Verification of Large Industrial Circuits Using SAT Based Reparameterization and Automated Abstraction-Refinement

Pankaj Chauhan

Ph.D. Thesis Proposal

Computer Science Dept., Carnegie Mellon University

Challenges for a Symbolic Model Checker

- State, transition relation representation
  - Characteristic function based
  - Non-characteristic function based
- Image Computation
- State Space Traversal
- Abstraction/Refinement

State Exploration is the Key

- Bounded model checking (BMC) the first step for finding bugs in the large industrial circuits
- BMC is also a key element of automatic abstraction-refinement algorithms
- For large circuits, even BMC fails[Chauhan et al. FMCAD'02, Kroening et al. VMCAI'03]

Scope of the Thesis

Increase the capacity of state exploration engines by leveraging efficient SAT procedures.

- Image computation algorithms using SAT based enumeration
- Symbolic simulation
- Abstraction-refinement

SAT Based Image Computation

Basic Reachability Algorithm

$$Img(S(x)) = \exists x, i.T(x, i, x') \wedge S(x)$$

REACHABILITY(S_0)

$S_{reach} \leftarrow \phi$

$i \leftarrow 0$

while $(S_i \neq \phi)$ {

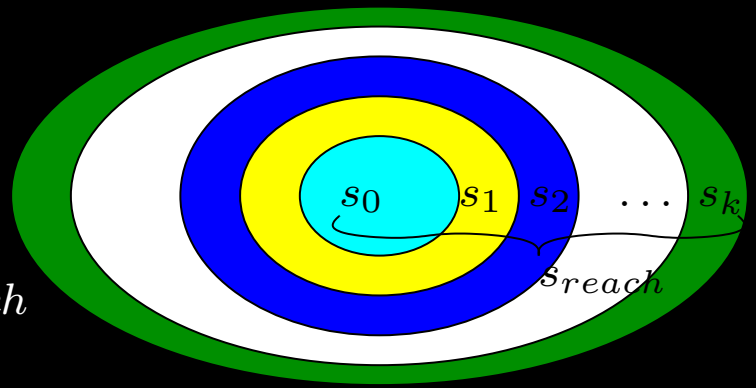
$S_{reach} \leftarrow S_{reach} \cup S_i$

$S_{i+1} \leftarrow Img(S_i) \backslash S_{reach}$

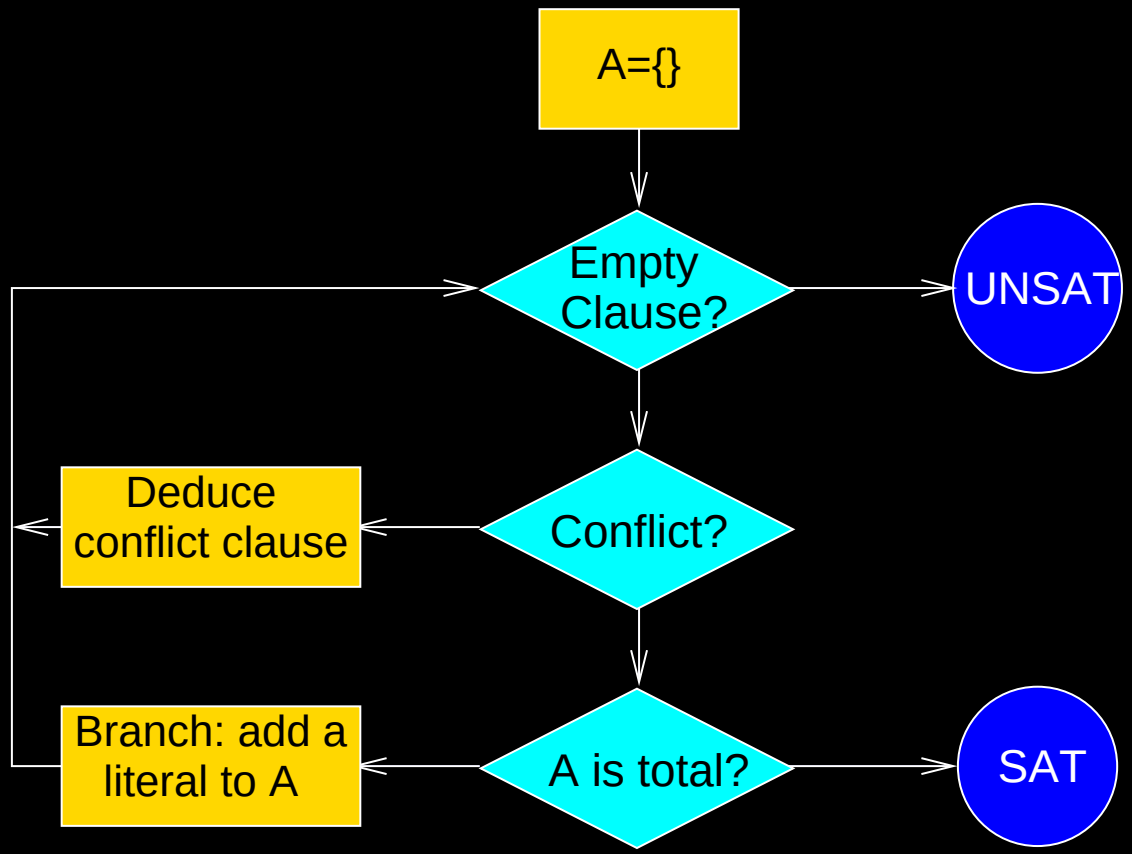
$i \leftarrow i + 1$

}

return S_{reach}



DPLL Style SAT Flowchart



SAT Based Reachability

A SAT Solver checks the validity of $\exists X, Y.f(X, Y)$.

