Efficient Decision Ordering Techniques for SAT-based Test Generation

Mingsong Chen, Xiaoke Qin and Prabhat Mishra

Computer and Information Science and Engineering

University of Florida, USA

DATE 10

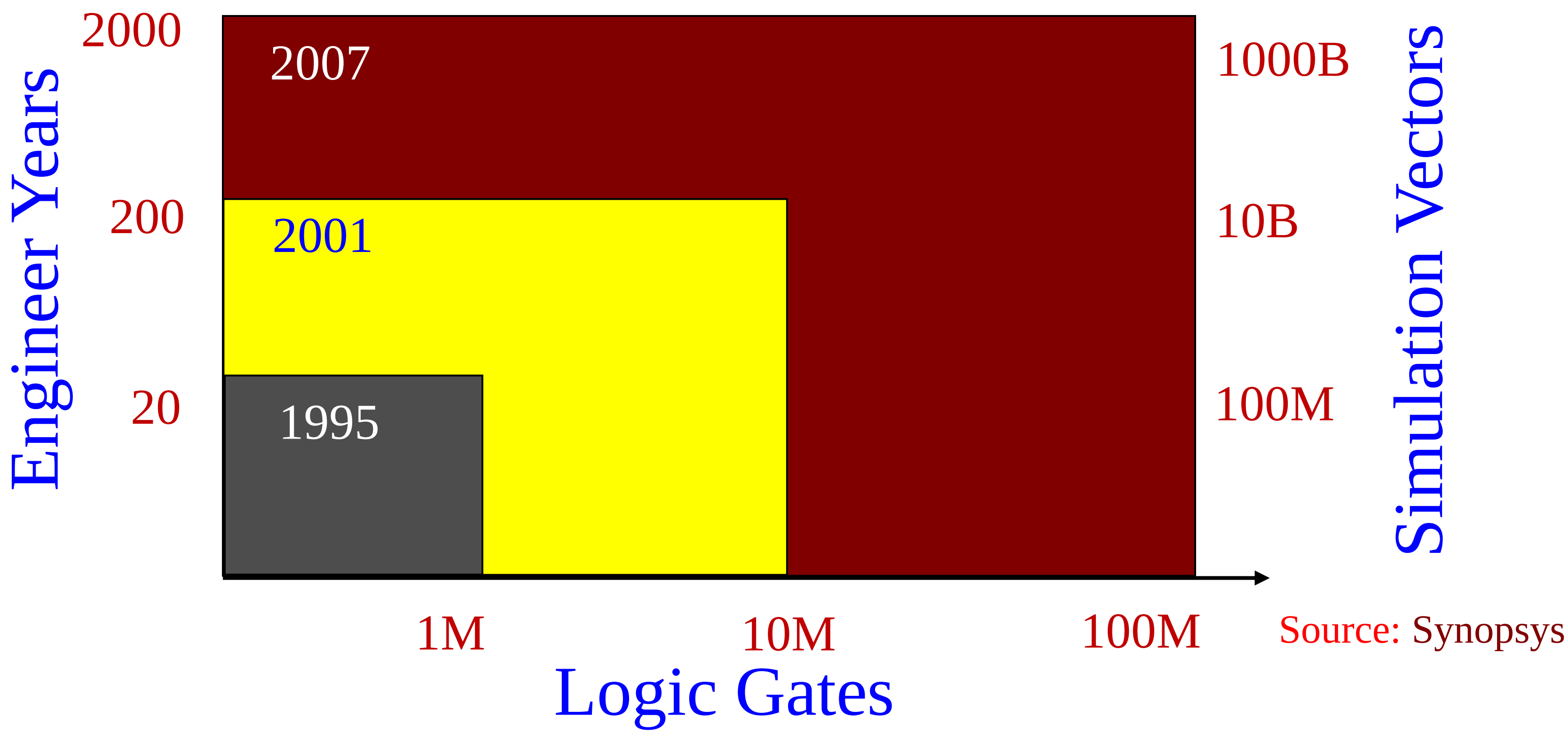
March 10, 2010

UNIVERSITY OF FLORIDA

Outline

- Introduction
- Simulation-based Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Bit value ordering
  - Variable ordering
  - Test generation using our methodology
- Experiments
- Conclusion

