Pebbling

Andreas Fellner[1] * and Bruno Woltzenlogel Paleo[2] **

[1] Université Paul Sabatier, Toulouse
joseph.boudou@matabio.net
[2] Vienna University of Technology
bruno@logic.at

Abstract.

1 Introduction

2 Propositional Resolution Calculus

A *literal* is a propositional variable or the negation of a propositional variable. The *complement* of a literal ℓ is denoted $\overline{\ell}$ (i.e. for any propositional variable p, $\overline{p} = \neg p$ and $\overline{\neg p} = p$). The set of all literals is denoted $\mathcal{L}$. A *clause* is a set of literals. $\perp$ denotes the *empty clause*.

Definition 1 (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 (i.e. $E \subset V \times \mathcal{L} \times V$ and $v_1 \xrightarrow{\ell} v_2$ denotes an edge from node v_1 to node v_2 labeled by ℓ), is a proof of a clause Γ iff it is inductively constructible according to the following cases:*

1. *If Γ is a clause, $\widehat{\Gamma}$ denotes some proof $\langle \{v\}, \varnothing, \Gamma \rangle$, where v is a new node.*
2. *If ψ_L is a proof $\langle V_L, E_L, \Gamma_L \rangle$ and ψ_R is a proof $\langle V_R, E_R, \Gamma_R \rangle$ and ℓ is a literal such that $\overline{\ell} \in \Gamma_L$ and $\ell \in \Gamma_R$, then $\psi_L \odot_\ell \psi_R$ denotes a proof $\langle V, E, \Gamma \rangle$ s.t.*

$$V = V_L \cup V_R \cup \{v\}$$
$$E = E_L \cup E_R \cup \left\{ v \xrightarrow{\overline{\ell}} \rho(\psi_L), v \xrightarrow{\ell} \rho(\psi_R) \right\}$$
$$\Gamma = \left(\Gamma_L \setminus \{\overline{\ell}\}\right) \cup \left(\Gamma_R \setminus \{\ell\}\right)$$

where v is a new 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. Contrary to the usual proof theoretic conventions but following the actual implementation of the data structures used by `LowerUnivalents`, edges are directed from children (resolvents) to their parents (premises). V_ψ, E_ψ and Γ_ψ denote, respectively, the nodes, edges and proved clause (conclusion) of ψ.

* Supported by the Google Summer of Code 2012 program.
** Supported by the Austrian Science Fund, project P24300.

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** $\varphi \in D$ *or* $\rho(\varphi)$ *has no premises* **then**
2   **return** φ ;
3 **else**
4   **let** φ_L*,* φ_R *and* ℓ be such that $\varphi = \varphi_L \odot_\ell \varphi_R$;
5   **let** $\varphi'_L =$ `delete`(φ_L,D) ;
6   **let** $\varphi'_R =$ `delete`(φ_R,D) ;
7   **if** $\varphi'_L \in D$ **then**
8   **return** φ'_R ;
9   **else if** $\varphi'_R \in D$ **then**
10   **return** φ'_L ;
11   **else if** $\overline{\ell} \notin \Gamma_{\varphi'_L}$ **then**
12   **return** φ'_L ;
13   **else if** $\ell \notin \Gamma_{\varphi'_R}$ **then**
14   **return** φ'_R ;
15   **else**
16   **return** $\varphi'_L \odot_\ell \varphi'_R$;

Algorithm 1: `delete`

Definition 2 (Active literals). *Given a proof ψ, the set of active literals $A_\psi(\varphi)$ of a subproof φ are the labels of edges coming into φ's root:*

$$A_\psi(\varphi) = \{\ell \mid \exists \varsigma \in V_\psi .\ \varsigma \xrightarrow{\ell} \rho(\varphi)\}$$

Two operations on proofs are used in this paper: the resolution operation $\odot_\ell$ introduced above and the deletion of a set of subproofs from a proof, denoted $\psi \setminus (\varphi_1 \ldots \varphi_n)$ where ψ is the whole proof and φ_i are the deleted subproofs. Algorithm 1 describes the deletion operation, with $\psi \setminus (\varphi_1 \ldots \varphi_n)$ being the result of `delete`(ψ,$\{\varphi_1, \ldots, \varphi_n\}$). Both the resolution and deletion operations are considered to be left associative.

