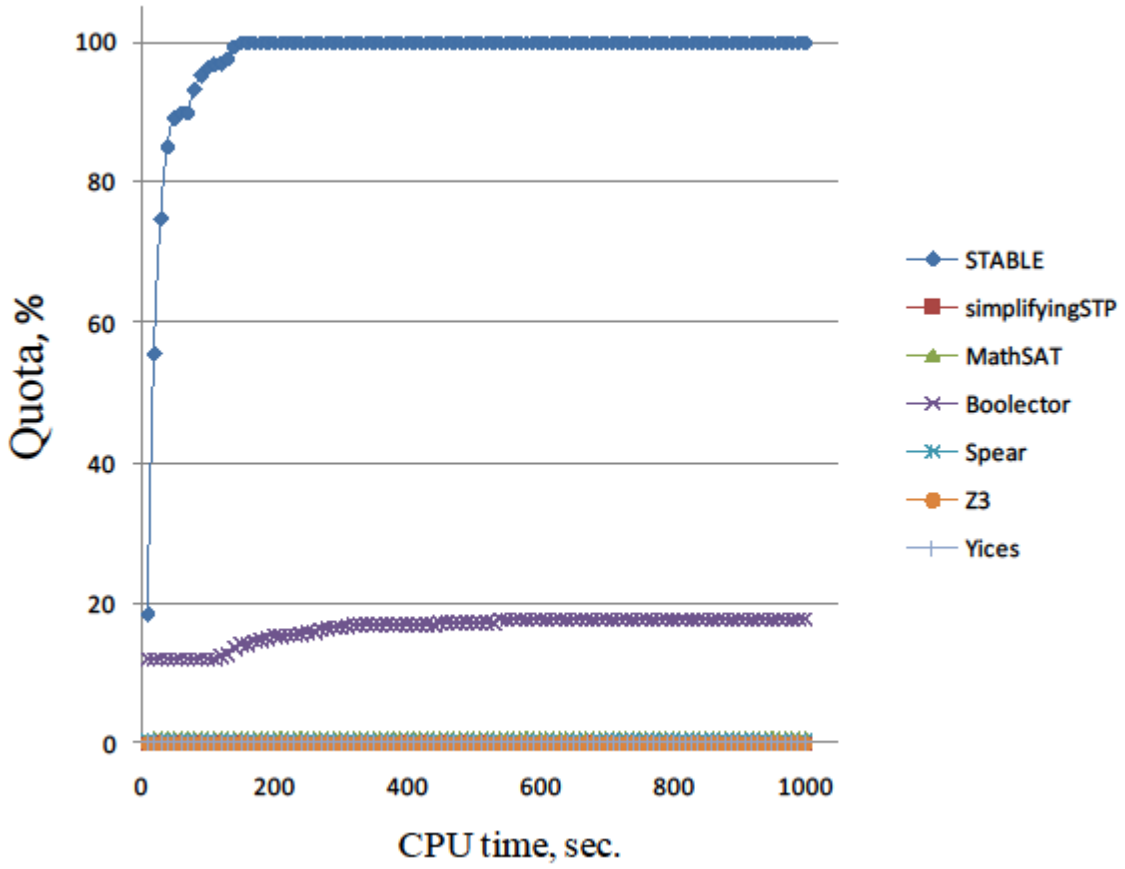
Preprocessing Polynomials for Arithmetic Reasoning within the SMT-Solver STABLE

Oliver Marx, Evgeny Pavlenko, Markus Wedler, Dominik Stoffel, Wolfgang Kunz
Electronic Design Automation Group
Dep. of Elect. and Comp. Eng., University of Kaiserslautern, Germany

Alexander Dreyer
Department of System Analysis, Prognosis and Control
Fraunhofer ITWM, Kaiserslautern, Germany

Gert-Martin Greuel
Computer Algebra Group
Department of Mathematics, University of Kaiserslautern, Germany

STABLE = SMT+ABL+GB



"STABLE: A new QF-BV SMT solver for hard verification problems combining Boolean reasoning with computer algebra", (DATE 2011), Grenoble, France.

