NEC Laboratories America
Relentless passion for Innovation

Bang for the Buck: Improvising and Scheduling Verification Engines for Effective Resource Utilization

HWVW 2010*
July 15, 2010

Malay Ganai and Weihong Li

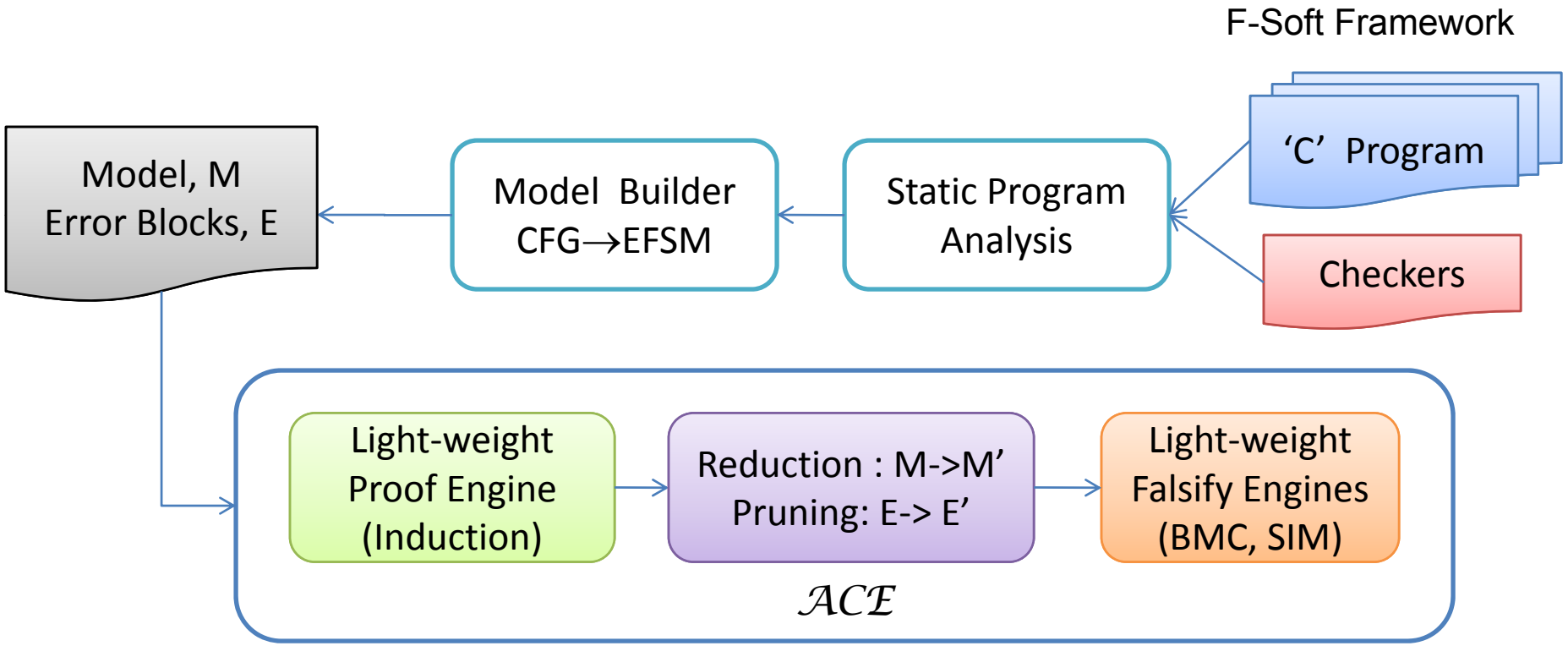
System Analysis & Verification Group
NEC Labs America
Princeton, U. S. A.

*Originally presented in Memocode'09

Embedded System Verification

- Many properties to be solved in a tight time budget
  - 10-12 min per function modules (NEC Verification services)
- Need good verification procedures
  - Without proofs, falsification engine consumes resources fruitlessly
  - Without falsification, proof engine consumes resources fruitlessly
- Goal: How to combine these engines to get ``bang for the buck”?

ACE Overview: Bang for the Buck



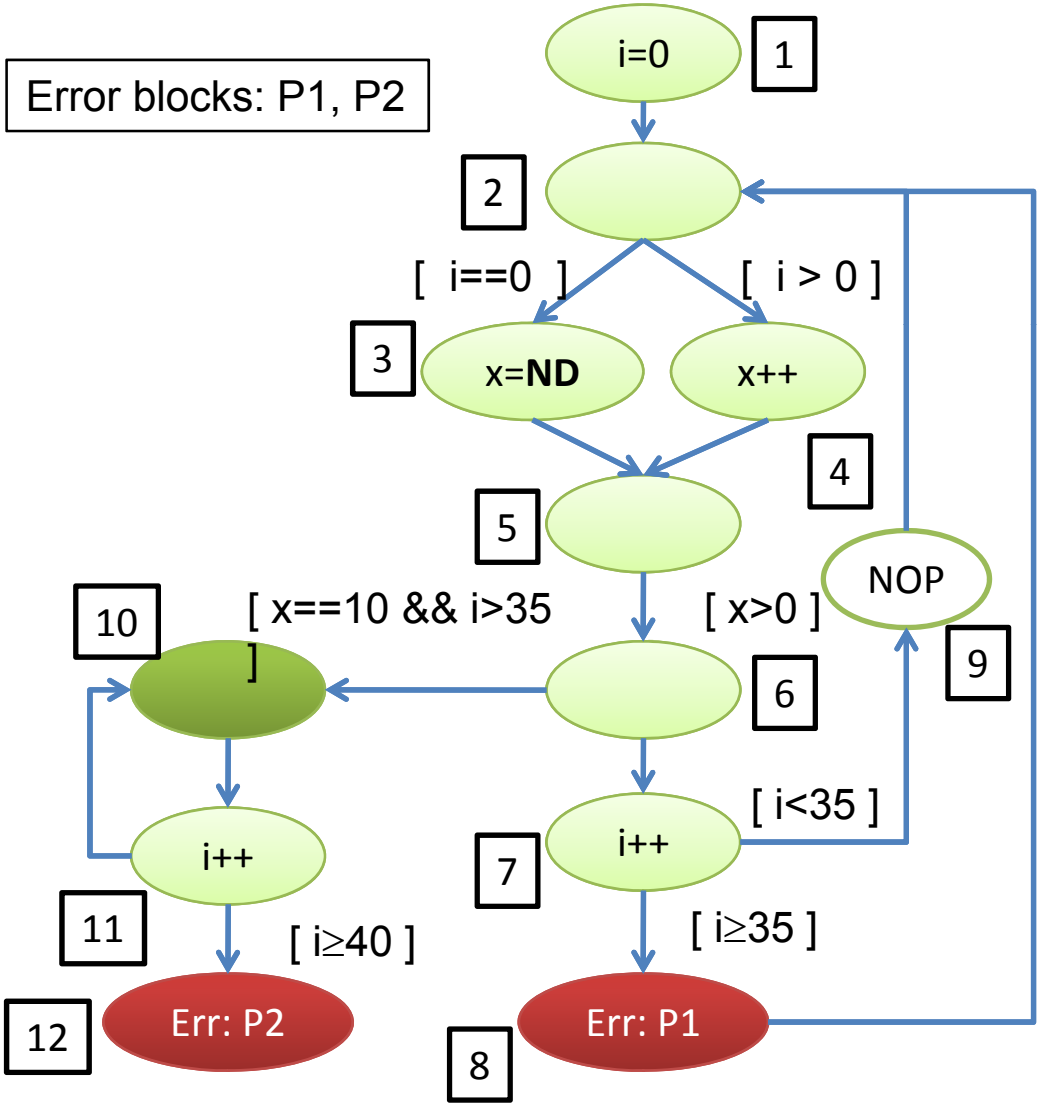
Augmentation and Combination of Verification Engines

Example

```
1. void foo(void) {
2.   i=0;
3.   while(1){
4.     if (i==0){
5.       x = ND();
6.       assume (x > 0);
7.     } else if (i>0)
8.       x++;
9.     if (x==10 && i>35)
10.      break;
11.    i++;
12.    assert(i<35); /* P1 */
13.  }
14.  while(1){
15.    i++;
16.    assert(i<40); /* P2 */
17.  }
18.}
```

