SAT Based Efficient Directed Test Generation Techniques

Presented

by

Mingsong Chen

Software Engineering Institute

East China Normal University

May 5, 2011

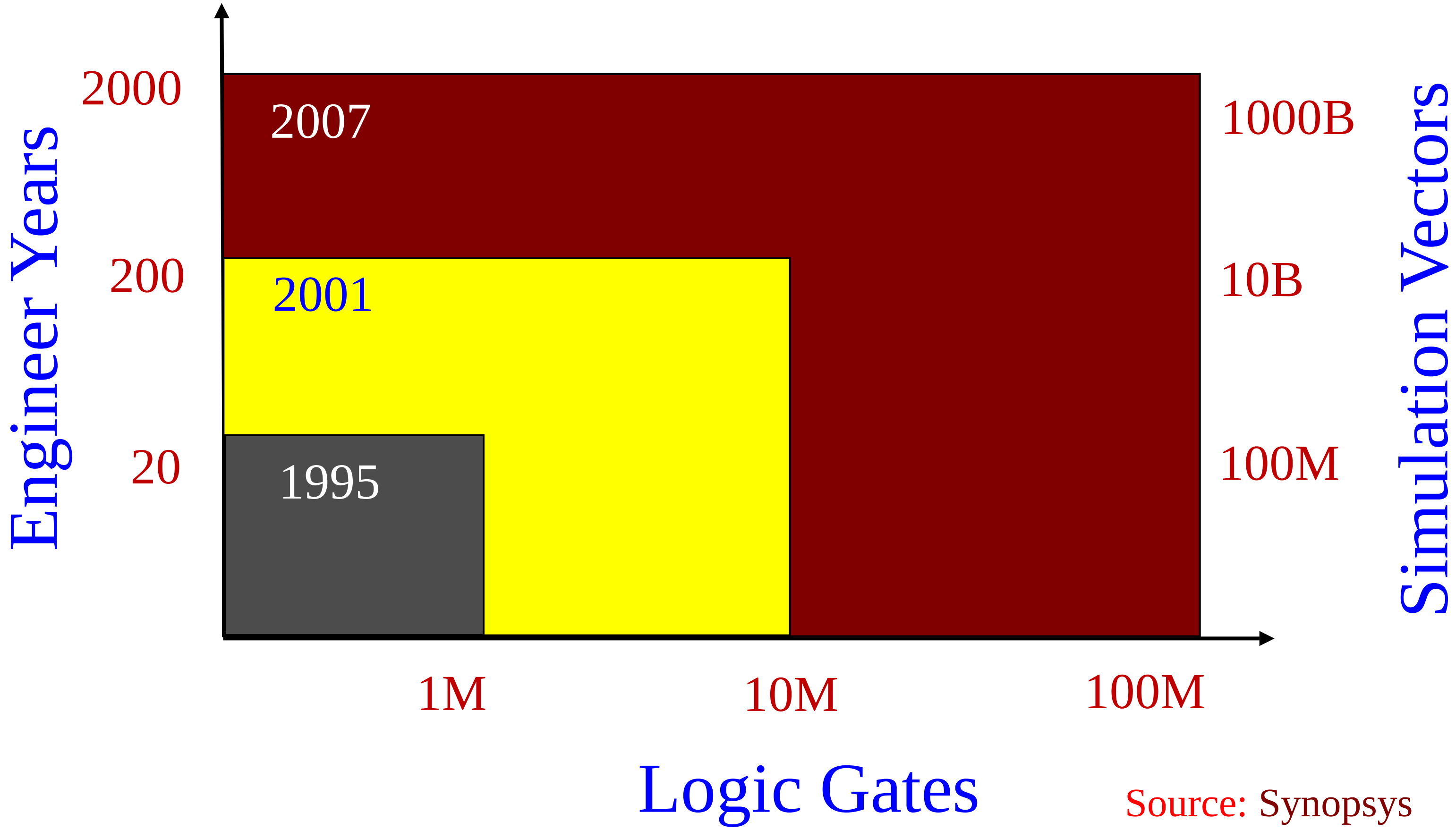
Outline

- **Introduction**
  - Model Checking Based Test Generation
  - SAT-based Bounded Model Checking
    - DPLL algorithm
    - Conflict clause
- **Efficient Test Generation Approaches**
  - Conflict clause forwarding based approaches
  - Decision ordering based techniques
  - Property decomposition based methods
- **Conclusion**

Outline

- **Introduction**
  - Model Checking Based Test Generation
  - SAT-based Bounded Model Checking
    - DPLL algorithm
    - Conflict clause
- **Efficient Test Generation Approaches**
  - Conflict clause forwarding based approaches
  - Decision ordering based techniques
  - Property decomposition based methods
- **Conclusion**

Functional Validation of SOC Designs



Source: Synopsys

Functional validation is a major bottleneck during SoC development! (up to 70% of time and resources are used)