- **Representation:** S_i and S_{reach} as DNF cubes
- **Computing Images:** Enumerate satisfying cubes $\Rightarrow$ compute $\exists X.f(X, Y)$.
- **Detection of Fixed Point:** At least one satisfying cube

SAT Based Images

Enumerate cubes in next state (x') variables to the formula

$$S_{i-1}(x) \wedge T(x, i, x') \wedge \neg S_{reach}(x')$$

- Convert S_{i-1} and T to CNF using intermediate variables.
- S_{reach} in DNF $\Rightarrow \neg S_{reach}$ in CNF
- Add each satisfying assignment as to S_i and in the SAT as *blocking clause*

Problems

- Time complexity
  - To many cubes to enumerate, each cube represents one state in S_i.
- Space complexity
  - Storage of cubes expensive and redundant
  - Cubes can be merged

Solving Time Problem

Given an assignment from SAT, reduce the number of assigned literals $\Rightarrow$ **Cube Enlargement**

- Circuit based heuristic to propagate free variables to infer the next state variable assignments that can be safely ignored
- An enlarged cube represents multiple states
- Enlarged cube is added as a blocking clause

Solving Space Problem

$$\left.\begin{array}{l} x_1 \wedge x_5 \wedge x_6 \\ x_1 \wedge x_5 \wedge \neg x_6 \end{array}\right\}$$ can be combined to form $x_1 \wedge x_5$

- Due to S_{reach} constraint, $x_1 \wedge x_5 \wedge x_6 \wedge x_8$ can never arise.
- Therefore, sufficient to check merging with cubes that have the same set of variables

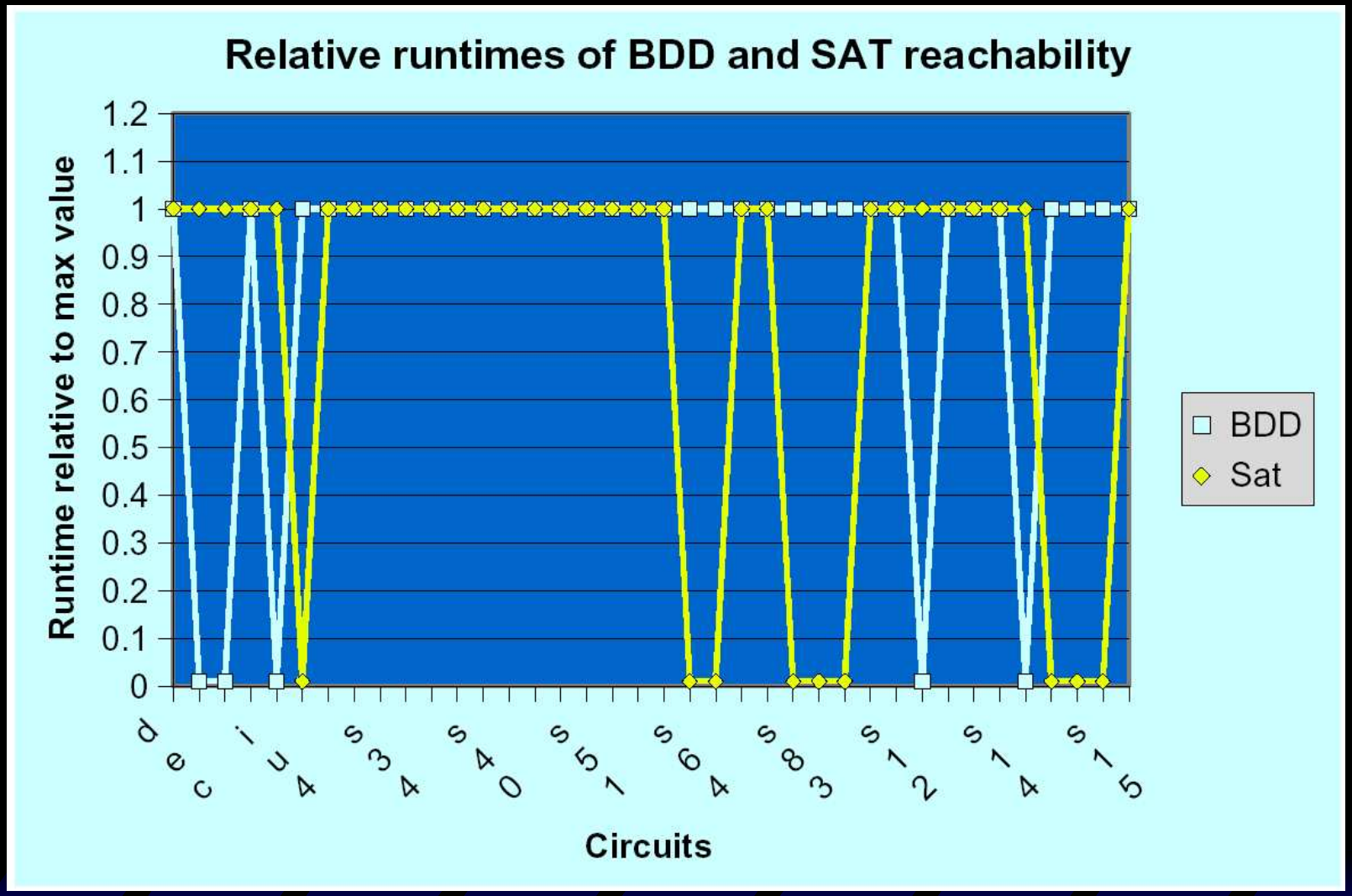
Solving Space Problem

A hash table based data structures to efficiently add a newly enumerated cube.

