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

Thesis Oral

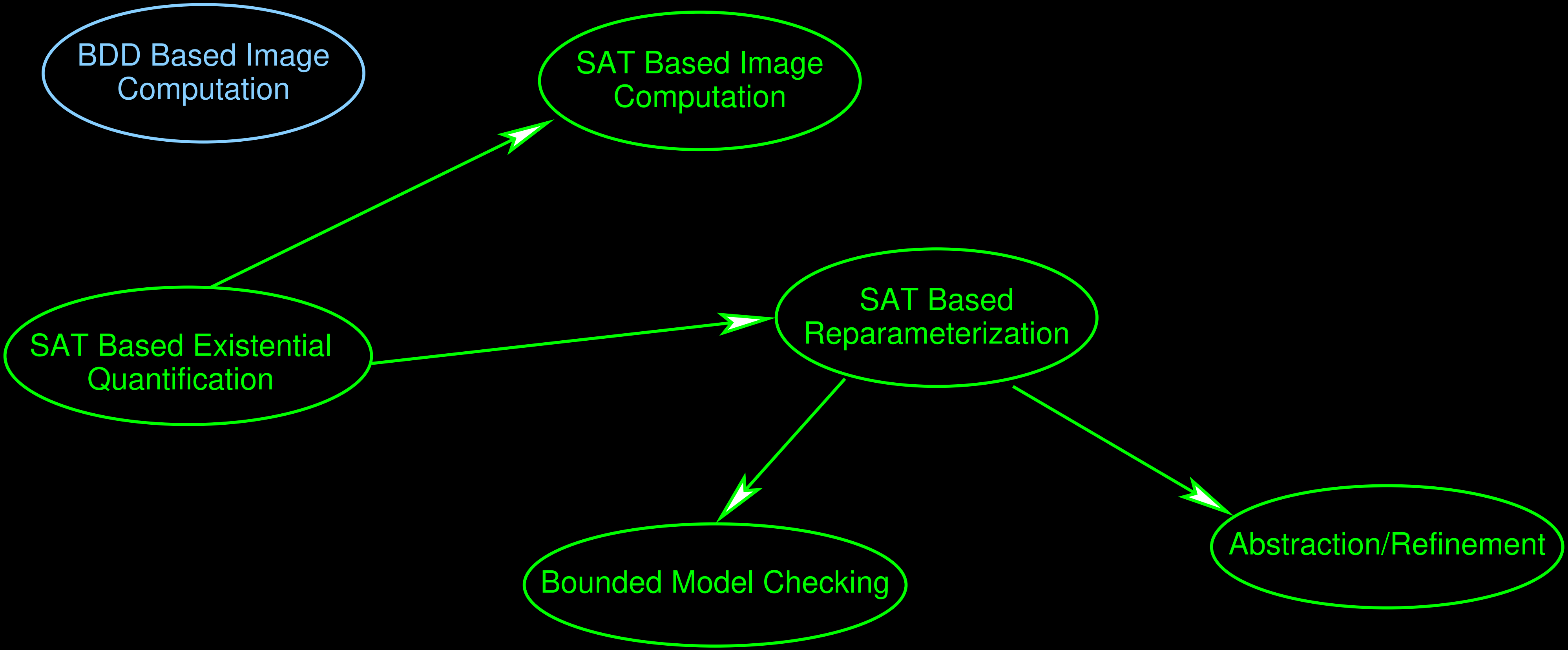
Pankaj Chauhan (pchauhan@cs.cmu.edu)

Computer Science Dept., Carnegie Mellon University

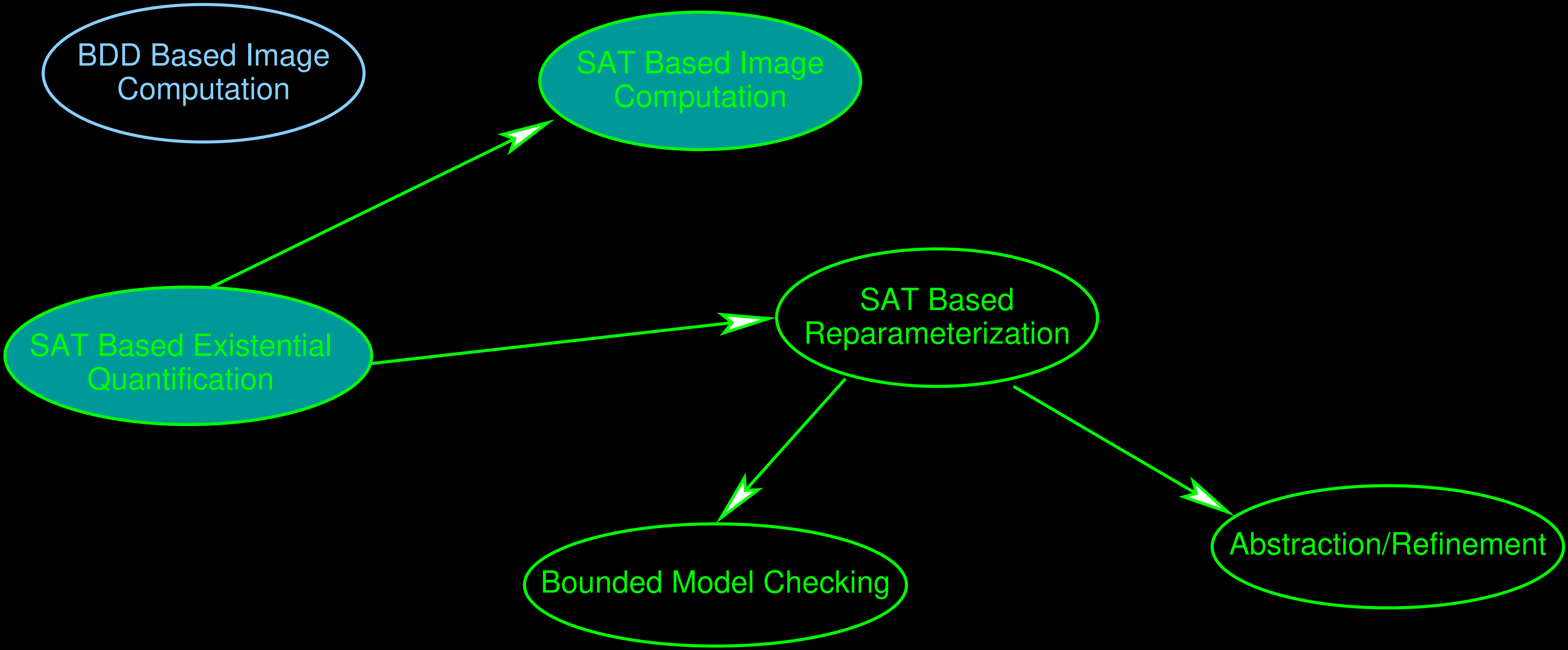
Challenges for a Symbolic Model Checker

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

Scope of the Research


BDD Based Image Computation
SAT Based Image Computation
SAT Based Existential Quantification
SAT Based Reparameterization
Bounded Model Checking
Abstraction/Refinement

