Automatic Verification of RMA Programs via Abstraction Extrapolation

Cedric Baumann[1], Andrei Marian Dan[1], Yuri Meshman[2],

Torsten Hoefler[1], Martin Vechev[1]

[1] Department of Computer Science, ETH Zurich, Switzerland

[2] IMDEA Software Institute, Madrid, Spain

Remote Memory Access (RMA) Networks

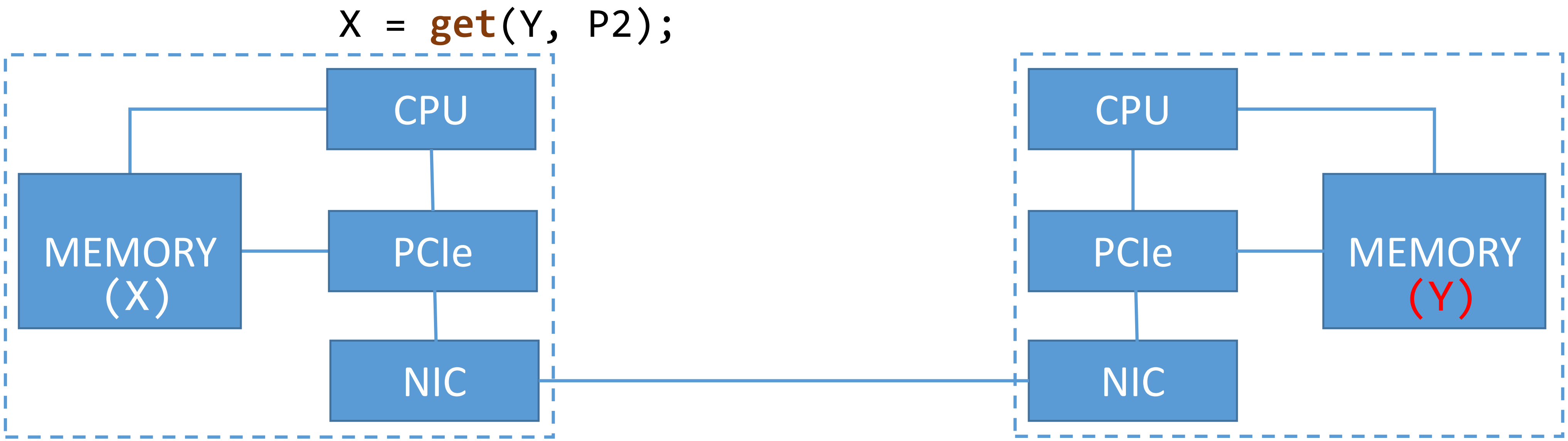
High-Performance Computing

Modern datacenters

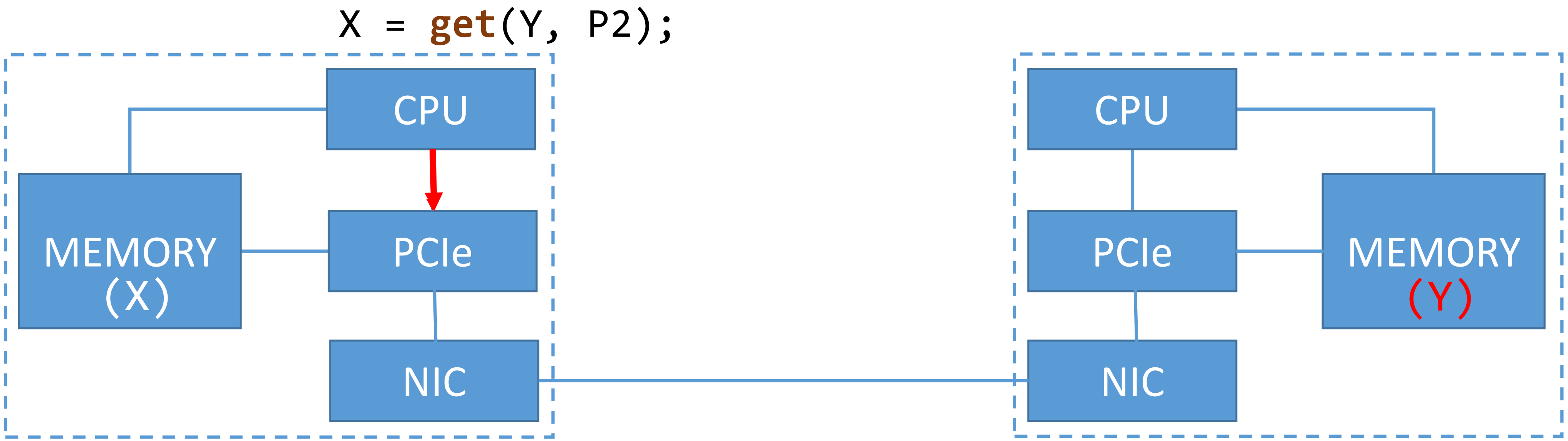
Widely supported (**Cray Aries** and **Gemini**, **Infiniband**, **IBM Blue Gene** and **Percs**)

Remote Memory Access (RMA) Networks

Remote Memory Access (RMA) Networks



Remote Memory Access (RMA) Networks

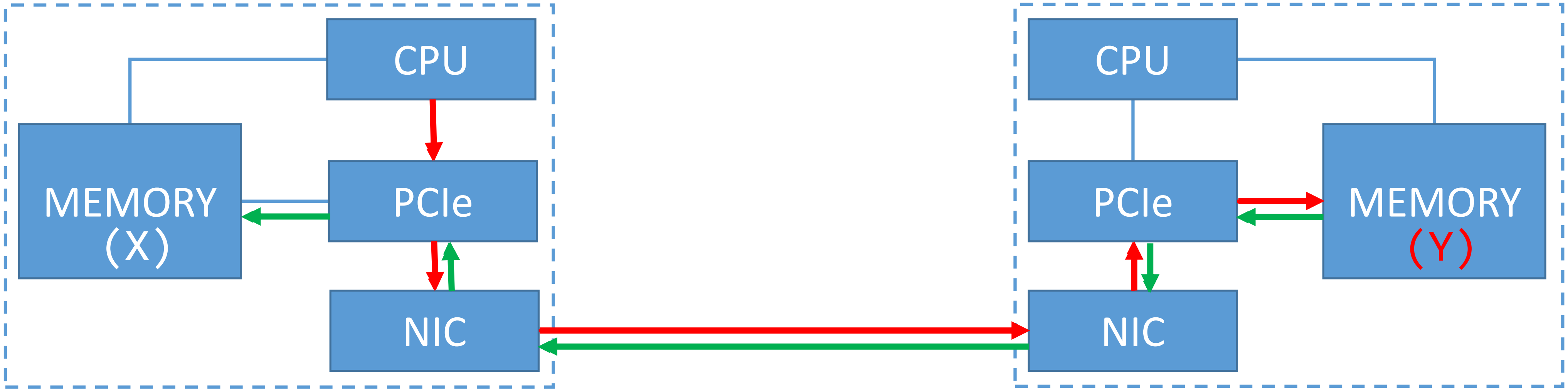


Remote Memory Access (RMA) Networks

Remote Memory Access (RMA) Networks

Remote Memory Access (RMA) Networks

Remote Memory Access (RMA) Networks



Low latency High bandwidth

Goal

Given an infinite-state program P running on an RMA network and a safety specification S, does P satisfy S under RMA?

$$P \vDash_{RMA} S$$

RMA asynchronous executions determine a weak-consistency memory model, more relaxed than x86 TSO, PSO, RMO

```
Process 1:

shared X = 1;
```

```
Process 2:

shared Y = 2, Z = 0;
local a;

put(X, P1, Y);
store Y = 3;
Z = get(X, P1);
load a = Z;
```

```
assert final (a != 3);
```

```
Process 1:

shared X = 1;
```

```
Process 2:

shared Y = 2, Z = 0;
local a;

put(X, P1, Y);
store Y = 3;
Z = get(X, P1);
load a = Z;
```

```
assert final (a != 3);
```

Sequential Consistency (SC)	Yes

```
Process 1:

shared X = 1;
```

```
Process 2:

shared Y = 2, Z = 0;
local a;

put(X, P1, Y);
store Y = 3;
Z = get(X, P1);
load a = Z;
```

```
assert final (a != 3);
```