The deletion algorithm is a minor variant of the Reconstruct-Proof algorithm presented in [**?**]. 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).

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 $\psi \setminus (A \cup B) = \psi \setminus (A) \setminus (B)$.*

Definition 3 (Valent literal). *In a proof ψ, a literal ℓ is* valent *for the subproof φ iff $\overline{\ell}$ belongs to the conclusion of $\psi \setminus (\varphi)$ but not to the conclusion of ψ.*

Proposition 2. *In a proof ψ, every valent literal of a subproof φ is an active literal of φ.*

Proof. Lines 2, 12, 14 and 16 from Algorithm 1 can not introduce a new literal in the conclusion of the subproof being processed. Let ℓ be a valent literal of φ in ψ. Because there is only one subproof to be deleted, $\overline{\ell}$ can only be introduced when processing a subproof φ' such that $\rho(\varphi') \xrightarrow{\ell} \rho(\varphi)$. □

Proposition 3. *Given a proof ψ and a set $D = \{\varphi_1 \ldots \varphi_n\}$ of ψ's subproofs, $\forall \ell \in \mathcal{L}$ s.t. ℓ is in the conclusion of $\psi \setminus (D)$ but not in ψ's conclusion, then $\exists i$ s.t. $\overline{\ell}$ is a valent literal of φ_i in ψ.*

3 Pebbling Game

4 Greedy Pebbling Algorithms

4.1 Last Child Heuristic

4.2 Node Distance Heuristic

4.3 Number of Children Heuristic

5 Experiments

`LowerUnivalents` and `LUnivRPI` have been implemented in the functional programming language Scala[1] as part of the Skeptik library[2]. `LowerUnivalents` has been implemented as a recursive `delete` improvement.

[1] http://www.scala-lang.org/
[2] https://github.com/Paradoxika/Skeptik

Input: a proof ψ

Output: a compressed proof ψ'

$\mathsf{Units} \leftarrow \varnothing$;

for *every subproof* φ *in a bottom-up traversal* **do**

  if φ *is a unit and has more than one child* **then**

    Enqueue φ in Units;

$\psi' \leftarrow$ `delete`(ψ,Units) ;

for *every unit* φ *in* Units **do**

  let $\{\ell\} = \Gamma_\varphi$;

  if $\bar{\ell} \in \Gamma_{\psi'}$ **then** $\psi' \leftarrow \psi' \odot_\ell \varphi$;

Algorithm 2: `LowerUnits`

Table 1: Number of proofs per benchmark category

Benchmark Category	Number of Proofs
QF_UF	3907
QF_IDL	475
QF_LIA	385
QF_UFIDL	156
QF_UFLIA	106
QF_RDL	30

The algorithms have been applied to 5 059 proofs produced by the SMT-solver veriT[3] on unsatisfiable benchmarks from the SMT-Lib[4]. The details on the number of proofs per SMT category are shown in Table 1. The proofs were translated into pure resolution proofs by considering every non-resolution inference as an axiom.

The experiment compared the following algorithms:

LU: the `LowerUnits` algorithm from [?];
LUniv: the `LowerUnivalents` algorithm;
RPILU: a non-sequential combination of `RPI` after `LowerUnits`;
RPILUniv: a non-sequential combination of `RPI` after `LowerUnivalents`;
LU.RPI: the sequential composition of `LowerUnits` after `RPI`;
LUnivRPI: the non-sequential combination of `LowerUnivalents` after `RPI` as described in Sect. 5;
RPI: the `RecyclePivotsWithIntersection` from [?];
Split: Cotton's `Split` algorithm ([?]);
RedRec: the `Reduce&Reconstruct` algorithm from [?];
Best RPILU/LU.RPI: which performs both `RPILU` and `LU.RPI` and chooses the smallest resulting compressed proof;

[3] http://www.verit-solver.org/
[4] http://www.smtlib.org/

Best RPILU/LUnivRPI: which performs `RPILU` and `LUnivRPI` and chooses the smallest resulting compressed proof.

