Noname manuscript No.
(will be inserted by the editor)

Partial Regularization of First-Order Resolution Proofs

Jan Gorzny · Ezequiel Postan · Bruno Woltzenlogel Paleo

the date of receipt and acceptance should be inserted later

Abstract Resolution and superposition are common techniques which have seen wide-spread use with propositional and first-order logic in modern theorem provers. In these cases, resolution proof production is a key feature of such tools; however, the proofs that they produce are not necessarily as concise as possible. For propositional resolution proofs, there are a wide variety of proof compression techniques. There are fewer techniques for compressing first-order resolution proofs generated by automated theorem provers. This paper describes an approach to compressing first-order logic proofs based on lifting proof compression ideas used in propositional logic to first-order logic. One method for propositional proof compression is *partial regularization*, which removes an inference η when it is redundant in the sense that its pivot literal already occurs as the pivot of another inference in every path from η to the root of the proof. This paper describes the generalization of the partial-regularization algorithm `RecyclePivotsWithIntersection` [10] from propositional logic to first-order logic. The generalized algorithm performs partial regularization of resolution proofs containing resolution and factoring inferences with *unification*. An empirical evaluation of the generalized algorithm and its combinations with the previously lifted `GreedyLinearFirstOrderLowerUnits` algorithm [12] is also presented.

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

J. Gorzny and E. Postan were Supported by the Google Summer of Code 2014 and Google Summer of Code 2016 programs.

Bruno ist Stipendiat der Österreichischen Akademie der Wissenschaft (APART) an der TU-Wien

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

E. Postan
Universidad Nacional de Rosario, Av. Pellegrini 250, S2000BTP Rosario, Santa Fe, Argentina
E-mail: ezequiel@fceia.unr.edu.ar

B. Woltzenlogel Paleo
Vienna University of Technology, Karlsplatz 13, 1040, Vienna, Austria
E-mail: bruno@logic.at

1 Introduction

First-order automated theorem provers, commonly based on refinements and extensions of resolution and superposition calculi [23,26,35,21,3,7,18], have recently achieved a high degree of maturity. Proof production is a key feature that has been gaining importance, as proofs are crucial for applications that require certification of a prover's answers or that extract additional information from proofs (e.g. unsat cores, interpolants, instances of quantified variables). Nevertheless, proof production is non-trivial [27], and the most efficient provers do not necessarily generate the shortest proofs. One reason for this is that efficient resolution provers use refinements that restrict the application of inference rules. Although fewer clauses are generated and the search space is reduced, refinements may exclude short proofs whose inferences do not satisfy the restriction.

Longer and larger proofs take longer to check, may consume more memory during proof-checking and occupy more storage space, and may have a larger unsat core, if more input clauses are used in the proof, and a larger Herbrand sequent, if more variables are instantiated [36,37,14,15,22]. For these technical reasons, it is worth pursuing efficient algorithms that compress proofs after they have been found. Furthermore, the problem of proof compression is closely related to Hilbert's 24th Problem [30], which asks for criteria to judge the simplicity of proofs. Proof length is arguably one possible criterion for some applications.

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 [9]. Compression algorithms based on rearranging and sharing chains of resolution inferences have been developed in [1] and [28]. Cotton [6] 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 [25] searches for locally redundant subproofs that can be rewritten into subproofs of stronger clauses and with fewer resolution steps. Bar-Ilan et al. [2] and Fontaine et al. [10] described a linear time proof compression algorithm based on partial regularization, which removes an inference η when it is redundant in the sense that its pivot literal already occurs as the pivot of another inference in every path from η to the root of the proof.

In contrast, although proof output has been a concern in first-order automated reasoning for a longer time than in propositional SAT-solving, there has been much less work on simplifying first-order proofs. For tree-like sequent calculus proofs, algorithms based on cut-introduction [20, 13] 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 Thousands of Problems for Theorem Provers (TPTP) [29] format (including DAG-like first-order resolution proofs), there is an algorithm [32] that looks for terms that occur often in any Thousands of Solutions from Theorem Provers (TSTP) [29] 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 [10], which delays resolution steps with unit clauses, and we lifted it to a new algorithm that we called `GreedyLinearFirstOrder-LowerUnits` (`GFOLU`) algorithm [12]. Here we continue this line of research by lifting the `RecyclePivotsWithIntersection` (`RPI`) algorithm [10], which improves the `RecyclePivots` (`RP`) algorithm [2] by detecting nodes that can be regularized even when they have multiple children.

Section 2 introduces the well-known first-order resolution calculus with notations that are suitable for describing and manipulating proofs as first-class objects. Section 3 summarizes the propositional `RPI` algorithm. Section 4 discusses the challenges that arise in the first-order case (mainly due to unification), which are not present in the propositional case, and conclude with conditions useful for first-order regularization. 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 [29] and on randomly generated proofs. Section 7 concludes the paper.

It is important to emphasize that this paper targets proofs in a pure first-order resolution calculus (with resolution and factoring rules only), without refinements or extensions, and without equality rules. As most state-of-the-art resolution-based provers use variations and extensions of this pure calculus and there exists no common proof format, the presented algorithm cannot be directly applied to the proofs generated by most provers, and even SPASS had to be specially configured to disable SPASS's extensions in order to generate pure resolution proofs for our experiments. By targeting the pure first-order resolution calculus, we address the common theoretical basis for the calculi of various provers. In the Conclusion (Section 7), we briefly discuss what could be done to tackle common variations and extensions, such as splitting and equality reasoning. Nevertheless, they remain topics for future research beyond the scope of this paper.

2 The Resolution Calculus

As usual, our language has 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 *underlying atom* of a literal ℓ is denoted $|\ell|$ (i.e. for any atom p, $|P| = P$ and $|\neg P| = P$). 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 $\{\neg P_1, \ldots, \neg P_n, Q_1, \ldots, Q_m\}$). $\mathrm{Var}(t)$ (resp. $\mathrm{Var}(\ell)$, $\mathrm{Var}(\Gamma)$) 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 literal ℓ *matches* another literal ℓ' if there is a substitution σ such that $\ell\sigma = \ell'$. A *unifier* of a set of literals is a substitution that makes all literals in the set equal. We will use $X \sqsubseteq Y$ to denote that X *subsumes* Y, when there exists a substitution σ such that $X\sigma \subseteq Y$.

