THE PROOF CHECKERS PACHECK AND PASTÈQUE FOR THE PRACTICAL ALGEBRAIC CALCULUS

Daniela Kaufmann, Mathias Fleury and Armin Biere
Johannes Kepler University
Linz, Austria

FMCAD 2020

September 24, 2020
Online

JOHANNES KEPLER
UNIVERSITY LINZ

Formal Verification using Computer Algebra

Renewed interest in recent years

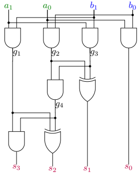
- A. Mahzoon, D. Große, and R. Drechsler. RevSCA: Using reverse engineering to bring light into backward rewriting for big and dirty multipliers. In DAC 2019.
- M. J. Ciesielski, T. Su, A. Yasin, and C. Yu. Understanding algebraic rewriting for arithmetic circuit verification: a bit-flow model. In IEEE TCAD 2019.
- A. Mahzoon, D. Große, C. Scholl, and R. Drechsler. Towards formal verification of optimized and industrial multipliers. In DATE 2020.

Algebraic reasoning in combination with SAT solving

- C. Bright, I. Kotsireas, and V. Ganesh. Applying computer algebra systems and SAT solvers to the Williamson conjecture. In Journal of Symbolic Computation, 2018.
- M. J. H. Heule. Computing small unit-distance graphs with chromatic number 5. CoRR, vol. abs/1805.12181, 2018.
- M. J. H. Heule, M. Kauers, and M. Seidl. Local search for fast matrix multiplication. In SAT 2019.
- D. Kaufmann, A. Biere, and M. Kauers. Verifying large multipliers by combining SAT and computer algebra. In FMCAD 2019.

Basic Idea of Algebraic Approach

System



Polynomials

$$
\begin{aligned}
B = \{ & \\
& x - a_0 * b_0, \\
& y - a_1 * b_1, \\
& s_0 - x * y, \\
& \dots \\
\} &
\end{aligned}
$$

Specification

$$\sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right)$$

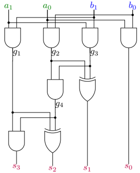
Implication

$= 0$ ✓

$\neq 0$ ✗

Basic Idea of Algebraic Approach

Multiplier



Polynomials

$$
\begin{aligned}
B = \{ \\
&x - a_0 * b_0, \\
&y - a_1 * b_1, \\
&s_0 - x * y, \\
&\dots \\
\}
\end{aligned}
$$

Specification

$$\sum_{i=0}^{2n-1} 2^i s_i - \Big(\sum_{i=0}^{n-1} 2^i a_i\Big)\Big(\sum_{i=0}^{n-1} 2^i b_i\Big)$$

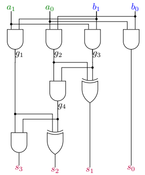
Implication

$= 0$ ✓

$\neq 0$ ✗

Basic Idea of Algebraic Approach

Multiplier



Polynomials

$$
\begin{aligned}
B = \{ & \\
& x - a_0 * b_0, \\
& y - a_1 * b_1, \\
& s_0 - x * y, \\
& \dots \\
\} &
\end{aligned}
$$

Specification

$$\sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right)$$

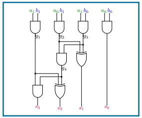
Ideal Membership

$= 0$ ✓

$\neq 0$ ✗

Motivation

Multiplier



Polynomials

$$B = \{ \; x - a_0 * b_0, \; y - a_1 * b_1, \; s_0 - x * y, \; \ldots \; \}$$

Specification

$$\sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right)$$

Ideal Membership

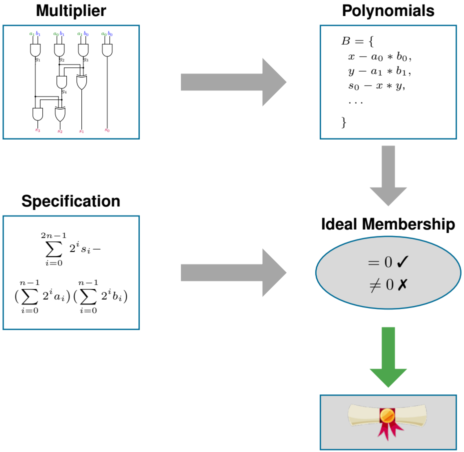
$= 0$ ✓

$\neq 0$ ✗

Correct?