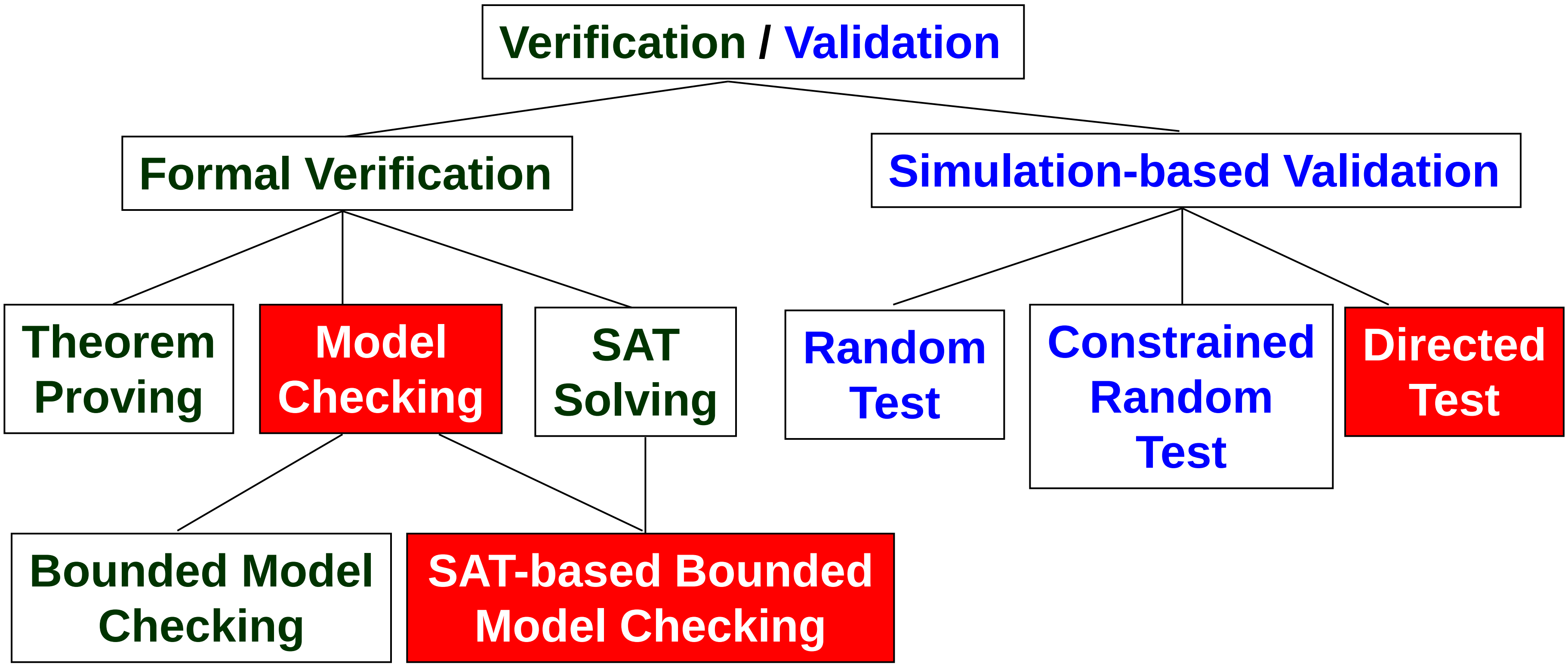
Source: G. Spirakis, keynote address at DATE 2004

Functional Validation Methods

- **Simulation (Validation)**
  - The process of gaining confidence by examining the behavior of the implementation using **input/output test vectors**
  - **Incompleteness** verification: not possible for all input vectors
  - Applicable to **large designs**

- **Formal (Verification)**
  - **Mathematical proof** that a system (implementation) behaves according to a given set of requirements (specification)
  - **Complete verification**
  - Applied to **small and critical components** due to the state space explosion problem

Approaches for Specification Validation



Validation using a combination of simulation based techniques and formal methods.

Test Generation using Model Checking

❑ Model Checking (MC)

- ❖ Specification is translated to formal models, e.g., SMV
- ❖ Desired behaviors in temporal logic properties, e.g. LTL
- ❖ Property falsification leads to counterexamples (tests)

❑ Test Generation

- ❖ ***Generate a counterexample: sequence of variable assignments***

Problem: **Test generation is very costly or not applicable in many complex scenarios.**

Approach: **Exploit learning to reduce validation complexity**
- Reduction of test generation time
- Enables test generation in complex scenarios