Sequential Consistency (SC)	Yes
Remote Memory Access (RMA)	No

```
Process 1:

shared X = 1;
```

```
Process 2:

shared Y = 2, Z = 0;
local a;

put(X, P1, Y);
store Y = 3;
Z = get(X, P1);
load a = Z;
```

```
assert final (a != 3);
```

Sequential Consistency (SC)	Yes
Remote Memory Access (RMA)	No

```
Process 1:

shared X = 1;
```

```
Process 2:

shared Y = 2, Z = 0;
local a;

put(X, P1, Y);
flush(P1);
store Y = 3;
Z = get(X, P1);
load a = Z;
```

```
assert final (a != 3);
```

```
Process 1:

shared X = 1;
```

```
Process 2:

shared Y = 2, Z = 0;
local a;

put(X, P1, Y);
flush(P1);
store Y = 3;
Z = get(X, P1);
load a = Z;
```

```
assert final (a != 3);
```

Remote Memory Access (RMA)	Yes

```
Process 1:

shared X = 1;
```

```
Process 2:

shared Y = 2, Z = 0;
local a;

put(X, P1, Y);
flush(P1);
store Y = 3;
Z = get(X, P1);
load a = Z;
```

```
assert final (a != 3);
```

Remote Memory Access (RMA) Yes

This work

Main steps:

1. Prove that P satisfies S under SC:

$$P \vDash_{SC} S$$

2. Construct P' under SC that captures all behaviors of P under RMA:

$$P' \vDash_{SC} S \quad \Rightarrow \quad P \vDash_{RMA} S$$

3. Prove that $P' \vDash_{SC} S$

This work

Main steps:

1. Prove that P satisfies S under SC:

$$P \models_{SC} S$$

2. Construct P’ under SC that captures all behaviors of P under RMA:

$$P' \models_{SC} S \quad \Rightarrow \quad P \models_{RMA} S$$

3. Prove that $P' \models_{SC} S$

Key Idea: Extrapolate the abstraction of P under SC to an abstraction of P under RMA

Predicate Abstraction

Successful for sequential program analysis:

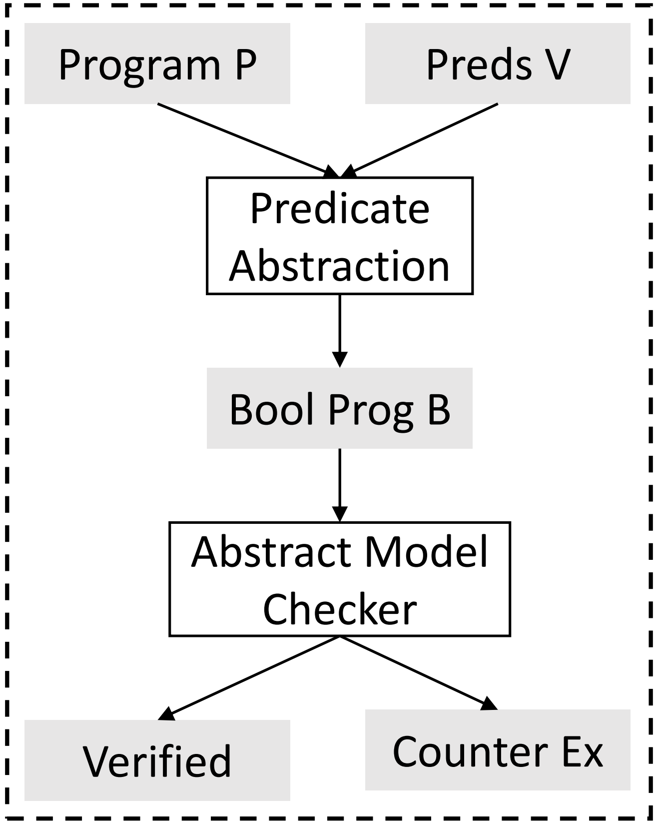
Original by Graf and Saidi (CAV '96)

Used by Microsoft's SLAM for device drivers (PLDI '01)

Work for SC concurrent programs and weak memory models (x86 TSO, PSO):

Kroening et al. (CAV '11), Gupta et al. (CAV '11), Dan et al. (SAS '13)

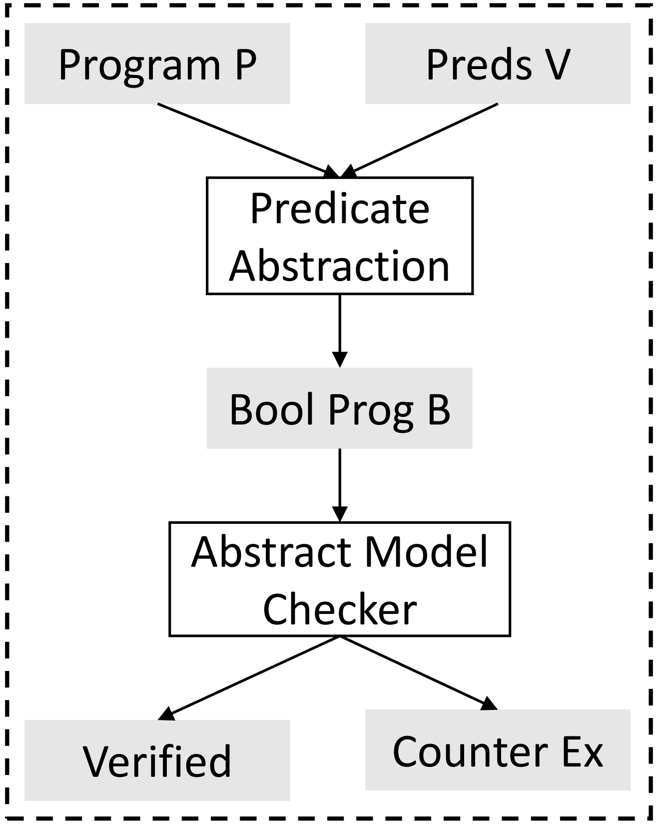
Classic Predicate Abstraction



Build a boolean program **B** that over-approximates the behaviors of **P**:

$$B \vDash_{SC} S \quad \Rightarrow \quad P \vDash_{SC} S$$

Classic Predicate Abstraction



Build a boolean program **B** that over-approximates the behaviors of **P**:

$$B \vDash_{SC} S \quad \Rightarrow \quad P \vDash_{SC} S$$

Using an abstract model checker, verify that **B** satisfies **S**:

$$B \vDash_{SC} S$$

Step 1: Verify program P under SC

Assume the RMA statements execute synchronously

```
put(Y, P1, X);  →  Y = X;
```

Step 1: Verify program P under SC

Assume the RMA statements execute synchronously

```
put(Y, P1, X);
```

$\longrightarrow$

```
Y = X;
```

Find a set of predicates V

Build the boolean program B that over-approximates P, using V

Verify that B satisfies the property S under sequential consistency

Step 2: Encode RMA effects into the program

Program P

↓

Reduction

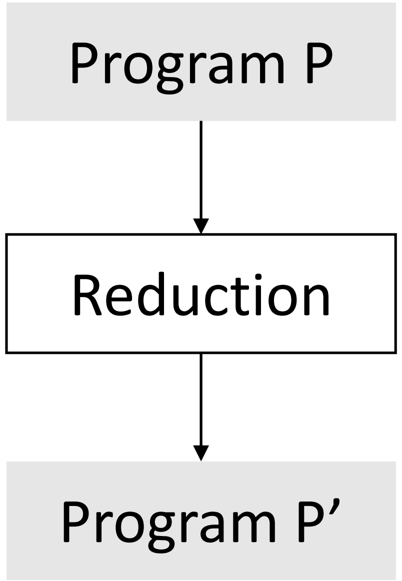
↓

Program P’

Reduce the problem of verifying P under RMA to the problem of verifying P’ under SC

$$P' \vDash_{SC} S \quad \Rightarrow \quad P \vDash_{RMA} S$$

Step 2: Encode RMA effects into the program




```
put(Y, P1, X);
```

Step 2: Encode RMA effects into the program

Program P

Reduction

Program P’

```
put(Y, P1, X);
```

```
if (!putActive)       //boolean flag
    putActive = true ;
    XSet = {X};       //set variable
else
    addToSet(XSet, X); //adds X to XSet
```

Example program P under RMA semantics

```
Process 1:

shared X = 0;

put(Y, P2, X);
store X = 1;
```

```
Process 2:

shared Y = 0;
local r;

load r = Y;
```

Example: Reduced program P’ under SC that captures the behaviors of P under RMA

```
Process 1:

shared X = 0;

//put(Y, P2, X);
putActive = true ;
XSet = {X};

//store X = 1;
store X = 1;
addToSet(Xset, X);
```

```
Process 2:

shared Y = 0;
local r;

//nondeterministic op
if (*)
    Y = randomElem(XSet);
    putActive = false;

//load r = Y;
load r = Y;
```

Theorem. P’ under SC **soundly approximates** P under RMA.

