Exploiting Symmetry for Deadlock Detection in Message Passing Programs

Rishabh Ranjan, IIT Delhi
Subodh Sharma, IIT Delhi

Introduction

Message Passing (MP) programs are prevalent

The curse of **deadlocks**

Non-determinism makes verification hard

Undecidable in general

NP-complete[1] for terminating programs

A simple MP program:

P_0	P_1	P_2
$R_{0,*}$	$S_{1,0}$	$S_{2,0}$
$R_{0,2}$		

$S_{i,j}$: non-blocking Send from P_i to P_j
$R_{i,j}$: blocking Recv at P_i from P_j
$R_{i,*}$: blocking wildcard Recv at P_i

Trace: sequence of matches allowed by MP semantics

A non-deadlocking trace: $\langle (S_{1,0}, R_{0,*}),\ (S_{2,0}, R_{0,2}) \rangle$

A deadlocking trace: $\langle (S_{2,0}, R_{0,*}),\ \text{<deadlock>} \rangle$

[1] Forejt, Vojtěch, et al. "**Precise predictive analysis for discovering communication deadlocks in MPI programs.**" *International Symposium on Formal Methods*. Springer, Cham, 2014.

Existing Approaches

- Symbolic Verification (**MPI-SV**[1]**):**
  - good coverage of input space as long as it can be captured symbolically
  - not scalable
- Dynamic Trace Verification (**ISP**[2], **Mopper**[3], **DAMPI**[4]):
  - limited to programs with terminating execution traces for fixed inputs
  - faster than symbolic verification
- Hybrid Approach (**Hermes**[5]**):**
  - symbolic execution for multiple path conditions; dynamic verification for interleavings within the path
  - **state-of-the-art** for dynamic verification of programs with terminating execution traces
- Mopper and Hermes are **SAT-based** methods.

[1] Chen, Zhenbang, et al. "**MPI-SV: a symbolic verifier for MPI programs.**" *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2020.
[2] Vakkalanka, Sarvani S., et al. "**ISP: a tool for model checking MPI programs.**" *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*. 2008.
[3] Forejt, Vojtěch, et al. "**Precise predictive analysis for discovering communication deadlocks in MPI programs.**" *International Symposium on Formal Methods*. Springer, Cham, 2014.
[4] Vo, Anh, et al. "**A scalable and distributed dynamic formal verifier for MPI programs.**" *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010.
[5] Khanna, Dhriti, et al. "**Dynamic symbolic verification of MPI programs.**" *International Symposium on Formal Methods*. Springer, Cham, 2018.

Symmetry in SAT

Consider the SAT formula: $F = a \vee b \vee c$

All solutions for (a, b, c): $\{(1,0,0),(0,1,0),(0,0,1),(1,1,0),(1,0,1),(0,1,1),(1,1,1)\}$

Symmetry: swapping a & b preserves satisfiability of F. If (A, B, C) is a solution so is (B, A, C).

Symmetry-breaking: $F' = F \wedge (a \vee \neg b) \wedge (b \vee \neg c)$

Solutions of F': $\{(1,0,0),(1,1,0),(1,1,1)\}$

Key idea: Reduction in search space due to symmetry breaking speeds up SAT solving.

Limitations of SAT-level symmetry breaking

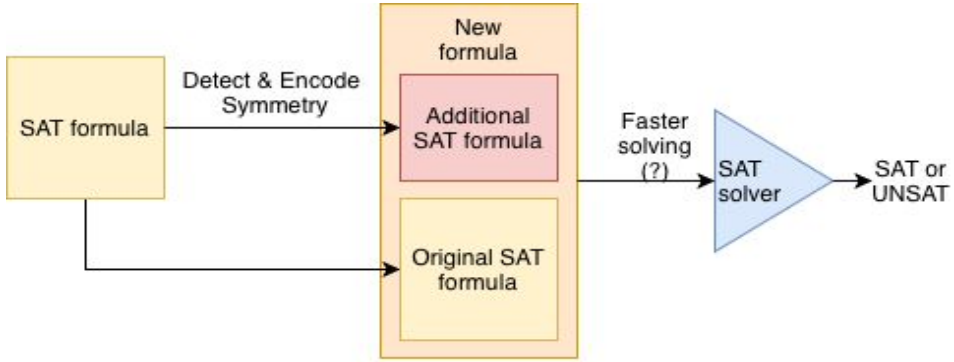


Fig. symmetry preprocessing as done by BreakID[1]/Shatter[2]

Formula generated by Hermes/Mopper is **large** (cubic in number of events)

Preprocessing time is unreasonably high

Preprocessed formula didn't show much gain in speed

Does not use **domain-specific information**

Solution: detect symmetries from the program itself

[1] Devriendt, Jo, et al. "**Improved static symmetry breaking for SAT**." *International Conference on Theory and Applications of Satisfiability Testing*. Springer, Cham, 2016.

