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. $\perp$ 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. $\mathrm{FV}(\ell)$, $\mathrm{FV}(\Gamma)$) 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. $E \subset V \times \mathcal{L} \times \mathcal{S} \times V$ 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 atom ℓ such that $\ell = \ell_L\sigma_L = \overline{\ell_R}\sigma_R$ or $\ell = \overline{\ell_L}\sigma_L = \ell_R\sigma_R$ is called the resolved atom*.*
- ***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, letting $\ell = \ell_k\sigma$ (for any $k \in \{1, \ldots, n\}$), $\lfloor\psi\rfloor_\sigma^\ell$ denotes a proof $\langle V, E, \Gamma\rangle$ s.t.*

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

  where v is a new (contraction) node 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 $\psi_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \psi_R$ and $\lfloor \psi \rfloor$ instead of $\lfloor \psi \rfloor_{\sigma}^{\ell}$. 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 $\psi_L \odot_{\ell \bar{\ell}}^{\emptyset \emptyset} \psi_R$ 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.

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`)`

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

Algorithm 1: `delete`

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 $(\psi \setminus \{\mu_1, \ldots, \mu_n\}) \odot \mu_1 \odot \ldots \odot \mu_n$, where $\mu_1, \ldots, \mu_n$ 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 Algorithm 3.

[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 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`

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** $\overline{\ell} \notin \Gamma_{\varphi'_R}$ **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 $\{\overline{\ell}\} = \Gamma_{\varphi}$;
    if $\ell \in \Gamma_{\psi^{\star}}$ **then** $\psi^{\star} \leftarrow \psi^{\star} \odot_{\ell} \varphi'$

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

4 First-Order Challenges

TODO by Jan (just writing some ideas so far–not yet final by any means)
todo: define the notion of set of literals resolved against a unit? might make things clearer

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 particular 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 $\ell\sigma$ contains the same variables as $\overline{\ell}$ (or such that $\overline{\ell}\sigma$ contains the same variables as ℓ). After applying σ to the resolvents, 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 we will postpone resolution with η_2:

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

In order to postpone 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 formula, in order to compress the proof soundly, we must first contract these formulas (η_3') into a single literal (η_4'). Then we can finish the proof by reintroducing our postponed resolution, 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 the requirement that there is a unifier σ that makes the unit clause syntactically equal to the formula it resolves away, 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.

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

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. Things must be pair-wise unifiable... TODO: this? Or leave this out and talk about it section 5?

Example 3. Although pair-wise unifiability of literals is necessary in order to achieve some compression, it is not 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 $\ell = \ell'\sigma_1 \ldots \sigma_n$ was pairwise unifiable (the last property was met) with all other literals, but these unifies σ_i would not be applied in the case of postponing resolution with u (or another unit clause). The following proof illustrates this case, where we would consider lowering $u = \eta_2$.

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

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

Property 2. Descendants must be pair-wise unifiable... TODO: this? Or leave this out and talk about it section 5?

Example 4. Consider the following, which shows the dangers of ambiguous resolution in the first order case:

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

Which if we lower $\eta_2\colon \vdash p(c)$, we would attempt to resolve the following two sequents

$$p(U), r(U\ V), r(V\ U), q(V) \vdash$$

$$\vdash r(X\ c)$$

Where we would have to be careful. If we used $r(U\ V)$, then we would use the unifier $U- > X, V- > C$, which would result in the following:

$$p(X), r(V\ X), q(c) \vdash$$

but the original proof does not have a method for resolving away $q(c)$, so we would not be able to complete the proof. On the other hand, if we chose $r(V\ U)$, we would unify with $V- > X, U- > c$, with which we could complete the proof:

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

A method to avoid this issue is discussed in section 6

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 get possibly more general literals in the root clause of the output proof.

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

if $\varphi \in D$ *or* $\rho(\varphi)$ *has no premises* **then return** φ
else
  let $\varphi_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \varphi_R = \varphi$;
  $\varphi'_L \leftarrow$ `delete(`φ_L,D`)` ;
  $\varphi'_R \leftarrow$ `delete(`φ_R,D`)` ;
  for *every* ℓ *in* Γ_{φ_L} *or* Γ_{φ_R} **do**
    $\ell^{\dagger} \leftarrow$ the literal in $\Gamma_{\varphi'_L}$ or $\Gamma_{\varphi'_R}$ corresponding to ℓ, otherwise `none` ;
  if $\varphi'_L \in D$ **then return** φ'_R **else if** $\varphi'_R \in D$ **then return** φ'_L
  else if $\ell^{\dagger}_L = none$ **then return** φ'_L **else if** $\ell^{\dagger}_R = none$ **then return** φ'_R
  else return $\varphi'_L \odot_{\ell^{\dagger}_L \ell^{\dagger}_R} \varphi'_R$

Algorithm 4: `fo-delete`

ToDo: The following paragraph is just a speculation. I still have to check it. I strongly suspect that, in the first-order case, we are not free to collect the units either in top-down or bottom-up traversal, as we were in the propositional case. Only one of them is the correct way (if we want to correctly overcome all challenges). But I still don't know which one.