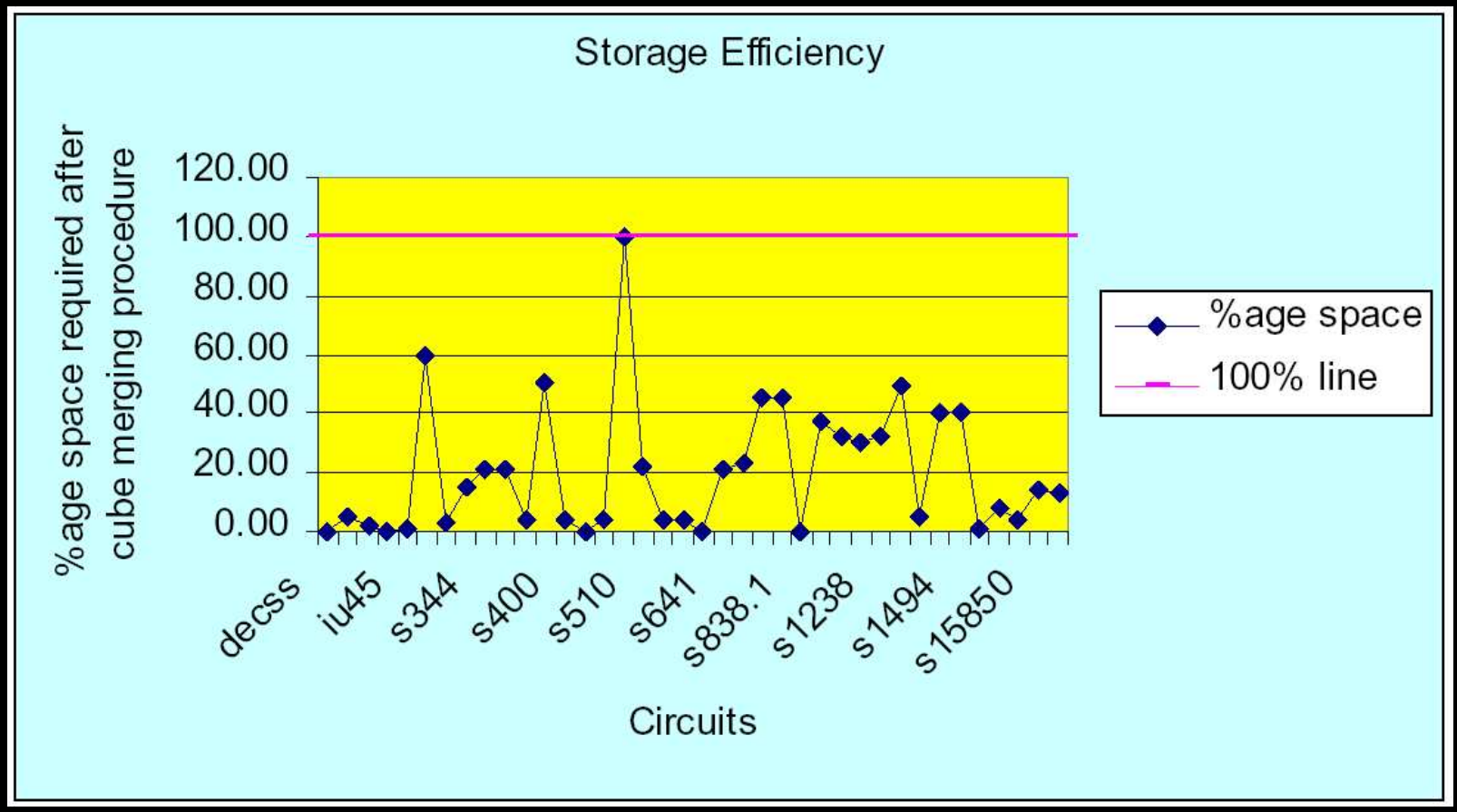
- For each added clause, use a hash table to find clauses with the same set of variables
- A smaller hash to find clauses differing only in one literal.

  Can use zDDs or other data structure.

Relative Runtime v/s BDD



Space Savings



SAT Enumeration Summary

- Algorithm is relatively unaffected by the number of variables to be quantified.
- Important to reduce the number of cubes to enumerate.
- The biggest bottleneck is the clausal representation.

Work in Progress

- Benchmarking on processor designs of McMillan et al. CAV'02, Lahiri et al. CAV'02
- Use of interpolation proofs for non-clausal representation.

SAT Based Reparameterization

Characteristic Functions

$$V = \{v_1, v_2\}$$

$$S = \{01, 10\}$$

Characteristic function of S is

$$\mathcal{X}(V) = (v_1 \wedge \neg v_2) \vee (\neg v_1 \wedge v_2).$$

S is given by

$$S = \{V | \mathcal{X}(V) = 1\}$$

Parametric Representation[CM'90]

$$\begin{aligned} \text{For } S &= \{01, 10\} \\ f_1(p_1) &= p_1 \\ f_2(p_1) &= \neg p_1 \end{aligned}$$

p_1	f_1	f_2
0	0	1
1	1	0

Parametric Representation[CM'90]

For $S = \{01, 10\}$

$$f_1(p_1) = p_1$$

$$f_2(p_1) = \neg p_1$$

p_1	f_1	f_2
0	0	1
1	1	0

Or

$$h_1(p_1, p_2) = p_1 \vee p_2$$

$$h_2(p_1, p2) = \neg p_2 \wedge \neg p_2$$

p_1	p_2	h_1	h_2
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Parametric Representation

If $S(V)$ is represented parametrically with a vector of n functions $F(P) = (f_1(P), f_2(P), \ldots, f_n(P))$ over k parameters $P = (p_1, p_2, \ldots, p_k)$, then

