Partial Regularization of First-Order Resolution Proofs

Jan Gorzny[1]*, Ezequiel Postan[2]*, and Bruno Woltzenlogel Paleo[3]†

[1] University of Waterloo, Waterloo, ON, Canada
jgorzny@uwaterloo.ca
[2] Universidad Nacional de Rosario, Rosario, Santa Fe, Argentina
ezequiel@fceia.unr.edu.ar
[3] Vienna University of Technology, Vienna, Austria
bruno@logic.at

Abstract

Proofs are a key interface of modern propositional and first-order theorem provers. However, this interface is complicated by proofs which are not necessarily as concise as possible. There are a wide variety of compression techniques for propositional resolution proofs, but fewer compression techniques for 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. An empirical evaluation of the approach is included.

1 Introduction

Proof production is a key feature that has been gaining importance for modern theorem provers. Proofs are a crucial interface 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). Mature first-order automated theorem provers, commonly based on refinements and extensions of resolution and superposition calculi [20, 21, 28, 18, 16], support proof generation. However, proof production is non-trivial [22], and the most efficient provers do not necessarily generate the shortest proofs.

Lengthy proofs complicate this interface: they take longer to check, consume more memory during proof-checking, occupy more storage space and are harder to exchange, may have a larger unsat core (if more input clauses are used in the proof), and have a larger Herbrand sequent if more variables are instantiated [29, 12, 13, 19]. 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 [25], which asks for criteria to judge the simplicity of proofs; proof length is one possible criterion. Efficient proof compression techniques can be integrated into theorem provers or external tools with minimal overhead. Moreover, compression techniques like those described in this paper may result in a stronger proof which uses a strict subset of the original axioms required, which could also be considered simpler.

Efficient proof compression techniques result in greater usability. Recent applications of SAT solvers to mathematical problems have resulted in very large proofs; e.g., the proof of a long-standing problem in combinatorics was initially 200GB [15]. Such proofs are hard to store, let alone validate. More practically, a restriction of 100GB of disk space per benchmark per solver prevented validation of proofs in the SAT 2014 competition [14]. The inability to write their results to disk renders these solvers useless in some cases. Moreover, even if the

*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

only direct improvement of shorter proofs is in the communication between systems, there are indirect benefits to the end-user of a tool e.g., in terms of its responsiveness.

For propositional resolution proofs, as those typically generated by SAT- and SMT-solvers, there is a wide variety of proof compression techniques. These techniques include investigating algebraic properties of resolution [6], rearranging and sharing chains of resolution inferences [1, 23], and splitting a proof according a literal which may result in a compressed proof when recombined [5]. Bar-Ilan et al. [2] and Fontaine et al. [7] 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.

By contrast, there has been much less work on simplifying first-order proofs. For tree-like sequent calculus proofs, algorithms based on cut-introduction [17, 11] have been proposed, but these may increase the size of the proof. For arbitrary proofs in the Thousands of Problems for Theorem Provers (TPTP) [24] format (including DAG-like first-order resolution proofs), there is an algorithm [26] that looks for terms that occur often in any Thousands of Solutions from Theorem Provers (TSTP) [24] 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. We first lifted the `LowerUnits` (`LU`) algorithm [7], which delays resolution steps with unit clauses, resulting in a new algorithm that we called `GreedyLinearFirstOrderLowerUnits` (`GFOLU`) [9]. Here we continue this line of research by lifting the `RecyclePivotsWithIntersection` (`RPI`) algorithm [7], which improves the `RecyclePivots` (`RP`) algorithm [2] by detecting nodes that can be regularized even when they have multiple children.

Section 2 defines the first-order resolution calculus and 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 concludes with conditions useful for first-order regularization. Section 6 presents experimental results obtained by applying this algorithm, and its combinations with `GFOLU`, on proofs generated by SPASS and randomly generated proofs. Section 7 concludes the paper.

2 The Resolution Calculus

As usual, our language has 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 $|\neg P| = P$). A *clause* is a multiset of literals. $\bot$ denotes the *empty clause*. A *unit clause* is a clause with a single literal. Sequent notation is used for clauses (i.e. $P_1, \ldots, P_n \vdash Q_1, \ldots, Q_m$ denotes the clause $\{\neg P_1, \ldots, \neg P_n, Q_1, \ldots, Q_m\}$). $\mathrm{Var}(t)$ (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 $X\sigma \subseteq Y$.