C program to CDFG

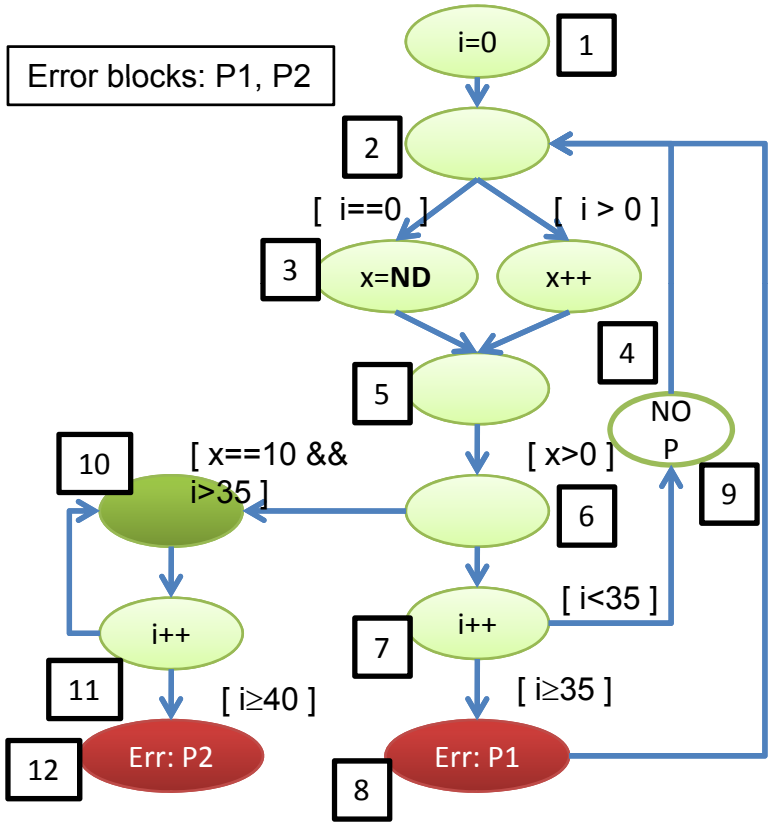
```
1.  void foo(void) {
2.    i=0;
3.    while(1){
4.      if (i==0){
5.        x = ND();
6.        assume (x > 0);
7.      } else if (i>0)
8.        x++;
9.      if (x==10 && i>35)
10.       break;
11.     i++;
12.     assert(i<35); /* P1 */
13.   }
14.   while(1){
15.     i++;
16.     assert(i<40); /* P2 */
17.   }
18. }
```

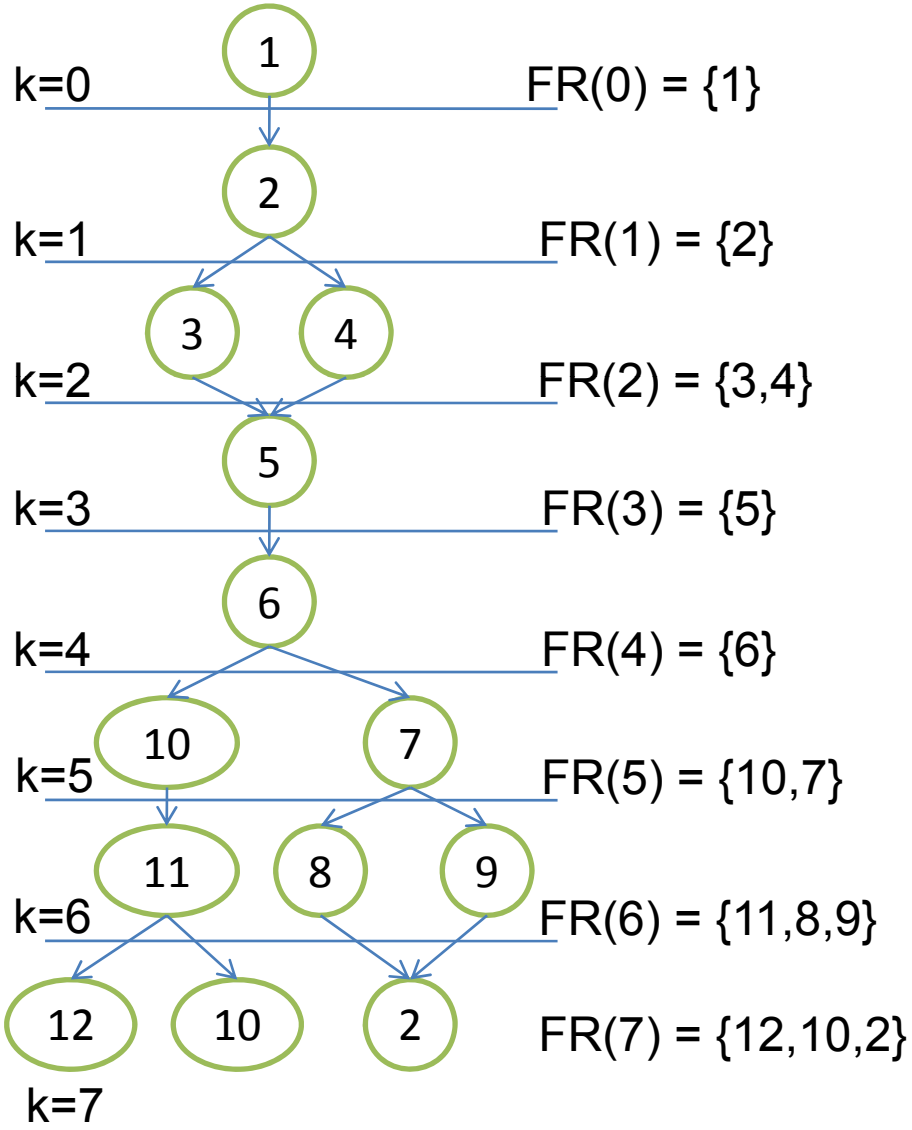


Transition Relation

- Control state variable, *PC*
  - Boolean Predicate, $B_r \equiv (PC = r)$
  - *domain*(*PC*) = {S1,…,S12}
- Next state functions
  - *next* (*PC*) = B_{S1} ? S2 : B_{S2} && *T23* ? S2 : B_{S2} && *T24*: S4 : … : *PC*;
  - *next* (*x*) = (B_{S3})? ND() : (B_{s4}) ? x++ : *x*;