Step 3: Prove that P’ $\vDash_{SC}$ S

Find new predicates for program P’

Predicates for the boolean flags:

```
(putActive == true)
```

Step 3: Prove that P’ $\vDash_{SC}$ S

Find new predicates for program P’

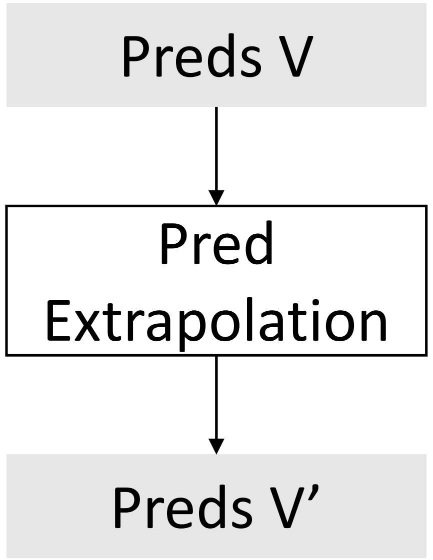
Predicates for the boolean flags:

`(putActive == true)`

Predicates for the set variables?

Step 3: Prove that P’ $\models_{SC}$ S

Idea: Discover predicates for P’ using the predicates for program P under SC.



Step 3: Prove that P’ $\vDash_{SC}$ S

Idea: Discover predicates for P’ using the predicates for program P under SC.

Preds V

↓

Pred Extrapolation

↓

Preds V’

`(X < 0)`

Step 3: Prove that P’ $\vDash_{SC}$ S

Idea: Discover predicates for P’ using the predicates for program P under SC.

Preds V

Pred Extrapolation

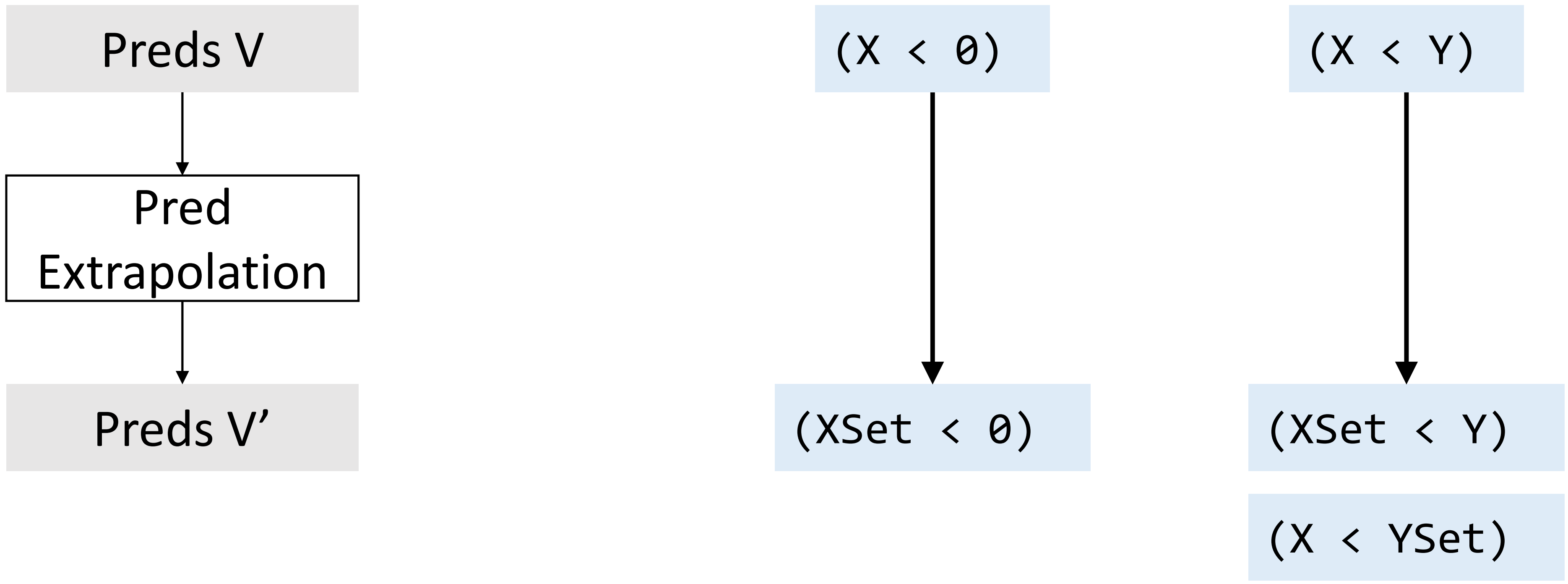
Preds V’

`(X < 0)`

`(XSet < 0)`

Step 3: Prove that P’ $\models_{SC}$ S

Idea: Discover predicates for P’ using the predicates for program P under SC.



Logic of the predicates (first attempt)

`(XSet < 0)`

- **true**, $\forall\ e \in XSet \mid e < 0$
- **false**, $\exists\ e \in XSet \mid e \geq 0$
- `*`, otherwise

Logic of the predicates (first attempt)

`(XSet < 0)`

- `true, ∀ e ∈ XSet | e < 0`
- `false, ∃ e ∈ XSet | e ≥ 0`
- `*, otherwise`

Problem: Would have to add the predicate `(XSet ≥ 0)` to track whether all elements of the set are greater than 0.

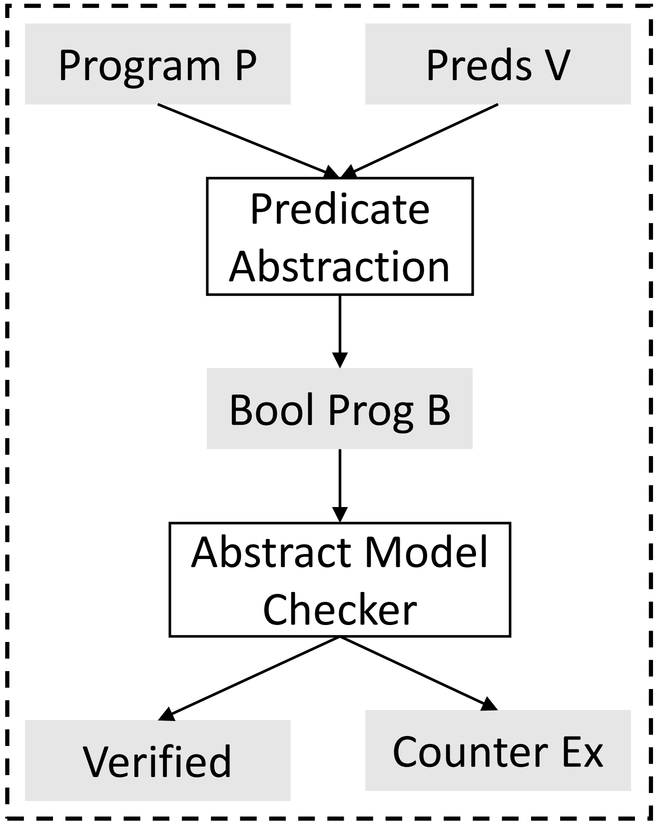
Logic of the predicates for the set variables

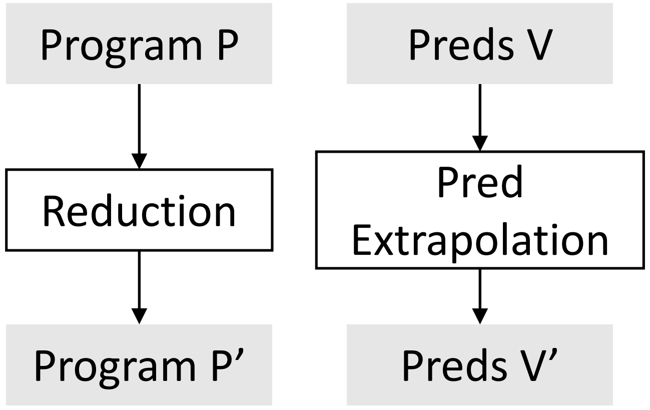
`(XSet < 0)`

- `true, ∀ e ∈ XSet | e < 0`
- `false, ∀ e ∈ XSet | e ≥ 0`
- `*, otherwise`

Solution: Refine the case when the predicate is **`false`**

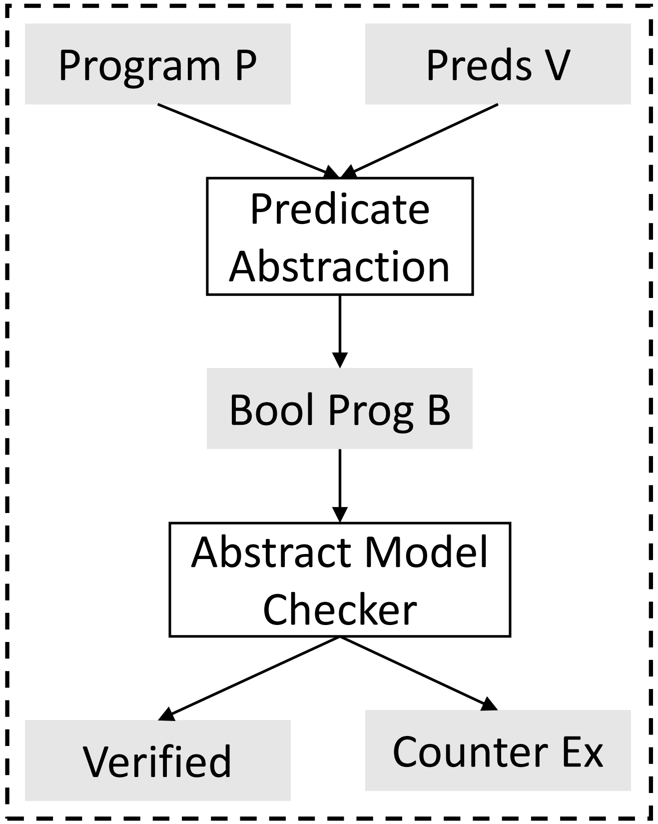
So far

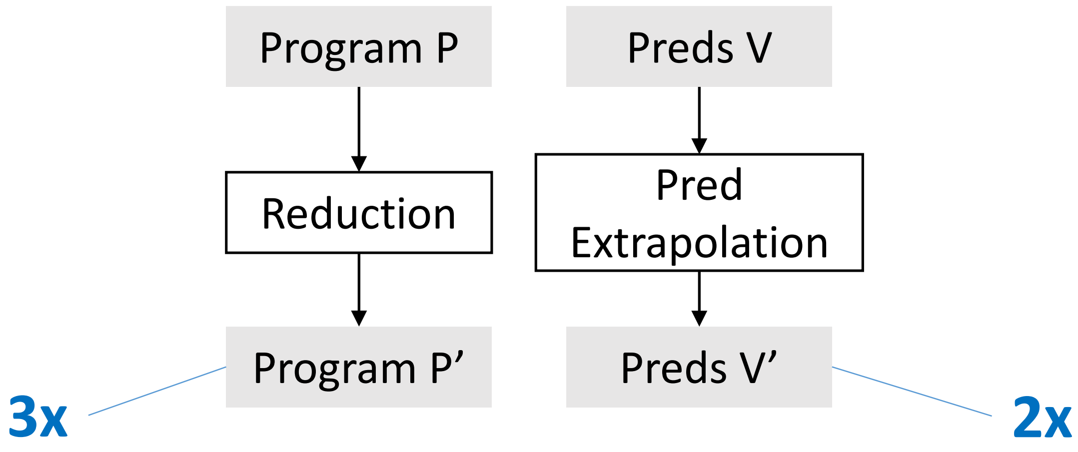




Prove that $P \vDash_{SC} S$

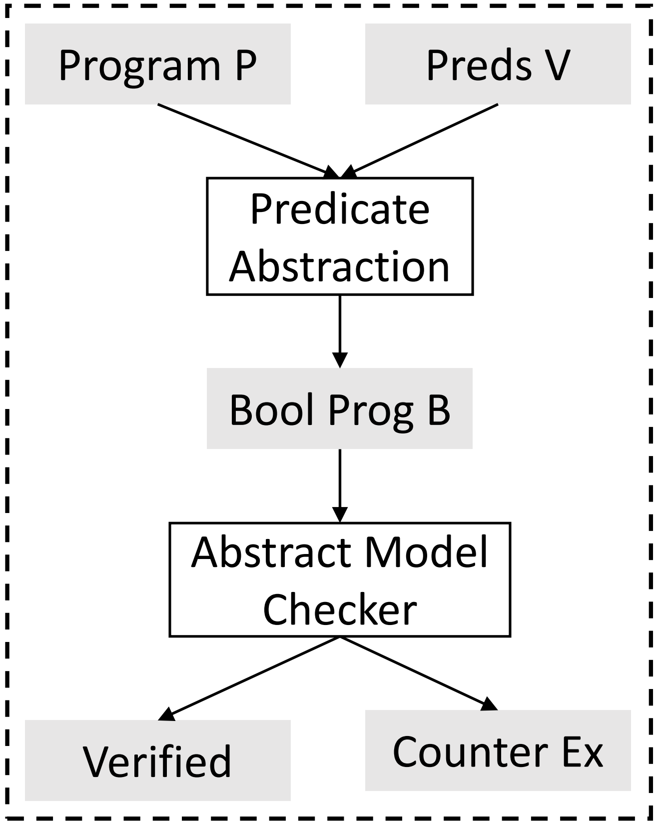
Problem

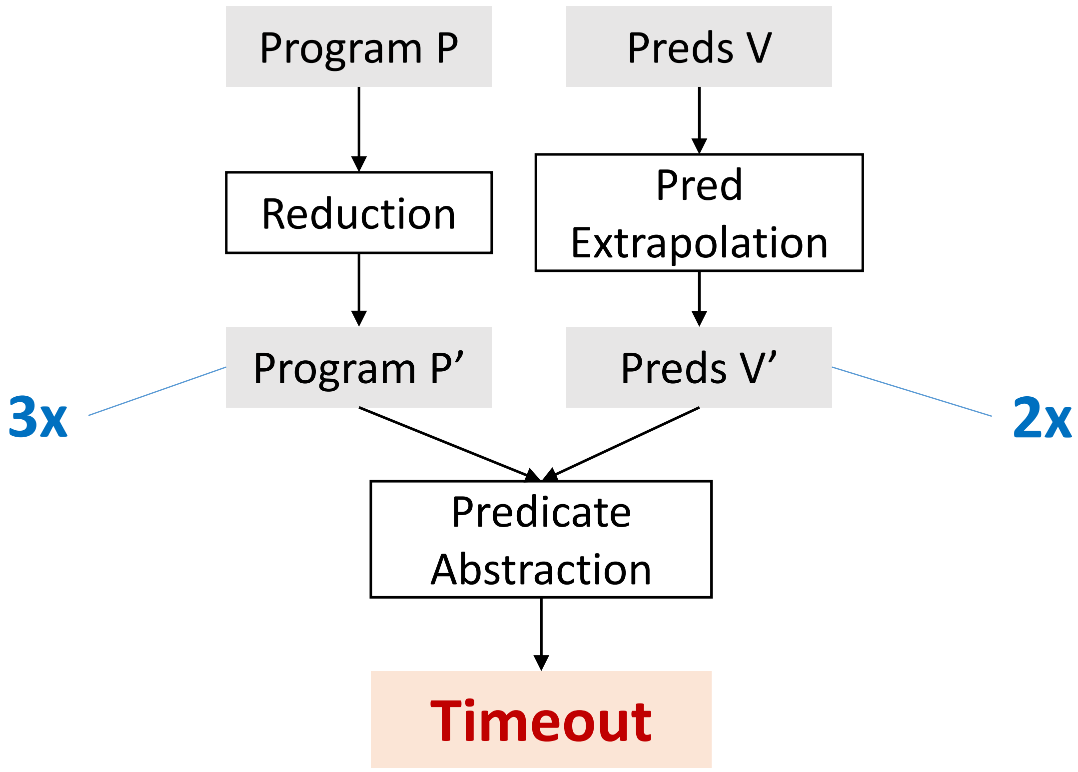




Prove that P $\vDash_{SC}$ S

