Partial Regularization of First-Order Resolution Proofs

Jan Gorzny[1] ⋆ and Bruno Woltzenlogel Paleo[2,3]

[1] jgorzny@uvic.ca, University of Victoria, Canada
[2] bruno@logic.at, Vienna University of Technology, Austria
[3] Australian National University

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

1 Introduction

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

For proofs using propositional resolution generated by SAT- and SMT-solvers, there is a wide variety of proof compression techniques. Algebraic properties of the resolution operation that might be useful for compression were investigated in [5]. Compression algorithms based on rearranging and sharing chains of resolution inferences have been developed in [2] and [10]. Cotton [4] 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 [9] searches for locally redundant subproofs that can be rewritten into subproofs of stronger clauses and with fewer resolution steps. A linear time proof compression algorithm based on partial regularization was proposed in [3] and improved in [6].

In contrast, there has been much less work on simplifying first-order proofs. For tree-like sequent calculus proofs, algorithms based on cut-introduction [8, 7] have been proposed. However, converting a DAG-like resolution or superposition proof, as usually generated by current provers, into a tree-like sequent calculus proof may increase the size of the proof. For arbitrary proofs in the TPTP

⋆ Supported by the Google Summer of Code 2014 program.

[11] format (including DAG-like first-order resolution proofs), there is a simple algorithm [12] that looks for terms that occur often in any TSTP [11] proof and introduces abbreviations for these terms.

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` algorithm, which delays resolution steps with unit clauses, and its lifting resulted in the `GreedyLinearFirstOrderLowerUnits` (`GFOLU`) algorithm. Here we continue this line of research by lifting the `Recycle-PivotsWithIntersection` (`RPI`) algorithm [], which is an improvement of the `RecyclePivots` (`RP`) algorithm [**?**], providing better compression on proofs where nodes have several children.

Section 2 introduces the first-order resolution calculus and the notations used in this paper. Section 4 discusses the challenges that arise in the first-order case (mainly due to unification), which are not present the propositional case. Section 5 describes an algorithm that overcomes these challenges. Section 6 concludes the paper by presenting experimental results obtained by applying this algorithm, and also its combinations with `GFOLU`, on hundreds of proofs generated with the SPASS theorem prover.

2 The Resolution Calculus

We assume that there are infinitely many variable symbols (e.g. X, Y, Z, X_1, X_2, $\ldots$), constant symbols (e.g. a, b, c, a_1, a_2, $\ldots$), function symbols of every arity (e.g f, g, f_1, f_2, $\ldots$) and predicate symbols of every arity (e.g. p, q, p_1, p_2,$\ldots$). A *term* is any variable, constant or the application of an n-ary function symbol to n terms. An *atomic formula* (*atom*) is the application of an n-ary predicate symbol to n terms. A *literal* is an atom or the negation of an atom. The *complement* of a literal ℓ is denoted $\overline{\ell}$ (i.e. for any atom p, $\overline{p} = \neg p$ and $\overline{\neg p} = p$). The set of all literals is denoted $\mathcal{L}$. A *clause* is a multiset of literals. $\bot$ denotes the *empty clause*. A *unit clause* is a clause with a single literal. Sequent notation is used for clauses (i.e. $p_1, \ldots, p_n \vdash q_1, \ldots, q_m$ denotes the clause $\{\neg p_1, \ldots, \neg p_n, q_1, \ldots, q_m\}$). $\mathrm{FV}(t)$ (resp. $\mathrm{FV}(\ell)$, $\mathrm{FV}(\Gamma)$) denotes the set of variables in the term t (resp. in the literal ℓ and in the clause Γ). A *substitution* $\{X_1\backslash t_1, X_2\backslash t_2, \ldots\}$ is a mapping from variables $\{X_1, X_2, \ldots\}$ to, respectively, terms $\{t_1, t_2, \ldots\}$. The application of a substitution σ to a term t, a literal ℓ or a clause Γ results in, respectively, the term $t\sigma$, the literal $\ell\sigma$ or the clause $\Gamma\sigma$, obtained from t, ℓ and Γ by replacing all occurrences of the variables in σ by the corresponding terms in σ. The set of all substitutions is denoted $\mathcal{S}$. A *unifier* of a set of literals is a substitution that makes all literals in the set equal. A *resolution proof* is a directed acyclic graph of clauses where the edges correspond to the inference rules of resolution and contraction (as explained in detail in Definition 1). A *resolution refutation* is a resolution proof with root $\bot$.

