

Compression of Propositional Resolution Proofs by Lowering Subproofs

Joseph Boudou, Bruno Woltzenlogel Paleo

► **To cite this version:**

Joseph Boudou, Bruno Woltzenlogel Paleo. Compression of Propositional Resolution Proofs by Lowering Subproofs. 22nd International Conference TABLEAUX: Automated Reasoning with Analytic Tableaux and Related Methods - TABLEAUX 2013, Sep 2013, Nancy, France. pp. 59-73. hal-01166922

HAL Id: hal-01166922

<https://hal.archives-ouvertes.fr/hal-01166922>

Submitted on 23 Jun 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Open Archive TOULOUSE Archive Ouverte (OATAO)

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in : <http://oatao.univ-toulouse.fr/>
Eprints ID : 12885

To link to this article : DOI:10.1007/978-3-642-40537-2_7
URL : http://dx.doi.org/10.1007/978-3-642-40537-2_7

To cite this version : Boudou, Joseph and Woltzenlogel Paleo, Bruno
Compression of Propositional Resolution Proofs by Lowering Subproofs. (2013) In: 22nd International Conference Tableaux and Related Methods - TABLEAUX, 2013, 16 September 2013 - 19 September 2013 (Nancy, France).

Any correspondence concerning this service should be sent to the repository administrator: staff-oatao@listes-diff.inp-toulouse.fr

Compression of Propositional Resolution Proofs by Lowering Subproofs

Joseph Boudou¹ * and Bruno Woltzenlogel Paleo² **

¹ Université Paul Sabatier, Toulouse

`joseph.boudou@matabio.net`

² Vienna University of Technology

`bruno@logic.at`

Abstract. This paper describes a generalization of the `LowerUnits` algorithm [10] for the compression of propositional resolution proofs. The generalized algorithm, called `LowerUnivalents`, is able to lower not only units but also subproofs of non-unit clauses, provided that they satisfy some additional conditions. This new algorithm is particularly suited to be combined with the `RecyclePivotsWithIntersection` algorithm [10]. A formal proof that `LowerUnivalents` always compresses more than `LowerUnits` is shown, and both algorithms are empirically compared on thousands of proofs produced by the SMT-Solver `veriT`.

1 Introduction

Propositional resolution is among the most successful proof calculi for automated deduction in propositional logic available today. It provides the foundation for DPLL- and CDCL-based Sat/SMT-solvers [5], which perform surprisingly well in practice [12], despite the NP-completeness of propositional satisfiability [7] and the theoretical difficulty associated with NP-complete problems.

Resolution refutations can also be output by Sat/SMT-solvers with an acceptable efficiency overhead and are detailed enough to allow easy implementation of efficient proof checkers. They can, therefore, be used as certificates of correctness for the answers provided by these tools in case of unsatisfiability.

However, as the refutations found by Sat/SMT-solvers are often redundant, techniques for compressing and improving resolution proofs in a post-processing stage have flourished. Algebraic properties of the resolution operation that might be useful for compression were investigated in [9]. Compression algorithms based on rearranging and sharing chains of resolution inferences have been developed in [1] and [15]. Cotton [8] proposed an algorithm that compresses a refutation by repeatedly splitting it into a proof of a heuristically chosen literal ℓ and a proof of $\bar{\ell}$, and then resolving them to form a new refutation. The `ReduceAndReconstruct` algorithm [14] searches for locally redundant subproofs

* Supported by the Google Summer of Code 2012 program.

** Supported by the Austrian Science Fund, project P24300.

that can be rewritten into subproofs of stronger clauses and with fewer resolution steps. In [2] two linear time compression algorithms are introduced. One of them is a partial regularization algorithm called `RecyclePivots`. An enhanced version of this latter algorithm, called `RecyclePivotsWithIntersection` (RPI), is proposed in [10], along with a new linear time algorithm called `LowerUnits`. These two last algorithms are complementary and better compression can easily be achieved by sequentially composing them (i.e. executing one after the other).

In this paper, the new algorithm `LowerUnivalents`, generalizing `LowerUnits`, is described. Its achieved goals are to compress more than `LowerUnits` and to allow fast *non-sequential* combination with RPI. While in a sequential combination one algorithm is simply executed after the other, in a non-sequential combination, both algorithms are executed simultaneously when the proof is traversed. Therefore, fewer traversals are needed.

The next section introduces the propositional resolution calculus along with the notations, operations and some theoretical results used in the paper. Section 3 briefly describes the `LowerUnits` algorithm. In Sect. 4 the new algorithm `LowerUnivalents` is introduced and it is proved that it always compresses more than `LowerUnits`. Section 5 describes the non-sequential combination of `LowerUnivalents` and RPI. Lastly, experimental results are discussed in Sect. 6.

2 Propositional Resolution Calculus

A *literal* is a propositional variable or the negation of a propositional variable. The *dual* of a literal ℓ is denoted $\bar{\ell}$ (i.e. for any propositional variable p , $\bar{p} = \neg p$ and $\overline{\neg p} = p$). The set of all literals is denoted \mathcal{L} . A *clause* is a set of literals. \perp denotes the *empty clause*.

