Journal Title 0 (2017) 1–0
IOS Press

Partial Regularization of First-Order Resolution Proofs

Jan Gorzny [a,*], and Bruno Woltzenlogel Paleo [b]

[a] *School of Computer Science, University of Waterloo, 200 University Ave. W., Waterloo, ON N2L 3G1, Canada*
E-mail: jgorzny@uwaterloo.ca

[b] *College of Engineering and Computer Science, Australian National University, Canberra ACT 0200, Australia*
Vienna University of Technology, Karlsplatz 13, 1040, Vienna, Austria
E-mail: bruno@logic.at

Abstract. This paper describes the generalization of the proof compression algorithm `RecyclePivotsWithIntersection` from propositional to first-order logic. The generalized algorithm performs partial regularization of resolution proofs containing resolution and factoring inferences with *unification*, as generated by many automated theorem provers. An empirical evaluation of the generalized algorithm and its combinations with `GreedyLinearFirstOrderLowerUnits` is also presented.

Keywords: proof compression, first-order logic, resolution, unification

1. Introduction

First-order automated theorem provers, commonly based on resolution and superposition calculi, have recently achieved a high degree of maturity. Proof production is a key feature that has been gaining importance, since proofs are crucial for applications that require certification of a prover's answers or that extract extra information from proofs (e.g. unsat cores, interpolants, instances of quantified variables). Nevertheless, proof production is non-trivial [12], and the best, most efficient provers do not necessarily generate the best, least redundant proofs.

For propositional resolution proofs, as those typically generated by SAT- and SMT-solvers, there is a wide variety of proof compression techniques. Algebraic properties of the resolution operation that are potentially useful for compression were investigated in [6]. Compression algorithms based on rearranging and sharing chains of resolution inferences have been developed in [2] and [13]. Cotton [5] proposed an algorithm that compresses a refutation by repeatedly 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 [11] 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 [3] and improved in [7].

In contrast, there has been much less work on simplifying first-order proofs. For tree-like sequent calculus proofs, algorithms based on cut-introduction [9, 10] have been proposed. However, converting a DAG-like resolution or superposition proof, as usually generated by current provers, into a tree-like sequent calculus proof may increase the size of the proof. For arbitrary proofs in the TPTP [14] format (including DAG-like first-order resolution proofs), there is a simple algorithm [16] that looks for terms that occur often in any TSTP [14] proof and abbreviates them.

The work reported in this paper is part of a new trend that aims at lifting successful propositional proof compression algorithms to first-order logic. Our first target was the propositional `LowerUnits` (`LU`) algorithm, which delays resolution steps with unit clauses, resulting in the

*Corresponding author. E-mail: jgorzny@uwaterloo.ca.

`GreedyLinearFirstOrderLowerUnits` (`GFOLU`) algorithm [8]. Here we continue this line of research by lifting the `RecyclePivotsWithIntersection` (`RPI`) algorithm [7], which improves the `RecyclePivots` (`RP`) algorithm [3] by detecting nodes that can be regularized even when they have several children.

Section 2 introduces the well-known first-order resolution calculus with notations that are suitable for manipulating proofs as first-class objects. Section 4 discusses the challenges that arise in the first-order case (mainly due to unification), which are not present the propositional case. Section 5 describes an algorithm that overcomes these challenges. Section 6 presents experimental results obtained by applying this algorithm, and its combinations with `GFOLU`, on hundreds of proofs generated with the SPASS theorem prover on TPTP benchmarks [14] and other randomly generated proofs. Section 7 concludes the paper.

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 *underlying atom* of a literal ℓ is denoted $|\ell|$ (i.e. for any atom p, $|p| = p$ and $|\neg p| = p$). The set of all literals is denoted $\mathcal{L}$. A *clause* is a multiset of literals. $\perp$ denotes the *empty clause*. A *unit clause* is a clause with a single literal. Sequent notation is used for clauses (i.e. $p_1, \ldots, p_n \vdash q_1, \ldots, q_m$ denotes the clause $\{\neg p_1, \ldots, \neg p_n, q_1, \ldots, q_m\}$). $\mathrm{FV}(t)$ (resp. $\mathrm{FV}(\ell)$, $\mathrm{FV}(\Gamma)$) denotes the set of variables in the term t (resp. in the literal ℓ and in the clause Γ). A *substitution* $\{X_1 \backslash t_1, X_2 \backslash t_2, \ldots\}$ is a mapping from variables $\{X_1, X_2, \ldots\}$ to, respectively, terms $\{t_1, t_2, \ldots\}$. The application of a substitution σ to a term t, a literal ℓ or a clause Γ results in, respectively, the term $t\sigma$, the literal $\ell\sigma$ or the clause $\Gamma\sigma$, obtained from t, ℓ and Γ by replacing all occurrences of the variables in σ by the corresponding terms in σ. 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 2.1). A *resolution refutation* is a resolution proof with root $\perp$.

Definition 2.1 (First-Order Resolution Proof)**.**
A directed acyclic graph $\langle V, E, \Gamma \rangle$, where V is a set of nodes and E is a set of edges labeled by literals and substitutions (i.e. $E \subset V \times 2^{\mathcal{L}} \times \mathcal{S} \times V$ and $v_1 \xrightarrow[\sigma]{\ell} v_2$ denotes an edge from node v_1 to node v_2 labeled by the literal ℓ and the substitution σ), is a proof of a clause Γ iff it is inductively constructible according to the following cases:

- **Axiom:** If Γ is a clause, $\widehat{\Gamma}$ denotes some proof $\langle \{v\}, \varnothing, \Gamma \rangle$, where v is a new (axiom) node.
- **Resolution:** If ψ_L is a proof $\langle V_L, E_L, \Gamma_L \rangle$ with $\ell_L \in \Gamma_L$ and ψ_R is a proof $\langle V_R, E_R, \Gamma_R \rangle$ with $\ell_R \in \Gamma_R$, and σ_L and σ_R are substitutions such that $\ell_L\sigma_L = \overline{\ell_R}\sigma_R$ and $\mathrm{FV}((\Gamma_L \setminus \{\ell_L\})\,\sigma_L) \cap \mathrm{FV}((\Gamma_R \setminus \{\ell_R\})\,\sigma_R) = \emptyset$, then $\psi_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \psi_R$ denotes a proof $\langle V, E, \Gamma \rangle$ s.t.

$$V = V_L \cup V_R \cup \{v\}$$

$$E = E_L \cup E_R \cup \left\{ \rho(\psi_L) \xrightarrow[\sigma_L]{\{\ell_L\}} v, \rho(\psi_R) \xrightarrow[\sigma_R]{\{\ell_R\}} v \right\}$$

$$\Gamma = (\Gamma_L \setminus \{\ell_L\})\,\sigma_L \cup (\Gamma_R \setminus \{\ell_R\})\,\sigma_R$$

  where v is a new (resolution) node and $\rho(\varphi)$ denotes the root node of φ. ℓ_L and ℓ_R are v's *resolved literals*, whereas $\ell_L\sigma_L$ and $\ell_R\sigma_R$ are its *instantiated resolved literals*. The *pivot* of v is the underlying atom of its instantiated resolved literals (i.e. $|\ell_L\sigma_L|$ or, equivalently, $|\ell_R\sigma_R|$).
- **Contraction:** If ψ' is a proof $\langle V', E', \Gamma' \rangle$ and σ is a unifier of $\{\ell_1, \ldots \ell_n\}$ with $\{\ell_1, \ldots \ell_n\} \subseteq \Gamma'$, then $\lfloor \psi \rfloor_{\{\ell_1, \ldots \ell_n\}}^{\sigma}$ denotes a proof $\langle V, E, \Gamma \rangle$ s.t.