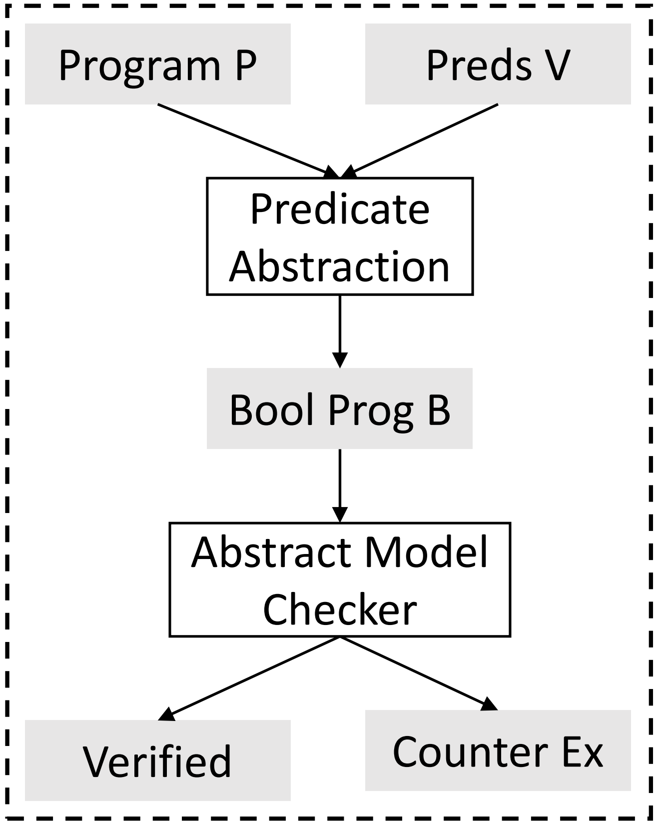
Problem



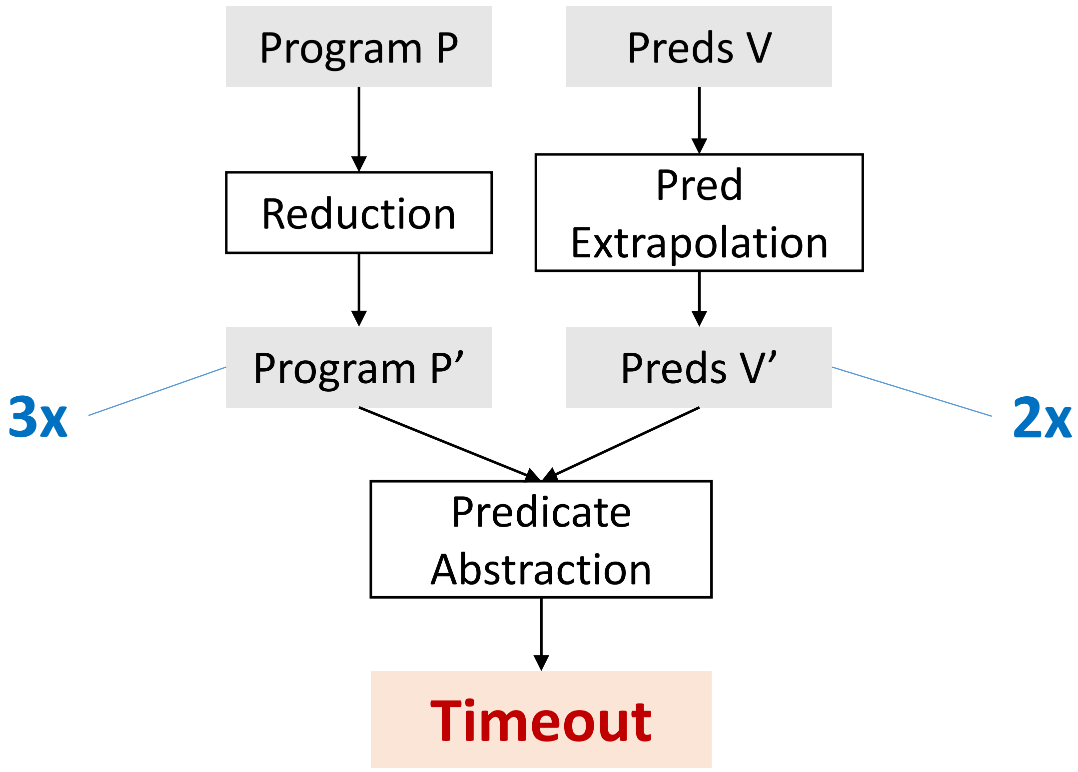


Prove that P $\vDash_{SC}$ S

Problem



Prove that $P \vDash_{SC} S$



Most time used by the SMT solver, computing the abstract transformers

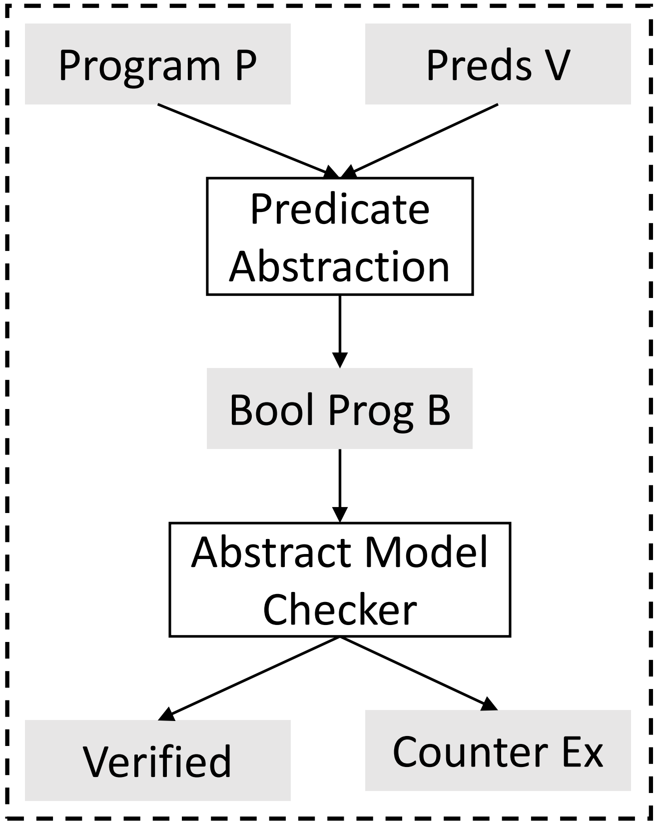
Core problem: computing abstract transformers

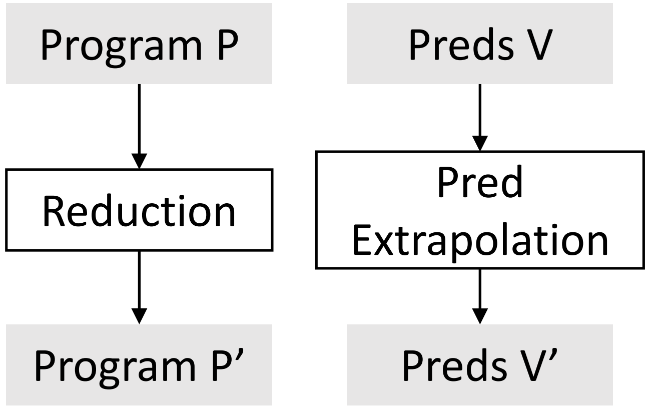
```
Literals q = p or q = ¬p, p ∈ V'
Cubes(V') = {q₁ ∧ … ∧ qⱼ}
```

