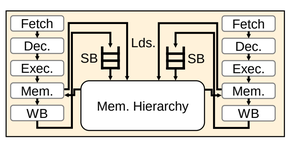
PipeProof: Automated Memory Consistency Proofs for Microarchitectural Specifications

Yatin A. Manerkar, Daniel Lustig (NVIDIA), Margaret Martonosi, Aarti Gupta

PRINCETON UNIVERSITY

Nominated for Best Paper

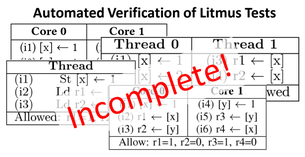
The Need for All-Program MCM Verification



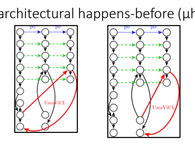
== TSO?

- Does a microarchitecture obey its MCM for all programs?
- Prior work is either incomplete or manual verification
- Can we **automatically** prove correctness for **all** programs?

Automated Verification of Litmus Tests



Microarchitectural happens-before (μhb) graphs



[Lustig et al. MICRO-47, ...]

All-Program Verification using Proof Assistants

[Vijayaraghavan et al. CAV 2015, Choi et al. ICFP 2017]
[Image: Coq Development Team, LGPL]

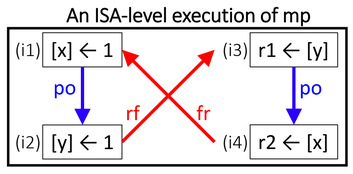
ISA-Level MCMs and Microarchitectural Ordering Specifications

- ISA-level MCMs defined in terms of acyclicity, irreflexivity, etc. of relational patterns [Alglave et al. TOPLAS 2014]
- Microarchitectural ordering specifications defined as set of μspec axioms [Lustig. et al. ASPLOS 2016]

ISA-Level MCM Specification of SC

```
acyclic (po U co U rf U fr)
```

An ISA-level execution of mp



Nodes: Instructions
Edges: ISA-level relations

Message passing (mp) litmus test

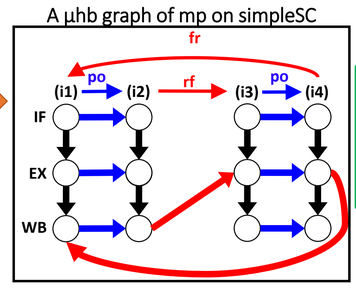
Core 0	Core 1
(i1) [x] ← 1	(i3) r1 ← [y]
(i2) [y] ← 1	(i4) r2 ← [x]
Under SC: Forbid r1=1, r2=0	

μspec Microarchitectural Ordering Specification

```
Axiom "Fetch_is_FIFO":
... EdgeExists ((i1, IF), (i2, IF))
      => AddEdge ((i1, EX), (i2, EX)).

...
```