For each of these algorithms, the time needed to compress the proof along with the number of nodes and the number of axioms (i.e. *unsat core* size) have been measured. Raw data of the experiment can be downloaded from the web[5].

The experiments were executed on the Vienna Scientific Cluster[6] VSC-2. Each algorithm was executed in a single core and had up to 16 GB of memory available. This amount of memory has been useful to compress the biggest proofs (with more than 10^6 nodes).

The overall results of the experiments are shown in Table **??**. The compression ratios in the second column are computed according to formula (3), in which ψ ranges over all the proofs in the benchmark and ψ' ranges over the corresponding compressed proofs.

$$1 - \frac{\sum |V_{\psi'}|}{\sum |V_\psi|} \tag{3}$$

The unsat core compression ratios are computed in the same way, but using the number of axioms instead of the number of nodes. The speeds on the fourth column are computed according to formula (**??**) in which d_ψ is the duration in milliseconds of ψ's compression by a given algorithm.

$$\frac{\sum |V_\psi|}{\sum d_\psi} \tag{4}$$

For the `Split` and `RedRec` algorithms, which must be repeated, a timeout has been fixed so that the speed is about 3 nodes per millisecond.

Figure **??** shows the comparison of `LowerUnits` with `LowerUnivalents`. Subfigures (a) and (b) are scatter plots where each dot represents a single benchmark proof. Subfigure (c) is a histogram showing, in the vertical axis, the proportion of proofs having *(normalized) compression ratio difference* within the intervals showed in the horizontal axis. This difference is computed using formula (**??**) with v_{LU} and v_{LUniv} being the compression ratios obtained respectively by `LowerUnits` and `LowerUnivalents`.

$$\frac{v_{\text{LU}} - v_{\text{LUniv}}}{\frac{v_{\text{LU}} + v_{\text{LUniv}}}{2}} \tag{5}$$

The number of proofs for which $v_{\text{LU}} = v_{\text{LUniv}}$ is not displayed in the histogram. The *(normalized) duration differences* in subfigure (d) are computed using the same formula (**??**) but with v_{LU} and v_{LUniv} being the time taken to compress the proof by `LowerUnits` and `LowerUnivalents` respectively.

As expected, `LowerUnivalents` always compresses more than `LowerUnits` (subfigure (a)) at the expense of a longer computation (subfigure (d)). And even

[5] http://www.matabio.net/skeptik/LUniv/experiments/

[6] http://vsc.ac.at/

Table 2: Total compression ratios

Algorithm	Compression	Unsat Core Compression	Speed
LU	7.5 %	0.0 %	22.4 n/ms
LUniv	8.0 %	0.8 %	20.4 n/ms
RPILU	22.0 %	3.6 %	7.4 n/ms
RPILUniv	22.1 %	3.6 %	6.5 n/ms
LU.RPI	21.7 %	3.1 %	15.1 n/ms
LUnivRPI	22.0 %	3.6 %	17.8 n/ms
RPI	17.8 %	3.1 %	31.3 n/ms
Split	21.0 %	0.8 %	2.9 n/ms
RedRec	26.4 %	0.4 %	2.9 n/ms
Best RPILU/LU.RPI	22.0 %	3.7 %	5.0 n/ms
Best RPILU/LUnivRPI	22.2 %	3.7 %	5.2 n/ms

if the compression gain is low on average (as noticeable in Table **??**), subfigure (a) shows that `LowerUnivalents` compresses some proofs significantly more than `LowerUnits`.

It has to be noticed that veriT already does its best to produce compact proofs. In particular, a forward subsumption algorithm is applied, which results in proofs not having two different subproofs with the same conclusion. This results in `LowerUnits` being unable to reduce unsat core. But as `LowerUnivalents` lowers non-unit subproofs and performs some partial regularization, it achieves some unsat core reduction, as noticeable in subfigure (b).

The comparison of the sequential `LU.RPI` with the non-sequential `LUnivRPI` shown in Fig. **??** outlines the ability of `LowerUnivalents` to be efficiently combined with other algorithms. Not only compression ratios are improved but `LUnivRPI` is faster than the sequential composition for more than 80 % of the proofs.

6 Conclusions and Future Work

Acknowledgments: