Recycling Algebraic Proof Certificates

Daniela Kaufmann[1,*], Clemens Hofstadler[2,*]

[1]TU Wien, Vienna, Austria
[2]Johannes Kepler University, Linz, Austria

Abstract

Proof certificates can be used to validate the correctness of algebraic derivations. However, in practice, we frequently observed that the exact same proof steps are repeated for different sets of variables, which leads to unnecessarily large proofs. To overcome this issue we extend the existing Practical Algebraic Calculus with linear combinations (LPAC) with two new proof rules that allow us to capture and reuse parts of the proof to derive a more condensed proof certificate. We integrate these rules into the proof checker Pacheck 2.0. Our experimental results demonstrate that the proposed extension helps to reduce both proof size and verification time.

Keywords

Proof Logging, Algebraic Calculus, Recycling Proofs

Setting: *Let X be a set of variables, $\mathbb{K}$ a field. Let $G \subseteq \mathbb{K}[X]$, $f \in \mathbb{K}[X]$. The goal is to derive a proof certificate validating that $f \in \langle G \rangle$.*

Although our proposed idea of proof recycling works in this general setting, our main focus is on applications where all variables X are Boolean. Recall that this can be modeled by adding, for each variable $x \in X$, the Boolean value polynomial $x(x-1) = x^2 - x$ to G. Therefore, we assume that $B(X) = \{x^2 - x \mid x \in X\} \subseteq G$. Our proof rules (and the existing rules of LPAC; see Figure 1) are instantiated for this particular use case, but they can be easily adapted to the general setting.

Two proof systems are commonly studied for our application, *polynomial calculus* (PC) [1], and *Nullstellensatz* (NSS) [2]. NSS proofs represent f as a linear combination of the polynomials in G. However, if the expansion of the linear combination is not equal to f, it is unclear how to locate the error in the proof. This limits the usefulness of a NSS proof for debugging. In contrast, proofs in PC capture whether f can be derived from G using proof steps from ideal theory. However, PC as originally defined [1] is not suitable for effective proof checking, because information of the origin of the proof steps is missing.

The Practical Algebraic Calculus (PAC) [3] addressed this by explicitly encoding each polynomial operation, enabling stepwise verification and hence error localization. In a later work [4], PC was combined with NSS in the proof calculus LPAC (PAC with linear combinations) to yield, shorter, yet traceable proofs.

However, in some applications, the same proof steps are repeated over and over again for different variable instances. For example, in *arithmetic circuit verification* it is required to reason over structurally equivalent building blocks, such as full- and half-adders, of the circuit. Until now (e.g., in [5]), all the proof steps for these building blocks were explicitly recomputed from scratch, despite differing only in variable names, rather than being cached and reused.

We now propose to extend LPAC with two new proof rules PatternNew and PatternApply, which allow us to save a pattern in the proof that can be reinstated with different polynomials to recycle parts of the proof.

SC² 2025: Satisfiability Checking and Symbolic Computation
*Corresponding author.
daniela.kaufmann@tuwien.ac.at (D. Kaufmann); clemens.hofstadler@jku.at (C. Hofstadler)
https://danielakaufmann.at/ (D. Kaufmann); https://clemenshofstadler.com/ (C. Hofstadler)
0000-0002-5645-0292 (D. Kaufmann); 0000-0002-3025-0604 (C. Hofstadler)

Axiom (i, p)

$$\frac{(X,P) \quad P(i)=\bot \quad p\in\mathbb{K}[X]}{(X\cup \mathrm{Var}(p), P(i\mapsto p))}$$

Deletion (i)

$$\frac{(X,P)}{(X,P(i\mapsto \bot))}$$

LinComb $(i, (j_1, \ldots, j_n), (q_1, \ldots, q_n), p)$

$$\frac{(X,P) \quad P(j_1)\neq\bot,\ldots,P(j_n)\neq\bot, P(i)=\bot \quad p,q_1,\ldots,q_n\in\mathbb{K}[X] \quad p=q_1\cdot P(j_1)+\cdots+q_n\cdot P(j_n) \bmod \langle B(X)\rangle}{(X,P(i\mapsto p))}$$

Ext (i, v, q)

$$\frac{(X,P) \quad P(i)=\bot \quad v\notin X \quad q\in\mathbb{K}[X] \quad q^2-q\in\langle B(X)\rangle}{(X\cup\{v\},P(i\mapsto -v+q))}$$

Figure 1: Existing Proof Rules of LPAC

1. LPAC

An LPAC proof certificate can be used to validate that a polynomial $f \in \mathbb{K}[X]$ can be derived from a given set of polynomials $G \subseteq \mathbb{K}[X]$ (= the proof axioms) using polynomial addition and multiplication. The semantics of LPAC can be seen in Figure 1. A more detailed explanation including correctness arguments is given in [4].