Forward Control State Reachability



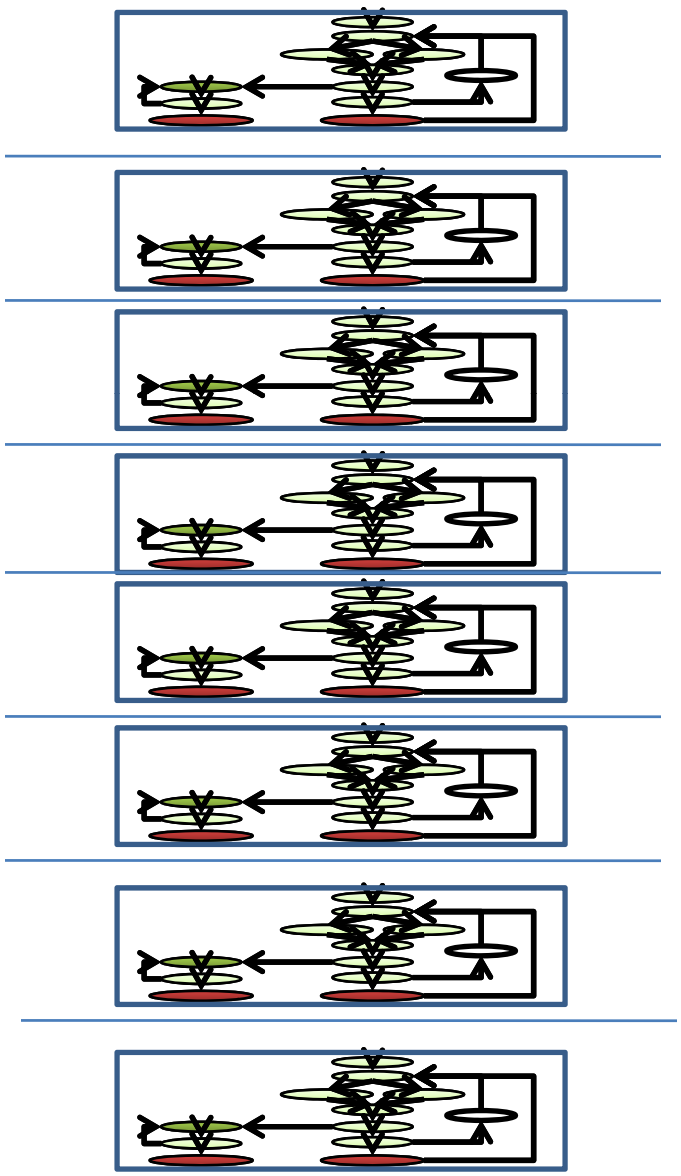


Transition Relation

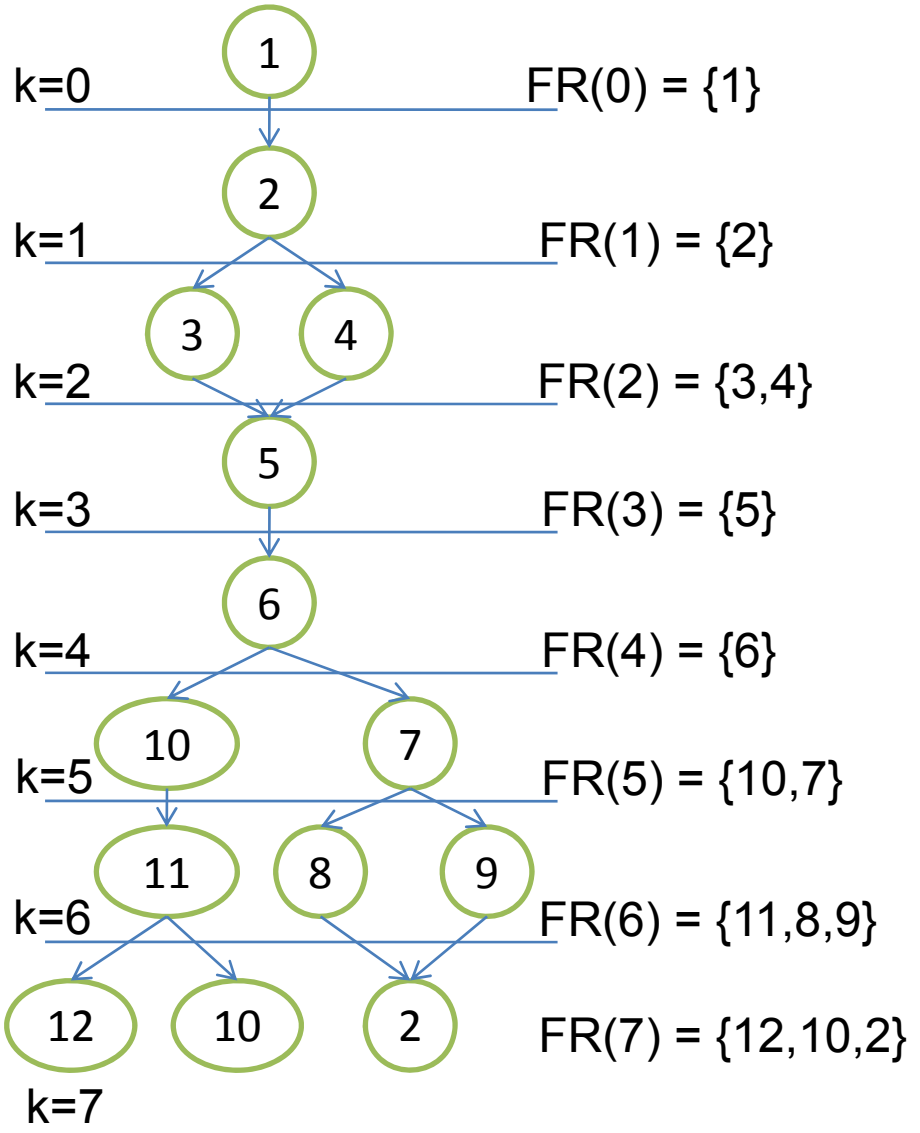
- Control state variable, *PC*
  - Boolean Predicate, $B_r \equiv (PC = r)$
  - *domain*(*PC*) = {S1,…,S12}
- Next state functions
  - *next* (*PC*) = B_{S1} ? S2 : B_{S2} && *T23* ? S2 : B_{S2} && *T24*: S4 : … : *PC*;
  - *next* (*x*) = (B_{S3})? ND() : (B_{S4}) ? x++ : *x*;

next(x) = x if S3,S4 $\notin$ FR(k)

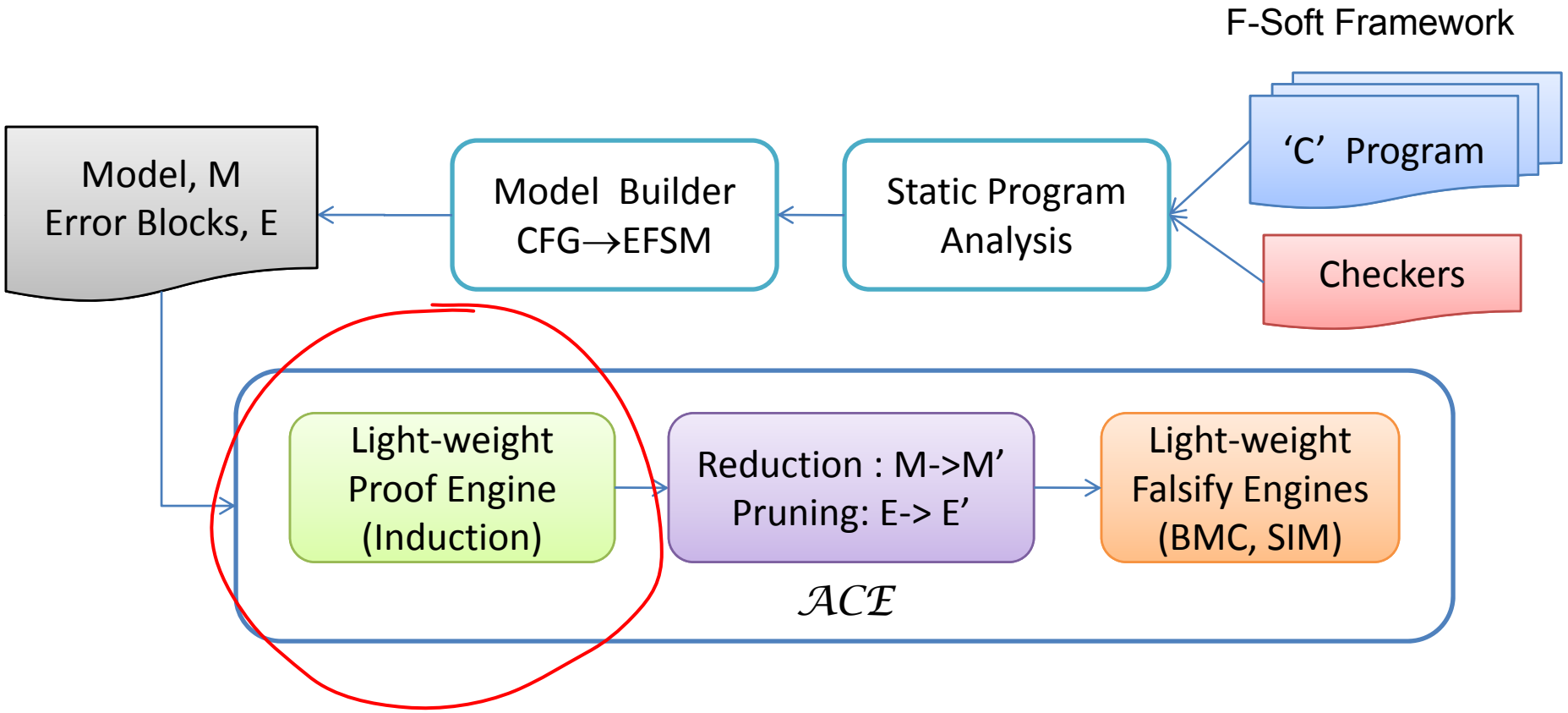
BMC simplification using CSR



Ganai et al ICCAD 2006



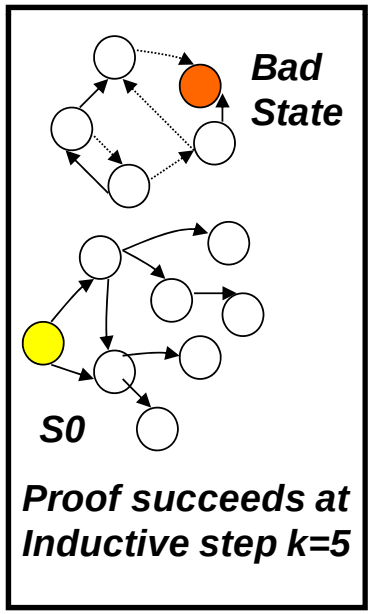
ACE Overview: Bang for Buck



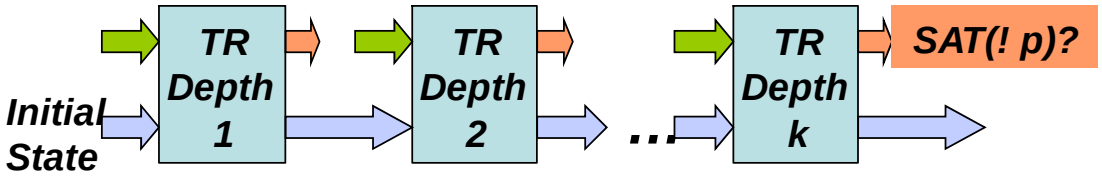
Augmentation and Combination of Verification Engines