Functional Verification of SOC Designs



- Functional validation is a major challenge
  - Majority of the SOC fails due to logic errors
- Simulation using directed tests is promising

Outline

- Introduction
- Simulation-based Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Bit value ordering
  - Variable ordering
  - Test generation using our methodology
- Experiments
- Conclusion

Test Generation using Model Checking

- **Model Checking**
  - **Design is modeled temporal specification, e.g., SMV**
  - **Desired behaviors in temporal logic properties**
  - **Property holds, or fails with a counterexample**

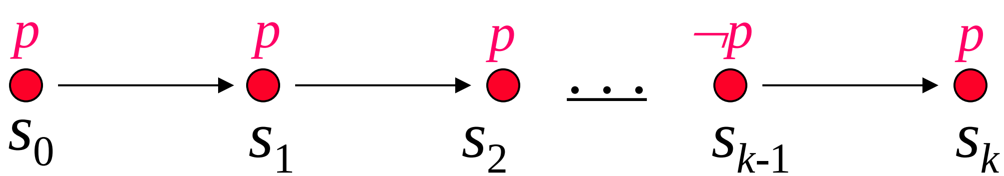
Problem: Test generation is very costly or not possible in many scenarios in the presence of complex SoCs and/or complex properties.

Approach: Exploit learning to reduce complexity
- Reduction of TG time & memory requirements