Definition 1 (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 (i.e. $E \subset V \times \mathcal{L} \times V$ and $v_1 \xrightarrow{\ell} v_2$ denotes an edge from node v_1 to node v_2 labeled by ℓ), is a *proof* of a clause Γ iff it is inductively constructible according to the following cases:

1. If Γ is a clause, $\hat{\Gamma}$ denotes some proof $\langle \{v\}, \emptyset, \Gamma \rangle$, where v is a new node.
2. If ψ_L is a proof $\langle V_L, E_L, \Gamma_L \rangle$ and ψ_R is a proof $\langle V_R, E_R, \Gamma_R \rangle$ and ℓ is a literal such that $\bar{\ell} \in \Gamma_L$ and $\ell \in \Gamma_R$, then $\psi_L \odot_{\ell} \psi_R$ denotes a proof $\langle V, E, \Gamma \rangle$ s.t.

$$\begin{aligned} V &= V_L \cup V_R \cup \{v\} \\ E &= E_L \cup E_R \cup \left\{ v \xrightarrow{\bar{\ell}} \rho(\psi_L), v \xrightarrow{\ell} \rho(\psi_R) \right\} \\ \Gamma &= (\Gamma_L \setminus \{\bar{\ell}\}) \cup (\Gamma_R \setminus \{\ell\}) \end{aligned}$$

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

If $\psi = \varphi_L \odot_{\ell} \varphi_R$, then 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.

<p>Input: a proof φ Input: D a set of subproofs Output: a proof φ' obtained by deleting the subproofs in D from φ</p> <pre> 1 if $\varphi \in D$ or $\rho(\varphi)$ has no premises then 2 return φ ; 3 else 4 let φ_L, φ_R and ℓ be such that $\varphi = \varphi_L \odot_\ell \varphi_R$; 5 let $\varphi'_L = \text{delete}(\varphi_L, D)$; 6 let $\varphi'_R = \text{delete}(\varphi_R, D)$; 7 if $\varphi'_L \in D$ then 8 return φ'_R ; 9 else if $\varphi'_R \in D$ then 10 return φ'_L ; 11 else if $\bar{\ell} \notin \Gamma_{\varphi'_L}$ then 12 return φ'_L ; 13 else if $\ell \notin \Gamma_{\varphi'_R}$ then 14 return φ'_R ; 15 else 16 return $\varphi'_L \odot_\ell \varphi'_R$; </pre>

Algorithm 1: delete

Contrary to the usual graph and proof theoretic conventions but following the actual implementation of the data structures used by `LowerUnivalents`, edges are directed from children (resolvents) to their parents (premises). V_ψ , E_ψ and Γ_ψ denote, respectively, the nodes, edges and proved clause (conclusion) of ψ .

Definition 2 (Active literals). *Given a proof ψ , the set of active literals $A_\psi(\varphi)$ of a subproof φ are the labels of edges coming into φ 's root:*

$$A_\psi(\varphi) = \{\ell \mid \exists \varsigma \in V_\psi. \varsigma \xrightarrow{\ell} \rho(\varphi)\}$$

Two operations on proofs are used in this paper: the resolution operation \odot_ℓ introduced above and the deletion of a set of subproofs from a proof, denoted $\psi \setminus (\varphi_1 \dots \varphi_n)$ where ψ is the whole proof and φ_i are the deleted subproofs. Algorithm 1 describes the deletion operation, with $\psi \setminus (\varphi_1 \dots \varphi_n)$ being the result of `delete`($\psi, \{\varphi_1, \dots, \varphi_n\}$). Both the resolution and deletion operations are considered to be left associative.

The basic idea of the deletion algorithm 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 direct subproof. The special case when both φ'_L and φ'_R belong to D is treated rather implicitly and deserves an explanation: in such a case, one might intuitively expect the result φ' to be undefined and arbitrary. Furthermore, to any child of φ , φ' ought to be seen as if it were in D , as if the deletion of φ'_L and φ'_R propagated to φ' as well. Instead of assigning some

arbitrary proof to φ' and adding it to D , the algorithm arbitrarily returns (in line 8) φ'_R (which is already in D) as the result φ' . In this way, the propagation of deletion is done automatically and implicitly. For instance, the following hold:

$$\varphi_1 \odot_\ell \varphi_2 \setminus (\varphi_1, \varphi_2) = \varphi_2 \quad (1)$$

$$\varphi_1 \odot_\ell \varphi_2 \odot_{\ell'} \varphi_3 \setminus (\varphi_1, \varphi_2) = \varphi_3 \setminus (\varphi_1, \varphi_2) \quad (2)$$

A side-effect of this clever implicit propagation of deletion is that the actual result of deletion is only meaningful if it is not in D . In the example (1), as $\varphi_1 \odot_\ell \varphi_2 \setminus (\varphi_1, \varphi_2) \in \{\varphi_1, \varphi_2\}$, the actual resulting proof is meaningless. Only the information that it is a deleted subproof is relevant, as it suffices to obtain meaningful results as shown in (2).