$|Cubes(V')| = 3^{|V'|}$

```
∀ st ∈ Statements
    ∀ p ∈ V'
        ∀ c ∈ Cubes(V')
            if c ⇒ wp(p, st)        //SMT call
                add c to the transformer
```

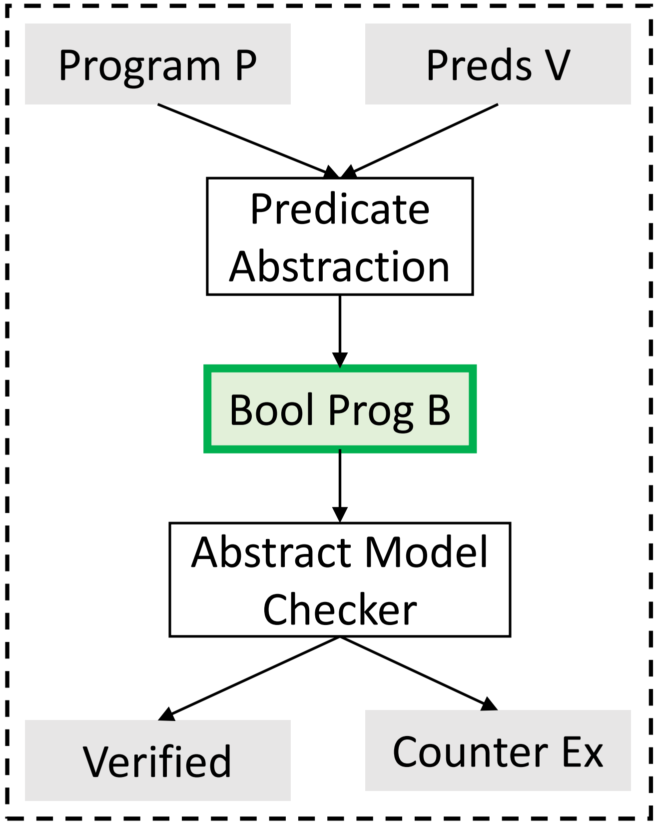
Key Idea



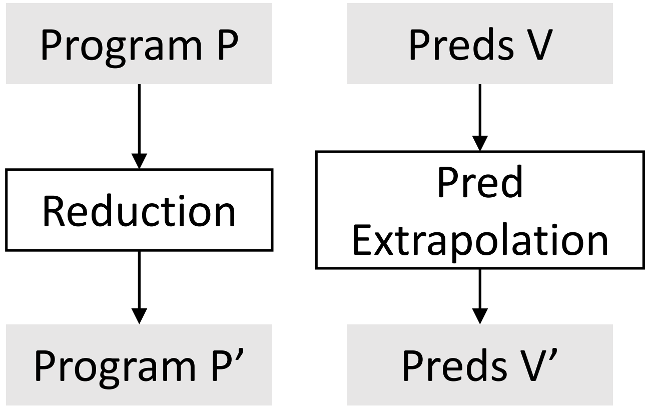


Prove that P $\models_{SC}$ S

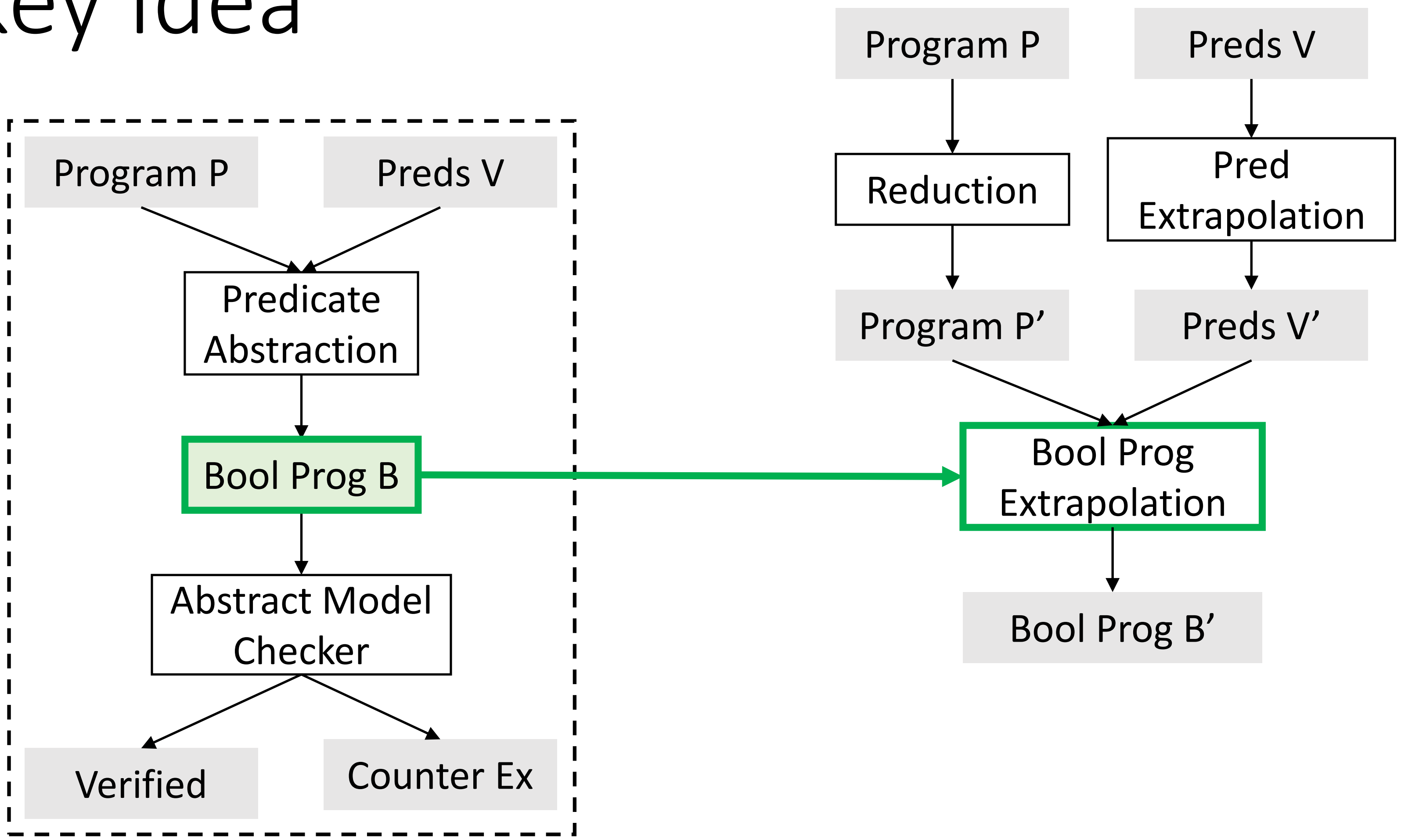
Key Idea



Prove that P $\vDash_{SC}$ S



Key Idea



Prove that P $\vDash_{SC}$ S

Boolean Program Extrapolation

Extrapolate the abstract transformers from the boolean program of the SC proof.

Zero calls to the SMT solver for building the boolean program

Theorem. The boolean program B’ **soundly approximates** of the boolean program PredicateAbstraction(P’, V’).

Abstract transformer for `randomElem`

Program P

Preds V

`(X>0), (Y>Z), (Z>0)`

Bool Prog B

Program P’

```
addElem(Yset, Y)
```

Preds V’

```
(X>0), (Y>Z), (Z>0)
(YSet>Z), ...
```

Bool Prog B’

?

Abstract transformer for **randomElem**

Program P

Preds V

```
(X>0), (Y>Z), (Z>0)
```

Bool Prog B

Program P’

```
addElem(Yset, Y)
```

Preds V’

```
(X>0), (Y>Z), (Z>0)
(YSet>Z), ...
```

Bool Prog B’

```
(YSet>Z) = true, (YSet>Z) ∧ (Y>Z)
           false, ¬(YSet>Z) ∧ ¬(Y>Z)
           *, otherwise
```

Abstract transformer for `randomElem`

Program P

Preds V

```
(X>0), (Y>Z), (Z>0)
```

Bool Prog B

Program P'

```
X = randomElem(YSet)
```

Preds V'

```
(X>0), (Y>Z), (Z>0)
(YSet>Z), ...
```

Bool Prog B'

?

