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, ...), constant symbols (e.g. a, b, c, a_1, a_2, ...), function symbols of every arity (e.g f, g, f_1, f_2, ...) and predicate symbols of every arity (e.g. p, q, p_1, p_2,...). A *term* is any variable, constant or the application of an n-ary function symbol to n terms. An *atomic formula* (*atom*) is the application of an n-ary predicate symbol to n terms. A *literal* is an atom or the negation of an atom. The *complement* of a literal ℓ is denoted $\overline{\ell}$ (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 $\bot$.

Definition 1 (First-Order Resolution Proof).
A directed acyclic graph $\langle V, E, \Gamma\rangle$, where V is a set of nodes and E is a set of edges labeled by literals and substitutions (i.e. $E \subset V \times \mathcal{L} \times \mathcal{S} \times V$ 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.*

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

  where v is a new (resolution) node and $\rho(\varphi)$ denotes the root node of φ.
- ***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.*

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

  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. 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 $\lfloor\psi\rfloor^{\ell}_{\sigma}$. 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 $\psi_L \odot^{\emptyset\emptyset}_{\ell\bar{\ell}} \psi_R$ 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.

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`

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 ← ∅;   // 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  ψ' ← delete(ψ,Units) ;
   // Reintroduce units
5  ψ* ← ψ' ;
6  for every unit φ in Units do
7      let {ℓ} = Γ_φ ;
8      if ℓ̄ ∈ Γ_ψ' then  ψ* ← ψ* ⊙_ℓ φ
```

Algorithm 2: `LowerUnits`

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

