Predicate Abstraction for Relaxed Memory Models

Andrei Dan
Martin Vechev

ETH Zurich

Yuri Meshman
Eran Yahav

Technion

Dekker's Algorithm

```
initial: flag[0] = false, flag[1] = false, turn = 0
```

```
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
```

```
spec: mutual exclusion over critical section
```

sequential consistency Yes

relaxed model x86 TSO No

Correct Dekker Algorithm

```
initial: flag[0] = false, flag[1] = false, turn = 0
```

```
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
```

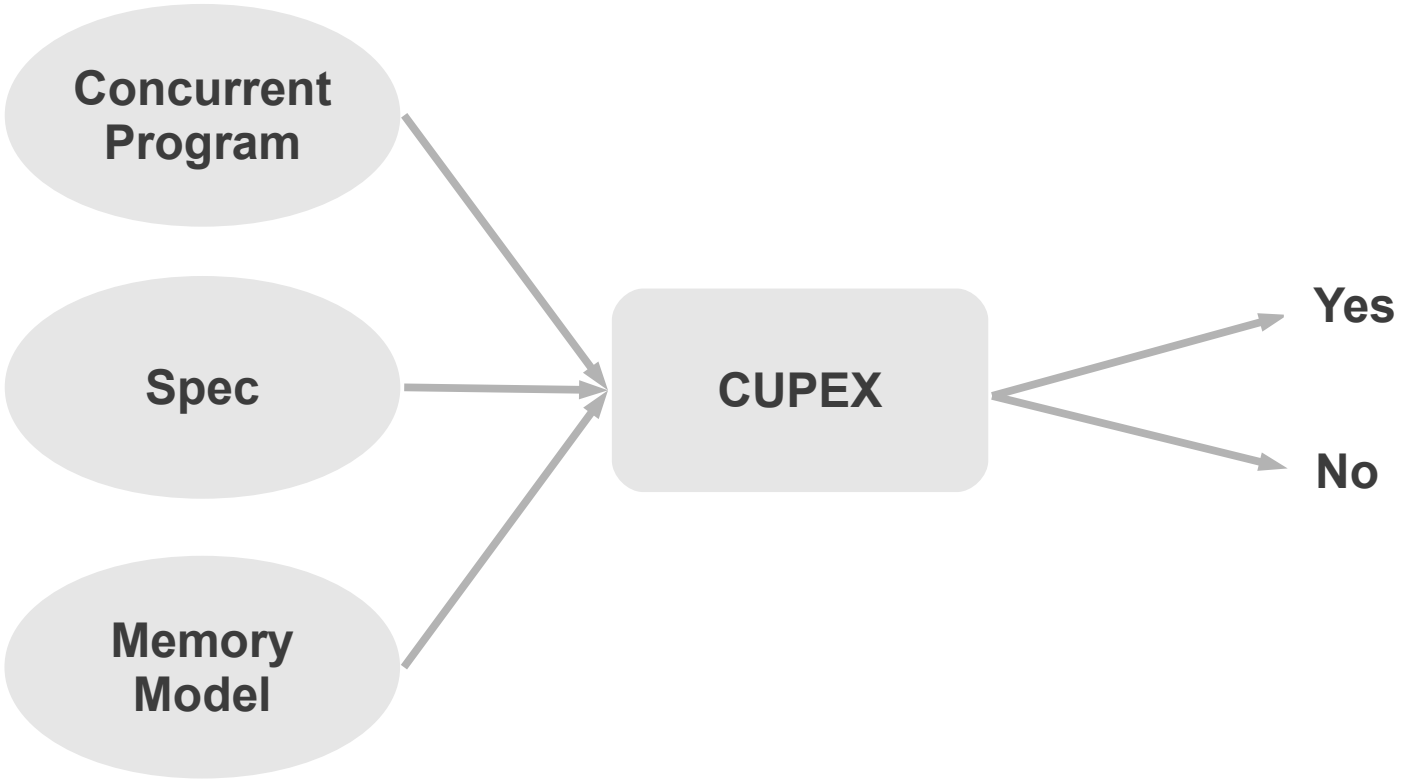
```
Thread 1:

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
```

```
spec: mutual exclusion over critical section
```

relaxed model x86 TSO Yes

Goal – automatic verification of concurrent programs on relaxed models



in this work we use predicate abstraction

Key Challenges

how to model the relaxed memory model effects ?

how to discover the necessary predicates
(for predicate abstraction) ?