Topic of this talk

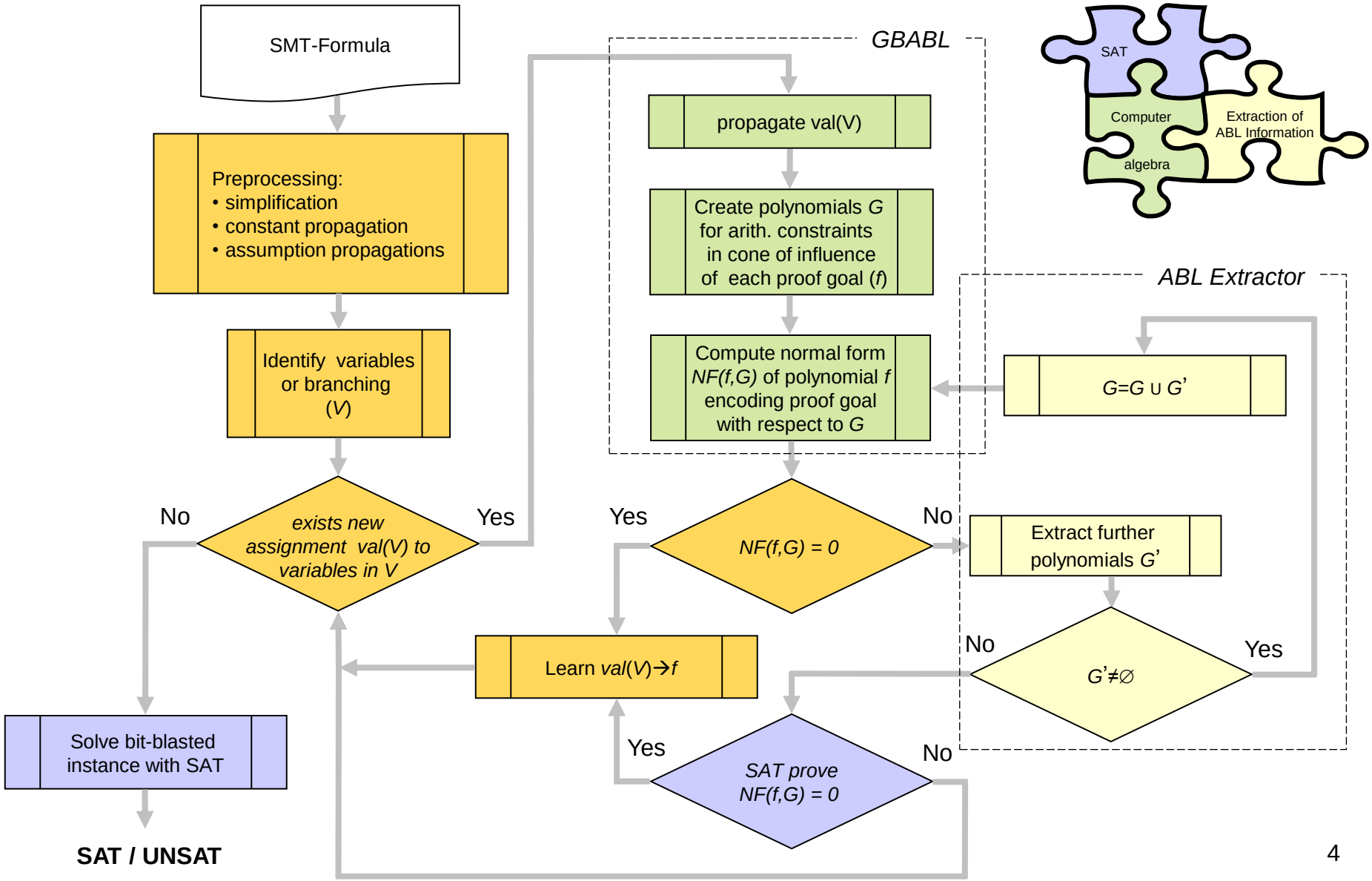
- The heart of STABLE is a computer-algebra-based engine. It incorporates an algebraic normal form algorithm that requires a polynomial model of the arithmetic parts of an SMT formula.