- Enables test generation in complex scenarios

SAT-based Bound Model Checking

- **For every finite model and a LTL property ϕ there exists *k* such that:** $M \models_k \phi \rightarrow M \models \phi$
- **Test generation needs to consider safety properties**
- **The safety property *P* is valid up to cycle *k* iff Ω(*k*) is not satisfiable.**

$$\Omega(k) = I(S_0) \wedge \bigwedge_{i=0}^{k-1} R(S_i, S_{i+1}) \wedge \bigvee_{i=0}^{k} \neg P(s_i)$$



- **If Ω(*k*) is satisfiable, then we can get an assignment which can be translated to a test.**

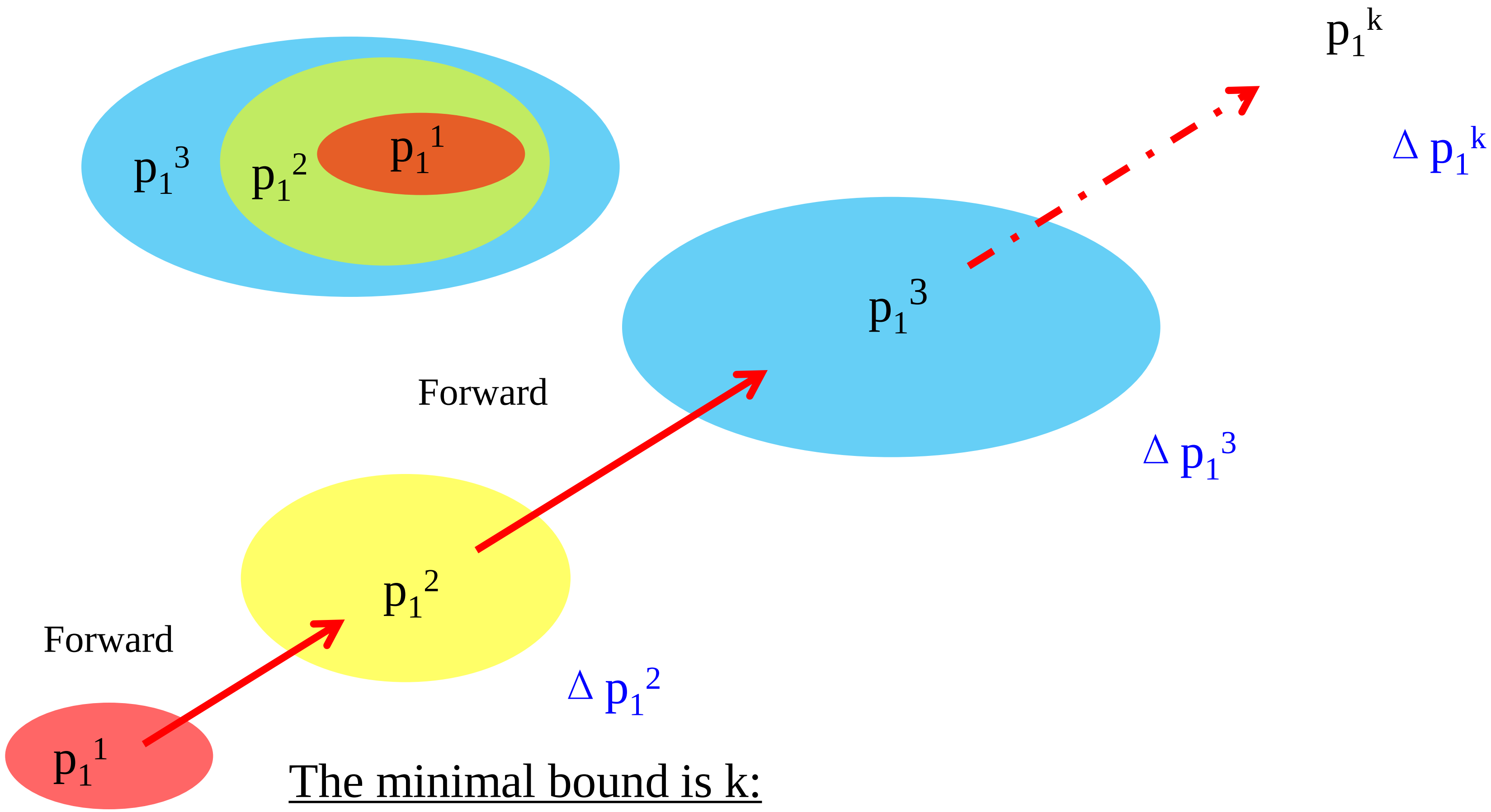
DPLL Algorithm

```
while (1){
    run_periodic_function();
    if( decide_next_branch() ){
        while (Implication == CONFLICT) {
                blevel = Conflict Backtrack ();
            if( blevel<0 )
                return UNSAT;
        }
    } else return SAT;
}
```