The resolution calculus used in this paper has the following inference rules:

Definition 1 (Resolution)

$$\frac{\eta_1\colon \Gamma'_L \cup \{\ell_L\} \qquad \eta_2\colon \Gamma'_R \cup \{\ell_R\}}{\psi\colon \Gamma'_L\sigma_L \cup \Gamma'_R\sigma_R}$$

where σ_L and σ_R are substitutions such that $\ell_L\sigma_L = \overline{\ell_R}\sigma_R$. The literals ℓ_L and ℓ_R are *resolved literals*, whereas $\ell_L\sigma_L$ and $\ell_R\sigma_R$ are its *instantiated resolved literals*. The *pivot* is the underlying atom of its instantiated resolved literals (i.e. $|\ell_L\sigma_L|$ or, equivalently, $|\ell_R\sigma_R|$).

Definition 2 (Factoring)

$$\frac{\eta_1\colon \Gamma' \cup \{\ell_1, \ldots, \ell_n\}}{\psi\colon \Gamma'\sigma \cup \{\ell\}}$$

where σ is a unifier of $\{\ell_1, \ldots, \ell_n\}$ and $\ell = \ell_i\sigma$ for any $i \in \{1, \ldots, n\}$.

A *resolution proof* is a directed acyclic graph of clauses where the edges correspond to the inference rules of resolution and factoring, as explained in detail in Definition 3. A *resolution refutation* is a resolution proof with root $\bot$.

Definition 3 (First-Order Resolution Proof) A directed acyclic graph $\langle V, E, \mathit{\Gamma} \rangle$, where V is a set of nodes and E is a set of edges labeled by literals and substitutions (i.e. $E \subset V \times 2^{\mathcal{L}} \times \mathcal{S} \times V$, where $\mathcal{L}$ is the set of all literals and $\mathcal{S}$ is the set of all substitutions, 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 $\mathit{\Gamma}$ iff it is inductively constructible according to the following cases:

- **Axiom:** If $\mathit{\Gamma}$ is a clause, $\widehat{\mathit{\Gamma}}$ denotes some proof $\langle \{v\}, \varnothing, \mathit{\Gamma} \rangle$, where v is a new (axiom) node.
- **Resolution**[1]**:** If ψ_L is a proof $\langle V_L, E_L, \mathit{\Gamma}_L \rangle$ and ψ_R is a proof $\langle V_R, E_R, \mathit{\Gamma}_R \rangle$, where $\mathit{\Gamma}_L$ and $\mathit{\Gamma}_R$ satisfy the requirements of Definition 1, then $\psi_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \psi_R$ denotes a proof $\langle V, E, \mathit{\Gamma} \rangle$ s.t.

$$V = V_L \cup V_R \cup \{v\}$$
$$E = E_L \cup E_R \cup \left\{ \rho(\psi_L) \xrightarrow[\sigma_L]{\{\ell_L\}} v, \rho(\psi_R) \xrightarrow[\sigma_R]{\{\ell_R\}} v \right\}$$
$$\mathit{\Gamma} = \mathit{\Gamma}_L' \sigma_L \cup \mathit{\Gamma}_R' \sigma_R$$

  where v is a new (resolution) node and $\rho(\varphi)$ denotes the root node of φ.
- **Factoring:** If ψ' is a proof $\langle V', E', \mathit{\Gamma}' \rangle$ such that $\mathit{\Gamma}$ satisfies the requirements of Definition 2, then $\lfloor \psi \rfloor_{\{\ell_1, \ldots \ell_n\}}^{\sigma}$ denotes a proof $\langle V, E, \mathit{\Gamma} \rangle$ s.t.

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

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

[1] This is referred to as "binary resolution" elsewhere, with the understanding that "binary" refers to the number of resolved literals, rather than the number of premises of the inference rule.

Example 1 An example first-order resolution proof is shown below.
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 factoring 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 2.

3 Algorithm `RecyclePivotsWithIntersection`

This section explains `RecyclePivots-WithIntersection` (`RPI`) [10], which aims to compress irregular propositional proofs. It can be seen as a simple but significant modification of the `RP` algorithm described in [2], from which it derives its name. Although in the worst case full regularization can increase the proof length exponentially [31], these algorithms show that many irregular proofs can have their length decreased if a careful partial regularization is performed.

We write $\psi[\eta]$ to denote a *proof-context* $\psi[\_]$ with a single placeholder replaced by the subproof η. We say that a proof of the form $\psi[\eta \odot_p \psi'[\eta' \odot_p \eta_2]]$ is *irregular*.

Example 2 Consider an irregular proof and assume, without loss of generality, that $p \in \eta$ and $p \in \eta'$, as in the proof of ψ below. The proof of ψ can be written as $(\eta \odot_p (\eta_1 \odot (\eta' \odot_p \eta'')))$, or $(\eta \odot_p \psi'[(\eta' \odot_p \eta'')])$ where $\psi'[(\eta' \odot_p \eta'')] = (\eta_1 \odot (\eta' \odot_p \eta''))$ is the sub-proof of $\neg p$.

$$\dfrac{\eta\colon p \qquad \dfrac{\eta_1\colon \neg r, \neg p \qquad \dfrac{\eta'\colon p \qquad \eta''\colon \neg p, r}{r}\,p}{\neg p}}{\psi\colon \bot}\,p$$

$$\dfrac{\dfrac{\dfrac{\eta_1\colon Q(x), Q(a) \vdash P(b) \qquad \eta_2\colon P(b) \vdash}{\eta_3\colon Q(x), Q(a) \vdash}}{\eta_3'\colon Q(a) \vdash} \qquad \dfrac{\eta_2 \quad \eta_4\colon \vdash P(b), Q(y)}{\eta_5\colon \vdash Q(y)}}{\psi\colon \bot}$$

Fig. 1: A sample resolution proof.