[2] Aloul, Fadi A., Karem A. Sakallah, and Igor L. Markov. "**Efficient symmetry breaking for boolean satisfiability.**" *IEEE Transactions on Computers* 55.5 (2006): 549-558.

Our contributions

Symmetry in MP communication patterns introduces verification **redundancies**

Theoretical framework and algorithms to **characterize** and **detect** these redundancies

Faster SAT solving -- symmetry-breaking formulation

Simian: a symmetry-aware verification tool for push-button deadlock detection of C/C++ programs using **MPI (Message Passing Interface)**

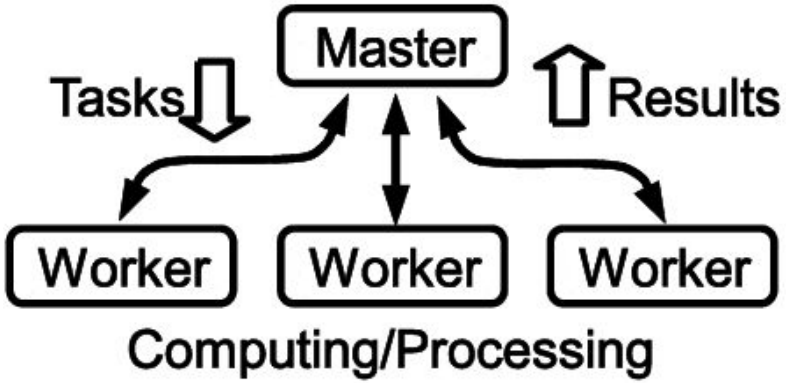


Fig. Master-Worker pattern

Runtime Comparison with Hermes

Benchmarks are from **FEVS benchmark suite**[1] and **Hermes**[2]

Timeout (**TO**) was 60s

Communication patterns include:

- Master-Worker pattern
- Grid/Torus topology
- Multiple paths

Benchmark	Procs	Hermes (s)	Simian (s)
adder	4	0.379	0.337
	8	0.829	0.718
	16	**TO**	1.512
	32	**TO**	4.854
diffusion2d	4	2.040	0.819
	8	29.358	2.989
	16	**TO**	20.088
	32	**TO**	**TO**
integrate	4	0.364	0.334
	8	0.763	0.719
	16	**TO**	1.539
	32	**TO**	4.523
matmul	4	0.328	0.346
	8	0.868	0.805
	16	3.064	1.625
	32	45.977	6.714
monte	4	1.395	0.503
	8	6.751	1.233
	16	**TO**	3.112
	32	**TO**	8.274

Disclaimer: this is a work-in-progress and results are preliminary

[1] Siegel, Stephen F., and Timothy K. Zirkel. "**FEVS: A functional equivalence verification suite for high-performance scientific computing.**" *Mathematics in Computer Science* 5.4 (2011): 427-435.

[2] Khanna, Dhriti, et al. "**Dynamic symbolic verification of MPI programs.**" *International Symposium on Formal Methods*. Springer, Cham, 2018.

Motivating Example

Program:

P_0	P_1	P_2
$S_{0,1}$	$R_{1,0}$	$R_{2,0}$
$S_{0,2}$	$S_{1,0}$	$S_{2,0}$
$R^a_{0,*}$		
$R^b_{0,*}$		

$S_{i,j}$: non-blocking Send from P_i to P_j
$R_{i,j}$: blocking Recv at P_i from P_j
$R_{i,*}$: blocking wildcard Recv at P_i
a, b: only to distinguish R's

Sample trace:

$$\tau : \langle (S_{0,1}, R_{1,0}),\ (S_{0,2}, R_{2,0}),\ (S_{1,0}, R^a_{0,*}),\ (S_{2,0}, R^b_{0,*}) \rangle$$