$$V = V' \cup \{v\}$$

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

$$\Gamma = (\Gamma' \setminus \{\ell_1, \ldots \ell_n\})\sigma \cup \{\ell\}$$

  where v is a new (contraction) node, $\ell = \ell_k\sigma$ (for any $k \in \{1, \ldots, n\}$) and $\rho(\varphi)$ denotes the root node of φ. □

Example 2.1. *An example first-order resolution proof is shown below.*

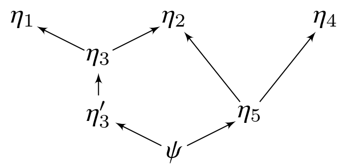


Fig. 1. The proof in Example 2.1.

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

The nodes η_1, η_2, and η_4 are axioms. Node η_3 is obtained by resolution on η_1 and η_2 where $\ell_L = p(b)$, $\ell_R = \neg p(b)$, and $\sigma_L = \sigma_R = \emptyset$. The node η_3' is obtained by a contraction on η_3 with $\sigma = \{X \setminus a\}$. The node η_5 is the result of resolution on η_2 and η_4 with $\ell_L = \neg p(b)$, $\ell_R = p(b)$, $\sigma_L = \sigma_R = \emptyset$. Lastly, the conclusion node ψ is the result of a resolution of η_3' and η_5, where $\ell_L = \neg q(a)$, $\ell_R = q(Y)$, $\sigma_L = \emptyset$, and $\sigma_R = \{Y \setminus a\}$. The directed acyclic graph representation of the proof (with edge labels omitted) is shown in Figure 1.

3. The Propositional Algorithm

`RPI` (formally defined in Appendix A) removes *irregularities*, which are resolution inferences with a node η when the resolved literal occurs as the pivot of another inference located below in the path from η to the root of the proof. In the worst case, regular resolution proofs can be exponentially bigger than irregular ones, but `RPI` takes care of regularizing the proof only partially, removing inferences only when this does not enlarge the proof.

`RPI` traverses the proof twice. On the first traversal (bottom-up), it stores for each node a set of *safe literals* that are resolved in all paths below it in the proof or that occur in the root clause of the proof. If one of the node's resolved literals belongs to the set of safe literals, then it is possible to *regularize* the node by replacing it by the parent containing the safe literal. To do this replacement efficiently, the replacement is postponed by marking the other parent as a `deletedNode`. Then, on a single second traversal (top-down), regularization is performed: any node that has a parent node marked as a `deletedNode` is replaced by its other parent.

The `RPI` and the `RP` algorithms differ from each other mainly in the computation of the safe literals of a node that has many children. While the former returns the intersection as shown in Algorithm 6, the latter returns the empty set. Moreover, while in `RPI` the safe literals of the root node contain all the literals of the root clause, in `RP` the root node is always assigned an empty set of literals.

4. First-Order Challenges

In this section, we describe challenges that have to be overcome in order to successfully adapt `RPI` to the first-order case. The first example illustrates the need to take unification into account. The other two examples discuss complex issues that can arise when unification is taken into account in a naive way.

Example 4.1. *Consider the following proof ψ. When computed as in the propositional case, the safe literals for η_3 are $\{q(c),\ p(a,X)\}$.*

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

As neither of η_3's pivots is syntactically equal to a safe literal, the propositional `RPI` algorithm would not change ψ. However, η_3's left pivot $p(W,X) \in \eta_1$ is unifiable with the safe literal $p(a,X)$. Regularizing η_3, by deleting the edge between η_2 and η_3 and replacing η_3 by η_1, leads to further deletion of η_4 (because it is not resolvable with η_1) and finally to the much shorter proof below.

$$\dfrac{\eta_1\colon \vdash p(W,X) \qquad \eta_6\colon p(Y,b) \vdash}{\psi'\colon \bot}$$

Unlike in the propositional case, where a pivot must be syntactically equal to a safe literal for regularization to be possible, the example above suggests that, in the first-order case, it might suffice that a pivot be unifiable with a safe literal. However, there are cases, as shown in the example below, where mere unifiability is not enough and greater care is needed.

Example 4.2. *Again, the safe literals for η_3, when computed as in the propositional case, are $\{q(c),\ p(a,X)\}$, and as the pivot $p(a,c)$ is unifiable with the safe literal $p(a,X)$, η_3 appears to be a candidate for regularization.*

$$\dfrac{\eta_6: p(Y,b) \vdash \qquad \dfrac{\dfrac{\eta_1: \vdash p(a,c) \qquad \eta_2: p(a,c) \vdash q(c)}{\eta_3: \vdash q(c)} \qquad \eta_4: q(c) \vdash p(a,X)}{\eta_5: \vdash p(a,X)}}{\psi: \bot}$$

However, if we attempt to regularize the proof, the same series of actions as in Example 4.1 would require resolution between η_1 and η_6, which is not possible.

One way to prevent the problem depicted above would be to require the pivot to be not only unifiable but in fact more general than a safe literal. A weaker (and better) requirement is possible, and requires a slight modification to the concept of safe literals.

Definition 4.1. (First-order) safe literals for a node η, denoted $\mathcal{S}(\eta)$, are a set of instantiated resolved literals used as pivots in all paths below η in the proof, or that occur in the root clause of the proof.

First-order safe literals are obtained by applying the unifier of the resolution step to the each pivot before adding it to the set of safe literals (cf. algorithm 3, lines 8 and 10). First-order safe literals will be referred to simply as safe literals when it is clear that we are dealing with first-order proofs.

In the case of Example 4.2, computing safe literals as defined above would result in η_3 having the safe literals $\{q(c),\ p(a,b)\}$, where clearly the pivot $p(a,c)$ in η_1 is not safe. This requirement is formalized below.

Definition 4.2. Let η be a node with safe literals $\mathcal{S}(\eta)$ and parents η_1 and η_2, assuming without loss of generality, $\eta_1 \xrightarrow[\sigma_1]{\{\ell_1\}} \eta$ such that ℓ_1 is unifiable with a safe literal $\ell^* \in \mathcal{S}(\eta)$. Let $\mathcal{R}(\eta)$ be the set of all resolved literals ℓ_2 such that $\eta_1 \xrightarrow[\sigma_1']{\{\ell_1\}} \eta'$, $\eta_2' \xrightarrow[\sigma_2']{\{\ell_2\}} \eta'$, and $\ell_1\sigma_1' = \overline{\ell_2}\sigma_2'$, for some nodes η_2' and η' and unifiers σ_1' and σ_2'. The node η is said to be *pre-regularizable* in the proof ψ if all elements in $\mathcal{R}(\eta) \cup \{\overline{\ell_1}\}$ are unifiable.