BCP = Implication Number + Conflict Backtrack

Boolean Constraint Propagation (BCP) consumes up to 80% of the time and resources during SAT solving

Same Property but Different Bounds

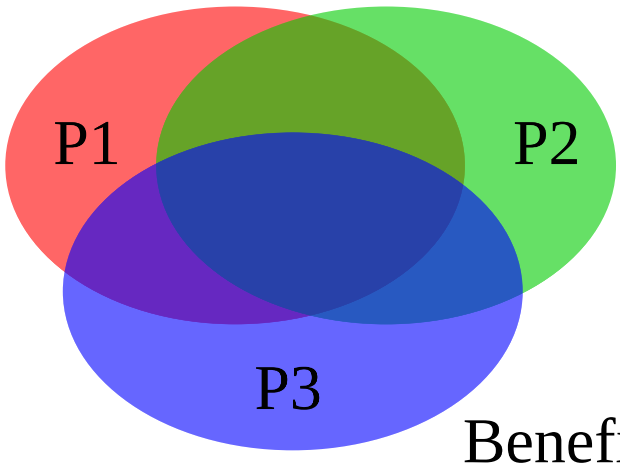


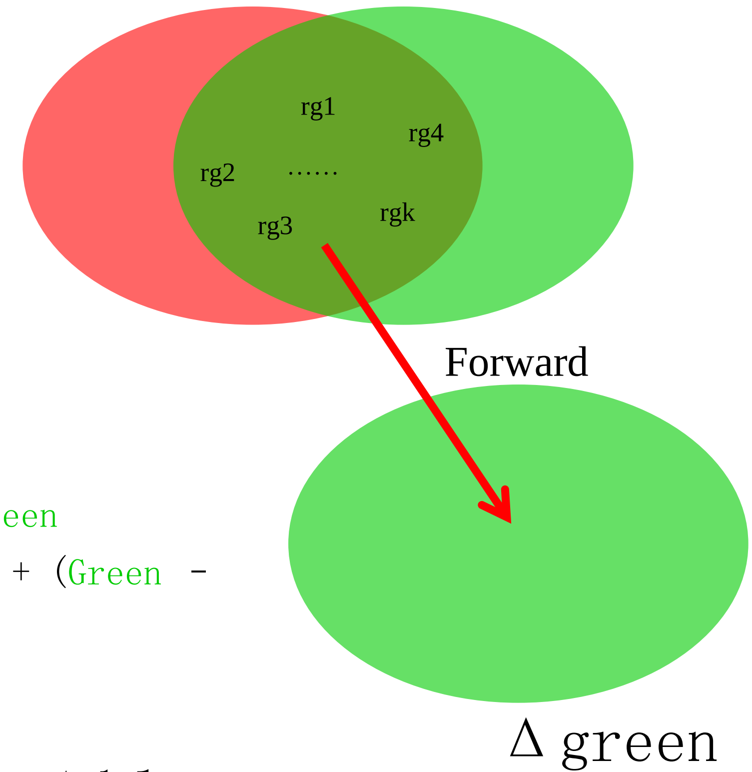
<u>The minimal bound is k:</u>

Save: $\Delta p_1^2 + \Delta p_1^3 + \ldots + \Delta p_1^{k-1} + \ldots + \Delta p_1^k$

O. Strichman. Pruning Techniques for the SAT-Based Bounded Model Checking Problems. ***CHARME*, *2001***

Same Design, Different Properties



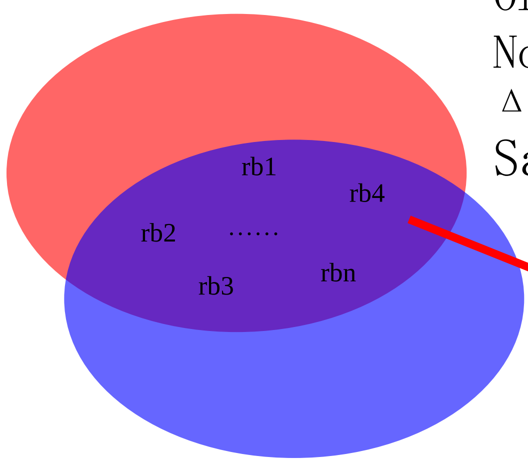


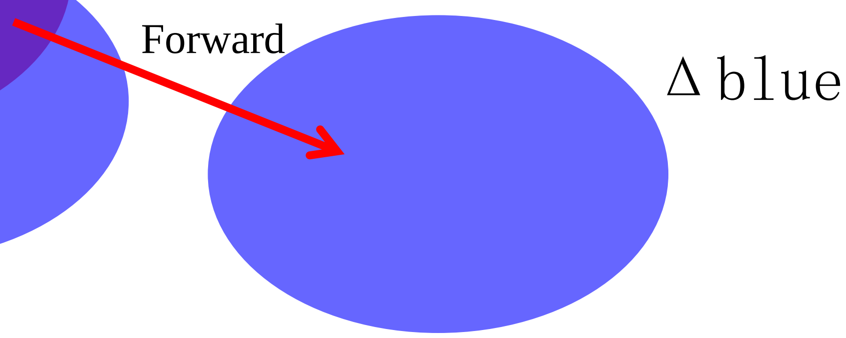
<u>Benefit:</u>

Original: Red + Blue + Green

Now: Red + (Blue - Δblue) + (Green - Δgreen)

Save: Δblue + Δgreen





P. Mishra and M. Chen. Efficient Techniques for Directed Test Generation using Incremental Satisfiabilty. *VLSI Design 2009*

Promising Observations

- Similar properties have the similar counter-examples (variable assignments).
  - Such important information can be reused.
- Current decision ordering techniques focus on the SAT problem instead of the real design.
  - For example, VSDIS, for each literal *lit* has a score
  - Initialization
    - ***score(lit) = literal count of lit in CNF clauses***
  - Periodical update (not include initialization)
    - ***score(lit) = score(lit) /2 + lit_in_conflict(lit)***