Proposition 1. *For any proof ψ and any sets A and B of ψ 's subproofs, either $\psi \setminus (A \cup B) \in A \cup B$ and $\psi \setminus (A) \setminus (B) \in A \cup B$, or $\psi \setminus (A \cup B) = \psi \setminus (A) \setminus (B)$.*

Definition 3 (Valent literal). *In a proof ψ , a literal ℓ is valent for the subproof φ iff $\bar{\ell}$ belongs to the conclusion of $\psi \setminus (\varphi)$ but not to the conclusion of ψ .*

Proposition 2. *In a proof ψ , every valent literal of a subproof φ is an active literal of φ .*

Proof. Lines 2, 12, 14 and 16 from Algorithm 1 can not introduce a new literal in the conclusion of the subproof being processed. Let ℓ be a valent literal of φ in ψ . Because there is only one subproof to be deleted, $\bar{\ell}$ can only be introduced when processing a subproof φ' such that $\rho(\varphi') \xrightarrow{\ell} \rho(\varphi)$. \square

Proposition 3. *Given a proof ψ and a set $D = \{\varphi_1 \dots \varphi_n\}$ of ψ 's subproofs, $\forall \ell \in \mathcal{L}$ s.t. ℓ is in the conclusion of $\psi \setminus (D)$ but not in ψ 's conclusion, then $\exists i$ s.t. $\bar{\ell}$ is a valent literal of φ_i in ψ .*

3 LowerUnits

When a subproof φ has more than one child in a proof ψ , it may be possible to *factor* all the corresponding resolutions: a new proof is constructed by removing φ from ψ and reintroducing it later. The resulting proof is smaller because φ participates in a single resolution inference in it (i.e. it has a single child), while in the original proof it participates in as many resolution inferences as the number of children it had. Such a factorization is called *lowering* of φ , because its delayed reintroduction makes φ appear at the bottom of the resulting proof.

Formally, a subproof φ in a proof ψ can be lowered if there exists a proof ψ' and a literal ℓ such that $\psi' = \psi \setminus (\varphi) \odot_\ell \varphi$ and $\Gamma_{\psi'} \subseteq \Gamma_\psi$. It has been noted in [10] that φ can always be lowered if it is a *unit*: its conclusion clause has only one literal. This led to the invention of the **LowerUnits** algorithm, which lowers every unit with more than one child, taking care to reintroduce units in an order

<p>Input: a proof ψ Output: a compressed proof ψ'</p> <pre> 1 Units $\leftarrow \emptyset$; 2 for every subproof φ in a bottom-up traversal do 3 if φ is a unit and has more than one child then 4 Enqueue φ in Units; 5 $\psi' \leftarrow \text{delete}(\psi, \text{Units})$; 6 for every unit φ in Units do 7 let $\{\ell\} = \Gamma_\varphi$; 8 if $\bar{\ell} \in \Gamma_{\psi'}$ then $\psi' \leftarrow \psi' \odot_\ell \varphi$;</pre>

Algorithm 2: LowerUnits

corresponding to the subproof relation: 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 .

A possible presentation of **LowerUnits** is shown in Algorithm 2. Units are collected during a first traversal. As this traversal is bottom-up, units are stored in a queue. The traversal could have been top-down and units stored in a stack. Units are effectively deleted during a second, top-down traversal. The last for-loop performs the reintroduction of units.

4 LowerUnivalents

LowerUnits does not lower every lowerable subproof. In particular, it does not take into account the already lowered subproofs. For instance, if a unit φ_1 proving $\{a\}$ has already been lowered, a subproof φ_2 with conclusion $\{\neg a, b\}$ may be lowered as well and reintroduced above φ_1 . The posterior reintroduction of φ_1 will resolve away $\neg a$ and guarantee that it does not occur in the resulting proof's conclusion. But care must also be taken not to lower φ_2 if $\neg a$ is a valent literal of φ_2 , otherwise a will undesirably occur in the resulting proof's conclusion.

Definition 4 (Univalent subproof). A subproof φ in a proof ψ is univalent w.r.t. a set Δ of literals iff φ has exactly one valent literal ℓ in ψ , $\ell \notin \Delta$ and $\Gamma_\varphi \subseteq \Delta \cup \{\ell\}$. ℓ is called the univalent literal of φ in ψ w.r.t. Δ .

The principle of **LowerUnivalents** is to lower all univalent subproofs. Having only one valent literal makes them behave essentially like units w.r.t. the technique of lowering. Δ is initialized to the empty set. Then the duals of the univalent literals are incrementally added to Δ . Proposition 4 ensures that the conclusion of the resulting proof subsumes the conclusion of the original one.