This property states that a node is pre-regularizable if, for a safe resolved literal ℓ' which is resolved against literals $\ell_1, \ldots, \ell_n$ in a proof ψ, $\ell_1, \ldots, \ell_n$, and $\overline{\ell'}$ are unifiable.

Example 4.3. *Satisfying the pre-regularizability is not sufficient. Consider the proof ψ in Figure 2. After collecting the safe literals, η_3's safe literals are $\{\neg q(R,S), \neg p(c,d), q(f(a,e),c)\}$. η_3's pivot $q(f(a,V),U)$ is unifiable to (and even more general than) the safe literal $q(f(a,e),c)$. Attempting to regularize η_3 would lead to the removal of η_2, the replacement of η_3 by η_1 and the removal of η_4 (because η_1 does not contain the pivot required by η_5), with η_5 also being replaced by η_1. Then resolution between η_1 and η_6 results in η_7', which cannot be resolved with η_8, as shown below.*

$$\dfrac{\eta_8: q(f(a,e),c) \vdash \qquad \dfrac{\eta_6: \vdash p(c,d) \qquad \eta_1: p(U,V) \vdash q(f(a,V),U)}{\eta_7': \vdash q(f(a,d),c)}}{\psi': ??}$$

η_1's literal $q(f(a,V),U)$, which would be resolved with η_8's literal, was changed to $q(f(a,d),c)$ due to the resolution between η_1 and η_6.

Thus we additionally require that the following property be satisfied.

Definition 4.3. Let η be pre-regularizable, with safe literals $\mathcal{S}(\eta)$ and parents η_1 and η_2, assuming without loss of generality that $\eta_2 \xrightarrow[\sigma_2]{\{\ell_2\}} \eta$ and $\overline{\ell_2}$ is unifiable with some $\ell^* \in \mathcal{S}(\eta)$. The node η is said to be *strongly regularizable* in ψ if there exists a substitution σ such that $\eta_1\sigma \subseteq \mathcal{S}(\eta)$.

This property ensures that the remainder of the proof does not expect a variable in η_1 to be unified to different values simultaneously. This property is not necessary in the propositional case, as the replacement node would not change lower in the proof.

An alternative to the strong regularizability may also be useful for proof compression in some cases. This alternative relies on knowledge of how literals are changed after the deletion of a node in a proof (similar to the *post-deletion unifiability property* observed for `FirstOrderLowerUnits` in [8]).

Definition 4.4. Let η be a pre-regularizable node with parents η_1 and η_2, assuming without loss of generality that $\eta_1 \xrightarrow[\sigma_1]{\{\ell_1\}} \eta$ such that ℓ_1 is unifiable with some $\ell^* \in \mathcal{S}(\eta)$. For each safe literal $\ell = \ell_s\sigma_s \in \mathcal{S}(\eta_1)$, let η_ℓ be a node on the path from η to the root of the proof such that $|\ell|$ is the pivot of η_ℓ. Let $\mathcal{R}(\eta_\ell)$ be the set of all resolved literals ℓ_s' such that $\eta_2' \xrightarrow[\sigma_s]{\{\ell_s\}} \eta_\ell$, $\eta_1' \xrightarrow[\sigma_s']{\{\ell_s'\}} \eta_\ell$, and $\ell_s\sigma_s = \overline{\ell_s'}\sigma_s'$, for some nodes η_2' and η_1' and unifier σ_s'; if no such node η_ℓ exists, define $\mathcal{R}(\eta_\ell) = \emptyset$. The node η is said to be *weakly regularizable* in ψ if, for all $\ell \in \mathcal{S}(\eta_1)$, all elements in $\mathcal{R}^\dagger(\eta_\ell) \cup \{\overline{\ell}^\dagger\}$ are unifiable, where $\overline{\ell}^\dagger$ is the literal in $\psi \setminus \{\eta_2\}$ corresponding to $\overline{\ell}$ in ψ and $\mathcal{R}^\dagger(\eta_\ell)$ is the set of literals in $\psi \setminus \{\eta_2\}$ corresponding to literals of $\mathcal{R}(\eta_\ell)$ in ψ.

$$
\dfrac{\eta_8\colon q(f(a,e),c) \vdash \qquad \dfrac{\eta_6\colon \vdash p(c,d) \qquad \dfrac{\dfrac{\eta_1\colon p(U,V) \vdash q(f(a,V),U) \qquad \eta_2\colon q(f(a,X),Y), q(T,X) \vdash q(f(a,Z),Y)}{\eta_3\colon p(U,V), q(T,V) \vdash q(f(a,Z),U)} \qquad \eta_4\colon \vdash q(R,S)}{\eta_5\colon p(U,V) \vdash q(f(a,Z),U)}}{\eta_7\colon \vdash q(f(a,Z),c)}}{\psi\colon \bot}
$$

Fig. 2. An example where pre-regularizability is not sufficient.

Informally, a node η is weakly regularizable in a proof if it can be replaced by one of its parents η_1, such that $|\ell|$ for each $\ell \in \mathcal{S}(\eta_1)$ can still be used as pivots in order to complete the proof. Weakly regularizable nodes differ from strongly regularizable nodes by not requiring the entire parent η_1 replacing the resolution η to be simultaneously matched to a subset of $\mathcal{S}(\eta)$, and requires knowledge of how literals will be instantiated after the removal of η_2 and η from the proof.

Note that there is always at least one node η_ℓ as used in the definition for any safe literal which was not contained in the root clause of the proof: the node which resulted in ℓ being a safe literal for the path from η to the root of the proof. Furthermore, it does not matter which node η_ℓ is used. To see this, consider some $\eta'_\ell \neq \eta_\ell$ for some ℓ with the same pivot. Consider arbitrary nodes η_1 and η_2 such that $\eta_2 \xrightarrow{\{\ell\}} \eta_\ell$ and $\eta_1 \xrightarrow[\sigma_1]{\{\ell_1\}} \eta_\ell$ where $\ell = \overline{\ell_1}\sigma_1$. Now consider arbitrary nodes η'_1 and η'_2 such that $\eta'_2 \xrightarrow{\{\ell\}} \eta'_\ell$ and $\eta'_1 \xrightarrow[\sigma'_1]{\{\ell'_1\}} \eta'_\ell$ where $\ell = \overline{\ell'_1}\sigma'_1$. Since the pivots for η_ℓ and η'_ℓ are equal, we must have that $|\ell| = |\ell_1\sigma_1|$ and $|\ell| = |\ell'_1\sigma'_1|$, and thus $|\ell_1\sigma_1| = |\ell'_1\sigma'_1|$. This shows that it does not matter which η_ℓ we use; the instantiated resolved literals will always be equal implying that both of the resolved literals ℓ_1 and ℓ'_1 will be contained in both $\mathcal{R}(\eta_\ell)$ and $\mathcal{R}(\eta'_\ell)$.