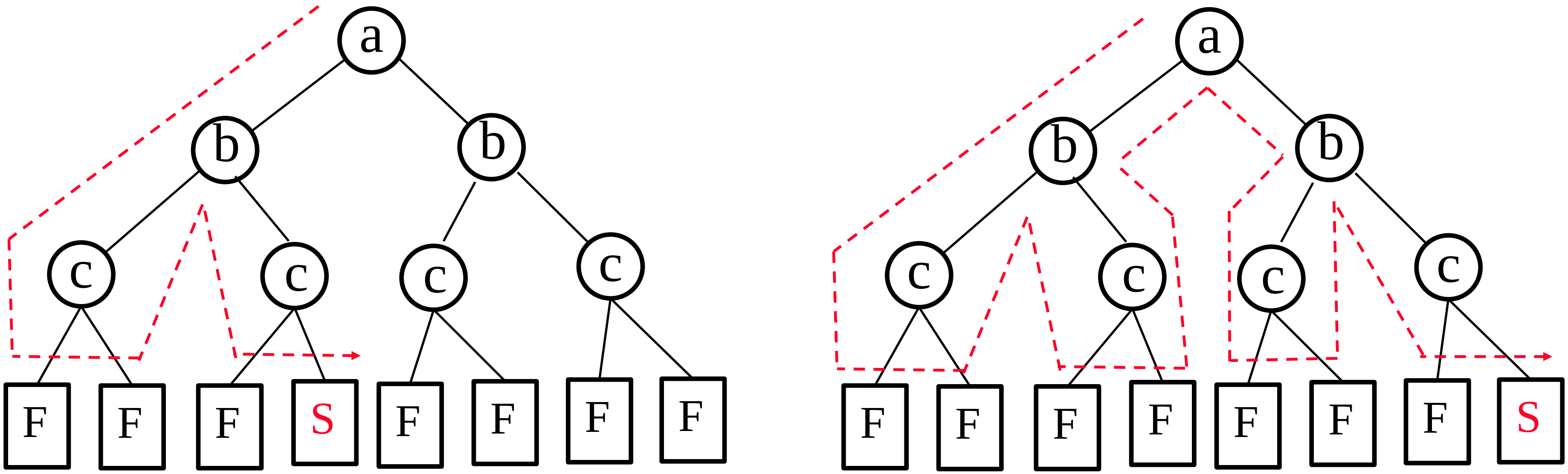
Outline

- Introduction
- Simulation-based Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Bit value ordering
  - Variable ordering
  - Test generation using our methodology
- Experiments
- Conclusion

Two Similar SAT Problems

SAT 1

SAT 2



Ordering: a, a', b, b', c, c'

Ordering: a, a', b, b', c, c'

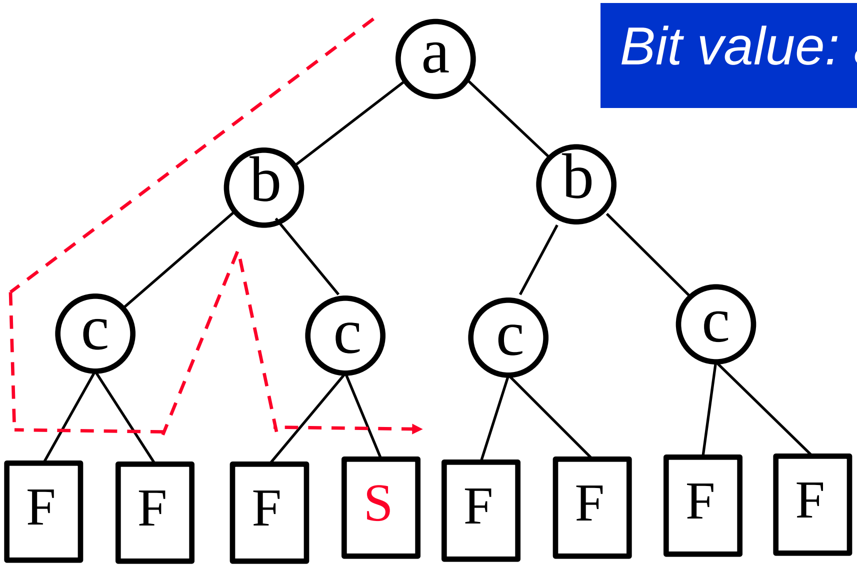
Without Learning, 7 conflicts in SAT2.