SAT-based Induction

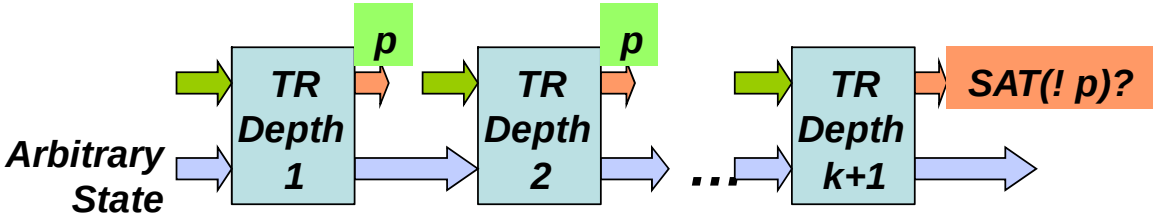
- **Proof by Induction with increasing depth (G p)**



Base Step: If *Sat(!p_k)*, then property is false



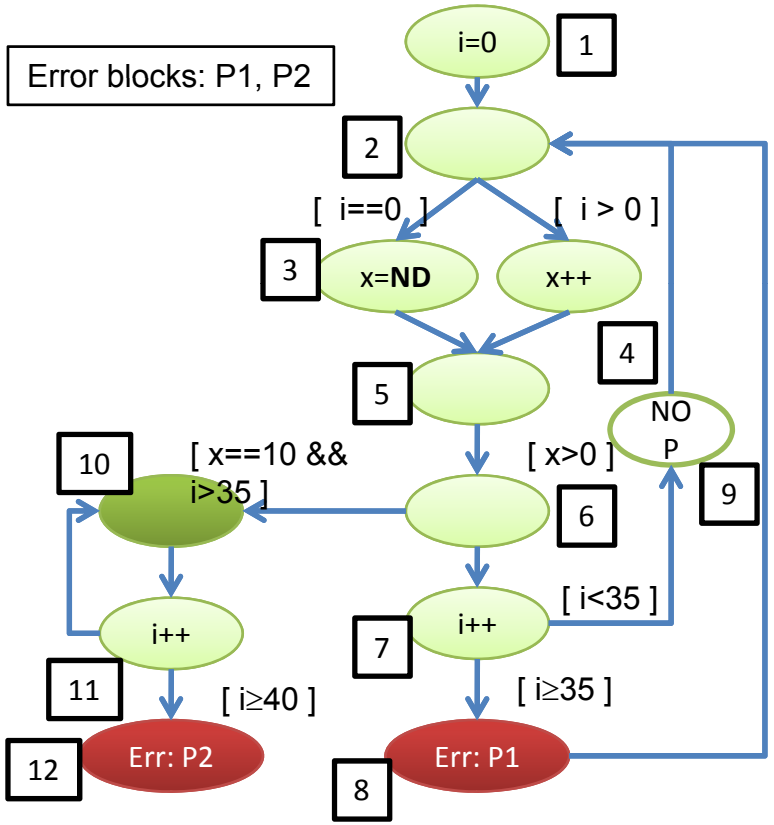
Inductive Step: If *Unsat(!p_k+1)*, then property is true

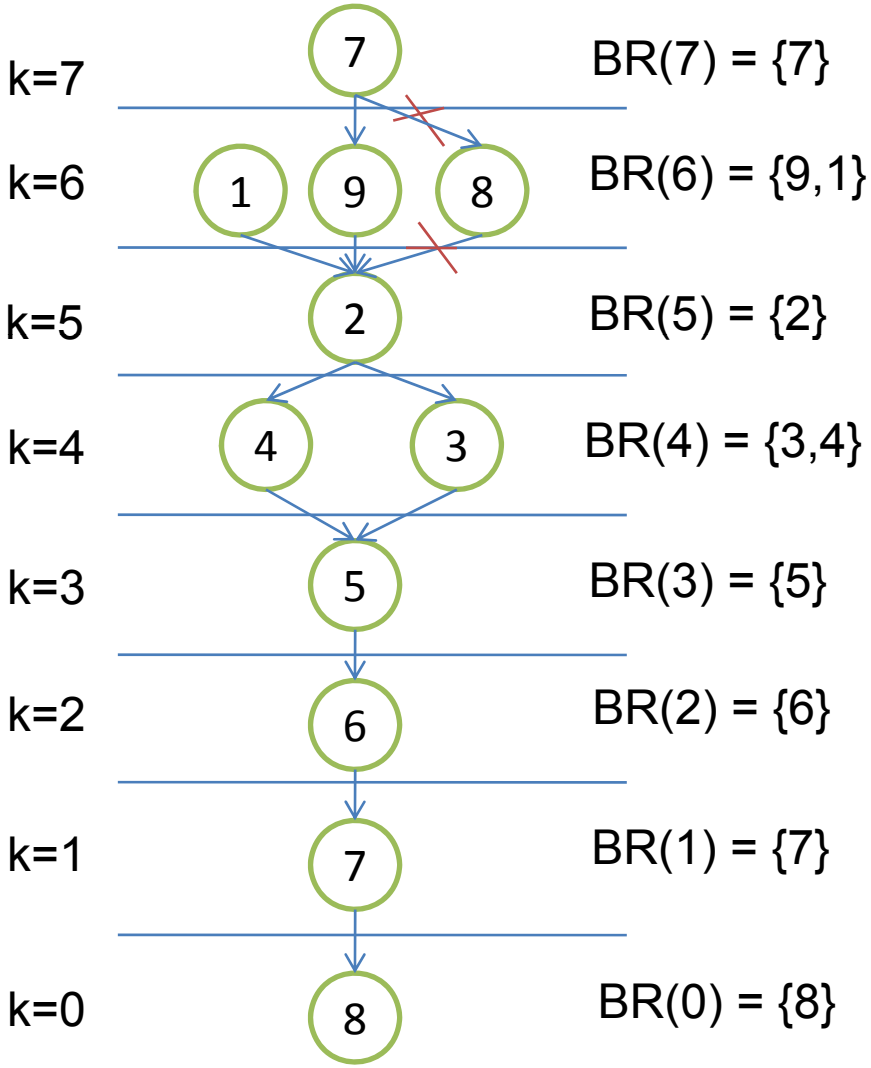


Else k++ (until resources are exhausted)

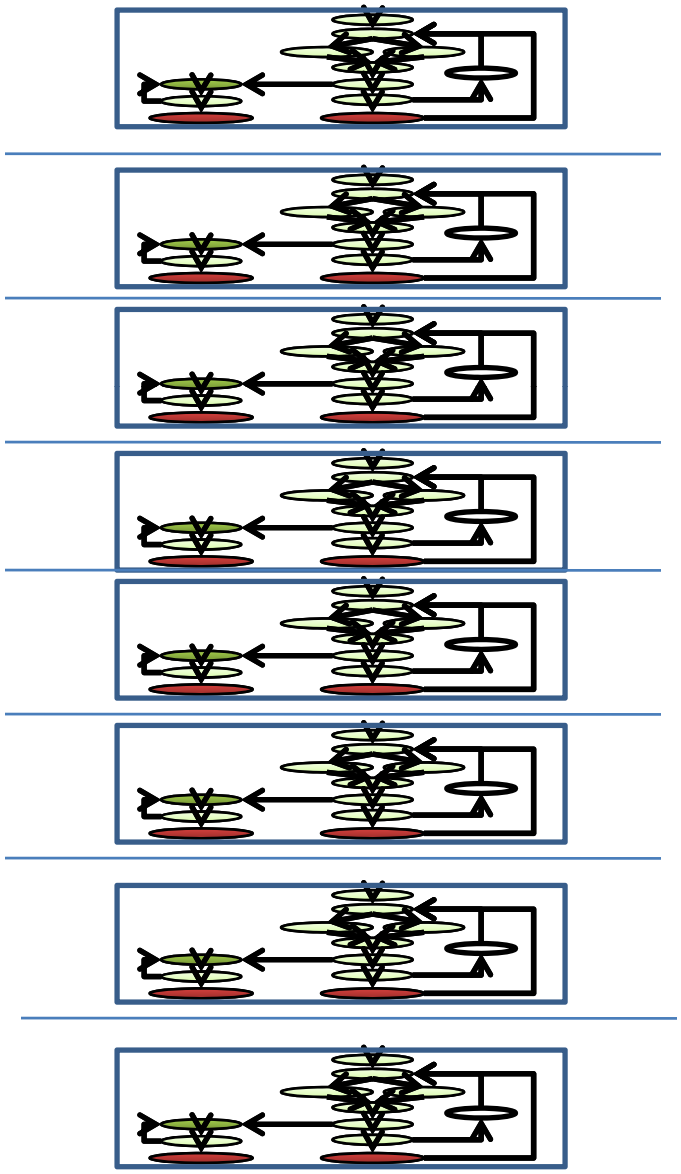
Sheeran et al

Backward Control State Reachability

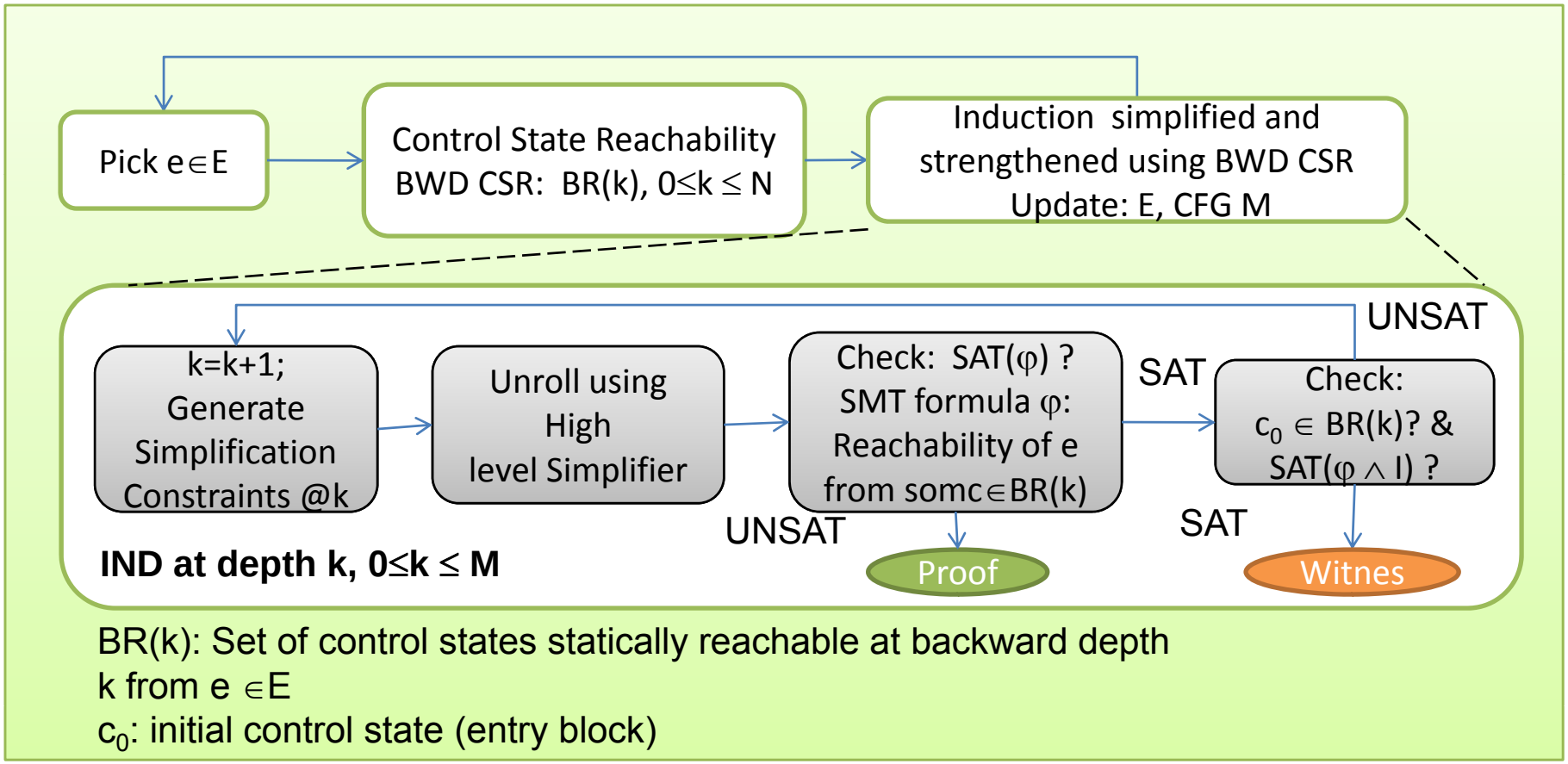




Induction Simplify using CSR

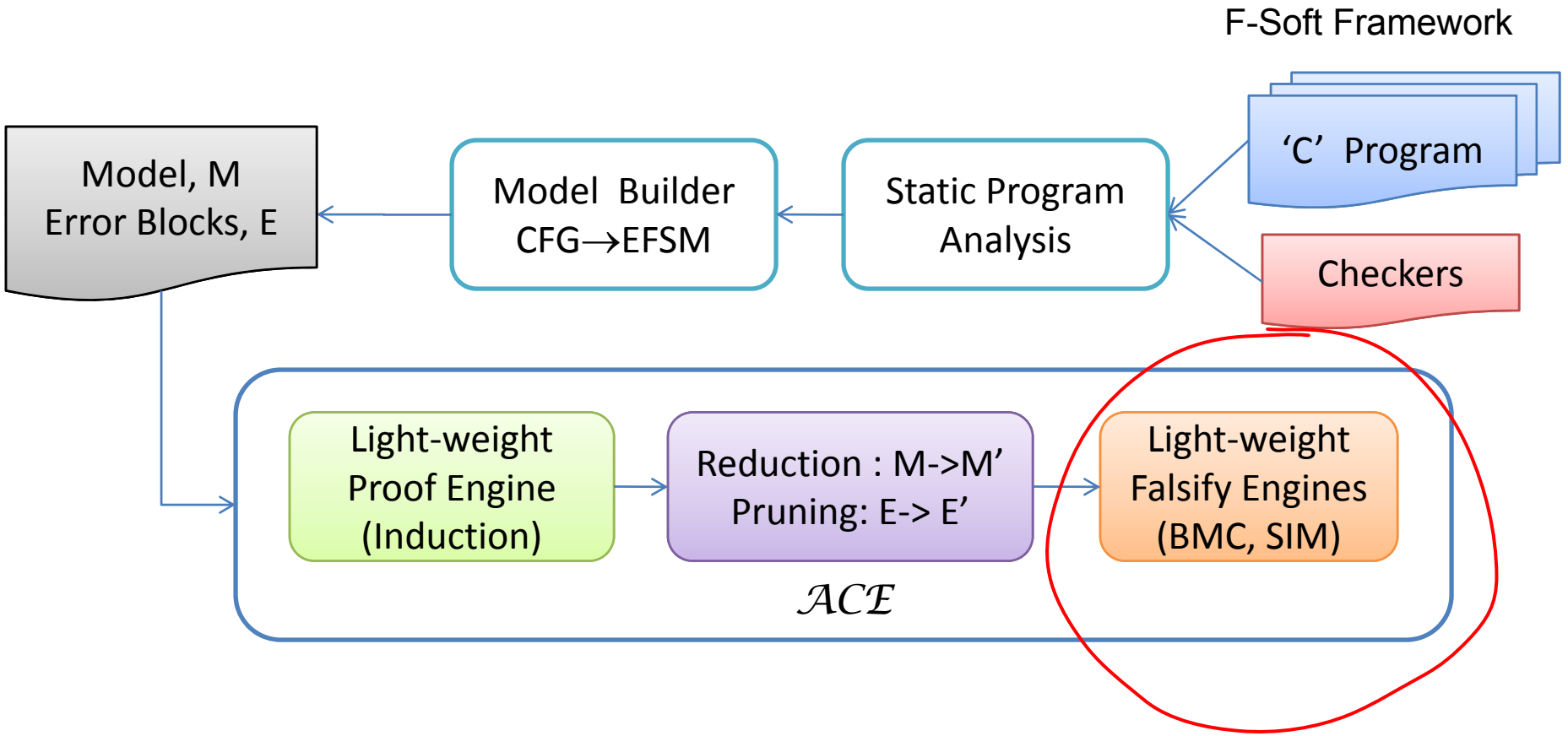


Light-weight Proof Engine



Goal: Induction Strengthening using BWD CSR

ACE Overview: Bang for Buck



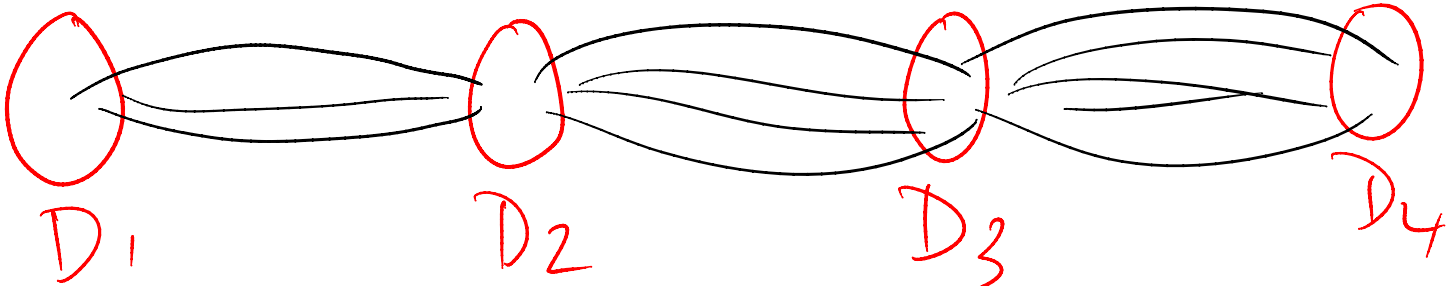
Augmentation and Combination of Verification Engines

Lighthouses

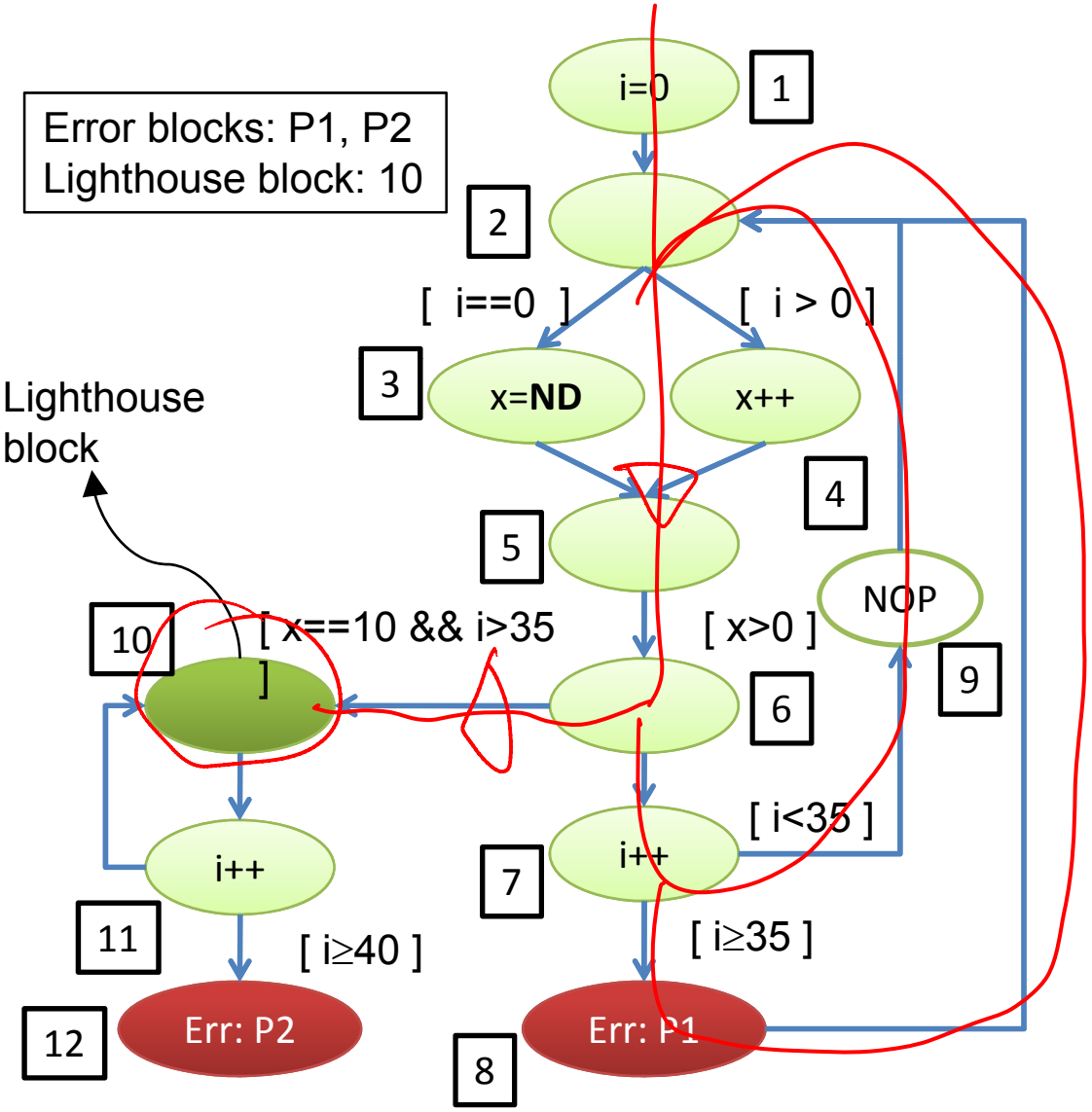
"Intermediate state predicates that guide the search"

Eg: Dominators (D_i)