Example 4.4. *This example illustrates the usefulness of weak regularizability as an alternative to the strong regularizability. In the proof below, note that η_6 is not satisfy the strongly regularizable: there is no unifier σ such that $\eta_6\sigma \subseteq \mathcal{S}(\eta_6)$ (where $\mathcal{S}(\eta_6) = \{\neg p(W), \neg r(Z), \neg q(Y)\}$).*

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

We show that η_6 is weakly regularizable, and that η_7 can be removed. To do this, first observe that η_6 is pre-regularizable, since $\neg p(X)$ is unifiable with $\neg p(W) \in \mathcal{S}(\eta_6)$ and $\mathcal{R}(\eta_6) \cup \{\overline{\neg p(W)}\}$ is unifiable (where $\mathcal{R}(\eta_6) = \{p(U), p(Y)\}$). Consider the following proof of $\psi \setminus \{\eta_7\}$:

$$
\dfrac{\eta_1\colon \vdash p(U) \qquad \dfrac{\dfrac{\dfrac{\eta_8\colon p(X), q(X), r(X) \vdash \qquad \eta_5\colon p(Z) \vdash q(Z)}{\eta'_4\colon p(Z), p(Z), r(Z) \vdash}}{\eta_4\colon p(Z), r(Z) \vdash} \qquad \eta_3\colon \vdash r(W)}{\eta_2\colon p(W) \vdash}}{\psi\colon \bot}
$$

Now observe that for each $\ell \in \mathcal{S}(\eta_8) = \{\neg q(Y), \neg r(Z), \neg p(W)\}$ we have the following, so that η_6 is weakly regularizable:

- $\ell = \neg q(Y)$*:* $\ell^\dagger = \neg q(X)$ *which is unifiable with* $\overline{\ell}^\dagger = q(Z)$
- $\ell = \neg r(Z)$*:* $\ell^\dagger = \neg r(Z)$ *which is unifiable with* $\overline{\ell}^\dagger = r(W)$
- $\ell = \neg p(W)$*:* $\ell^\dagger = \neg p(W)$ *which is unifiable with* $\overline{\ell}^\dagger = p(U)$

Weakly regularizable is implied by the previous two properties, as shown in the next theorem.

Theorem 4.5. *Let η be a node that is pre-regularizable in some proof. Then η is weakly regularizable.*

Proof. Let η be a pre-regularizable node with parents η_1 and η_2. Let $\mathcal{R}(\eta_\ell)$ and $\mathcal{R}^\dagger(\eta_\ell)$ be defined as in Definition 4.4 for a safe literal literal $\ell \in \mathcal{S}(\eta_1)$.

Let $\ell \in \mathcal{S}(\eta_1)$ be a safe literal of η_1 that is contained in the root clause of the proof such that there does not exists a node η_ℓ whose pivot is $|\ell|$. Then $\mathcal{R}^\dagger(\eta_\ell) \cup \{\overline{\ell}\} = \emptyset \cup \{\overline{\ell}\} = \{\overline{\ell}\}$ is trivially unifiable. Thus we may assume that for all $\ell \in \mathcal{S}(\eta_1)$, such a node η_ℓ exists.

If $\ell \notin \eta_1$, then $\ell^\dagger = \ell$ and $\overline{\ell}^\dagger = \overline{\ell}$, there is nothing to prove (neither ℓ or $\overline{\ell}$ have changed in $\psi \setminus \{\eta_2\}$). So we may assume $\ell \in \eta_1$.

Consider $\overline{\ell} \in \mathcal{R}(\eta_\ell)$: ℓ and $\overline{\ell}$ are unifiable in ψ by definition of $\mathcal{R}(\eta_\ell)$. We will show that $\ell^\dagger$ and $\overline{\ell}^\dagger$ are unifiable in $\psi \setminus \{\eta_2\}$, where $\overline{\ell}^\dagger \in \mathcal{R}^\dagger(\eta_\ell)$.

Since η_ℓ exists (with $|\ell|$ as a pivot by definition), there exists nodes η_L and η_R such that $\eta_L \xrightarrow[\sigma_L]{\{\ell_L\}} \eta_\ell$ and

input : A first-order proof ψ
output: A possibly less-irregular first-order proof ψ'

$\psi' \leftarrow \psi$;
traverse ψ' bottom-up and **foreach** *node* η *in* ψ' **do**
  if η *is a resolvent node* **then**
    setSafeLiterals(η) ;
    regularizeIfPossible(η)
$\psi' \leftarrow$ fix(ψ') ;
return ψ';

Algorithm 1: `FORPI`

$\eta_R \xrightarrow[\sigma_R]{\{\ell_R\}} \eta_\ell$ for some ℓ_L, ℓ_R, σ_L, and σ_R. Since $|\ell|$ was the pivot, we have that $|\ell| = |\ell_L\sigma_L|$ or $|\ell| = |\ell_R\sigma_R|$. Without loss of generality, assume that $|\ell| = |\ell_L\sigma_L|$. Thus we can write

$$\ell = \ell_L\sigma_L = \overline{\ell_R}\sigma_R$$

Note that $\overline{\ell}^\dagger = \overline{\ell_R}$ as $\overline{\ell_R}$ is unchanged in $\psi \setminus \{\eta_2\}$. Since η_1 replaces η in $\psi \setminus \{\eta_2\}$, we have that $\ell^\dagger = \ell$.

□

5. First-Order RecyclePivotsWithIntersection

This section presents `FirstOrderRecyclePivotsWithIntersection` (`FORPI`), Algorithm 1, a first-order generalization of `RPI`. `FORPI` traverses the proof in a bottom-up manner, storing for every node a set of safe literals. The set of safe literals for a node ψ is computed from the set of safe literals of its children (cf. Algorithm 3), similarly to the propositional case, but additionally applying unifiers to the resolved pivots (cf. Example 4.2). If one of the node's resolved literals can be unified to a literal in the set of safe literals, then it may be possible to regularize the node by replacing it by one of its parents.

In the first-order case, we additionally check for strongly regularizability (cf. lines 2 and 6 of Algorithm 2). Similarly to `RPI`, instead of replacing the irregular node by one of its parents immediately, its other parent is marked as a `deletedNode`, as shown in Algorithm 2. As in the propositional case, fixing of the proof is postponed to another (single) traversal, as regularization proceeds top-down and only nodes below a regularized node may require fixing. During fixing, the irregular node is actually replaced by the parent that is not marked as `deletedNode`. During proof fixing, factoring inferences can be applied, in order to compress the proof further.

input : A node $\psi = \psi_L \odot^{\sigma_L\sigma_R}_{\ell_L\ell_R} \psi_R$
output: nothing (but the proof containing ψ may be changed)

1 **if** $\exists\sigma$ *and* $\ell \in \psi$.safeLiterals *such that* $\ell\sigma = \ell_R$ *or* $\ell = \ell_R\sigma$ **then**
2   **if** $\exists\sigma'$ *such that* $\psi_R\sigma' \subseteq \psi$.safeLiterals **then**
3   mark ψ_L as `deletedNode` ;
4   mark ψ as regularized
5 **else if** $\exists\sigma$ *and* $\ell \in \psi$.safeLiterals *such that* $\ell\sigma = \ell_L$ *or* $\ell = \ell_L\sigma$ **then**
6   **if** $\exists\sigma'$ *such that* $\psi_L\sigma' \subseteq \psi$.safeLiterals **then**
7   mark ψ_R as `deletedNode` ;
8   mark ψ as regularized