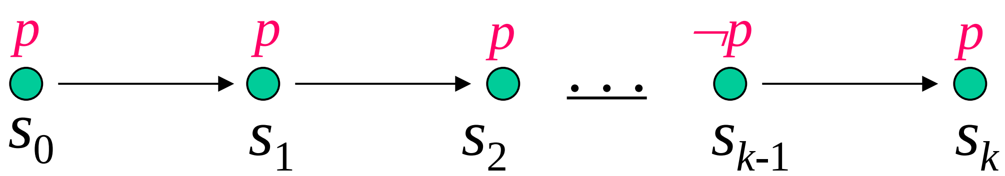
Outline

- **Introduction**
  - Model Checking Based Test Generation
  - SAT-based Bounded Model Checking
    - DPLL algorithm
    - Conflict clause
- **Efficient Test Generation Approaches**
  - Conflict clause forwarding based approaches
  - Decision ordering based techniques
  - Property decomposition based methods
- **Conclusion**

SAT-based Bounded Model Checking

- The safety property ***P*** is valid up to cycle ***k*** iff **Ω(*k*)** is not satisfiable.

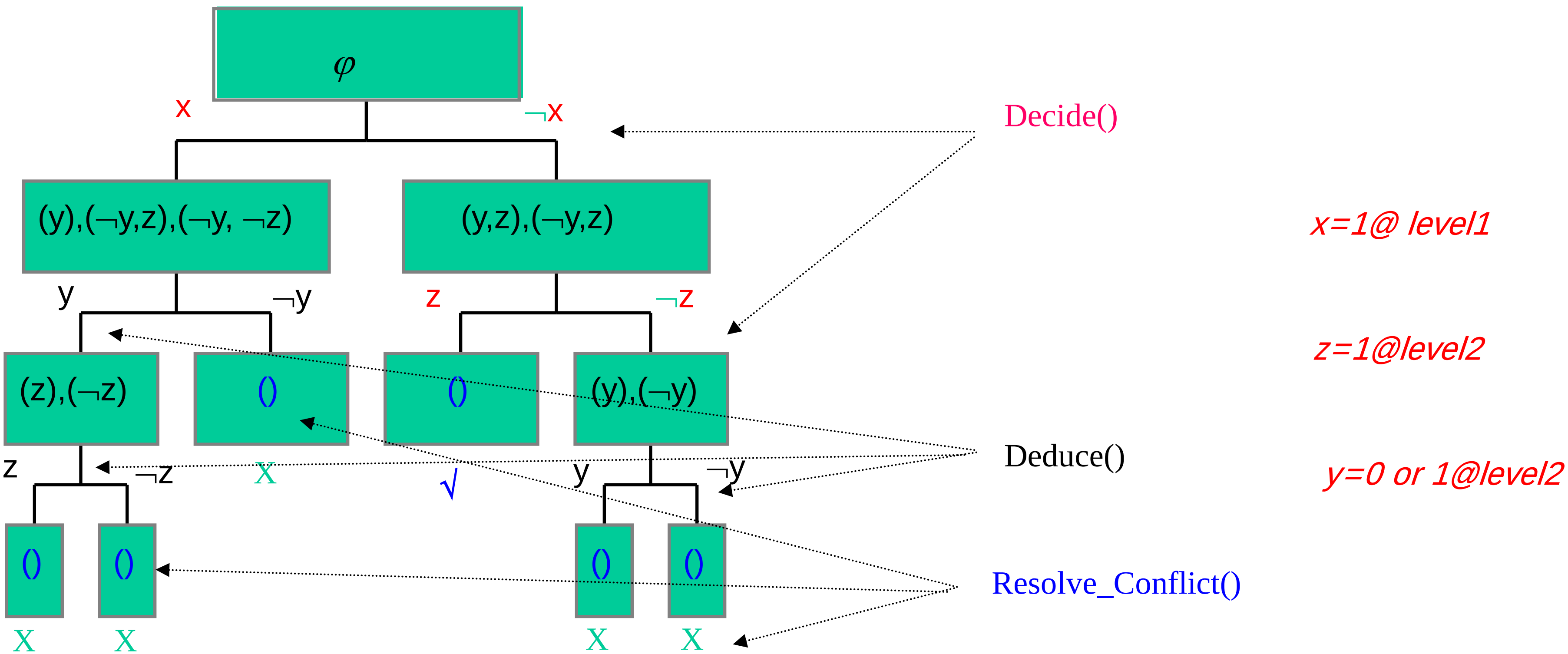
$$\Omega(k) = I(S_0) \wedge \bigwedge_{i=0}^{k-1} R(S_i, S_{i+1}) \wedge \bigvee_{i=0}^{k} \neg P(s_i)$$