Problem: Verification might not be error free

Motivation



Problem: Verification might not be error free

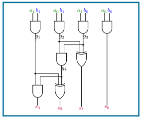
Goal: Validate result of verification process

- Generate machine-checkable proofs
- Check by independent proof checkers

$\Rightarrow$ SC'2 2018: Practical Algebraic Calculus (PAC) based on polynomial calculus

Motivation

Multiplier



Polynomials

$$B = \{\ x - a_0 * b_0,\ y - a_1 * b_1,\ s_0 - x * y,\ \dots\ \}$$

Specification

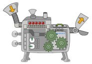
$$\sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right)$$

Ideal Membership

$= 0$ ✓

$\neq 0$ ✗

Proof Checker



Problem: Verification might not be error free

Goal: Validate result of verification process

- Generate machine-checkable proofs
- Check by independent proof checkers

$\Rightarrow$ SC'2 2018: Practical Algebraic Calculus (PAC) based on polynomial calculus

Contribution:

- Extending PAC:
  indexing, deletion and extension rules
- Proof checker PACHECK and PASTÈQUE

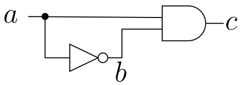
Ideal

Ideal. A nonempty subset $I \subseteq \mathbb{Z}[X]$ is called an ideal if

$$\forall\, p, q \in I : p + q \in I \qquad \text{and} \qquad \forall\, p \in \mathbb{Z}[X]\; \forall\, q \in I : pq \in I$$

Ideal membership problem. Given a polynomial $f \in \mathbb{Z}[X]$ and a (finite) set of polynomials $P \subseteq \mathbb{Z}[X]$, decide whether $f \in \langle P \rangle$, where $\langle P \rangle$ is the smallest ideal containing all elements of P, also known as the ideal *generated by* P.

Practical Algebraic Calculus - SC'2 2018




```
   P = -b+1-a,
       -c+a*b,
       -a^2+a
Spec = c
```

$b = \neg a$

$c = a \wedge b = a \wedge \neg a$

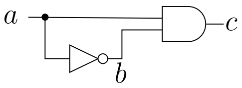
$a = \bot \vee a = \top$

$c = \bot$

```
* : -b+1-a,       a,       -a*b+a-a^2;
* : -a^2+a,       -1,      a^2-a;
+ : -a*b+a-a^2,   a^2-a,   -a*b;
+ : -a*b,         -c+a*b,  -c;
* : -c,           -1,      c;
```

$\forall\, p, q \in I : p + q \in I$ and $\forall\, p \in \mathbb{Z}[X]\ \forall\, q \in I : pq \in I$

Practical Algebraic Calculus - SC'2 2018




```
P    =  -b+1-a,
        -c+a*b,
        -a^2+a
Spec =  c
```

$b = \neg a$

$c = a \wedge b = a \wedge \neg a$

$a = \bot \vee a = \top$

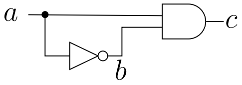
$c = \bot$

```
* : -b+1-a,        a,        -a*b+a-a^2;
* : -a^2+a,        -1,       a^2-a;
+ : -a*b+a-a^2,    a^2-a,    -a*b;
+ : -a*b,          -c+a*b,   -c;
* : -c,            -1,       c;
```

Can be checked by our older proof checker PacTrim.

1. Boolean Variables

Handle Boolean-value constraints implicitly to reduce number of proof steps.

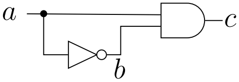



```
   P =  -b+1-a,
        -c+a*b,
        -a^2+a
Spec =  c
```

```
* : -b+1-a,       a,        -a*b;
+ : -a*b,         -c+a*b,   -c;
* : -c,           -1,       c;
```

2. Indices

Introduce indices to reduce proof size.

