Symbolic Model Checking
With SAT

Edmund M. Clarke, Jr.
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

(A. Biere, A. Cimatti, Y. Zhu,
A. Gupta, J. Kukula, D. Kroening, O. Strichman)

Temporal Logic Model Checking

Technique for verifying finite state machines:

- Specification – temporal logic.
- Implementation or Model – finite state machine.

Advantages (at least in theory!)

- Always terminates.
- Automatic.
- Usually fast.
- Can handle partially specified models.
- Counterexample if specification is false.

Symbolic Model Checking

Method used by most "industrial strength" model checkers:

- uses Boolean encoding for state machine and sets of states.
- can handle much larger designs – hundreds of state variables.
- BDDs traditionally used to represent Boolean functions.

Problems with BDDs

- BDDs are a canonical representation. Often become too large.
- Variable ordering must be uniform along paths.
- Selecting right variable ordering very important for obtaining small BDDs.
  - Often time consuming or needs manual intervention.
  - Sometimes, no space efficient variable ordering exists.

We describe an alternative approach to symbolic model checking that uses SAT procedures.

Advantages of SAT Procedures

- SAT procedures also operate on Boolean expressions but do not use canonical forms.
- Do not suffer from the potential space explosion of BDDs.
- Different split orderings possible on different branches.
- Very efficient implementations available.

Bounded Model Checking
(Clarke, Biere, Cimatti, Fujita, Zhu)

- Bounded model checking uses a SAT procedure instead of BDDs.
- We construct Boolean formula that is satisfiable iff there is a counterexample of length k.
- We look for longer and longer counterexamples by incrementing the bound k.
- After some number of iterations, we may conclude no counterexample exists and specification holds.
- For example, to verify safety properties, number of iterations is bounded by diameter of finite state machine.

Main Advantages of Our Approach

- Bounded model checking finds counterexamples fast. This is due to depth first nature of SAT search procedures.
- It finds counterexamples of minimal length. This feature helps user understand counterexample more easily.
- It uses much less space than BDD based approaches.
- Does not need manually selected variable order or costly reordering. Default splitting heuristics usually sufficient.
- Bounded model checking of LTL formulas does not require a tableau or automaton construction.

Implementation

- We have implemented a tool BMC for our approach.
- It accepts a subset of the SMV language.
- Given k, BMC outputs a formula that is satisfiable iff counterexample exists of length k.
- If counterexample exists, a standard SAT solver generates a truth assignment for the formula.

Performance

- We give examples where BMC significantly outperforms BDD based model checking.
- In some cases BMC detects errors instantly, while SMV fails to construct BDD for initial state.

Outline

- Bounded Model Checking:
  - Definitions and notation.
  - Example to illustrate bounded model checking.
  - Reduction of bounded model checking for LTL to SAT.
  - Experimental results.
  - Tuning SAT checkers for bounded model checking
  - Efficient computation of diameters
- Abstraction / refinement with SAT
- Directions for future research.

Basic Definitions and Notation

- We use linear temporal logic (LTL) for specifications.
- Basic LTL operators:

next time	'**X**'	*eventuality*	'**F**'
globally	'**G**'	*until*	'**U**'
release	'**R**'		

- Only consider existential LTL formulas $\mathbf{E}f$, where
  - **E** is the existential path quantifier, and
  - f is a temporal formula with no path quantifiers.
- Recall that **E** is the dual of the universal path quantifier **A**.
- Finding a witness for $\mathbf{E}f$ is equivalent to finding a counterexample for $\mathbf{A}\neg f$.

Definitions and Notation (Cont.)

- System described as a Kripke structure $M = (S, I, T, \ell)$, where
  - S is a finite set of states,
  - I is the set of initial states,
  - $T \subseteq S \times S$ is the transition relation, and
  - $\ell\colon S \to \mathcal{P}(\mathcal{A})$ is the state labeling.
- We assume every state has a successor state.

Definitions and Notation (Cont.)