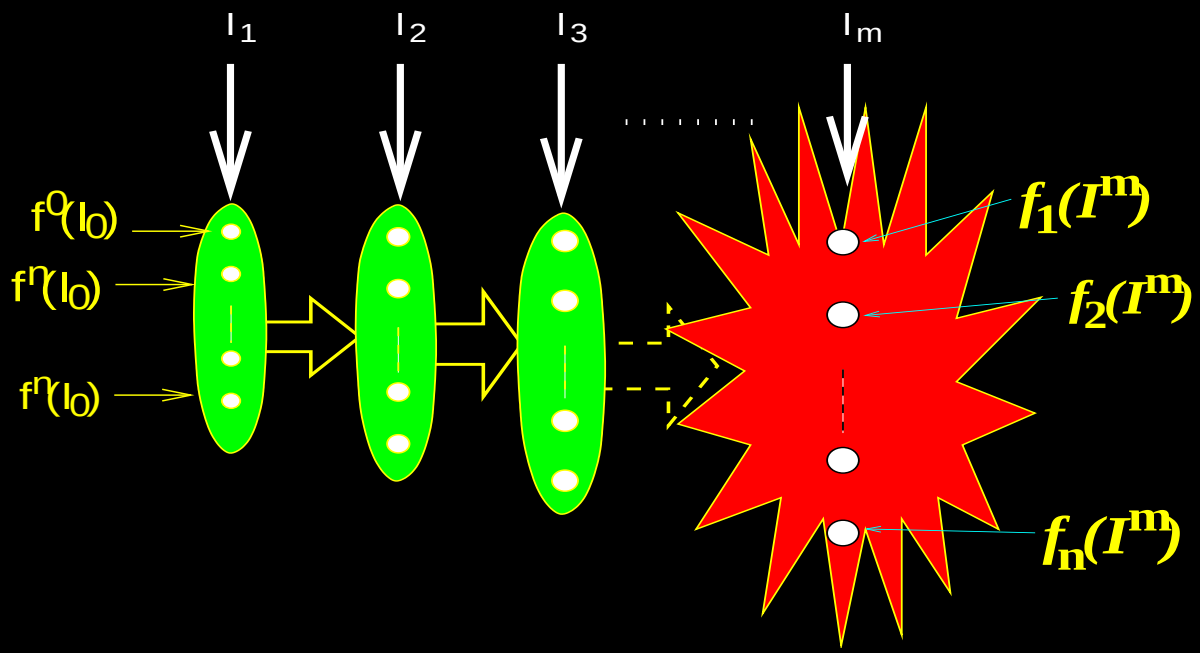
$$S(V) = \left\{ V | \exists P. \begin{pmatrix} v_1 = f_1(P) & \wedge \\ v_2 = f_2(P) & \wedge \\ \vdots & \wedge \\ v_n = f_n(P) & \end{pmatrix} \right\}.$$

Parametric $\Rightarrow$ Characteristic

A parametric representation can be easily converted to a characteristic function by using the following equation.

$$\mathcal{X}(V) = \exists P. \begin{pmatrix} v_1 \leftrightarrow f_1(P) & \wedge \\ v_2 \leftrightarrow f_2(P) & \wedge \\ \vdots & \wedge \\ v_n \leftrightarrow f_n(P) & \end{pmatrix}$$

Parametric Representation in Symbolic Simulation



If $I^m = I_0 \cup I_1 \cup \ldots \cup I_m$, then after m steps of symbolic simulation,

$$S_f(V) = \{V | \exists I^m . v_1 = f_1(I^m) \wedge \ldots \wedge v_n = f_n(I^m)\}$$

Parametric Representation in Symbolic Simulation

Problems:

- The number of parameters, $|I^m|$, gets larger and larger with the number of simulation steps.
- Functions $f_1, f_2, \ldots$ keep on getting bigger and bigger.

Reparameterization

- Find functions $h_1(P), h_2(P), \ldots, h_n(P)$ in parameters P, such that $|P| \ll |I^m|$ and $S_h(V) = S_f(V)$.

Reparameterization

- Find functions $h_1(P), h_2(P), \ldots, h_n(P)$ in parameters P, such that $|P| \ll |I^m|$ and $S_h(V) = S_f(V)$.
- It has been shown that a set of vectors in n variables can be represented by parametric functions of n variables.

Reparameterization

- Find functions $h_1(P), h_2(P), \ldots, h_n(P)$ in parameters P, such that $|P| \ll |I^m|$ and $S_h(V) = S_f(V)$.
- It has been shown that a set of vectors in n variables can be represented by parametric functions of n variables.
- We would like $|P| \leq n$.

Reparameterization Algorithm

- The algorithm computes functions $h_1(P), h_2(P), \ldots, h_n(P)$ in that order.

Reparameterization Algorithm

- The algorithm computes functions $h_1(P), h_2(P), \ldots, h_n(P)$ in that order.
- $h_i(p_1, \ldots, p_i)$. This means that h_i depends on the fi rst i parameters only. The parantheses denote the arguments to the function in the rest of the talk.

Reparameterization Algorithm

- The algorithm computes functions $h_1(P), h_2(P), \ldots, h_n(P)$ in that order.
- $h_i(p_1, \ldots, p_i)$. This means that h_i depends on the fi rst i parameters only. The parantheses denote the arguments to the function in the rest of the talk.
- We think of each new parameter p_i as free variable for the i^{th} state bit v_i.

A Known Decomposition [CM'90]

$$h_i(p_1, \ldots, p_i) = h_i^1(p_1, \ldots, p_{i-1}) \vee p_i \cdot h_i^c(p_1, \ldots, p_{i-1}).$$

where,

- $h_i^1(p_1, \ldots, p_{i-1})$: Boolean condition under which v_i is forced to 1

A Known Decomposition [CM'90]

$$h_i(p_1, \ldots, p_i) = h_i^1(p_1, \ldots, p_{i-1}) \vee p_i \cdot h_i^c(p_1, \ldots, p_{i-1}).$$

where,

- $h_i^1(p_1, \ldots, p_{i-1})$: Boolean condition under which v_i is forced to 1
- $h_i^0(p_1, \ldots, p_{i-1})$: Boolean condition under which v_i is forced to 0

A Known Decomposition [CM'90]

$$h_i(p_1, \ldots, p_i) = h_i^1(p_1, \ldots, p_{i-1}) \vee p_i \cdot h_i^c(p_1, \ldots, p_{i-1}).$$

where,

- $h_i^1(p_1, \ldots, p_{i-1})$: Boolean condition under which v_i is forced to 1
- $h_i^0(p_1, \ldots, p_{i-1})$: Boolean condition under which v_i is forced to 0
- $h_i^c(p_1, \ldots, p_{i-1})$: Boolean condition under which v_i is free to choose any value

A Known Decomposition [CM'90]

For $S = \{01, 10\}$, suppose $h_1(p_1) = p_1$, then $p_1 = 0 \Rightarrow v_1 = 0 \Rightarrow v_2 = 1$. Thus, $h_2^1(p1) = \neg p_1$. Moreover,

$$h_i^c = \neg(h_i^1 \vee h_i^0) = \neg h_i^1 \wedge \neg h_i^0$$

i.e., h_i^1, h_i^0 and h_i^c are mutually exclusive and complete.

