Towards the Compression of First-Order Resolution Proofs by Lowering Unit Clauses

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

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

Abstract. The recently developed `LowerUnits` algorithm compresses propositional resolution proofs generated by SAT- and SMT-solvers by lowering (i.e. postponing) resolution inferences involving unit clauses (i.e. clauses having exactly one literal). This paper describes a generalization of this algorithm to the case of first-order resolution proofs generated by automated theorem provers. An empirical evaluation of a simplified version of this algorithm on hundreds of proofs shows promising results.

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. And while the foundational problem of simplicity of proofs can be traced back at least to Hilbert's 24th Problem [**?**], the maturity of automated deduction has made it particularly relevant today. 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 [7]. Compression algorithms based on rearranging and sharing chains of resolution inferences have been developed in [1] and [12]. Cotton [6] 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

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

new refutation. The `Reduce&Reconstruct` algorithm [11] searches for locally redundant subproofs that can be rewritten into subproofs of stronger clauses and with fewer resolution steps. A linear time proof compression algorithm based on partial regularization was proposed in [2] and improved in [8]. Furthermore, [8] 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 [**?**,**?**], 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 [**?**] that looks for terms that occur often in any TSTP [**?**] 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 initiate the process of lifting propositional proof compression techniques to the first-order case, starting with the simplest known algorithm: `LowerUnits` (described in Section 3). As shown in Section 4, even for this simple algorithm, 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 **??** we describe a sophisticated algorithm that overcomes these challenges. Furthermore, in Section 5 we describe a simpler version of this algorithm, which is easier to implement and possibly more efficient, at the cost of compressing less. In Section 6 we present experimental results obtained by applying the simpler algorithm 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, ...), 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 set of all literals is denoted $\mathcal{L}$. A *clause* is a multiset of literals. $\bot$ denotes the *empty clause*. A *unit clause* is 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. $E \subset V \times \mathcal{L} \times \mathcal{S} \times V$ 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.
- ***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, letting $\ell = \ell_k\sigma$ (for any $k \in \{1, \ldots, n\}$), $\lfloor\psi\rfloor_\sigma^\ell$ denotes a proof $\langle V, E, \Gamma\rangle$ s.t.*

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

  where v is a new (contraction) node and $\rho(\varphi)$ denotes the root node of φ. □

If $\psi = \varphi_L \odot_\ell \varphi_R$, then 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.

Input: a proof φ
Input: D a set of subproofs
Output: a proof φ' obtained by deleting the subproofs in D from φ

```
1  if φ ∈ D or ρ(φ) has no premises then
2      return φ ;
3  else
4      let φ_L, φ_R and ℓ be such that φ = φ_L ⊙_ℓ φ_R ;
5      let φ'_L = delete(φ_L,D) ;
6      let φ'_R = delete(φ_R,D) ;
7      if φ'_L ∈ D then
8          return φ'_R ;
9      else if φ'_R ∈ D then
10         return φ'_L ;
11     else if ℓ̄ ∉ Γ_{φ'_L} then
12         return φ'_L ;
13     else if ℓ ∉ Γ_{φ'_R} then
14         return φ'_R ;
15     else
16         return  φ'_L ⊙_ℓ φ'_R ;
```

Algorithm 1: `delete`

3 The Propositional LowerUnits Algorithm

ToDo: by Bruno

The deletion algorithm is a minor variant of the Reconstruct-Proof algorithm presented in [3]. The basic idea is to traverse the proof in a top-down manner, replacing each subproof having one of its premises marked for deletion (i.e. in D) by its other direct subproof. The special case when both φ'_L and φ'_R belong to D is treated rather implicitly and deserves an explanation: in such a case, one might intuitively expect the result φ' to be undefined and arbitrary. Furthermore, to any child of φ, φ' ought to be seen as if it were in D, as if the deletion of φ'_L and φ'_R propagated to φ' as well. Instead of assigning some arbitrary proof to φ' and adding it to D, the algorithm arbitrarily returns (in line 8) φ'_R (which is already in D) as the result φ'. In this way, the propagation of deletion is done automatically and implicitly. For instance, the following hold:

$$\varphi_1 \odot_\ell \varphi_2 \setminus (\varphi_1, \varphi_2) = \varphi_2 \tag{1}$$

$$\varphi_1 \odot_\ell \varphi_2 \odot_{\ell'} \varphi_3 \setminus (\varphi_1, \varphi_2) = \varphi_3 \setminus (\varphi_1, \varphi_2) \tag{2}$$

A side-effect of this clever implicit propagation of deletion is that the actual result of deletion is only meaningful if it is not in D. In the example (1), as $\varphi_1 \odot_\ell \varphi_2 \setminus (\varphi_1, \varphi_2) \in \{\varphi_1, \varphi_2\}$, the actual resulting proof is meaningless. Only the information that it is a deleted subproof is relevant, as it suffices to obtain meaningful results as shown in (2).