Our approach

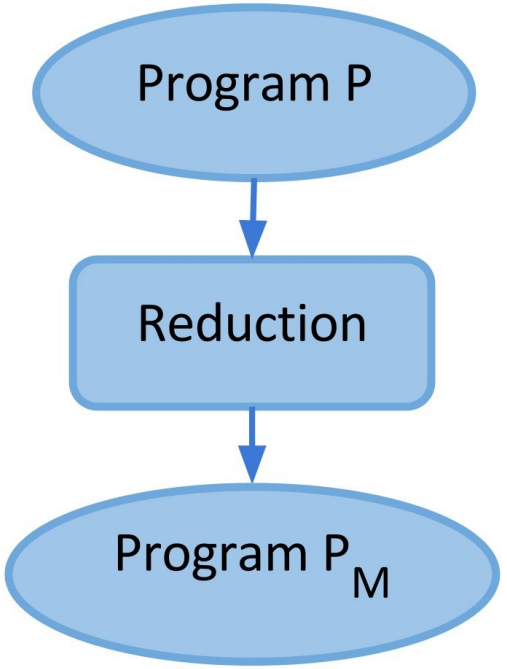
Step 1:

encode memory model semantics into the program P, obtaining a new program P_M

Step 2:

discover predicates necessary to verify P_M by extrapolating the proof of P obtained under sequential consistency

Step 1: Encode memory model effects



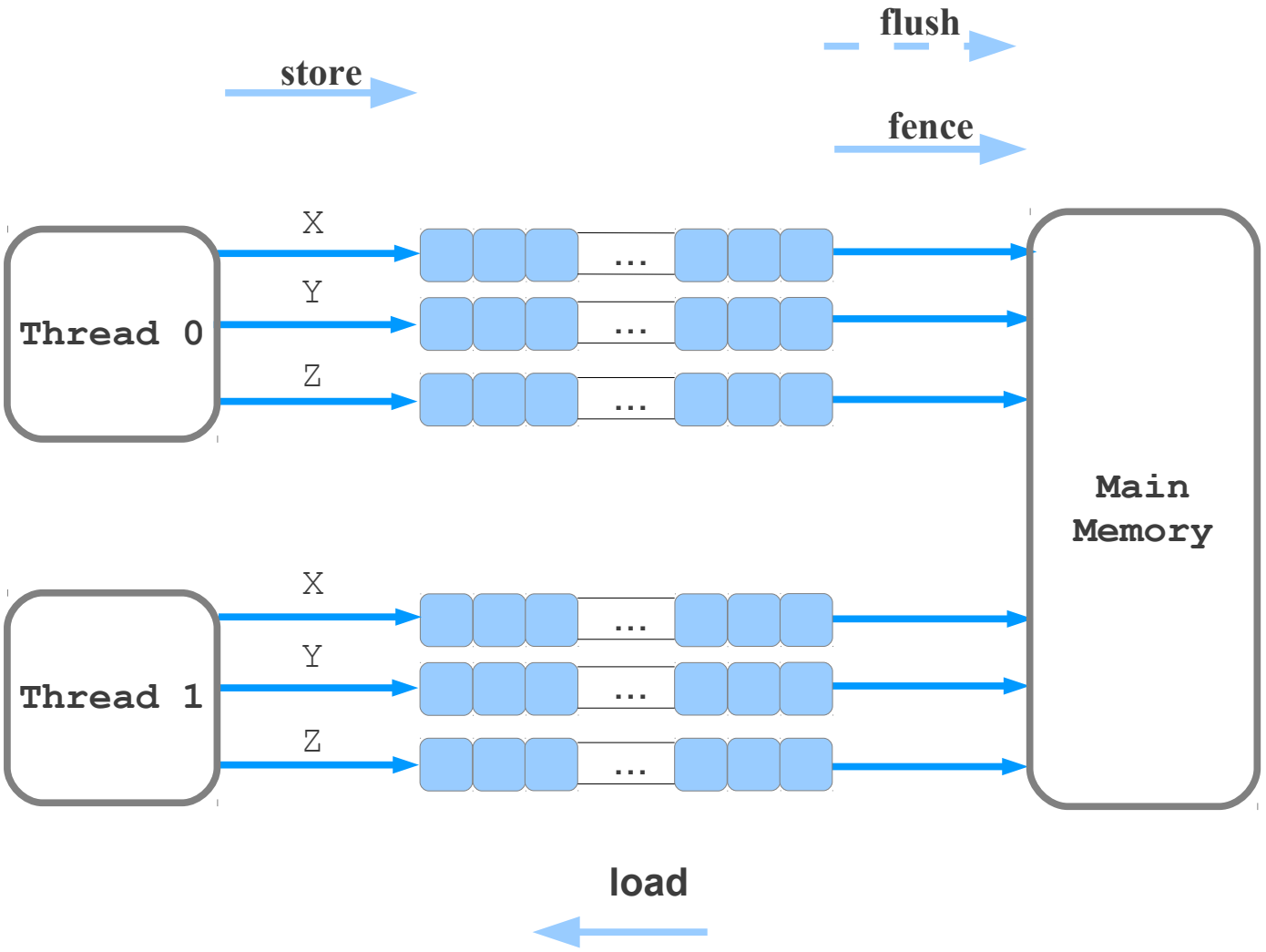
encode semantics of memory model M into program P, producing a new program P_M