SAT Based Reachability



BDD Based Image Computation

SAT Based Image Computation

SAT Based Existential Quantification

SAT Based Reparameterization

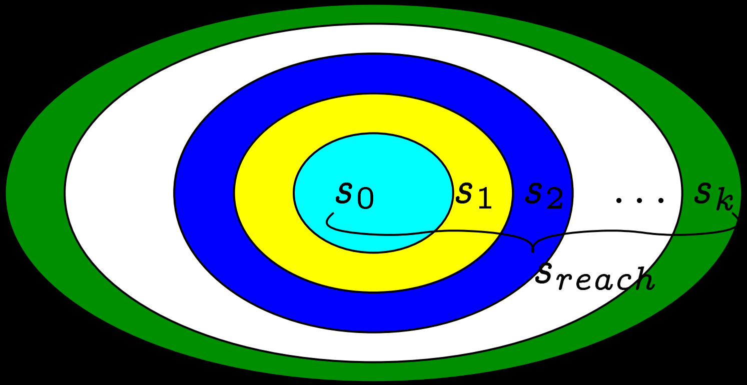
Bounded Model Checking

Abstraction/Refinement

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 $S_{reach}$
```



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 the next state (x') variables to

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

- 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 to S_i and in the SAT as *blocking clause*

Problems

- Time complexity
  - Too 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 ⇒ **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

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

- 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

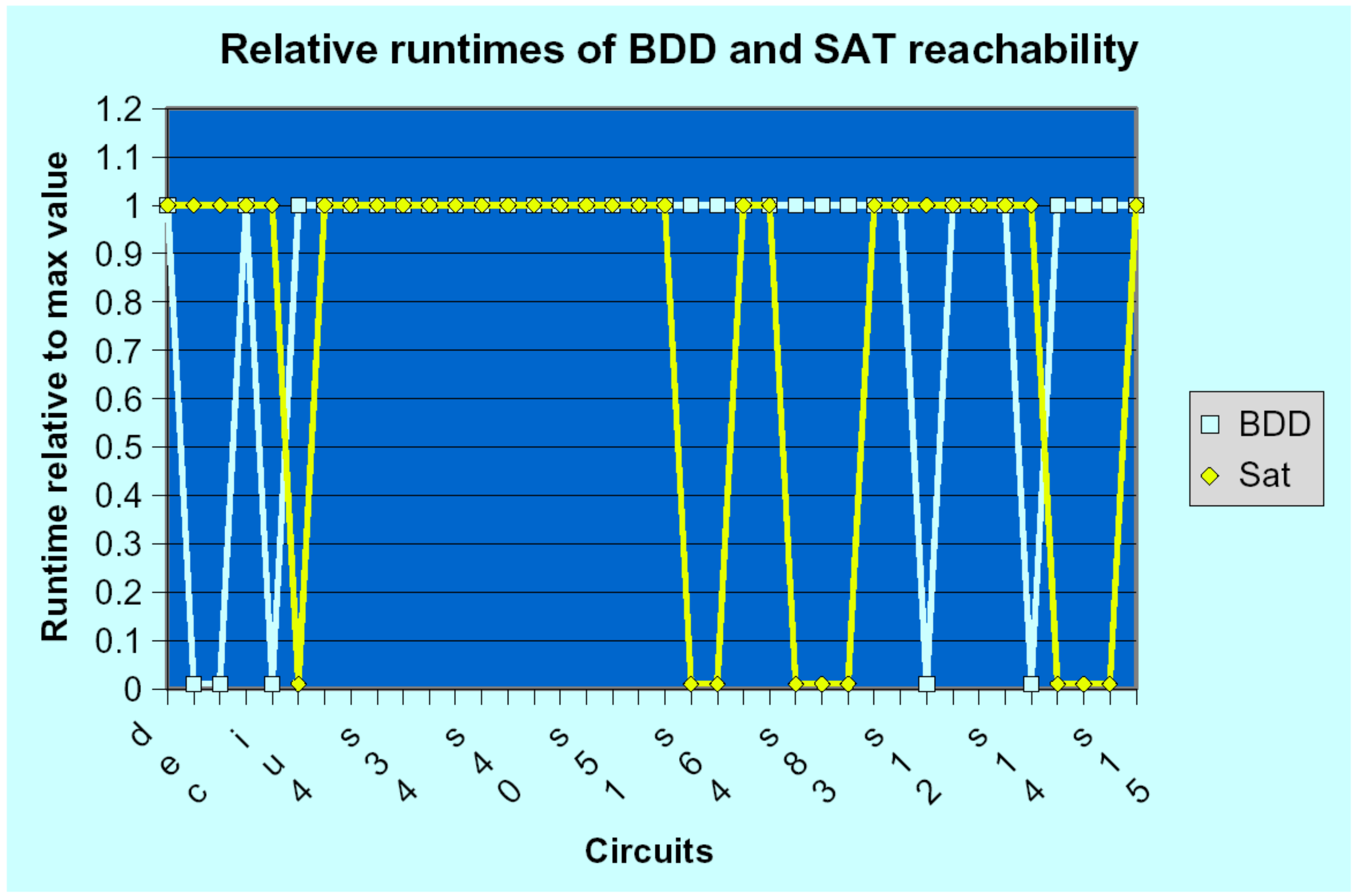
Solving Space Problem

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

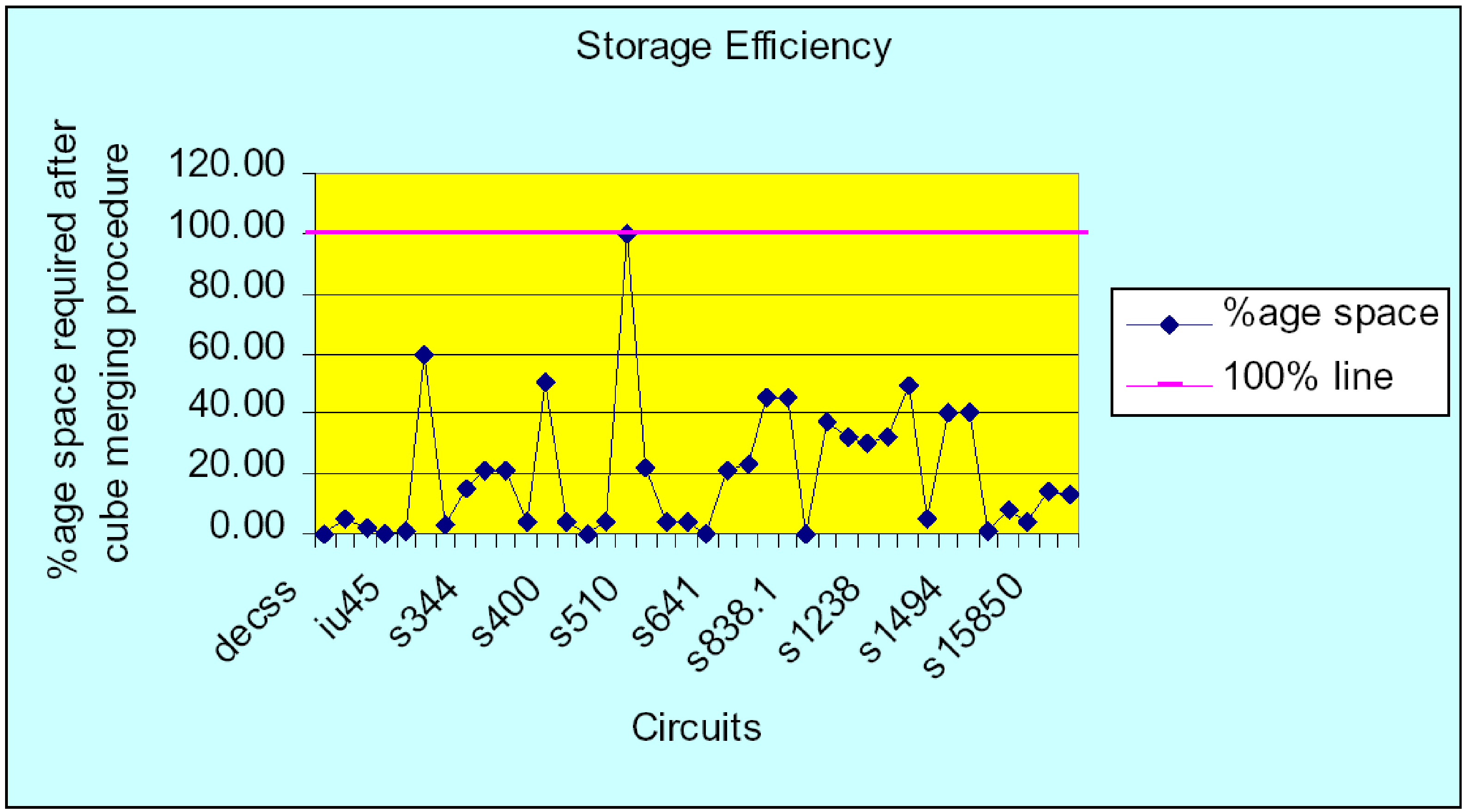
- For each added clause, use a hash table to find clauses with the same set of variables
- A smaller hash to find clauses differing only in one literal.

Can use zDDs or other data structure.

Relative Runtime v/s BDD



Space Savings



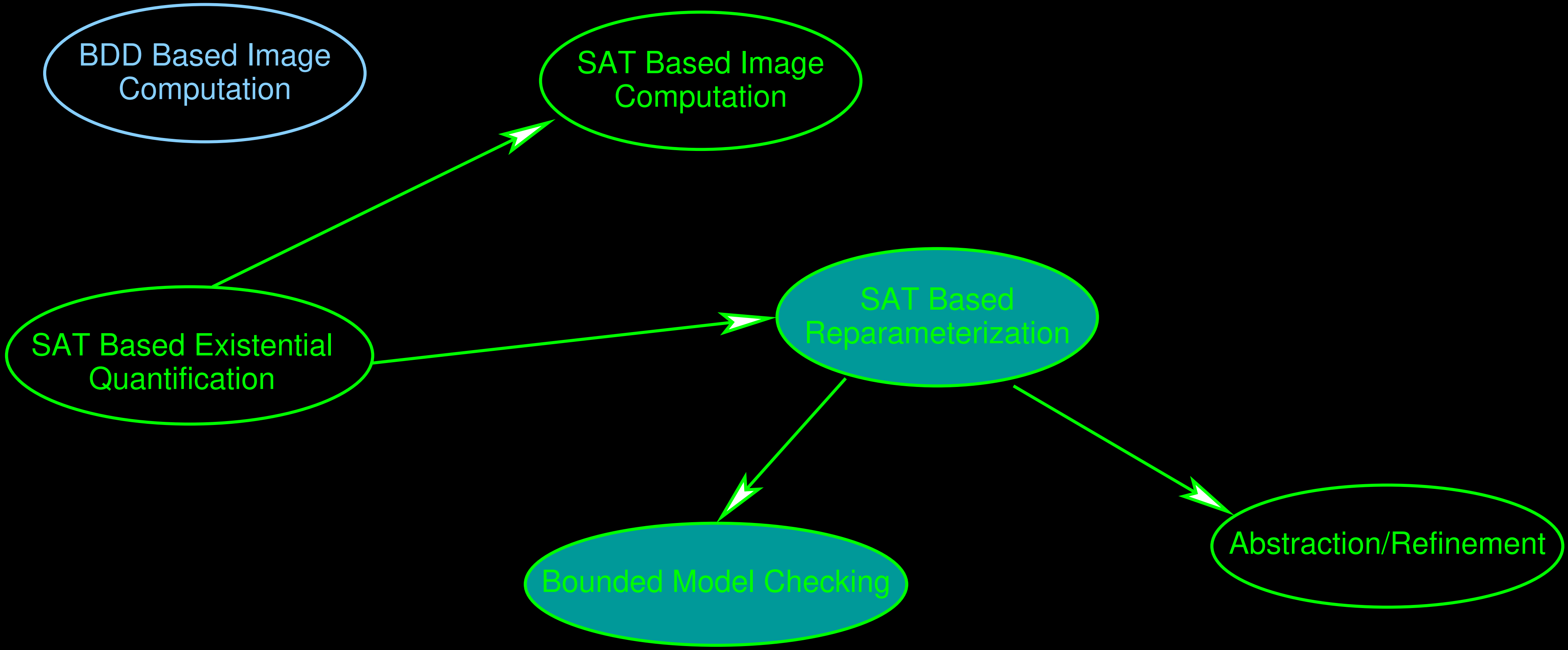
SAT Enumeration Summary