The second challenge is much harder to overcome. In the propositional case, collecting units and deleting units can be done in two distinct 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. Therefore, there is no hope for a single-traversal first-order `LowerUnits` algorithm generalizing 3. The collection of units must be done in a bottom-up traversal (as in 2), so that when we have to test whether a unit η can be lowered (in line 3), we can run the deletion algorithm to see what will happen with the root of the proof when we delete η and all other units already collected below η. This unlocks the apparent deadlock, at the high cost of having to run the deletion algorithm once for every tested unit. The worst-case run-time complexity is, therefore, quadratic in the length of the input proof to be compressed.

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 ToDo ;
2 Units $\leftarrow \varnothing$;   `// queue to store collected units`
3 **for** *every subproof* φ*, in a bottom-up traversal of* ψ **do**
4   **if** φ *is a unit with more than one child* **then** enqueue φ in Units
5 $\psi' \leftarrow$ `delete(`ψ,Units`)` ;
  `// Reintroduce units`
6 $\psi^{\star} \leftarrow \psi'$;
7 **for** *every unit* φ *in* Units **do**
8   **let** $\{\ell\} = \Gamma_{\varphi}$;
9   **if** $\overline{\ell} \in \Gamma_{\psi'}$ **then** $\psi^{\star} \leftarrow \psi^{\star} \odot_{\ell} \varphi$

Algorithm 5: `FirstOrderLowerUnits`

6 A Simpler First-Order LowerUnits

ToDo by Jan

Recall example 4. In order to avoid this, we introduce a proof rule that applies a substitution. So that we would get the following proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\eta_1\colon p(U), r(U\ V), r(V\ U), q(V) \vdash}{\eta_2\colon p(c), r(c\ V), r(V\ c), q(V) \vdash} \quad \eta_3\colon \vdash r(X\ c)}{\eta_4\colon p(c), r(c\ X), q(X) \vdash} \quad \eta_5\colon \vdash r(W\ V)}{\eta_6\colon p(c), q(V) \vdash} \quad \eta_7\colon p(Z) \vdash q(d)}{\eta_8\colon p(c), p(Z) \vdash}}{\eta_9\colon p(c) \vdash} \quad \eta_{10}\colon \vdash p(c)}{\psi\colon \bot}$$

Now $r(V, c)$ appears in the first left resolvent, which was the left aux formula in the original proof. Thus, the implementation can find that formula, and choose it in order to resolve the ambiguous resolution, instead of guessing a formula from the left resolvent that unifies with the right resolvent, which might go wrong.

TODO: Explain where the sub came from.

TODO: define the rule formally here?

TODO: describe when the rule is invoked in the implementation

7 Experiments

ToDo by Jan

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

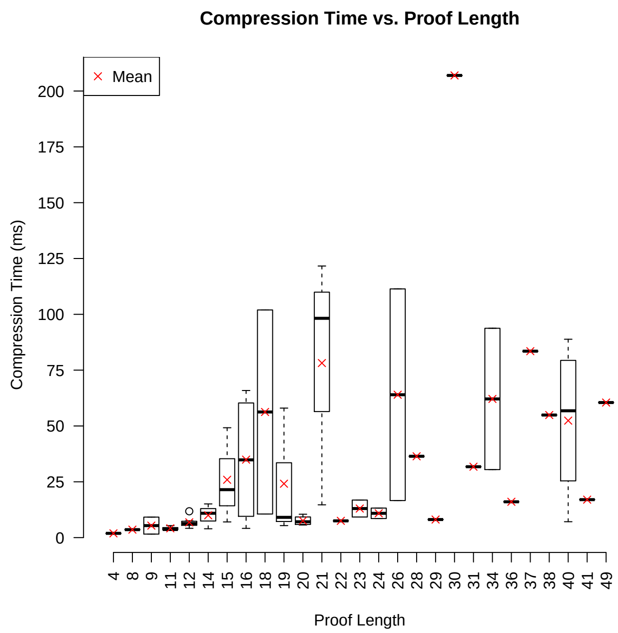
The algorithm has been applied to **308** proofs produced by the SPASS[5] theorem prover on unsatisfiable benchmarks from the TPTP Problem Library[6]. The proofs used were restricted to those which could be solved within 300 seconds by SPASS on the Euler Cluster at the University of Victoria[7] using only the contraction and unifying resolution inference rules.

