(X\ c)}{\eta'_3\colon p(c), r(c\ X), q(X) \vdash} \qquad \eta'_4\colon \vdash r(W\ V)}{\eta'_5\colon p(c), q(V) \vdash}}{\eta'_7\colon p(c), p(Z) \vdash}}{\eta'_8\colon p(c) \vdash}}{\psi\colon \bot}$$

Note that this issue is not present in the propositional case since there is no unification, and resolution cannot affect any terms in a premise that are not removed during resolution. In practice, this issue can be avoided by careful book keeping which would not be necessary in the propositional case.

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.

Input: a proof φ
Input: D a set of subproofs
Output: a proof φ' obtained by deleting the subproofs in D from φ
Data: a map $.'$, initially empty, eventually mapping any ξ to `delete(`ξ, D`)`
Data: a map $.^{\dagger}$, initially empty, eventually mapping literals to changed literals

1 **if** $\varphi \in D$ *or* $\rho(\varphi)$ *has no premises* **then return** φ
2 **else if** $\varphi = \varphi_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \varphi_R$ **then**
3     $\varphi'_L \leftarrow$ `delete(`φ_L,D`)` ;
4     $\varphi'_R \leftarrow$ `delete(`φ_R,D`)` ;
5   **for** *every* ℓ *in* Γ_{φ_L} *or* Γ_{φ_R} **do**
6         $\ell^{\dagger} \leftarrow$ the literal in $\Gamma_{\varphi'_L}$ or $\Gamma_{\varphi'_R}$ corresponding to ℓ, otherwise `none` ;
7   **if** $\varphi'_L \in D$ **then return** φ'_R **else if** $\varphi'_R \in D$ **then return** φ'_L
8   **else if** $\ell_L^{\dagger} = none$ **then return** φ'_L **else if** $\ell_R^{\dagger} = none$ **then return** φ'_R
9   **else return** $\varphi'_L \odot_{\ell_L^{\dagger} \ell_R^{\dagger}} \varphi'_R$
10 **else if** $\varphi = \lfloor \varphi_c \rfloor_{\{\ell_1,\ldots,\ell_n\}}^{\sigma}$ **then**
11     $\varphi'_c \leftarrow$ `delete(`φ_c,D`)` ;
12   **for** *every* ℓ *in* Γ_{φ_c} **do**
13         $\ell^{\dagger} \leftarrow$ the literal in $\Gamma_{\varphi'_c}$ corresponding to ℓ, otherwise `none` ;
14   **return** $\lfloor \varphi_c \rfloor_{\{\ell_1^{\dagger},\ldots,\ell_n^{\dagger}\} \setminus \{none\}}$;

Algorithm 4: `fo-delete`

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 **??**). 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

```
Input: a proof ψ
Output: a compressed proof ψ*
Data: a map .′: after line 4, it maps any φ to delete(φ, D)
Data: a map .†,mapping literals to changed literals,updated after every deletion
1  Units ← ∅;  // queue to store collected units
2  for every subproof φ, in a bottom-up traversal of ψ do
3      if φ is a unit with more than one child then enqueue φ in Units
4  s ← false;  // indicator of successful reintroduction of all units
5  while ¬s do
6      ψ′ ← delete(ψ,Units) ;
       // Reintroduce units
7      s ← true ;
8      ψ* ← ψ′ ;
9      for every unit φ in Units do
10         let {ℓ} = Γ_φ ;
11         let {ℓ_1, ..., ℓ_n} be the literals resolved against ℓ in ψ ;
12         let c = {ℓ_1†, ..., ℓ_n†}\{none} ;
13         let c↓ be the descendants of c's literals in Γ_ψ′ ;
14         if c↓'s literals are not unifiable then
15             s ← false ;
16             remove φ from Units ;
               // interrupt the for-loop
17             break;
18         else if c↓ ≠ ∅ then
19             let σ be the unifier of c↓'s literals and ℓ^c the unified literal ;
20             ψ* ← ⌊ψ*⌋^σ_{c↓} ⊙_{ℓ^c ℓ†} φ′ ;
```

Algorithm 5: `FirstOrderLowerUnits`

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 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 Simpler First-Order LowerUnits

A simple way to decrease the complexity to linear with respect to the length of the proof is to return to the ideas used in the propositional case. In particular,

Input: a proof ψ
Output: a compressed proof $\psi^\star$
Data: a map $.'$: after line 4, it maps any φ to `delete`(φ, D)

1 Units $\leftarrow \varnothing$; `// queue to store collected units and literals resolved away by each unit`
2 **for** *every subproof* φ*, in a bottom-up traversal of* ψ **do**
3   **if** φ *is a unit with more than one child* **then** enqueue (φ, ℓ_φ) in Units, where ℓ_φ is the set of literals resolved away by φ
4 `check`(Units)
5 $\psi' \leftarrow$ `simple-fo-delete`(ψ, Units) ;
`// Reintroduce units`
6 $\psi^\star \leftarrow \psi'$;
7 **for** *every unit* φ *in* Units **do**
8   **let** $\{\ell\} = \Gamma_\varphi$;
9   **let** $c = \{\ell_1, \ldots, \ell_n\}$ *be the literals resolved against* ℓ *in* ψ ;
10   **if** c*'s literals are not unifiable* **then**
11   **return** ψ
12   **else if** $c \neq \emptyset$ **then**
13   **let** σ *be the unifier of* c*'s literals and* ℓ^c *the unified literal* ;
14         $\psi^\star \leftarrow \lfloor \psi^\star \rceil_c^\sigma \odot_{\ell^c \ell^\dagger} \varphi'$;

Algorithm 6: `SimpleFirstOrderLowerUnits`

by performing a traversal to collect the units of a proof, and then optimistically deleting units, some compression can often be achieved. By ignoring whether or not a unit satisfies Property 2, we can attempt to lower it, and should compression fail because deletions changed the substitutions to the point where contraction was not possible, we simply return the original proof.

Algorithm 6 works similarly to the propositional algorithm. It first performs a bottom up traversal to collect potential units and the literals that are resolved away from by those units, adding the units to a queue (line 1). As seen in Examples 2 in Section 4, unification of the resolved away literals is necessary, so it performs a check to make sure these literals satisfy Property 1 (line 4). If it succeeds, it attempts to re-introduce all the removed units at the bottom of the proof, where it attempts to compress the literals that would be resolved away by each unit (lines 6-15). Note that this requires the implementation to track which literals should be resolved against each unit. In order to avoid traversing the proof to find these again after the deletion of every potential unit (as is done in Algorithm 5), we use a modified `delete` function, called `simple-fo-delete`, which is the same as Algorithm 1 except with line 6 changed to the following:

if $\varphi'_L \in D$ **then return** $(\varphi'_L \sigma_R)$ **else if** $\varphi'_R \in D$ **then return** $(\varphi'_R \sigma_R)$

`simple-fo-delete` is designed to reduce the complexity of tracking literals. `simple-fo-delete` behaves much more closely to the propositional case and requires none of the additional data structures required by `fo-delete`. In this function, when a unit node is returned, instead of returning the opposite node (respectively ψ'_L or ψ'_R, line 6) in the resolution (which is done in the propositional case), or tracking the literals (which is done in `fo-delete`), we return the opposite node with σ_L (respectively σ_R) applied to it. In this way, the literals not resolved with the unit will look like they would have in the original proof, and the literal which was not resolved due to the deletion looks like it is syntactically equal with the unit literal at this stage. The fact that the other literals look like they did in the original proof is key: now resolution in the compressed proof can use the old literals, which should appear as they before, and not worry about choosing the wrong literal in case of ambiguous resolution.

Additionally, by modifying delete in this manner we can longer guarantee that Property 2 is satisfied. Property 2 so the appearance of literals that were to be resolved away from a unit clause may have changed, preventing completion of the proof. If this happens `SimpleFirstOrderLowerUnits` will attempt to re-introduce this node and fail, returning the original input proof (line 12). As a result, some proofs that can be compressed are returned unmodified, but those that do not require this additional property can be compressed much more quickly.

7 Experiments

`SimpleFirstOrderLowerUnits` has been implemented as a prototype[2] in the functional programming language Scala[3] as part of the Skeptik library[4].

The algorithm has been applied to 308 proofs produced by the SPASS[5] theorem prover on unsatisfiable benchmarks from the TPTP Problem Library[6]. The proofs used were restricted to those which could be solved within 300 seconds by SPASS on the Euler Cluster at the University of Victoria[7] using only the contraction and unifying resolution inference rules. The experiments were executed on a laptop (2.8GHz Intel Core i7 processor with 4 GB of RAM (1333MHz DDR3) available to the Java Virtual Machine), and the prototype implementation performed well on this system.

For each proof ψ (with the result of `SimpleFirstOrderLowerUnits` applied to the proof denoted by $\alpha(\psi)$), the time to compress the proof ($t(\psi)$) and the resolution compression ratio $((|\psi|_R - |\alpha(\psi)|_R)/|\psi|_R)$ were measured[8], where $|\psi|_R$ indicates the number of resolution inference rules in the proof ψ.

Figure 1 (a) shows the average compression ratio sorted by proof length without the uncompressed proofs, and the average including the uncompressed

---

[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/
[7] https://rcf.uvic.ca/euler.php
[8] The raw data is available at https://docs.google.com/spreadsheets/d/1F1-t2OuhypmTQhLU6yTj42aiZ5CqqaZvhVvOzeFgn0k/edit#gid=1182923972

proofs (b). Uncompressed proofs are those which had no valid units to lower or for which `SimpleFirstOrderLowerUnits` returned the original proof (there were 14 such proofs). In the longer proofs, there appear to be more valid units to lower, and these could reduce the number of resolutions in the proof by as much as 20%.

Figure 1 (c) shows the number of proofs of each length compressed and the combined number of proof nodes before and after compression (e). Note that there is a trend that as the length of the proof increases, a higher percentage of proofs can be compressed. This is expected, as larger proofs are more likely to include more valid unit clauses. The total number of proof nodes starts to drop of quickly once these larger proofs are accounted for. Figure 1 (f) provides a better look at this behavior on the left; as a result of about the last 20 proofs, over 500 proof nodes are saved. Figure 1 (d) shows the original proof length compared against the compressed proof length on the right.

8 Conclusions and Future Work

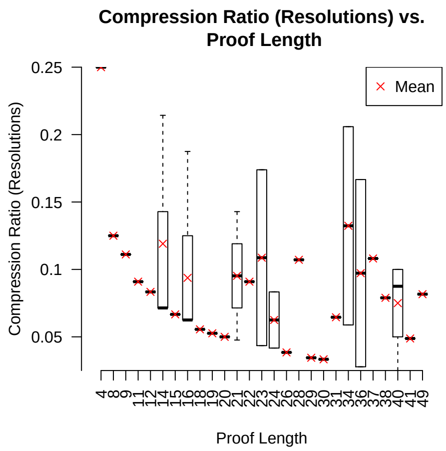
ToDo: by Bruno

`LowerUnivalents`, the algorithm presented here, has been shown in the previous section to compress more than `LowerUnits`. This is so because, as demonstrated in Proposition **??**, the set of subproofs it lowers is always a superset of the set of subproofs lowered by `LowerUnits`. It might be possible to lower even more subproofs by finding a characterization of (efficiently) lowerable subproofs broader than that of univalent subproofs considered here. This direction for future work promises to be challenging, though, as evidenced by the nontriviality of the optimizations discussed in Section **??** for obtaining a linear-time implementation of `LowerUnivalents`.

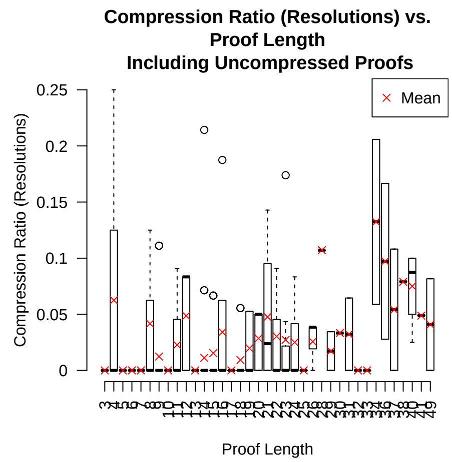
As discussed in Section **??**, the proposed algorithm can be embedded in the deletion traversal of other algorithms. As an example, it has been shown that the combination of `LowerUnivalents` with `RPI`, compared to the sequential composition of `LowerUnits` after `RPI`, results in a better compression ratio with only a small processing time overhead (Figure **??**). Other compression algorithms that also have a subproof deletion or reconstruction phase (e.g. `Reduce&Reconstruct`) could probably benefit from being combined with `LowerUnivalents` as well.

References

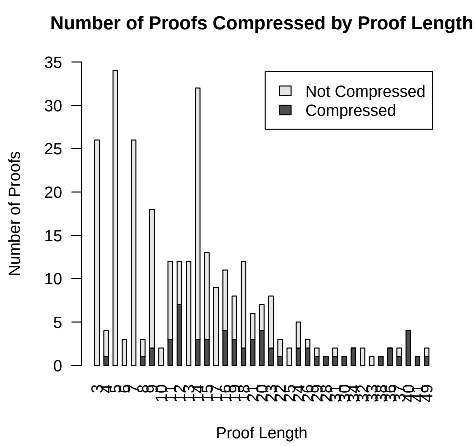
1. Amjad, H.: Compressing propositional refutations. Electr. Notes Theor. Comput. Sci. 185, 3–15 (2007)
2. Bar-Ilan, O., Fuhrmann, O., Hoory, S., Shacham, O., Strichman, O.: Linear-time reductions of resolution proofs. In: Chockler, H., Hu, A.J. (eds.) Haifa Verification Conference. Lecture Notes in Computer Science, vol. 5394, pp. 114–128. Springer (2008)
3. Bar-Ilan, O., Fuhrmann, O., Hoory, S., Shacham, O., Strichman, O.: Reducing the size of resolution proofs in linear time. STTT 13(3), 263–272 (2011)



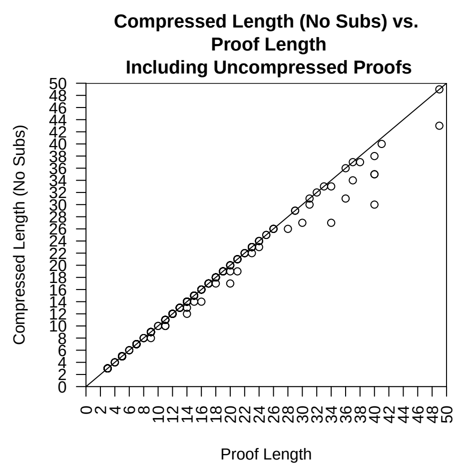
(a) Average compression (only success)



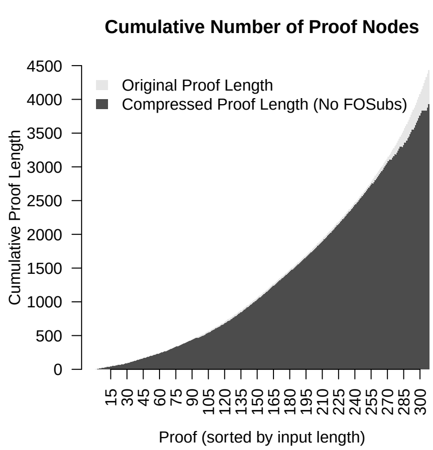
(b) Overall average compression



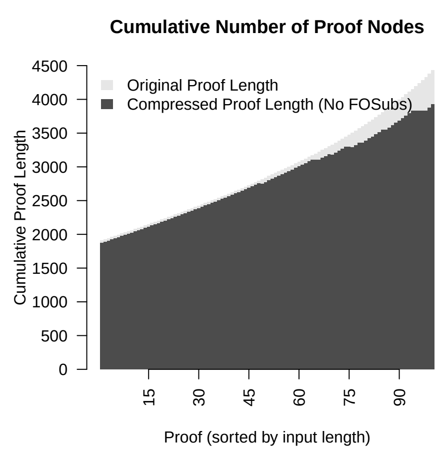
(c) Number of proofs of each length compressed



(d) Compressed length against input length



(e) Total proof nodes



(f) Node reduction resulting from 100 largest proofs

Fig. 1: Empirical evaluation results.

4. Cotton, S.: Two techniques for minimizing resolution proofs. In: Strichman, O., Szeider, S. (eds.) Theory and Applications of Satisfiability Testing SAT 2010, Lecture Notes in Computer Science, vol. 6175, pp. 306–312. Springer (2010)
5. Fontaine, P., Merz, S., Woltzenlogel Paleo, B.: Exploring and exploiting algebraic and graphical properties of resolution. In: 8th International Workshop on Satisfiability Modulo Theories - SMT 2010. Edinburgh, Royaume-Uni (Jul 2010), http://hal.inria.fr/inria-00544658
6. Fontaine, P., Merz, S., Woltzenlogel Paleo, B.: Compression of propositional resolution proofs via partial regularization. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE. Lecture Notes in Computer Science, vol. 6803, pp. 237–251. Springer (2011)
7. Rollini, S.F., Bruttomesso, R., Sharygina, N.: An efficient and flexible approach to resolution proof reduction. In: Barner, S., Harris, I., Kroening, D., Raz, O. (eds.) Hardware and Software: Verification and Testing, Lecture Notes in Computer Science, vol. 6504, pp. 182–196. Springer (2011)
8. Sinz, C.: Compressing propositional proofs by common subproof extraction. In: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (eds.) EUROCAST. Lecture Notes in Computer Science, vol. 4739, pp. 547–555. Springer (2007)