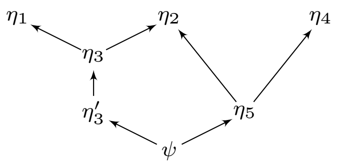


Fig. 2: The proof in Example 1.

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 $\neg p \notin \eta \odot_p \psi'[\eta'']$. In this case, the proof can be regularized as follows.

$$\dfrac{\eta\colon p \qquad \dfrac{\eta_1\colon \neg r, \neg p \qquad \eta''\colon \neg p, r}{\neg p}\; p}{\psi\colon \bot}$$

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 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 1).

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 2. 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 3). 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

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 1: `RPI`

$$\cfrac{\cfrac{\eta_1 \quad \cfrac{\eta_2 : a, c, \neg b \qquad \cfrac{\eta_1 : \neg a \qquad \eta_3 : a, b}{\eta_4 : b}\,a}{\eta_5 : a, c}\,b}{\eta_6 : c}\,a \qquad \cfrac{\cfrac{\eta_4 \qquad \eta_7 : a, \neg b, \neg c}{\eta_8 : a, \neg c}\,b \qquad \eta_1}{\eta_9 : \neg c}\,a}{\psi : \bot}\,c$$

(a) A propositional proof before compression by RPI.

$$\cfrac{\cfrac{\eta_1 : \neg a \qquad \cfrac{\eta_2 : a, c, \neg b \qquad \eta_3 : a, b}{\eta_5 : a, c}}{\eta_6 : c} \qquad \cfrac{\cfrac{\eta_3 \qquad \eta_7 : a, \neg c, \neg b}{\eta_8 : a, \neg c} \qquad \eta_1}{\eta_9 : \neg c}}{\psi : \bot}$$

(b) A propositional proof after compression by RPI.

Fig. 3: A RPI example.

input : A node η
output: nothing (but the proof containing η may be changed)

if η.rightResolvedLiteral $\in \mathcal{S}(\eta)$ **then**
    mark left parent of η as `deletedNode` ;
    mark η as regularized
else if η.leftResolvedLiteral $\in \mathcal{S}(\eta)$ **then**
    mark right parent of η as `deletedNode` ;
    mark η as regularized

Algorithm 2: `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**
    $\mathcal{S}(\eta) \leftarrow \eta$.clause
else
    foreach $\eta' \in \eta$.children **do**
        if *η' is marked as regularized* **then**
            safeLiteralsFrom(η') $\leftarrow \mathcal{S}(\eta')$;
        else if *η is left parent of η'* **then**
            safeLiteralsFrom(η') $\leftarrow \mathcal{S}(\eta') \cup$ { η'.rightResolvedLiteral } ;
        else if *η is right parent of η'* **then**
            safeLiteralsFrom(η') $\leftarrow \mathcal{S}(\eta') \cup$ { η'.leftResolvedLiteral } ;
    $\mathcal{S}(\eta) \leftarrow \bigcap_{\eta' \in \eta.\mathtt{children}}$ safeLiteralsFrom(η')

Algorithm 3: `setSafeLiterals`

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 3, RP returns the empty set (cf. Algorithm 4). 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 3 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.

Example 3 When applied to the proof ψ shown in Figure 3a, the algorithm `RPI` assigns $\{a, c\}$ and $\{a, \neg c\}$ as the safe literals of, respectively, η_5 and η_8. The safe literals of η_4 w.r.t. its children η_5 and η_8 are respectively $\{a, c, b\}$ and $\{a, \neg c, b\}$, and hence the safe literals of η_4 are $\{a, b\}$ (the intersection of $\{a, c, b\}$ and $\{a, \neg c, b\}$). Since the right resolved literal of η_4 (a) belongs to η_4's safe literals, η_4 is correctly detected as a redundant node and hence regularized: η_4 is replaced by its right parent η_3. The resulting proof is shown in Figure 3b.

4 Lifting to First-Order

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 Consider the proof ψ in Figure 4. When computed as in the propositional case, the safe literals for η_3 are $\{Q(c),\ P(a, x)\}$.

As neither of η_3's resolved literals is syntactically equal to a safe literal, the propositional `RPI` algorithm would not change ψ. However, η_3's left resolved literal $P(w, x) \in \eta_1$ 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.

$$\frac{\eta_1\colon\ \vdash P(w, x) \qquad \eta_6\colon P(y, b) \vdash}{\psi'\colon \bot}$$

Unlike in the propositional case, where a resolved literal 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 the resolved literal 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 5 The node η_3 in Figure 5 appears to be a candidate for regularization when the safe literals are computed as in the propositional case and unification is considered naïvely. Note that $\mathcal{S}(\eta_3) = \{Q(c),\ P(a, x)\}$, and the resolved literal $P(a, c)$ is unifiable with the safe literal $P(a, x)$.

However, if we attempt to regularize the proof, the same series of actions as in Example 4 would result in the proof in Figure 5, which requires resolution between η_1 and η_6, which is not possible.

One way to prevent the problem depicted above would be to require the resolved literal to be not only unifiable but subsume a safe literal. A weaker (and better) requirement is possible, and requires a slight modification of the concept of safe literals, taking into account the unifications that occur on the paths from a node to the root.

```
input  : A node η
output: nothing (but the node η gets a set of safe literals)
1  if η is a root node with no children then
2      S(η) ← ∅
3  else
4      if η has only one child η′ then
5          if η′ is marked as regularized then
6              S(η) ← S(η′) ;
7          else if η is left parent of η′ then
8              S(η) ← S(η′) ∪ { η′.rightResolvedLiteral } ;
9          else if η is right parent of η′ then
10             S(η) ← S(η′) ∪ { η′.leftResolvedLiteral } ;
11     else
12         S(η) ← ∅
```

Algorithm 4: `setSafeLiterals` for `RP`

$$\dfrac{\eta_6\colon P(y,b) \vdash \qquad \dfrac{\dfrac{\eta_1\colon \vdash P(w,x) \qquad \eta_2\colon P(w,x) \vdash Q(c)}{\eta_3\colon \vdash Q(c)} \qquad \eta_4\colon Q(c) \vdash P(a,x)}{\eta_5\colon \vdash P(a,x)}}{\psi\colon \bot}$$

Fig. 4: The proof for Example 4.

$$\dfrac{\eta_6\colon P(y,b) \vdash \qquad \dfrac{\dfrac{\eta_1\colon \vdash P(a,c) \qquad \eta_2\colon P(a,c) \vdash Q(c)}{\eta_3\colon \vdash Q(c)} \qquad \eta_4\colon Q(c) \vdash P(a,x)}{\eta_5\colon \vdash P(a,x)}}{\psi\colon \bot}$$

Fig. 5: The proof for Example 5.