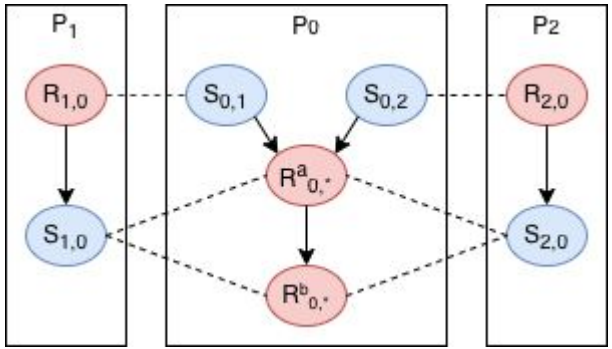
Equivalent trace:

$$\tau' : \langle (S_{0,2}, R_{2,0}),\ (S_{0,1}, R_{1,0}),\ (S_{2,0}, R^a_{0,*}),\ (S_{1,0}, R^b_{0,*}) \rangle$$

Idea: reduce search space to contain only one (or few) representatives for every class of equivalent traces.

Symmetry Extraction

Program Graph (PG):



Directed edges represent **matches-before order**.
Undirected edges represent **potential matches**.

Sample automorphism of PG:

$\phi : E \to E$, given as:
$S_{0,1} \mapsto S_{0,2}$, $S_{0,2} \mapsto S_{0,1}$,
$R_{1,0} \mapsto R_{2,0}$, $R_{2,0} \mapsto R_{1,0}$,
$S_{1,0} \mapsto S_{2,0}$, $S_{2,0} \mapsto S_{1,0}$,
$R^a_{0,*} \mapsto R^a_{0,*}$, $R^b_{0,*} \mapsto R^b_{0,*}$.

Note that $\phi(\tau) = \tau'$

Automorphism of traces (defn):

$\phi : E \to E$ which preserves deadlocking behavior on all traces.

Key Result: The automorphism group of PG is a subgroup of the automorphism group of traces.

The Graph Automorphism Problem

Automorphisms of a graph form a **group**

Automorphism group is described by **generators**, which are at most **linear** in the graph size

No worst-case polynomial algorithm known, but **very efficient** in practice[1]

Expected runtime is **linear** in the size of the graph

BreakID/Shatter also use graph automorphism -- input graph **orders of magnitude larger** than ours

[1] McKay, Brendan D., and Adolfo Piperno. "**Practical graph isomorphism, II.**" *Journal of Symbolic Computation* 60 (2014): 94-112.

SAT Formulation

1. **Trace Encoding:**
   - Encodes the MP semantics
   - Solutions correspond to valid traces

2. **Safety Constraints:**
   - Satisfied by deadlocking traces

3. **Symmetry-breaking Constraints:**
   - Reduce search space
   - Preserve satisfiability of 1 & 2

SAT constraints from Mopper

1-6 and 9-11 encode valid traces

7-8 enforce deadlock

A solution to this formula corresponds to a deadlocking trace

Partial order
$$\bigwedge_{b \in \mathcal{C}} \bigwedge_{a \in Imm(b)} t_{ab} \quad (1)$$

Unique match for **send**
$$\bigwedge_{(a,b) \in \mathbb{M}^+} \bigwedge_{c \in \mathbb{M}^+(a), c \neq b} (s_{ab} \rightarrow \neg s_{ac}) \quad (2)$$

Unique match for **receive**
$$\bigwedge_{(a,b) \in \mathbb{M}^+} \bigwedge_{c \in \mathbb{M}^+(b), c \neq a} (s_{ab} \rightarrow \neg s_{cb}) \quad (3)$$

Match correct
$$\bigwedge_{a \in \mathsf{R}} (m_a \rightarrow \bigvee_{b \in \mathbb{M}^+(a)} s_{ba}) \wedge \bigwedge_{a \in \mathsf{S}} (m_a \rightarrow \bigvee_{b \in \mathbb{M}^+(a)} s_{ab}) \quad (4)$$

Matched only
$$\bigwedge_{\alpha \in \mathbb{M}^+} (s_\alpha \rightarrow \bigwedge_{a \in \alpha} m_a) \quad (5)$$

No match possible
$$\bigwedge_{\alpha \in \mathbb{M}^+} (\bigvee_{a \in \alpha} (m_a \vee \neg r_a)) \quad (6)$$

All ancestors matched
$$\bigwedge_{b \in \mathcal{C}} (r_b \leftrightarrow \bigwedge_{a \in Imm(b)} m_a) \quad (7)$$