Algorithm 2: `FOregularizeIfPossible`

input : A first-order resolution node ψ
output: nothing (but the node ψ gets a set of safe literals)

if ψ *is a root node with no children* **then**
  ψ.safeLiterals $\leftarrow \psi$.clause
else
  foreach $\psi' \in \psi$.children **do**
    if ψ' *is marked as regularized* **then**
      safeLiteralsFrom(ψ') $\leftarrow \psi'$.safeLiterals ;
    else if $\psi' = \psi \odot^{\sigma_L\sigma_R}_{\ell_L\ell_R} \psi_R$ *for some* ψ_R **then**
      safeLiteralsFrom(ψ') $\leftarrow \psi'$.safeLiterals $\cup \{\ell_R\sigma_R\}$
    else if $\psi' = \psi_L \odot^{\sigma_L\sigma_R}_{\ell_L\ell_R} \psi$ *for some* ψ_L **then**
      safeLiteralsFrom(ψ') $\leftarrow \psi'$.safeLiterals $\cup \{\ell_L\sigma_L\}$
  ψ.safeLiterals $\leftarrow \bigcap_{\psi' \in \psi.\text{children}}$ safeLiteralsFrom(ψ')

Algorithm 3: `FOsetSafeLiterals`

6. Experiments

A prototype version of `FORPI` has been implemented in the functional programming language Scala as part of the Skeptik library. This library includes an implementation of `GFOLU` [8]. In order to evaluate the algorithm's effectiveness, `FORPI` was tested on two data sets: proofs generated by a real theorem prover and randomly-generated resolution proofs. The proofs are included in the source code repository, available at https://github.com/jgorzny/Skeptik. Note that by implementing the algorithms in this library, we are able to guarantee the correctness of the compressed proofs, as in Skeptik every inference rule (e.g. resolution, contraction) is implemented as a small class with a constructor that checks whether the conditions for the application of the rule are met, thereby preventing the creation of objects representing incorrect proof nodes (i.e. unsound inferences). With logical soundness guaranteed by Skeptik's data structures, to ensure the soundness of a proof compression algo-

TODO: how small?

rithm, we only need to check that the root clause of the compressed proof is equal to or stronger than the root clause of the input proof and that the set of axioms used in the compressed proof is a (possibly non-proper) subset of the set of axioms used in the input proof.

First, `FORPI` was evaluated on the same proofs used to evaluate `GFOLU`. This data was generated by executing the SPASS (http://www.spass-prover.org/) theorem prover on 2280 real first-order problems without equality of the TPTP Problem Library (among them, 1032 problems are known to be unsatisfiable). In order to generate pure resolution proofs, the advanced inference rules of SPASS were disabled. The proofs were originally generated on the Euler Cluster at the University of Victoria with a time limit of 300 seconds per problem. Under these conditions, SPASS generated 308 proofs. The proofs generated by SPASS were small: proof lengths varied from 3 to 49, and the number of resolutions in a proof ranged from 1 to 32.

In order to test `FORPI`'s effectiveness on larger proofs, a total of 2280 proofs were randomly generated and then used as a second benchmark set. The randomly generated proofs were much larger than those of the first data set: proof lengths varied from 95 to 700, while the number of resolutions in a proof ranged from 48 to 368. Proofs were generated by the following procedure: start with a root node whose conclusion is $\perp$, and make two premises η_1 and η_2 using a randomly generated literal such that the desired conclusion is the result of resolving η_1 and η_2. For each node η_i, determine the inference rule used to make its conclusion: with probability $p = 0.9$, η_i is the result of a resolution, otherwise it is the result of a contraction. Literals are generated by uniformly choosing a number from $\{1, \ldots, k, k+1\}$ where k is the number of predicates generated so far; if the chosen number j is between 1 and k, the j-th predicate is used; otherwise, if the chosen number is $k+1$, a new predicate with a new random arity (at most four) is generated and used. Each argument is a constant with probability $p = 0.7$ and a complex term (i.e. a function applied to other terms) otherwise; functions are generated similarly to predicates. If a node η should be the result of a resolution, then with probability $p = 0.2$ we generate a left parent η_ℓ and a right parent η_r for η (i.e. $\eta = \eta_\ell \odot \eta_r$) having a common parent η_c (i.e. $\eta_l = (\eta_\ell)_\ell \odot \eta_c$ and $\eta_r = \eta_c \odot (\eta_r)_r$, for some newly generated nodes $(\eta_\ell)_\ell$ and $(\eta_r)_r$). The common parent ensures that also non-tree-like DAG proofs are generated. This procedure is recursively applied to the generated parent nodes. Each parent of a resolution has each of its constants and functions (that are not contained in the pivot literal) replaced by a fresh variable with probability $p = 0.7$. At each recursive call, the additional minimum height required for the remainder of the branch is decreased by one with probability $p = 0.5$. Thus if each branch always decreases the additional required height, the proof has height equal to the initial minimum value. The process stops when every branch is required to add a subproof of height zero, or after a timeout. The end of each branch is taken as an axiom. The minimum height was set to 7 (which is the minimum number of nodes in an irregular proof plus one) and the timeout was set to 300 seconds (the same timeout allowed for SPASS).

For consistency, the same system and metrics were used. Proof compression and proof generation was performed on a laptop (2.8GHz Intel Core i7 processor with 4GB of RAM (1333MHz DDR3) available to the Java Virtual Machine). For each proof ψ, we measured the time needed to compress the proof ($t(\psi)$) and the compression ratio ($(|\psi| - |\alpha(\psi)|)/|\psi|$) where $|\psi|$ is the number of resolutions in the proof, and $\alpha(\psi)$ is the result of applying a compression algorithm or some composition of `FORPI` and `GFOLU`. Note that we consider only the number of resolutions in order to compare the results of these algorithms to their propositional variants (where contractions are implicit). Moreover, contractions could be made implicit within resolution inferences even in the first-order case and we use explicit contractions only for technical convenience.

Table 1 summarizes the results of `FORPI` and its combinations with `GFOLU`. The first set of columns describes the percentage of proofs that were compressed by each compression algorithm. The algorithm 'Best' runs both combinations of `GFOLU` and `FORPI` and returns the shortest proof output by either of them. The total number of proofs is $308 + 2280 = 2588$ and the total number of resolution nodes is $2,249 + 393,883 = 396,132$. The percentages in the last three columns are computed by $(\Sigma_{\psi \in \Psi}|\psi| - \Sigma_{\psi \in \Psi}|\alpha(\psi)|)/(\Sigma_{\psi \in \Psi}|\psi|)$ for each data set Ψ (TPTP, Random, or Both). The use of `FORPI` alongside `GFOLU` allows at least an additional 5% of proofs to be compressed. Furthermore, the use of both algorithms removes more than twice as many nodes than any single algorithm.