Definition 4 The set of *safe literals* for a node η in a proof ψ with root clause Γ, denoted $\mathcal{S}(\eta)$, is such that $\ell \in \mathcal{S}(\eta)$ if and only if $\ell \in \Gamma$ or for all paths from η to the root of ψ there is an edge $v_1 \xrightarrow[\sigma]{\ell'} v_2$ with $\ell'\sigma = \ell$.

As in the propositional case, safe literals can be computed in a bottom-up traversal of the proof. Initially, at the root, the safe literals are exactly the literals that occur in the root clause. As we go up, the safe literals $\mathcal{S}(\eta')$ of a parent node η' of η where $\eta' \xrightarrow[\sigma]{\ell} \eta$ is set to $\mathcal{S}(\eta) \cup \{\ell\sigma\}$. Note that we apply the substitution to the resolved literal before adding it to the set of safe literals (cf. algorithm 3, lines 8 and 10). In other words, in the first-order case, the set of safe literals has to be a set of *instantiated* resolved literals.

In the case of Example 5, computing safe literals as defined above would result in $\mathcal{S}(\eta_3) = \{Q(c),\ P(a,b)\}$, where clearly the pivot $P(a,c)$ in η_1 is not safe. A generalization of this requirement is formalized below.

Definition 5 Let η be a node with safe literals $\mathcal{S}(\eta)$ and parents η_1 and η_2, assuming without loss of generality, $\eta_1 \xrightarrow[\sigma_1]{\{\ell_1\}} \eta$. The node η is said to be *pre-regularizable* in the proof ψ if $\ell_1\sigma_1$ matches a safe literal $\ell^* \in \mathcal{S}(\eta)$.

This property states that a node is pre-regularizable if an instantiated resolved literal ℓ' matches a safe literal. The notion of *pre-regulariziability* can be thought of as a *necessary* condition for recycling the node η.

$$\dfrac{\eta_8\colon Q(f(a,e),c) \vdash \qquad \dfrac{\eta_6\colon \vdash P(c,d) \qquad \dfrac{\dfrac{\eta_1\colon P(u,v) \vdash Q(f(a,v),u) \qquad \eta_2\colon Q(f(a,x),y), Q(t,x) \vdash Q(f(a,z),y)}{\eta_3\colon P(u,v), Q(t,v) \vdash Q(f(a,z),u)} \qquad \eta_4\colon \vdash Q(r,s)}{\eta_5\colon P(u,v) \vdash Q(f(a,z),u)}}{\eta_7\colon \vdash Q(f(a,z),c)}}{\psi\colon \bot}$$

Fig. 6: An example where pre-regularizability is not sufficient.

Example 6 Satisfying the pre-regularizability is not sufficient. Consider the proof ψ in Figure 6. After collecting the safe literals, $\mathcal{S}(\eta_3) = \{\neg Q(r,v), \neg P(c,d), Q(f(a,e),c)\}$. η_3's pivot $Q(f(a,v),u)$ matches 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 in Figure 7.
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 condition be satisfied.

Definition 6 Let η be pre-regularizable, with safe literals $\mathcal{S}(\eta)$ and parents η_1 and η_2, with clauses Γ_1 and Γ_2 respectively, assuming without loss of generality that $\eta_1 \xrightarrow[\sigma_1]{\{\ell_1\}} \eta$ such that $\ell_1\sigma_1$ matches a safe literal $\ell^* \in \mathcal{S}(\eta)$. The node η is said to be *strongly regularizable* in ψ if $\Gamma_1\sigma_1 \sqsubseteq \mathcal{S}(\eta)$.

This condition 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 literals of the replacement node would not change lower in the proof.

The notion of *strongly regularizable* can be thought of as a *sufficient* condition.

Theorem 41 *Let ψ be a proof with root clause Γ and η be a node in ψ. Let $\psi^\dagger = \psi \setminus \{\eta\}$ and $\Gamma^\dagger$ be the root of $\psi^\dagger$. If η is strongly regularizable, then $\Gamma^\dagger \sqsubseteq \Gamma$.*

Proof By definition of strong regularizability, η is such that there is a node η' with clause Γ' and such that $\eta' \xrightarrow[\sigma']{\{\ell'\}} \eta$ and $\ell'\sigma'$ matches a safe literal $\ell^* \in \mathcal{S}(\eta)$ and $\Gamma'\sigma' \sqsubseteq \mathcal{S}(\eta)$.

Firstly, in $\psi^\dagger$, η has been replaced by η'. Since $\Gamma'\sigma' \sqsubseteq \mathcal{S}(\eta)$, by definition of $\mathcal{S}(\eta)$, every literal ℓ in Γ' either subsumes a single literal that occurs as a pivot on every path from η to the root in ψ (and hence on every new path from η' to the root in $\psi^\dagger$) or subsumes literals $\ell\sigma_1,\ldots,\ell\sigma_n$ in Γ. In the former case, ℓ is resolved away in the construction of $\psi^\dagger$ (by contracting the descendants of ℓ with the pivots in each path). In the latter case, the literal $\ell\sigma_k$ $(1 \leq k \leq n)$ in Γ is a descendant of ℓ through a path k and the substitution σ_k is the composition of all substitutions on this path. When η is replaced by η', two things may happen to $\ell\sigma_k$. If the path k does not go through η, $\ell\sigma_k$ remains unchanged (i.e. $\ell\sigma_k \in \Gamma^\dagger$ unless the path k ceases to exist in $\psi^\dagger$). If the path k goes through η, the literal is changed to $\ell\sigma_k^\dagger$, where $\sigma_k^\dagger$ is such that $\sigma_k = \sigma'\sigma_k^\dagger$.

