Predicate Abstraction for Relaxed Memory Models

Andrei Dan
Martin Vechev

ETH Zurich

Yuri Meshman
Eran Yahav

Technion

Sequential Consistency

We expect programs to have "interleaving semantics"

"The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program."
– Leslie Lamport, 1973

Dekker's Algorithm for Mutual Exclusion

```
Thread 0:
flag[0] := true
while flag[1] = true {
  if turn ≠ 0 {
    flag[0] := false
    while turn ≠ 0 { }
    flag[0] := true
  }
}
// critical section
turn := 1
flag[0] := false
```

```
Thread 1:
flag[1] := true
while flag[0] = true {
  if turn ≠ 1 {
    flag[1] := false
    while turn ≠ 1 { }
    flag[1] := true
  }
}
// critical section
turn := 0
flag[1] := false
```

Specification: mutual exclusion over critical section
Memory Model: Intel's x86 (one of the strongest models)

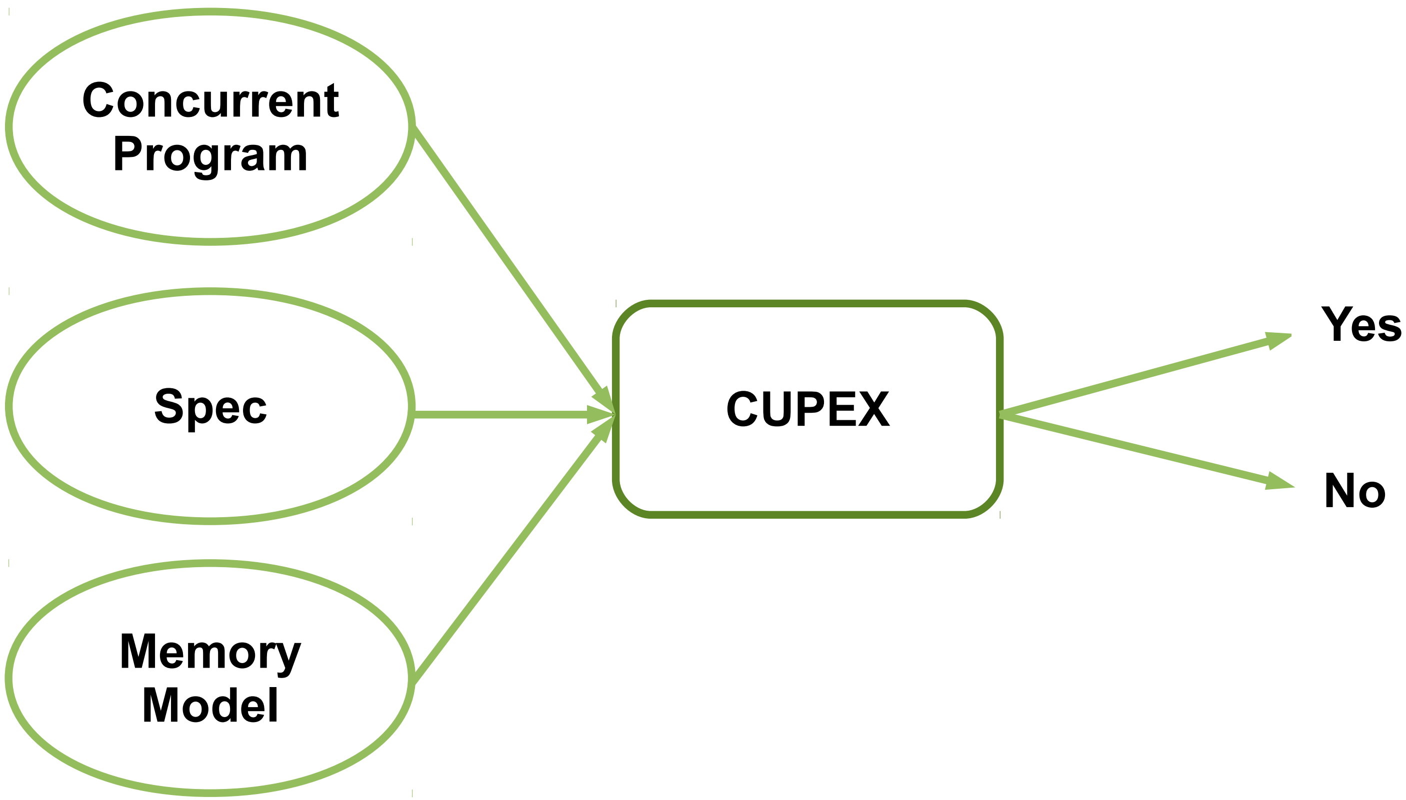
Correct Dekker Algorithm

```
Thread 0:
flag[0] := true
fence
while flag[1] = true {
  if turn ≠ 0 {
    flag[0] := false
    while turn ≠ 0 { }
    flag[0] := true
    fence
  }
}
// critical section
turn := 1
flag[0] := false
```

```
Thread 1:
flag[1] := true
fence
while flag[0] = true {
  if turn ≠ 1 {
    flag[1] := false
    while turn ≠ 1 { }
    flag[1] := true
    fence
  }
}
// critical section
turn := 0
flag[1] := false
```

