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

ToDo:

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 [?], which perform surprisingly well in practice [?], despite the NP-completeness of propositional satisfiability [?] 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 [?]. Compression algorithms based on rearranging and sharing chains of resolution inferences have been developed in [?] and [?]. Cotton [?] proposed an algorithm that compresses a refutation by repeteadly splitting it into a proof of a heuristically chosen literal ℓ and a proof of $\overline{\ell}$, and then resolving them to form a new refutation. The `Reduce&Reconstruct` algorithm [?] searches for locally redundant subproofs that can be rewritten into subproofs of stronger clauses and with fewer resolution steps. In [?] two

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

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 [**?**], 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 using notations that are more convenient for describing proof transformation operations. It also describes the new concepts of *active literals* and *valent literals* and proves basic but essential results about them. Section 3 briefly describes the `LowerUnits` algorithm. In Sect. **??** 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 The Resolution Calculus

We assume that there are infinitely many variable symbols (e.g. X, Y, Z, X_1, X_2, $\ldots$), constant symbols (e.g. a, b, c, a_1, a_2, $\ldots$),function symbols of every arity (e.g f, g, f_1, f_2, $\ldots$) and predicate symbols of every arity (e.g. p, q, p_1, p_2,$\ldots$). A *term* is any variable, constant or the application of an n-ary function symbol to n terms. An *atomic formula* (*atom*) is the application of an n-ary predicate symbol to n terms. A *literal* is an atom or the negation of an atom. The *complement* of a literal ℓ is denoted $\overline{\ell}$ (i.e. for any atom p, $\overline{p} = \neg p$ and $\overline{\neg p} = p$). The set of all literals is denoted $\mathcal{L}$. A *clause* is a multiset of literals. $\bot$ denotes the *empty clause*. $\mathrm{FV}(t)$ (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 $\mathcal{S}$. If 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 $\psi_L \odot_\ell \psi_R$ 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.
- ***Contraction:*** *If ψ' is a proof $\langle V', E', \Gamma' \rangle$ and σ is a unifier of $\{\ell_1, \ldots \ell_n\}$ with $\{\ell_1, \ldots \ell_n\} \subseteq \Gamma'$ and $\ell = \ell_k \sigma$ $(1 \leq k \leq n)$, then $\lfloor \psi \rfloor_\sigma^\ell$ 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. □

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. V_ψ, E_ψ and Γ_ψ denote, respectively, the nodes, edges and proved clause (conclusion) of ψ.

The deletion algorithm is a minor variant of the Reconstruct-Proof algorithm presented in [**?**]. 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 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:

$$\varphi_1 \odot_\ell \varphi_2 \setminus (\varphi_1, \varphi_2) = \varphi_2 \tag{1}$$
$$\varphi_1 \odot_\ell \varphi_2 \odot_{\ell'} \varphi_3 \setminus (\varphi_1, \varphi_2) = \varphi_3 \setminus (\varphi_1, \varphi_2) \tag{2}$$

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, φ_R *and* ℓ be such that $\varphi = \varphi_L \odot_\ell \varphi_R$;
    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 $\overline{\ell} \notin \Gamma_{\varphi'_L}$ **then**
        return φ'_L ;
    else if $\ell \notin \Gamma_{\varphi'_R}$ **then**
        return φ'_R ;
    else
        return $\varphi'_L \odot_\ell \varphi'_R$;

Algorithm 1: `delete`

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 $\psi \setminus (A \cup B) = \psi \setminus (A) \setminus (B)$.*

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 [?] 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

Input: a proof ψ
Output: a compressed proof ψ'

$\mathsf{Units} \leftarrow \varnothing$;
for *every subproof* φ *in a bottom-up traversal* **do**
  if φ *is a unit and has more than one child* **then**
    Enqueue φ in Units;
$\psi' \leftarrow \texttt{delete}(\psi, \mathsf{Units})$;
for *every unit* φ *in* Units **do**
  let $\{\ell\} = \Gamma_\varphi$;
  if $\bar{\ell} \in \Gamma_{\psi'}$ **then** $\psi' \leftarrow \psi' \odot_\ell \varphi$;

Algorithm 2: `LowerUnits`

every unit with more than one child, taking care to reintroduce units in an order 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 First-Order LowerUnits

`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 2 (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 complements of the univalent literals are incrementally added to Δ. Proposition 2 ensures that the conclusion of the resulting proof subsumes the conclusion of the original one.

Proposition 2. *Given a proof ψ, if there is a sequence $U = (\varphi_1 \ldots \varphi_n)$ of ψ's subproofs and a sequence $(\ell_1 \ldots \ell_n)$ of literals such that $\forall i \in [1 \ldots n]$, ℓ_i is the*

Input: a proof ψ
Output: a compressed proof ψ'

Univalents $\leftarrow \varnothing$;
$\Delta \leftarrow \varnothing$;
for *every subproof φ, in a top-down traversal* **do**
  $\psi' \leftarrow$ `delete`(φ,Univalents) ;
  if ψ' *is univalent w.r.t.* Δ **then**
    let ℓ be the univalent literal ;
    push $\overline{\ell}$ onto Δ ;
    push ψ' onto Univalents ;
`// At this point,` $\psi' = \psi \setminus (\text{Univalents})$
while Univalents $\neq \varnothing$ **do**
  $\varphi \leftarrow$ **pop** from Univalents;
  $\ell \leftarrow$ **pop** from Δ ;
  if $\ell \in \Gamma_{\psi'}$ **then** $\psi' \leftarrow \varphi \odot_\ell \psi'$;