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

SAT Decision Procedure

Given a φ in CNF: $(x+y+z)(\neg x+y)(\neg y+z)(\neg x+\neg y+\neg z)$



DPLL Algorithm

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

Implication

Conflict Backtrack

BCP = Implication Number + Conflict Backtrack

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

Implication Graph, Conflict Clause

$$\omega_1 = (x_2 \vee x_6 \vee \neg x_4)$$

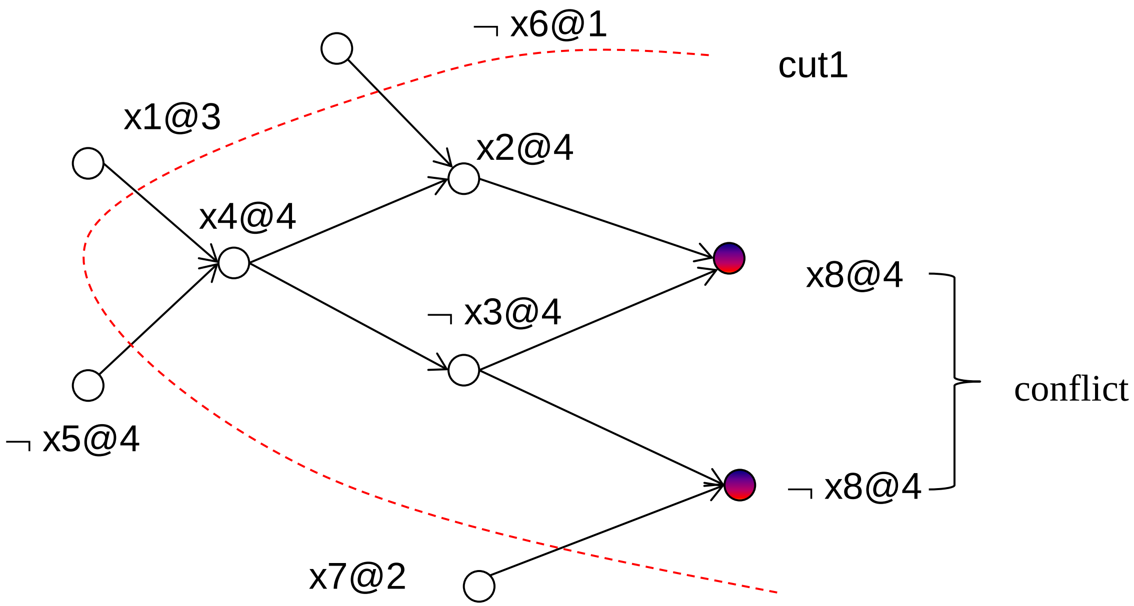
$$\omega_2 = (\neg x_8 \vee x_3 \vee \neg x_7)$$

$$\omega_3 = (\neg x_1 \vee x_4 \vee x_5)$$

$$\omega_4 = (\neg x_3 \vee \neg x_4)$$

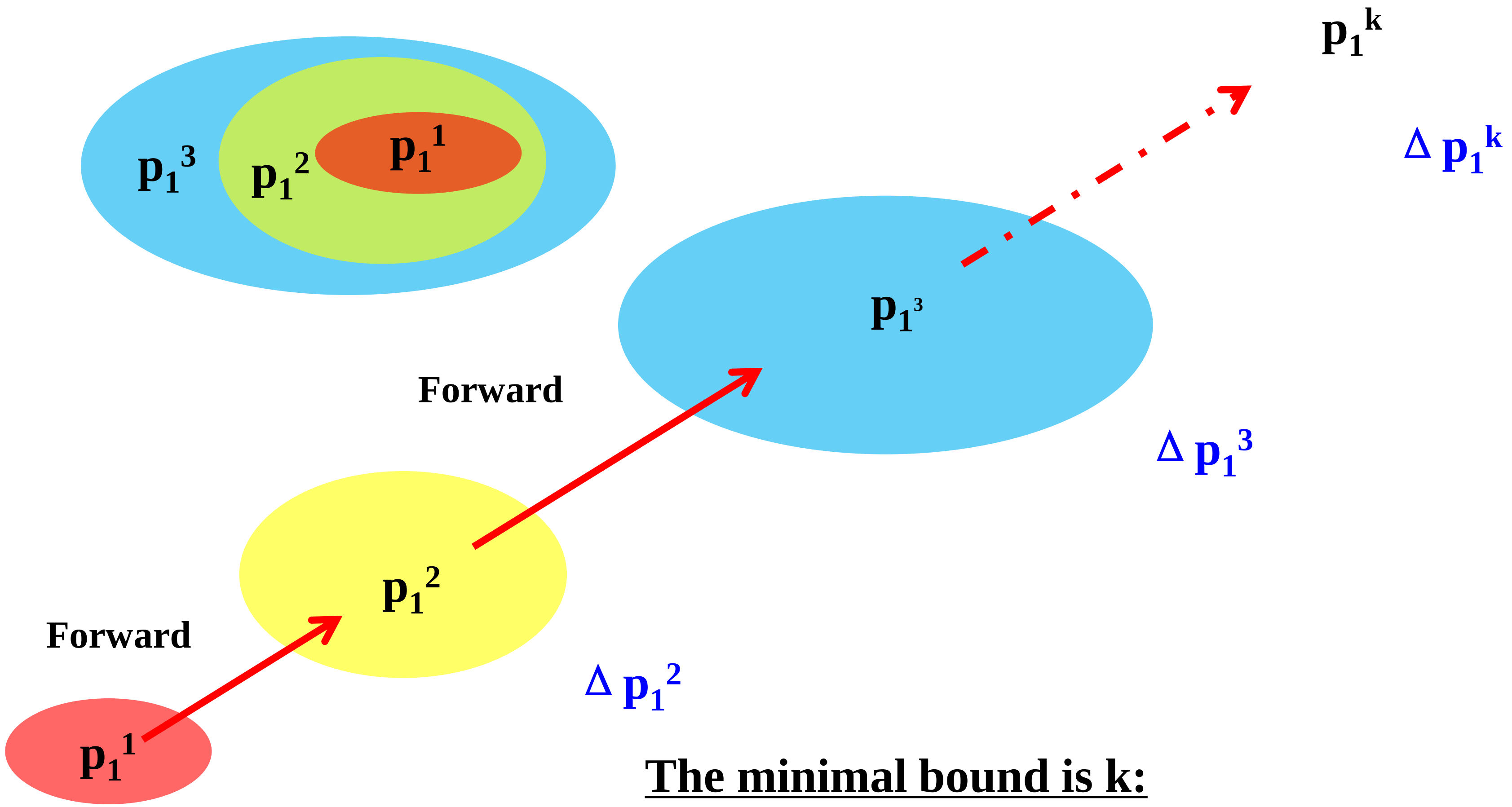
$$\omega_5 = (\neg x_2 \vee x_3 \vee x_8)$$