Let X denote a set of Boolean variables and let P be a sequence of polynomials, which can be accessed via indices. An LPAC proof is a sequence of states (X, P), with the initial state $(\{\}, [])$. At any state, P will only contain polynomials that are either axioms or can be derived from polynomials already contained in P using the inference rules of the calculus.

Using the Axiom rule, we add polynomials p to P. The Deletion rule allows us to remove polynomials from P when they are no longer needed, which helps to save memory during checking. The LinComb rule can be used to add polynomials to P that can be derived as a linear combination of polynomials already stored in P. Using the Ext rule we can add polynomials to the proof, which can be used to introduce placeholder variables v for Boolean expressions q.

Proof Checking: LPAC proofs can be checked using, e.g., Pacheck 2.0 [4]. Proof checking is applied on the fly. For instance, after parsing a LinComb proof step, Pacheck 2.0 checks whether all input polynomials exist, calculates the linear combination of the given input, and compares it to the given conclusion polynomial p of the LinComb rule. Additionally, it is checked whether one of the conclusion polynomials of a LinComb step is equal to the target f at any point in the proof.

2. Recycling Proof Steps

In the following, we introduce the new proof rules PatternNew and PatternApply to add and instantiate proof patterns. In principle, these rules could be generalized and used to extend any polynomial proof system, including those that do not work over Boolean variables. Here, we focus on extending LPAC and instantiate the new rules to work for this particular proof system. For that, we also extend an LPAC proof from (X, P) to (X, P, C), where C denotes a sequence of patterns. Initially $C = []$. Note that C cannot be changed by any of the rules in Figure 1.

PatternNew: A pattern is identified with a unique index i that will be used in the PatternApply rule to identify the pattern. A pattern has three components: *sequence of inputs* I, *sequence of proof steps* S, and *set of outputs* O.

The sequence of inputs represents the axioms of the pattern. The proof steps S form a self-contained proof fragment. In "Correct-LPAC-Proof(I, S)" it is checked whether the proof steps in S can be derived

PatternNew (i, I, S, O)

$$\frac{(X,P,C) \quad C(i)=\bot \quad \text{Correct-LPAC-Proof}(I,S) \quad O\subseteq\text{Conclusions}(S) \quad V_{\text{ext}}=\text{ExtensionVar}(O)}{(X,P,C\cup(i,I,O,V_{\text{ext}}))}$$

PatternApply $(i, X_{\text{ext}}, \varphi, (j_1, \ldots, j_m), \{(k_1, p_1), \ldots, (k_n, p_n)\})$

$$\frac{\begin{array}{c}(X,P,C) \quad C(i)\neq\bot \quad X_{\text{ext}}\not\subseteq X \quad \forall v\in C(i).V_{\text{ext}} : \varphi(v)\in X_{\text{ext}} \\ \forall v\in \text{Var}(C(i)) : \varphi(v)^2-\varphi(v)\in\langle B(X\cup X_{\text{ext}})\rangle \quad P(j_1)\neq\bot,\ldots,P(j_m)\neq\bot \quad C(i).I=_\varphi\{P(j_1),\ldots,P(j_m)\} \\ P(k_1)=\bot,\ldots,P(k_n)=\bot \quad p_1,\ldots,p_n\in\mathbb{K}[X\cup X_{\text{ext}}] \quad C(i).O=_\varphi\{p_1,\ldots,p_n\}\end{array}}{(X\cup X_{\text{ext}},P(k_1\mapsto p_1,\ldots,k_n\mapsto p_n),C)}$$

Figure 2: New Proof Rules of LPAC

by the LPAC inference rules using the axioms given in I. The set of outputs O contains those conclusion polynomials of the proof steps in S, which will be stored as output polynomials of the pattern.