Computing h_i^1 and h_i^c

Restriction Function

$$\rho_i(p_1, \ldots, p_{i-1}, I^m) = \bigwedge_{j=1}^{i-1} h_j(p_1, \ldots, p_j) = f_j(I^m)$$

Set of input vectors for which the functions f_1 to f_{i-1} compute the same values as those computed by h_1 to h_{i-1} for the given parameter assignment $p_1, p_2, \ldots, p_{i-1}$.

Note that $\rho_1 = 1$.

Computing h_i^1 and h_i^c

$$h_i^1(p_1, \ldots, p_{i-1}) = \forall I^m . \rho_i(p_1, \ldots, p_{i-1}, I^m) \rightarrow f_i(I^m) = 1$$

Computing h_i^1 and h_i^c

$$h_i^1(p_1, \ldots, p_{i-1}) = \forall I^m . \rho_i(p_1, \ldots, p_{i-1}, I^m) \rightarrow f_i(I^m) = 1$$

$$h_i^0(p_1, \ldots, p_{i-1}) = \forall I^m . \rho_i(p_1, \ldots, p_{i-1}, I^m) \rightarrow f_i(I^m) = 0$$

Computing h_i^1 and h_i^c

$$h_i^1(p_1, \ldots, p_{i-1}) = \forall I^m . \rho_i(p_1, \ldots, p_{i-1}, I^m) \rightarrow f_i(I^m) = 1$$

$$h_i^0(p_1, \ldots, p_{i-1}) = \forall I^m . \rho_i(p_1, \ldots, p_{i-1}, I^m) \rightarrow f_i(I^m) = 0$$

Reparameterization is very expensive with BDDS, even with an altogether different set union based approach.

High Level Algorithm

ORDEREDREPARAM($\bar{f}(I^m) = (f_1(I^m), \ldots, f_n(I^m))$)

for $i = 1$ **to** n

$\rho_i \leftarrow 1$

for $j = 1$ **to** $i - 1$

$\rho_i \leftarrow \rho_i \wedge (h_j = f_j)$

$h_i^1 \leftarrow \forall I^m.(\rho_i \leftarrow f_i = 1)$

$h_i^0 \leftarrow \forall I^m.(\rho_i \leftarrow f_i = 0)$

$h_i^c \leftarrow \neg(h_i^1 \vee h_i^0)$

$h_i \leftarrow h_i^1 \vee (p_i \wedge h_i^c)$

endfor

return $(h_1(P), h_2(P), \ldots, h_n(P))$

SAT Based Enumeration

$$\begin{aligned} h_i^{\alpha}(p_1,\ldots,p_{i-1}) &= \forall I^m.\rho_i(p_1,\ldots,p_{i-1},I^m) \rightarrow f_i(I^m) = \alpha \\ &= \neg\exists I^m.\neg\left(\rho_i(p_1,\ldots,p_{i-1},I^m) \rightarrow f_i(I^m) = \alpha\right) \\ &= \neg\exists I^m.\rho_i(p_1,\ldots,p_{i-1},I^m) \wedge f_i(I^m) \neq \alpha \end{aligned}$$

- SAT based enumeration computes $\neg h_i^{\alpha}$ in DNF, thus h_i^1 and h_i^0 are in CNF.
- Enumerates on $\rho_i(p_1,\ldots,p_{i-1},I^m)$ to compute both $\neg h_i^1$ and $\neg h_i^0$ in a single call.

Incremental SAT is the key to the performance.

Incremental SAT

Enumeration formula for h_i^α is very similar to the formula for h_{i-1}^α.

- Remove blocking clauses and the conflict clauses derived from them from ρ_{i-1}.
- Add clauses for $h_{i-1} = f_{i-1}$, repeat

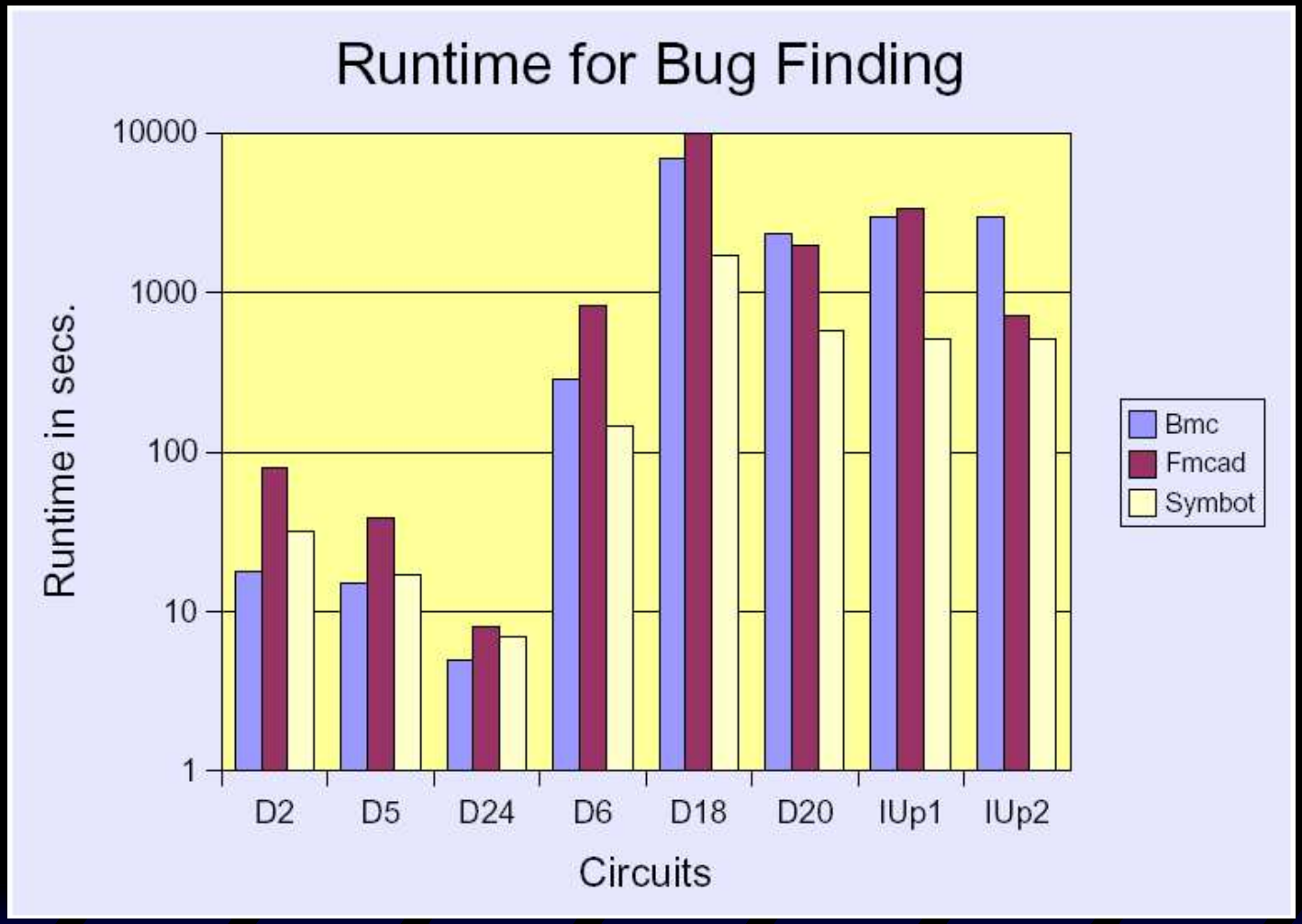
Experimental Results

ckt	# regs	# inp	bug len	bmc time	fmcad bug time	symbot time	symbot max len	symbot time	# reps
D2$^+$	94	11	15	18	79	32	221	1000	8
D5$^+$	343	7	32	15	38.2	17	127	1000	13
D24	223	47	10	5	8	7	543	1000	9
D6	161	16	20	289	833	145	64	1000	5
D18	498	247	28	6834	9955	1698	56	3000	7
D20	532	30	14	2349	1947	574	89	3000	9
IUp1	4494	361	true	3000*	3350	-	183	3000	45
IUp2	4494	361	true	3000*	712	-	183	3000	45

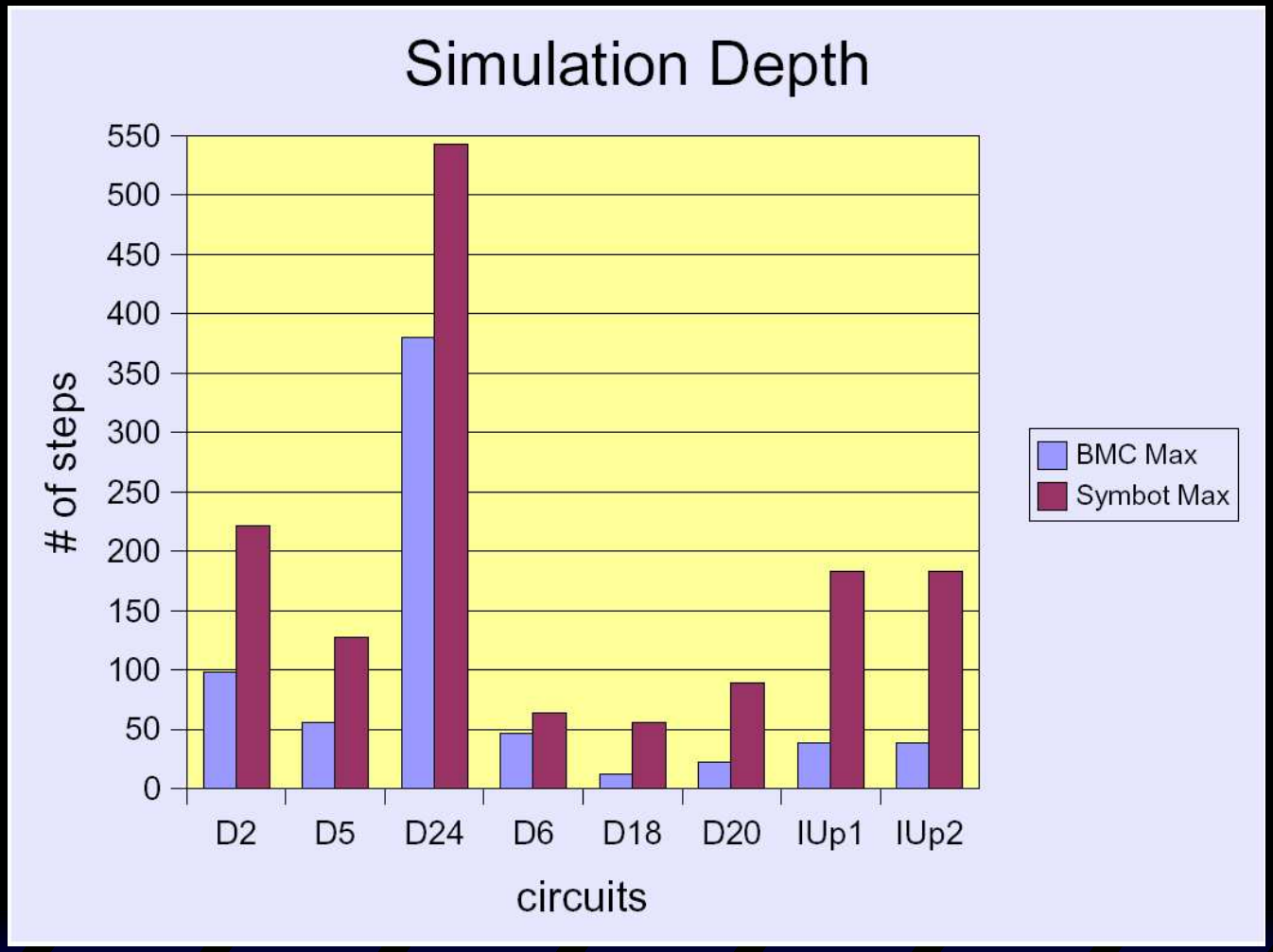
* BMC could complete only 39 steps of simulation before running out of space