- 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.
- Related work: Satori (UCSB), McMillan (CAV'02)

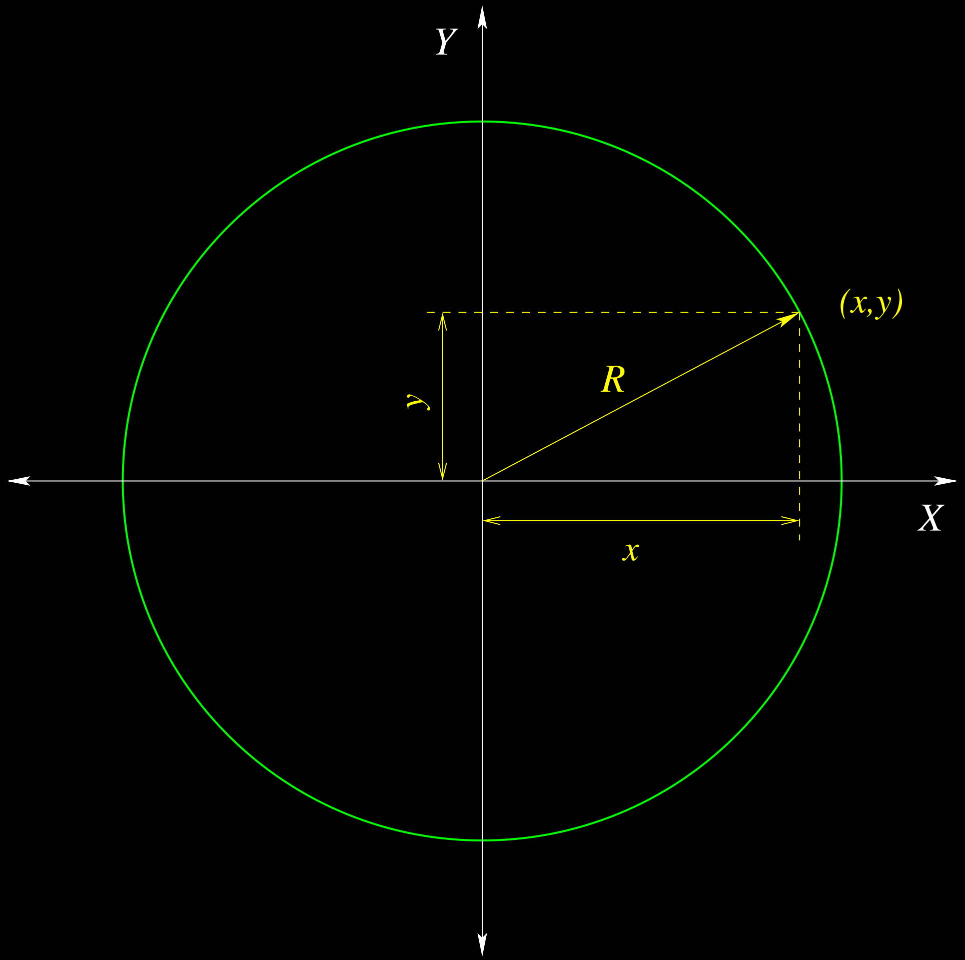
Future Work for SAT Based Reachability

- Space efficient SAT enumeration
- Use of Boolean minimizers like Espresso
- Non-clausal representation

Reparameterization

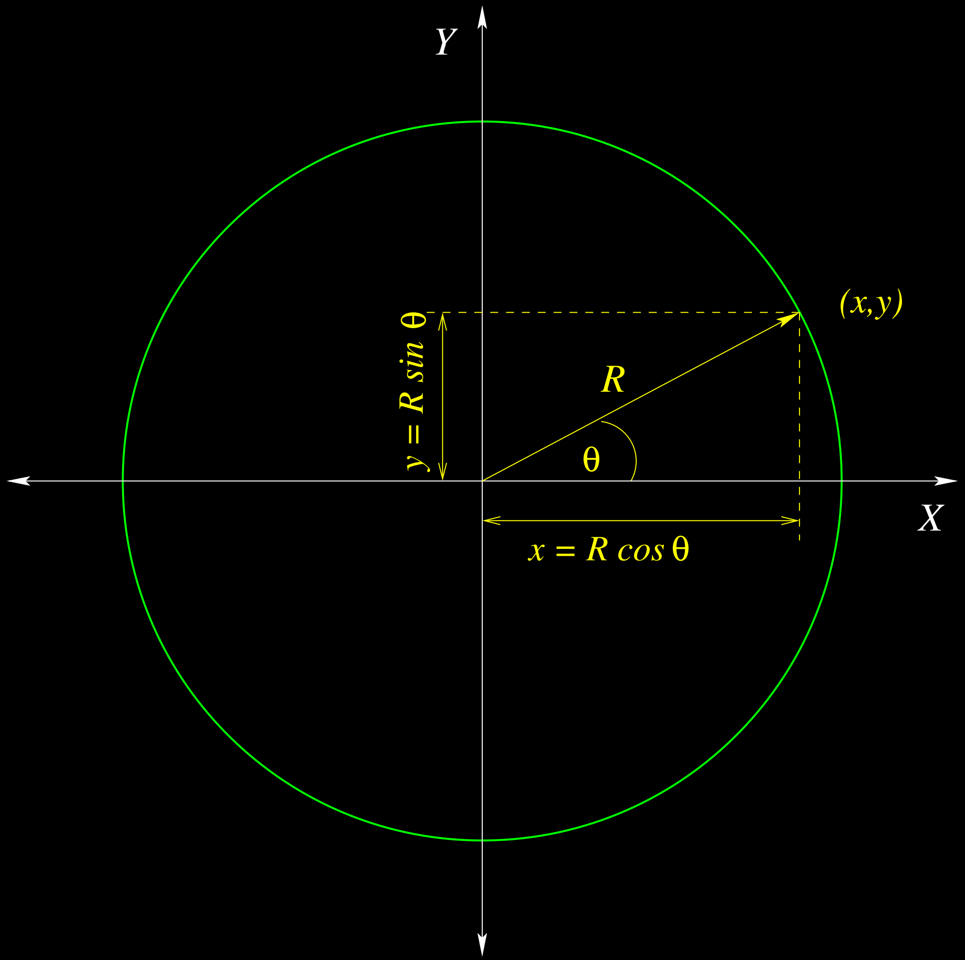

BDD Based Image Computation
SAT Based Image Computation
SAT Based Existential Quantification
SAT Based Reparameterization
Bounded Model Checking
Abstraction/Refinement

Parametric Representation: An Example in R^2



$$x^2 + y^2 = R^2$$

Parametric Representation: An Example in R^2


Y
X
(x,y)
R
θ
y = R sin θ
x = R cos θ

$$x^2 + y^2 = R^2$$

$$\left.\begin{aligned} x &= R \cdot \cos\theta \\ y &= R \cdot \sin\theta \end{aligned}\right\} 0 \leq \theta \leq 2\pi$$

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]

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

p_1	f_1	f_2
0	0	1
1	1	0

Or

$$\begin{aligned} h_1(i_1, i_2) &= i_1 \vee i_2 \\ h_2(i_1, i_2) &= \neg i_1 \wedge \neg i_2 \end{aligned}$$

i_1	i_2	h_1	h_2
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Parametric Representation

$S(V)$: Set of states
$F(P) = (f_1(P), \ldots, f_n(P))$: parametric functions
$P = (p_1, p_2, \ldots, p_k)$: parameters

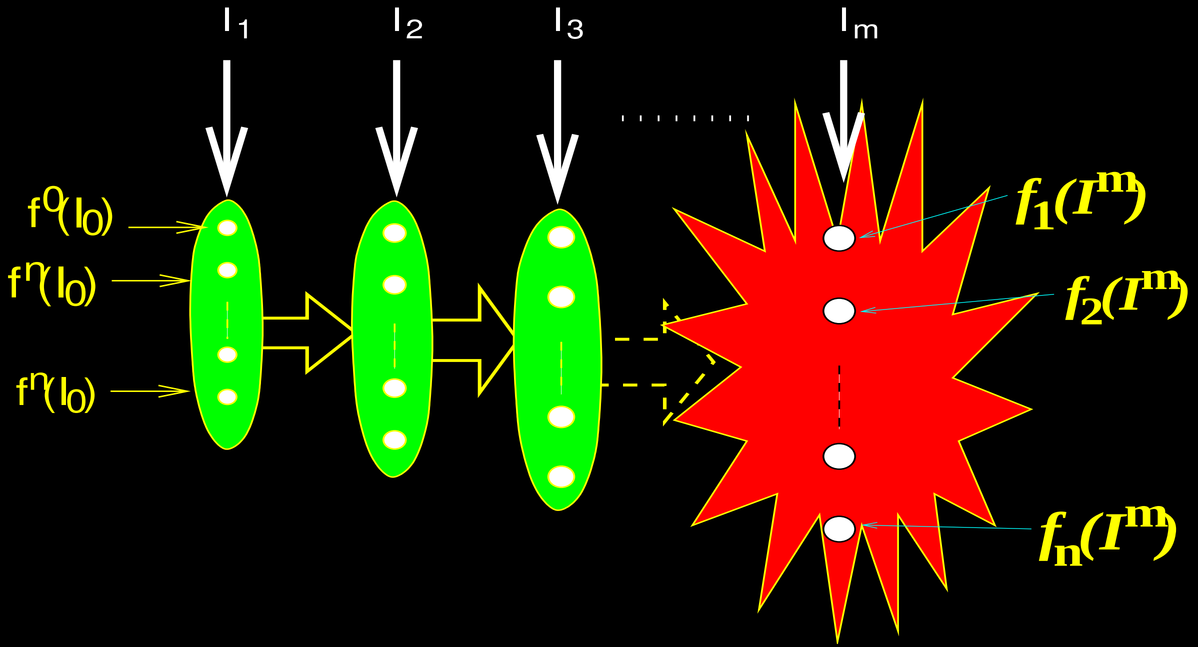
$$S(V) = \left\{ V \middle| \exists P. \begin{pmatrix} v_1 = f_1(P) & \wedge \\ v_2 = f_2(P) & \wedge \\ \vdots & \wedge \\ v_n = f_n(P) & \end{pmatrix} \right\}$$

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



I_1
I_2
I_3
I_m
$f^0(I_0)$
$f^{\eta}(I_0)$
$f^{\eta}(I_0)$
$f_1(I^m)$
$f_2(I^m)$
$f_n(I^m)$

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

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

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 can be 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 can be 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 first i parameters only.

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 first i parameters only.
- We think of each new parameter p_i as free variable for the i^{th} state bit v_i.

A Known Decomposition [GB'03]

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

where,

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

A Known Decomposition [GB'03]

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

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 [GB'03]

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

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 [GB'03]

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