- The effectiveness of this algorithm may heavily depend on the initial formulation of the polynomials.

  - **Preprocessing of polynomials is crucial for successful performance of the normal form algorithm.**

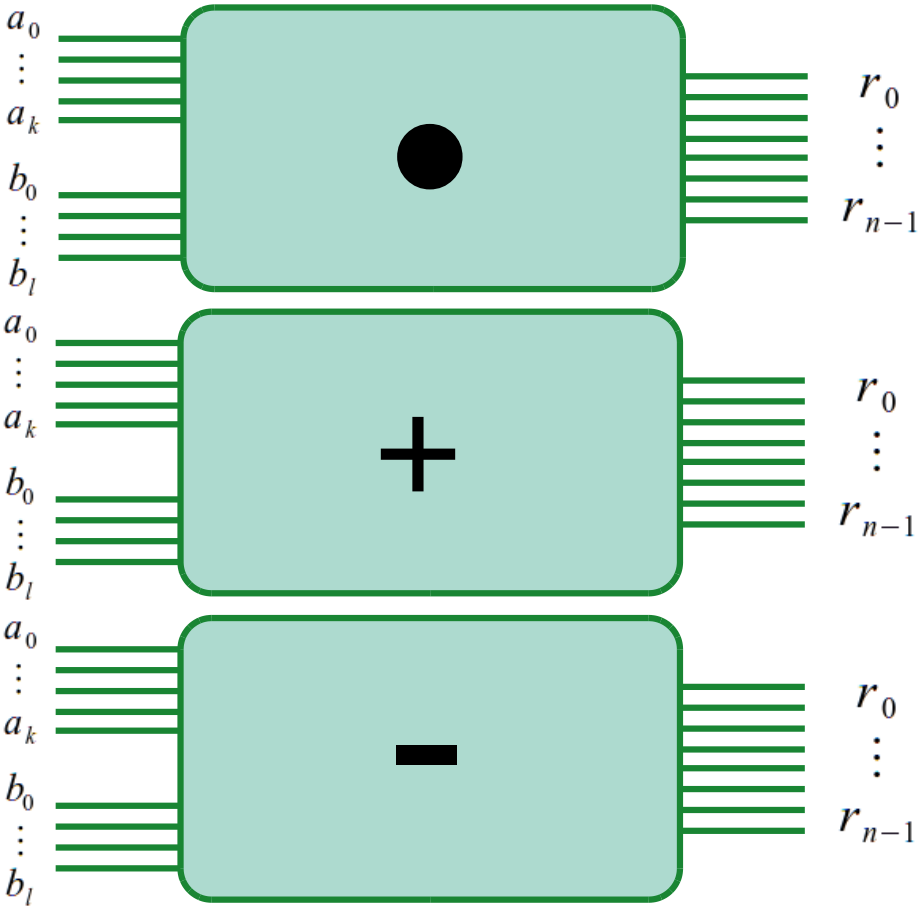
… however, for proper understanding of this problem, let us briefly recall how arithmetic models are treated in STABLE.

Design of the SMT-solver STABLE

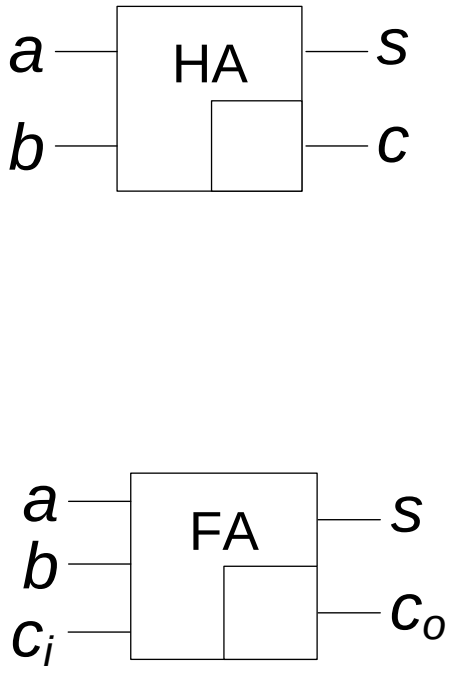


Modeling arithmetic problem parts

- Polynomial models available for:



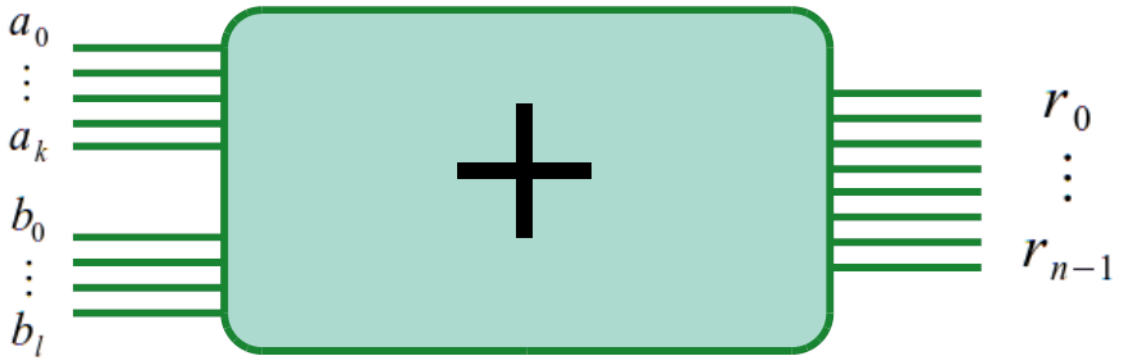
arithmetic word level



arithmetic bit level (ABL)

Modeling arithmetic problem parts

- Model each individual data path component



using polynomials over the ring $Z/2^N$

$$2^{n-1}r_{n-1}+\ldots+r_0-(2^k a_k+\ldots+a_0+2^l b_l+\ldots+b_0)+2^n s_n,\quad k<n,\quad l<n$$

$$2^{n-2}r_{n-2}+\ldots+r_0-(2^k a_k+\ldots+a_0+2^l b_l+\ldots+b_0)+2^{n-1} s_{n-1},\quad k<n-1,\quad l<n-1$$

$$\vdots$$

$$r_0-(a_0+b_0)+2s_1$$

Slack variables are artificial variables that are introduced to mimic the wrap around semantics of the components within a larger ring.

Application of Gröbner basis techniques

- Overall polynomial system

$$G = \left\{ g_{j,k} : \begin{array}{l} j = 1, \ldots, \text{number of components} \\ k = 1, \ldots, \text{bitwidth of } j\text{-th component} \end{array} \right\}$$

- Polynomial proof goal

$$g \in Z / 2^N [x_1, ..., x_n]$$

- Example: *k*-bit equality comparison

$$g = \sum_{i=0}^{k-1} 2^i (a_i - b_i)$$

Application of Gröbner basis techniques

- Validity of proof goal is equivalent to

$$V(\langle G \rangle) \subseteq V(g)$$

- Since G is a strong Gröbner basis this holds if and only if

$$NF(2^{N-n} g, G) = 0$$

- Model can be refined using bit-valued variables such that zero function test for $NF(2^{N-n} g, G)$ becomes unnecessary.

Application of Gröbner basis techniques

- Possible reasons for $NF(2^{N-n}g, G) \neq 0$:

  Outer: The arithmetic parts of a design and a property, out of which an SMT formula was generated, might include custom-designed components, non-arithmetic constraints, and bugs:

  - *Treated by polynomial extraction and resource-limited SAT checks.*

  Inner: Processing of slack variables is not powerful enough in polynomial systems of the computer-algebra engine:

  - *Explained on the next slides.*

Slack variables in STABLE

$$G_j^{(t)} : \sum_{i=0}^{t-1} 2^i r_i^{(j)} = f_j^{(t)}(a_1^{(j)}, a_2^{(j)}, \ldots, a_{m_j}^{(j)}) \bmod 2^t$$

$$\tilde{G}_j^{(t)} := \sum_{i=0}^{t-1} 2^i r_i^{(j)} - f_j^{(t)}(a_1^{(j)}, a_2^{(j)}, \ldots, a_{m_j}^{(j)}) + 2^t s_t^{(j)}$$

- Pro: Enables transformation of an arithmetic equation into an element of the polynomial ring over $Z / 2^N$.
- Contra: Slack variables are only visible inside the algebraic engine.
  Replacement of slack variables by their defining polynomials may eventually result in a computational blow-up.