We explicitly extract any extension variables V_{ext} from the outputs O of the pattern that have been added in the proof steps S. When applying the pattern, these variables have to be instantiated as fresh variables. On the other hand, it is not necessary to store any internal extension variables of S that are not contained in any polynomial in O, since they are only used internally to derive the outputs O. They are not needed for the application of the pattern.

After a pattern is checked for correctness, we only store I, O, and V_{ext} in C. The proof steps S are discarded. On a practical note, since a pattern can be seen as a standalone LPAC proof, we are free to (re-)use any variables and indices.

PatternApply: The index i refers to a pattern in C. The set X_{ext} contains all extension variables that will be added to the LPAC proof through the application of the pattern. These variables are not allowed to be contained in the set of variables X of the current proof state (X, P, C).

The mapping φ connects the pattern $C(i)$ to the LPAC proof P by mapping all variables (including those in V_{ext}) in $C(i)$ to polynomials over the variables $X \cup X_{\text{ext}}$. This mapping has to satisfy two restrictions: First, it has to map the extension variables used in $C(i)$ to the new extension variables in X_{ext}, that is, for all $v \in V_{\text{ext}}$, we must have $\varphi(v) \in X_{\text{ext}}$. Moreover, φ has to comply with the Boolean axioms in the following way: for each variable v in $C(i)$, its image under φ must satisfy the condition $\varphi(v)^2 - \varphi(v) \in \langle B(X \cup X_{\text{ext}})\rangle$. This restriction comes from the fact that all variables are assumed Boolean in an LPAC proof and the Boolean axioms are used implicitly (see, for instance, the LinComb rule in Figure 1). In general, when a proof system treats certain relations implicitly, the mapping φ has to be consistent with those relations. In contrast, for a proof system that does not use any implicit relations, φ can map the variables from $C(i)$ to arbitrary polynomials.

The tuple $(j_1, \ldots, j_m)$ contains the indices to those polynomials of P that are used as inputs to the pattern. We verify that these polynomials match the pattern's inputs under the mapping φ, denoted by $=_\varphi$ in Figure 2. We do an analogous check for all the output polynomials p_i in the set $\{(k_1, p_1), \ldots, (k_n, p_n)\}$. If the pattern is correctly applied, we store the polynomials p_i as correctly derived polynomials in P, i.e., we set $P(k_i) \mapsto p_i$.

Proof Checking: For the PatternNew rule, we have to apply the following checks during proof checking:

(i) do we already have a pattern with index i;
(ii) do I and S form a correct standalone LPAC proof;
(iii) are all polynomials in O conclusion polynomials of proof steps in I or S.

For the PatternApply rule, we have to apply the following checks:

(i) does pattern $C(i)$ exist;
(ii) are all variables in X_{ext} fresh variables that are not contained in X;
(iii) does the mapping φ map the extension variables of the pattern to the fresh variables in X_{ext};
(iv) does the mapping φ comply with the Boolean axioms $B(X \cup X_{\text{ext}})$;
(v) do the input polynomials I correspond to the input polynomials of $C(i)$ after the mapping φ is applied;
(vi) do the output polynomials O correspond to the output polynomials of $C(i)$ after φ is applied.

3. Examples

We illustrate the new proof rules on two examples. In the first example we show how patterns can help to reduce the size of the proof. In the second example we show how to treat extension variables in proof patterns.

Example 1. *Let $G = \{x+2y-2, -y-z+1, a+2b-2, -b-z+1, a-x+1\}$. We use the PatternNew and PatternApply rules to derive $1 \in \langle G \rangle$ in an LPAC proof. We use p_i as indices for the proof steps inside the pattern and l_i as indices for the proof steps outside the pattern.*

1 Axiom$(l_1,\ x+2y-2)$
2 Axiom$(l_2,\ -y-z+1)$
3 Axiom$(l_3,\ a+2b-2)$
4 Axiom$(l_4,\ -b-z+1)$
5 Axiom$(l_5,\ a-x+1)$
6 PatternNew$(1,\ [(p_1\ \ v_1-2v_2),\ (p_2\ \ v_2-v_3)],$
$\quad$ $[$LinComb$(p_3,\ (p_1,p_2),\ (1,2),\ v_1-2v_3)],\ \{p_3\})$
7 PatternApply$(1,\ \{\},\ [v_1\mapsto x,\ v_2\mapsto 1-y,\ v_3\mapsto z],\ (l_1,l_2),\ \{(l_6,\ x-2z)\})$
8 PatternApply$(1,\ \{\},\ [v_1\mapsto a,\ v_2\mapsto 1-b,\ v_3\mapsto z],\ (l_3,l_4),\ \{(l_7,\ a-2z)\})$
9 LinComb$(l_8,\ (l_5,l_6,l_7),\ (1,1,-1),\ 1)$