For each proof ψ (with the result of `LowerUnits` applied to the proof denoted by $\alpha(\psi)$), the time to compress the proof ($t(\psi)$), the compression ratio $((|\psi| - |\alpha(\psi)|)/|\psi|)$, the resolution compression ratio $((|\psi|_R - |\alpha(\psi)|_R)/|\psi|_R)$, the compression speed $((|\psi| - |\alpha(\psi)|)/t(\psi))$, and resolution compression speed

[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.verit-solver.org/
[6] http://www.cs.miami.edu/ tptp/
[7] https://rcf.uvic.ca/euler.php

$((|\psi|_R - |\alpha(\psi)|_R)/t(\psi))$ were measured[8], where $|\psi|_R$ indicates the number of resolution inference rules in the proof ψ.

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. Figure **??** shows the compression time $t(\psi)$ for each proof, sorted by proof length, and figure **??** (respectively figure **??**) shows the compression speed (respectively resolution compression speed) for each proof, also sorted by proof length.

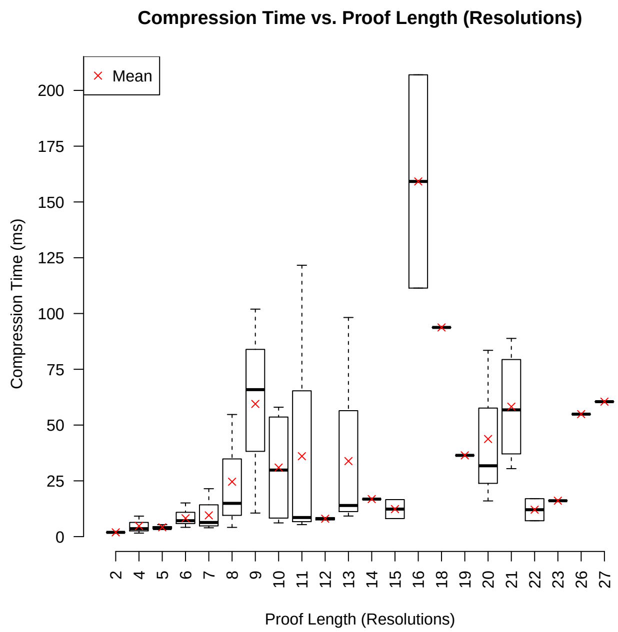


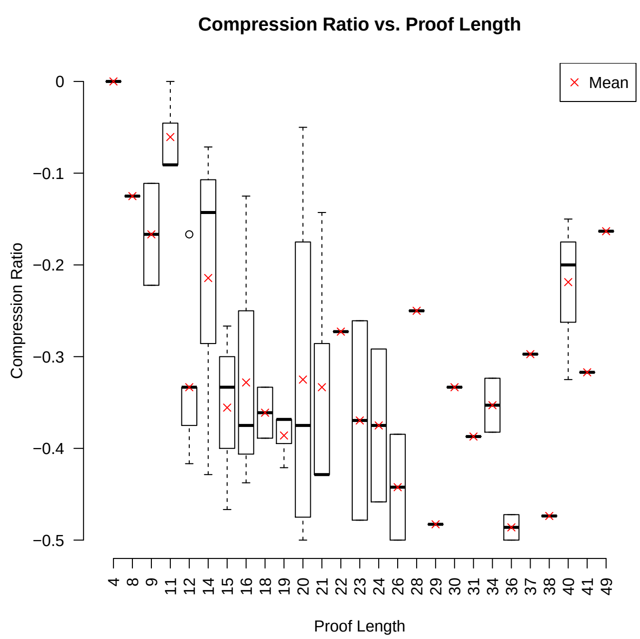
8 Conclusions and Future Work

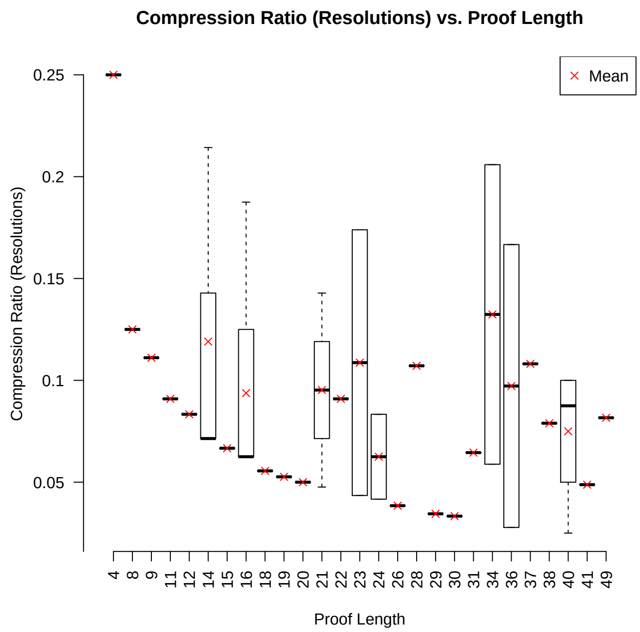
ToDo: by Bruno

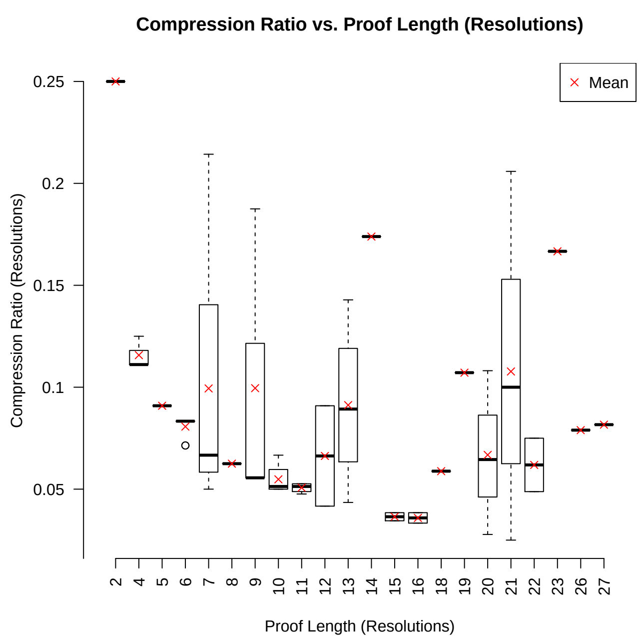
`LowerUnivalents`, the algorithm presented here, has been shown in the previous section to compress more than `LowerUnits`. This is so because, as demonstrated in Proposition **??**, the set of subproofs it lowers is always a superset of the set of subproofs lowered by `LowerUnits`. It might be possible to lower even more subproofs by finding a characterization of (efficiently) lowerable subproofs broader than that of univalent subproofs considered here. This direction

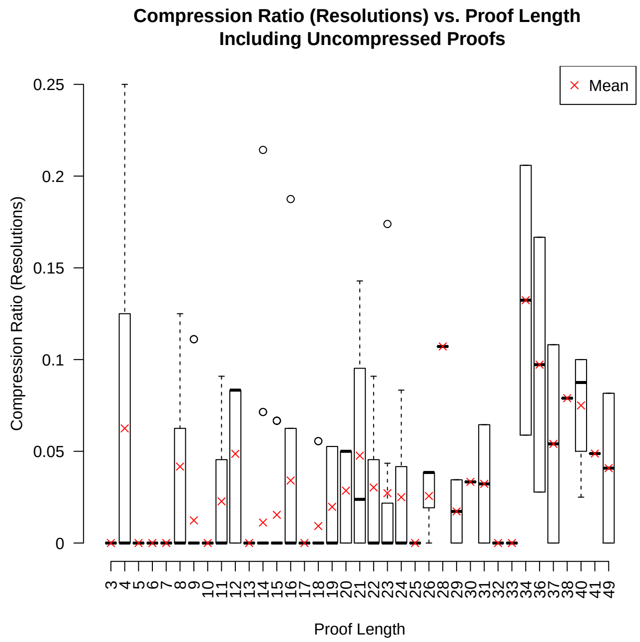
[8] The raw data is available at https://docs.google.com/spreadsheets/d/1F1-t2OuhypmTQhLU6yTj42aiZ5CqqaZvhVvOzeFgn0k/edit#gid=1182923972