Not all matched
$$\bigvee_{a \in \mathcal{C}} \neg m_a \quad (8)$$

Match only issued
$$\bigwedge_{a \in \mathcal{C}} (m_a \rightarrow r_a) \quad (9)$$

Clock equality
$$\bigwedge_{(a,b) \in \mathbb{M}^+} (s_{ab} \rightarrow (clk_a = clk_b) \quad (10)$$

Clock difference
$$\bigwedge_{a,b \in \mathcal{C}} (t_{ab} \rightarrow (clk_a < clk_b)) \quad (11)$$

Symmetry-breaking Constraints

Result: If ø is an automorphism of traces, then:
(..., S_ab, ...) is a solution of the trace encoding iff
(..., S_ø(a)ø(b), ...) is a solution.
Thus, ø induces an **equivalence class** of solutions.

Lex-Leader[1] constraints:

(..., S_ab, ...) $\geq_{\text{LEXICOGRAPHICAL}}$ (..., S_ø(a)ø(b), ...)

SAT encoding for M $\geq_{\text{LEXICOGRAPHICAL}}$ N:

M and N are size-n bit-vectors.

$$M[1] \geq N[1]$$

$$X[1] \leftrightarrow (M[1] = N[1])$$

$$\forall 1 \leq i \leq n-2, X[i+1] \leftrightarrow \left[X[i] \wedge (M[i+1] = N[i+1])\right]$$

$$\forall 1 \leq i \leq n-1, X[i] \rightarrow (M[i+1] \geq N[i+1])$$

Soundness: at least one representative from each class remains a solution

Completeness: only one representative from each class remains a solution

Using all automorphisms ensures completeness but they may be **exponential** in number

Lex-Leader constraint for each **generator**:

- complete in common special cases
- good quality in general

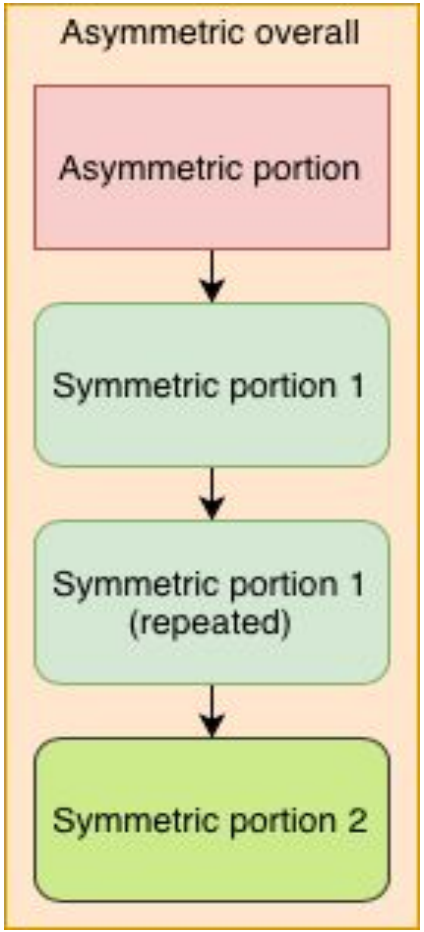
Size of symmetry breaking formula is **asymptotically same** as size of original formula

[1] Walsh, Toby. "**Symmetry breaking constraints: Recent results.**" *arXiv preprint arXiv:1204.3348* (2012).

From Global Symmetry to Local Symmetry

Overall symmetry - rare in real-world programs

Locally symmetric portions and repetitions are common



Communication Epochs

Theorem: The *strongly connected components* of the Program Graph can be verified independently