Proposition 4. 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. $\Delta_{i-1} = \{\bar{\ell}_1 \dots \bar{\ell}_{i-1}\}$, then the conclusion of

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

```

Input: a proof  $\psi$ 
Output: a compressed proof  $\psi'$ 

1 Univalents  $\leftarrow \emptyset$  ;
2  $\Delta \leftarrow \emptyset$  ;
3 for every subproof  $\varphi$ , in a top-down traversal do
4    $\psi' \leftarrow \text{delete}(\varphi, \text{Univalents})$  ;
5   if  $\psi'$  is univalent w.r.t.  $\Delta$  then
6     let  $\ell$  be the univalent literal ;
7     push  $\bar{\ell}$  onto  $\Delta$  ;
8     push  $\psi'$  onto Univalents ;

// At this point,  $\psi' = \psi \setminus (\text{Univalents})$ 
9 while Univalents  $\neq \emptyset$  do
10   $\varphi \leftarrow \text{pop}$  from Univalents;
11   $\ell \leftarrow \text{pop}$  from  $\Delta$  ;
12  if  $\ell \in \Gamma_{\psi'}$  then  $\psi' \leftarrow \varphi \odot_{\ell} \psi'$  ;

```

Algorithm 3: Simplified LowerUnivalents

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 $n = 0$, $U = \emptyset$ 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 $\bar{\ell}_{n+1}$, 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 $\Gamma \cup \{\bar{\ell}_{n+1}\}$. 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 $\psi' \odot_{\ell_{n+1}} \varphi_{n+1} \dots \odot_{\ell_1} \varphi_1$ is included in Γ_{ψ} . \square

For this principle to lead to proof compression, it is important to take care of the mutual inclusion of univalent subproofs. Suppose, for instance, that $\varphi_i, \varphi_j, \varphi_k \in U$, $i < j < k$, φ_j is a subproof of φ_i but not a subproof of $\psi \setminus (\varphi_i)$, and $\bar{\ell}_j \in \Gamma_{\varphi_k}$. In this case, φ_j will have one more child in

$$\psi \setminus (U) \odot_{\ell_n} \varphi_n \dots \odot_{\ell_k} \varphi_k \dots \odot_{\ell_j} \varphi_j \dots \odot_{\ell_i} \varphi_i \dots \odot_{\ell_1} \varphi_1$$

than in the original proof ψ . The additional child is created when φ_j is reintroduced. All the other children are reintroduced with the reintroduction of φ_i , because φ_j was not deleted from φ_i .

To solve this issue, **LowerUnivalents** traverses the proof in a top-down manner and simultaneously deletes already collected univalent subproofs, as sketched in Algorithm 3.

Figure 1 shows an example proof and the result of compressing it with **LowerUnivalents**. The top-down traversal starts with the leaves (axioms) and only visits a child when all its parents have already been visited. Assuming the unit with conclusion $\{a\}$ is the first visited leaf, it passes the univalent test

in line 5, is marked for lowering (line 8) and the dual of its univalent literal is pushed onto Δ (line ??). When the subproof with conclusion $\{\bar{a}, b\}$ is considered, $\Delta = \{\bar{a}\}$. As this subproof has only one valent literal $b \notin \Delta$ and $\{\bar{a}, b\} \subseteq \Delta \cup \{b\}$, it is marked for lowering as well. At this point, $\Delta = \{\bar{a}, \bar{b}\}$, **Univalents** contains the two subproofs marked for lowering and ψ' is the subproof with conclusion $\{\bar{a}, \bar{b}\}$ shown in Subfig. (b) (i.e. the result of deleting the two marked subproofs from the original proof in Subfig. (a)). No other subproof is univalent; no other subproof is marked for lowering. The final compressed proof (Subfig. (b)) is obtained by reintroducing the two univalent subproofs that had been marked (lines ?? – 12). It has one resolution less than the original. This is so because the subproof with conclusion $\{\bar{a}, b\}$ had been used (resolved) twice in the original proof, but lowering delays its use to a point where a single use is sufficient.

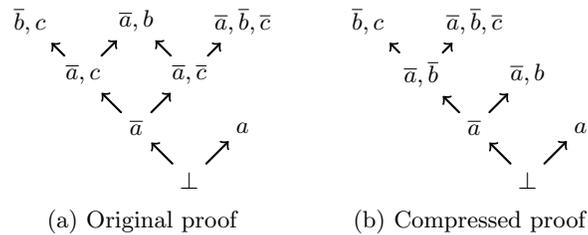


Fig. 1: Example of proof compression by **LowerUnivalents**

Although the call to **delete** inside the first loop (line 3 to 8) suggests quadratic time complexity, this loop (line 3 to 8) can be (and has been) actually implemented as a recursive function extending a recursive implementation of **delete**. With such an implementation, **LowerUnivalents** has a time complexity linear w.r.t. the size of the proof, assuming the univalent test (at line 5) is performed in constant bounded time.

Determining whether a literal is valent is expensive. But thanks to Proposition 2, subproofs with one active literal which is not in Γ_ψ can be considered instead of subproofs with one valent literal. If the active literal is not valent, the corresponding subproof will simply not be reintroduced later (i.e. the condition in line 28 of Algorithm 4 will fail).

While verifying if a subproof could be univalent, some edges might be deleted. If a subproof φ_i has already been collected as univalent subproof with univalent literal ℓ_i and the subproof φ' being considered now has ℓ_i as active literal, the corresponding incoming edges can be removed. Even if ℓ_i is valent for φ' , only $\bar{\ell}_i$ would be introduced, and it would be resolved away when reintroducing φ_i . The **delete** operation can be easily modified to remove both nodes and edges.