➢ Generate as few slack variables as possible!

➢ Minimize the set of slack variables as much as possible!

Simplification of polynomials in STABLE

- **Estimate bounds**

  Compute an upper bound f_{ub} of $f(X)$ in the polynomial

$$p = \sum_{i=0}^{n-1} r_i - f(X) + 2^n s.$$

  The term $2^n s$ can be omitted if $f_{ub} < 2^n$.

  Example:

$$r_0 = x_1 \cdot x_2 \bmod 2 \quad \Rightarrow \quad r_0 - x_1 \cdot x_2 + 2s \in Z / 2^N [X]$$

$$x_1, x_2 \in \{0,1\} \quad \text{and} \quad (x_1 \cdot x_2)_{ub} < 2 \quad \Rightarrow \quad s = 0$$

$$r_0 - x_1 \cdot x_2$$

Simplification of polynomials in STABLE

- **Improve bounds**

  For a particular node, reuse the bounds computed in fan-ins and propagate them towards fan-outs.

  Example:

$$r_0 - (x_1 + x_2) + 2s \in Z / 2^N [X],$$

$$\text{where} \quad (x_1)_{ub} = 1 \quad \text{and} \quad (x_2)_{ub} = 0,$$

$$\text{then} \quad (x_1 + x_2)_{ub} = 1 \quad \text{and} \quad (x_1 + x_2)_{ub} < 2 \quad \Rightarrow \quad s = 0.$$

Simplification of polynomials in STABLE

- **Match polynomials**

  a) Detect common slack variables from different components.

  Example:

$$\begin{cases} p_1 = f_1(x) - 2^{n_1} s_1 \\ p_2 = f_2(x) - 2^{n_2} s_2 \end{cases}$$

$$\text{if} \quad f_1(x) = f_2(x) \quad \text{and} \quad n_1 = n_2 \quad \text{then} \quad s_1 = s_2 .$$

Simplification of polynomials in STABLE

- **Match polynomials**

  b) Reuse slack variables for different polynomials originating from the same SMT constraint.

Example:

$$r_0 + 2r_1 = (a_0 + b_0) + 2(a_1 + b_1) \bmod 2^2, \quad \text{with} \quad a_1 = 1, b_1 = 0$$

$$\begin{cases} p_1 = r_0 - (a_0 + b_0) + 2s_0 \\ p_2 = r_0 + 2r_1 - ((a_0 + b_0) + 2) + 4s_1 \end{cases}$$

However, if the partial assignment $(a_1 = 1, b_1 = 0)$ results in an overflow then it occurs simultaneously for both polynomials. Therefore, it always holds that $s_0 = s_1$.