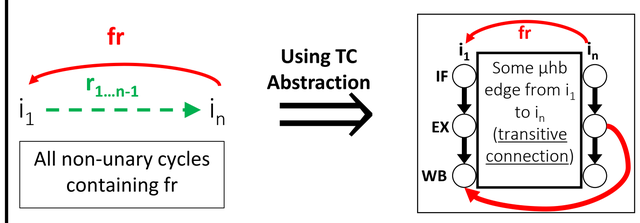
A μhb graph of mp on simpleSC



Nodes: Instr. sub-events
Edges: Happens-before relations

Cyclic graph => Unobservable
Acyclic graph => Observable

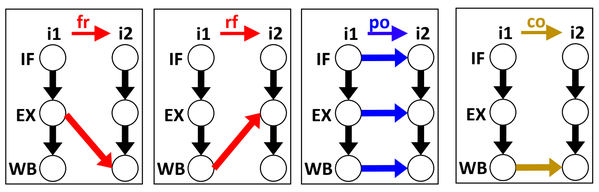
The Transitive Chain (TC) Abstraction



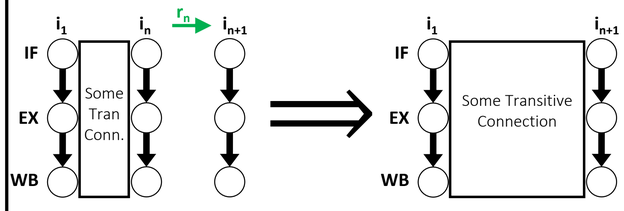
- Abstractions enable a finite representation of an infinite set of executions
- PipeProof's novel TC Abstraction represents sequence (chain) of ISA-level edges as μhb edge (transitive connection) from start to end of chain
- Intermediate instructions in chain are not explicitly modelled
- Verification of **infinite** number of ISA-level cycles => verification across a **finite** number of transitive connections
- Microarchitectural support of abstraction automatically proven as a supporting proof

TC Abstraction Support Proof

- Ensure that ISA-level pattern and μarch. support TC Abstraction
- Proof is inductive
- Base case: Do initial ISA-level edges guarantee a transitive connection?