Algorithm 4 sums up the previous remarks for an efficient implementation of **LowerUnivalents**. As noticed above, sometimes this algorithm may consider a subproof as univalent when it is actually not. But as care is taken when reintroducing subproofs (at line 28), the resulting conclusion still subsumes the original.

The test that $\ell \in \Gamma_\varphi$ at line 20 is mandatory since ℓ might have been deleted from Γ_φ by the deletion of previously collected subproofs.

Every node in a proof $\langle V, E, \Gamma \rangle$ has exactly two outgoing edges unless it is the root of an axiom. Hence the number of axioms is $|V| - \frac{1}{2}|E|$ and because there is at least one axiom, the average number of active literals per node is strictly less than two. Therefore, if `LowerUnivalents` is implemented as an improved recursive `delete`, its time complexity remains linear, assuming membership of literals to the set Δ is computed in constant time.

Proposition 5. *Given a proof ψ , `LowerUnits` (ψ) has at least as many nodes as `LowerUnivalents` (ψ) if there are no two units in ψ with the same conclusion.*

Proof. A unit φ has exactly one active literal ℓ . Therefore φ is collected by `LowerUnivalents` unless $\bar{\ell} \in \Delta$ or $\ell \in \Delta$. If $\bar{\ell} \in \Delta$ all the incoming edges to $\rho(\varphi)$ are deleted. If $\ell \in \Delta$, every edge $v \xrightarrow{\bar{\ell}} v'$ where v is on a path from $\rho(\psi)$ to $\rho(\varphi)$ is deleted. In particular, for every edge $v \xrightarrow{\ell} \rho(\varphi)$ the edge $v \xrightarrow{\bar{\ell}} v'$ is deleted. Moreover, as ℓ is the only literal of φ 's conclusion, φ is propagated down the proof until the univalent subproof with valent literal $\bar{\ell}$ is reintroduced. \square

In the case where there are at least two units with the same conclusion in ψ , the compressed proof depends on the order in which the units are collected. For both algorithms, only one of these units appears in the compressed proof.

5 Remarks about Combining LowerUnivalents with RPI

Definition 5 (Regular proof [16]). *A proof ψ is regular iff on every path from its root to any of its axioms, each literal labels at most one edge. Otherwise, ψ is irregular.*

Any irregular proof can be converted into a regular proof having the same axioms and the same conclusion. But it has been proved [11] that such a total regularization might result in a proof exponentially bigger than the original.

Nevertheless, *partial* regularization algorithms, such as `RecyclePivots` [2] and `RecyclePivotsWithIntersection` (RPI) [10], carefully avoid the worst case of total regularization and do efficiently compress proofs. For any subproof φ of a proof ψ , RPI removes the edge $\rho(\varphi) \xrightarrow{\ell} v$ if ℓ is a safe literal for φ .

Definition 6 (Safe literal). *A literal ℓ is safe for a subproof φ in a proof ψ iff ℓ labels at least one edge on every path from $\rho(\psi)$ to $\rho(\varphi)$.*

RPI performs two traversals. During the first one, safe literals are collected and edges are marked for deletion. The second traversal is the effective deletion similar to the `delete` algorithm.

Both sequential compositions of `LowerUnits` with RPI have been shown to achieve good compression ratio [10]. However, the best combination order (`LowerUnits` after RPI (LU.RPI) or RPI after `LowerUnits` (RPI.LU)) depends on

```

Data: a proof  $\psi$ , compressed in place
Input: a set  $D_V$  of subproofs to delete
Input: a set  $D_E$  of edges to delete

1 Univalents  $\leftarrow \emptyset$  ;
2  $\Delta \leftarrow \emptyset$  ;
3 for every subproof  $\varphi$ , in a top-down traversal of  $\psi$  do
    // The deletion part.
4   if  $\varphi$  is not an axiom then
5     let  $\varphi = \varphi_L \odot_\ell \varphi_R$  ;
6     if  $\varphi_L \in D_V$  or  $\rho(\varphi) \xrightarrow{\bar{\ell}} \rho(\varphi_L) \in D_E$  then
7       if  $\rho(\varphi) \xrightarrow{\ell} \rho(\varphi_R) \in D_E$  then
8         add  $\varphi$  to  $D_V$  ;
9       else
10        replace  $\varphi$  by  $\varphi_R$  ;
11      else if  $\varphi_R \in D_V$  or  $\rho(\varphi) \xrightarrow{\bar{\ell}} \rho(\varphi_R) \in D_E$  then
12        if  $\rho(\varphi) \xrightarrow{\ell} \rho(\varphi_L) \in D_E$  then
13          add  $\varphi$  to  $D_V$  ;
14        else
15          replace  $\varphi$  by  $\varphi_L$  ;

    // Test whether  $\varphi$  is univalent.
16   ActiveLiterals  $\leftarrow \emptyset$  ;
17   for each incoming edge  $e = v \xrightarrow{\ell} \rho(\varphi)$ ,  $e \notin D_E$  do
18     if  $\bar{\ell} \in \Delta$  then
19       add  $e$  to  $D_E$  ;
20     else if  $\ell \notin \Delta$ ,  $\ell \in \Gamma_\varphi$  and  $\ell \notin \Gamma_\psi$  then
21       add  $\ell$  to ActiveLiterals;
22   if ActiveLiterals =  $\{\ell\}$  and  $\Gamma_\varphi \subseteq \Delta \cup \{\ell\}$  then
23     push  $\bar{\ell}$  onto  $\Delta$  ;
24     push  $\varphi$  onto Univalents;

    // Reintroduce lowered subproofs.
25 while Univalents  $\neq \emptyset$  do
26    $\varphi \leftarrow$  pop from Univalents;
27    $\ell \leftarrow$  pop from  $\Delta$  ;
28   if  $\ell \in \Gamma_\psi$  then
29     replace  $\psi$  by  $\varphi \odot_\ell \psi$  ;