Secondly, when η is replaced by η', the edge from η's other parent η'' to η ceases to exist in $\psi^\dagger$. Consequently, any literal ℓ in Γ that is a descendant of a literal ℓ'' in the clause of η'' through a path via η will not belong to $\Gamma^\dagger$.

Thirdly, a literal from Γ that descends neither from η' nor from η'' either remains unchanged in $\Gamma^\dagger$ or, if the path to the node from which it descends ceases to exist in

$$\dfrac{\eta_8\colon Q(f(a,e),c) \vdash \qquad \dfrac{\eta_6\colon \vdash P(c,d) \qquad \eta_1\colon P(u,v) \vdash Q(f(a,v),u)}{\eta_7'\colon \vdash Q(f(a,d),c)}}{\psi'\colon ??}$$

Fig. 7: The proof for Example 6.

the construction of $\psi^\dagger$, does not belong to $\Gamma^\dagger$ at all.

Therefore, by the three facts above, $\Gamma^\dagger\sigma' \sqsubseteq \Gamma$, and hence $\Gamma^\dagger \sqsubseteq \Gamma$.

As the name suggests, strong regularizability is stronger than necessary. In some cases, nodes may be regularizable even if they are not strongly regularizable. A weaker condition (conjectured to be sufficient) is presented below. This alternative relies on knowledge of how literals are changed after the deletion of a node in a proof (and it is inspired by the *post-deletion unifiability condition* described for `FirstOrderLowerUnits` in [12]). However, since weak regularizability is more complicated to check, it is not as suitable for implementation as strong regularizability.

Definition 7 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 $\psi \setminus \{\eta_2\}$ that used to be[2] $\overline{\ell}$ in ψ and $\mathcal{R}^\dagger(\eta_\ell)$ is the set of literals in $\psi \setminus \{\eta_2\}$ that used to be the literals of $\mathcal{R}(\eta_\ell)$ in ψ.

[2] Because of the removal of η_2, $\overline{\ell}^\dagger$ may differ from $\overline{\ell}$.

This condition requires the ability to determine the underlying (uninstantiated) literal for each safe literal of a weakly regularizable node η. To achieve this, one could store safe literals as a pair (ℓ_s, σ_s), rather than as an instantiated literal $\ell_s\sigma_s$, although this is not necessary for the previous conditions.

Note further that there is always at least one node η_ℓ as assumed in the definition for any safe literal which was not contained in the root clause of the proof: the node which resulted in $\ell = \ell_s\sigma_s \in \mathcal{S}(\eta)$ 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 node $\eta_\ell' \neq \eta_\ell$ with the same pivot $|\ell| = |\ell_s\sigma_s|$. Consider arbitrary nodes η_1 and η_2 such that $\eta_2 \xrightarrow[\sigma_s]{\{\ell_s\}} \eta_\ell$ and $\eta_1 \xrightarrow[\sigma_1]{\{\ell_1\}} \eta_\ell$ where $\ell_s\sigma_s = \overline{\ell_1}\sigma_1$. Now consider arbitrary nodes η_1' and η_2' such that $\eta_2' \xrightarrow[\sigma_s]{\{\ell_s\}} \eta_\ell'$ and $\eta_1' \xrightarrow[\sigma_1']{\{\ell_1'\}} \eta_\ell'$ where $\ell_s\sigma_s = \overline{\ell_1'}\sigma_1'$. Since the pivots for η_ℓ and η_ℓ' are equal, we must have that $|\ell_s\sigma_s| = |\ell_1\sigma_1|$ and $|\ell_s\sigma_s| = |\ell_1'\sigma_1'|$, 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 $\mathcal{R}(\eta_\ell')$.

Informally, a node η is weakly regularizable in a proof if it can be replaced by one of its parents η_1, such that for each $\ell \in \mathcal{S}(\eta_1)$, $|\ell|$ can still be used as a pivot 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.

Example 7 This example illustrates a case where a node is weakly regularizable but not strongly regularizable. Table 1 shows the sets $\mathcal{S}(\eta)$, $\mathcal{R}(\eta)$ and $\mathcal{R}^\dagger(\eta)$ for the nodes η in the proof below. Observe that η_6 is pre-regularizable, since $\neg P(x)$ is unifiable with $\neg P(w) \in \mathcal{S}(\eta_6)$. In fact, η_6 is the only pre-regularizable node in the proof, and thus the sets $\mathcal{R}(\eta) = \emptyset$ for all $\eta \neq \eta_6$. In the proof in Figure 8, note that η_6 is not strongly regularizable: there is no unifier σ such that $\{\neg P(x), \neg Q(x), \neg R(x)\}\sigma \subseteq \mathcal{S}(\eta_6)$.

We show that η_6 is weakly regularizable, and that η_7 can be removed. Recalling that η_6 is pre-regularizable, observe that $\mathcal{R}^\dagger(\eta_6) \cup \{\overline{\neg P(w)}\}$ is unifiable. Consider the proof of $\psi \setminus \{\eta_7\}$ in Figure 9.

Now observe that for each $\ell \in \mathcal{S}(\eta_8)$ we have the following, showing that η_6 is weakly regularizable:

- $\ell = \neg Q(y)$: $\ell^\dagger = \neg Q(x)$ which is unifiable with $\overline{\ell}^\dagger = Q(z)$
- $\ell = \neg R(a)$: $\ell^\dagger = \neg R(a)$ which is (trivially) unifiable with $\overline{\ell}^\dagger = R(a)$
- $\ell = \neg P(w)$: $\ell^\dagger = \neg P(z)$ which is unifiable with $\overline{\ell}^\dagger = P(u)$
- $\ell = \neg P(y)$: $\ell^\dagger = \neg P(z)$ which is unifiable with $\overline{\ell}^\dagger = P(u)$

If a node η with parents η_1 and η_2 is pre-regularizable and strongly regularizable in ψ, then η is also weakly regularizable in ψ.