```
$\textsc{OrderedReparam}(\bar{f}(I^m) = (f_1(I^m), \ldots, f_n(I^m))$
$\rho \leftarrow 1$
for $i = 1$ to $n$
        $h_i^1 \leftarrow \forall I^m.(\rho \rightarrow f_i = 1)$
        $h_i^0 \leftarrow \forall I^m.(\rho \rightarrow f_i = 0)$
        $h_i^c \leftarrow \neg(h_i^1 \vee h_i^0)$
        $h_i \leftarrow h_i^1 \vee (p_i \wedge h_i^c)$
        $\rho \leftarrow \rho \wedge (h_i = f_i)$
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

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

Incremental SAT is the key to performance!

Correctness of the Algorithm

- Non-trivial to see that the algorithm is correct.
- The algorithm was rigorously proved, detailed proof in the thesis.
- Proof was also done in PVS in about 8 days.

Experimental Results (Time)

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.

Experimental Results (Memory)

ckt	# regs	# PIs	bug	Bug time		BMC max		sym max		
			len.	BMC	sym	len	time	len.	time	# rest.
D2^{+}	94	11	15	18	32	64	8084^{M}	4336	21600^{T}	163
D5^{+}	343	7	32	15	17	45	3594^{M}	2793	21600^{T}	338
D24	223	47	10	5	7	913	13293^{M}	10298	21600^{T}	152
D6	161	16	20	289	145	48	6094^{M}	1521	21600^{T}	93
D19	285	49	32	6834	1698	23	13721^{M}	399	21600^{T}	144
D20	532	30	14	2349	574	36	3984^{M}	1856	21600^{T}	185
M3	334	155	true	-	-	68	7039^{M}	781	21600^{T}	22
M4	744	95	true	-	-	26	12695^{M}	302	21600^{T}	38
M5	316	104	true	-	-	41	7492^{M}	518	21600^{T}	45
IUp1	4494	361	true	-	-	39	2870^{M}	1278	21600^{T}	902
IUp2	4494	361	true	-	-	39	3192^{M}	1103	21600^{T}	1242
IUp3	4494	361	true	-	-	39	2994^{M}	1284	21600^{T}	856

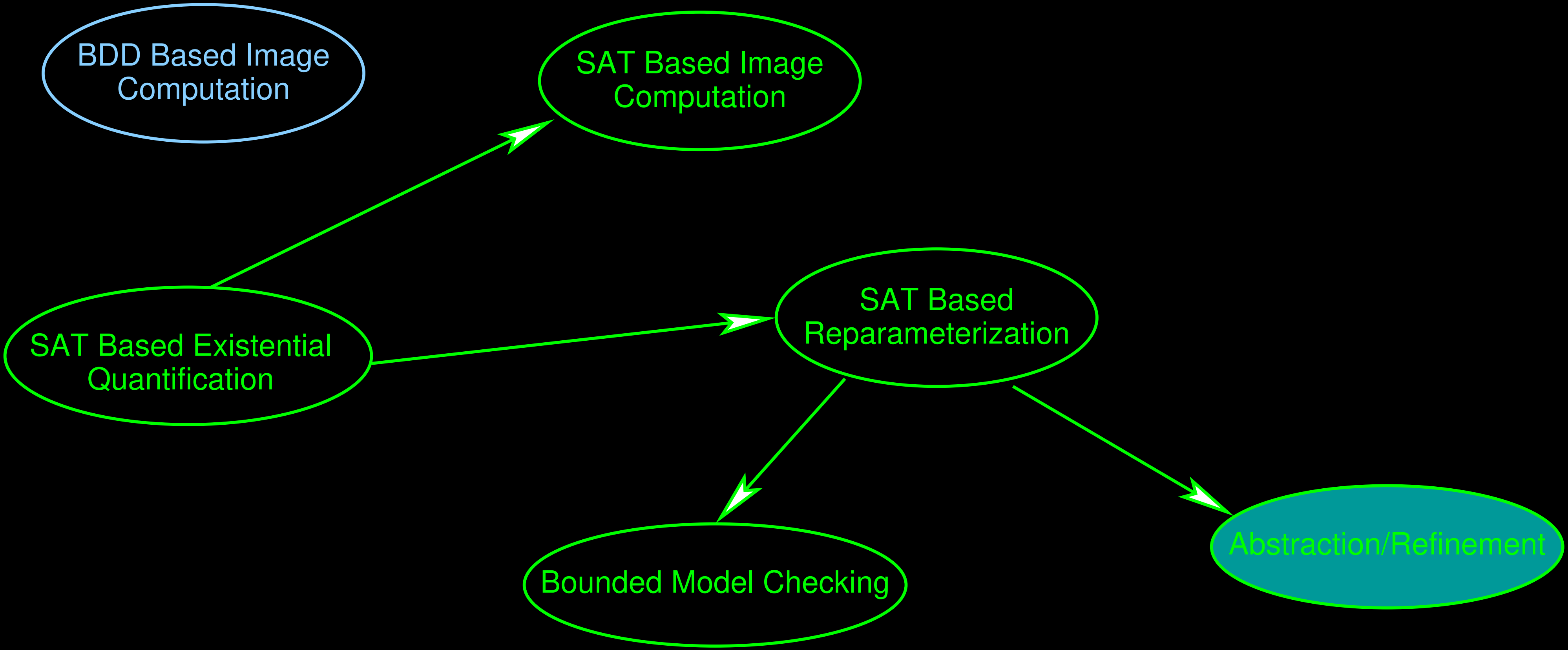
Extensions

- √ Counterexample generation
- √ Handling of general transition constraints, SMV style `INVAR`, `TRANS`
  - Need to check for empty set of states
- √ Fixedpoints
  - By adding self-loops to states
  - By using a parameterized set union algorithm

Future Work for Reparameterization

- Effect of variable orders
- Better circuit representation and BMC like optimizations
- Parametric Abstraction
- Proving properties true?

Abstraction/Refinement



BDD Based Image Computation
SAT Based Image Computation
SAT Based Existential Quantification
SAT Based Reparameterization
Bounded Model Checking
Abstraction/Refinement

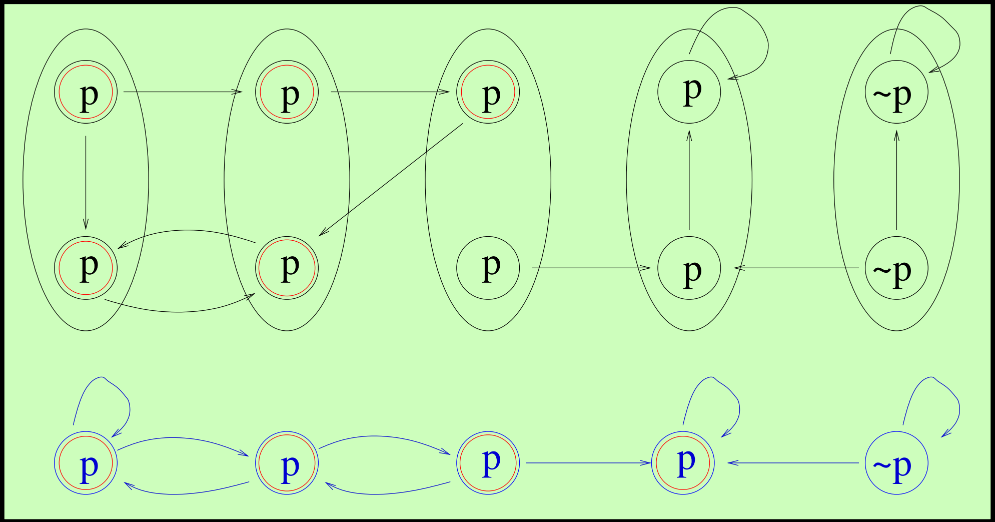
CEGAR Loop

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

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



Approaches to Automated Abstraction/Refinement

- Clarke *et al.*, CAV,02, FMCAD’02: Analyse failed counterexample and use SAT checker for deriving refinement information

Approaches to Automated Abstraction/Refinement

- Clarke *et al.*, CAV,02, FMCAD'02: Analyse failed counterexample and use SAT checker for deriving refinement information
- McMillan, Amla TACAS'03: Use BMC upto the length of abstract counterexample, get unsatisfiability proof from SAT

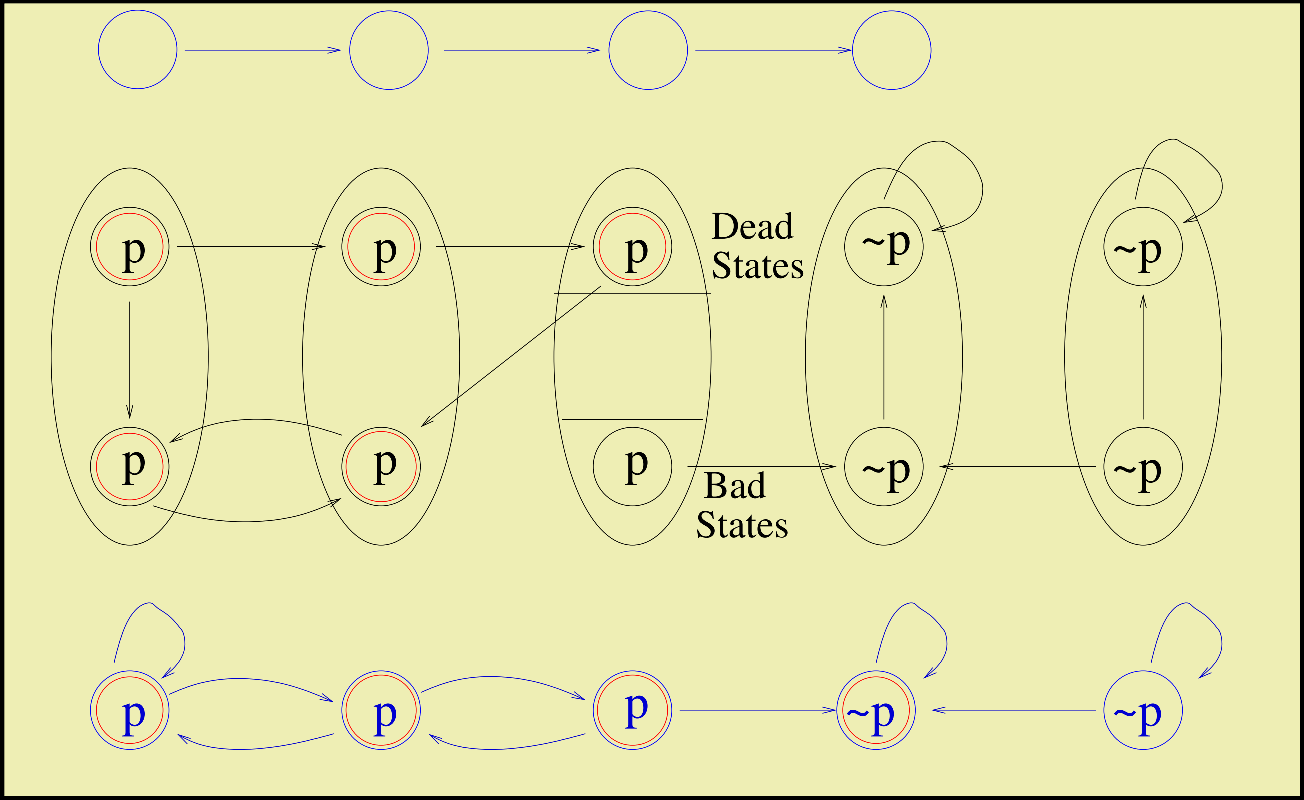
Approaches to Automated Abstraction/Refinement

- Clarke *et al.*, CAV,02, FMCAD'02: Analyse failed counterexample and use SAT checker for deriving refinement information
- McMillan, Amla TACAS'03: Use BMC upto the length of abstract counterexample, get unsatisfiability proof from SAT
- McMillan, CAV'03: Failed BMC instance provides interpolation proofs that are used to simplify fixed-point computation

BMC is the Key

- All abstraction/refinement approaches need to simulate the large concrete state machine.
- Refinement is derived from analysis of some failed SAT formula.
- Abstract counterexamples can be very long.

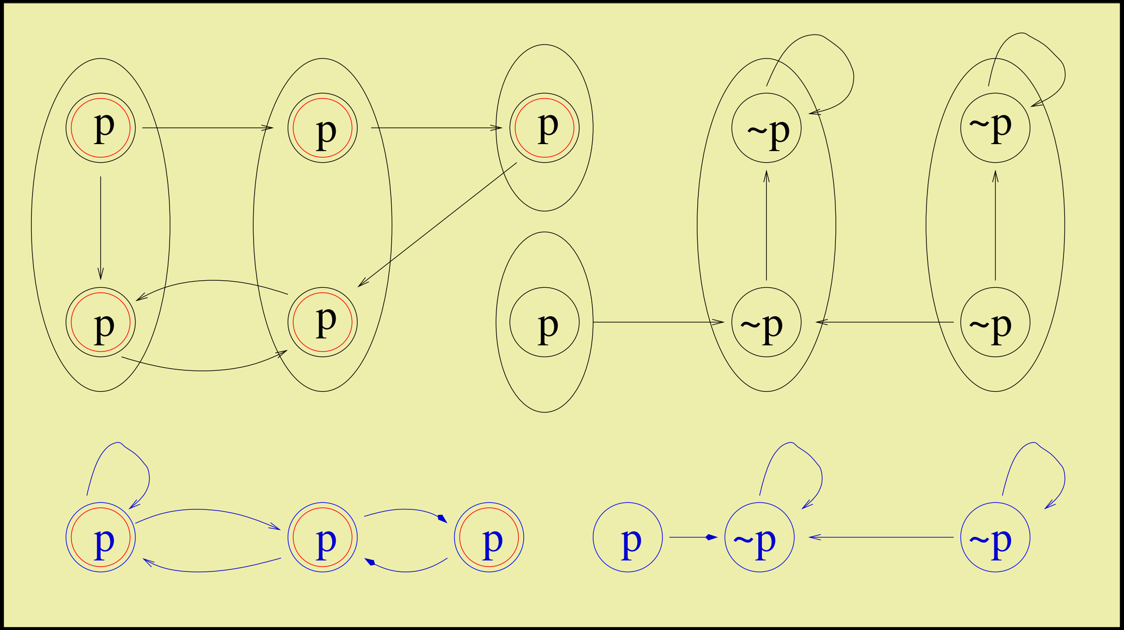
Refinement [Clarke *et al.*]


p
p
p
Dead
States
~p
~p
p
p
p
Bad
States
~p
~p
p
p
p
~p
~p

Spurious transition because *deadend* states and *bad* states lie in the same abstract state.