Table 2 compares the results of `FORPI` and its combinations with `GFOLU` with their propositional variants as evaluated in [4]. The first column describes the mean compression ratio for each algorithm including proofs that were not compressed by the algorithm,

Algorithm	# of Proofs Compressed			# of Removed Nodes		
	TPTP	Random	Both	TPTP	Random	Both
GFOLU(p)	55 (17.9%)	815 (35.7%)	870 (33.6%)	107 (4.8%)	17,730 (4.5%)	17,837 (4.5%)
FORPI(p)	11 (3.6%)	252 (11.1%)	263 (10.2%)	13 (0.6%)	15,913 (4.0%)	15,926 (4.0%)
GFOLU(FORPI(p))	55 (17.9%)	993 (43.6%)	1048 (40.5%)	108 (4.8%)	33,956(9.1%)	34,064 (9.1%)
FORPI(GFOLU(p))	11 (3.6%)	993 (43.6%)	1004 (38.8%)	108 (4.8%)	36,070 (9.1%)	36,178 (9.1%)
Best	56 (18.2%)	993 (43.6%)	1049 (40.5%)	108 (4.8%)	39,742 (10.1%)	39,850 (10.1%)

Table 1

Number of proofs compressed and number of overall nodes removed

Algorithm	First-Order Compression		Algorithm	Propositional Compression [4]
	All	Compressed Only		
GFOLU(p)	3.4%	33.1%	LU(p)	7.5%
FORPI(p)	4.5%	13.4%	RPI(p)	17.8%
GFOLU(FORPI(p))	7.6%	19.7%	(LU(RPI(p))	21.7%
FORPI(GFOLU(p))	8.1%	21.0%	(RPI(LU(p))	22.0%
Best	9.2%	22.8%	Best	22.0%

Table 2

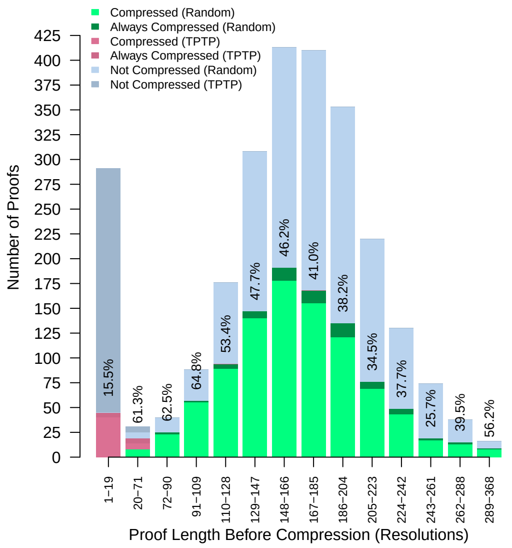
Mean compression results

while the second column calculates the mean compression ratio considering only compressed proofs. It is unsurprising that the first column is lower than the propositional mean for each algorithm: there are stricter requirements to apply these algorithms to first-order proofs. In particular, additional properties must be satisfied before a unit can be lowered, or before a pivot can be recycled. On the other hand, when first-order proofs are compressed, the levels of compression are on par with or better than their propositional counterparts.

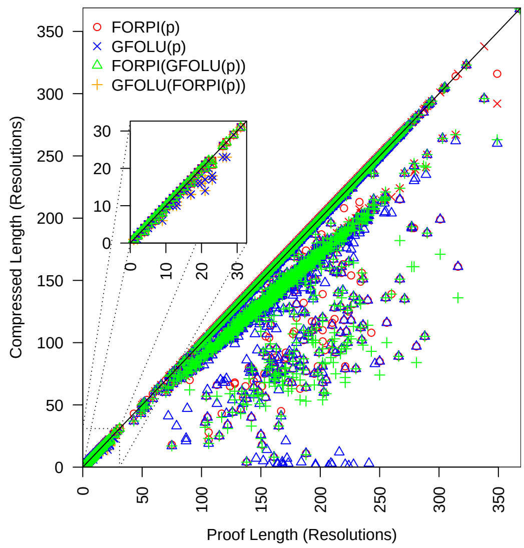
Figure 3 (a) shows the number of proofs (compressed and uncompressed) per grouping based on number of resolutions in the proof. The red (resp. dark grey) data shows the number of compressed (resp. uncompressed) proofs for the TPTP data set, while the green (resp. light grey) data shows the number of compressed (resp. uncompressed) proofs for the random proofs. The number of proofs in each group is the sum of the heights of each coloured bar in to that group. The overall percentage of proofs compressed in a group is indicated on each bar. Dark colors indicate the number of proofs compressed by FORPI, GFOLU, and both compositions of these algorithms; light colors indicate cases were FORPI succeeded, but at least one of GFOLU or a combination of these algorithms achieved zero compression. Given the size of the TPTP proofs, it is unsurprising that few are compressed: small proofs are a priori less likely to contain irregularities. On the other hand, at least 25% of randomly generated proofs in each size group could be compressed.

Figure 3 (b) is a scatter plot comparing the number of resolutions of the input proof against the number of resolutions in the compressed proof for each algorithm. The results on the TPTP data are magnified in the sub-plot. For the randomly generated proofs (points outside of the sub-plot), it is often the case that the compressed proof is significantly shorter than the input proof. Interestingly, GFOLU appears to reduce the number of resolutions by a linear factor in many cases. This is likely due to a linear growth in the number of non-interacting irregularities (i.e. irregularities for which the lowered units share no common literals with any other sub-proofs), which leads to a linear number of nodes removed.

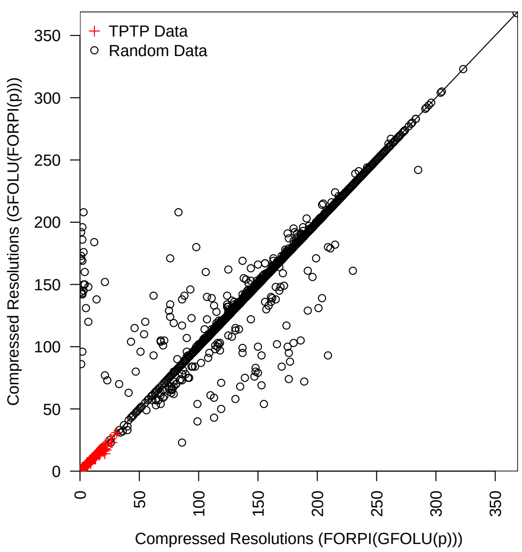
Figure 3 (c) is a scatter plot comparing the size of compression obtained by applying FORPI before GFOLU versus GFOLU before FORPI. Data obtained from the TPTP data set is marked in red; the remaining points are obtained from randomly generated proofs. Points that lie on the diagonal line have the same size after each combination. There are 165 points beneath the line and 258 points above the line. Therefore, as in the propositional case [7], it is not a priori clear which combination is better. Nevertheless, the distinctly greater number of points above the line suggests that it is more often the case that FORPI should be applied after GFOLU. Not only this combination is more likely to maximize the likelihood of compres-



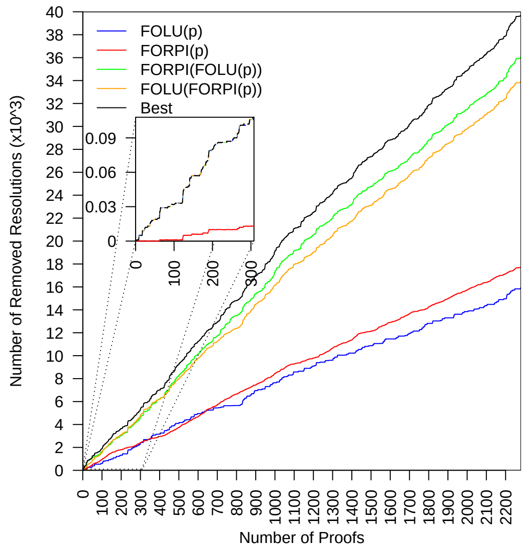
(a) Number of (non-)compressed proofs



(b) Compressed length against input length



(c) `FORPI` (`GFOLU` (p)) vs. `GFOLU` (`FORPI` (p))



(d) Cumulative proof compression

Fig. 3. `GFOLU` & `FORPI` Combination Results

sion, but the achieved compression also tends to be larger.