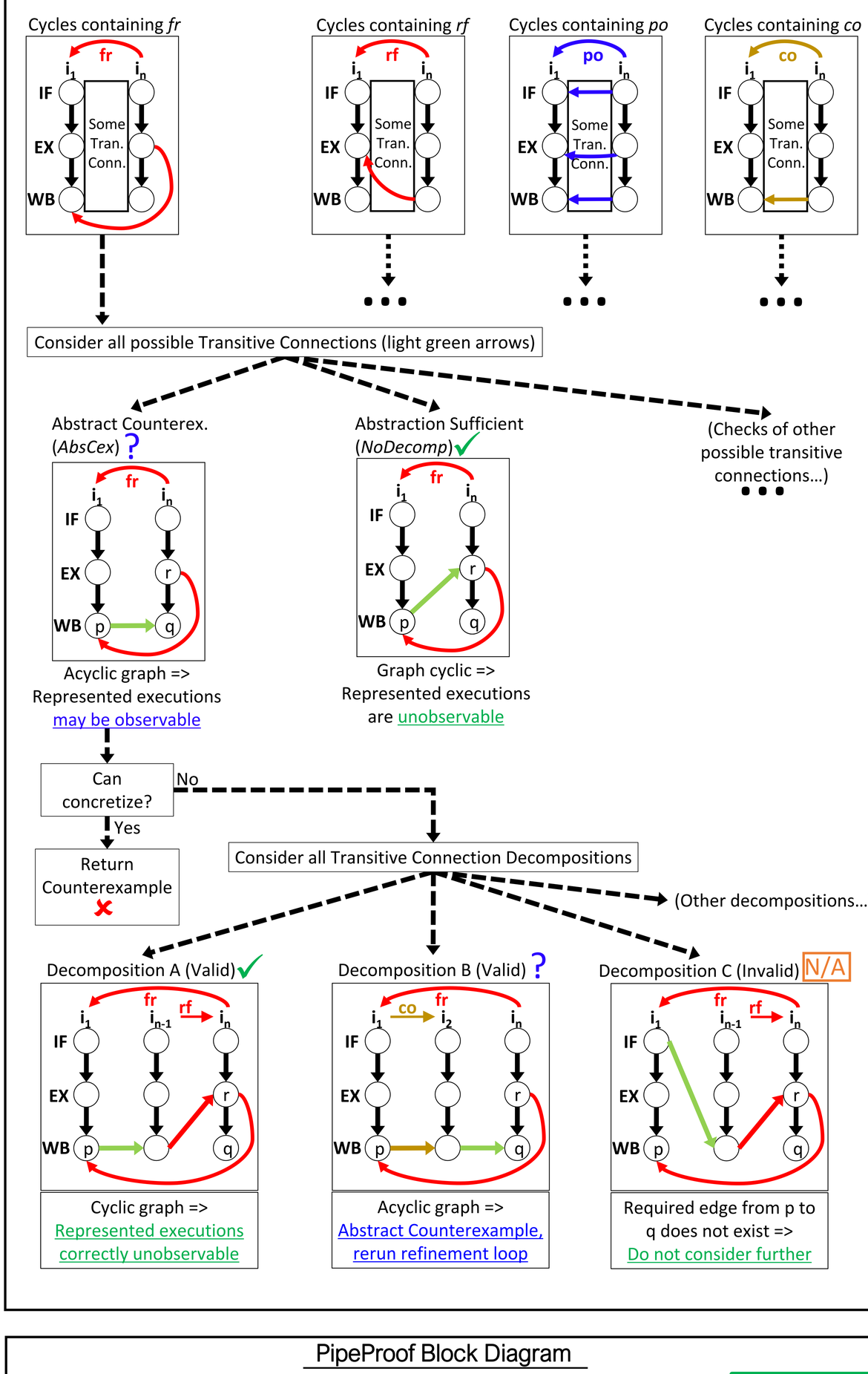


- Inductive case: Does extending the transitive chain extend the transitive connection?



- If proof fails, μarch may be buggy; can check for cyclic counterexamples with a bounded search

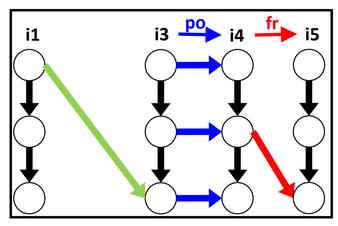
Microarchitectural Correctness Proof



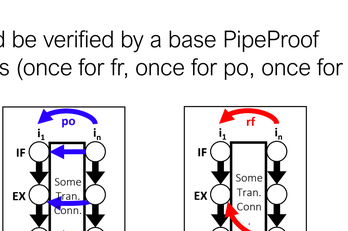
Chain Invariants

- Abstractly represent repeated ISA-level edge patterns
- Sometimes needed for refinement loop to terminate
- Inductively proven by PipeProof before their use elsewhere

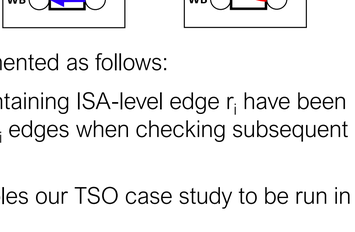
Abstract Counterexample



Repeating ISA-Level Pattern

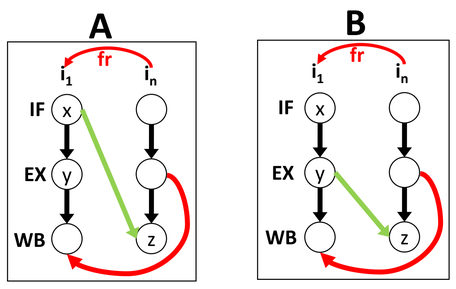


Chain Invariant Applied



Covering Sets Optimization

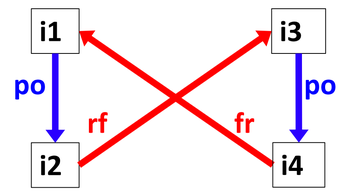
- PipeProof must conduct verification across all possible transitive connections
- Each decomposition creates a new set of transitive connections
- This can quickly lead to a case explosion
- The Covering Sets Optimization eliminates redundant transitive connections



- Graph A has a transitive connection from x to z
- Graph B has a transitive connection from y to z, but also has an edge from x to z through transitivity
- Correctness of A => Correctness of B, because B contains all of A's edges + an edge from y to z
- B does not need to be explicitly checked, and is eliminated by the Covering Sets Optimization

Memoization Optimization

- Base PipeProof algorithm examines some ISA-level cycles multiple times
- Memoization eliminates redundant checks of cycles that have already been verified



- The above cycle would be verified by a base PipeProof implementation 3 times (once for fr, once for po, once for rf)

- Optimization is implemented as follows:

If all ISA-level cycles containing ISA-level edge r_i have been verified, do not peel off r_i edges when checking subsequent ISA-level cycles