property:

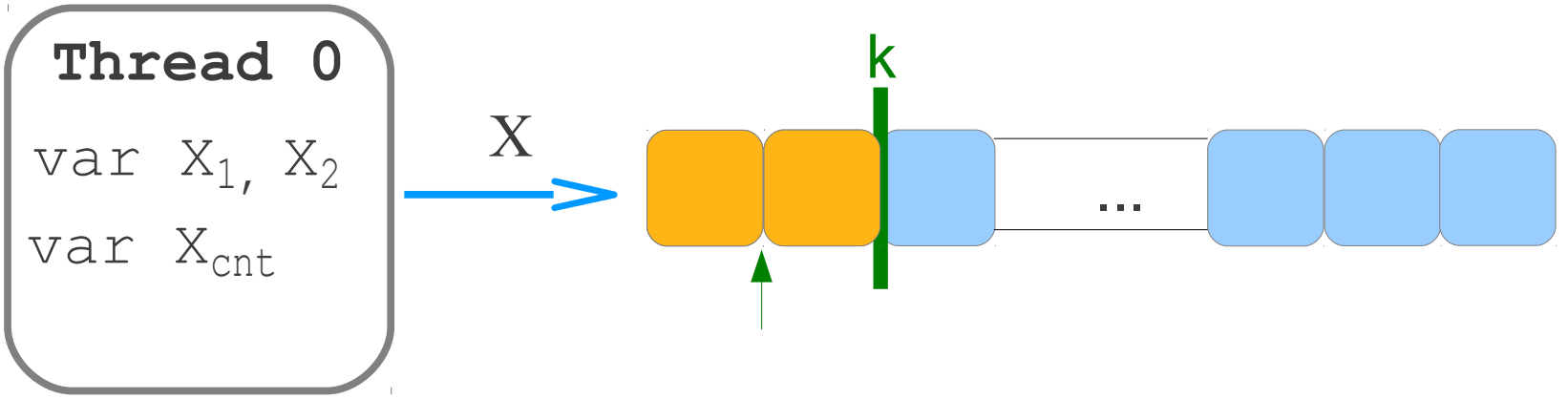
the behavior of P_M under sequential consistency is the same as the behavior of P running on M

Store-buffer-based models

PSO



Write buffer encoding



choose a bound k for store buffers (sound)

encode buffer elements as local variables: X_1, …., X_k

an additional local variable for current buffer size: X_{cnt}

Statements translation (k = 1)

X – shared var
t – local var

Program P

Program P_M

```
t := X
```

```
if X_cnt == 0
    t := X
if X_cnt == 1
    t := X_1
```

```
X := t
```

```
if X_cnt == k
    "overflow"
X_cnt := X_cnt + 1
if X_cnt == 1
    X_1 := t
```

Our approach

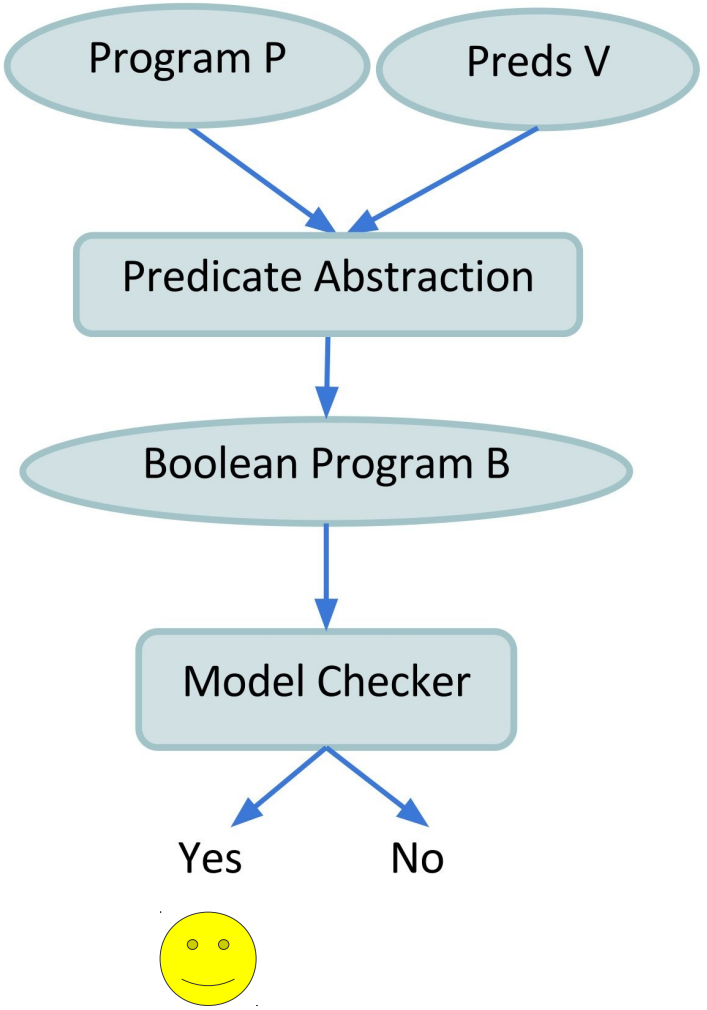
Step 1:

encode memory model semantics into the program P, obtaining a new program P_M

Step 2:

discover predicates necessary to verify P_M by extrapolating the proof of P obtained under sequential consistency

Predicate abstraction



find a set of predicates V

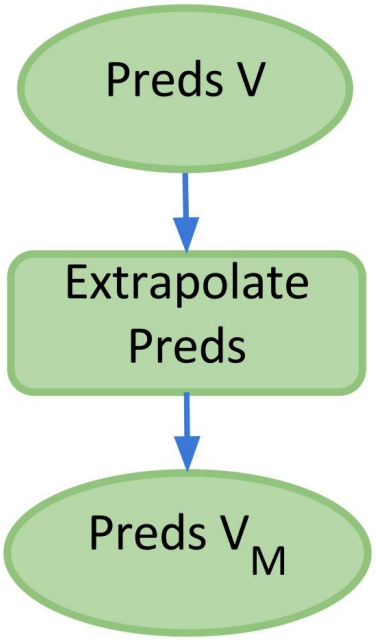
build the boolean program B(P, V)

Objective

verify program P_M using predicate abstraction

- discover predicate set V_M

Predicate Extrapolation Example



for all shared variable X, for all i = 0..k:

$(X_{cnt} = i)$ in V_M

$(X_i = X_{i-1})$ in V_M, $i \neq 0$

for all predicates p in V, we generate corresponding predicates in V_M

if p is "$(X < Y)$":

$(X_i < Y)$ in V_M

$(X < Y_i)$ in V_M

Problem

building boolean program is exponential in number of predicates

for some benchmarks, we cannot build boolean program...

keeps running after 10 hours...

what is the core problem ?

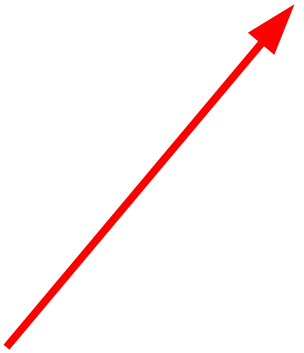
Core problem: abstract transformer

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

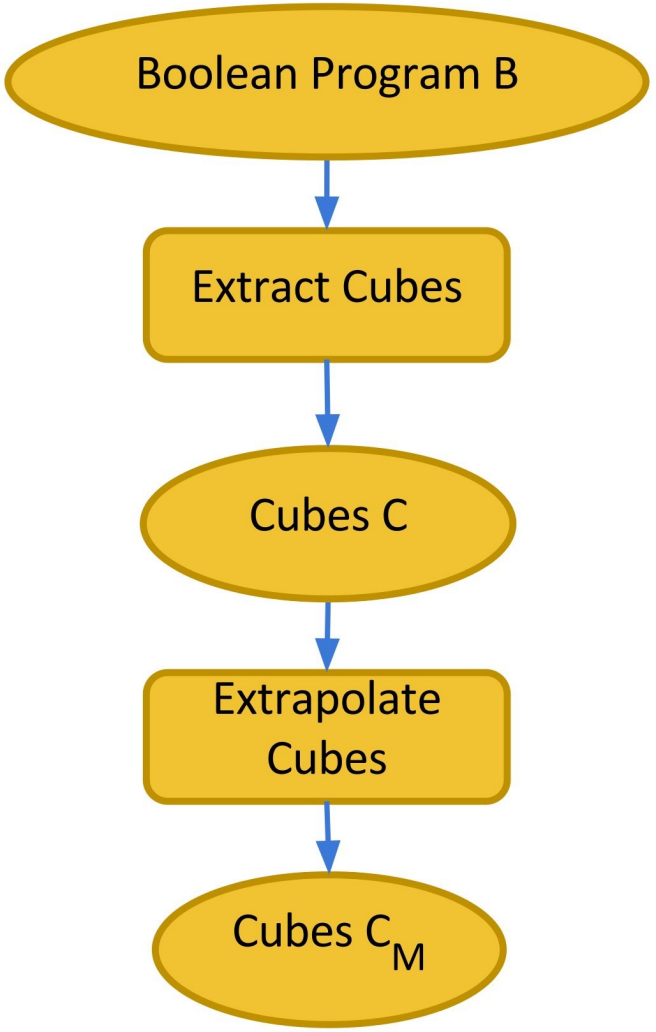
Cubes(V_M) $= \{q_{i_1} \wedge \ldots \wedge q_{i_j}, j \leq |V_M|\}$

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

```
for st ∈ Statements(P_M)
   for p ∈ V_M
      f = wp(p, st)
      for c ∈ Cubes(V_M)
         if c ⇒ f     //SMT call
            use c in the transformer
```



Key Idea: Cube Extrapolation



reuse more information from
the proof under sequential consistency

extrapolate from the actual cubes
that are used in the boolean program

Cube Extrapolation Example

$V = \{(X \geq 0),\ (X < Y)\}$

cube from boolean program under SC:

$(X \geq 0) \wedge (X < Y)$ 

candidate cubes for RMM:

$(X_1 \geq 0) \wedge (X_1 < Y)$

...

$(X \geq 0) \wedge (X < Y_1)$

...

New abstract transformers

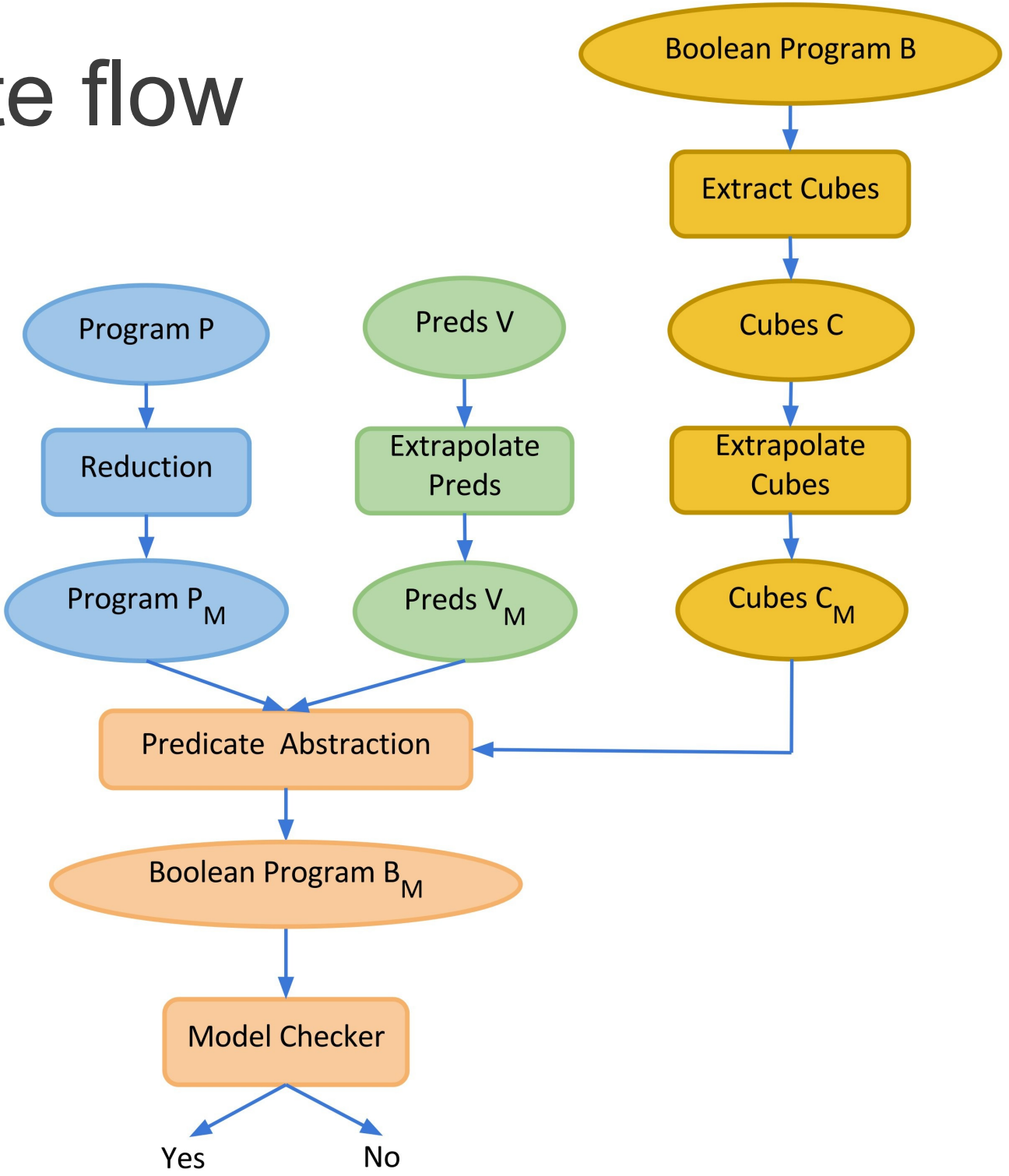
~~$Cubes(V_M) = \{q_{i_1} \wedge \ldots \wedge q_{i_j},\ j \leq |V_M|\}$~~