$$\omega_6 : (\neg x_1 \vee x_5 \vee x_6 \vee \neg x_7)$$



- **Conflict clause can be treated as the knowledge learned during the SAT solving. It is a restriction of the variable assignment.**

Same Property but Different Bounds



The minimal bound is k:

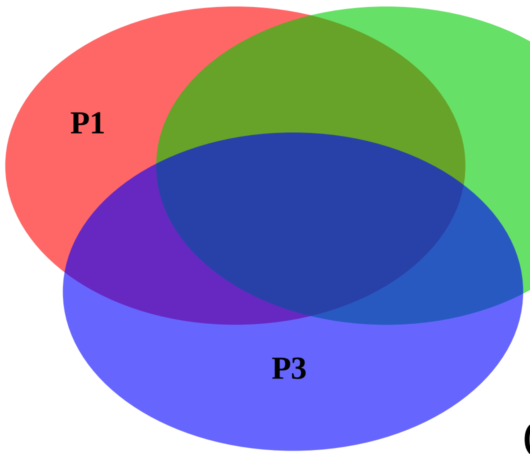
Save： $\Delta\, p_1^2 + \Delta\, p_1^3 + \ldots + \Delta\, p_1^{k-1} + \ldots + \Delta\, p_1^k$

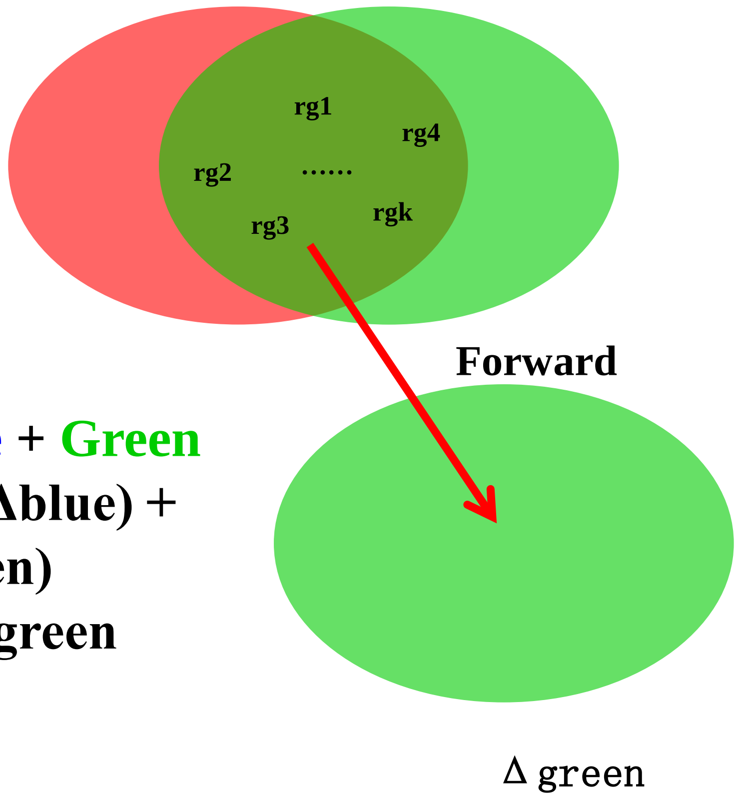
O. Strichman. Pruning Techniques for the SAT-Based Bounded Model Checking Problems. *CHARME , 2001*