Definition 1 (First-Order Resolution Proof).
A directed acyclic graph $\langle V, E, \Gamma\rangle$, where V is a set of nodes and E is a set

of edges labeled by literals and substitutions (i.e. $E \subset V \times 2^{\mathcal{L}} \times \mathcal{S} \times V$ and $v_1 \xrightarrow[\sigma]{\ell} v_2$ denotes an edge from node v_1 to node v_2 labeled by the literal ℓ and the substitution σ), is a proof of a clause Γ iff it is inductively constructible according to the following cases:

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

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

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

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

where v is a new (resolution) node and $\rho(\varphi)$ denotes the root node of φ. The resolved atom *ℓ is such that $\ell = \ell_L \sigma_L = \overline{\ell_R} \sigma_R$ or $\ell = \overline{\ell_L} \sigma_L = \ell_R \sigma_R$.*
– ***Contraction:*** *If ψ' is a proof $\langle V', E', \Gamma' \rangle$ and σ is a unifier of $\{\ell_1, \ldots \ell_n\}$ with $\{\ell_1, \ldots \ell_n\} \subseteq \Gamma'$, then $\lfloor \psi \rfloor_{\{\ell_1, \ldots \ell_n\}}^{\sigma}$ denotes a proof $\langle V, E, \Gamma \rangle$ s.t.*

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

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

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

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

When we write $\psi_L \odot_{\ell_L \ell_R} \psi_R$, we assume that the omitted substitutions are such that the resolved atom is most general. When the literals and substitutions are irrelevant or clear from the context, we may write simply $\psi_L \odot \psi_R$. The $\odot$ operator is assumed to be left-associative. In the propositional case, we omit contractions (treating clauses as sets instead of multisets) and $\psi_L \odot_{\ell \overline{\ell}}^{\emptyset \emptyset} \psi_R$ is abbreviated by $\psi_L \odot_\ell \psi_R$.

If $\psi = \varphi_L \odot \varphi_R$ or $\psi = \lfloor \varphi \rfloor$, then φ, φ_L and φ_R are *direct subproofs* of ψ and ψ is a *child* of both φ_L and φ_R. The transitive closure of the direct subproof relation is the *subproof* relation. A subproof which has no direct subproof is an *axiom* of the proof. V_ψ, E_ψ and Γ_ψ denote, respectively, the nodes, edges and proved clause (conclusion) of ψ. If ψ is a proof ending with a resolution node, then ψ_L and ψ_R denote, respectively, the left and right premises of ψ.

3 The Propositional Algorithm

`RPI` removes *irregularities*, which are resolution inferences with a node η when the resolved literal (a.k.a. *pivot*) 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.

ToDo: Informal textual description of the propositional algorithm, explaining what safe literals are. Refer reader to the CADE 2011 paper (where RPI is described) for a formal description of the propositional algorithm. Consider adding the formal description to an appendix in this paper, for the convenience of the reviewer.

The `RPI` and the `RP` algorithms differ from each other mainly in the computation of the safe literals of a node that has many children. While the former returns the intersection as shown in Algorithm 3, the latter returns the empty set. Further, while in `RPI` the safe literals of the root node contain all the literals of the root clause, in `RP` the root node is always assigned an empty set of literals.

4 First-Order Challenges

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

Example 1. Consider the following irregular proof ψ. Naively computed, the safe literals for η_3 are $\{\vdash q(c),\ p(a,X)\}$. η_3's left pivot $p(W,X) \in \eta_1$ is unifiable with $p(a,X)$ in its safe literals, and thus the proof can be regularized by recycling η_1.

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

Regularization of the proof by recycling η_1 results in removing the inference between η_2 and η_3, which in turn replaces η_3 by η_1. Since η_1 cannot be resolved against η_4, and η_1 contained safe literals, η_5 is replaced by η_1. The result is the much shorter proof below.

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

Unlike in the propositional case, where the pivots and their corresponding safe literal list are all syntactically equal, in the first-order case, this is not necessarily the case. As illustrated above, $p(W,X)$ and $p(a,X)$ are not syntactically equal. Nevertheless, they are unifiable, and the proof can be regularized.

Example 2. There are cases, as shown below, that require more careful care when attempting to regularize. Again, naively computed, the safe literals for η_3 are $\{\vdash q(c),\ p(a,X)\}$, and so η_1 appears to be a candidate for regularization.

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

However, if we attempt to regularize the proof, the same series of actions as in Example 1 would require resolution between η_1 and η_5, which is not possible.

This observation implies that the following property should be satisfied before attempting regularization.