The pattern created in this proof captures the derivation of a specific linear combination of two polynomials: Starting from $p_1 = v_1 - 2v_2$ and $p_2 = v_2 - v_3$, defined over variables v_1, v_2, v_3, we can derive the linear combination $p_3 = p_1 + 2p_2 = v_1 - 2v_3$. We then apply this pattern in two different ways.

First, we instantiate v_1, v_2, v_3 by x, $1-y$, and z, respectively. Note that this substitution complies with the Boolean axioms. For instance, $\varphi(v_2)^2 - \varphi(v_2) = (1-y)^2 - (1-y) = y^2 - y \in \langle B(\{a,b,x,y,z\})\rangle$. Under this substitution, p_1 and p_2 reduce to the axioms l_1 and l_2, and the output polynomial p_3 becomes $x-2z$. In the second application, we substitute v_1, v_2, v_3 with a, $1-b$, and z, respectively. Then p_1 and p_2 reduce to the axioms l_3 and l_4, and the output polynomial p_3 becomes $a-2z$.

By using the pattern, we have to compute the linear combination $p_1 + 2p_2$ only once and can then reuse it through instantiation in each application. In contrast, a classical LPAC proof would require computing each linear combination $l_1 + 2l_2$ and $l_3 + 2l_4$ separately.

Example 2. *In this example, we algebraically mimic the Boolean resolution rule. That is, from $\neg x \vee \neg y$ and $y \vee z$, we derive $\neg x \vee z$. Algebraically, the two axioms can be encoded as $G = \{xy, yz - y - z + 1\}$. The goal is to derive $x\bar{z} \in \langle G \cup \{\bar{z} - (1-z)\}\rangle$, with $\bar{z}$ being a new variable representing $\bar{z} = 1 - z$.*

1 Axiom$(l_1,\ xy)$
2 Axiom$(l_2,\ yz-y-z+1)$
3 PatternNew$(1,\ [(p_1\ \ v_1v_2),\ (p_2\ \ v_2v_3-v_2-v_3+1)],$
$\quad$ $[$Ext$(p_3,\ (w_3),\ (1-v_3)),$
$\quad$ LinComb$(p_4,\ (p_1,p_2,p_3),\ (w_3,v_1,v_1v_2-v_1),\ v_1w_3)],\ \{p_3,p_4\})$
4 PatternApply$(1,\ \{\bar{z}\},\ [v_1\mapsto x,\ v_2\mapsto y,\ v_3\mapsto z,\ w_3\mapsto \bar{z}],\ (l_1,l_2),$
$\quad$ $\{(l_3,\ -\bar{z}+1-z),\ (l_4,\ x\bar{z})\})$

Name	Axioms (10^3)	LPAC without Patterns Steps (10^3)	File (MB)	Mem (MB)	Time (s)	LPAC with Patterns Steps (10^3)	#	Apply	max $\|S\|$	File (MB)	Mem (MB)	Time (s)
abc	64	196	454	982	10.17	48	9	4032	46	439	913	9.61
abc-rsn	64	196	454	982	10.35	48	12	4033	65	439	913	9.94
abc-cmp	64	196	454	982	10.45	48	12	4033	46	439	913	9.90
sp-ar-rc	96	270	676	1377	15.86	107	18	6509	83	663	1315	14.87
sp-bd-rc	98	284	1130	2098	32.33	111	42	6520	87	1117	2038	29.45
sp-dt-rc	96	292	1789	3112	56.63	108	58	7082	84	1776	3056	55.75
sp-os-rc	99	289	1314	2380	38.62	112	52	6542	87	1300	2321	38.48
sp-ar-cl	108	2043	959	1867	25.98	1820	79	3999	421	924	1749	24.37

Table 1
Benchmark Results

4. Experimental Evaluation

We have extended PACHECK 2.0 with the PATTERNNEW and PATTERNAPPLY rules and show the effect of adding those rules on a small set of benchmarks that originate from multiplier verification. In that setting, we prove that the circuit specification is contained in the ideal generated by the circuit encoding.