We call each strongly connected component an **epoch**

Apply symmetry techniques on epochs separately

Epochs can be **cached**

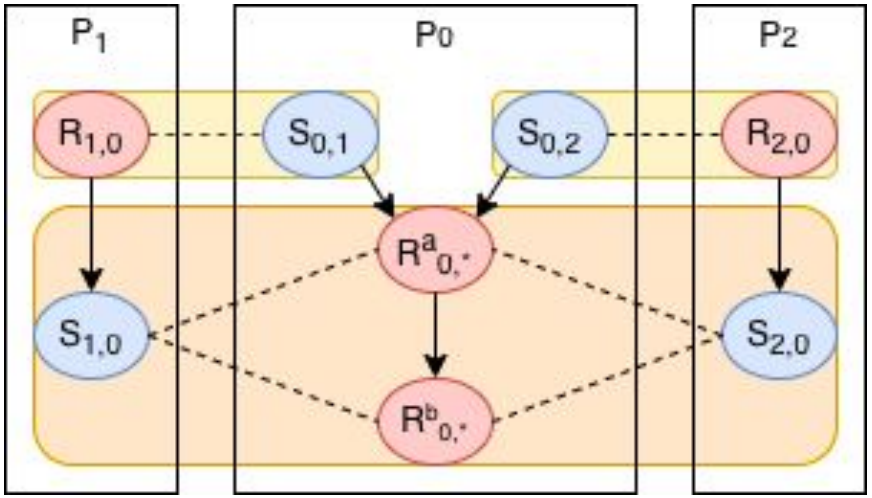


Fig. Epochs in our running example

From Single-Path to Multi-Path

Single-Path Programs: Control flow does not depend on inter-process communication

Multi-Path Programs: Multiple control flows depending on inter-process communication

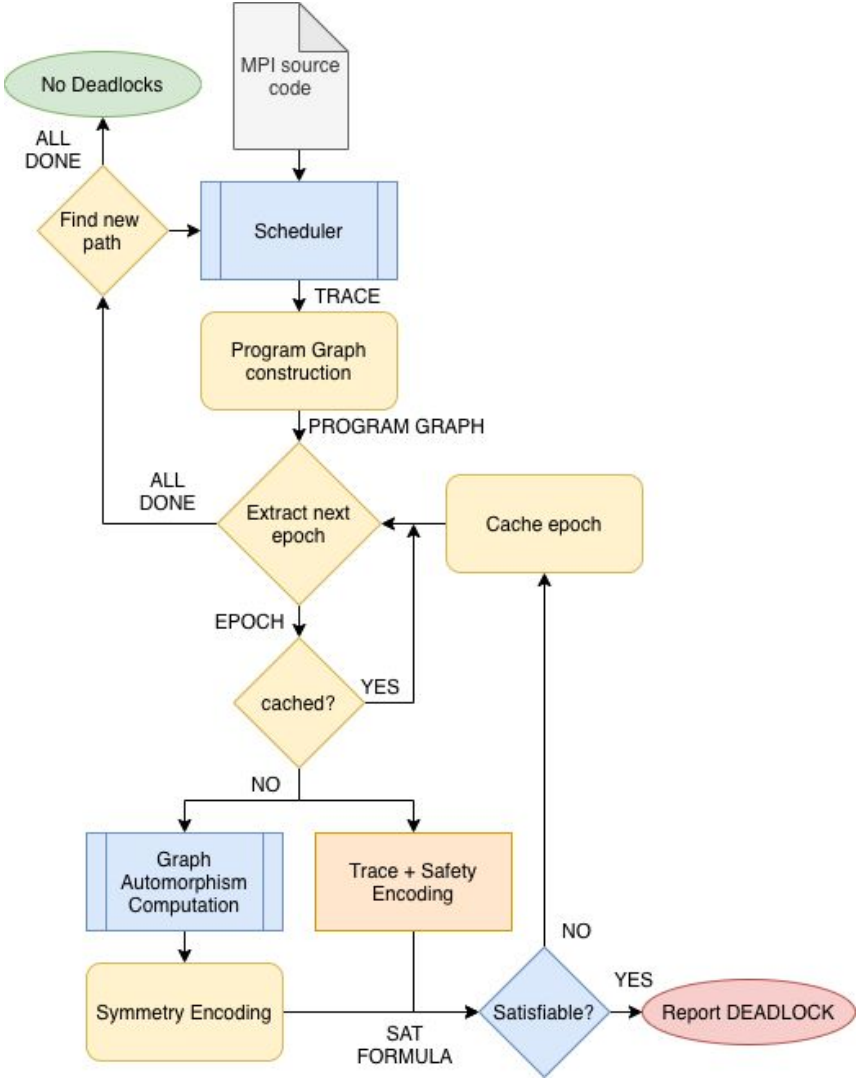
Multiple traces need to be generated and verified for multi-path programs

Generating a trace uses SAT for path feasibility[1] -- important **bottleneck**

Symmetry and epochs help to drastically reduce runtime here as well

[1] Khanna, Dhriti, et al. "**Dynamic symbolic verification of MPI programs.**" *International Symposium on Formal Methods*. Springer, Cham, 2018.

Tool Flow Chart



Thank you!