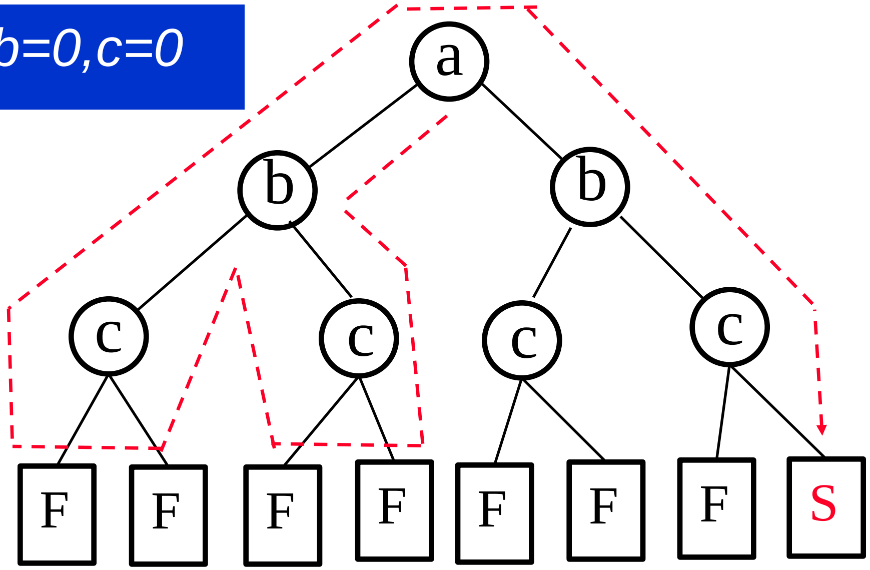
Learning: Bit Value Ordering

SAT 1

SAT 2

Bit value: a=1,b=0,c=0





Ordering: a, a', b, b', c, c'

Ordering: a, a', b', b, c', c

With bit value learning, 4 conflicts in SAT2.