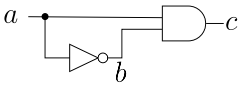



```
  P  =  1 -b+1-a;
        2 -c+a*b;
Spec =  c
```

```
3   * 1,   a,   -a*b;
4   + 3,   2,   -c;
5   * 4,  -1,    c;
```

3. Deletion Rule

Introduce a deletion rule to reduce the memory usage of the proof checker.




```
     P =   1 -b+1-a;
           2 -c+a*b;
  Spec =   c
```

```
3   * 1,   a,   -a*b;
1   d;
4   + 3,   2,   -c;
2   d;
3   d;
5   * 4,  -1,   c;
```

4. Extension Rule

The extension rule allows to add model preserving polynomials to the constraint set.

$$\frac{\overline{x} \vee \overline{y} \qquad y \vee z}{\overline{x} \vee z}$$

4. Extension Rule

The extension rule allows to add model preserving polynomials to the constraint set.

$$\frac{xy \qquad (1-y)(1-z)}{x(1-z)}$$

4. Extension Rule

The extension rule allows to add model preserving polynomials to the constraint set.

$$\frac{xy \qquad (1-y)(1-z)}{x(1-z)}$$

```
   P =   1 x*y;
         2 y*z-y-z+1;
Spec =   -x*z+x
```

```
3  =  f, -z+1;
4  *  3,  y-1, -f*y+f-y*z+y+z-1;
5  +  2,    4, -f*y+f;
```

Ext(i, v, p) $\quad (X, P) \Rightarrow (X \cup \{v\}, P(i \mapsto -v + p))$
provided that $P(i) = \perp$ and $v \notin X$ and $p \in \mathbb{Z}[X]/\langle B(X)\rangle$,
and $p^2 - p \equiv 0 \mod \langle B(X)\rangle$.

4. Extension Rule

The extension rule allows to add model preserving polynomials to the constraint set.

$$\frac{xy \qquad (1-y)(1-z)}{x(1-z)}$$

```
   P =   1 x*y;
         2 y*z-y-z+1;
Spec =   -x*z+x
```

```
 3  =  f, -z+1;
 4  *  3,  y-1, -f*y+f-y*z+y+z-1;
 5  +  2,    4, -f*y+f;
 6  *  1,    f, f*x*y;
 7  *  5,    x, -f*x*y+f*x;
 8  +  6,    7, f*x;
 9  *  3,    x, -f*x-x*z+x;
10  +  8,    9, -x*z+x;
```

PACHECK

- 1700 lines of C code
- supports new and old PAC format

- PACHECK reads three input files `<input>`, `<proof>`, and `<target>`.
- Verifies that the polynomial in `<target>` is contained in the ideal generated by the polynomials in `<input>` using the rules provided in `<proof>`.

PACHECK

- 1700 lines of C code
- supports new and old PAC format

- PACHECK reads three input files `<input>`, `<proof>`, and `<target>`.
- Verifies that the polynomial in `<target>` is contained in the ideal generated by the polynomials in `<input>` using the rules provided in `<proof>`.

- Polynomial arithmetic is implemented from scratch.
- A polynomial is represented as a linked list of monomials:
  - Coefficients are represented using the GMP library.
  - Terms are ordered linked lists of variables.

Variable ordering

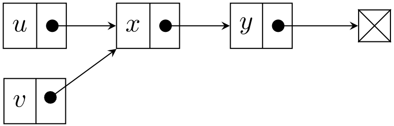
- The order of the variables has an enormous effect on the memory usage:
  - Terms are internally shared

Variable ordering

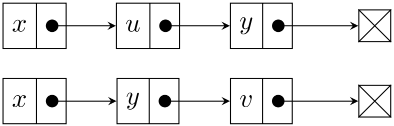
- The order of the variables has an enormous effect on the memory usage:
  - Terms are internally shared

Assume we want to represent the terms uxy and vxy.

$v > u > x > y$



$x > u > y > v$



Variable ordering

- The order of the variables has an enormous effect on the memory usage:
  - Terms are internally shared
- Example: 7 mio. rules, 50% memory increase using a different ordering

Implemented Orderings:

- Default: Variables are ordered using `strcmp`.
- Alternative: Same variable ordering as used in input files.
- Both orderings can also be reversed.