Input: a proof ψ
Output: a compressed proof ψ'

```
1 Units ← ∅ ;
2 for every subproof φ in a bottom-up traversal do
3     if φ is a unit and has more than one child then
4         Enqueue φ in Units;
5 ψ' ← delete(ψ,Units) ;
6 for every unit φ in Units do
7     let {ℓ} = Γ_φ ;
8     if ℓ̄ ∈ Γ_ψ' then ψ' ← ψ' ⊙_ℓ φ ;
9     ;
```

Algorithm 2: `LowerUnits`

Proposition 1. *For any proof ψ and any sets A and B of ψ's subproofs, either $\psi \setminus (A \cup B) \in A \cup B$ and $\psi \setminus (A) \setminus (B) \in A \cup B$, or $\psi \setminus (A \cup B) = \psi \setminus (A) \setminus (B)$.*

When a subproof φ has more than one child in a proof ψ, it may be possible to *factor* all the corresponding resolutions: a new proof is constructed by removing φ from ψ and reintroducing it later. The resulting proof is smaller because φ participates in a single resolution inference in it (i.e. it has a single child), while in the original proof it participates in as many resolution inferences as the number of children it had. Such a factorization is called *lowering* of φ, because its delayed reintroduction makes φ appear at the bottom of the resulting proof.

Formally, a subproof φ in a proof ψ can be lowered if there exists a proof ψ' and a literal ℓ such that $\psi' = \psi \setminus (\varphi) \odot_\ell \varphi$ and $\Gamma_{\psi'} \subseteq \Gamma_\psi$. It has been noted in [8] that φ can always be lowered if it is a *unit*: its conclusion clause has only one literal. This led to the invention of the `LowerUnits` algorithm, which lowers every unit with more than one child, taking care to reintroduce units in an order corresponding to the subproof relation: if a unit φ_2 is a subproof of a unit φ_1 then φ_2 has to be reintroduced later than (i.e. below) φ_1.

A possible presentation of `LowerUnits` is shown in Algorithm 2. Units are collected during a first traversal. As this traversal is bottom-up, units are stored in a queue. The traversal could have been top-down and units stored in a stack. Units are effectively deleted during a second, top-down traversal. The last for-loop performs the reintroduction of units.

4 First-Order Challenges

TODO by Jan (just writing some ideas so far–not yet final by any means)
Does this belong here?And is this what you had in mind for interesting examples? And are the proof formatted correctly, or should I change them? aside from the first one going over the margin right now of course

In this section, we discuss additional requirements for lowering a unit formula in the first order case that are not required in the propositional case.

Example 1. The following example shows why we must check pair-wise unifiability with the literals resolved against the unit we're trying to lower.

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

Example 2. The following shows why the above is not necessarily enough (we must check the original sources of the aux formulas, and see if those can be contracted), otherwise we might not save anything.

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

5 A Simpler First-Order LowerUnits

ToDo by Jan

6 Experiments

ToDo by Jan

`LowerUnits` has been implemented as a prototype[1] in the functional programming language Scala[2] as part of the Skeptik library[3]. `LowerUnits` has been implemented as a recursive `delete` improvement.

The algorithm has been applied to **308** proofs produced by the SPASS[4] theorem prover on unsatisfiable benchmarks from the TPTP Problem Library[5]. The proofs used were restricted to those which could be solved within 300 seconds by SPASS on the Euler Cluster at the University of Victoria[6] using only the contraction and unifying resolution inference rules.

For each proof ψ (with the result of `LowerUnits` applied to the proof denoted by $\alpha(\psi)$), the time to compress the proof ($t(\psi)$), the compression ratio $((|\psi| - |\alpha(\psi)|)/|\psi|)$, the resolution compression ratio $((|\psi|_R - |\alpha(\psi)|_R)/|\psi|_R)$, the compression speed $((|\psi| - |\alpha(\psi)|)/t(\psi))$, and resolution compression speed

[1] Source code available at https://github.com/jgorzny/Skeptik
[2] http://www.scala-lang.org/
[3] https://github.com/Paradoxika/Skeptik
[4] http://www.verit-solver.org/
[5] http://www.cs.miami.edu/ tptp/
[6] https://rcf.uvic.ca/euler.php

$((|\psi|_R - |\alpha(\psi)|_R)/t(\psi))$ were measured[7], where $|\psi|_R$ indicates the number of resolution inference rules in the proof ψ.

