Compression of First-Order Resolution Proofs by Partial Regularization

Jan Gorzny[1] * and Bruno Woltzenlogel Paleo[2] **

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

Abstract. This paper describes a generalization of the recently developed `RecyclePivotsWithIntersection` algorithm applicable to first-order logic proofs generated by automated theorem provers. `RecyclePivotsWithIntersection` removes inferences with a node η when its pivot literal occurs as the pivot of another inference located below in the path from η to the root of the proof. The algorithm is then combined with `GreedyLinearFirstOrderLowerUnits` in order to compress first-order proofs further. A preliminary empirical evaluation of the combination of these compression algorithms is presented.

1 Introduction

Most of the effort in automated reasoning so far has been dedicated to the design and implementation of proof systems and efficient theorem proving procedures. As a result, saturation-based first-order automated theorem provers have achieved a high degree of maturity, with resolution and superposition being among the most common underlying proof calculi. Proof production is an essential feature of modern state-of-the-art provers and proofs are crucial for applications where the user requires certification of the answer provided by the prover. Nevertheless, efficient proof production is non-trivial, and it is to be expected that the best, most efficient, provers do not necessarily generate the best, least redundant, proofs. Therefore, it is a timely moment to develop methods that post-process and simplify proofs.

For proofs generated by SAT- and SMT-solvers, which use propositional resolution as the basis for the DPLL and CDCL decision procedures, there is now 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 [1] and [10]. Cotton [4] proposed an algorithm that compresses a refutation by repeteadly splitting it into a proof of a heuristically chosen literal ℓ and a proof of $\overline{\ell}$, and then resolving them to form a new refutation. The `Reduce&Reconstruct` algorithm [9] searches for locally redundant subproofs that can be rewritten into subproofs of stronger clauses and

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

with fewer resolution steps. A linear time proof compression algorithm based on partial regularization was proposed in [2] and improved in [6]. Furthermore, [6] also described a new linear time algorithm called `LowerUnits`, which delays resolution with unit clauses.

In contrast, for first-order theorem provers, there has been up to now (to the best of our knowledge) no attempt to design and implement an algorithm capable of taking a first-order resolution DAG-proof and efficiently simplifying it, outputting a possibly shorter pure first-order resolution DAG-proof. There are algorithms aimed at simplifying first-order sequent calculus tree-like proofs, based on cut-introduction [8, 7], and while in principle resolution DAG-proofs can be translated to sequent-calculus tree-like proofs (and then back), such translations lead to undesirable efficiency overheads. There is also an algorithm [12] that looks for terms that occur often in any TSTP [11] proof (including first-order resolution DAG-proofs) and introduces abbreviations for these terms. However, as the definitions of the abbreviations are not part of the output proof, it cannot be checked by a pure first-order resolution proof checker.

In this paper, we continue the process of lifting propositional proof compression techniques to the first-order case, by lifting the `RecyclePivotsWithIntersection` algorithm. As shown in Section 3, the fact that first-order resolution makes use of unification leads to many challenges that simply do not exist in the propositional case. In Section 4 we describe modifications necessary to overcome these challenges. In Section 5 we present preliminary experimental results obtained by applying this algorithm along with `GreedyLinearFirstOrderLowerUnits` on hundreds of proofs generated with the SPASS theorem prover. The next section introduces the first-order resolution calculus using notations that are more convenient for describing proof transformation operations.

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.*

$$
\begin{aligned}
V &= V_L \cup V_R \cup \{v\} \\
E &= E_L \cup E_R \cup \left\{ \rho(\psi_L) \xrightarrow[\sigma_L]{\{\ell_L\}} v, \rho(\psi_R) \xrightarrow[\sigma_R]{\{\ell_R\}} v \right\} \\
\Gamma &= (\Gamma_L \setminus \{\ell_L\})\,\sigma_L \cup (\Gamma_R \setminus \{\ell_R\})\,\sigma_R
\end{aligned}
$$