Outline

- ❑ **Introduction**
  - ❖ Model Checking Based Test Generation
  - ❖ SAT-based Bound Model Checking
    - ➢ DPLL algorithm
    - ➢ Conflict clause
- ❑ **Efficient Test Generation Approaches**
  - ❖ Conflict clause forwarding based approaches
  - ❖ Decision ordering based techniques
  - ❖ Property decomposition based methods
- ❑ **Conclusion**

Same Design, Different Properties



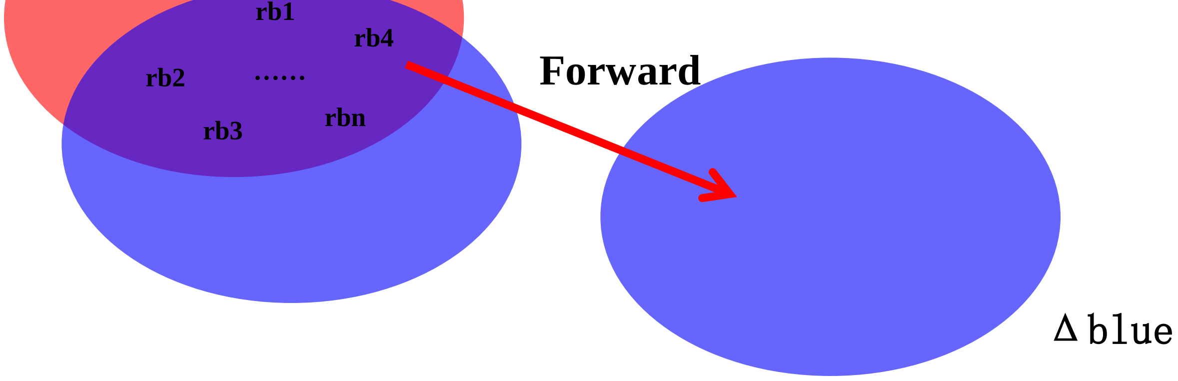


<u>Benefit:</u>

Original: Red + Blue + Green

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

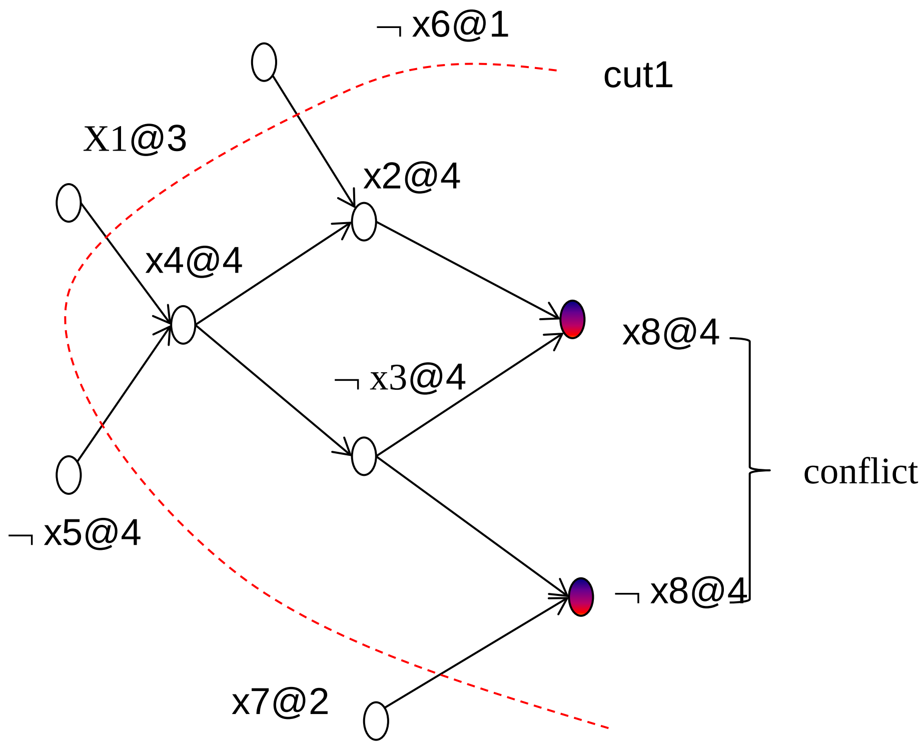
Save: Δblue + Δgreen



Property Clustering

- **Clustering properties is to exploit the structural and behavior similarity and maximize the validation reuse**
- **Property clustering methods:**
  - Based on structural similarity
  - Based on textual similarity
  - Based on Influence (Cone of Influence)
  - Based on CNF intersections