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 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. $E \subset V \times 2^{\mathcal{L}} \times \mathcal{S} \times V$, 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 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 node.
- **Resolution**[1]**:** If ψ_L is a proof $\langle V_L, E_L, \Gamma_L \rangle$ and ψ_R is a proof $\langle V_R, E_R, \Gamma_R \rangle$, σ_L and σ_R are substitutions s.t. $\ell_L\sigma_L = \overline{\ell_R}\sigma_R$, then $\psi_L \odot^{\sigma_L \sigma_R}_{\ell_L \ell_R} \psi_R$ denotes a proof $\langle V, E, \Gamma \rangle$ s.t.

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

  where v is a new (resolution) node and $\rho(\varphi)$ denotes the root node of φ. 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, $|\ell_R\sigma_R|$).
- **Factoring:** If ψ' is a proof $\langle V', E', \Gamma' \rangle$, σ is a unifier of $\{\ell_1, \ldots, \ell_n\}$, and $\ell = \ell_i\sigma$ for any $i \in \{1, \ldots, n\}$, then $\lfloor \psi \rfloor^{\sigma}_{\{\ell_1, \ldots \ell_n\}}$ denotes a proof $\langle V, E, \Gamma \rangle$ s.t.

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

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

3 The Propositional Algorithm

`RPI` (formally defined in [7]) removes *irregularities*, which are resolution inferences deriving 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 computes and stores for each node a set of *safe literals*: literals that are resolved in all paths from the node to the root of the proof or that occur in the root clause. 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),

[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.

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

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

Figure 1: A proof ψ (left), and a regularized proof ψ' (right).

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, 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's set of safe literals is always empty.

4 Lifting to First-Order

Example 4.1. *Consider the proof ψ in Figure 1. 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 ψ' in Figure 1.*

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 where mere unifiability is not enough and greater care is needed: e.g., when $\eta_1 :\vdash P(a,c)$ and $\eta_2 : P(a,c) \vdash Q(c)$ in Example 4.1. One way to prevent these cases is to require the resolved literal to be not only unifiable but subsume a safe literal. A slight modification to the concept of safe literals, which takes into account the unifications that occur on the paths from a node to the root, results in a weaker (and better) requirement.

Definition 4.1. 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 2, 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 modified case of Example 4.1, 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, which can be thought of a *necessary* condition, follows.

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

Figure 2: An example where pre-regularizability is not sufficient.

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, $\eta_1 \xrightarrow[\sigma_1]{\{\ell_1\}} \eta$. The node η is said to be *pre-regularizable* in the proof ψ if $\ell_1\sigma_1$ matches a safe literal $\ell^* \in \mathcal{S}(\eta)$.

Example 4.2. *Satisfying the pre-regularizability is not sufficient. Consider the proof ψ in Figure 2. 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 below.*

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

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, which 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 do not change lower in the proof.

Definition 4.3. 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 $\Gamma_1\sigma_1 \sqsubseteq \mathcal{S}(\eta)$.

The notion of *strongly regularizable* can be thought of as a *sufficient* condition, and the following is proved in the longer version of this paper (available on the ArXiv [10]), which also discusses a conjectured weaker condition.

Theorem 4.3. *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 $\Gamma^\dagger \sqsubseteq \Gamma$.*

5 Implementation

`FirstOrderRecyclePivotsWithIntersection` (`FORPI`) (cf. Algorithm 1) is a first-order generalization of the propositional `RPI`. `FORPI` traverses the proof in a bottom-up manner, storing for

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`

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') $\leftarrow \mathcal{S}(\psi') \cup \{\ell_L \sigma_L\}$;
    $\mathcal{S}(\psi) \leftarrow \bigcap_{\psi' \in \psi.\mathrm{children}}$ safeLiteralsFrom(ψ')

Algorithm 2: `setSafeLiterals` for `FORPI`

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 2), similarly to the propositional case, but additionally applying unifiers to the resolved literals.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 3). 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 3. 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.

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` [9].