Definition 2. *Let η be a clause with literal ℓ' with corresponding safe literal ℓ which is resolved against literals $\ell_1, \ldots, \ell_n$ in a proof ψ. η is said to satisfy the* pre-regularization unifiability property *in ψ if $\ell_1, \ldots, \ell_n$, and $\overline{\ell'}$ are unifiable.*

One technique to ensure this property is met is to apply the unifier of a resolution to each resolvent before computing the safe literals. In the case of Example 2, this would result in η_3 having the safe literals $\{\vdash q(c),\ p(a,b)\}$, and now it is clear that the literal in η_1 is not safe.

Example 3. Satisfying the pre-regularization unifiability property is not sufficient to attempt regularization. Consider the proof ψ below. After collecting the safe literals, η_3's safe literals are $\{q(T,V), p(c,d) \vdash q(f(a,e),c)\}$.

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

Since $q(f(a,X),Y) \in \eta_2$ and $q(T,V)$ (in η_3's safe literals) are unifiable, regularization would be attempted. In this case, the inference between η_2 and η_3 for would be removed, and as a result, η_3 will be replaced with η_1. η_1 does not contain the required pivot for η_5, and so η_5 is also replaced with η_1, and resolution is attempted before η_1 and η_6, which results in η_7', and an inability to complete the proof, as shown below.

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

In order to avoid these scenarios, we perform an additional check during inference removal. The node $\eta*$ which will replace a resolution η (because η would have a deleted parent), must be entirely contained, via unification which modifies only η^*'s variables, in the safe literals of η. In this example, η_1 does not satisfy this property: in order to unify with η_3's safe literals, it would be necessary to send $V \rightarrow Z$ due to η_1's second literal, but leave V unchanged due η_1's first literal, which is not possible. This check is not necessary in the propositional case, as the replacement node would be contained exactly in the set of safe literals, and would not change lower in the proof.

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`

5 First-Order RecyclePivotsWithIntersection

This section presents `FirstOrderRecyclePivotsWithIntersection` (`FORPI`), Algorithm 1, a first order generalization of `RPI`. `FORPI` traverses the proof in a bottom-up manner, storing for every node a set of safe literals that are resolved in all paths below it in the proof (or that already occurred in the root clause of the original proof). If one of the node's resolved literals can be unified to a literal in the set of safe literals, then it may be possible to regularize the node by replacing it by one of its parents.

In the propositional case, regularization of a node replaces it by the parent whose clause contains the resolved literal that is safe. In the first order case, because unification introduces complications like those seen in Example 3, we ensure that the replacement parent is (possibly after unification) contained entirely in the safe literals. This ensures that the remainder of the proof does not expect a variable to be unified to different values simultaneously. After regularization, all nodes below the regularized node may have to be fixed. Similar to `RPI`, instead of replacing the irregular node by one of its parents immediately, its other parent is replaced by `deletedNodeMarker`, as shown in Algorithm 2. As in the propositional case, fixing of the proof is postponed to another (single) traversal, as regularization proceeds bottom up and only nodes below a regularized node may require fixing. During fixing, the irregular node is actually replaced by the parent that is not `deletedNodeMarker`. With careful bookkeeping, it is often possible to contract nodes during proof fixing to compress the proof further.

This is easy accomplished in the first order case by changing lines 11 and 2, respectively, of Algorithm 3. This makes a difference only when the proof is not a refutation.

The set of safe literals for a node ψ is computed from the set of safe literals of its children (cf. Algorithm 3), similar to the propositional case, but additionally applies unifiers to the resolved pivots (cf. Example 2).

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

if $\exists\sigma$ *and* $\ell \in \psi$.safeLiterals *such that* $\sigma\ell = \ell_R$ *or* $\ell = \sigma\ell_R$ **then**
    if $\exists\sigma'$ *such that* $\sigma'\psi_R \subseteq \psi$.safeLiterals **then**
        replace ψ_L by `deletedNodeMarker` ;
        mark ψ as regularized
else if $\exists\sigma$ *and* $\ell \in \psi$.safeLiterals *such that* $\sigma\ell = \ell_L$ *or* $\ell = \sigma\ell_L$ **then**
    if $\exists\sigma'$ *such that* $\sigma'\psi_L \subseteq \psi$.safeLiterals **then**
        replace ψ_R by `deletedNodeMarker` ;
        mark ψ as regularized