Algorithm 3: Simplified `LowerUnivalents`

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 $\overline{\ell_{n+1}} \notin \Delta_n$, 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 $\psi' \odot_{\ell_{n+1}} \varphi_{n+1} \dots \odot_{\ell_1} \varphi_1$ is included in Γ_ψ. $\square$

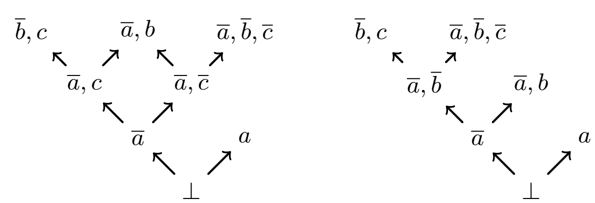
For this principle to lead to proof compression, it is important to take care of the mutual inclusion of univalent subproofs. Suppose, for instance, that $\varphi_i, \varphi_j, \varphi_k \in U$, $i < j < k$, φ_j is a subproof of φ_i but not a subproof of $\psi \setminus (\varphi_i)$, and $\overline{\ell_j} \in \Gamma_{\varphi_k}$. 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 complement of its univalent literal is pushed onto Δ (line 7). When the subproof with conclusion $\{\overline{a}, b\}$ is considered, $\Delta = \{\overline{a}\}$. As this subproof has only one valent literal $b \notin \Delta$ and $\{\overline{a}, b\} \subseteq \Delta \cup \{b\}$, it is marked for lowering as well. At this point, $\Delta = \{\overline{a}, \overline{b}\}$, `Univalents` contains the two subproofs marked for lowering and ψ' is the subproof with conclusion $\{\overline{a}, \overline{b}\}$ 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 9 – 12). It has one resolution less than the original. This is so because the subproof with conclusion $\{\overline{a}, b\}$ had been used (resolved) twice in the original proof, but lowering delays its use to a point where a single use is sufficient.



(a) Original proof (b) Compressed proof

Fig. 1: Example of proof crompression 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 **??**, 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 $\overline{\ell_i}$ 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 3. *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 $\overline{\ell} \in \Delta$ or $\ell \in \Delta$. If $\overline{\ell} \in \Delta$ all the incoming edges to $\rho(\varphi)$ are deleted. If $\ell \in \Delta$, every edge $v \xrightarrow{\overline{\ell}} v'$ 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 $v \xrightarrow{\overline{\ell}} v'$ 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 $\overline{\ell}$ 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 3 (Regular proof [?]). *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 [?] that such a total regularization might result in a proof exponentially bigger than the original.

Nevertheless, *partial* regularization algorithms, such as `RecyclePivots` [?] and `RecyclePivotsWithIntersection` (`RPI`) [?], 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 4 (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.

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

Univalents $\leftarrow \varnothing$;
$\Delta \leftarrow \varnothing$;
for *every subproof* φ*, in a top-down traversal of* ψ **do**
  `// The deletion part.`
  if φ *is not an axiom* **then**
    let $\varphi = \varphi_L \odot_\ell \varphi_R$;
    if $\varphi_L \in D_V$ *or* $\rho(\varphi) \xrightarrow{\overline{\ell}} \rho(\varphi_L) \in D_E$ **then**
      if $\rho(\varphi) \xrightarrow{\ell} \rho(\varphi_R) \in D_E$ **then**
        add φ to D_V ;
      else
        replace φ by φ_R ;
    else if $\varphi_R \in D_V$ *or* $\rho(\varphi) \xrightarrow{\overline{\ell}} \rho(\varphi_R) \in D_E$ **then**
      if $\rho(\varphi) \xrightarrow{\ell} \rho(\varphi_L) \in D_E$ **then**
        add φ to D_V ;
      else
        replace φ by φ_L ;
  `// Test whether` φ `is univalent.`
  ActiveLiterals $\leftarrow \varnothing$;
  for *each incoming edge* $e = v \xrightarrow{\ell} \rho(\varphi)$, $e \notin D_E$ **do**
    if $\overline{\ell} \in \Delta$ **then**
      add e to D_E ;
    else if $\ell \notin \Delta$, $\ell \in \Gamma_\varphi$ *and* $\ell \notin \Gamma_\psi$ **then**
      add ℓ to ActiveLiterals;
  if ActiveLiterals $= \{\ell\}$ *and* $\Gamma_\varphi \subseteq \Delta \cup \{\ell\}$ **then**
    push $\overline{\ell}$ onto Δ ;
    push φ onto Univalents;

`// Reintroduce lowered subproofs.`
while Univalents $\neq \varnothing$ **do**
  $\varphi \leftarrow$ **pop** from Univalents;
  $\ell \leftarrow$ **pop** from Δ ;
  if $\ell \in \Gamma_\psi$ **then**
    replace ψ by $\varphi \odot_\ell \psi$;

Algorithm 4: Optimized `LowerUnivalents` as an enhanced `delete`

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

ToDo by Jan

`LowerUnits` has been implemented as a prototype[1] in the functional programming language Scala[2] as part of the Skeptik library[3]. `LowerUnits` has been implemented as a recursive `delete` improvement.

The algorithm has been applied to **308** proofs produced by the SPASS[4] theorem prover on unsatisfiable benchmarks from the TPTP Problem Library[5]. The proofs used were restricted to those which could be solved within 300 seconds

[1] Source code available at https://github.com/jgorzny/Skeptik
[2] http://www.scala-lang.org/
[3] https://github.com/Paradoxika/Skeptik
[4] http://www.verit-solver.org/
[5] http://www.cs.miami.edu/ tptp/

by SPASS on the Euler Cluster at the University of Victoria[6] 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[7], 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.

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 3, 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`.

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 **??**). 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.

---

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

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