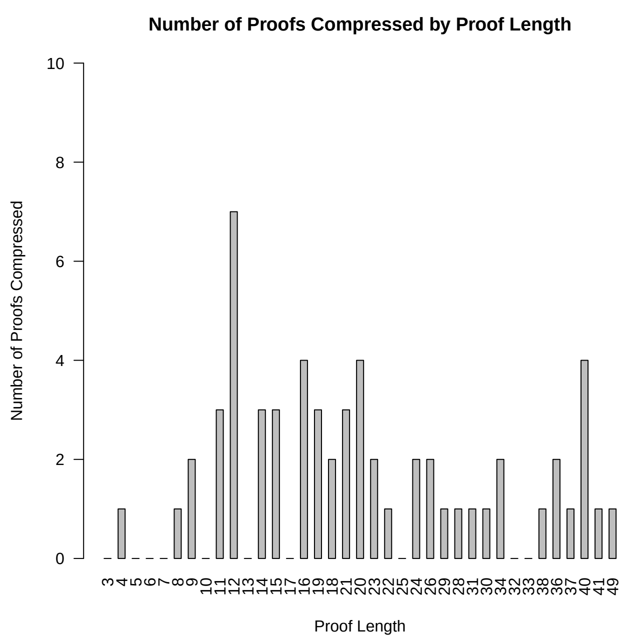

Compression Time vs. Proof Length (Resolutions)
Mean
Compression Time (ms)
0
25
50
75
100
125
150
175
200
2
4
5
6
7
8
9
10
11
12
13
14
15
16
18
19
20
21
22
23
26
27
Proof Length (Resolutions)

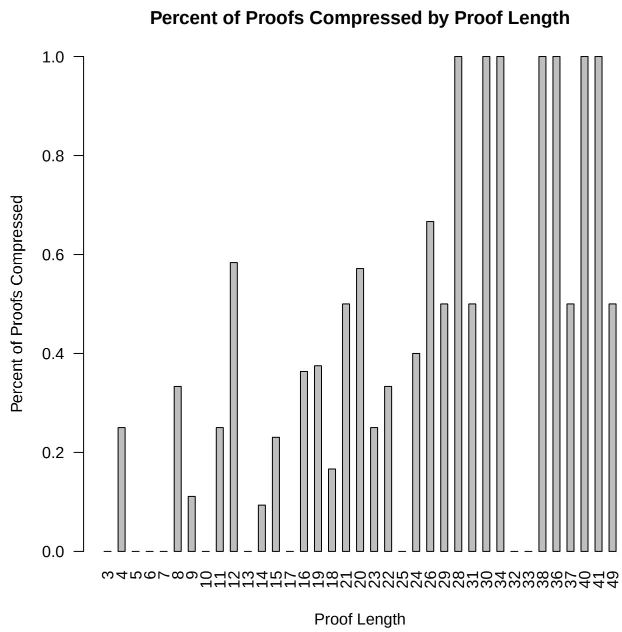

Compression Ratio vs. Proof Length
Mean
Compression Ratio
0
−0.1
−0.2
−0.3
−0.4
−0.5
4
8
9
11
12
14
15
16
18
19
20
21
22
23
24
26
28
29
30
31
34
36
37
38
40
41
49
Proof Length

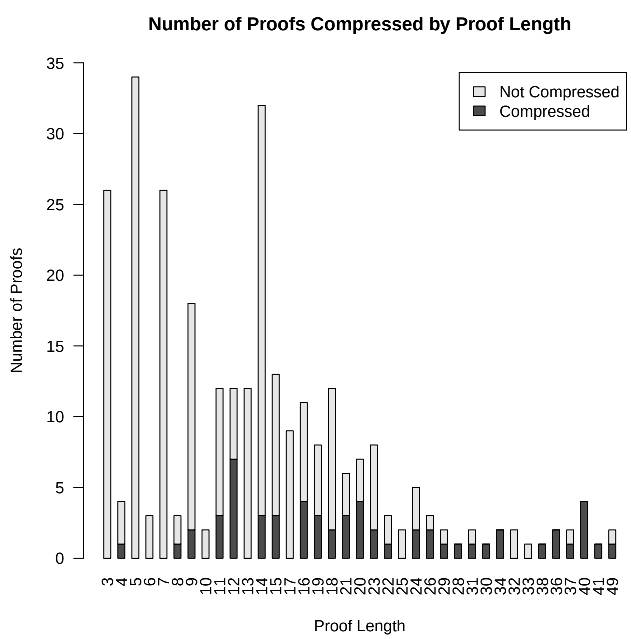

Compression Ratio (Resolutions) vs. Proof Length
Mean
Compression Ratio (Resolutions)
0.25
0.2
0.15
0.1
0.05
4 8 9 11 12 14 15 16 18 19 20 21 22 23 24 26 28 29 30 31 34 36 37 38 40 41 49
Proof Length