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

Algorithm 3: `regularizeIfPossible` for `FORPI`

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

Table 1: Number of proofs compressed and number of overall nodes removed

Algorithm	First-Order Compression		Algorithm	Propositional Compression [3]
	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%

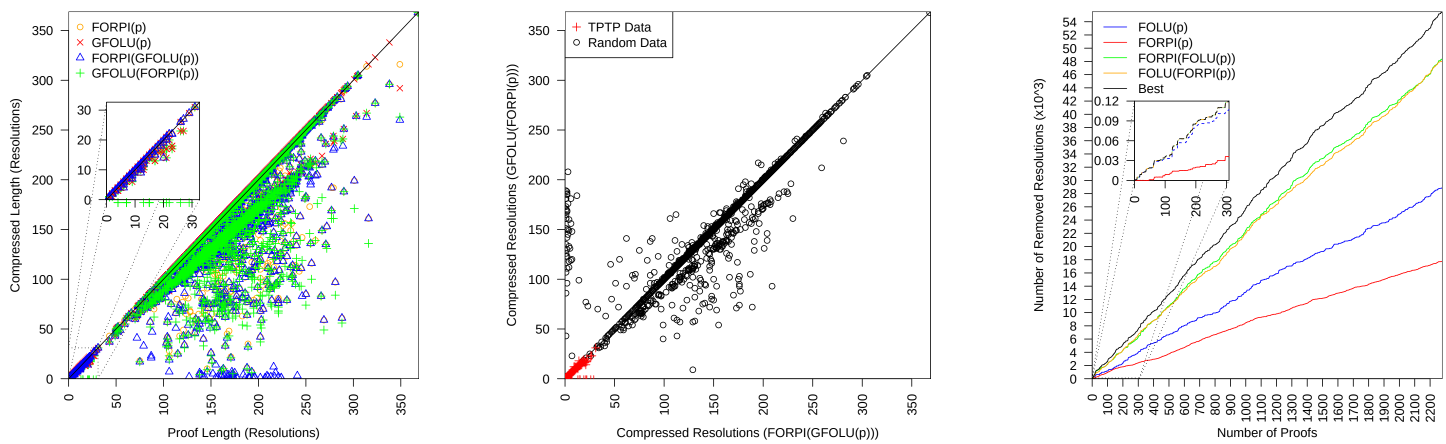
Table 2: Mean compression results

Note that by implementing the algorithms in this library, we have a relative guarantee that the compressed proofs are correct, as in Skeptik every 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 subset of the set of axioms used in the input proof.

`FORPI` was evaluated on the same 308 proofs generated by SPASS to evaluate `GFOLU`, as well as 2280 (the same number of problems initially given to SPASS) randomly generated proofs. Proof lengths varied from 3 to 700, while the number of resolutions in a proof ranged from 1 to 368 (1-32 resolutions for proofs in the TPTP data set; 1-368 resolutions for the proofs in the random data set). The same laptop was used to perform proof compression. Details and a discussion regarding the realism reflected in the random proofs can be found in [10], and the proofs are available at https://github.com/jgorzny/Skeptik.

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 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 union of the other two data sets). The use of both algorithms 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. Only nine proofs from the TPTP data set were compressed by `FORPI`, reducing the number of resolutions by at least one and at most



(a) Compressed length against input length (b) `FORPI` (`GFOLU` (p)) vs. `GFOLU` (`FORPI` (p)) (c) Cumulative proof compression

Figure 3: `GFOLU` & `FORPI` Combination Results

three. Given the size of the TPTP proofs, it is unsurprising that few are compressed: small proofs are a priori less likely to contain irregularities. However, 252 (0.11%) of the randomly generated proofs achieved some compression using only `FORPI`.

Table 2 compares the results of `FORPI` and its combinations with `GFOLU` with their propositional variants as evaluated in [3]. 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 the other hand, when first-order proofs are compressed, the compression ratios are on par with or better than their propositional counterparts.

Figure 3 (a) 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 (b) 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 249 points beneath the line and 326 points above the line. Therefore, as in the propositional case [7], it is not a priori clear which combination will compress a proof more. Applying `FORPI` after `GFOLU` is more likely to maximize the likelihood of compression, and the achieved compression also tends to be larger.

Figure 3 (c) 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). The data shows that the best approach is to try both combinations of FORPI and GFOLU and choose the best result.