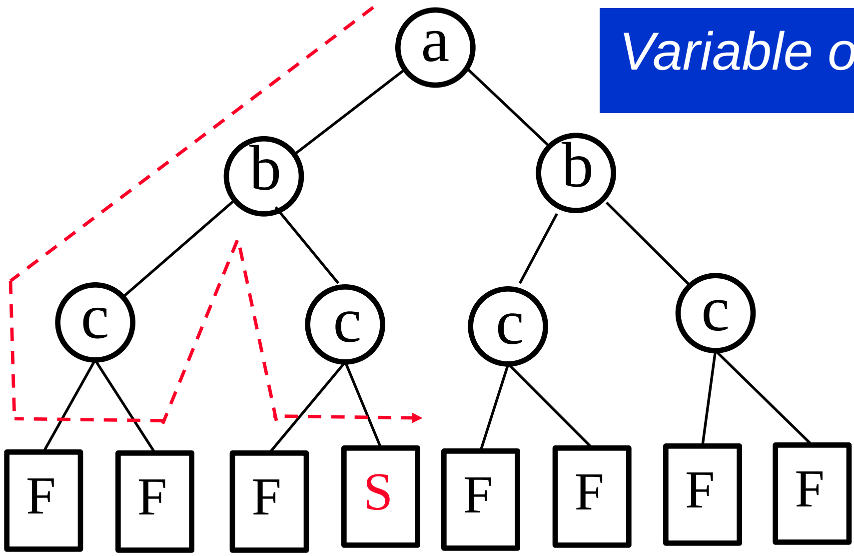
Learning: Variable Ordering

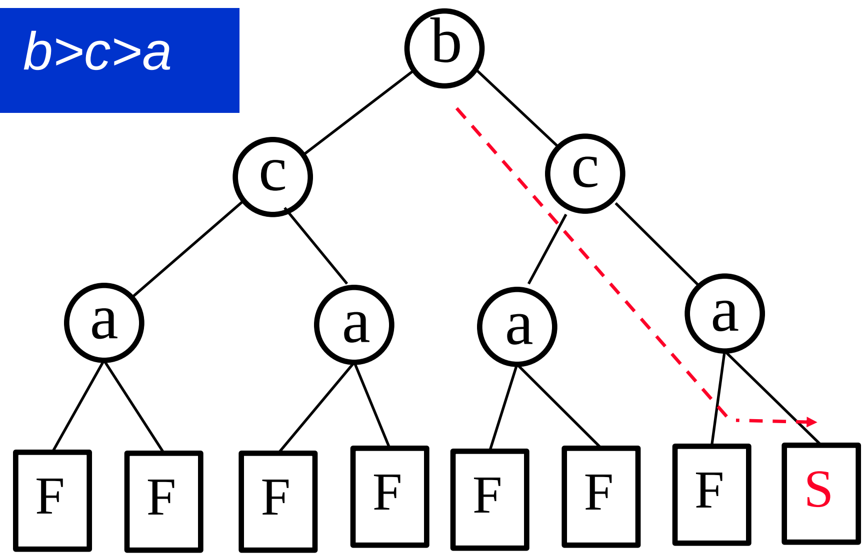
SAT 1

Bit value: a=1,b=0,c=0

Variable order: b>c>a

SAT 2





Ordering: a, a’, b, b’, c, c’

Ordering: b’, b, c’, c, a, a’

With bit value+ variable order learning, 1 conflict in SAT2.

Test Generation Using Our Method

Inputs: a) Formal model, *D*

b) A cluster of properties *P* with satisfiable bounds

1. Initialize *varStat*
2. Select the base property p_1, and generate CNF_1
3. ($assignment_1$, $test_1$) = SAT(CNF_1)
4. *Test-suite* = {$test_1$}
5. for *i* is from 2 to the size of *P*
   a) Update *varStat* using assignment_{i-1}
   b) Generate CNF_i = BMC(*D*, p_i, $bound_i$)
   c) ($assignment_i$, $test_i$) = SAT(CNF_i)
   d) *Test-suite* = *Test-suite* U {$test_i$}

   endfor
6. Return *Test-suite*

An Illustrative Example with 3 properties

VarStat	a	b	c	d	…
[0] V	0	0	0	0	…
[1] V	0	0	0	0	…

P1: a=0, b=0, c=1, d=1

VarStat	a	b	c	d	…
[0] V	1	1	0	0	…
[1] V	0	0	1	1	…

P2: a=0, b=0, c=1, d=0

VarStat	a	b	c	d	…
[0] V	2	2	0	1	…
[1] V	0	0	2	1	…

Predict ordering for P3

P3: a=0, b=0, c=1, d=?

score(a) ↑, score(a')↑

score(b) ↑, score(b')↑

score(c) ↑, score(c')↑

Approach: Using the statistics of the counterexamples when checking the properties in a cluster

- Count the number of values ➔ bit value ordering
- Variance of counts of two literals ➔ variable ordering