Abstract transformer for `randomElem`

Program P

```
X = Y
```

Preds V

```
(X>0), (Y>Z), (Z>0)
```

Bool Prog B

```
(X>0) = true, (Y>Z) ∧ (Z>0)
        false, ¬(Y>Z) ∧ ¬(Z>0)
        *, otherwise
```

Program P'

```
X = randomElem(YSet)
```

Preds V'

```
(X>0), (Y>Z), (Z>0)
(YSet>Z), ...
```

Bool Prog B'

?

Abstract transformer for `randomElem`

Program P

```
X = Y
```

Program P’

```
X = randomElem(YSet)
```

Preds V

```
(X>0), (Y>Z), (Z>0)
```

Preds V’

```
(X>0), (Y>Z), (Z>0)
(YSet>Z), ...
```

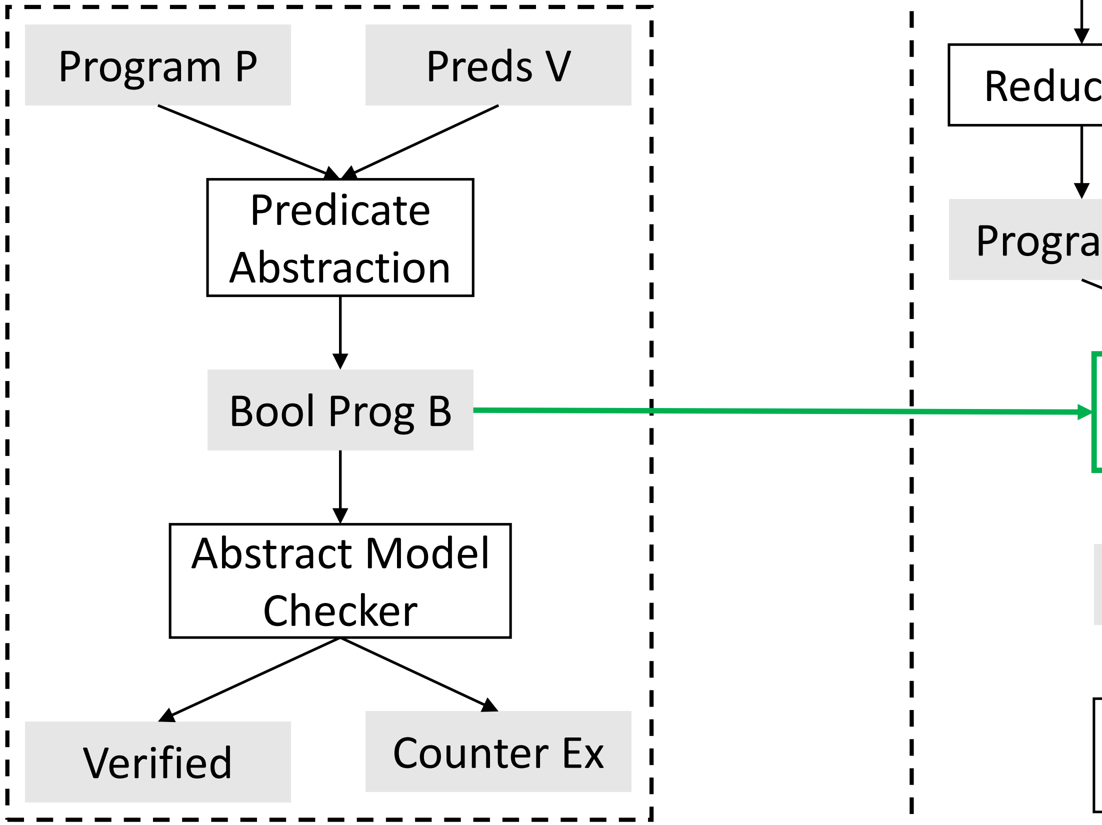
Bool Prog B

```
(X>0) = true, (Y>Z) ∧ (Z>0)
        false, ¬(Y>Z) ∧ ¬(Z>0)
        *, otherwise
```

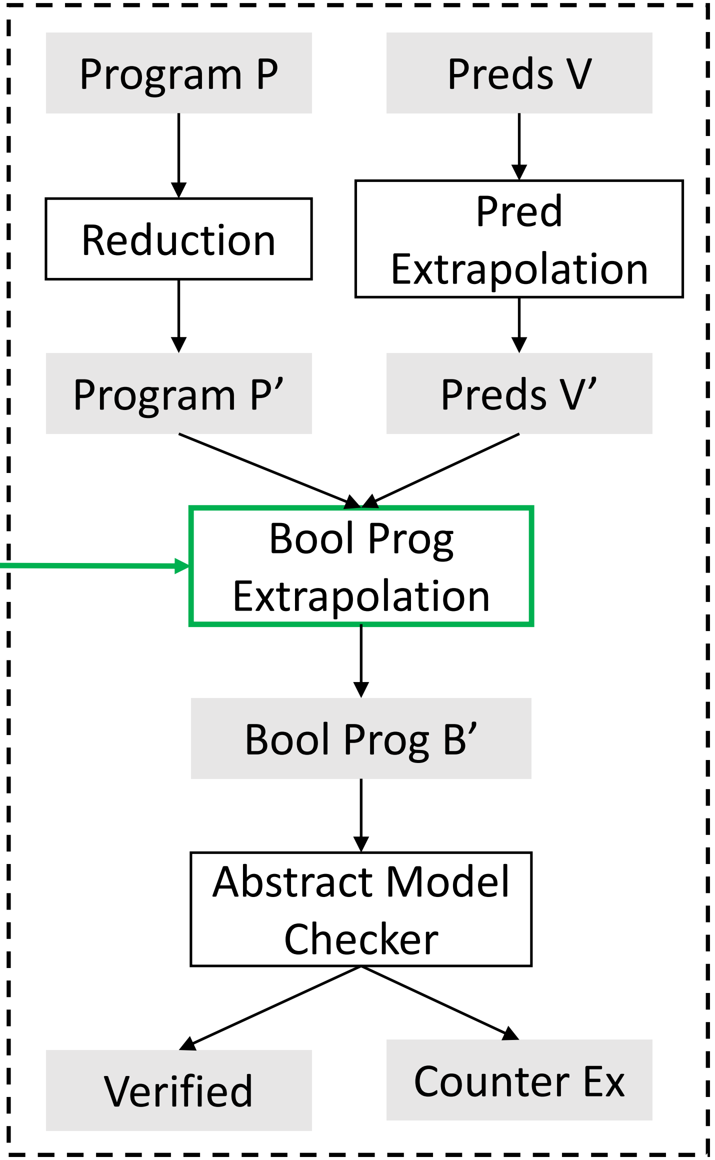
→

Bool Prog B’

```
(X>0) = true, (YSet>Z) ∧ (Z>0)
        false, ¬(YSet>Z) ∧ ¬(Z>0)
        *, otherwise
```



Prove that P $\models_{SC}$ S



Prove that P $\models_{RMA}$ S

Implementation

Predicate Abstraction: cone of influence, Z3 SMT solver

3-valued model checker: Fender

Benchmarks: 14 concurrent algorithms, 2-3 processes, 25-85 lines of code, several have infinite number of states

Specifications: mutual exclusion or reachability invariants

Flush search: start with `flush` after each remote statement, and try removing

	SC Predicate Abstraction			RMA Predicate Abstraction			
Algorithm	|V|	B (s)	B (loc)	|V'|	B' (loc)	Fender (s)	Min `flush`
Dekker	11	1	498	29	2068	294	4/12
Peterson	10	1	356	21	1045	3	4/7
ABP	16	1	485	20	662	1	2/2
Pc1	18	2	658	35	3797	65	2/7
Pgsql	12	1	418	18	1549	1	2/4
Qw	13	2	487	29	1544	1345	4/5
Sober	23	7	831	48	8466	4	0/9
Kessel	18	3	534	36	1621	45	4/10
Loop2_TLM	29	66	1068	43	1986	2204	2/4
Szymanski	34	228	1182	64	7081	316	7/14
Queue	13	35	572	22	1104	14	1/2
Ticket	17	117	640	43	3615	3493	5/6
Bakery	19	337	828	41	2947	203	6/10
RMA Lock	24	50	763	60	5932	65679	9/18