Figure 3 (d) shows a plot comparing the difference between the cumulative number of resolutions of the first x input proofs and the cumulative number of resolutions in the first x proofs after compression (i.e. the cumulative number of *removed* resolutions). The TPTP data is displayed in the sub-plot; note that the lines for everything except FORPI largely overlap (since the values are almost identical; cf. Table 1). Observe that the use of both algorithms is always better than using a single algorithm. The data also shows that using FORPI after GFOLU is normally the preferred order of composition, as it typically results in a greater number of nodes removed than the other combination. An even better approach is to try both combinations and choose the best result (as shown in the 'Best' curve).

SPASS required approximately 40 minutes (running on a cluster and including proof generation time for each problem) to generate all the 308 TPTP proofs. The total time to apply both FORPI and GFOLU on all these proofs was just over 8 seconds on a simple laptop computer. The random proofs were generated in 70 minutes, and took approximately 453 seconds (or 7.5 minutes) to compress, both measured on the same computer. All times include parsing time. These compression algorithms continue to be very fast, and may simplify the proof considerably for a relatively small cost in time.

7. Conclusions and Future Work

The main contribution of this paper is the lifting of the propositional proof compression algorithm RPI to the first-order case. As indicated in Section 4, the generalization is challenging, because unification instantiates literals and, consequently, a node may be regularizable even if its resolved literals are not syntactically equal to any safe literal. Therefore, unification must be taken into account when collecting safe literals and marking nodes for deletion.

We first evaluated the algorithm on all 308 real proofs that the SPASS theorem prover (with only standard resolution enabled) was capable of generating when executed on unsatisfiable TPTP problems without equality. Although the compression achieved by the first-order FORPI algorithm was not as good as the compression achieved by the propositional RPI algorithm on real proofs generated by SAT and SMT solvers [7], this is due to the fact that the 308 proofs were too short (less than 32 resolutions) to contain a significant amount of irregularities. In contrast, the propositional proofs used in the evaluation of the propositional RPI algorithm had thousands (and sometimes hundreds of thousands) of resolutions.

Our second evaluation used larger, but randomly generated, proofs. The compression achieved by FORPI in a short amount of time on this data set was compatible with our expectations and previous experience in the propositional level. The obtained results indicate that FORPI is a promising compression technique to be reconsidered when first-order theorem provers become capable of producing larger proofs. It is important to note that randomly generated proofs may differ from real proofs in shape and may be more or less likely to contain irregularities exploitable by our algorithm. Resolution restrictions and refinements (e.g. ordered resolution [**?**], hyper-resolution [**?**], unit-resulting resolution [**?**]) tend to result in longer chains of resolutions and, therefore, in proofs with a possibly larger height to length ratio. As the number of irregularities increases with height, such proofs would have a higher number of irregularities in relation to length.

In this paper, for the sake of simplicity, we considered a pure resolution calculus without restrictions, refinements or extensions. However, in practice, theorem provers do use restrictions and extensions. It is conceptually easy to adapt the algorithm described here to such variations of resolution. For instance, restricted forms of resolution (e.g. ordered resolution, hyper-resolution, unit-resulting resolution) can be simply regarded as chains of unrestricted resolutions for the purpose of proof compression. The compression process would break the chains and change the structure of the proof, but the compressed proof would still be a correct unrestricted resolution proof, albeit not necessarily satisfying the restrictions that the input proof satisfied. In the case of extensions for equality reasoning using paramodulation-like inferences, it might be necessary to apply the paramodulations to the corresponding safe literals. Another common extension of resolution is the splitting technique [**?**]. When splitting is used, each split sub-problem is solved by a separate refutation, and the compression algorithm described here could be applied to each refutation independently.

Acknowledgements

We thank the Google Summer of Code 2014 program for financial support of this research.

References

[1] *Logic for Programming, Artificial Intelligence, and Reasoning 16th International Conference, Dakar, Senegal, Revised Selected Papers*, LNCS. Springer, 2010.

[2] H. Amjad. Compressing propositional refutations. *Electronic Notes in Theoretical Computer Science*, 185:3–15, 2007.

[3] O. Bar-Ilan, O. Fuhrmann, S. Hoory, O. Shacham, and O. Strichman. Linear-time reductions of resolution proofs. In *Haifa Verification Conference*, LNCS, pages 114–128. Springer, 2008.

[4] J. Boudou and B. Woltzenlogel Paleo. Compression of propositional resolution proofs by lowering subproofs. In *Automated Reasoning with Analytic Tableaux and Related Methods - 22th International Conference*, LNCS, pages 59–73. Springer, 2013.

[5] S. Cotton. Two techniques for minimizing resolution proofs. In Ofer Strichman and Stefan Szeider, editors, *SAT 2010*, LNCS, pages 306–312. Springer, 2010.

[6] P. Fontaine, S. Merz, and B. Woltzenlogel Paleo. Exploring and exploiting algebraic and graphical properties of resolution. In *8th International Workshop on SMT*, 2010.

[7] P. Fontaine, S. Merz, and B. Woltzenlogel Paleo. Compression of propositional resolution proofs via partial regularization. In *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, LNCS, pages 237–251. Springer, 2011.

[8] J. Gorzny and B. Woltzenlogel Paleo. Towards the compression of first-order resolution proofs by lowering unit clauses. In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015. Proceedings*, 2015.

[9] S. Hetzl, A. Leitsch, G. Reis, and D. Weller. Algorithmic introduction of quantified cuts. *Theoretical Computer Science*, 549:1–16, 2014.

[10] B. Woltzenlogel Paleo. Atomic cut introduction by resolution: Proof structuring and compression. In *LPAR-16* [1], pages 463–480.

[11] S. F. Rollini, R. Bruttomesso, and N. Sharygina. An efficient and flexible approach to resolution proof reduction. In *Hardware and Software: Verification and Testing*, LNCS, pages 182–196. Springer, 2011.

[12] S. Schulz and G. Sutcliffe. Proof generation for saturating first-order theorem provers. In D. Delahaye and B. Woltzenlogel Paleo, editors, *All about Proofs, Proofs for All*, volume 55 of *Mathematical Logic and Foundations*. College Publications, London, UK, 2015.

[13] C. Sinz. Compressing propositional proofs by common subproof extraction. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *EUROCAST*, LNCS, pages 547–555. Springer, 2007.

[14] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[15] G. S. Tseitin. On the complexity of derivation in propositional calculus. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning: Classical Papers in Computational Logic 1967-1970*. Springer-Verlag, 1983.