Refinement

Put deadend and bad states in separate abstract states.


p
p
p
~p
~p
p
p
p
~p
~p
p
p
p
p
~p
~p

Refinement Using SAT Conflict Analysis

- The SAT formula

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

  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 unsatisfiable

Refinement Using SAT Conflict Analysis

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

Refinement Using SAT Conflict Analysis

We proposed two methods [FMCAD'02] 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*

Afterwards, the concept of *unsatisfiable core* was introduced.

Reparameterization in Abstraction-Refinement

- Reparameterization is used for simulating abstract counterexample(s) on the original machine.

Reparameterization in Abstraction-Refinement

- Reparameterization is used for simulating abstract counterexample(s) on the original machine.
- Once reparameterized, we lose all the original circuit variables upto that point in the trace.

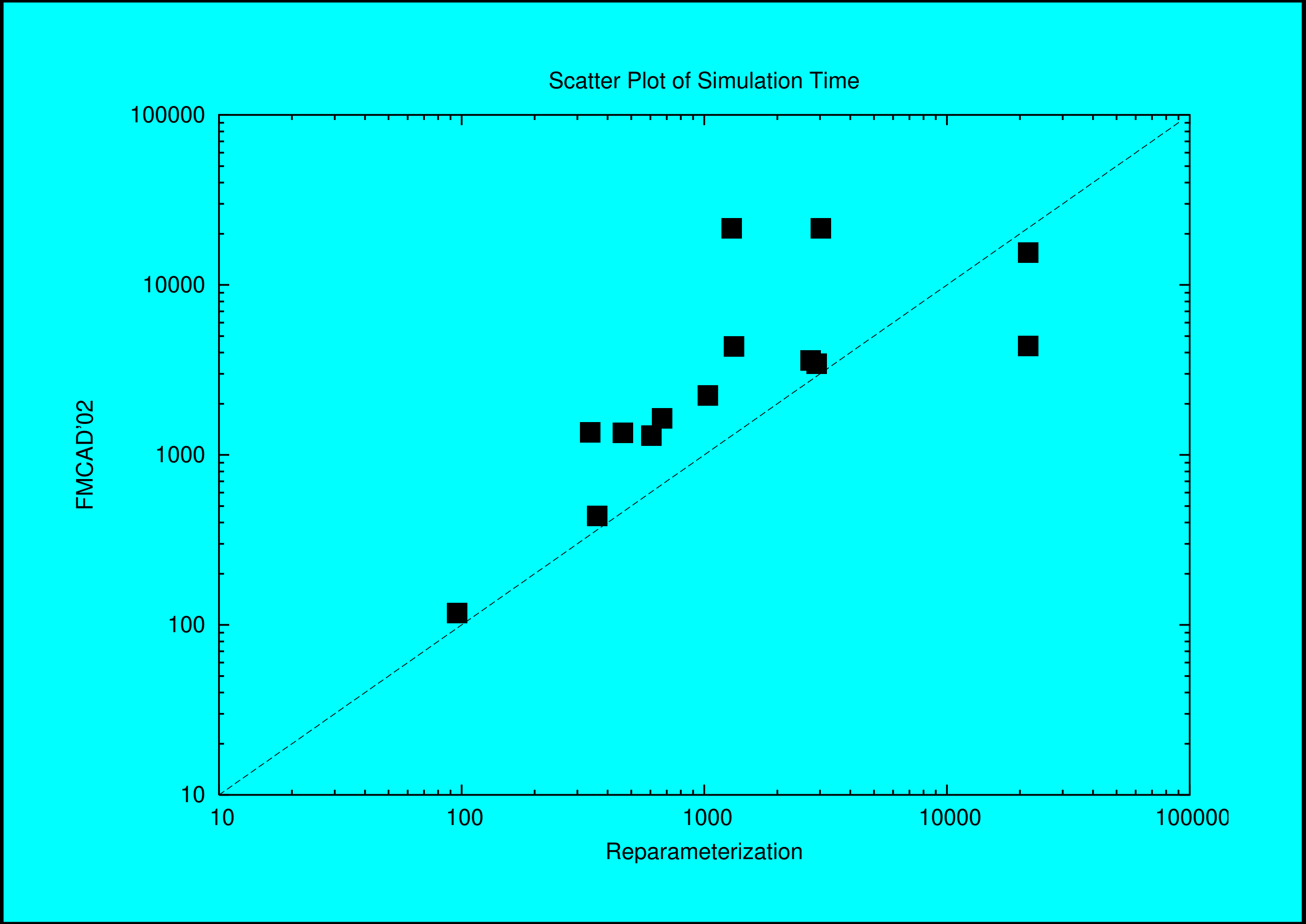
Reparameterization in Abstraction-Refinement

- Reparameterization is used for simulating abstract counterexample(s) on the original machine.
- Once reparameterized, we lose all the original circuit variables upto that point in the trace.
- Due to this, the refinement algorithm only reasons about partial trace, from the point where the last reparameterization was done.

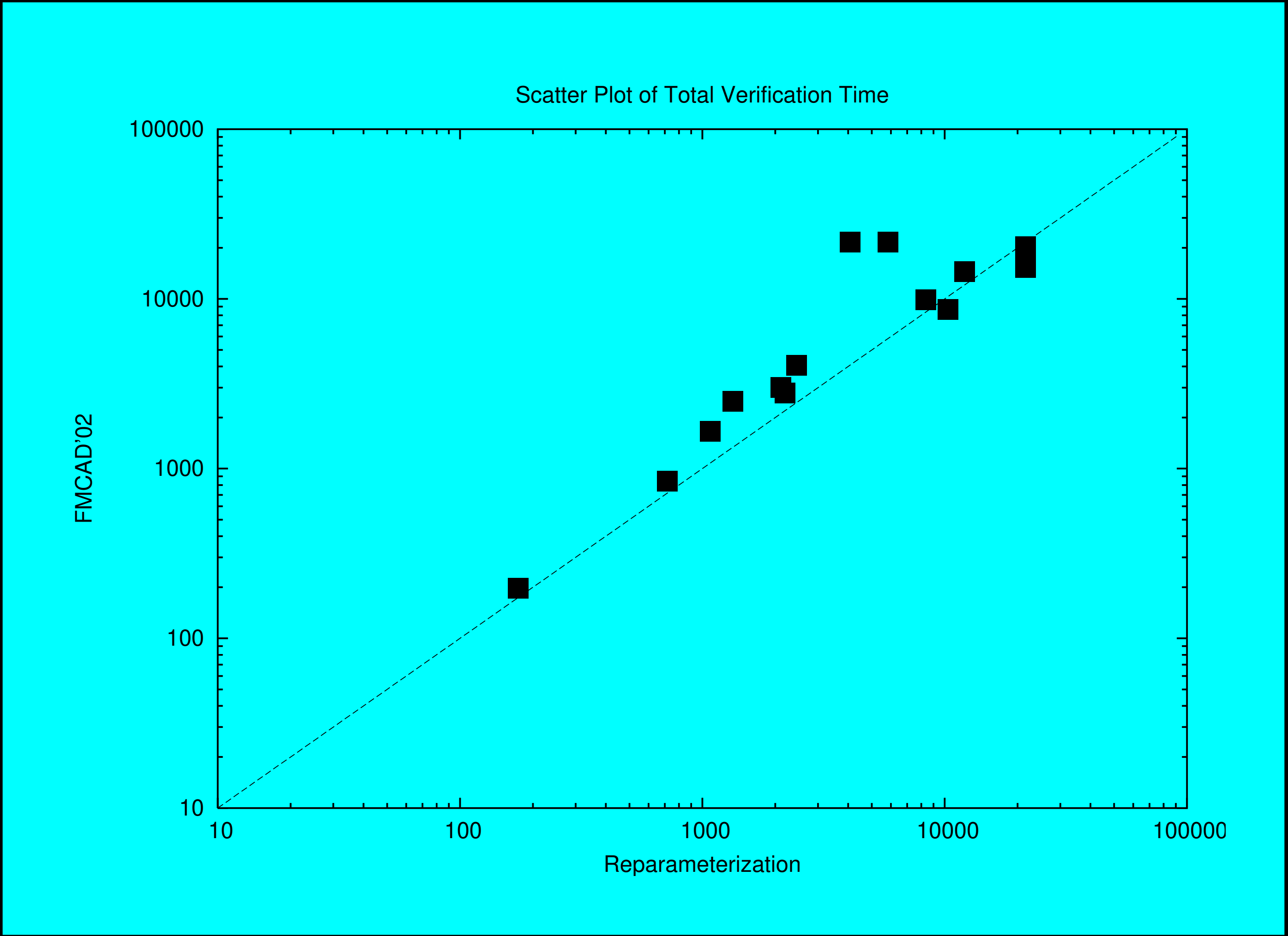
Reparameterization in Abstraction-Refinement