```

Algorithm 4: Optimized LowerUnivalents as an enhanced delete

the input proof. A reasonable solution is to perform both combinations and then to choose the smallest compressed proof, but sequential composition is time consuming. To speed up DAG traversal, it is useful to topologically sort the nodes of the graph first. But in case of sequential composition this costly operation has to be done twice. Moreover, some traversals, like deletion, are identical in both algorithms and might be shared. Whereas implementing a non-sequential combination of RPI after `LowerUnits` is not difficult, a non-sequential combination of `LowerUnits` after RPI would be complicated. The difficulty is that RPI could create some new units which would be visible only after the deletion phase. A solution could be to test for units during deletion. But if units are effectively lowered during this deletion, their deletion would cause some units to become non-units. And postponing deletions of units until a second deletion traversal would prevent the sharing of this traversal and would cause one more topological sorting to be performed, because the deletion phase significantly transforms the structure of the DAG.

Apart from having an improved compression ratio, another advantage of `LowerUnivalents` over `LowerUnits` is that `LowerUnivalents` can be implemented as an enhanced `delete` operation. With such an implementation, a simple non-sequential combination of `LowerUnivalents` after RPI can be implemented just by replacing the second traversal of RPI by `LowerUnivalents`. After the first traversal of RPI, as all edges labeled by a safe literal have been marked for deletion, the remaining active literals are all valent, because for every edge $\rho(\varphi) \xrightarrow{\ell} \rho(\varphi')$, ℓ is either a safe literal of φ or a valent literal of φ' . Therefore, in the second traversal of the non-sequential combination (deletion enhanced by `LowerUnivalents`), all univalent subproofs are lowered.

6 Experiments

`LowerUnivalents` and `LUnivRPI` have been implemented in the functional programming language Scala¹ as part of the `Skeptik` library². `LowerUnivalents` has been implemented as a recursive `delete` improvement.

The algorithms have been experimented on 5 059 proofs produced by the SMT-solver `veriT`³ on unsatisfiable benchmarks from the SMT-Lib⁴. The details on the number of proofs per SMT category are shown in Table 1. The proofs were translated into pure resolution proofs by considering every non-resolution inference as an axiom.

The experiment compared the following algorithms:

- LU:** the `LowerUnits` algorithm from [10];
- LUniv:** the `LowerUnivalents` algorithm;
- RPILU:** a non-sequential combination of RPI after `LowerUnits`;

¹ <http://www.scala-lang.org/>

² <https://github.com/Paradoxika/Skeptik>

³ <http://www.verit-solver.org/>

⁴ <http://www.smtlib.org/>

Table 1: Number of proofs per benchmark category

Benchmark Category	Number of Proofs
QF_UF	3907
QF_IDL	475
QF_LIA	385
QF_UFIDL	156
QF_UFLIA	106
QF_RDL	30

- RPILUniv:** a non-sequential combination of RPI after LowerUnivalents;
LU.RPI: the sequential composition of LowerUnits after RPI;
LUnivRPI: the non-sequential combination of LowerUnivalents after RPI as described in Sect. 5;
RPI: the `RecyclePivotsWithIntersection` from [10];
Split: Cotton’s `Split` algorithm ([8]);
RedRec: the `Reduce&Reconstruct` algorithm from [14];
Best RPILU/LU.RPI: which performs both RPILU and LU.RPI and chooses the smallest resulting compressed proof;
Best RPILU/LUnivRPI: which performs RPILU and LUnivRPI and chooses the smallest resulting compressed proof.

For each of these algorithms, the time needed to compress the proof along with the number of nodes and the number of axioms (i.e. *unsat core* size) have been measured. Raw data of the experiment can be downloaded from Skeptik’s repository⁵.

The experiments were executed on the Vienna Scientific Cluster⁶ VSC-2. Each algorithm was executed in a single core and had up to 16 GB of memory available. This amount of memory has been useful to compress the biggest proofs (with more than 10^6 nodes).