Pastèque

Theorem Prover Isabelle/HOL

Refinement Approach, relying on Isabelle's Refinement Framework

- abstract specification on ideals: specification in ideal
- final step: executable checker

Isabelle's Archive of Formal Proofs 8 000 lines of code

Refinement

$$\boxed{\text{no error} \implies \text{spec} \in \langle P \rangle \wedge \langle P \rangle\big|_X \subseteq \langle P_0 \rangle\big|_X}$$

Refinement

$\boxed{\text{no error} \implies \text{spec} \in \langle P \rangle \wedge \langle P \rangle\big|_X \subseteq \langle P_0 \rangle\big|_X}$

Entering nondeterminism monad

$\boxed{\text{check rule by rule for the side conditions}}$

Refinement

no error $\implies$ spec $\in \langle P \rangle \wedge \langle P \rangle\big|_X \subseteq \langle P_0 \rangle\big|_X$

Entering nondeterminism monad

check rule by rule for the side conditions

Executable polynomials

Use multisets for polynomials

Refinement

no error $\implies$ spec $\in \langle P \rangle \wedge \langle P \rangle\big|_X \subseteq \langle P_0 \rangle\big|_X$

↓ Entering nondeterminism monad

check rule by rule for the side conditions

↓ Executable polynomials

Use multisets for polynomials

↓ Specify all operations

Use multisets for lists

Refinement

no error $\implies$ spec $\in \langle P \rangle \wedge \langle P \rangle\big|_X \subseteq \langle P_0 \rangle\big|_X$

Entering nondeterminism monad

check rule by rule for the side conditions

Executable polynomials

Use multisets for polynomials

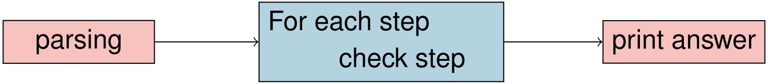
Specify all operations

Use multisets for lists

Automatic efficient imperative code

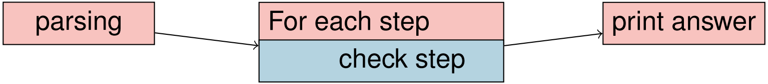
Imperative code

Refinement



- Translation to Standard ML is trusted
- Parser and plumbing (including answer printing) also trusted

Parsing vs Loop



Alternative, `uloop` variant:

- same functions to check the steps
- but: hand-written (trusted) version of the loop that iterates through the steps