Specification: mutual exclusion over critical section
Memory Model: Intel's x86 (one of the strongest models)

Goal – Automatic Verification of Concurrent Programs on RMM



Little work for infinite state verification

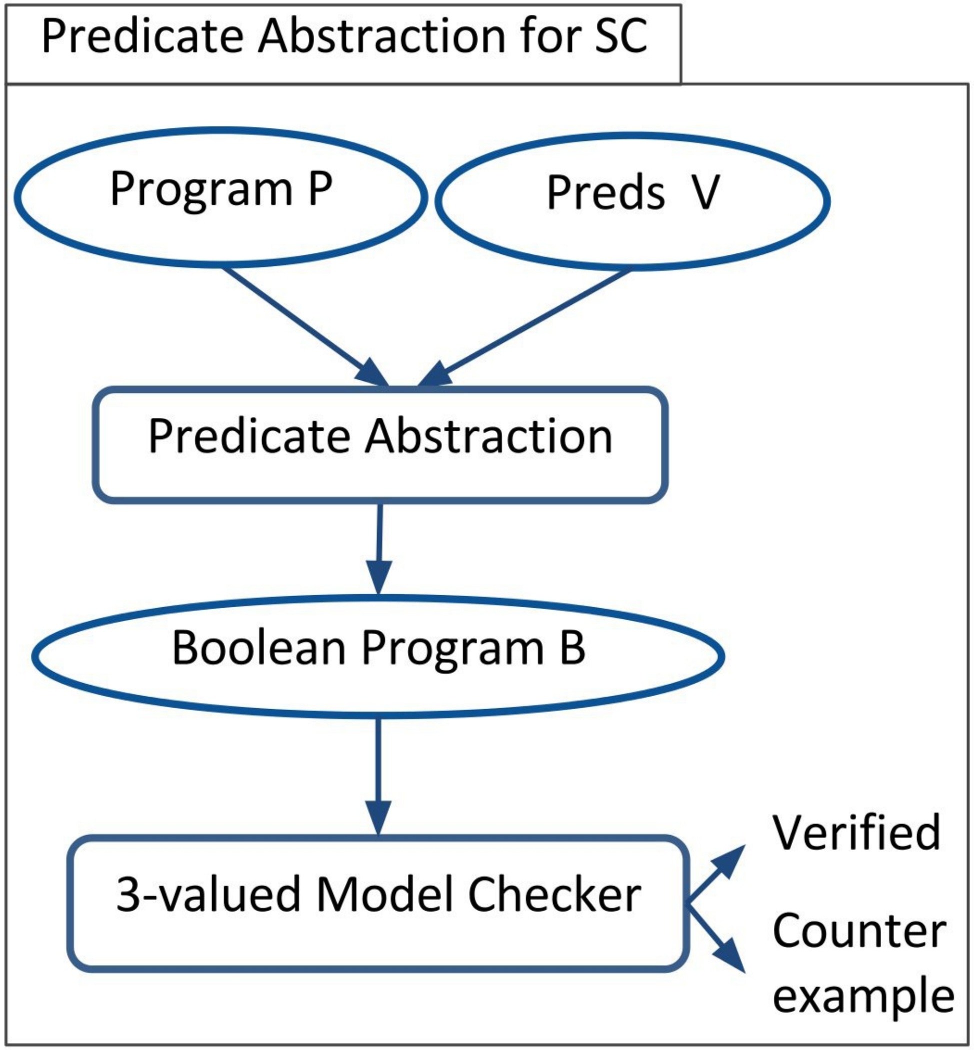
Technique: Predicate Abstraction

- Successful for sequential program analysis:
  - Original by Graf and Saidi (CAV' 96)
  - Used by Microsoft's SLAM for device drivers