Experimental results

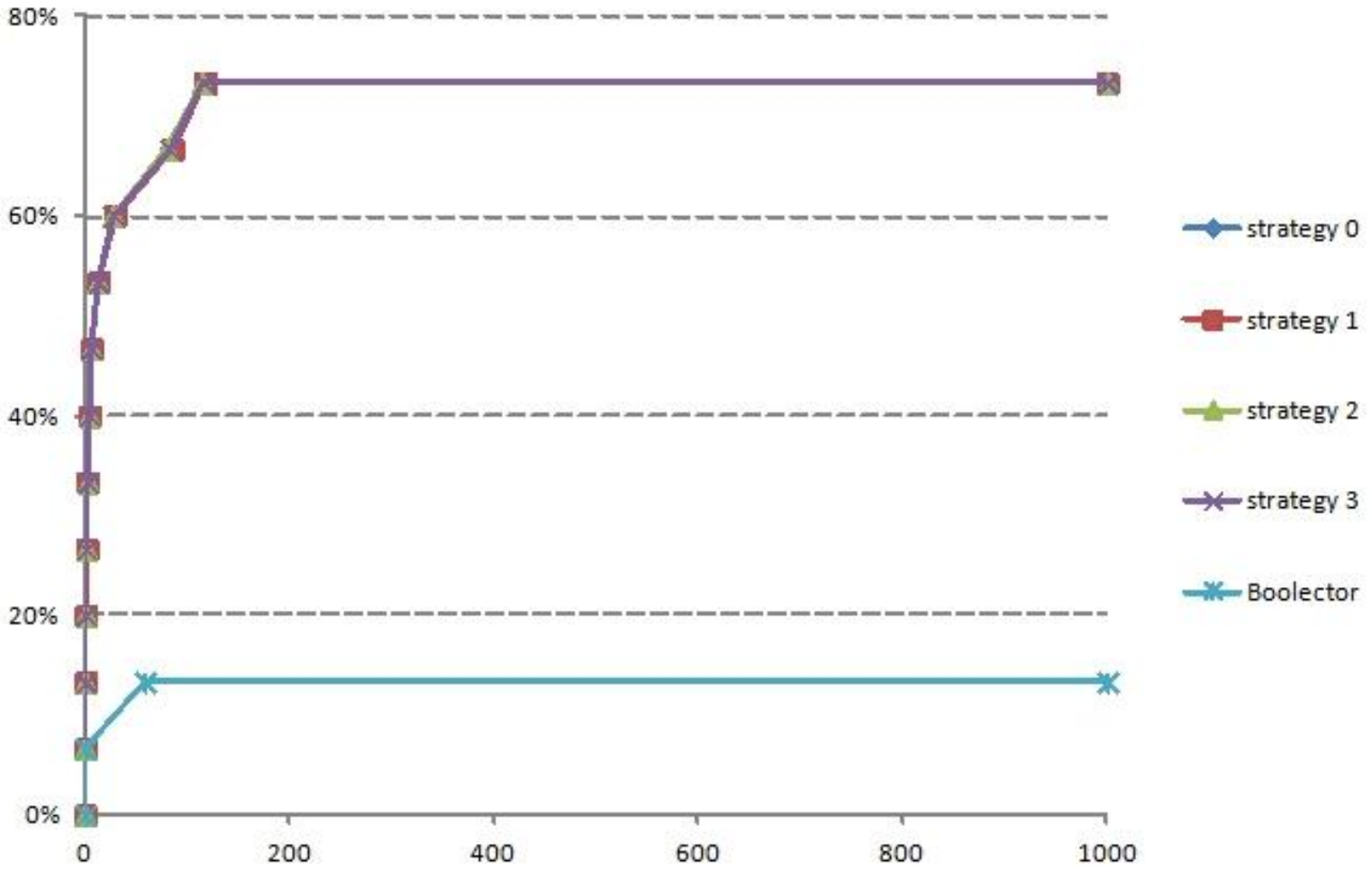
- The proposed simplification algorithms were categorized into four slack strategies in STABLE as follows:
  - *Strategy 0:* *no slack optimizations*
  - *Strategy 1:* *matching techniques*
  - *Strategy 2:* *bounding techniques*
  - *Strategy 3:* *matching and bounding techniques*

- Additionally, performance of STABLE was compared against Boolector (winner 2008 SMT-competition).

- All experiments were carried out on Intel Xeon CPU E5440 2,83 GHz 24 GB RAM running Linux.

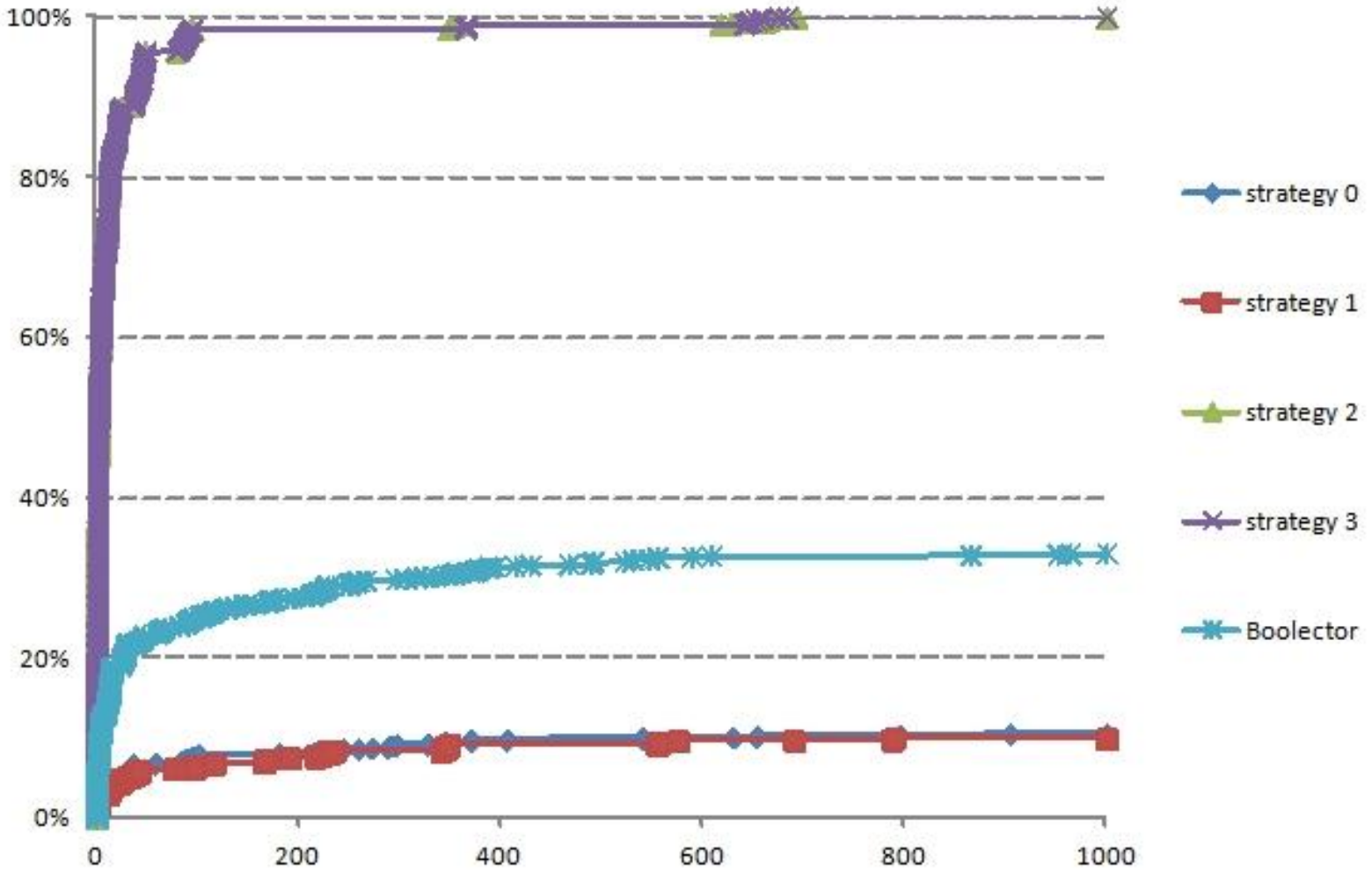
Experimental results

1) Suite of self-generated SMT instances that verify the function $f = a \cdot (b \cdot (c \cdot d))$ by a multiplier design with bit widths: 4 to 24.



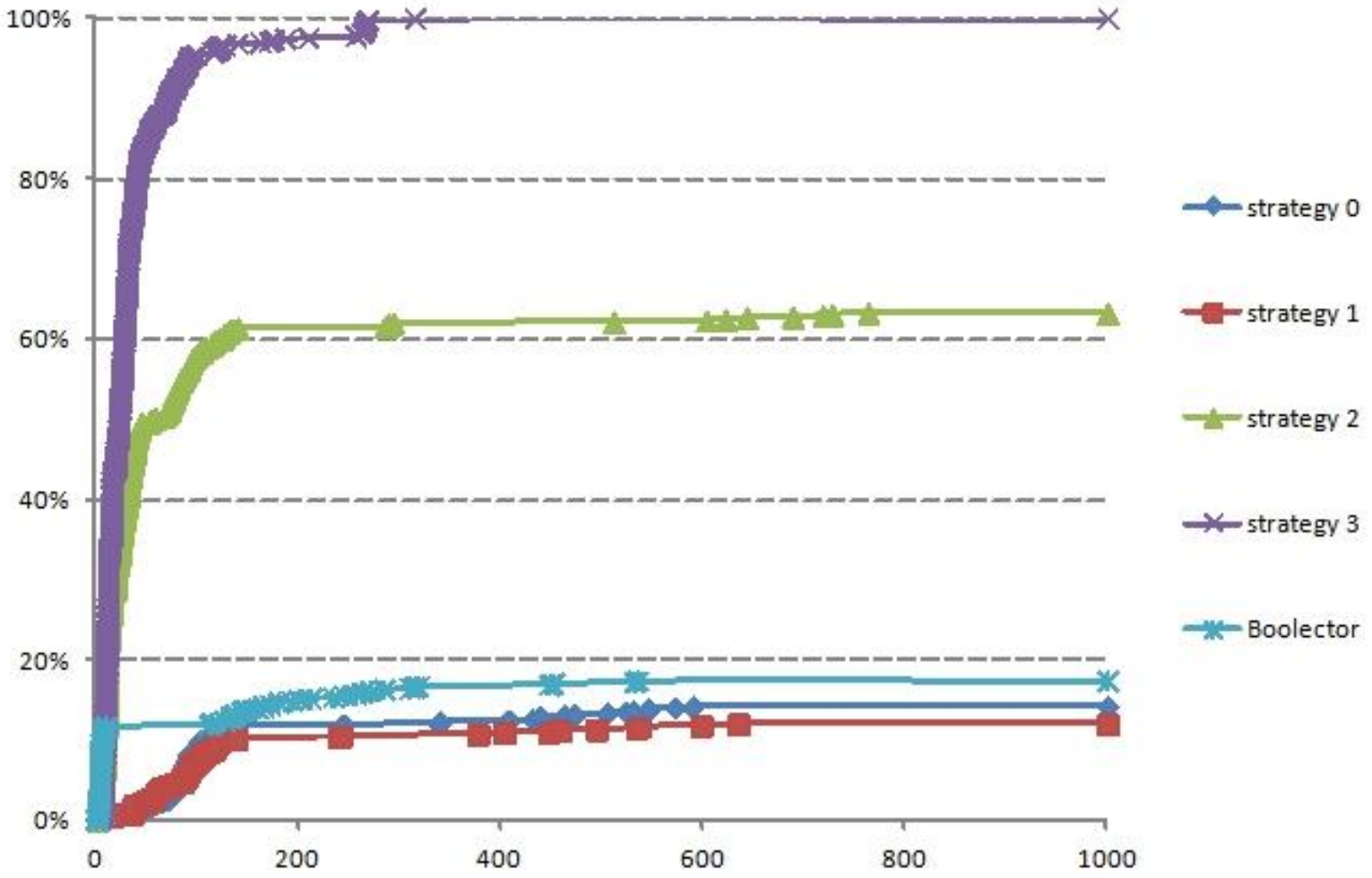
Experimental results

2) Suite of 1040 SMT instances that verify designs of industrial Booth-encoded multipliers for bit width from 4 up to 64 bits.



Experimental results

3) Suite of 640 SMT instances that verify arithmetic operations in an RTL design of the industrial TriCore processor.



Conclusion

- STABLE demonstrates outstanding performance when solving data-path verification problems at the register transfer level.

- Preprocessing techniques for polynomial models are vital for the practical efficiency of the overall SMT-based verification flow in STABLE.

Thank you for your attention!