The overall results of the experiments are shown in Table 2. The compression ratios in the second column are computed according to formula (3), in which ψ ranges over all the proofs in the benchmark and ψ' ranges over the corresponding compressed proofs.

$$1 - \frac{\sum |V_{\psi'}|}{\sum |V_{\psi}|} \quad (3)$$

The *unsat core* compression ratios are computed in the same way, but using the number of axioms instead of the number of nodes. The speeds on the fourth column are computed according to formula (4) in which d_{ψ} is the duration in

⁵ <https://raw.githubusercontent.com/Paradoxika/Skeptik/master/doc/papers/LUniv/all-final.csv>

⁶ <http://vsc.ac.at/>

Table 2: Total compression ratios

Algorithm	Compression	Unsat Core Compression	Speed
LU	7.5 %	0.0 %	22.4 n/ms
LUniv	8.0 %	0.8 %	20.4 n/ms
RPILU	22.0 %	3.6 %	7.4 n/ms
RPILUniv	22.1 %	3.6 %	6.5 n/ms
LU.RPI	21.7 %	3.1 %	15.1 n/ms
LUnivRPI	22.0 %	3.6 %	17.8 n/ms
RPI	17.8 %	3.1 %	31.3 n/ms
Split	21.0 %	0.8 %	2.9 n/ms
RedRec	26.4 %	0.4 %	2.9 n/ms
Best RPILU/LU.RPI	22.0 %	3.7 %	5.0 n/ms
Best RPILU/LUnivRPI	22.2 %	3.7 %	5.2 n/ms

milliseconds of ψ 's compression by a given algorithm.

$$\frac{\sum |V_\psi|}{\sum d_\psi} \quad (4)$$

For the **Split** and **RedRec** algorithms, which must be repeated, a timeout has been fixed so that the speed is about 3 nodes per millisecond.

Figure 2 shows the comparison of **LowerUnits** with **LowerUnivalents**. Subfigures (a) and (b) are scatter plots where each dot represents a single benchmark proof. Subfigure (c) is a histogram showing, in the vertical axis, the proportion of proofs having (*normalized*) *compression ratio difference* within the intervals showed in the horizontal axis. This difference is computed using formula (5) with v_{LU} and v_{LUniv} being the compression ratios obtained respectively by **LowerUnits** and **LowerUnivalents**.

$$\frac{v_{LU} - v_{LUniv}}{\frac{v_{LU} + v_{LUniv}}{2}} \quad (5)$$

The number of proofs for which $v_{LU} = v_{LUniv}$ is not displayed in the histogram. The (*normalized*) *duration differences* in subfigure (d) are computed using the same formula (5) but with v_{LU} and v_{LUniv} being the time taken to compress the proof by **LowerUnits** and **LowerUnivalents** respectively.

As expected, **LowerUnivalents** always compresses more than **LowerUnits** (subfigure (a)) at the expense of a longer computation (subfigure (d)). And even if the compression gain is low on average (as noticeable in Table 2), subfigure (a) shows that **LowerUnivalents** compresses some proofs significantly more than **LowerUnits**.

It has to be noticed that veriT already does its best to produce compact proofs. In particular, a forward subsumption algorithm is applied, which results in proofs not having two different subproofs with the same conclusion. This results in **LowerUnits** being unable to reduce unsat core. But as **LowerUnivalents**

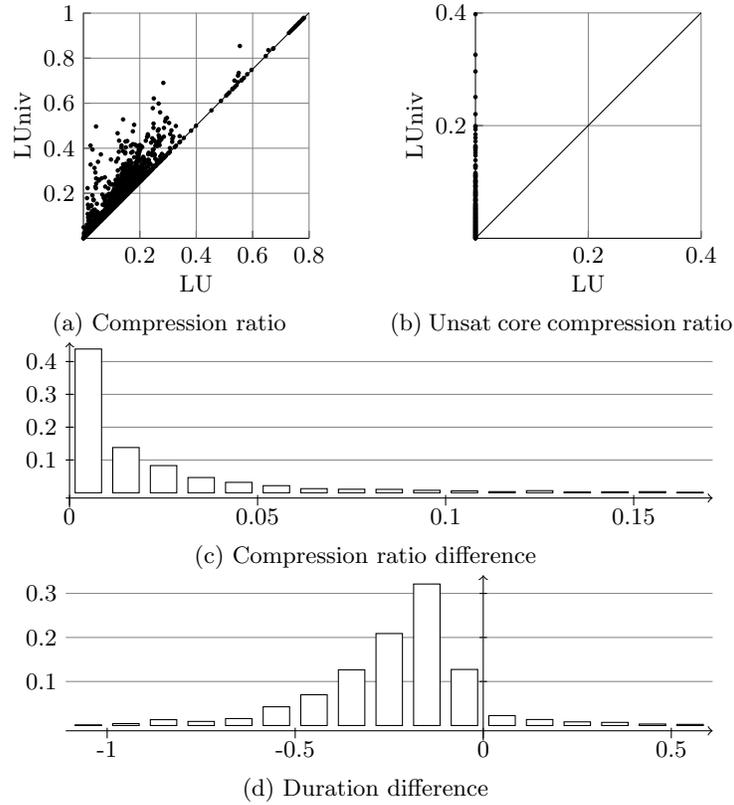


Fig. 2: Comparison between LU and LUniv

lowers non-unit subproofs and performs some partial regularization, it achieves some unsat core reduction, as noticeable in subfigure (b).