Algorithm 2: `regularizeIfPossible`

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

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

Algorithm 3: `setSafeLiterals`

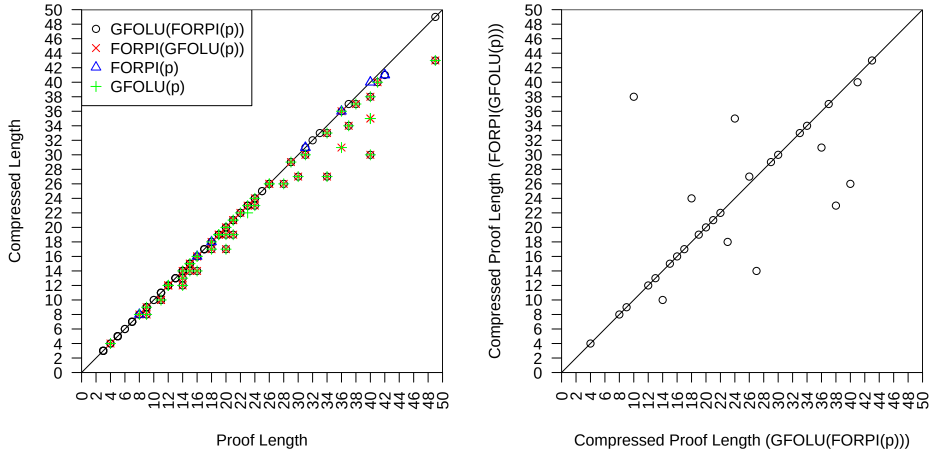
6 Experiments

A prototype[1] version of `FORPI` has been implemented in the functional programming language Scala as part of the Skeptik library. Evaluation of this algorithm was performed on the same 308 real first-order proofs generated to evaluate `GFOLU`, and for consistency, the same system and metrics were used (see []).

Figure 1 (a) shows the compression results of applying `FORPI` and `GFOLU` to the same proof (in both application orders), as well as each of these algorithms individually. Unsurprisingly, applying both algorithms generally does better than either algorithm alone. Figure 1 (b) shows that the order of compression matters, and applying `FORPI` after `GFOLU` is more successful than the alternative. This is consistent with propositional results for these algorithms.

SPASS required approximately 40 minutes to solve and generate the proofs; the total time for `GFOLU` and `FORPI` to be executed on all 308 proofs was just under 5 seconds (both include parsing time). These compression algorithms con-

[1] Source code available at https://github.com/jgorzny/Skeptik



(a) Compressed length against input length (b) `FORPI` (`GFOLU` (p)) vs. `GFOLU` (`FORPI` (p))

Fig. 1: Experimental results

tinue to be very fast, and may simplify the proof considerably for a relatively quick time cost.

References

1. *Logic for Programming, Artificial Intelligence, and Reasoning - 16th Intl. Conf., Dakar, Senegal, Rev. Selected Papers*, LNCS. Springer, 2010.
2. Hasan Amjad. Compressing propositional refutations. *Electr. Notes Theor. Comput. Sci.*, 185:3–15, 2007.
3. O. Bar-Ilan, O. Fuhrmann, S. Hoory, O. Shacham, and O. Strichman. Linear-time reductions of resolution proofs. In *Haifa Verif. Conf.*, LNCS, pages 114–128. Springer, 2008.
4. Scott Cotton. Two techniques for minimizing resolution proofs. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing SAT 2010*, volume 6175 of *LNCS*, pages 306–312. Springer, 2010.
5. P. Fontaine, S. Merz, and B. Woltzenlogel Paleo. Exploring and exploiting algebraic and graphical properties of resolution. In *8th Intl. Wkshp. on SMT*, Edinburgh, 2010.
6. P. Fontaine, S. Merz, and B. Woltzenlogel Paleo. Compression of propositional resolution proofs via partial regularization. In *CADE*, LNCS, pages 237–251. Springer, 2011.
7. S. Hetzl, A. Leitsch, G. Reis, and D. Weller. Algorithmic introduction of quantified cuts. *Theor. Comput. Sci.*, 549:1–16, 2014.
8. B. Woltzenlogel Paleo. Atomic cut introduction by resolution: Proof structuring and compression. In *LPAR-16* [1], pages 463–480.

9. 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.
10. Carsten Sinz. Compressing propositional proofs by common subproof extraction. In Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *EUROCAST*, volume 4739 of *LNCS*, pages 547–555. Springer, 2007.
11. 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.
12. J. Vyskocil, D. Stanovský, and J. Urban. Automated proof compression by invention of new definitions. In *LPAR-16* [1], pages 447–462.