5 Implementation

`FirstOrderRecyclePivotsWith-Intersection` (`FORPI`) (cf. Algorithm 5) is a first-order generalization of the propositional `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 7), similarly to the propositional case, but additionally applying unifiers to the resolved literals (cf. Example 5). If one of the node's resolved literals matches 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 strong regularizability (cf. lines 2 and 6 of Algorithm 6). 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 6. 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.

Note that, in order to reduce notation clutter in the pseudocodes, we slightly abuse notation and do not explicitly distinguish proofs, their root nodes and the clauses stored in their root nodes. It is clear from the context whether ψ refers to a proof, to its root node or to its root clause.

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` [12]. 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 have a relative guarantee that the compressed proofs are correct, as in Skeptik ev-

η	$\mathcal{S}(\eta)$	$\mathcal{R}(\eta)$	$\mathcal{R}^{\dagger}(\eta)$
η_1	$\{P(w)\}$	$\emptyset$	$\emptyset$
η_2	$\{\neg P(w)\}$	$\emptyset$	$\emptyset$
η_3	$\{R(a), \neg P(w)\}$	$\emptyset$	$\emptyset$
η_4	$\{\neg R(a), \neg P(w)\}$	$\emptyset$	$\emptyset$
η_5	$\{Q(z), \neg R(a), \neg P(w)\}$	$\emptyset$	$\emptyset$
η_6	$\{\neg P(w), \neg Q(z), \neg R(a)\}$	$\{P(u), P(y)\}$	$\{P(u)\}$
η_7	$\{P(y), \neg P(w), \neg Q(z), \neg R(a)\}$	$\emptyset$	$\emptyset$
η_8	$\{\neg P(y), \neg P(w), \neg Q(z), \neg R(a)\}$	$\emptyset$	$\emptyset$

Table 1: The sets $\mathcal{S}(\eta)$ and $\mathcal{R}(\eta)$ for each node η in the first proof of Example 7.

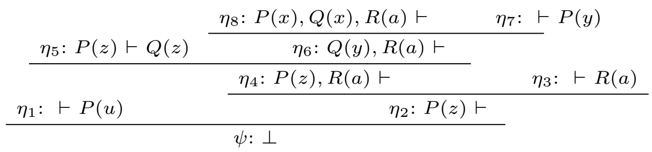


Fig. 8: The proof for Example 7.

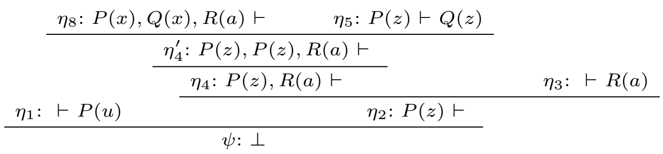


Fig. 9: The proof for Example 7 after regularization.

ery inference rule (e.g. resolution, factoring) is implemented as a small class (each at most 178 lines of code that is assumed correct) 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). 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`. These proofs were generated by executing the SPASS theorem prover (`http://www.spass-prover.org/`) on 1032 real-world unsatisfiable first-order problems without equality from the TPTP Problem Library [29]. In order to generate pure resolution proofs, the advanced inference rules of SPASS were disabled: the only enabled inference rules used were “Standard Resolution” and “Condensation”. 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 was able to generate 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.

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 5: `FORPI`

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)

if $\exists \sigma$ *and* $\ell \in \mathcal{S}(\psi)$ *such that* $\ell = \ell_R \sigma_R \sigma$ **then**
  if $\psi_R \sigma_R \sigma \subseteq \mathcal{S}(\psi)$ **then**
    mark ψ_L as `deletedNode` ;
    mark ψ as regularized
else if $\exists \sigma$ *and* $\ell \in \mathcal{S}(\psi)$ *such that* $\ell = \ell_L \sigma_L \sigma$ **then**
  if $\psi_L \sigma_L \sigma \subseteq \mathcal{S}(\psi)$ **then**
    mark ψ_R as `deletedNode` ;
    mark ψ as regularized

Algorithm 6: `regularizeIfPossible` for `FORPI`

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**
  $\mathcal{S}(\psi) \leftarrow \psi$.clause
else
  foreach $\psi' \in \psi$.children **do**
    if ψ' *is marked as regularized* **then**
      safeLiteralsFrom(ψ') $\leftarrow \mathcal{S}(\psi')$;
    else if $\psi' = \psi \odot^{\sigma_L \sigma_R}_{\ell_L \ell_R} \psi_R$ *for some* ψ_R **then**
      safeLiteralsFrom(ψ') $\leftarrow \mathcal{S}(\psi') \cup \{\ell_R \sigma_R\}$
    else if $\psi' = \psi_L \odot^{\sigma_L \sigma_R}_{\ell_L \ell_R} \psi$ *for some* ψ_L **then**
      safeLiteralsFrom(ψ') $\leftarrow \mathcal{S}(\psi') \cup \{\ell_L \sigma_L\}$
  $\mathcal{S}(\psi) \leftarrow \bigcap_{\psi' \in \psi.\mathtt{children}}$ safeLiteralsFrom(ψ')

Algorithm 7: `setSafeLiterals` for `FORPI`

6.1 Proof Generation

Additional 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 factoring.

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 terms not contained in the pivot 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 is reached. In any case, the topmost generated node for each branch is generated as an axiom node.

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). The probability values used in the random generation were carefully chosen to produce random proofs similar in shape to the real proofs obtained by SPASS. For instance, the probability of a new node being a resolution (respectively, factoring) is approximately the same as the frequency of resolutions (respectively, factorings) observed in the real proofs produced by SPASS.

6.2 Results

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 factoring is implicit). Moreover, factoring could be made implicit within resolution inferences even in the first-order case and we use explicit factoring only for technical convenience.

Table 2 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 17.5% of proofs to be compressed. Furthermore, the use of both algorithms removes almost twice as many nodes than any single algorithm.