- Reparameterization is used for simulating abstract counterexample(s) on the original machine.
- Once reparameterized, we lose all the original circuit variables upto that point in the trace.
- Due to this, the refinement algorithm only reasons about partial trace, from the point where the last reparameterization was done.
- Quality of refinement may be worse than plain simulation, but on the other hand, we can simulate much deeper, a tradeoff.

Experimental Results for Abstraction Refinement



Experimental Results for Abstraction Refinement



Future Work

- Extend the method for liveness checking using safety checking, e.g., Biere et al. STTT'03
- Better refinement strategies for partial simulations.

Summary

- 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
- BDD based image computation is part of the thesis, but is not covered in the talk

Questions?

Circuit Characteristics

circuit	# latches	# inputs	counterexample length
D6	161	16	20
D18	498	247	28
D19	285	49	32
D20	532	30	14
M3	334	155	true
M4	744	95	true
M5	316	104	true
IUp1	4494	361	true
IUp2	4494	361	true
IUp3	4494	361	true
s3271	116	26	true
s13207	669	31	true
s15850	597	14	true
s38417	1636	28	true

Experimental Results

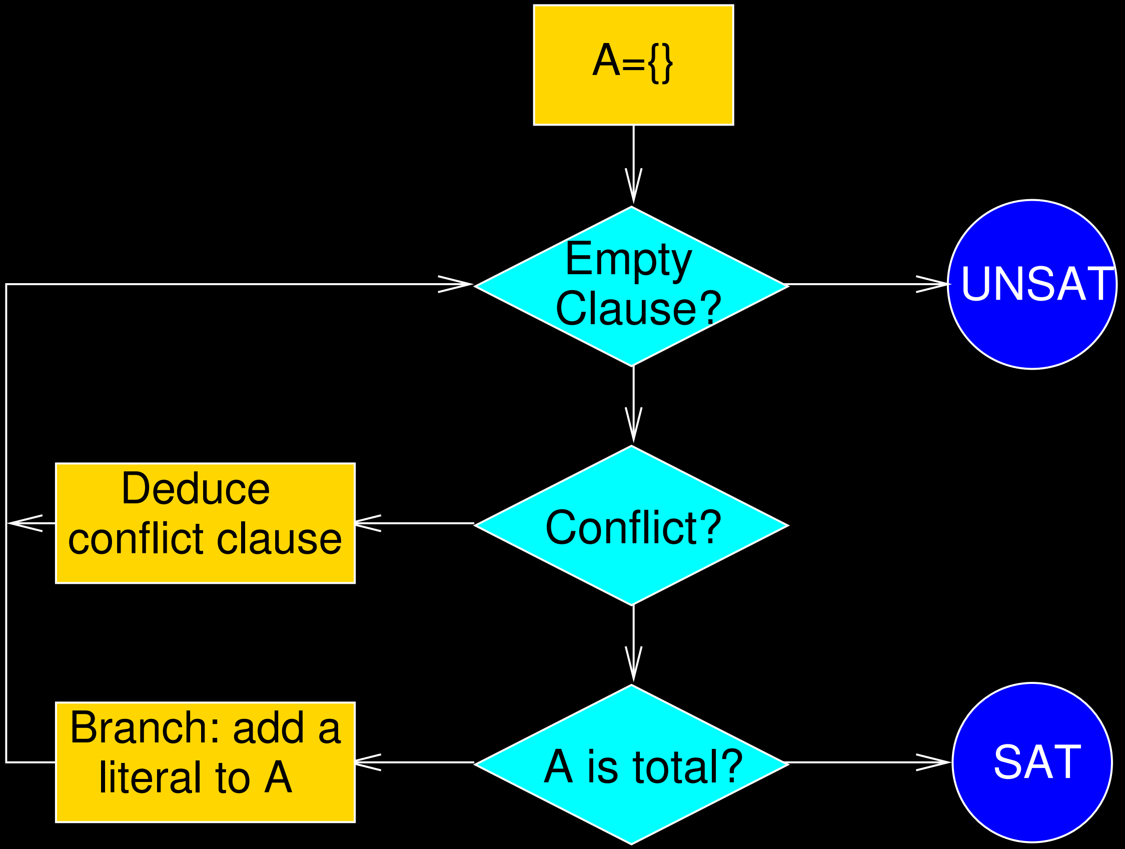
ckt	# refn		|reg|		max |CE|		sim. time		total time		# rep
	fmcad	sym	fmcad	sym	fmcad	sym	fmcad	sym	fmcad	sym	
D6	48	48	39	39	20	20	438	362	845	718	23
D18	142	127	253	253	28	28	3598	2740	9873	8349	56
D19	37	49	103	112	32	32	4348	1329	14528	12087	95
D20	74	74	265	265	14	14	1359	338	2794	2192	23
M3	58	**42†**	128	**87†**	54	**54†**	4378	**2088†**	15306	>**21600†**	3
M4	173	**94†**	336	**184†**	44	**39†**	15540	**4776†**	20327	>**21600†**	21
M5	7	11	30	30	6	10	3427	2902	8653	10312	3
IUp1	**8‡**	13	**12‡**	19	**72‡**	72	**3390‡**	1295	**4877‡**	4063	117
IUp2	6	6	13	13	22	22	1298	605	2498	1335	16
IUp3	**17⋆**	32	**19⋆**	41	**52⋆**	67	> **21600⋆**	3022	> **21600⋆**	5836	325
s3271	32	32	38	38	48	48	117	96	198	174	3
s13207	15	15	23	23	43	43	2231	1035	4066	2454	13
s15850	8	8	18	18	56	36	1643	669	2998	2108	8
s38417	19	19	29	29	53	53	1347	462	1655	1077	14

† denotes that the model checking of abstract model timed out, ‡ denotes that the simulation of counterexample failed due to memory limit, and ⋆ denotes that the simulation of counterexample timed out.

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]

DPLL Style SAT Flowchart


A={}
Empty Clause?
UNSAT
Deduce conflict clause
Conflict?
Branch: add a literal to A
A is total?
SAT