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 $\{\ell_1, \dots \ell_n\}$ with $\{\ell_1, \dots \ell_n\} \subseteq \Gamma'$, then $\lfloor \psi \rfloor^{\sigma}_{\{\ell_1, \dots \ell_n\}}$ denotes a proof $\langle V, E, \Gamma \rangle$ s.t.*

$$
\begin{aligned}
V &= V' \cup \{v\} \\
E &= E' \cup \{ \rho(\psi') \xrightarrow[\sigma]{\{\ell_1, \dots \ell_n\}} v \} \\
\Gamma &= (\Gamma' \setminus \{\ell_1, \dots \ell_n\})\sigma \cup \{\ell\}
\end{aligned}
$$

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

The resolution and contraction (factoring) rules described above are the standard rules of the resolution calculus, except for the fact that we do not require resolution to use most general unifiers. The presentation of the resolution rule here uses two substitutions, in order to explicitly handle the necessary renaming of variables, which is often left implicit in other presentations of resolution.

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. We write $\lfloor \psi \rfloor$ for an arbitrary maximal contraction, and $\lfloor \psi \rfloor^{\sigma}$ for a (pseudo-)contraction that does merge no literals but merely applies the substitution σ. When the literals and substitutions are irrelevant or clear from the context, we may write simply $\psi_L \odot \psi_R$ instead of $\psi_L \odot_{\ell_L \ell_R}^{\sigma_L \sigma_R} \psi_R$. 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 First-Order Challenges

In this section, we describe challenges that have to be overcome in order to successfully adapt `RecyclePivotsWithIntersection` 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 take into account in a naive way.

Example 1. Consider the following proof ψ, noting that the proof is largely redundant. Naively computed, the safe literals for η_3 are $\{\vdash q(c),\ p(a, X)\}$. η_1 and η_5 and these two literals are unifiable. Further, the safe literals for η_1 includes η_5. Thus the proof can be regularized by recycling η_1.

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

Regularization of the proof by recycling η_1 results in deleting the edge 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.

$$\dfrac{\eta_1\colon\ \vdash p(W,X) \qquad \eta_2\colon p(Y,b) \vdash}{\psi'\colon \bot}$$

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 and η_2 appear to be candidates 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 result in the following resolution, which cannot be completed.

$$\dfrac{\eta_1\colon\ \vdash p(a,c) \qquad \eta_2\colon p(Y,b) \vdash}{\psi'\colon ??}$$

The observations above lead to the idea of requiring pivots to satisfy the following property before collecting them to be reused.

Definition 2. *Let η be a clause with literal ℓ' with corresponding safe literal ℓ which is resolved against literals ℓ_1, ..., ℓ_n in a proof ψ. η is said to satisfy the* pre-regularization unifiability property *in ψ if ℓ_1,...,ℓ_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_1 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. Due to complications with unification, there are also cases where satisfying the pre-regularization unifiability property is not sufficient to attempt regularization. Consider the proof of ψ below. After collecting the safe literals, η_3's safe literals are $\{q(B,V), p(c,d) \vdash q(f(a,e),c)\}$.

$$\dfrac{\eta_8\colon Q(f(ae)c) \vdash \qquad \dfrac{\eta_6\colon\ \vdash P(cd) \qquad \dfrac{\dfrac{\eta_1\colon P(UV) \vdash Q(f(aV)U) \qquad \eta_2\colon Q(f(aX)Y)Q(BX) \vdash Q(f(aZ)Y)}{\eta_3\colon P(UV)Q(BV) \vdash Q(f(aZ)U)} \quad \eta_4\colon\ \vdash Q(RS)}{\eta_5\colon P(UV) \vdash Q(f(aZ)U)}}{\eta_7\colon\ \vdash Q(f(aZ)c)}}{\psi\colon \bot}$$

Since $q(f(a,X),Y) \in \eta_2$ and $q(B,V)$ (in η_3's safe literals) are unifiable, regularization would be attempted. However, in this case, we would mark the edge between η_2 and η_3 for deletion, 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.

$$\dfrac{\eta_8\colon Q(f(ae)c) \vdash \qquad \dfrac{\eta_6\colon\ \vdash P(cd) \qquad \eta_1\colon P(UV) \vdash Q(f(aV)U)}{\eta_7'\colon\ \vdash Q(f(ad)c)}}{\psi'\colon ??}$$

In order to avoid these scenarios, we perform an additional check during edge deletion. 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 the case of this example, η_1 would 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. Note that this check is not necessary in the propositional case, as the replacement node would be contained exactly in the set of safe literals.