Proof Generation: In our recent work on circuit verification [6], we extract linear relations from subcircuits, which are then used for the ideal membership test. These subcircuits are defined syntactically. Since circuits typically consist of repeated usage of the same building blocks, such as full- and half-adders, we often identify the same subcircuits multiple times. To avoid redundant computations, we started caching these subcircuits.

To detect isomorphic subcircuits, we map all gate variables of a subcircuit to a standardized set of variables and check for syntactic equivalence. Using this approach, we are able to cache over 80% of the identified subcircuits. For each computation within a subcircuit, we store a corresponding proof pattern. Whenever a cached pattern is retrieved, we instantiate and apply the corresponding proof pattern to avoid the need for re-computation.

In the following, we use a subset of the benchmarks from [6] to demonstrate the effect of using proof patterns. All benchmarks are drawn from the experimental evaluations in [7, 6], and are available from the artifact [8] of the paper [7].

Table 1 summarizes our results in two blocks: one for the original version of LPAC ("LPAC without Patterns"), the second one with our two new proof rules ("LPAC with Patterns"). None of the proofs contains EXT or DELETION rules. The second column lists the number of axioms, which is equal for both approaches. The next four columns represent LPAC proofs without patterns. We list the number of linear combination proof steps ("Steps"), the size of the proof file in MB ("File"), the used memory in MB ("Mem"), and the time needed to check the proof ("Time") in seconds. The next seven columns show the results when our two new proof rules are added. We list the number of PATTERNNEW steps ("#"), the total number of PATTERNAPPLY steps ("Apply"), and the maximal number of steps in a pattern ("max $|S|$").

It can be seen that we can significantly reduce the number of proof steps when using patterns and we always improve on the file size of the proof and the memory and time used to check the proof.

Further improving the efficiency of proof checking is part of future work. In particular, we want to investigate the effects of recently developed techniques for finding short proofs of ideal membership [9].

Acknowledgments

This research was supported by Austrian Science Fund (FWF) [10.55776/ESP666] and by the LIT AI Lab funded by the state of Upper Austria.

References

[1] M. Clegg, J. Edmonds, R. Impagliazzo, Using the Groebner Basis Algorithm to Find Proofs of Unsatisfiability, in: STOC 1996, ACM, 1996, pp. 174–183.

[2] P. Beame, R. Impagliazzo, J. Krajícek, T. Pitassi, P. Pudlák, Lower Bounds on Hilbert's Nullstellensatz and Propositional Proofs, in: Proc. London Math. Society, volume s3-73, 1996, pp. 1–26.

[3] D. Ritirc, A. Biere, M. Kauers, A Practical Polynomial Calculus for Arithmetic Circuit Verification, in: SC2 2018, CEUR-WS, 2018, pp. 61–76.

[4] D. Kaufmann, M. Fleury, A. Biere, M. Kauers, Practical Algebraic Calculus and Nullstellensatz with the Checkers Pacheck and Pastèque and Nuss-Checker, Formal Methods Syst. Des. 64 (2022) 73–107. doi:10.1007/s10703-022-00391-x.

[5] D. Kaufmann, A. Biere, AMulet 2.0 for Verifying Multiplier Circuits, in: TACAS 2021, volume 12652 of *LNCS*, Springer, 2021, pp. 357–364. doi:10.1007/978-3-030-72013-1_19.

[6] C. Hofstadler, D. Kaufmann, Guess and Prove: A Hybrid Approach to Linear Polynomial Recovery in Circuit Verification, in: CP 2025, LIPIcs, 2025. To appear.

[7] D. Kaufmann, J. Berthomieu, Extracting Linear Relations from Gröbner Bases for Formal Verification of And-Inverter Graphs, in: TACAS 2025, volume 15696 of *LNCS*, Springer, 2025, pp. 355–374. doi:10.1007/978-3-031-90643-5_19.

[8] D. Kaufmann, J. Berthomieu, MultiLinG – Extracting Linear Relations from Gröbner Bases for Formal Verification of And- Inverter Graphs (Artifact) , 2025. doi:10.5281/zenodo.14610365.

[9] C. Hofstadler, T. Verron, Short proofs of ideal membership, J. Symbolic Comput. 125 (2024) 102325. doi:10.1016/j.jsc.2024.102325.