- Some work for SC concurrent programs:
  - Kroening et al. (CAV' 11)
  - Gupta et al. (CAV' 11)

How can we apply Predicate Abstraction to relaxed memory model verification ?

Classic Predicate abstraction



Key High-Level Idea: adapt proof

The hypothesis is that a program running on a relaxed memory model (RMM) has much in common with the sequentially consistent (SC) program and does not diverge arbitrarily.

Step 1: verify program on sequential consistency

Step 2: adapt the predicates used in SC proof to verify program under RMM

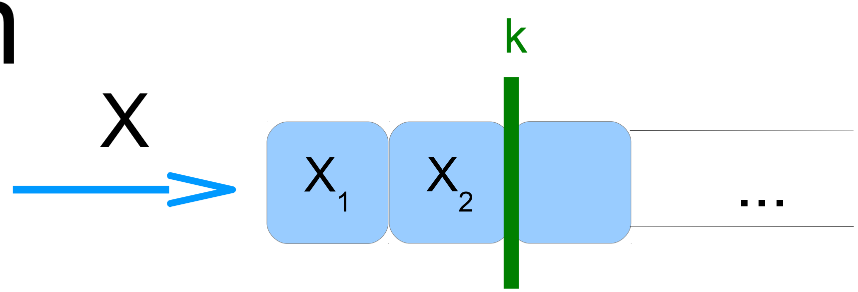
Our Approach: 3 steps

- Obtain SC proof: prove program P under SC using some predicates
- Obtain P_M : encode RMM effects into the program P and get an SC program P_M without RMM effects !
- Extrapolate predicates $Preds_M$ for P_M from SC proof

Step 1: Verify program under SC (using a known technique)

- Find a set of predicates Preds
- Build the boolean program B(P, Preds)
- Verify B satisfies property S under sequential consistency, that is: $B(P, Preds) \models_{SC} S$

Step 2: Encode RMM effects into program



PSO

- Choose a bound k for store buffers (sound)
- Encode store buffers as program variables
- Shared variable X gets encoded as:
  - X_{cnt} is a counter for the buffer
  - X_1, …. ,X_k for each buffer element

Encode program: example for k = 1

X – shared variable

load t = X →

```
if (X_cnt == 0)
  load t = X;
if (X_cnt == 1)
  t = X_1;
```

store X = t →

```
if (X_cnt == k)
  "overflow"
X_cnt ++;
if (X_cnt == 1)
  X_1 = t;
```