Proof generation required approximately 110 minutes (including some cluster time), while the total time to apply both algorithms on all these proofs was just over 7.5 minutes, only 6.8% more time than generating proofs in the first place, on a simple laptop computer. All times include parsing time. These compression algorithms are still fast in the first-order case, and may simplify the proof considerably for a relatively small cost in time.

The use of FORPI alongside GFOLU allows at least an additional 17.5% of proofs to be compressed. Furthermore, the likelihood of compression is maximized by applying FORPI after GFOLU, and trying both compositions may be even more beneficial. On large proofs, thousands of nodes may be removed for only a small slow-down relative to the time required to generate the proof in the first place.

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. Unification must be taken into account when collecting safe literals and marking nodes for deletion.

We evaluated the algorithm on two data sets, and 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.

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, a common extension of resolution is the splitting technique [27]. When splitting is used, each split sub-problem is solved by a separate refutation, and FORPI could be applied to each refutation independently.

References

[1] H. Amjad (2007): *Compressing Propositional Refutations. ENTCS* 185, pp. 3–15. Available at http://dx.doi.org/10.1016/j.entcs.2007.05.025.

[2] O. Bar-Ilan, O. Fuhrmann, S. Hoory, O. Shacham & O. Strichman (2008): *Linear-Time Reductions of Resolution Proofs.* In: *Haifa Verification Conference*, LNCS, Springer, pp. 114–128. Available at http://dx.doi.org/10.1007/978-3-642-01702-5_14.

[3] J. Boudou & B. Woltzenlogel Paleo (2013): *Compression of Propositional Resolution Proofs by Lowering Subproofs.* In Galmiche & Larchey-Wendling [8], pp. 59–73, doi:10.1007/978-3-642-40537-2_7. Available at https://doi.org/10.1007/978-3-642-40537-2.

[4] E. M. Clarke & A. Voronkov, editors (2010): *Logic for Programming, Artificial Intelligence, and Reasoning 16th International Conference, Dakar, Senegal, Revised Selected Papers*. LNCS, Springer, doi:10.1007/978-3-642-17511-4.

[5] S. Cotton (2010): *Two Techniques for Minimizing Resolution Proofs*. In O. Strichman & S. Szeider, editors: *SAT 2010*, LNCS, Springer, pp. 306–312. Available at http://dx.doi.org/10.1007/978-3-642-14186-7_26.

[6] P. Fontaine, S. Merz & B. Woltzenlogel Paleo (2010): *Exploring and Exploiting Algebraic and Graphical Properties of Resolution*. In: *8th International Workshop on SMT*.

[7] P. Fontaine, S. Merz & B. Woltzenlogel Paleo (2011): *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* 6803, Springer, pp. 237–251. Available at http://dx.doi.org/10.1007/978-3-642-22438-6_19.

[8] D. Galmiche & D. Larchey-Wendling, editors (2013): *Automated Reasoning with Analytic Tableaux and Related Methods - 22th International Conference, TABLEAUX 2013, Nancy, France, September 16-19, 2013. Proceedings*. *LNCS* 8123, Springer, doi:10.1007/978-3-642-40537-2. Available at https://doi.org/10.1007/978-3-642-40537-2.

[9] J. Gorzny & B. Woltzenlogel Paleo (2015): *Towards the Compression of First-Order Resolution Proofs by Lowering Unit Clauses*. *LNCS* 9195, Springer, pp. 356–366, doi:10.1007/978-3-319-21401-6. Available at http://dx.doi.org/10.1007/978-3-319-21401-6.

[10] J. Gorzny, E. Postan & B. Woltzenlogel Paleo (2018): *Partial Regularization of First-Order Resolution Proofs*. *CoRR* abs/1804.06531. Available at https://arxiv.org/abs/1804.06531.

[11] S. Hetzl, A. Leitsch, G. Reis & D. Weller (2014): *Algorithmic introduction of quantified cuts*. *Theoretical Computer Science* 549, pp. 1–16, doi:10.1016/j.tcs.2014.05.018.

[12] S. Hetzl, A. Leitsch, D. Weller & B. Woltzenlogel Paleo (2008): *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, Springer, pp. 462–477, doi:10.1007/978-3-540-85110-3_38. Available at http://dx.doi.org/10.1007/978-3-540-85110-3_38.