```
1  D ← ∅;   // set for storing subproofs that need to be deleted
2  Units ← ∅;   // stack for storing collected units
3  for every subproof φ, in a top-down traversal of ψ do
4      if φ is an axiom then φ' ← φ else
5          let φ_L ⊙_ℓ φ_R = φ ;
6          if  φ_L ∈ D and φ_R ∈ D then  add φ to D  else if φ_L ∈ D then
           φ' ← φ'_R  else if  φ_R ∈ D then  φ' ← φ'_L
7          else if ℓ ∉ Γ_φ'_L then  φ' ← φ'_L  else if ℓ̄ ∉ Γ_φ'_R then  φ' ← φ'_R
8          else  φ' ← φ'_L ⊙_ℓ φ'_R
9      if φ is a unit with more than one child then
10         push φ' onto Units;
11         add φ to D ;
   // Reintroduce units
12 ψ* ← ψ' ;
13 while Units ≠ ∅ do
14     φ' ← pop from Units;
15     let {ℓ̄} = Γ_φ ;
16     if ℓ ∈ Γ_ψ*  then  ψ* ← ψ* ⊙_ℓ φ'
```

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)

In this section, we discuss additional requirements for lowering a unit formula in the first order case that are not required in the propositional case.

Example 1. The following example shows why we want to check formula unifiability with the unit we're trying to lower.

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

which we could compress to

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

Example 2. The following example shows why we must check pair-wise unifiability with the literals resolved against the unit we're trying to lower.

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

Example 3. The following shows why the above is not necessarily enough (we must check the original sources of the aux formulas, and see if those can be contracted), otherwise we might not save anything.

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

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

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

Input: a proof φ
Input: D a set of subproofs
Output: a proof φ' obtained by deleting the subproofs in D from φ

if $\varphi \in D$ *or* $\rho(\varphi)$ *has no premises* **then return** φ
else
  let φ_L *and* φ_R be such that $\varphi = \varphi_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \varphi_R$;
  let $\varphi'_L =$ `delete(`φ_L`,`D`)` ;
  let $\varphi'_R =$ `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 $\varphi'_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \varphi'_R$

Algorithm 4: `fo-delete`

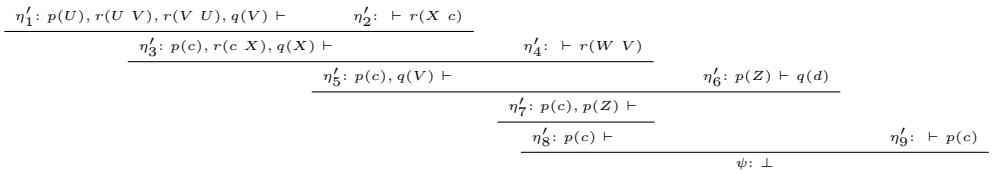
Which if we lower η_2: $\vdash p(c)$, we would attempt to resolve the following two sequents

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

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:



A method to avoid this issue is discussed in section 6

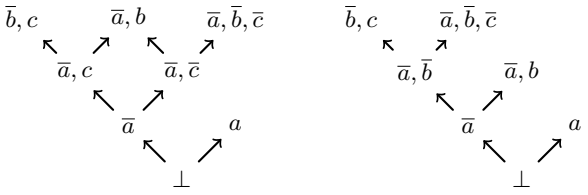
5 First-Order LowerUnits

Proposition 1. *Given a proof ψ, if there is a sequence $U = (\varphi_1 \dots \varphi_n)$ of ψ's subproofs and a sequence $(\ell_1 \dots \ell_n)$ of literals such that $\forall i \in [1 \dots n]$, ℓ_i is the univalent literal of φ_i w.r.t. $\Delta_{i-1} = \{\overline{\ell_1} \dots \overline{\ell_{i-1}}\}$, then the conclusion of*

$$\psi' = \psi \setminus \{U\} \odot_{\ell_n} \varphi_n \dots \odot_{\ell_1} \varphi_1$$

subsumes the conclusion of ψ.

Proof. The proposition is proven by induction on n, along with the fact that $\psi \setminus \{U\} \notin U$. For $n = 0$, $U = \varnothing$ and the properties trivially hold. Suppose a subproof φ_{n+1} of ψ is univalent w.r.t. Δ_n, with univalent literal ℓ_{n+1}. Because $\ell_{n+1} \notin \Delta_n$, there exists a subproof of $\psi \setminus \{U\}$ with conclusion containing $\overline{\ell_{n+1}}$, and therefore $\psi \setminus \{U\} \setminus \{\varphi_{n+1}\} \notin U \cup \{\varphi_{n+1}\}$. Let Γ be the conclusion of $\psi \setminus \{U\}$. The conclusion of $\psi' = \psi \setminus \{U \cup \{\varphi_{n+1}\}\} = \psi \setminus \{U\} \setminus \{\varphi_{n+1}\}$ is included in $\Gamma \cup \{\overline{\ell_{n+1}}\}$. The conclusion of $\psi' \odot_{\ell_{n+1}} \varphi_{n+1}$ is included in $\Gamma \cup \Delta_n$. As $\Gamma \subseteq \Gamma_\psi \cup \Delta_n$, the conclusion of $\psi' \odot_{\ell_{n+1}} \varphi_{n+1} \ldots \odot_{\ell_1} \varphi_1$ is included in Γ_ψ. □



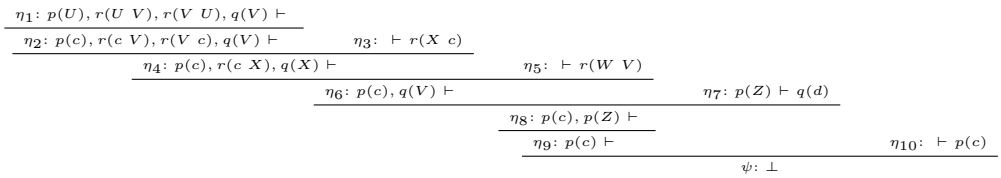
(a) Original proof (b) Compressed proof

Fig. 1: Example of proof compression by `LowerUnivalents`

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



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 ($(|\psi|_R - |\alpha(\psi)|_R)/t(\psi)$) 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

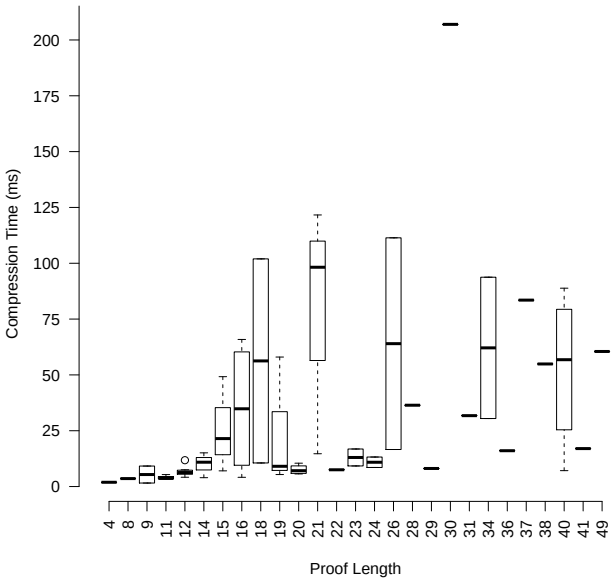
ToDo: by Bruno

`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 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`.

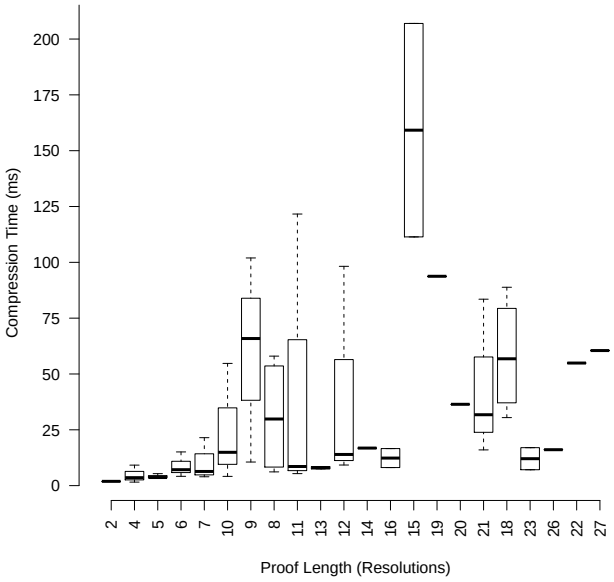
---

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

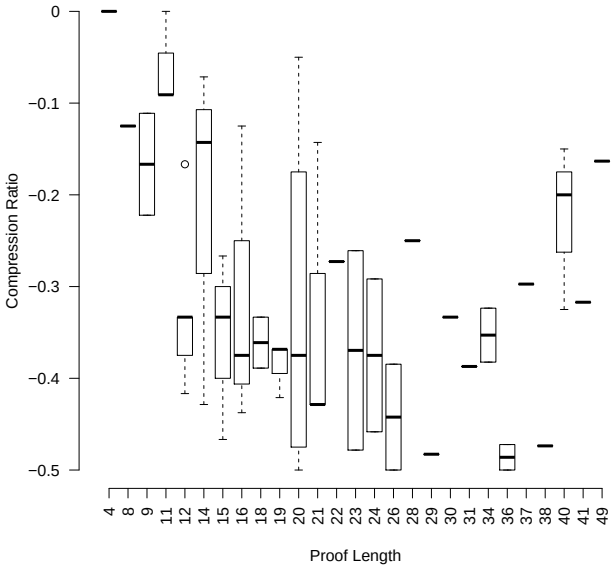
Compression Time vs. Proof Length



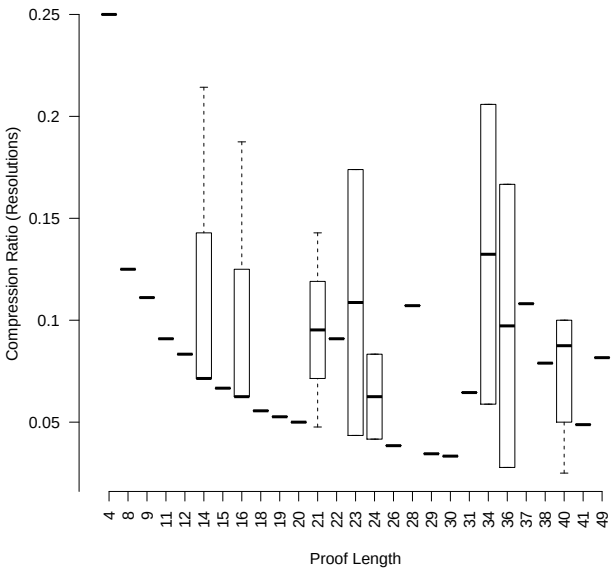
Compression Time vs. Proof Length (Resolutions)

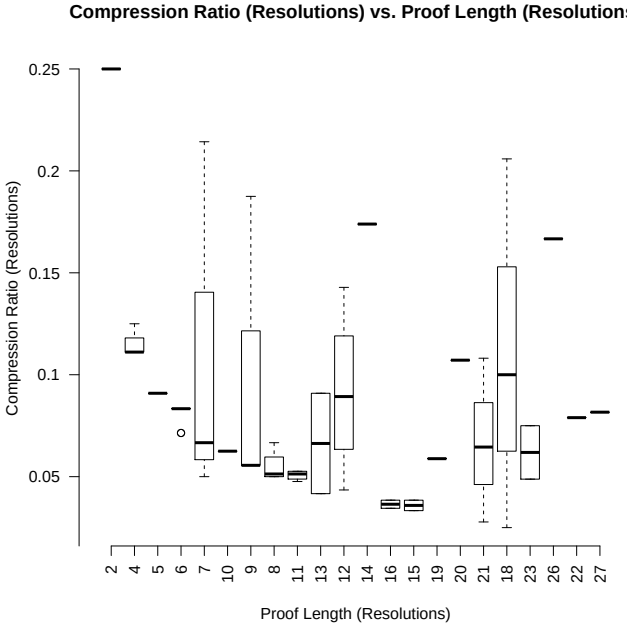


Compression Ratio vs. Proof Length



Compression Ratio (Resolutions) vs. Proof Length





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)