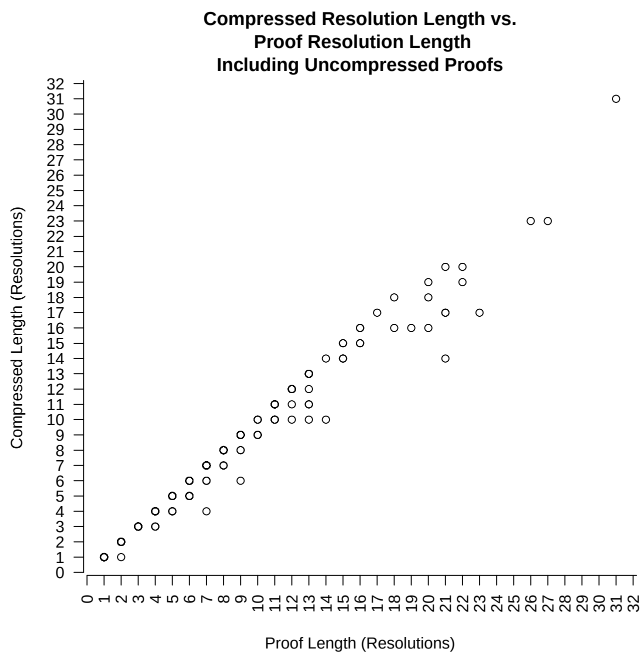

Compression Ratio vs. Proof Length (Resolutions)
Mean
Compression Ratio (Resolutions)
0.25
0.2
0.15
0.1
0.05
2 4 5 6 7 8 9 10 11 12 13 14 15 16 18 19 20 21 22 23 26 27
Proof Length (Resolutions)

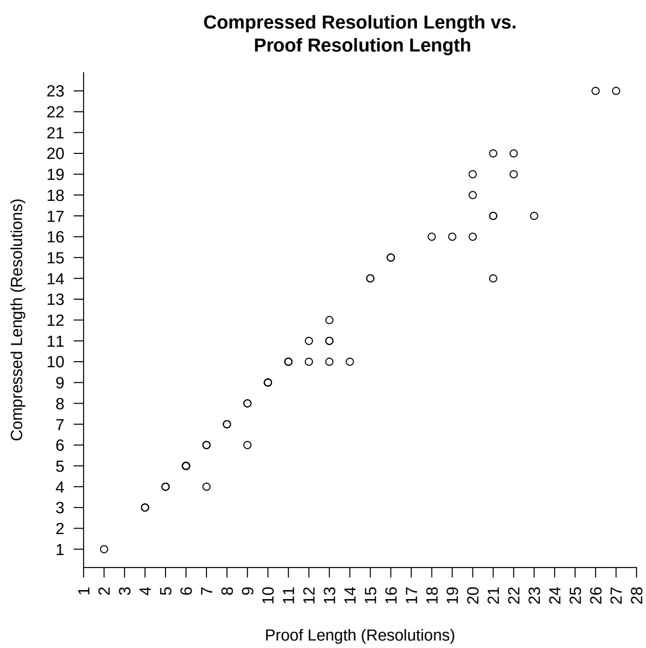

Compression Ratio (Resolutions) vs. Proof Length
Including Uncompressed Proofs
Mean
Compression Ratio (Resolutions)
0.25
0.2
0.15
0.1
0.05
0
3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 28 29 30 31 32 33 34 36 37 38 40 41 49
Proof Length

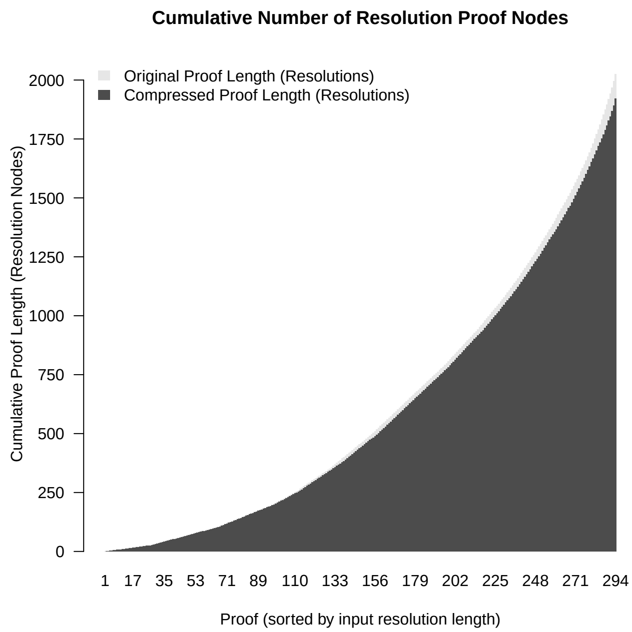

Number of Proofs Compressed by Proof Length
Number of Proofs Compressed
10
8
6
4
2
0
3 4 5 6 7 8 9 10 11 12 13 14 15 17 16 19 18 21 20 23 22 25 24 26 29 28 31 30 34 32 33 38 36 37 40 41 49
Proof Length