Table 3 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, 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

Algorithm	# of Proofs Compressed			# of Removed Nodes		
	TPTP	Random	Both	TPTP	Random	Both
`GFOLU`(p)	55	817	872	107	17,769	17,876
	17.9%	35.9%	33.7%	4.8%	4.5%	4.5%
`FORPI`(p)	23	666	689	36	28,904	28,940
	7.5%	29.2%	26.2%	1.6%	7.3%	7.3%
`GFOLU`(`FORPI`(p))	55	1303	1358	120	48,126	48,246
	17.9%	57.1%	52.5%	5.4%	12.2%	12.2%
`FORPI`(`GFOLU`(p))	23	1302	1325	120	48,434	48,554
`FORPI`(`GFOLU`(p))	7.5%	57.1%	51.2%	5.4%	12.3%	12.3%
Best	59	1303	1362	120	55,530	55,650
	19.2%	57.1%	52.5%	5.4%	14.1%	14.0%

Table 2: Number of proofs compressed and number of overall nodes removed

Algorithm	First-Order Compression		Algorithm	Propositional Compression [4]
	All	Compressed Only		
`GFOLU`(p)	4.5%	13.5%	`LU`(p)	7.5%
`FORPI`(p)	6.2%	23.2%	`RPI`(p)	17.8%
`GFOLU`(`FORPI`(p))	10.6%	23.0%	(`LU`(`RPI`(p))	21.7%
`FORPI`(`GFOLU`(p))	11.1%	21.5%	(`RPI`(`LU`(p))	22.0%
Best	12.6%	24.4%	Best	22.0%

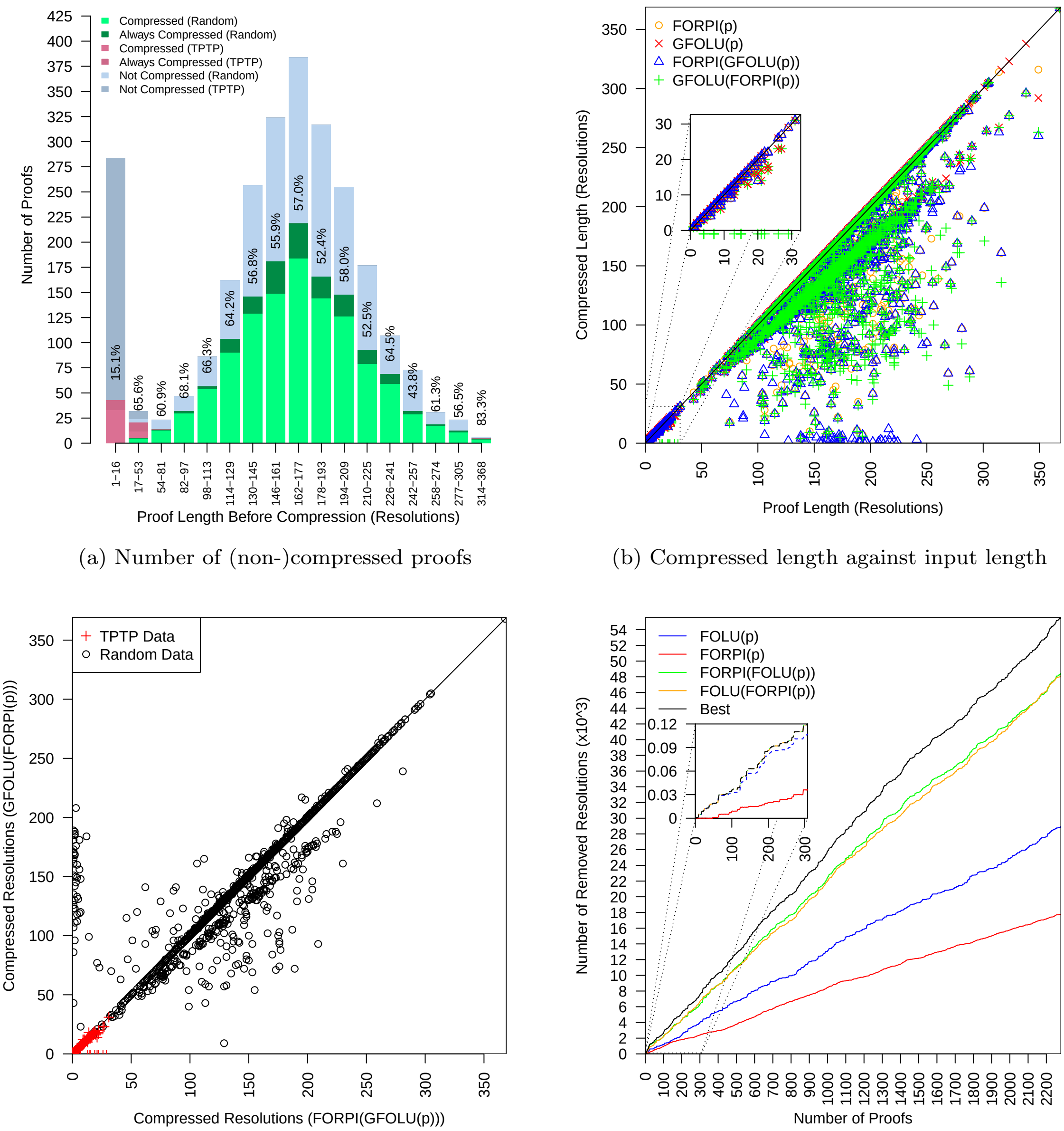
Table 3: Mean compression results

the other hand, when first-order proofs are compressed, the compression ratios are on par with or better than their propositional counterparts.

Figure 10 (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 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 43% of the randomly generated proofs in each size group could be compressed.

Figure 10 (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 subplot), 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 subproofs), which leads to a linear number of nodes removed.

Figure 10 (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



(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. 10: `GFOLU` & `FORPI` Combination Results

remaining points are obtained from randomly generated proofs. Points that lie on the diagonal line have the same size after each combination. There are 249 points beneath the line and 326 points above the line. Therefore, as in the propositional case [10], it is not a priori clear which combination will compress a proof more. 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 compression, but the achieved compression also tends to be larger.

