Satisfiability Modulo Extensional Constant Arrays

Mathias Preiner, Aina Niemetz, Clark Barrett

CAV, July 27, 2026

Stanford University

Why Constant Arrays?

SMT-LIB Arrays

- Model memory in hardware and software verification
  - Theory of arrays: `read` / `write` + extensionality (equalities between arrays)

Why Constant Arrays?

SMT-LIB Arrays

- Model memory in hardware and software verification
  - Theory of arrays: `read` / `write` + extensionality (equalities between arrays)
- Modeling initialized memory (e.g., zero-initialized) is cumbersome
  - Infinite index domains: requires quantifiers
  - Finite index domains: individual write operations to set default value for all indices

Why Constant Arrays?

SMT-LIB Arrays

- Model memory in hardware and software verification
  - Theory of arrays: `read` / `write` + extensionality (equalities between arrays)
- Modeling initialized memory (e.g., zero-initialized) is cumbersome
  - Infinite index domains: requires quantifiers
  - Finite index domains: individual write operations to set default value for all indices

Constant Arrays Extension

- Store one default value at every index: succinct, natural encoding
- Allows to represent array model values

Why Constant Arrays?

SMT-LIB Arrays

- Model memory in hardware and software verification
  - Theory of arrays: `read` / `write` + extensionality (equalities between arrays)
- Modeling initialized memory (e.g., zero-initialized) is cumbersome
  - Infinite index domains: requires quantifiers
  - Finite index domains: individual write operations to set default value for all indices

Constant Arrays Extension

- Store one default value at every index: succinct, natural encoding
- Allows to represent array model values
- Until now: extensional constant arrays only for infinite index domains
  - Finite domains: non-extensional fragments supported, some solvers unsound

Why Constant Arrays?

SMT-LIB Arrays

- Model memory in hardware and software verification
  - Theory of arrays: `read` / `write` + extensionality (equalities between arrays)
- Modeling initialized memory (e.g., zero-initialized) is cumbersome
  - Infinite index domains: requires quantifiers
  - Finite index domains: individual write operations to set default value for all indices

Constant Arrays Extension

- Store one default value at every index: succinct, natural encoding
- Allows to represent array model values
- Until now: extensional constant arrays only for infinite index domains
  - Finite domains: non-extensional fragments supported, some solvers unsound

This Work

- Calculus for extensional constant arrays, independent of index domain
- Refutational and satisfiability soundness proof
- Implementation and evaluation

Theory of Extensional Arrays in SMT

Function symbols: `read`, `write`

- $a[i]$ ... read element of array a at index i
  SMT-LIB: `(select a i)`
- $a\langle i \lhd u \rangle$ ... construct new array identical to a except for index i updated with element u
  SMT-LIB: `(store a i u)`

Axiomatization

1. Read-over-write 1: $\forall a, i, j, u : i \approx j \implies a\langle i \lhd u \rangle[j] \approx u$
   Reading the written index returns the new value u
2. Read-over-write 2: $\forall a, i, j, u : i \not\approx j \implies a\langle i \lhd u \rangle[j] \approx a[j]$
   Reading any other index is unaffected by the write
3. Extensionality: $\forall a, b : a \approx b \iff \forall i : a[i] \approx b[i]$
   Two arrays are equal iff they agree at every index

Constant Arrays Extension

Constant array constructor

- $\langle v \rangle_\sigma$ ... construct new array with default value v at every index of sort σ
  SMT-LIB (non-standard): `((as const (Array IndexSort ElementSort)) v)`

Axiomatization

4. Read-over-constant: $\forall i, v : \langle v \rangle[i] \approx v$
   Reading a constant array at any index returns its default value v

Support so far

- Existing extensional procedures: infinite index domains only
  [Stump 2001; de Moura 2009; Hoenicke 2019]
- Finite index domains (e.g., bit-vectors) are more challenging

Example: Infinite vs. Finite Index Domains

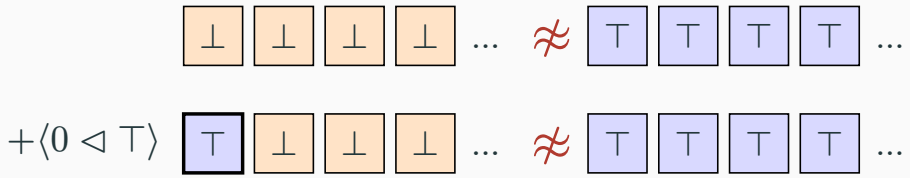
Infinite constant arrays

Constant arrays $\langle \bot \rangle$ and $\langle \top \rangle$ over indices of sort $\mathbb{Z}$

Example: Infinite vs. Finite Index Domains

Infinite constant arrays

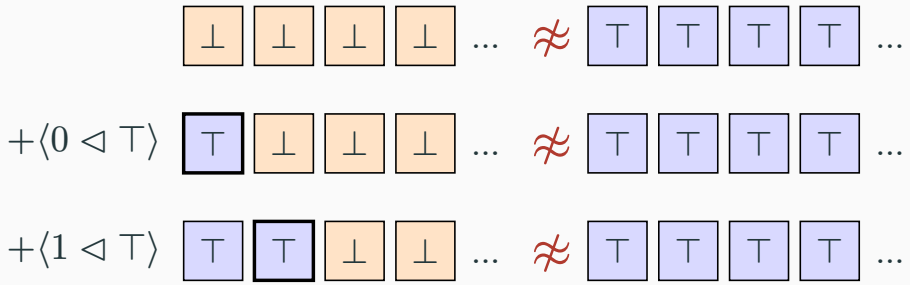
Constant arrays $\langle \bot \rangle$ and $\langle \top \rangle$ over indices of sort $\mathbb{Z}$



Example: Infinite vs. Finite Index Domains

Infinite constant arrays

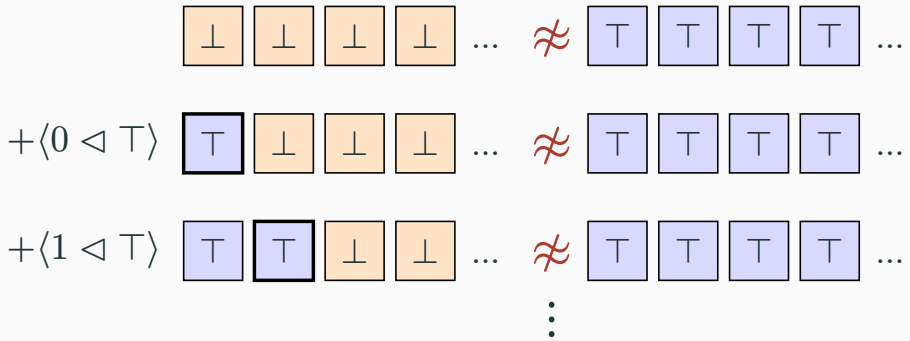
Constant arrays $\langle \bot \rangle$ and $\langle \top \rangle$ over indices of sort $\mathbb{Z}$



Example: Infinite vs. Finite Index Domains

Infinite constant arrays

Constant arrays $\langle \bot \rangle$ and $\langle \top \rangle$ over indices of sort $\mathbb{Z}$



× **UNSAT** — no finite number of stores covers $\mathbb{Z}$

➤ Direct or indirect equalities of constant arrays with different default value are always false

Example: Infinite vs. Finite Index Domains

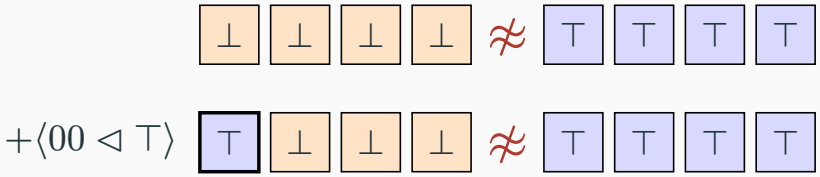
Finite constant arrays

Constant arrays $\langle \bot \rangle$ and $\langle \top \rangle$ over indices of bit-vectors of size 2

Example: Infinite vs. Finite Index Domains

Finite constant arrays

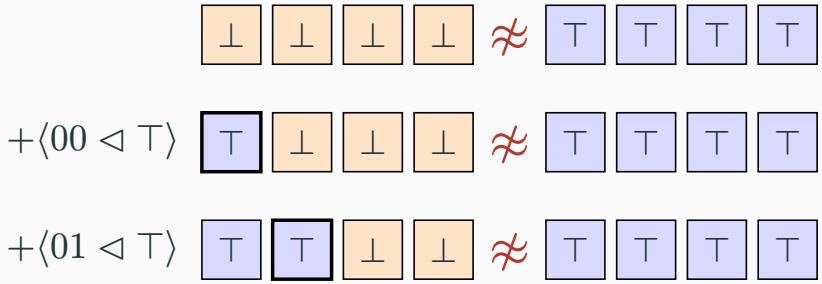
Constant arrays $\langle \bot \rangle$ and $\langle \top \rangle$ over indices of bit-vectors of size 2



Example: Infinite vs. Finite Index Domains

Finite constant arrays

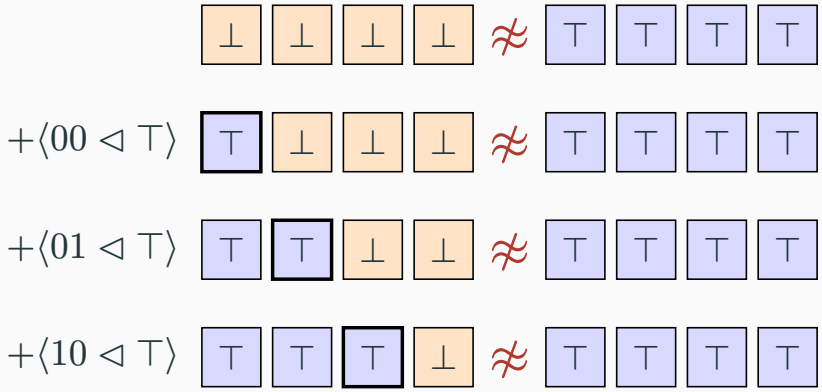
Constant arrays $\langle \bot \rangle$ and $\langle \top \rangle$ over indices of bit-vectors of size 2



Example: Infinite vs. Finite Index Domains

Finite constant arrays

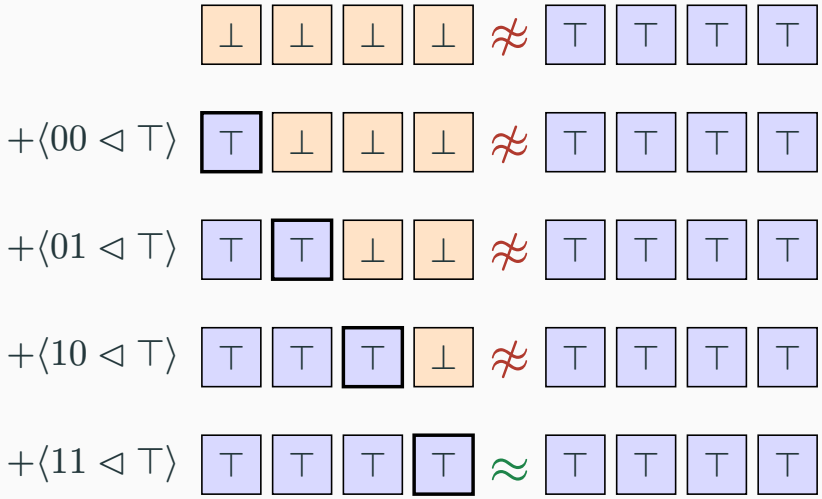
Constant arrays $\langle \bot \rangle$ and $\langle \top \rangle$ over indices of bit-vectors of size 2



Example: Infinite vs. Finite Index Domains

Finite constant arrays

Constant arrays $\langle \bot \rangle$ and $\langle \top \rangle$ over indices of bit-vectors of size 2

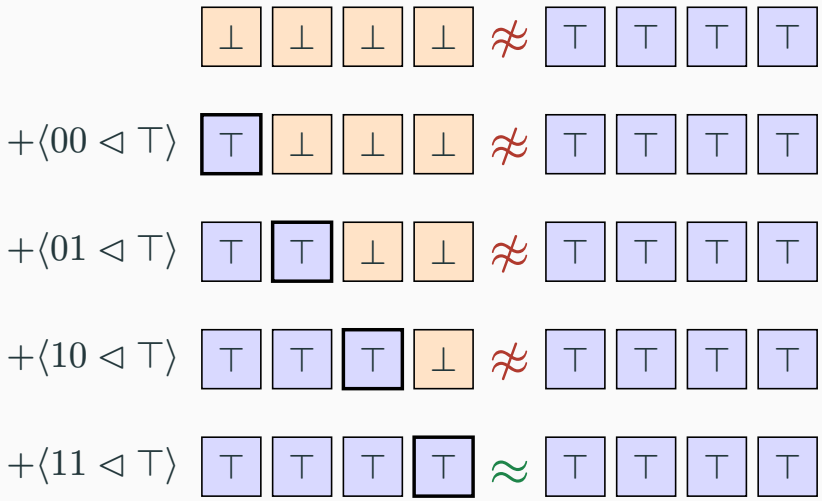


✓ **SAT** — 4 stores cover all of $\{00, 01, 10, 11\}$

Example: Infinite vs. Finite Index Domains

Finite constant arrays

Constant arrays $\langle \bot \rangle$ and $\langle \top \rangle$ over indices of bit-vectors of size 2



✓ **SAT** — 4 stores cover all of $\{00, 01, 10, 11\}$

- Find arrangement of write indices to “make them equal”
- Requires write indices to cover full index domain

Calculus

Idea

Search lazily under candidate model $\mathcal{I}$, propagating facts until saturation or a conflict.

Calculus

Idea

Search lazily under candidate model $\mathcal{I}$, propagating facts until saturation or a conflict.

Base calculus (10 rules)

- Generalizes lemmas on demand array procedure [Brummayer 2009]
  - Decoupled from solving paradigm, arbitrary index/element theories
- Propagates array reads along `write`, $\approx$

  Example: $a[j] \approx u_1 \wedge a\langle i \lhd u_2 \rangle \approx b$

  ➤ if $j \not\approx i$ propagate $b[j] \approx u_1$
- Conflict: Two reads with the same index propagated to same array, but yield different values

Constant array extension (7 rules)

- Propagates default value of $\langle v \rangle$, tracking updated indices I_v along `write`, $\approx$

  Example: $\langle v \rangle \langle i \lhd u \rangle \approx b$ ➤ propagate v to b while tracking $I_v = \{i\}$

Constant array extension (7 rules)

- Propagates default value of $\langle v \rangle$, tracking updated indices I_v along `write`, $\approx$

  Example: $\langle v \rangle \langle i \lhd u \rangle \approx b$ ➤ propagate v to b while tracking $I_v = \{i\}$

- Non-updated indices $\bar{I}_v$ have to assume default value v

- Conflict: $\langle v \rangle$ and $\langle w \rangle$ propagated to the same array and $v \not\approx w$ and $\bar{I}_v \cap \bar{I}_w \neq \varnothing$

  ➤ Diversify I_v and I_w such that $\bar{I}_v \cap \bar{I}_w = \varnothing$

  ➤ $\bar{I}_v \cap \bar{I}_w = \varnothing$ requires $I_v \cup I_w$ to cover the full index domain D

  ➤ $|I_v \cup I_w| < |D| \rightarrow$ no coverage possible

Evaluation: Setup

Implementation

- Integrated into Bitwuzla, config $\text{Bitwuzla}_{\langle \mathcal{A} \rangle}$
- Compared against baseline Bitwuzla, cvc5, Z3, MathSAT5
- Limits: 1 core, 8 GB, 1200 seconds

Benchmark sets **(constant arrays are not part of SMT-LIB)**

- smtlib (5,112): quantified ABV/ABVFP/ABVFPLRA, ext. const. arrays in Bitwuzla MBQI subqueries
- mbqi (4,147): quantifier-free ext. const. arrays from smtlib MBQI subqueries
- k-ind (2,838): incr. queries, Pono k-induction on HWMCC 2019–2025, Bitwuzla failed (const. arrays)
- crafted (410): const. arrays native vs. quantified encodings, different extensional patterns

Evaluation: Results

Benchmarks	Bitwuzla$_{\langle\mathcal{A}\rangle}$	Bitwuzla	Z3	cvc5	MathSAT5
smtlib (5,112)	3,545	717	846	276	—
mbqi (4,147)	4,147	4	4,126	2,972	4,034
k-ind (2,838)	2,838	372	2,142	1,230	883
crafted native (205)	172	0	153*	0	107*
crafted quant (205)	117	113	28	42	—

— excluded (no quantifiers) * includes wrong answers

- 5× more solved than baseline on *smtlib* (3,545 vs. 717)
  - 27× faster on commonly solved
- Solves every *mbqi* and *k-ind* query the baseline failed
  - 140× faster than Z3 on commonly solved *mbqi*
- Bitwuzla$_{\langle\mathcal{A}\rangle}$ only solver sound on finite extensional constant arrays
  - Z3 and MathSAT5 accept the input but report wrong answers, reported to developers

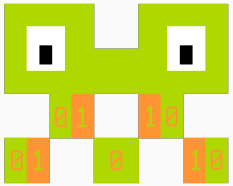
Conclusion

A decision procedure for extensional constant arrays

- Supports arbitrary index domains: finite and infinite
- Abstract calculus, decoupled from the solver architecture
- Refutational and satisfiability soundness proven
- Implemented in Bitwuzla, significantly expands its solving capabilities

Resources

- Bitwuzla: https://bitwuzla.github.io
- Artifact: https://zenodo.org/records/19713260



References

[Stump 2001] Stump, A., C.W. Barrett, D.L. Dill, and J.R. Levitt. A Decision Procedure for an Extensional Theory of Arrays. *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, IEEE Computer Society, pp. 29–37, 2001.

[de Moura 2009] de Moura, L.M., and N.S. Bjørner. Generalized, efficient array decision procedures. *Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA*, IEEE, pp. 45–52, 2009.

[Hoenicke 2019] Hoenicke, J., and T. Schindler. Solving and Interpolating Constant Arrays Based on Weak Equivalences. *Verification, Model Checking, and Abstract Interpretation - 20th International Conference, VMCAI 2019, Cascais, Portugal, January 13-15, 2019, Proceedings*, Springer, pp. 297–317, 2019.

[Brummayer 2009] Brummayer, R., and A. Biere. Lemmas on Demand for the Extensional Theory of Arrays. *J. Satisf. Boolean Model. Comput.*, pp. 165–201, 2009.