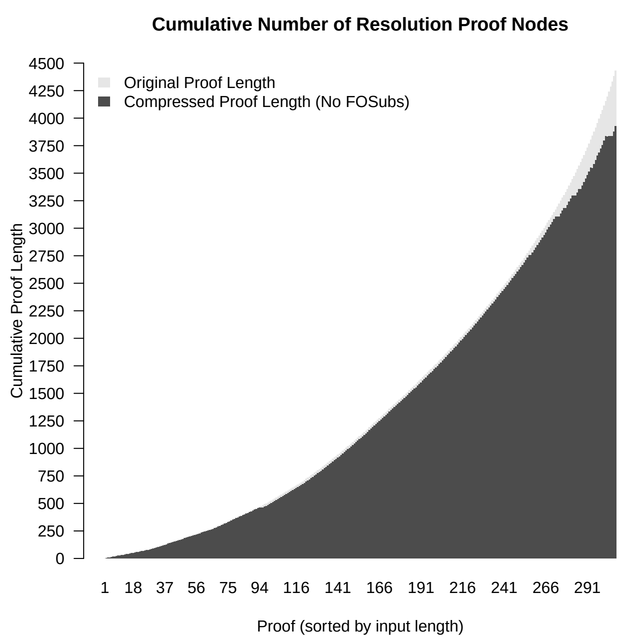

Percent of Proofs Compressed by Proof Length
Percent of Proofs Compressed
0.0
0.2
0.4
0.6
0.8
1.0
3 4 5 6 7 8 9 10 11 12 13 14 15 17 16 19 18 21 20 23 22 25 24 26 29 28 31 30 34 32 33 38 36 37 40 41 49
Proof Length

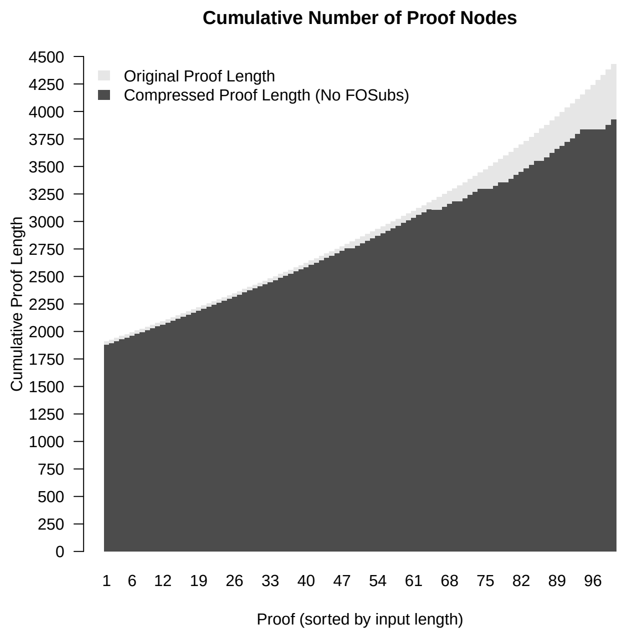

Number of Proofs Compressed by Proof Length
Not Compressed
Compressed
Number of Proofs
0
5
10
15
20
25
30
35
3 4 5 6 7 8 9 10 11 12 13 14 15 17 16 19 18 21 20 23 22 25 24 26 29 28 31 30 34 32 33 38 36 37 40 41 49
Proof Length

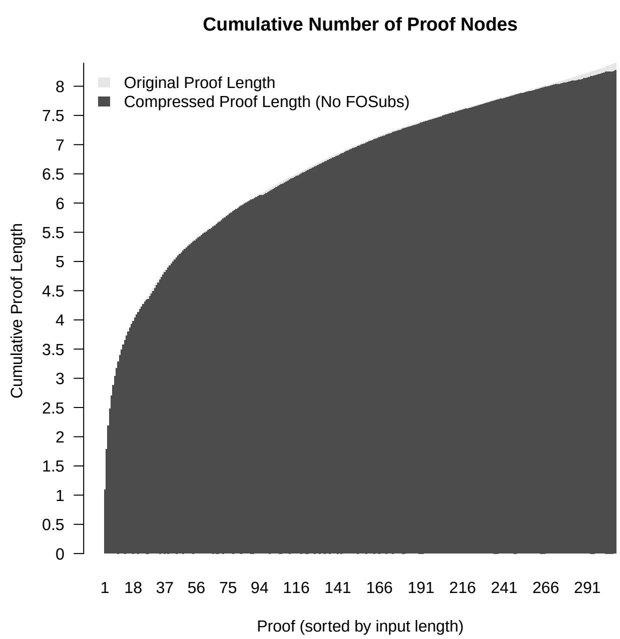

Compressed Resolution Length vs.
Proof Resolution Length
Including Uncompressed Proofs
Compressed Length (Resolutions)
Proof Length (Resolutions)