Step 3: Predicate Extrapolation: discover new predicates for RMM

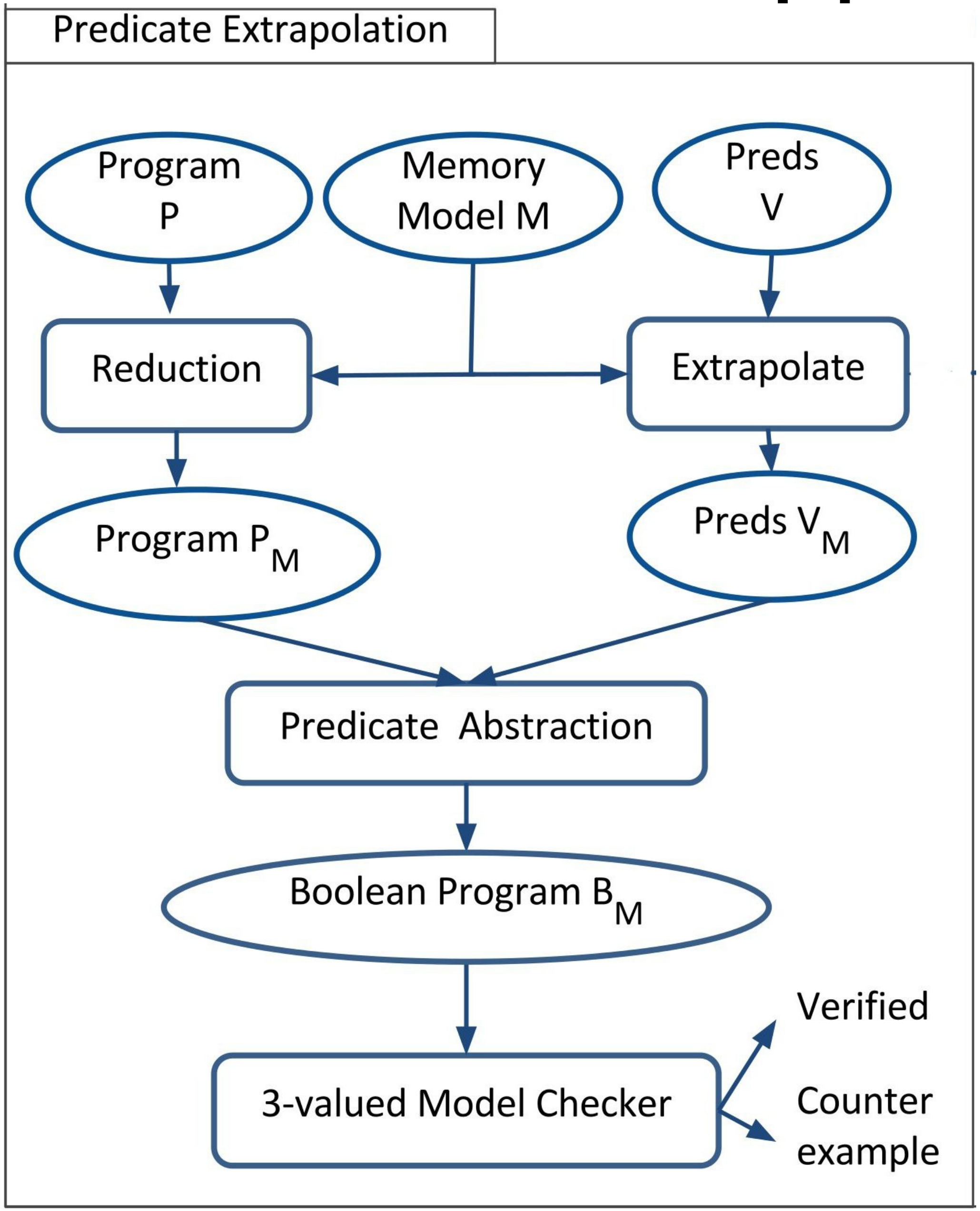
Key Idea: adapt the predicates used in the SC proof for the proof under RMM.

- buffer size (precision: enumerate all possible values)
- buffer elements (learned from SC predicates)

Predicate Extrapolation Example

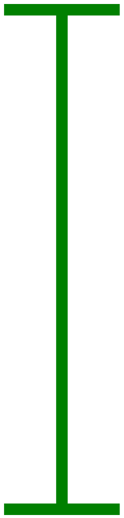
- $\forall$ X $\in$ shared variables, $\forall$ i = 0..k:
  - $(X_{cnt} = i)$ tracks buffer size
  - $(X_i = X_{i-1})$, $i \neq 0$ for flush actions
- $\forall$ p $\in$ $Preds_{SC}$, where p is "(X < Y)":
  - $(X_i < Y)$
  - $(X < Y_i)$

Our approach so far



This approach works for some programs but not for all programs we tried.

Why ?



Standard predicate abstraction machinery

The Problem

Building boolean program is exponential in the number of predicates. For some benchmarks, we cannot even build the boolean program !

For example, the process for Bakery continues after 10 hours...

What is the core problem ?