Advantage: more memory efficient

Tool Demonstration

$$\frac{xy \qquad (1-y)(1-z)}{x(1-z)}$$

```
P =    1 x*y;
       2 y*z-y-z+1;
Spec = -x*z+x
```

```
3  =  f, -z+1;
4  *  3,  y-1, -f*y+f-y*z+y+z-1;
5  +  2,    4, -f*y+f;
2  d;
4  d;
6  *  1,    f, f*x*y;
1  d;
7  *  5,    x, -f*x*y+f*x;
5  d;
8  +  6,    7, f*x;
6  d;
7  d;
9  *  3,    x, -f*x-x*z+x;
3  d;
10 +  8,    9, -x*z+x;
```

```
$ pacheck ex1.{input,proof,target}
[pacheck] Pacheck Version 001
[pacheck] Practical Algebraic Calculus Proof Checker
[pacheck] Copyright (C) 2020, Daniela Kaufmann, Johannes Kepler University Linz
[pacheck] compressed mode with indices assumed
[pacheck] sorting according to strcmp
[pacheck] checking target enabled
[pacheck] reading target polynomial from 'ex1.target'
[pacheck] read 8 bytes from 'ex1.target'
[pacheck] reading original polynomials from 'ex1.input'
[pacheck] found 2 original polynomials in 'ex1.input'
[pacheck] read 20 bytes from 'ex1.input'
[pacheck] reading polynomial algebraic calculus proof from 'ex1.proof'
[pacheck] found and checked 8 inferences in 'ex1.proof'
[pacheck] read 219 bytes from 'ex1.proof'
[pacheck] found 1 target polynomial inference
[pacheck] proof length 10 (number of polynomials)
[pacheck] proof size 25 (on average 2.5 terms per polynomial)
[pacheck] proof degree 3 (internal maximum degree 3)
[pacheck] searched 32 inferences 0.1 average collisions
[pacheck] 10 inferences, 3.2 average searches
[pacheck] original inferences 2 (20% of total rules)
[pacheck] inference rules 8 (80% of total rules)
[pacheck] addition inference rules 3 (38% of inference rules)
[pacheck] multiplication inference rules 4 (50% of inference rules)
[pacheck] extension rules 1 (12% of inference rules)
[pacheck] deletion inference rules 3 (30% of total rules)
[pacheck] maximum 9 of total 10 terms (90%)
[pacheck] searched 52 terms 81% hits 0.3 average collisions
[pacheck] maximum 2261 bytes allocated (0.0 MB)
[pacheck] maximum resident set size 5922816 bytes (5.6 MB)
[pacheck] process time 0.001 seconds
[pacheck] TARGET CHECKED
```

```
$ pasteque ex1.{input,proof,target}
c polys parsed
c *****************
c pac parsed
c spec parsed
c Now checking
s SUCCESSFULL
c
c
c ***** stats *****
c parsing polys file init (nonGC): 0.000 s = 0.000 s (usr) 0.000 s (sys)
c parsing pac file init (nonGC): 0.000 s = 0.000 s (usr) 0.000 s (sys)
c full init (nonGC): 0.000 s = 0.000 s (usr) 0.000 s (sys)
c time solving (nonGC): 0.000 s = 0.000 s (usr) 0.000 s (sys)
c time GC: 0.000 s = 0.000 s (usr) 0.000 s (sys)
c time solving(full): 0.000 s
c Overall (nonGC): 0.000 s = 0.000 s (usr) 0.000 s (sys)
c overall GC: 0.000 s = 0.000 s (usr) 0.000 s (sys)
c Overall(full): 0.000 s
```

```
$ vim ex1.proof
```

```
3  = f, -z+1;
4  *  3,  y-1, -f*y+f-y*z+y+z-1;
5  +  2,    4, -f*y+f;
2  d;
4  d;

6  *  1,   f, f*x * y;
1  d;

7  *  5,    x, -f*x*y+f*x;
8  +  6,    7, f*x;
9  *  3,    x, -f*x-x*z+x;
10 +  8,    9, -x*z+x;
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

```
3  = f, -z+1;
4  *  3,  y-1, -f*y+f-y*z+y+z-1;
5  +  2,    4, -f*y+f;
2  d;
4  d;

6  *  1,   f, f*x + y;
1  d;

7  *  5,    x, -f*x*y+f*x;
8  +  6,    7, f*x;
9  *  3,    x, -f*x-x*z+x;
10 +  8,    9, -x*z+x;
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

```
$ pacheck ex1.{input,proof,target}
[pacheck] Pacheck Version 001
[pacheck] Practical Algebraic Calculus Proof Checker
[pacheck] Copyright (C) 2020, Daniela Kaufmann, Johannes Kepler University Linz
[pacheck] compressed mode with indices assumed
[pacheck] sorting according to strcmp
[pacheck] checking target enabled
[pacheck] reading target polynomial from 'ex1.target'
[pacheck] read 8 bytes from 'ex1.target'
[pacheck] reading original polynomials from 'ex1.input'
[pacheck] found 2 original polynomials in 'ex1.input'
[pacheck] read 8 bytes from 'ex1.target'
[pacheck] reading original polynomials from 'ex1.input'
[pacheck] found 2 original polynomials in 'ex1.input'
[pacheck] read 20 bytes from 'ex1.input'
[pacheck] reading polynomial algebraic calculus proof from 'ex1.proof'
*** 'pacheck' error in polynomial multiplication rule 4 with index 6  in 'ex1.proof' line 7: conclusion polynomial:
f*x+y
does not match expected result:
f*x*y
```