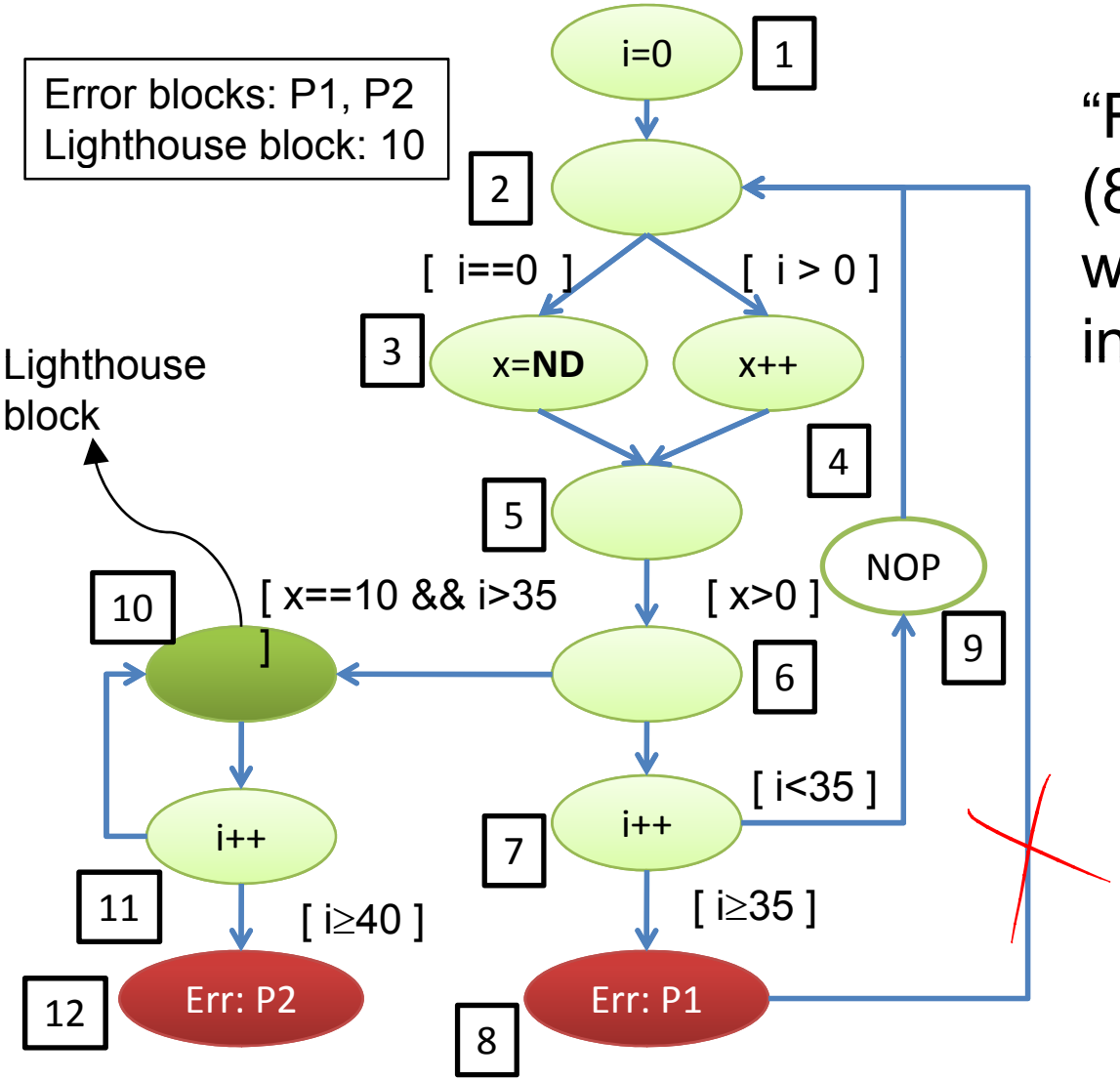
Observation: A fewer paths (due to less branching conditions) but longer (due to loops) from D_i to D_{i+1} make simulation engine to reach D_{i+1} more likely.



Lighthouses: Control Dominators



Lighthouses: Error Blocks



"Removing the out edge (8→2) of error block P1 we obtain conditional, inductive proof for P2"

"P1 dominates P2"

Proofs on M and M’

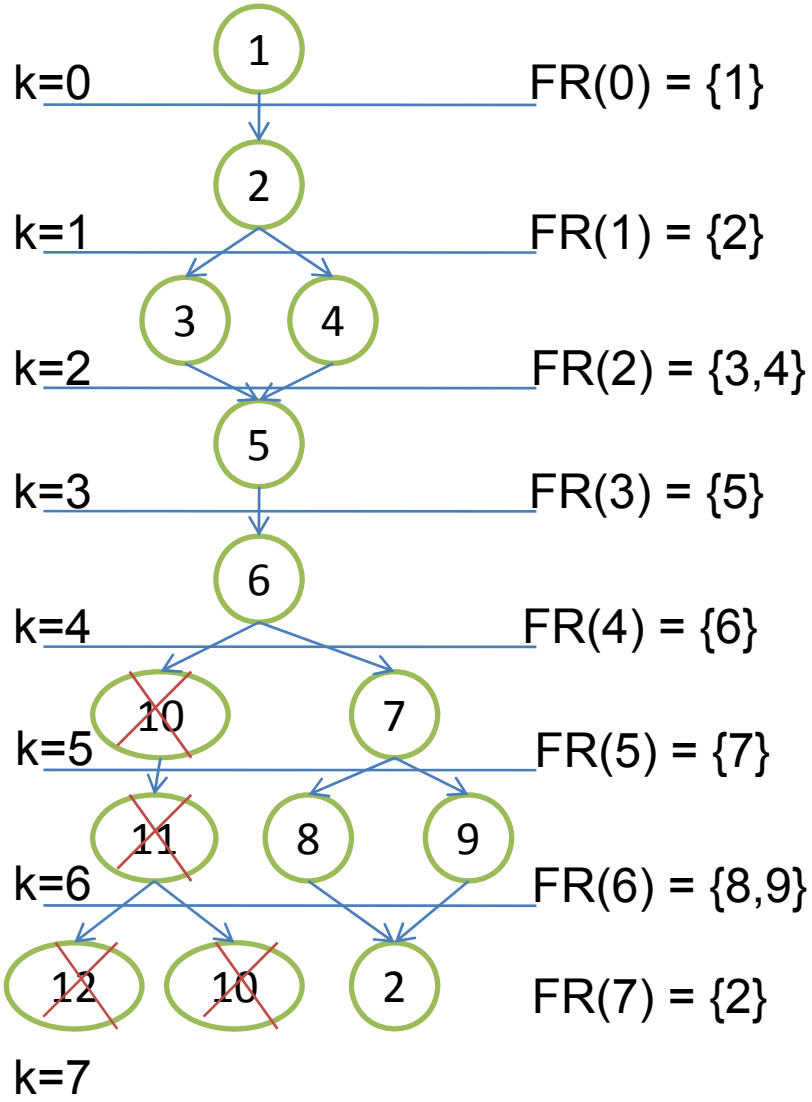
Ex (# functions)	#P	M	M’
S1 (14)	166	14	48
S2 (19)	127	4	80
S3 (36)	96	16	40
S4 (8)	82	43	59
S5 (4)	39	10	12
S6 (4)	34	4	23
S7 (4)	4	0	0
Total	**548**	**91**	**262**

M’ obtained from M by removing “error-out” edge

Upshot: Error blocks are good candidates for lighthouses

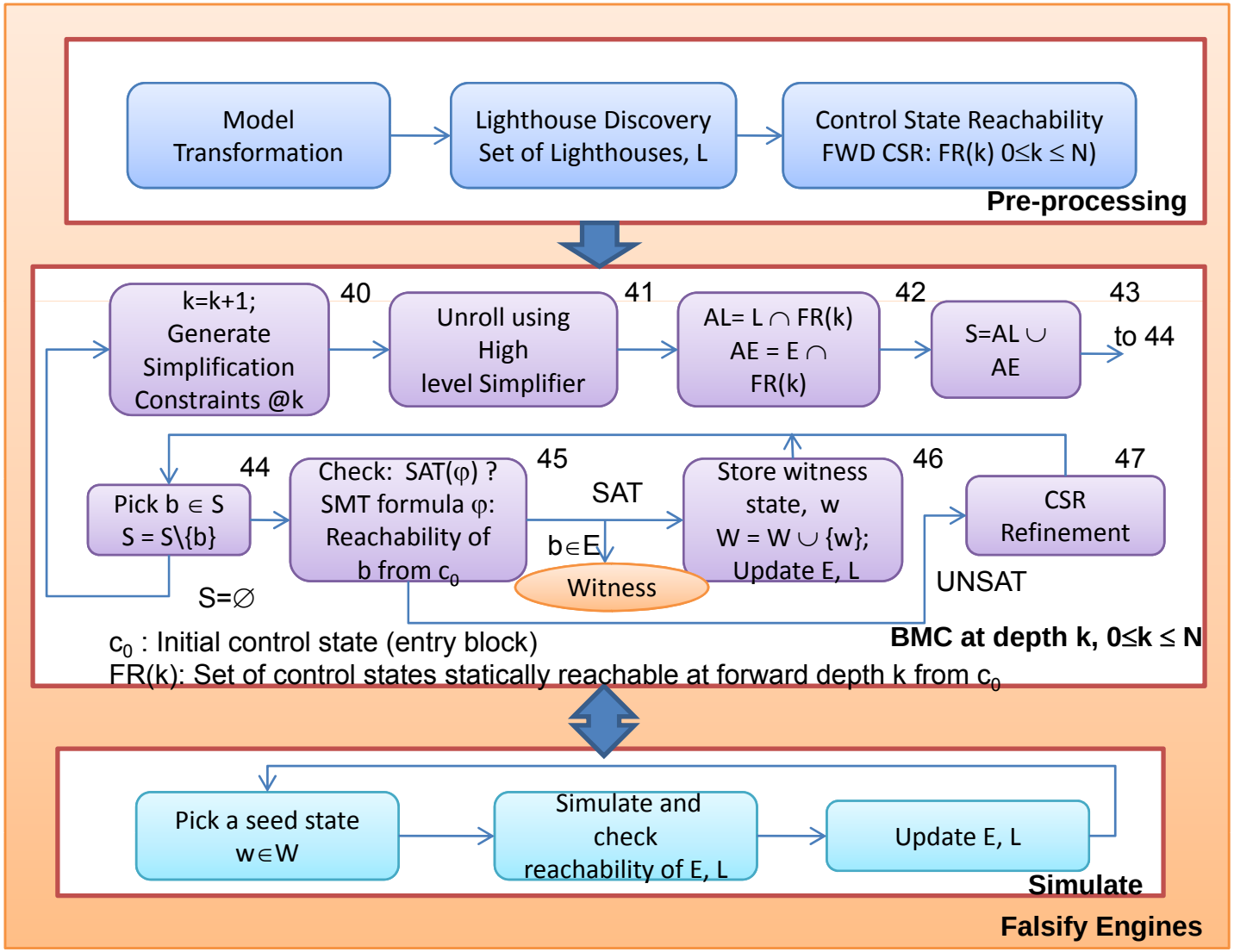
Intel Xeon 2.8Ghz, 4Gb RAM 1 min per functions
Proof Engine used LPE(smt)