4 First-Order RecyclePivotsWithIntersection

This section presents `FirstOrderRecyclePivotsWithIntersection` (Algorithm 1), a first order generalization of `RecyclePivotsWithIntersection`, which aims to compress irregular proofs. Recall that `RecyclePivotsWithIntersection` is a modification of the `RecyclePivots` algorithm, and `RecyclePivotsWithIntersection` provides better compression on proofs where nodes have several children, when compared to `RecyclePivots`. Through a small modification to our algorithm (described later), a first order generalization of `RecyclePivots` is also possible.

Our generalization, Algorithm 1, follows the propositional 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). 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 `RecyclePivotsWithIntersection`, 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

```
input  : A first-order proof $\psi$
output: A possibly less-irregular first-order proof $\psi'$

1 $\psi' \leftarrow \psi$;
2 traverse $\psi'$ bottom-up and foreach node $\eta$ in $\psi'$ do
3     if $\eta$ is a resolvent node then
4         setSafeLiterals($\eta$) ;
5         regularizeIfPossible($\eta$)
6 $\psi' \leftarrow$ fix($\psi'$) ;
7 return $\psi'$;
```

Algorithm 1: `FirstOrderRecyclePivotsWithIntersection`

```
input  : A node $\psi = \psi_L \odot^{\sigma_L \sigma_R}_{\ell_L \ell_R} \psi_R$
output: nothing (but the proof containing $\psi$ may be changed)
if $\exists\sigma$ and $l \in \psi$.safeLiterals such that $\sigma l = l_R$ or $l = \sigma l_R$ then
    if $\exists\sigma$ such that $\sigma\psi_R \subseteq \psi$.safeLiterals then
        replace $\psi_L$ of $\eta$ by deletedNodeMarker ;
        mark $\psi$ as regularized
else if $\exists\sigma$ and $l \in \psi$.safeLiterals such that $\sigma l = l_L$ or $l = \sigma l_L$ then
    if $\exists\sigma$ such that $\sigma\psi_L \subseteq \psi$.safeLiterals then
        replace $\psi_R$ by deletedNodeMarker ;
        mark $\psi$ as regularized
```

Algorithm 2: `regularizeIfPossible`

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`.

The `RecyclePivotsWithIntersection` and the `RecyclePivots` 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 `RecyclePivotsWithIntersection` the safe literals of the root node contain all the literals of the root clause, in `RecyclePivots` the root node is always assigned an empty set of literals. 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 of a node η can be computed from the set of safe literals of its children (cf. Algorithm 3), in a manner identical to the propositional case.

```
   input  : A node $\eta$
   output: nothing (but the node $\eta$ gets a set of safe literals)
 1 if $\eta$ is a root node with no children then
 2     $\eta$.safeLiterals $\leftarrow$ $\eta$.clause
 3 else
 4     foreach $\eta' \in \eta$.children do
 5         if $\eta'$ is marked as regularized then
 6             safeLiteralsFrom($\eta'$) $\leftarrow$ $\eta'$.safeLiterals ;
 7         else if $\eta$ is left parent of $\eta'$ then
 8             safeLiteralsFrom($\eta'$) $\leftarrow$ $\eta'$.safeLiterals $\cup$ { $\eta'$.rightResolvedLiteral } ;
 9         else if $\eta$ is right parent of $\eta'$ then
10             safeLiteralsFrom($\eta'$) $\leftarrow$ $\eta'$.safeLiterals $\cup$ { $\eta'$.leftResolvedLiteral } ;
11     $\eta$.safeLiterals $\leftarrow \bigcap_{\eta' \in \eta.\text{children}}$ safeLiteralsFrom($\eta'$)
```