[13] S. Hetzl, T. Libal, M. Riener & M. Rukhaia (2013): *Understanding Resolution Proofs through Herbrand's Theorem*. In Galmiche & Larchey-Wendling [8], pp. 157–171, doi:10.1007/978-3-642-40537-2_15. Available at https://doi.org/10.1007/978-3-642-40537-2_15.

[14] M. Heule & A. Biere (2016): *Clausal Proof Compression*. In B. Konev, S. Schulz & L. Simon, editors: *IWIL-2015. 11th International Workshop on the Implementation of Logics*, *EPiC Series in Computing* 40, EasyChair, pp. 21–26, doi:10.29007/sgpl. Available at https://easychair.org/publications/paper/fgBF.

[15] M. Heule, O. Kullmann & V. Marek (2016): *Solving and verifying the boolean pythagorean triples problem via cube-and-conquer*. In: *International Conference on Theory and Applications of Satisfiability Testing*, Springer, pp. 228–245.

[16] W. McCune (2005–2010): *Prover9 and Mace4*. Available at http://www.cs.unm.edu/~mccune/prover9/.

[17] B. Woltzenlogel Paleo (2010): *Atomic Cut Introduction by Resolution: Proof Structuring and Compression*. In Clarke & Voronkov [4], pp. 463–480, doi:10.1007/978-3-642-17511-4_26.

[18] V. Prevosto & U. Waldmann (2006): *SPASS+T*. In G. Sutcliffe, R. Schmidt & S. Schulz, editors: *ESCoR*, CEUR Workshop Proceedings, pp. 18–33.

[19] G. Reis (2015): *Importing SMT and Connection proofs as expansion trees*. In C. Kaliszyk & A. Paskevich, editors: *Proceedings Fourth Workshop on Proof eXchange for Theorem Proving, PxTP 2015, Berlin, Germany, August 2-3, 2015.*, *EPTCS* 186, pp. 3–10, doi:10.4204/EPTCS.186.3. Available at https://doi.org/10.4204/EPTCS.186.3.

[20] A. Riazanov & A. Voronkov (2002): *The design and implementation of VAMPIRE*. *AI Commun.* (2-3), pp. 91–110. Available at http://iospress.metapress.com/content/ajar8kjbdtdf7kc2/.

[21] S. Schulz (2013): *System Description: E 1.8.* In K. L. McMillan, A. Middeldorp & A. Voronkov, editors: *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, *LNCS* 8312, Springer, pp. 735–743, doi:10.1007/978-3-642-45221-5_49. Available at http://dx.doi.org/10.1007/978-3-642-45221-5_49.

[22] S. Schulz & G. Sutcliffe (2015): *Proof Generation for Saturating First-Order Theorem Provers.* In D. Delahaye & B. Woltzenlogel Paleo, editors: *All about Proofs, Proofs for All*, *Mathematical Logic and Foundations* 55, College Publications, London, UK.

[23] C. Sinz (2007): *Compressing Propositional Proofs by Common Subproof Extraction.* In R. Moreno-Díaz, F. Pichler & A. Quesada-Arencibia, editors: *EUROCAST*, LNCS, Springer, pp. 547–555. Available at http://dx.doi.org/10.1007/978-3-540-75867-9_69.

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

[25] R. Thiele (2003): *Hilbert's Twenty-Fourth Problem.* *The American Mathematical Monthly* 110(1), pp. 1–24. Available at http://www.jstor.org/stable/3072340.

[26] J. Vyskocil, D. Stanovský & J. Urban (2010): *Automated Proof Compression by Invention of New Definitions.* In Clarke & Voronkov [4], pp. 447–462, doi:10.1007/978-3-642-17511-4_25.

[27] C. Weidenbach (2001): *Combining Superposition, Sorts and Splitting.* In J. A. Robinson & A. Voronkov, editors: *Handbook of Automated Reasoning (in 2 volumes)*, Elsevier and MIT Press, pp. 1965–2013.

[28] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda & P. Wischnewski (2009): *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*, *LNCS* 5663, Springer, pp. 140–145, doi:10.1007/978-3-642-02959-2_10. Available at http://dx.doi.org/10.1007/978-3-642-02959-2_10.

[29] B. Woltzenlogel Paleo (2007): *Herbrand Sequent Extraction.* M.sc. thesis, Technische Universität Dresden; Technische Universität Wien, Dresden, Germany; Wien, Austria.