$ECubes(V_M)$ = { CubeExtrapolation(Cubes(B)) }

$$|ECubes(V_M)| << |Cubes(V_M)|$$

```
for st ∈ Statements(P_M)
   for p ∈ V_M
      f = wp(p, st)
      for c ∈ ECubes(V_M)
         if c ⇒ f     //SMT call
            use c in the transformer
```

Complete flow



Implementation

build the boolean program

- bounded cube size search and cone of influence
- Yices SMT solver

use a three-valued model checker

verify safety properties

Results: Predicate Extrapolation

		Predicate Abstraction		Model check	
Algorithm	Memory model	# Predicates	Time (sec)	# States (K)	Time(sec)
Dekker	SC	7	0.1	14	1
	PSO	20	6	80	5
	x86 TSO	18	5	45	3
Peterson	SC	7	0.1	7	1
	PSO	20	3	31	3
	x86 TSO	18	3	25	2
ABP	SC	8	0.5	0.6	0.6
	PSO	15	4	2	1
	x86 TSO	17	5	2	1
Szymanski	SC	20	3.3	12	2
	PSO	35	33	61	4
	x86 TSO	37	35	61	5

Results: Cube Extrapolation

		Predicate Abstraction		Model check	
Algorithm	Memory model	# Predicates	Time (sec)	# States (K)	Time(sec)
Queue	SC	7	5	1	1
	PSO	15	1,457	11	1
			17	11	2
	x86 TSO	16	2,778	12	1
			31	12	2
Bakery	SC	15	161	20	0.6
	PSO	38	T/O	-	-
			1,773	979	104
	x86 TSO	36	T/O	-	-
			1,386	730	121
Ticket	SC	11	134	2	2
	PSO	56	T/O	-	-
			2,163	193	40
	x86 TSO	48	T/O	-	-
			1,518	71	54

Comment

proof extrapolation worked for all benchmarks

for “Queue”, automatic extrapolation was imprecise on the first run

approach:

strengthen the set of predicates V for SC and rerun extrapolation

Future work

- apply standard abstract interpretation

  numerical domains: polyhedra, octagon,...

- compare with refinement techniques for predicate discovery

- handle unbounded buffer size

- theoretical guarantees on proof extrapolation

- handle other relaxed models: C++, Power, ARM

Conclusion

- a tool based on predicate abstraction for concurrency verification under relaxed models: x86 TSO and PSO
- automatically verified several challenging concurrent algorithms (finite and infinite state) for both x86 TSO and PSO
- key idea: discover new proof for a weaker memory model by extrapolating from the program proof under SC