	SC Predicate Abstraction			RMA Predicate Abstraction			
Algorithm	|V|	B (s)	B (loc)	|V'|	B' (loc)	Fender (s)	Min `flush`
Dekker	11	1	498	29	2068	294	4/12
Peterson	10	1	356	21	1045	3	4/7
ABP	16	1	485	20	662	1	2/2
Pc1	18	2	658	35	3797	65	2/7
Pgsql	12	1	418	18	1549	1	2/4
Qw	13	2	487	29	1544	1345	4/5
Sober	23	7	831	48	8466	4	0/9
Kessel	18	3	534	36	1621	45	4/10
Loop2_TLM	29	66	1068	43	1986	2204	2/4
Szymanski	34	228	1182	64	7081	316	7/14
Queue	13	35	572	22	1104	14	1/2
Ticket	17	117	640	43	3615	3493	5/6
Bakery	19	337	828	41	2947	203	6/10
RMA Lock	24	50	763	60	5932	65679	9/18

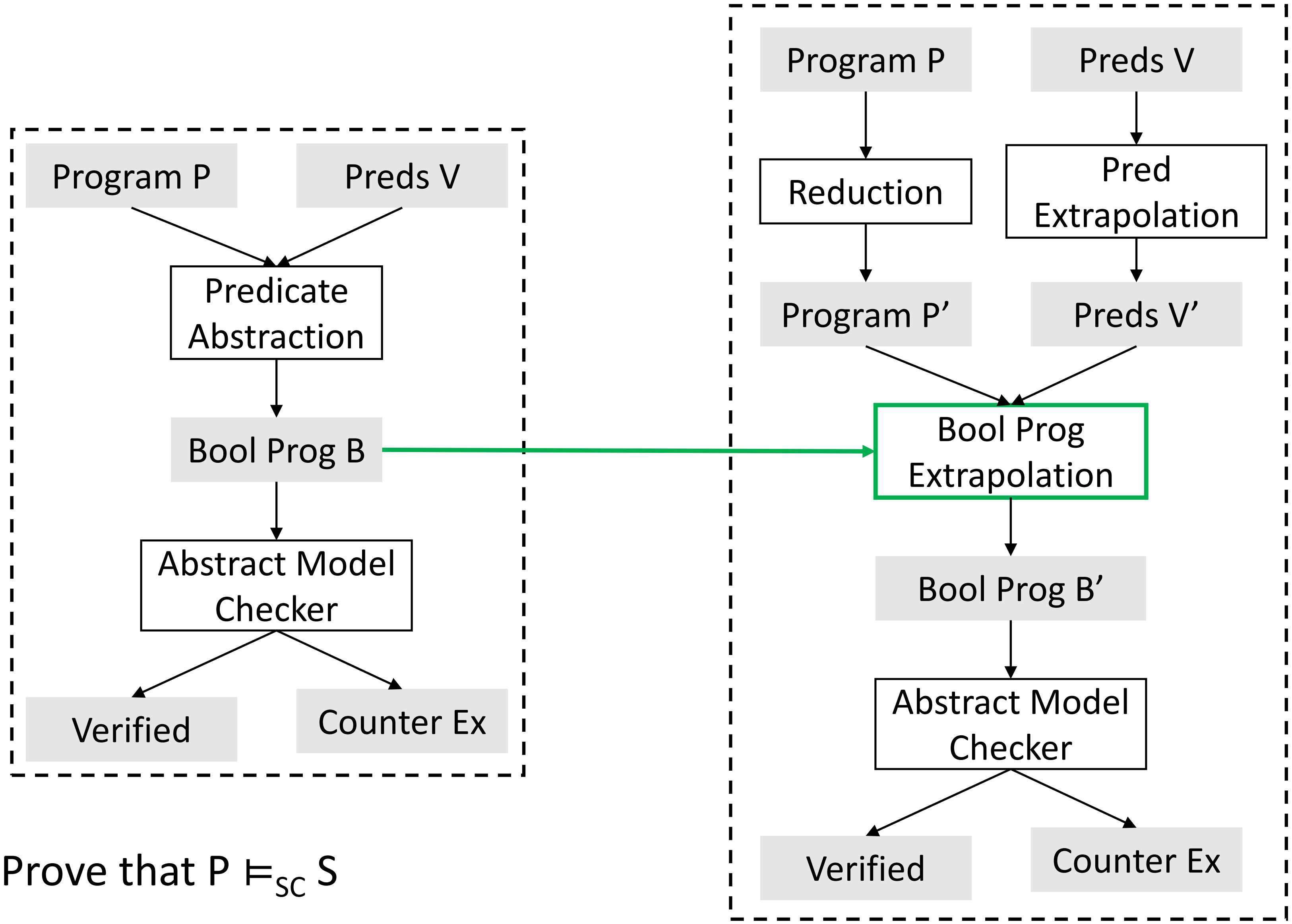
	SC Predicate Abstraction			RMA Predicate Abstraction			
Algorithm	|V|	B (s)	B (loc)	|V'|	B' (loc)	Fender (s)	Min flush
Dekker	11	1	498	29	2068	294	4/12
Peterson	10	1	356	21	1045	3	4/7
ABP	16	1	485	20	662	1	2/2
Pc1	18	2	658	35	3797	65	2/7
Pgsql	12	1	418	18	1549	1	2/4
Qw	13	2	487	29	1544	1345	4/5
Sober	23	7	831	48	8466	4	0/9
Kessel	18	3	534	36	1621	45	4/10
Loop2_TLM	29	66	1068	43	1986	2204	2/4
Szymanski	34	228	1182	64	7081	316	7/14
Queue	13	35	572	22	1104	14	1/2
Ticket	17	117	640	43	3615	3493	5/6
Bakery	19	337	828	41	2947	203	6/10
RMA Lock	24	50	763	60	5932	65679	9/18

	SC Predicate Abstraction			RMA Predicate Abstraction			
Algorithm	|V|	B (s)	B (loc)	|V'|	B' (loc)	Fender (s)	Min `flush`
Dekker	11	1	498	29	2068	294	4/12
Peterson	10	1	356	21	1045	3	4/7
ABP	16	1	485	20	662	1	2/2
Pc1	18	2	658	35	3797	65	2/7
Pgsql	12	1	418	18	1549	1	2/4
Qw	13	2	487	29	1544	1345	4/5
Sober	23	7	831	48	8466	4	0/9
Kessel	18	3	534	36	1621	45	4/10
Loop2_TLM	29	66	1068	43	1986	2204	2/4
Szymanski	34	228	1182	64	7081	316	7/14
Queue	13	35	572	22	1104	14	1/2
Ticket	17	117	640	43	3615	3493	5/6
Bakery	19	337	828	41	2947	203	6/10
RMA Lock	24	50	763	60	5932	65679	9/18

	SC Predicate Abstraction			RMA Predicate Abstraction			
Algorithm	|V|	B (s)	B (loc)	|V'|	B' (loc)	Fender (s)	Min `flush`
Dekker	11	1	498	29	2068	294	4/12
Peterson	10	1	356	21	1045	3	4/7
ABP	16	1	485	20	662	1	2/2
Pc1	18	2	658	35	3797	65	2/7
Pgsql	12	1	418	18	1549	1	2/4
Qw	13	2	487	29	1544	1345	4/5
Sober	23	7	831	48	8466	4	0/9
Kessel	18	3	534	36	1621	45	4/10
Loop2_TLM	29	66	1068	43	1986	2204	2/4
Szymanski	34	228	1182	64	7081	316	7/14
Queue	13	35	572	22	1104	14	1/2
Ticket	17	117	640	43	3615	3493	5/6
Bakery	19	337	828	41	2947	203	6/10
RMA Lock	24	50	763	60	5932	65679	9/18

Conclusion

$P \vDash_{SC} S$		$P \vDash_{RMA} S$
Program P →	Reduction	→ Program P’
Preds V →	Pred Extrapolation	→ Preds V’
Bool Prog B →	Bool Prog Extrapolation	→ Bool Prog B’



Prove that $P \vDash_{SC} S$

Prove that $P \vDash_{RMA} S$