The comparison of the sequential LU.RPI with the non-sequential LUnivRPI shown in Fig. 3 outlines the ability of **LowerUnivalents** to be efficiently combined with other algorithms. Not only compression ratios are improved but LUnivRPI is faster than the sequential composition for more than 80 % of the proofs.

7 Conclusions and Future Work

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 5, 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

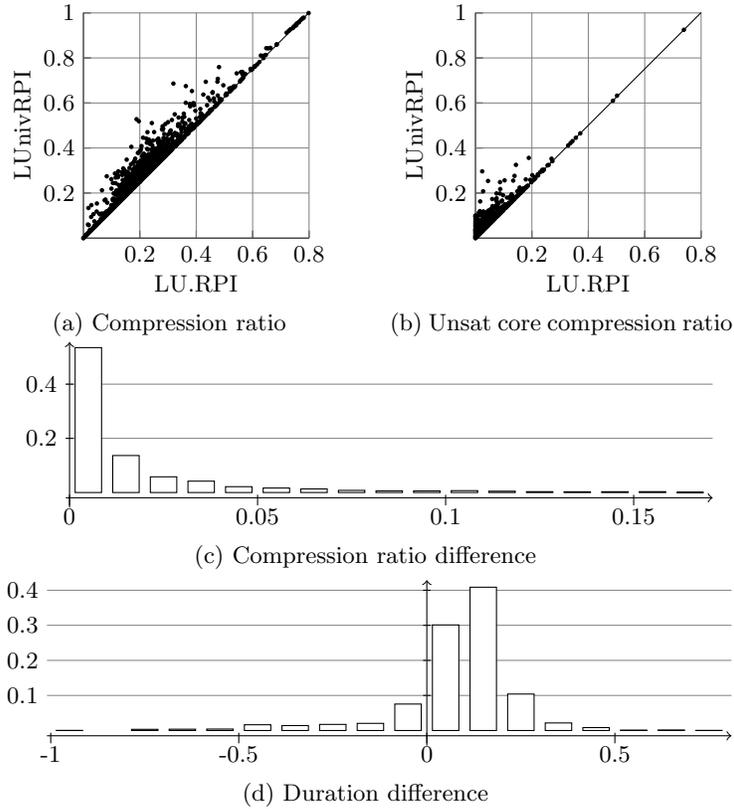


Fig. 3: Comparison between LU.RPI and LUnivRPI

promises to be challenging, though, as evidenced by the non-triviality of the optimizations discussed in Section 4 for obtaining a linear-time implementation of **LowerUnivalents**.

As discussed in Section 5, 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 3). 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.

Acknowledgments: The authors would like to thank Pascal Fontaine for providing veriT’s proofs for the experiments, for co-organizing our joint workshops on proof compression⁷, and for several interesting and useful discussions on this topic.

⁷ http://www.logic.at/people/bruno/MediaWiki/index.php/Amadeus_Vienna-Nancy_Joint_Project_on_Proof_Compression

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. Bjørner, N., Sofronie-Stokkermans, V. (eds.): *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction*, Wroclaw, Poland, July 31 - August 5, 2011. *Proceedings, Lecture Notes in Computer Science*, vol. 6803. Springer (2011)
4. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending Sledgehammer with SMT solvers. In: Bjørner and Sofronie-Stokkermans [3], pp. 116–130
5. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: *verit: An open, trustable and efficient smt-solver*. In: Schmidt, R.A. (ed.) *CADE. Lecture Notes in Computer Science*, vol. 5663, pp. 151–156. Springer (2009)
6. Brown, C.E.: *Satallax: An automatic higher-order prover*. In: Gramlich, B., Miller, D., Sattler, U. (eds.) *IJCAR. Lecture Notes in Computer Science*, vol. 7364, pp. 111–117. Springer (2012)
7. Cook, S.A.: The complexity of theorem-proving procedures. In: Harrison, M.A., Banerji, R.B., Ullman, J.D. (eds.) *STOC*. pp. 151–158. ACM (1971)
8. 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)
9. 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>
10. Fontaine, P., Merz, S., Woltzenlogel Paleo, B.: Compression of propositional resolution proofs via partial regularization. In: Bjørner and Sofronie-Stokkermans [3], pp. 237–251
11. Goerdt, A.: Comparing the complexity of regular and unrestricted resolution. In: Marburger, H. (ed.) *GWAI. Informatik-Fachberichte*, vol. 251, pp. 181–185. Springer (1990)
12. Järvisalo, M., Le Berre, D., Roussel, O., Simon, L.: The international SAT solver competitions. *AI Magazine* 33(1), 89–92 (2012)
13. Korovin, K.: *iprover - an instantiation-based theorem prover for first-order logic (system description)*. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) *IJCAR. Lecture Notes in Computer Science*, vol. 5195, pp. 292–298. Springer (2008)
14. 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)
15. 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)
16. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Siekmann, J., Wrightson, G. (eds.) *Automation of Reasoning: Classical Papers in Computational Logic 1967-1970*, vol. 2. Springer-Verlag (1983)