- This optimization enables our TSO case study to be run in under an hour

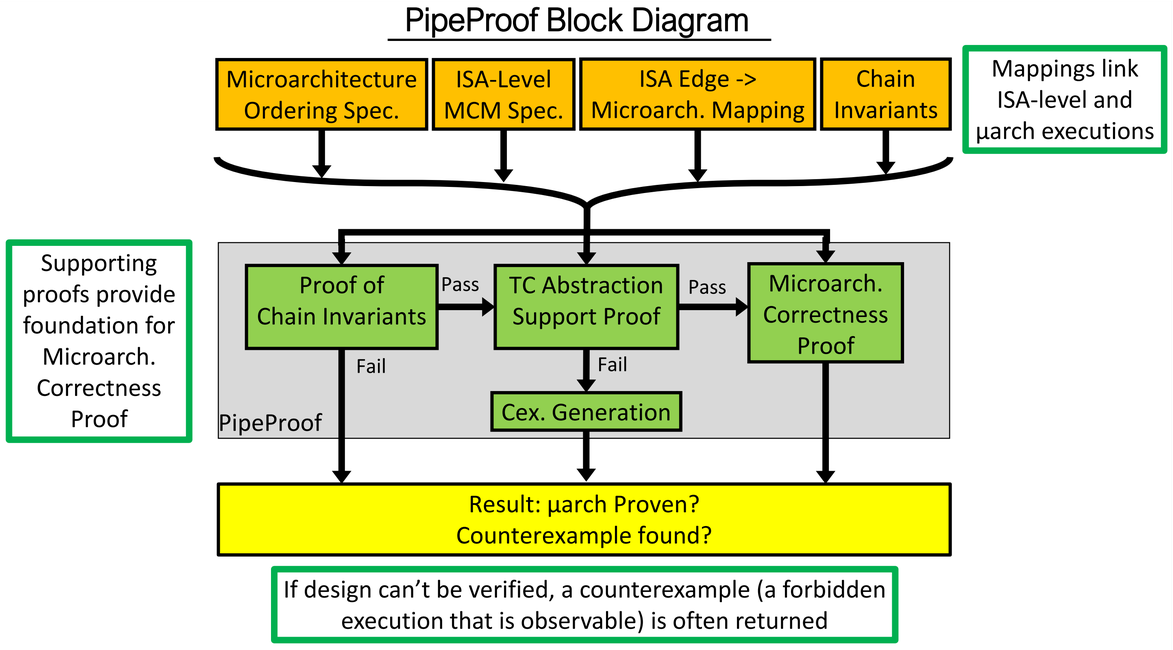
Results

- Ran PipeProof on two microarchitectures
- simpleSC (SC) and simpleTSO (TSO)
- 3-stage in-order pipelines
- simpleTSO relaxes Write->Read ordering

Configuration	simpleSC	simpleTSO
Without Optimizations	225.9 sec	Timeout
With Covering Sets	36.4 sec (6.2x speedup)	19885.4 sec (≈ 331 mins)
With Covering Sets + Memoization	19.1 sec (11.8x speedup)	2449.7 sec (≈ 41 mins) (8.1x speedup)

- simpleTSO is infeasible without optimizations, but becomes feasible with Covering Sets Optimization
- With Covering Sets + Memoization, simpleSC verified in under 20 seconds and simpleTSO verified in under 41 mins

PipeProof Block Diagram



Conclusions

- PipeProof: Automated **All-Program** Microarchitectural Memory Consistency Verification
  - User need only provide ISA-level and μarch models, mappings, and chain invariants
  - Designers no longer need to choose between completeness and automation
- **Transitive Chain Abstraction** allows inductive modelling and verification of the infinite set of all possible executions
  - Abstraction is automatically refined as necessary to prove correctness
- Verified simple microarchitectures implementing SC and TSO in < 1 hour!
  - Covering Sets Optimization and Memoization greatly reduce runtime

Code available at
https://github.com/ymanerka/pipeproof