The experiments were executed on a laptop (2.8GHz Intel Core i7 processor with 4 GB of RAM (1333MHz DDR3) available to the Java Virtual Machine), and the prototype implementation performed well on this system. Figure **??** shows the compression time $t(\psi)$ for each proof, sorted by proof length, and figure **??** (respectively figure **??**) shows the compression speed (respectively resolution compression speed) for each proof, also sorted by proof length.

7 Conclusions and Future Work

ToDo: by Bruno

`LowerUnivalents`, the algorithm presented here, has been shown in the previous section to compress more than `LowerUnits`. This is so because, as demonstrated in Proposition **??**, the set of subproofs it lowers is always a superset of the set of subproofs lowered by `LowerUnits`. It might be possible to lower even more subproofs by finding a characterization of (efficiently) lowerable subproofs broader than that of univalent subproofs considered here. This direction for future work promises to be challenging, though, as evidenced by the non-triviality of the optimizations discussed in Section **??** for obtaining a linear-time implementation of `LowerUnivalents`.

As discussed in Section **??**, the proposed algorithm can be embedded in the deletion traversal of other algorithms. As an example, it has been shown that the combination of `LowerUnivalents` with `RPI`, compared to the sequential composition of `LowerUnits` after `RPI`, results in a better compression ratio with only a small processing time overhead (Figure **??**). Other compression algorithms that also have a subproof deletion or reconstruction phase (e.g. `Reduce&Reconstruct`) could probably benefit from being combined with `LowerUnivalents` as well.

References

1. Amjad, H.: Compressing propositional refutations. Electr. Notes Theor. Comput. Sci. 185, 3–15 (2007)
2. Bar-Ilan, O., Fuhrmann, O., Hoory, S., Shacham, O., Strichman, O.: Linear-time reductions of resolution proofs. In: Chockler, H., Hu, A.J. (eds.) Haifa Verification Conference. Lecture Notes in Computer Science, vol. 5394, pp. 114–128. Springer (2008)
3. Bar-Ilan, O., Fuhrmann, O., Hoory, S., Shacham, O., Strichman, O.: Reducing the size of resolution proofs in linear time. STTT 13(3), 263–272 (2011)
4. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: verit: An open, trustable and efficient smt-solver. In: Schmidt, R.A. (ed.) CADE. Lecture Notes in Computer Science, vol. 5663, pp. 151–156. Springer (2009)

[7] The raw data is available at https://docs.google.com/spreadsheets/d/1F1-t2OuhypmTQhLU6yTj42aiZ5CqqaZvhVvOzeFgn0k/edit#gid=1182923972

5. Cook, S.A.: The complexity of theorem-proving procedures. In: Harrison, M.A., Banerji, R.B., Ullman, J.D. (eds.) STOC. pp. 151–158. ACM (1971)
6. Cotton, S.: Two techniques for minimizing resolution proofs. In: Strichman, O., Szeider, S. (eds.) Theory and Applications of Satisfiability Testing SAT 2010, Lecture Notes in Computer Science, vol. 6175, pp. 306–312. Springer (2010)
7. Fontaine, P., Merz, S., Woltzenlogel Paleo, B.: Exploring and exploiting algebraic and graphical properties of resolution. In: 8th International Workshop on Satisfiability Modulo Theories - SMT 2010. Edinburgh, Royaume-Uni (Jul 2010), http://hal.inria.fr/inria-00544658
8. Fontaine, P., Merz, S., Woltzenlogel Paleo, B.: Compression of propositional resolution proofs via partial regularization. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE. Lecture Notes in Computer Science, vol. 6803, pp. 237–251. Springer (2011)
9. Goerdt, A.: Comparing the complexity of regular and unrestricted resolution. In: Marburger, H. (ed.) GWAI. Informatik-Fachberichte, vol. 251, pp. 181–185. Springer (1990)
10. Järvisalo, M., Le Berre, D., Roussel, O., Simon, L.: The international SAT solver competitions. AI Magazine 33(1), 89–92 (2012)
11. Rollini, S.F., Bruttomesso, R., Sharygina, N.: An efficient and flexible approach to resolution proof reduction. In: Barner, S., Harris, I., Kroening, D., Raz, O. (eds.) Hardware and Software: Verification and Testing, Lecture Notes in Computer Science, vol. 6504, pp. 182–196. Springer (2011)
12. Sinz, C.: Compressing propositional proofs by common subproof extraction. In: Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A. (eds.) EUROCAST. Lecture Notes in Computer Science, vol. 4739, pp. 547–555. Springer (2007)
13. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Siekmann, J., Wrightson, G. (eds.) Automation of Reasoning: Classical Papers in Computational Logic 1967-1970, vol. 2. Springer-Verlag (1983)