[16] J. Vyskocil, D. Stanovský, and J. Urban. Automated proof compression by invention of new definitions. In *LPAR* [1], pages 447–462.

Appendix A. Algorithm `RecyclePivotsWithIntersection`

Note: for the reviewers' convenience, this appendix summarizes [7].

`RecyclePivotsWithIntersection` (`RPI`) [7] aims at compressing irregular proofs. It can be seen as a simple but significant modification of the `RP` algorithm described in [3], from which it derives its name. Although in the worst case full regularization can increase the proof length exponentially [15], these algorithms show that many irregular proofs can have their length decreased if a careful partial regularization is performed.

Consider an irregular proof of the form $\psi[\eta \odot_p \psi'[\eta' \odot_p \eta'']]$ and assume, without loss of generality, that $p \in \eta$ and $p \in \eta'$. Then, if $\eta' \odot_p \eta''$ is replaced by η'' within the proof-context $\psi'[\,]$, the clause $\eta \odot_p \psi'[\eta'']$ subsumes the clause $\eta \odot_p \psi'[\eta' \odot_p \eta'']$, because even though the literal $\neg p$ of η'' is propagated down, it gets resolved against the literal p of η later on below in the proof. More precisely, even though it might be the case that $\neg p \in \psi'[\eta'']$ while $\neg p \notin \psi'[\eta' \odot_p \eta'']$, it is necessarily the case that $\neg p \notin \eta \odot_p \psi'[\eta' \odot_p \eta'']$ and $\neg p \notin \eta \odot_p \psi'[\eta'']$.

Although the remarks above suggest that it is safe to replace $\eta' \odot_p \eta''$ by η'' within the proof-context $\psi'[\,]$, this is not always the case. If a node in $\psi'[\,]$ has a child in $\psi[\,]$, then the literal $\neg p$ might be propagated down to the root of the proof, and hence, the clause $\psi[\eta \odot_p \psi'[\eta'']]$ might not subsume the clause $\psi[\eta \odot_p \psi'[\eta' \odot_p \eta'']]$. Therefore, it is only safe to do the replacement if the literal $\neg p$ gets resolved in all paths from η'' to the root or if it already occurs in the root clause of the original proof $\psi[\eta \odot_p \psi'[\eta' \odot_p \eta'']]$.

These observations lead to the idea of traversing the proof in a bottom-up manner, storing for every node a set of *safe literals* that get resolved in all paths below it

input : A proof ψ
output: A possibly less-irregular proof ψ'

$\psi' \leftarrow \psi$;
traverse ψ' bottom-up and **foreach** *node* η *in* ψ' **do**
  if η *is a resolvent node* **then**
    setSafeLiterals(η) ;
    regularizeIfPossible(η)
$\psi' \leftarrow$ fix(ψ') ;
return ψ';

Algorithm 4: `RPI`

in the proof (or that already occurred in the root clause of the original proof). Moreover, if one of the node's resolved literals belongs to the set of safe literals, then it is possible to regularize the node by replacing it by one of its parents (cf. Algorithm 4).

The regularization of a node should replace a node by one of its parents, and more precisely by the parent whose clause contains the resolved literal that is safe. After regularization, all nodes below the regularized node may have to be fixed. However, since the regularization is done with a bottom-up traversal, and only nodes below the regularized node need to be fixed, it is again possible to postpone fixing and do it with only a single traversal afterwards. Therefore, instead of replacing the irregular node by one of its parents immediately, its other parent is marked as `deletedNode`, as shown in Algorithm 5. Only later during fixing, the irregular node is actually replaced by its surviving parent (i.e. the parent that is not marked as `deletedNode`).

The set of safe literals of a node η can be computed from the set of safe literals of its children (cf. Algorithm 6). In the case when η has a single child ς, the safe literals of η are simply the safe literals of ς together with the resolved literal p of ς belonging to η (p is safe for η, because whenever p is propagated down the proof through η, p gets resolved in ς). It is important to note, however, that if ς has been marked as regularized, it will eventually be replaced by η, and hence p should not be added to the safe literals of η. In this case, the safe literals of η should be exactly the same as the safe literals of ς. When η has several children, the safe literals of η w.r.t. a child ς_i contain literals that are safe on all paths that go from η through ς_i to the root. For a literal to be safe for all paths from η to the root, it should therefore be in the intersection of the sets of safe literals w.r.t. each child.

The `RP` and the `RPI` algorithms differ from each other mainly in the computation of the safe literals of a node that has many children. While `RPI` returns the intersection as shown in Algorithm 6, `RP` returns the empty set (cf. Algorithm 7). Additionally, while in `RPI` the safe literals of the root node contain all the literals of the root clause, in `RP` the root node is always assigned an empty set of literals. (Of course, this makes a difference only when the proof is not a refutation.) Note that during a traversal of the proof, the lines from 5 to 10 in Algorithm 6 are executed as many times as the number of edges in the proof. Since every node has at most two parents, the number of edges is at most twice the number of nodes. Therefore, during a traversal of a proof with n nodes, lines from 5 to 10 are executed at most $2n$ times, and the algorithm remains linear. In our prototype implementation, the sets of safe literals are instances of Scala's `mutable.HashSet` class. Being mutable, new elements can be added efficiently. And being HashSets, membership checking is done in constant time in the average case, and set intersection (line 12) can be done in $O(k.s)$, where k is the number of sets and s is the size of the smallest set.

```
input  : A node η
output: nothing (but the proof containing η may be changed)
if η.rightResolvedLiteral ∈ η.safeLiterals then
    mark left parent of η as deletedNode ;
    mark η as regularized
else if η.leftResolvedLiteral ∈ η.safeLiterals then
    mark right parent of η as deletedNode ;
    mark η as regularized
```

Algorithm 5: `regularizeIfPossible`

```
input  : A node η
output: nothing (but the node η gets a set of safe literals)
if η is a root node with no children then
    η.safeLiterals ← η.clause
else
    foreach η′ ∈ η.children do
        if η′ is marked as regularized then
            safeLiteralsFrom(η′) ← η′.safeLiterals ;
        else if η is left parent of η′ then
            safeLiteralsFrom(η′) ← η′.safeLiterals ∪ {
            η′.rightResolvedLiteral } ;
        else if η is right parent of η′ then
            safeLiteralsFrom(η′) ← η′.safeLiterals ∪ {
            η′.leftResolvedLiteral } ;
    η.safeLiterals ← ⋂_{η′∈η.children} safeLiteralsFrom(η′)
```

Algorithm 6: `setSafeLiterals`

```
input  : A node η
output: nothing (but the node η gets a set of safe literals)
if η is a root node with no children then
    η.safeLiterals ← ∅
else
    if η has only one child η′ then
        if η′ is marked as regularized then
            η.safeLiterals ← η′.safeLiterals ;
        else if η is left parent of η′ then
            η.safeLiterals ← η′.safeLiterals ∪ {
            η′.rightResolvedLiteral } ;
        else if η is right parent of η′ then
            η.safeLiterals ← η′.safeLiterals ∪ {
            η′.leftResolvedLiteral } ;
    else
        η.safeLiterals ← ∅
```

Algorithm 7: `setSafeLiterals` for `RP`