CSR Refinement (dynamic)

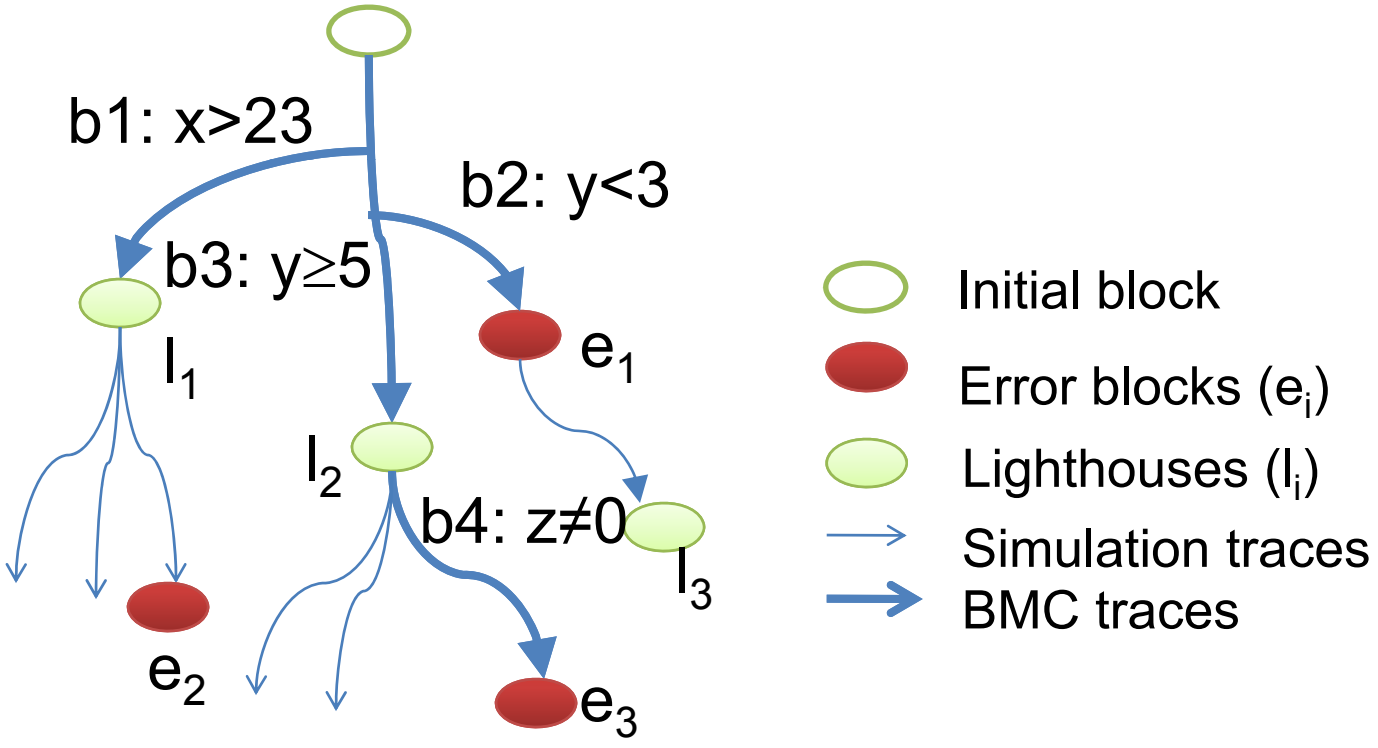


Pruning by CSR Refinement

ACE: LPE, BMC, Guided Simulation



BMC + Guided Sim



Experiments (1/2)

- 7 Benchmarks C programs (1K-3k LoC)
  - information management system utilities, ftp utilities, network applications, embedded apps.
  - array bound violations, pointer validity, …
  - # functions: 4 to 36
    - each associated with multiple properties

Experiments (2/2)

- Model Checkers (for comparison)
  - SAT-based Induction (Sheeran et al. FMCAD 2000)
  - SAT-based UMC (Ganai et al. ICCAD 2004)
  - MIX [Composite Model Checking] (Yang, MEMOCODE 2006)
  - SMT-based BMC (Ganai et al. ICCAD 2006)
  - MIX-SA [MIX combined with widening] (Wang, CAV 2007)
- Solvers
  - Hybrid SAT Solver (Ganai et al. DAC 2002)
  - Yices SMT solver (Dutertre et al. CAV 2006)
  - Omega [Presburger Solver] (Pugh et al. AC 1991)

Results: SAT-based Proof Engines

Ex (# functions)	#P	UMC	IND	IND (No LFP)	LPE
S1 (14)	166	8	13	13	14
S2 (19)	127	2	2	2	4
S3 (36)	96	4	4	7	15
S4 (8)	82	26	28	31	41
S5 (4)	39	5	4	6	10
S6 (4)	34	4	4	4	4
S7 (4)	4	0	0	0	0
Total	**548**	**49**	**55**	**63**	**88**

Upshot: LPE proves more than any other engine

Intel Xeon 2.8Ghz, 4Gb RAM 1 min per function
All uses Hybrid SAT Solver (Ganai et al. DAC 2002)

Results: Proof Engines

Ex (# functions)	#P	LPE (SMT)	MIX-SA (Presb.)
S1 (14)	166	14	0
S2 (19)	127	4	6
S3 (36)	96	16	12
S4 (8)	82	43	11
S5 (4)	39	10	0
S6 (4)	34	4	4
S7 (4)	4	0	0
Total	**548**	**91**	**33**
Exclusive		**62**	**4**

Upshot: LPE proves more than MIX

Intel Xeon 2.8Ghz, 4Gb RAM 1 min per function
SMT solver: yices,
Presburger Solver: Omega library

Falsify Engines Configurations

Feature Code	Description	FE_ALL	FE_NoLH	FE_NoSim	FE
a	Static context-sensitive fwd CSR Model Transformation CSR learning and simplification	X	X	X	X
b	Lighthouse	X		X	
c	CSR Refinement	X	X	X	
e	Guided-Simulation	X	X		

Results: Comparing ACE and MIX

Ex (#func)	#P	ACE (ALL)		ACE(NoLH)		ACE(NoSim)		ACE()		MIX	
		#P	#W	#P	#W	#P	#W	#P	#W	#P	#W
S1 (14)	166	14	122	14	103	14	92	14	81	8	84
S2 (19)	127	4	33	4	38	4	37	4	38	8	35
S3 (36)	96	16	77	16	71	16	77	16	71	18	77
S4 (8)	82	43	16	43	16	43	16	43	15	23	16
S5 (4)	39	10	17	10	2	10	3	10	2	0	10
S6 (4)	34	4	30	4	30	4	30	4	30	4	30
S7 (4)	4	0	0	0	0	0	0	0	0	4	0
Total	**548**	**91**	**295**	**91**	**260**	**91**	**255**	**91**	**237**	**57**	**252**
Excl.		**45**	**47**							**11**	**4**

Upshot: ACE(all) solves more than MIX

Intel Xeon 2.8Ghz, 4Gb RAM 10 min per function
SMT solver: yices, Presburger Solver: Omega library

Summary/Conclusions

- Presented an improved verification flow ACE combining
  - light weight proof engines
  - bmc engines
  - guided simulation using lighthouses
- Geared towards checking many properties in a tight time budget
- Integrated in industry verification environment
  - provided control experimentation
- In progress: distribute ACE on network of workstations

Thank you!