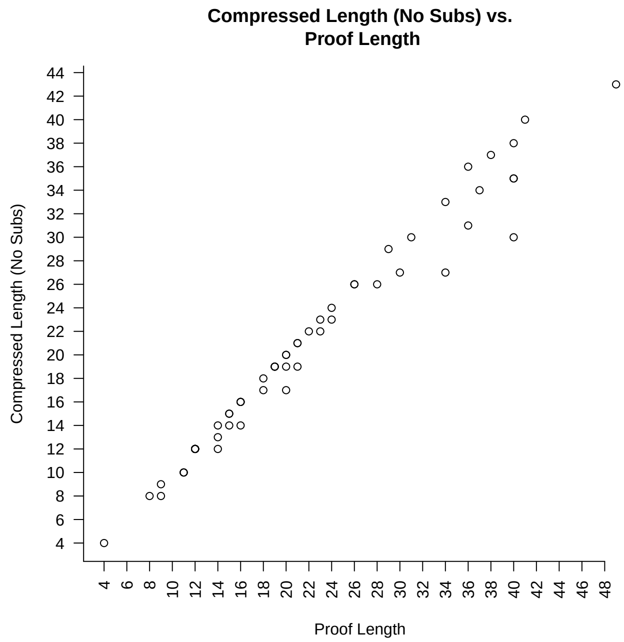

Compressed Resolution Length vs.
Proof Resolution Length
Compressed Length (Resolutions)
Proof Length (Resolutions)

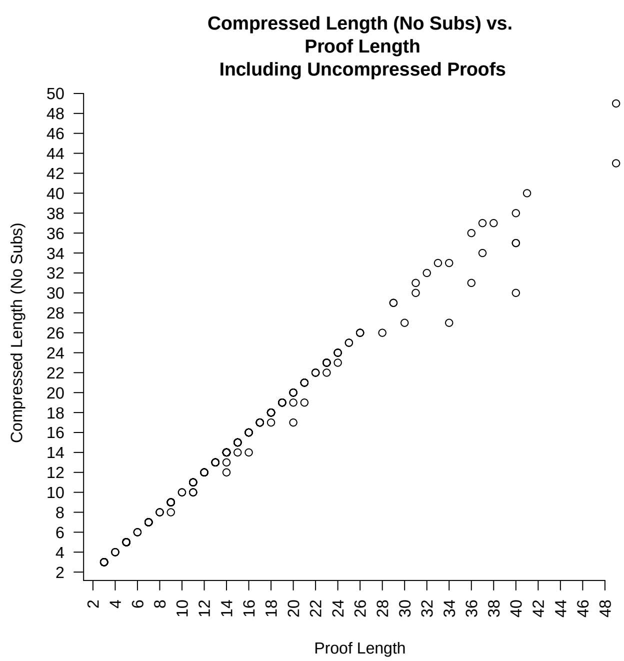

Cumulative Number of Resolution Proof Nodes
Original Proof Length (Resolutions)
Compressed Proof Length (Resolutions)
Cumulative Proof Length (Resolution Nodes)
0
250
500
750
1000
1250
1500
1750
2000
1 17 35 53 71 89 110 133 156 179 202 225 248 271 294
Proof (sorted by input resolution length)


Cumulative Number of Resolution Proof Nodes
Original Proof Length
Compressed Proof Length (No FOSubs)
Cumulative Proof Length
0
250
500
750
1000
1250
1500
1750
2000
2250
2500
2750
3000
3250
3500
3750
4000
4250
4500
1 18 37 56 75 94 116 141 166 191 216 241 266 291
Proof (sorted by input length)


Cumulative Number of Proof Nodes
Original Proof Length
Compressed Proof Length (No FOSubs)
Cumulative Proof Length
4500
4250
4000
3750
3500
3250
3000
2750
2500
2250
2000
1750
1500
1250
1000
750
500
250
0
1 6 12 19 26 33 40 47 54 61 68 75 82 89 96
Proof (sorted by input length)


Cumulative Number of Proof Nodes
Original Proof Length
Compressed Proof Length (No FOSubs)
Cumulative Proof Length
8
7.5
7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1 18 37 56 75 94 116 141 166 191 216 241 266 291
Proof (sorted by input length)


Compressed Length (No Subs) vs.
Proof Length
Compressed Length (No Subs)
Proof Length


Compressed Length (No Subs) vs.
Proof Length
Including Uncompressed Proofs
Compressed Length (No Subs)
Proof Length

for future work promises to be challenging, though, as evidenced by the non-triviality 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)
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)