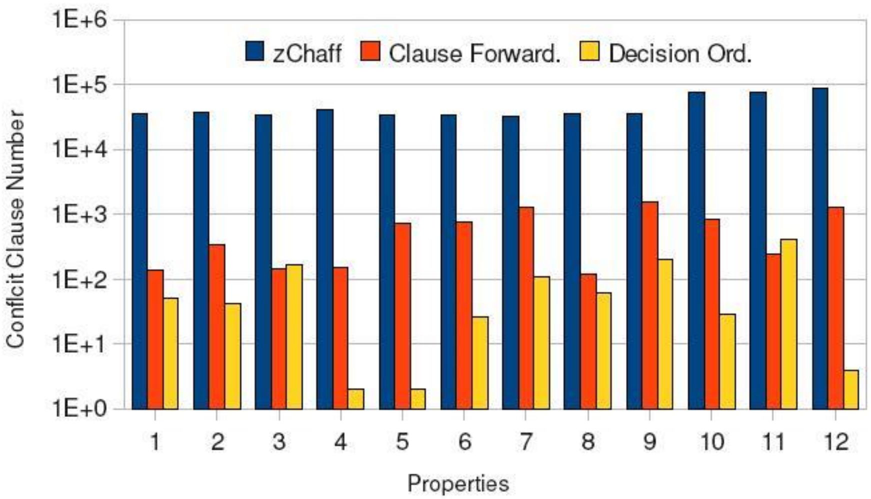
Outline

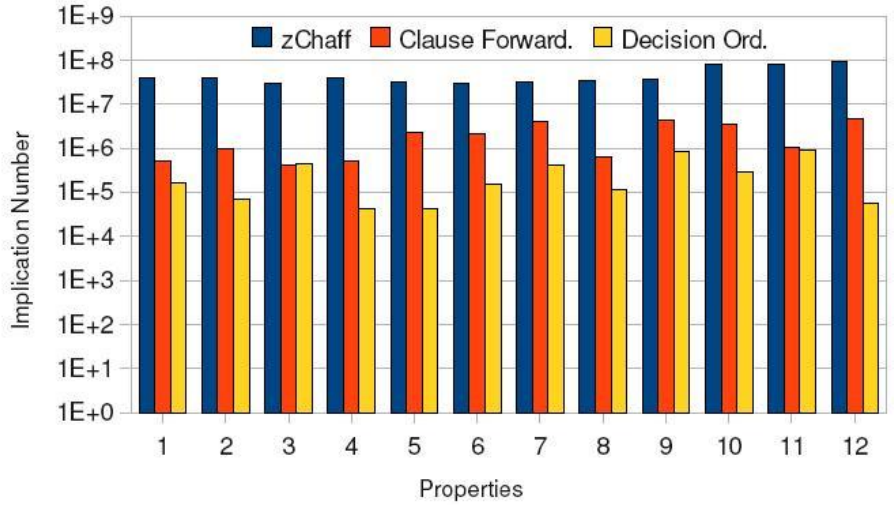
- Introduction
- Simulation-based Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Bit value ordering
  - Variable ordering
  - Test generation using our methodology
- Experiments
- Conclusion

Case Study: MIPS Processor

Property (test)	zChaff (sec)	Conflict Clause Forwarding	Improvement Factor	Decision Ordering	Improvement Factor
ALU	23.20	23.20	**1**	23.20	**1**
P1	20.73	2.74	**7.57**	0.18	**15.22**
P2	21.33	3.01	**7.09**	0.15	**20.07**
P3	18.03	2.70	**6.68**	0.29	**9.31**
DIV	18.78	18.78	**1**	18.78	**1**
P4	23.55	2.72	**8.66**	0.13	**20.92**
P5	18.31	3.60	**5.09**	0.14	**25.71**
P6	18.11	3.72	**4.87**	0.18	**20.67**
FADD	22.90	22.90	**1**	22.90	**1**
P7	16.95	4.46	**3.80**	0.23	**19.39**
P8	18.89	2.71	**6.97**	0.16	**16.94**
P9	19.80	4.70	**4.21**	0.39	**12.05**
MUL	64.21	64.21	**1**	64.21	**1**
P10	59.15	3.36	**17.60**	0.24	**14.00**
P11	59.65	3.85	**15.49**	0.45	**8.56**
P12	73.98	6.28	**11.78**	0.18	**34.89**

Case Study: MIPS Processor





Indications: Test generation complexity is significantly improved

- Reduction of conflict clauses
- Reduction of implication number

Case Study: OSES

- This case study is a on-line stock exchange system. The activity diagram consists of 27 activities, 29 transitions and 18 key paths.

Cluster	Size	zChaff	Conflict Forward	Improve ment Factor	Decision Ordering	Improvement Factor
C1	3	1.18	2.18	**0.54**	0.70	**3.11**
C2	4	14.53	9.53	**1.52**	0.78	**12.22**
C3	8	375.91	170.06	**2.21**	36.19	**4.70**
C4	4	12.98	8.33	**1.56**	1.24	**6.72**
C5	4	7.13	16.88	**0.42**	1.02	**16.55**
C6	8	720.13	474.68	**1.52**	28.60	**16.60**
C7	4	10.80	24.55	**0.44**	1.95	**12.59**
C8	8	656.95	321.14	**2.05**	77.65	**4.14**
C9	8	248.17	82.42	**3.01**	37.93	**2.17**
Average	**-**	**227.53**	**123.21**	**1.85**	**20.67**	**5.97**

Conclusions

- Functional validation is a major bottleneck
  - SAT-based approaches are promising for automated test generation.
- Proposed an efficient technique for generation of directed tests using learning techniques
  - Developed a novel decision ordering technique using both bit-value ordering and variable ordering
- Successfully applied on both hardware and software designs
  - Significant reduction in overall validation effort

Thank you !