Problem: abstract transformer

Literals $q_i = p_i$ or $q_i = \neg p_i$, $p_i \in$ Preds

Cubes(Preds) = $\{q_1 \wedge \dots \wedge q_j\}$

|Cubes(Preds)| = $3^{|Preds|}$

$\forall$ st $\in$ Statements
  $\forall$ $p_i \in$ Preds
    f = wp(p_i, st)
    $\forall$ c $\in$ Cubes(Preds)
      if c $\Rightarrow$ f   //SMT call
        add c to the transformer

Key Idea

Reuse more information from the SC proof:

In addition to input predicates, extrapolate from the actual cubes that are used in the boolean program!

Cube Extrapolation Example

Cube in the SC boolean program B:

$(X >= 0) \wedge (X < Y)$ →

Potential cubes for RMM boolean program:

$(X_1 >= 0) \wedge (X_1 < Y)$

...

$(X_k >= 0) \wedge (X_k < Y)$

$(X >= 0) \wedge (X < Y_1)$

...

$(X >= 0) \wedge (X < Y_k)$

New abstract transformers

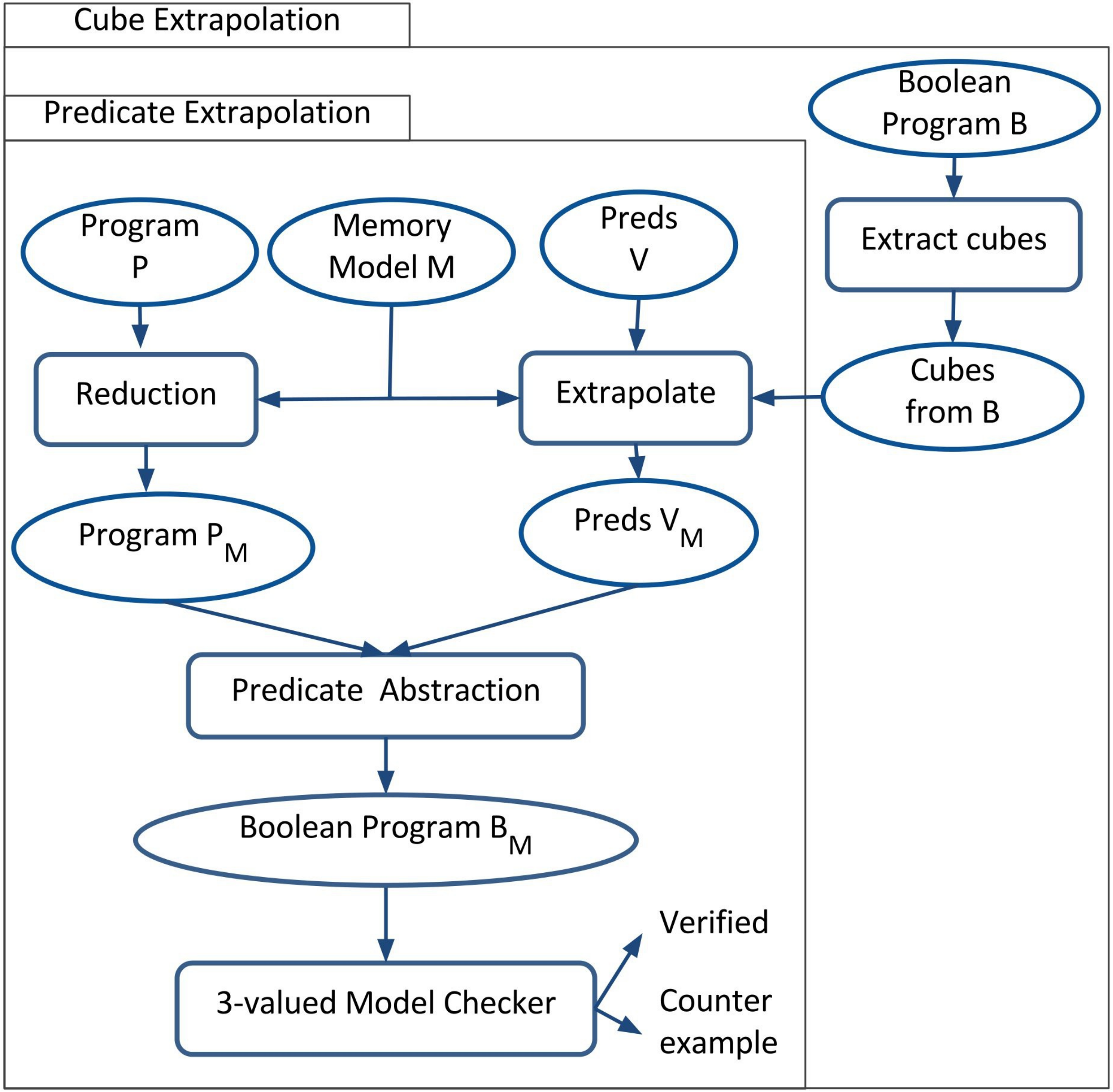
~~Cubes(Preds) = $\{q_1 \wedge \ldots \wedge q_j\}$~~