Figure 10 (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 2). Observe that it is always better to use both algorithms than to use 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 of CPU time (running on a cluster) 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 461 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 in the first-order case, 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 [10], 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. Although we carefully selected generation probabilites in accordance with frequencies observed in real proofs, it is important to note that randomly generated proofs may still 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 [16, **?**], hyper-resolution [19,24], unit-resulting resolution [17,18]) may 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 could 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 many 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. Alternatively, equality inferences could be replaced by resolutions with instances of equality axioms, and the proof compression algorithm could be applied to the proof resulting from this replacement. Another common extension of resolution is the splitting technique [34]. When splitting is used, each split subproblem is solved by a separate refutation, and the compression algorithm described here could be applied to each refutation independently.

References

1. H. Amjad. Compressing propositional refutations. *Electronic Notes in Theoretical Computer Science*, 185:3–15, 2007.
2. 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.
3. P. Baumgartner, J. Bax, and U. Waldmann. Beagle - A hierarchic superposition theorem prover. In Felty and Middeldorp [8], pages 367–377.
4. J. Boudou and B. Woltzenlogel Paleo. Compression of propositional resolution proofs by lowering subproofs. In Galmiche and Larchey-Wendling [11], pages 59–73.
5. E. M. Clarke and A. Voronkov, editors. *Logic for Programming, Artificial Intelligence, and Reasoning 16th International Conference, Dakar, Senegal, Revised Selected Papers*, LNCS. Springer, 2010.
6. S. Cotton. Two techniques for minimizing resolution proofs. In O. Strichman and S. Szeider, editors, *SAT 2010*, LNCS, pages 306–312. Springer, 2010.
7. S. Cruanes. *Extending superposition with integer arithmetic, structural induction, and beyond*. PhD thesis, École polytechnique, 2015.
8. A. P. Felty and A. Middeldorp, editors. *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*. Springer, 2015.
9. P. Fontaine, S. Merz, and B. Woltzenlogel Paleo. Exploring and exploiting algebraic and graphical properties of resolution. In *8th International Workshop on SMT*, 2010.
10. 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*, volume 6803 of *LNCS*, pages 237–251. Springer, 2011.
11. D. Galmiche and D. Larchey-Wendling, editors. *Automated Reasoning with Analytic Tableaux and Related Methods - 22th International Conference, TABLEAUX 2013, Nancy, France, September 16-19, 2013. Proceedings*, volume 8123 of *Lecture Notes in Computer Science*. Springer, 2013.
12. J. Gorzny and B. Woltzenlogel Paleo. Towards the compression of first-order resolution proofs by lowering unit clauses. In Felty and Middeldorp [8], pages 356–366.
13. S. Hetzl, A. Leitsch, G. Reis, and D. Weller. Algorithmic introduction of quantified cuts. *Theoretical Computer Science*, 549:1–16, 2014.
14. S. Hetzl, A. Leitsch, D. Weller, and B. Woltzenlogel Paleo. Herbrand sequent extraction. In *Intelligent Computer Mathematics, 9th Int. Conference, AISC 2008, 15th Symposium, Calculemus 2008, 7th Int. Conference, MKM 2008, Birmingham, UK, July 28 - August 1, 2008. Proceedings*, LNCS, pages 462–477. Springer, 2008.
15. S. Hetzl, T. Libal, M. Riener, and M. Rukhaia. Understanding resolution proofs through herbrand's theorem. In Galmiche and Larchey-Wendling [11], pages 157–171.
16. J. Hsiang and M. Rusinowitch. Proving refutational completeness of theorem-proving strategies: the transfinite semantic tree method. *J. ACM*, 38(3):558–586, 1991.
17. J. McCharen, R. Overbeek, and L. Wos. Complexity and related enhancements for automated theorem-proving programs.

Computers and Mathematics with Applications, 2:1–16, 1976.
18. W. McCune. Prover9 and mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.
19. R. A. Overbeek. An implementation of hyper-resolution. *Computers & Mathematics with Applications*, 1(2):201 – 214, 1975.
20. B. Woltzenlogel Paleo. Atomic cut introduction by resolution: Proof structuring and compression. In Clarke and Voronkov [5], pages 463–480.
21. V. Prevosto and U. Waldmann. SPASS+T. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *ESCoR*, CEUR Workshop Proceedings, pages 18–33, 2006.
22. G. Reis. Importing SMT and connection proofs as expansion trees. In Cezary Kaliszyk and Andrei Paskevich, editors, *Proceedings Fourth Workshop on Proof eXchange for Theorem Proving, PxTP 2015, Berlin, Germany, August 2-3, 2015.*, volume 186 of *EPTCS*, pages 3–10, 2015.
23. A. Riazanov and A. Voronkov. The design and implementation of vampire. *AI Commun.*, (2-3):91–110, 2002.
24. J. A. Robinson. Automatic deduction with hyper-resolution. *International Journal of Computing and Mathematics*, 1:227–234, 1965.
25. 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.
26. S. Schulz. System description: E 1.8. In K. L. McMillan, A. Middeldorp, and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, volume 8312 of *Lecture Notes in Computer Science*, pages 735–743. Springer, 2013.
27. 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.
28. 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.
29. 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.
30. R. Thiele. Hilbert's twenty-fourth problem. *The American Mathematical Monthly*, 110(1):1–24, 2003.
31. 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.
32. J. Vyskocil, D. Stanovský, and J. Urban. Automated proof compression by invention of new definitions. In Clarke and Voronkov [5], pages 447–462.
33. U. Waldmann. Ordered resolution. In B. Woltzenlogel Paleo, editor, *Towards an Encyclopaedia of Proof Systems*, pages 12–12. College Publications, London, UK, 1 edition, 1 2017.
34. C. Weidenbach. Combining superposition, sorts and splitting. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 1965–2013. Elsevier and MIT Press, 2001.
35. C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS version 3.5. In R. A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009.
36. B. Woltzenlogel Paleo. Herbrand sequent extraction. M.sc. thesis, Technische Universität Dresden; Technische Universität Wien, Dresden, Germany; Wien, Austria, 07 2007.
37. B. Woltzenlogel Paleo. *Herbrand Sequent Extraction [M.Sc. Thesis]*. VDM-Verlag, Saarbrücken, Germany, 2008.