Identification of Common Conflict Clauses



Conflict Clause

(¬ X1 ∨ X5 ∨ X6 ∨ ¬ X7 **)**

Conflict Side Clauses

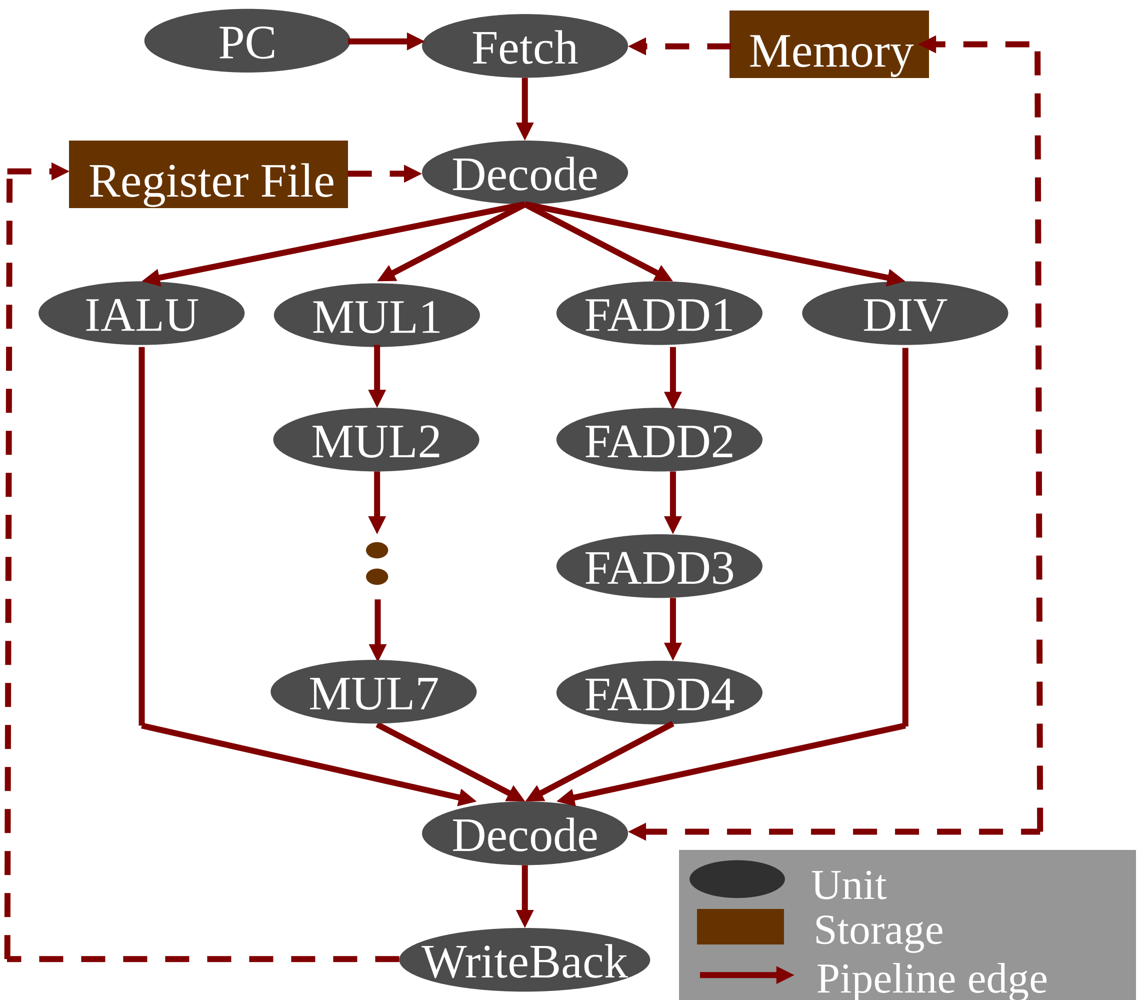
Clauses	Group ID 4	3	2	1
(¬X2 ∨X3 ∨X8)	**0**	**1**	**1**	**1**
(X3 ∨¬X7 ∨¬X8)	**1**	**0**	**1**	**0**
(X2 ∨¬X3 ∨X6)	**1**	**1**	**1**	**1**
(¬X3 ∨¬X4)	**1**	**0**	**1**	**0**
(¬X1 ∨X4 ∨X5)	**1**	**1**	**1**	**0**

Let ∧ be the bit "AND" operation. (0111 ∧ 1010 ∧ 1111 ∧ 1010 ∧ 1110) = 0010.
So the conflict clause (¬X1 ∨X5 ∨X6 ∨¬X7) can be reused for property 2.