$^+$ BDD based symbolic simulator could do only these two circuits.

Runtime for Bug Finding



Simulation Depth v/s BMC



Ongoing and Future Work

- Counterexample generation
- Safety property checking
- Handling of general transition constraints, SMV style `INVAR`, `TRANS`
- Effect of variable orders, not as severe as BDDs

Ongoing and Future Research

- Improve enumeration algorithm
- Effects of variable ordering on parameterized representation
- Proving properties true?

Abstraction Refinement

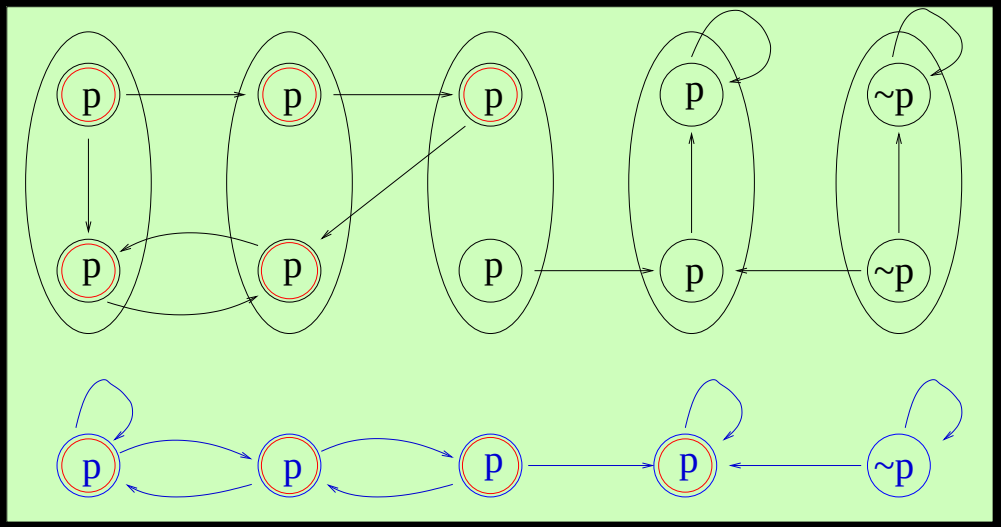
CEGAR Loop

1. Generate an initial abstraction function h.
2. Build abstract machine $\hat{M}$ based on h. 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 h, and go to step 2.

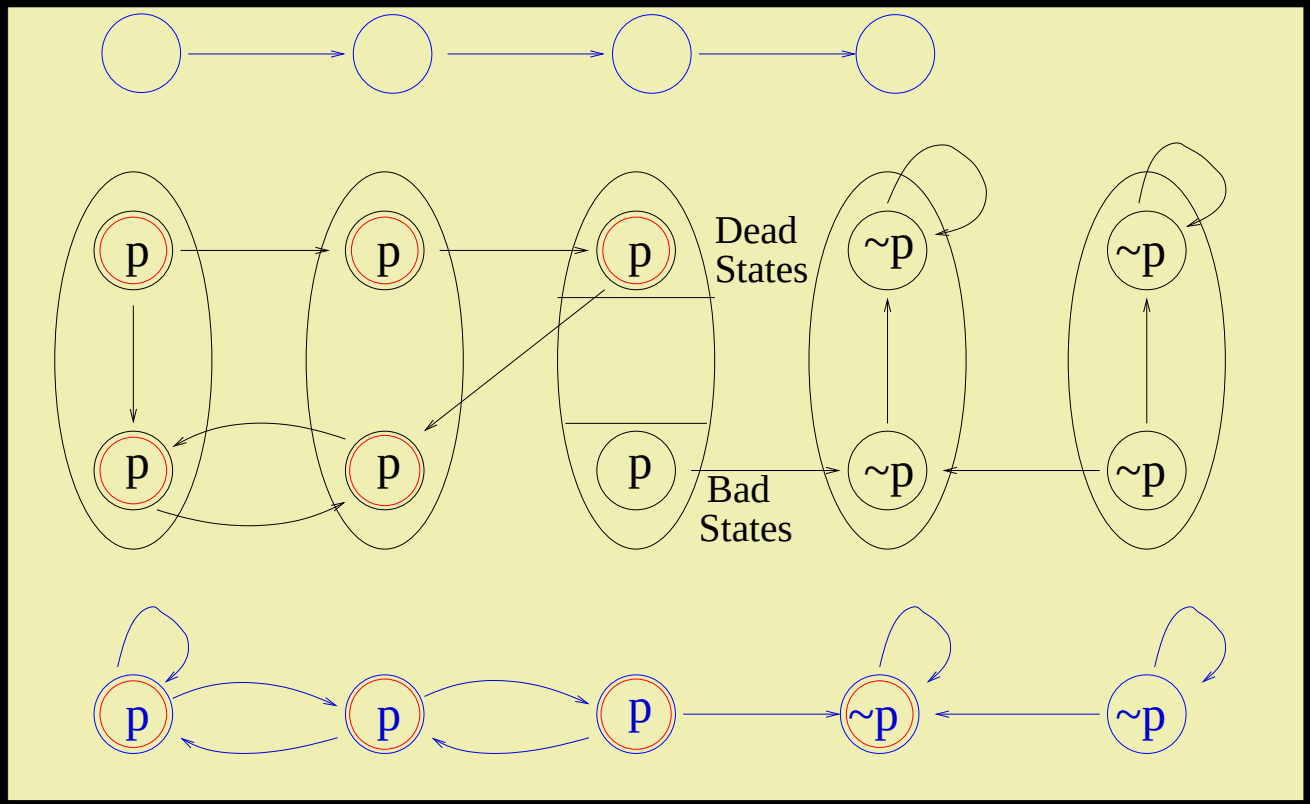
Preservation Theorem

Let $\hat{M}$ be an abstraction of M corresponding to the abstraction function h, and p be a propositional formula that respects h. Then

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

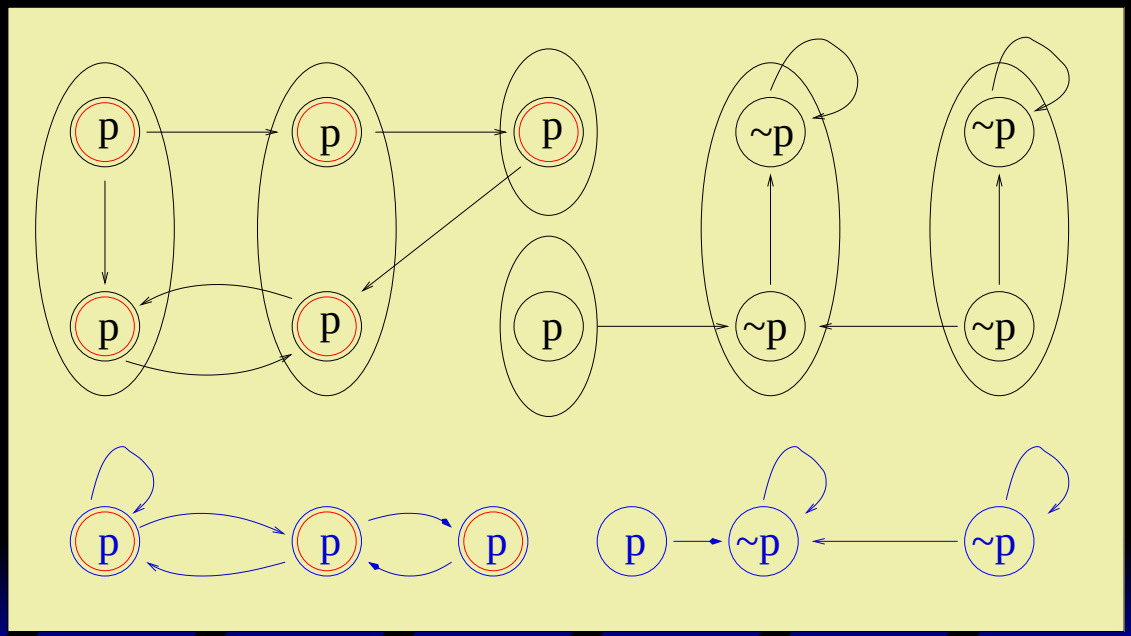


Refinement



Refinement

- Spurious transition because *deadend* states and *bad* states lie in the same abstract state.
- Refinement : Put deadend and bad states in separate abstract states.



Refinement Using SAT Conflict Analysis

- The SAT formula

$$\psi_m = \{\langle s_1 \dots s_m \rangle \mid I(s_1) \wedge \bigwedge_{i=1}^{m-1} R(s_i, s_{i+1}) \wedge \bigwedge_{i=1}^{m} h(s_i) = \hat{s}_i\}$$

  describes the set of paths corresponding to the abstract counterexample

- We solve ψ_m with a SAT solver
- For a spurious counterexample, ψ_m is unsatisfi able

Refinement Using SAT Conflict Analysis

SAT solvers record the important reasons for the unsatisfi ability during the SAT check by Boolean constraint propagation and implication graphs.

We proposed[FMCAD'02] two methods to identify important variables by analysing conflicts generated during the SAT check

- Heuristically score the variables during the SAT check
- Identify important variables by *conflict dependency graphs*

Experimental Results for Refinement by Conflict Analysis

Apart from the smaller IU circuits shown earlier, these techniques are able to handle large D series circuits and an IU circuit with 5000 latches.

circuit	# regs	ctrex length	CSMV time	Heuristic Score			Dependency		
				time	iters	# regs	time	iters	# regs
D2	105	15	152	105	10	51	79	11	39
D5	350	32	1,192	29	3	16	38.2	8	10
D6	177	20	45,596	784	24	121	833	48	90
D18	745	28	>4 hrs	12,086	69	346	9,995	142	253
D20	562	14	>7 hrs	1,493	56	281	1,947	74	265
D24	270	10	7,850	14	1	6	8	1	4
IU-p1	4855	true	-	9,138	22	107	3,350*	13	19
IU-p2	4855	true	-	2,820	7	36	712	6	13

Ongoing and Future Work

- Use symbolic simulation with SAT based reparameterization to simulate abstract counterexamples
- Derive refinement information from the partial simulation
- Extend the method for liveness checking using safety checking as in Biere et al. STTT'03

Expected Thesis Contributions

- SAT based enumeration algorithms for image computation
- Powerful symbolic simulator to simulate deep large circuits with thousands of latches
- Completeness of property checking via automated SAT based abstraction-refinement

Timeline

- **Feb’04:**
- **June’04:**
- **Sept’04:**