- In symbolic model checking, a state is represented by a vector of state variables $s = (s(1), \dots, s(n))$.
- We define propositional formulas $f_I(s)$, $f_T(s,t)$ and $f_p(s)$ as follows:
  - $f_I(s)$ iff $s \in I$,
  - $f_T(s,t)$ iff $(s,t) \in T$, and
  - $f_p(s)$ iff $p \in \ell(s)$.
- We write $T(s,t)$ instead of $f_T(s,t)$, etc.

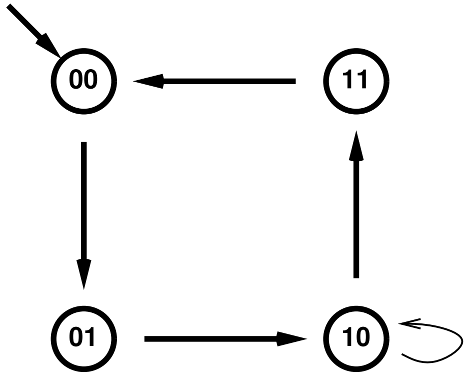
Definitions and Notation (Cont.)

- Will sometimes write $s \to t$ when $(s, t) \in T$.
- If $\pi = (s_0, s_1, \ldots)$, then $\pi(i) = s_i$ and $\pi^i = (s_i, s_{i+1}, \ldots)$.
- π is a path if $\pi(i) \to \pi(i+1)$ for all i.
- $\mathbf{E}f$ is true in M ($M \models \mathbf{E}f$) iff there is a path π in M with $\pi \models f$ and $\pi(0) \in I$.
- Model checking is the problem of determining the truth of an LTL formula in a Kripke structure. Equivalently,

  Does a witness exist for the LTL formula?

Example To Illustrate New Technique

Two-bit counter with an erroneous transition:



- Each state s is represented by two state variables $s[1]$ and $s[0]$.
- In initial state, value of the counter is 0. Thus, $I(s) = \neg s[1] \wedge \neg s[0]$.
- Let $inc(s, s') = (s'[0] \leftrightarrow \neg s[0]) \wedge (s'[1] \leftrightarrow (s[0] \oplus s[1]))$
- Define $T(s, s') = inc(s, s') \vee (s[1] \wedge \neg s[0] \wedge s'[1] \wedge \neg s'[0])$
- Have deliberately added erroneous transition!!

Example (Cont.)

- Suppose we want to know if counter will eventually reach state (11).
- Can specify the property by $\mathbf{AF}q$, where $q(s) = s[1] \wedge s[0]$.

  On all execution paths, there is a state where $q(s)$ holds.
- Equivalently, we can check if there is a path on which counter never reaches state (11).
- This is expressed by $\mathbf{EG}p$, where $p(s) = \neg s[1] \vee \neg s[0]$.

  There exists a path such that $p(s)$ holds globally along it.

Example (Cont.)

- In bounded model checking, we consider paths of length k.
- We start with $k = 0$ and increment k until a witness is found.
- Assume k equals 2. Call the states s_0, s_1, s_2.
- We formulate constraints on s_0, s_1, and s_2 in propositional logic.
- Constraints guarantee that (s_0, s_1, s_2) is a witness for $\mathbf{EG}p$ and, hence, a counterexample for $\mathbf{AF}q$.

Example (Cont.)

- First, we constrain (s_0, s_1, s_2) to be a valid path starting from the initial state.
- Obtain a propositional formula

$$[\![\ M\ ]\!] = I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2).$$

Example (Cont.)

- Second, we constrain the shape of the path.
- The sequence of states s_0, s_1, s_2 can be a loop.
- If so, there is a transition from s_2 to the initial state s_0, s_1 or itself.
- We write ${}_lL = T(s_2, s_l)$ to denote the transition from s_2 to a state s_l where $l \in [0, 2]$.
- We define L as $\bigvee_{l=0}^{2} {}_lL$. Thus $\neg L$ denotes the case where no loop exists.

Example (Cont.)