Evaluation

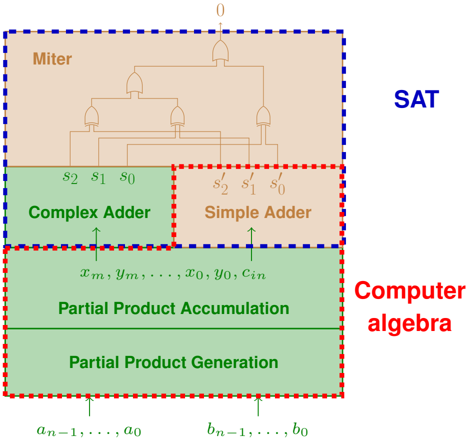
Comparison between:

PACTRIM the older version

PACHECK with no delete, no index, everything

PASTÈQUE with and without `uloop`

Evaluation



Complex adder: Replace it by simpler adder [DATE20,Kaufmann et al.] via extensions.

Evaluation Results

multiplier	n	steps (10^6)	ext	deg	PACTRIM sec	PACTRIM MB	PACHECK no index sec	PACHECK no index MB	PACHECK no delete sec	PACHECK no delete MB	PACHECK default sec	PACHECK default MB
btor	128	0.4	0	3	10	105	11	100	**5**	273	**5**	92
btor	256	1.6	0	3	60	459	62	435	**25**	1 144	**25**	364
btor	512	6.3	0	3	395	2 066	402	1 972	**138**	4 956	141	1 461
sp-ar-rc	128	0.6	0	4	16	156	16	148	**6**	454	**6**	136
sp-ar-rc	256	2.3	0	4	92	687	96	651	29	1 858	**27**	541
sp-ar-rc	512	9.4	0	4	587	3 107	617	2 965	146	7 683	**134**	2 171
sp-ar-cl	32	1.6	2 081	256	31	405	36	354	23	773	**21**	353
sp-dt-lf	32	0.3	1 684	46	3	82	3	73	**2**	122	**2**	73
bp-ct-bk	32	0.2	1 405	25	2	57	2	52	**1**	86	**1**	51
bp-wt-cl	32	5.6	5 740	764	242	1 716	302	1 430	193	4 324	**181**	1 428

Evaluation Results

multiplier	n	steps (10^6)	ext	deg	PACHECK default sec	PACHECK default MB	PASTÈQUE default sec	PASTÈQUE default MB	PASTÈQUE uloop sec	PASTÈQUE uloop MB
btor	128	0.4	0	3	**5**	92	22	3 886	17	1 773
btor	256	1.6	0	3	**25**	364	105	21 157	79	4 364
btor	512	6.3	0	3	**141**	1 461	531	64 412	416	22 292
sp-ar-rc	128	0.6	0	4	**6**	136	31	5 002	23	1 608
sp-ar-rc	256	2.3	0	4	**27**	541	139	32 525	102	8 769
sp-ar-rc	512	9.4	0	4	**134**	2 171	608	64 412	471	25 632
sp-ar-cl	32	1.6	2 081	256	**21**	353	121	40 654	113	9 492
sp-dt-lf	32	0.3	1 684	46	**2**	73	11	1 679	11	886
bp-ct-bk	32	0.2	1 405	25	**1**	51	8	1 600	7	1 068
bp-wt-cl	32	5.6	5 740	764	**181**	1 428	786	58 867	774	64 404

Conclusion & Future Work

- Proof Checker:
  - PACHECK is 30-80% faster than older proof checker.
  - PASTÈQUE is 4$\times$-slower, but fully verified.
- New PAC format:
  - No need to add extension variables to input polynomials anymore.
  - Indices and assumption of Boolean variables reduces the proof size.
  - Deletion rules reduce the memory usage.

- Investigate whether lifting the restrictions on EXT is useful.
- Explore connection to Nullstellensatz Proofs to gain shorter proofs:
  - D. Kaufmann and A. Biere. Nullstellensatz-Proofs for Multiplier Verification. In CASC 2020.

THE PROOF CHECKERS PACHECK AND PASTÈQUE FOR THE PRACTICAL ALGEBRAIC CALCULUS

Daniela Kaufmann, Mathias Fleury and Armin Biere
Johannes Kepler University
Linz, Austria

FMCAD 2020

September 24, 2020
Online

JOHANNES KEPLER UNIVERSITY LINZ