Cubes'(Preds$_M$) = { CubeExtrapolation(B) }

$|\text{Cubes'}(\text{Preds}_M)| \ll |\text{Cubes}(\text{Preds}_M)|$

$\forall$ st $\in$ Statements
  $\forall$ p_i $\in$ Preds
    f = wp(p_i, st)
    $\forall$ c $\in$ Cubes'(Preds$_M$)
      if c $\Rightarrow$ f //SMT call
        Add c to the transformer

Complete approach



Implementation

- Build the boolean program
  - bounded cube size search and cone of influence
  - Yices SMT solver

- Use a three-valued model checker
  - merge states after updates
  - partial concretization of assume conditions

Results: Predicate Extrapolation

		Build Boolean Program					Model check		
algorithm	memory model	# input preds	# SMT calls (K)	time (sec)	# cubes used	cube size	# states (K)	memory (MB)	time (sec)
Dekker	SC	7	0.7	0.1	0	1	14	6	1
	PSO	28	71	16	0		437	151	26
	TSO	26	60	14	0		433	147	19
Peterson	SC	7	0.6	0.1	2	2	7	3	1
	PSO	28	44	10	2		120	44	8
	TSO	26	36	8	2		231	81	11
ABP	SC	8	2	0.5	5	2	0.6	1	0.6
	PSO	15	20	4	5		2	3	1
	TSO	17	23	5	5		2	3	1
Szymanski	SC	20	16	3.3	1	2	12	6	2
	PSO	47	302	67	1		2,838	978	165
	TSO	51	405	95	1		3,251	1,128	199

Results: Cube Extrapolation

		Build Boolean Program							Model check		
algorithm	memory model	method	# input preds	# input cubes	# SMT calls (K)	time (sec)	# cubes used	cube size	# states (K)	memory (MB)	time (sec)
Queue	SC	Trad	7	-	20	5	50	4	1	2	1
	PSO	PE	15	-	5,747	1,475	412		1	4	1
		CE		99	98	17	99		11	6	2
	TSO	PE	16	-	11,133	2,778	412		12	4	1
		CE		99	163	31	99		12	7	2
Bakery	SC	Trad	15	-	1,552	355	161	4	20	8	2
	PSO	PE	38	-	-	T/O	-		-	-	-
		CE		422	9,018	1,773	381		979	375	104
	TSO	PE	36	-	-	T/O	-		-	-	-
		CE		422	7,048	1,386	383		730	285	121
Ticket	SC	Trad	11	-	218	51	134	4	2	2	1
	PSO	PE	56	-	-	T/O	-		-	-	-
		CE		622	15,644	2,163	380		193	123	40
	TSO	PE	48	-	-	T/O	-		-	-	-
		CE		622	6,941	1,518	582		71	67	545

Cube extrapolation can be used to verify the simpler programs, but is not needed, as PE works.

Related Work

- Atig et al. (CAV' 11)
  - code-to-code translation
  - bounds on store age or context switches
  - applied for detecting bugs, no verification
- Abdulla et al. (SAS' 12)
  - iterative predicate abstraction
  - rely only on CEGAR refinement
  - not reusing existing proofs

Conclusion

- New predicates discovered by extrapolating the predicates used in verifying the program under sequential consistency work

- New cubes discovered by extrapolating SC cubes are precise enough to satisfy the specification

Future work

- Other relaxed models
  - hardware, software
- When is proof extrapolation possible?
  - theoretical guarantees
- Refinement techniques
  - buffer size
  - counter example guided
  - enforce predicate set

Thank you!

Store Buffer Based Models

- Ex: PSO