- The temporal property $\mathbf{G}p$ must hold on (s_0, s_1, s_2).
- If no loop exists, $\mathbf{G}p$ does not hold and $[\![\ \mathbf{G}p\ ]\!]$ is $false$.
- To be a witness for $\mathbf{G}p$, the path must contain a loop (condition L, given previously).
- Finally, p must hold at every state on the path

$$[\![\ \mathbf{G}p\ ]\!] = p(s_0) \wedge p(s_1) \wedge p(s_2).$$

- We combine all the constraints to obtain the propositional formula

$$[\![\ M\ ]\!] \wedge ((\neg L \wedge false) \vee \bigvee_{l=0}^{2} ({}_lL \wedge [\![\ \mathbf{G}p\ ]\!])).$$

Example (Cont.)

- In this example, the formula is satisfiable.
- Truth assignment corresponds to counterexample path (00), (01), (10) followed by self-loop at (10).
- If self-loop at (10) is removed, then formula is unsatisfiable.

Sequential Multiplier Example

	SMV$_1$		SMV$_2$		SATO		PROVER	
bit	sec	MB	sec	MB	sec	MB	sec	MB
0	919	13	25	79	0	0	0	1
1	1978	13	25	79	0	0	0	1
2	2916	13	26	80	0	0	0	1
3	4744	13	27	82	0	0	1	2
4	6580	15	33	92	2	0	1	2
5	10803	25	67	102	12	0	1	2
6	43983	73	258	172	55	0	2	2
7	>17h		1741	492	209	0	7	3
8				>1GB	473	0	29	3
9					856	1	58	3
10					1837	1	91	3
11					2367	1	125	3
12					3830	1	156	4
13					5128	1	186	4
14					4752	1	226	4
15					4449	1	183	5
sum	71923		2202		23970		1066	

Model Checking: 16x16 bit sequential shift and add multiplier with overflow flag and 16 output bits.

DME Example

	SMV$_1$		SMV$_2$		SATO $k=5$		PROVER $k=5$		SATO $k=10$		PROVER $k=10$	
cells	sec	MB	sec	MB	sec	MB	sec	MB	sec	MB	sec	MB
4	846	11	159	217	0	3	1	3	3	6	54	5
5	2166	15	530	703	0	4	2	3	9	8	95	5
6	4857	18	1762	703	0	4	3	3	7	9	149	6
7	9985	24	6563	833	0	5	4	4	15	10	224	8
8	19595	31		$>$1GB	1	6	6	5	16	12	323	8
9	$>$10h				1	6	9	5	24	13	444	9
10					1	7	10	5	36	15	614	10
11					1	8	13	6	38	16	820	11
12					1	9	16	6	40	18	1044	11
13					1	9	19	8	107	19	1317	12
14					1	10	22	8	70	21	1634	14
15					1	11	27	8	168	22	1992	15

Model Checking: Liveness for one user in the DME.

“Buggy” DME Example

	SMV$_1$		SMV$_2$		SATO		PROVER	
cells	sec	MB	sec	MB	sec	MB	sec	MB
4	799	11	14	44	0	1	0	2
5	1661	14	24	57	0	1	0	2
6	3155	21	40	76	0	1	0	2
7	5622	38	74	137	0	1	0	2
8	9449	73	118	217	0	1	0	2
9	segmentation		172	220	0	1	1	2
10	fault		244	702	0	1	0	3
11			413	702	0	1	0	3
12			719	702	0	2	1	3
13			843	702	0	2	1	3
14			1060	702	0	2	1	3
15			1429	702	0	2	1	3

Model Checking: Counterexample for liveness in a buggy DME implementation.

Tuning SAT checkers for BMC
(O. Strichman, CAV00)

- Use the variable dependency graph for deriving a static variable ordering.
- Use the regular structure of $\mathbf{AG}p$ formulas to replicate conflict clauses:

$$\varphi : I_0 \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} p_i$$

The transition relation appears k times in φ, each time with different variables.

This symmetry indicates that under certain conditions, for each conflict clause we can compute additional $k-1$ clauses 'for free'.

Tuning SAT checkers for BMC (cont'd)

- Use the incremental nature of BMC to reuse conflict clauses.

  Some of the clauses that were computed while solving BMC with e.g. k=10 can be reused when solving the subsequent instance with k=11.

- Restrict decisions to model variables only (ignore CNF auxiliary vars).

  It is possible to decide the formula without the auxiliary variables (they will be implied). In many examples they are 80%-90% of the variables in the CNF instance.

- ...

BMC of some hardware designs w/wo tuning SAT

Design #	K	RB1	RB2	Grasp	Tuned
1	18	7	6	282	3
2	5	70	8	1.1	0.8
3	14	597	375	76	3
4	24	690	261	510	12
5	12	803	184	24	2
6	22		356		18
7	9		2671	10	2
8	35			6317	20
9	38			9035	25
10	31				312
11	32	152	60		
12	31	1419	1126		
13	14		3626		

RuleBase is IBM's BDD based symbolic model-checker.

RB1 - RuleBase first run (with BDD dynamic reordering).

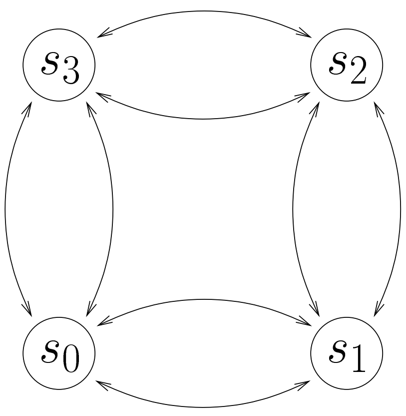
RB2 - RuleBase second run (without BDD dynamic reordering).

Diameter

- Diameter d: Least number of steps to reach all reachable states. If the property holds for $k \geq d$, the property holds for all reachable states.
- Finding d is computationally hard:
  - State s is reachable in j steps:
  $$R_j(s) := \exists s_0, \ldots, s_j : s = s_j \wedge I(s_0) \wedge \bigwedge_{i=0}^{j-1} T(s_i, s_{i+1})$$
  - Thus, k is greater or equal than the diameter d if
  $$\forall s : R_{k+1}(s) \Longrightarrow \exists j \leq k : R_j(s)$$
  This requires an efficient QBF checker!

A Compromise: Recurrence Diameter

- Recurrence Diameter rd: Least number of steps n such that all valid paths of length n have at least one cycle



Example:

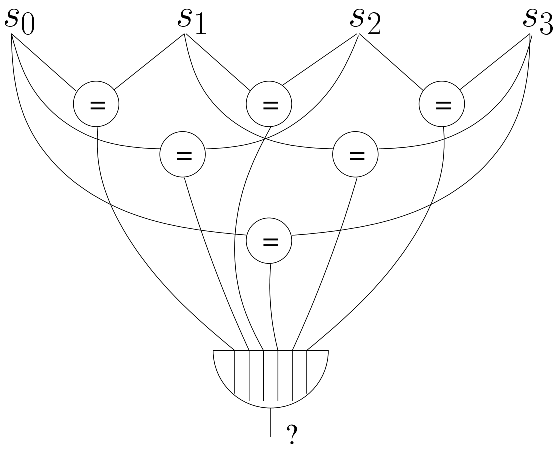
- All states are reachable from s_0 in two steps, i.e., $d = 2$
- All paths with at least one cycle have a minimum length of four steps, i.e., $rd = 4$

- Theorem: Recurrence Diameter rd is an upper bound for the Diameter d

Testing the Recurrence Diameter

- Recurrence Diameter test in BMC:
  Find cycles by comparing all states with each other

$$\forall s_0, \ldots, s_k : I(s_0) \wedge \bigwedge_{i=0}^{n-1} T(s_i, s_{i+1}) \Longrightarrow \bigvee_{l=0}^{k-1} \bigvee_{j=l+1}^{k} s_l = s_j$$



- Size of CNF: $O(k^2)$
- Too expensive for big k

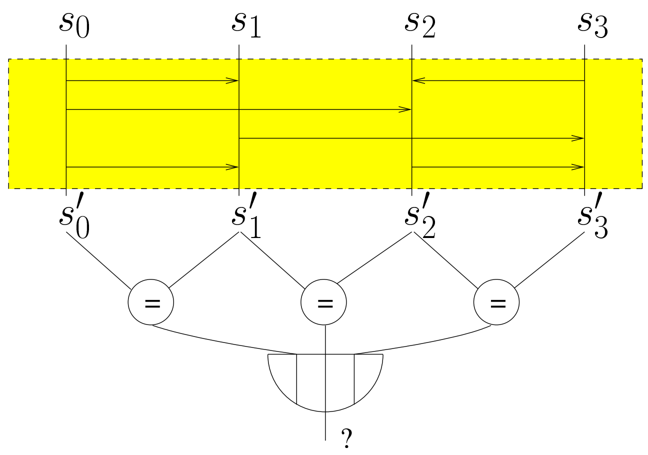
Recurrence Diameter Test using Sorting Networks (D. Kroening)

- Idea: Look for cycles using a Sorting Network
- First, sort the $k+1$ states symbolically:

$$s'_0, \ldots, s'_k \text{ are permutation of } s_0, \ldots, s_k \text{ such that } s'_0 \leq s'_1 \leq \ldots \leq s'_k$$

- Sorting can be done with CNF of size $O(k \log k)$. Practical implementations, e.g., Bitonic sort, have size $O(k \log^2 k)$.
- Now only check neighbors in the sorted sequence:

$$(\exists i : s'_i = s'_{i+1}) \iff (\exists l, j : l \neq j \wedge s_l = s_j)$$



Recurrence Diameter Test using Sorting Networks

- Example CNF size comparison (without transition system):

k	$O(k^2)$ Alg.		$O(k \log^2 k)$ Alg.	
	Variables	Clauses	Variables	Clauses
32	5,777	25,793	7,862	34,493
64	22,817	104,833	21,494	95,341
128	90,689	422,657	56,438	252,109
256	361,601	1,697,281	143,606	644,557
512	1,444,097	6,802,433	356,342	1,604,813

Future Research Directions

We believe our techniques may be able to handle much larger designs than is currently possible. Nevertheless, there are a number of directions for future research:

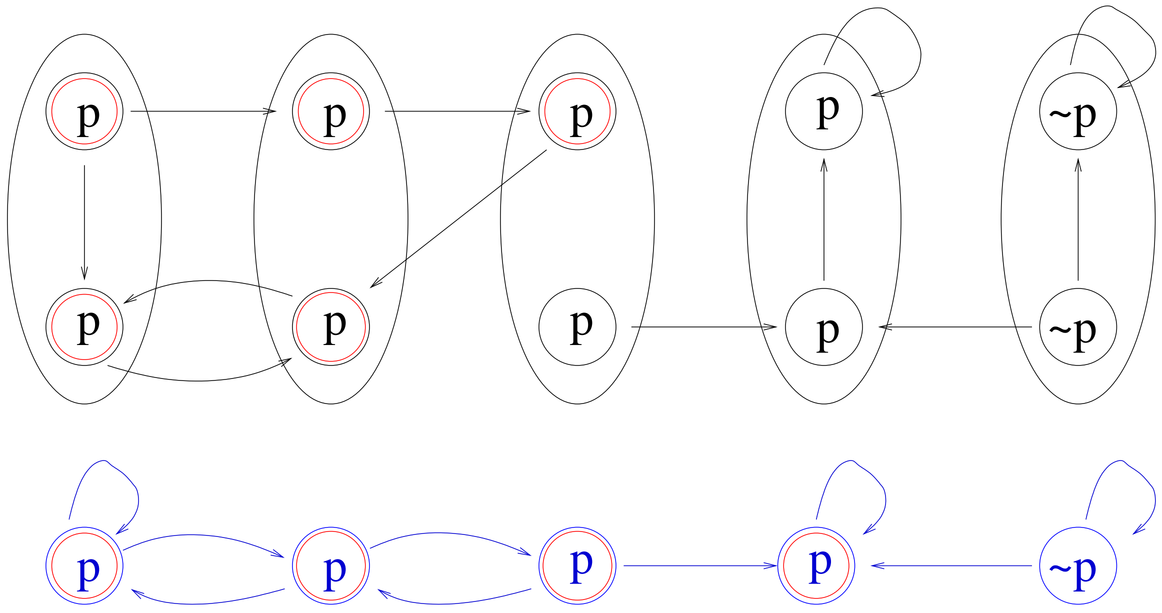
Bounded Model Checking:

- Techniques for generating short propositional formulas need to be studied.
- Want to investigate further the use of domain knowledge to guide search in SAT procedures.
- A practical decision procedure for QBF would also be useful.
- Combining bounded model checking with other reduction techniques is also a fruitful direction.

SAT based Abstraction-Refinement
(Clarke, Gupta, Kukula, Strichman, CAV'02)

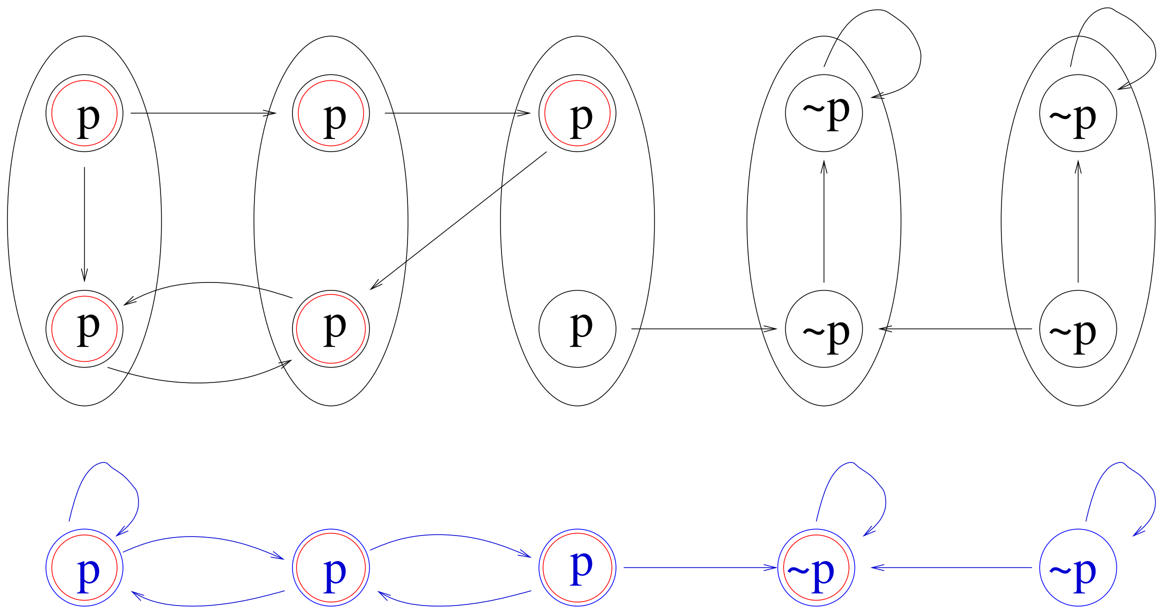
$\hat{M}$ is a conservative abstraction of M if the following preservation theorem holds:

$$\hat{M} \models \mathbf{G}p \Rightarrow M \models \mathbf{G}p$$



Converse of Preservation Theorem

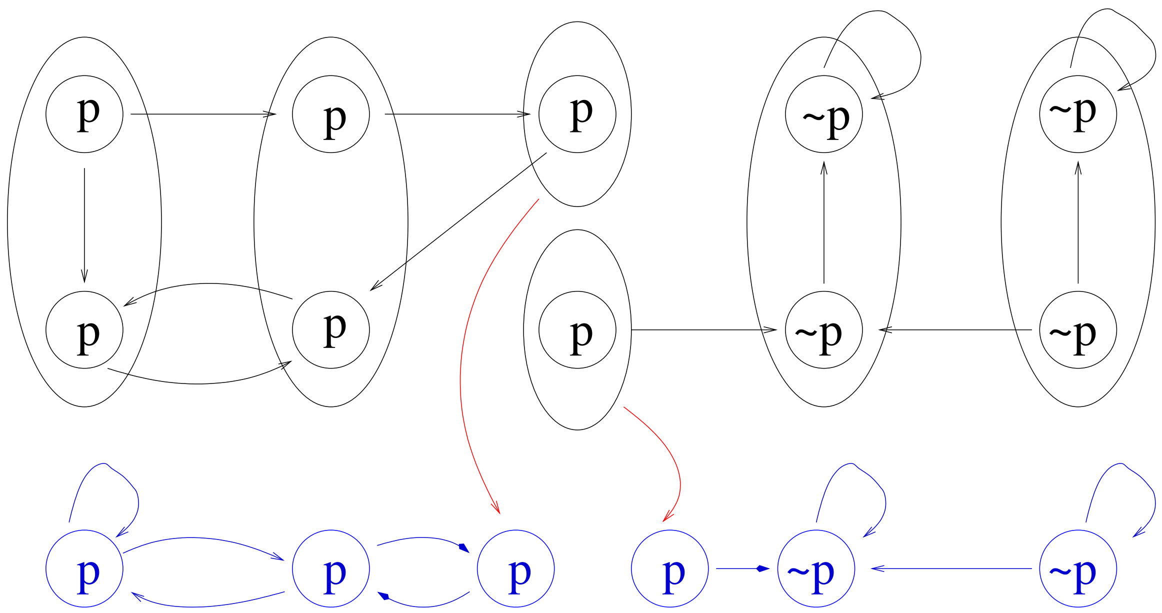
$$\hat{M} \not\models \mathbf{G}p \not\Rightarrow M \not\models \mathbf{G}p$$



Counterexample is spurious. Abstraction is too coarse.

Refinement

To refine the system, we need to split abstract states.



Abstraction-Refinement

1. Generate an initial abstraction.
2. Build abstract machine $\hat{M}$. Model check $\hat{M}$. If $\hat{M} \models \varphi$, then $M \models \varphi$. Return TRUE.
3. If $\hat{M} \not\models \varphi$, check the counterexample on the concrete model. If the counterexample is real, $M \not\models \varphi$. Return FALSE.
4. Refine the abstraction, and go to step 2.

Abstraction Function

- Partition variables V into visible($\mathcal{V}$) and invisible($\mathcal{I}$) variables. $\mathcal{V} = \{v_1, \ldots, v_k\}$.
- States having the same values for the visible variables are mapped to the same abstract state.

$$\left.\begin{array}{cccc} x1 & x2 & x3 & x4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{array}\right\} \begin{array}{cc} x1 & x2 \\ 0 & 0 \end{array}$$

- Refinement : Move variables from $\mathcal{I}$ to $\mathcal{V}$.

Checking the Counterexample

- Counterexample : $\langle \hat{s}_1, \hat{s}_2, \ldots \hat{s}_m \rangle$
- Set of concrete paths for counterexample:

$$\psi_m = \{ I(s_1) \wedge \bigwedge_{i=1}^{m-1} R(s_i, s_{i+1}) \wedge \bigwedge_{i=1}^{m} \bigwedge_{v \in \mathcal{V}} s_i(v) = \hat{s}_i(v) \}$$

- The right-most conjunct is a restriction of the visible variables to their values in the counterexample.
- Solve ψ_m with a SAT solver.
- If ψ_m is satisfiable we found a real bug; else, refine.

Refinement

- Abstraction corresponds to unifying states. Therefore, refinement is the problem of splitting (separating) abstract states.

- We use techniques like ILP and decision-tree learning to refine the system.

- We also try to extract information from the SAT checker in order to guide the refinement.