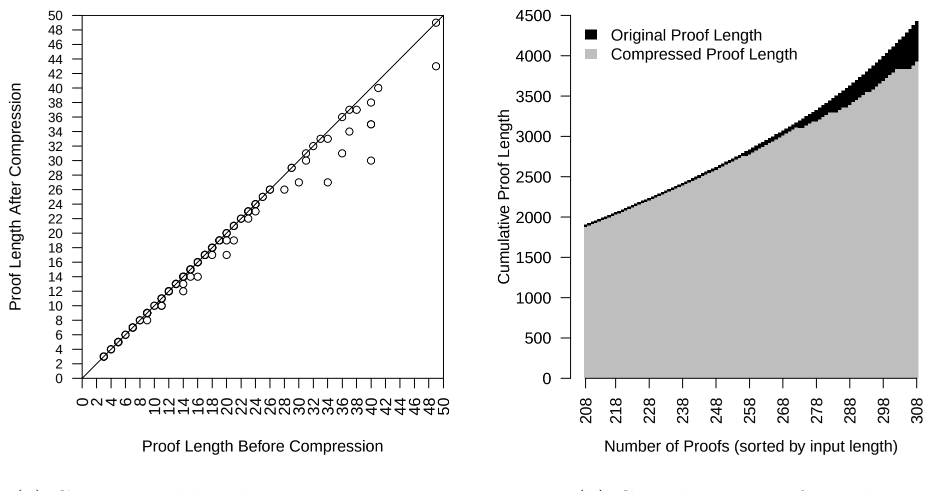
Algorithm 3: `setSafeLiterals`

5 Experiments

A prototype[1] version of `FirstOrderRecyclePivotsWithIntersection` has been implemented in the functional programming language Scala[2] as part of the Skeptik library[3]. `GreedyLinearFirstOrderLowerUnits` prototype was developed earlier for the same system [].

Evaluation of this algorithm was performed on the same 308 real first-order proofs generated to evaluate `GreedyLinearFirstOrderLowerUnits`. For consistency, the same hardware and metrics were recorded. For full details, see [].

The total time required by SPASS to solve the 308 problems for which proofs were generated was approximately 40 minutes (running on a cluster and including parsing time and proof generation time for each problem). The total time for `GreedyLinearFirstOrderLowerUnits` and `FirstOrderRecyclePivotsWithIntersection` to be executed on all 308 proofs was just under 5 seconds on a simple laptop (including parsing each proof). These compression algorithms continue to be very fast, and may simplify the proof considerably for a relatively quick time cost.



(a) Compressed length against input length (b) Cumulative proof lengths

Fig. 1: Experimental results

[1] Source code available at https://github.com/jgorzny/Skeptik
[2] http://www.scala-lang.org/
[3] https://github.com/Paradoxika/Skeptik

References

1. Hasan Amjad. Compressing propositional refutations. *Electr. Notes Theor. Comput. Sci.*, 185:3–15, 2007.
2. Omer Bar-Ilan, Oded Fuhrmann, Shlomo Hoory, Ohad Shacham, and Ofer Strichman. Linear-time reductions of resolution proofs. In Hana Chockler and Alan J. Hu, editors, *Haifa Verification Conference*, volume 5394 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2008.
3. Edmund M. Clarke and Andrei Voronkov, editors. *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, volume 6355 of *Lecture Notes in Computer Science*. Springer, 2010.
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 *Lecture Notes in Computer Science*, pages 306–312. Springer, 2010.
5. Pascal Fontaine, Stephan Merz, and Bruno Woltzenlogel Paleo. Exploring and exploiting algebraic and graphical properties of resolution. In *8th International Workshop on Satisfiability Modulo Theories - SMT 2010*, Edinburgh, Royaume-Uni, July 2010.
6. Pascal Fontaine, Stephan Merz, and Bruno Woltzenlogel Paleo. Compression of propositional resolution proofs via partial regularization. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2011.
7. Stefan Hetzl, Alexander Leitsch, Giselle Reis, and Daniel Weller. Algorithmic introduction of quantified cuts. *Theor. Comput. Sci.*, 549:1–16, 2014.
8. Bruno Woltzenlogel Paleo. Atomic cut introduction by resolution: Proof structuring and compression. In Clarke and Voronkov [3], pages 463–480.
9. Simone Fulvio Rollini, Roberto Bruttomesso, and Natasha Sharygina. An efficient and flexible approach to resolution proof reduction. In Sharon Barner, Ian Harris, Daniel Kroening, and Orna Raz, editors, *Hardware and Software: Verification and Testing*, volume 6504 of *Lecture Notes in Computer Science*, 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 *Lecture Notes in Computer Science*, 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. Jirí Vyskocil, David Stanovský, and Josef Urban. Automated proof compression by invention of new definitions. In Clarke and Voronkov [3], pages 447–462.