Test Generation For A Property Cluster

1. **Cluster** the properties based on similarity
2. **for** each cluster i, of properties
   ① **Select** base property p^i_1, and generate CNF^i_1
   ② **for** each CNF^i_j of p^i_j ($j \neq 1$) in cluster i
      a) Perform name substitution on CNF^i_j
      b) Compute intersection INT^i_j between CNF^i_1 and CNF^i_j
      c) Mark the clauses of CNF^i_1 using INT^i_j
      endfor
   ③ **Solve** CNF^i_1 to get the conflict clauses CC^i_1 and $test^i_1$
   ④ **for** each CNF^i_j ($j \neq 1$)
      a) $CNF^i_j = CNF^i_j$ + Filter (CC^i_j, j)
      b) Solve CNF^i_j to get $test^i_j$
      endfor
   endfor

Case Study 1 : MIPS Processor



The Architecture

MIPS Processor
- 20 nodes
- 24 edges
- 91 instructions

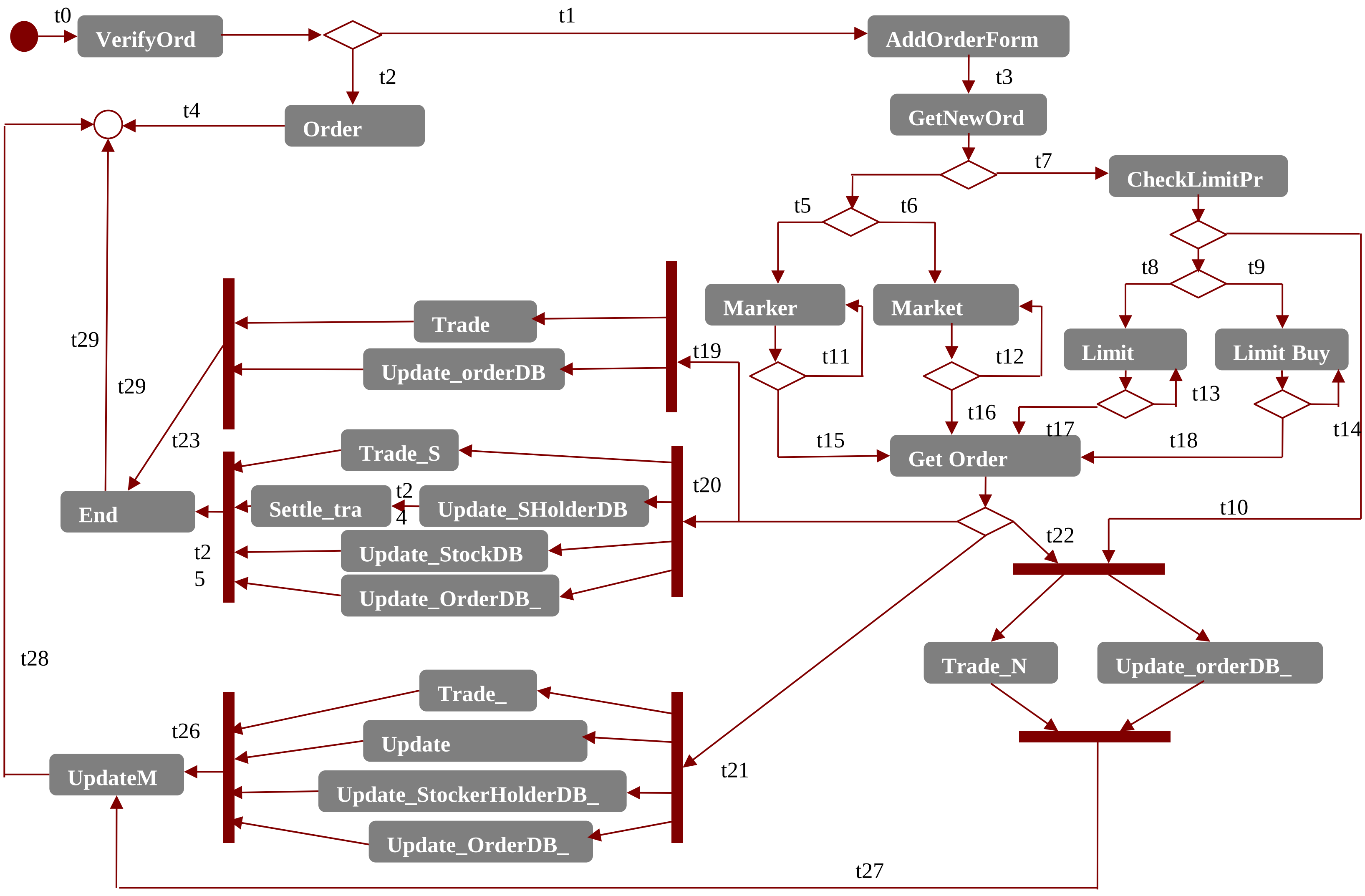
MIPS Processor Results

- The processor has five pipeline stages: fetch, decode, execute, memory and writeback.
- There are totally 171 properties generated.

Methods	*Structure*	*Textual*	*Influence*	*Intersection*
Num. of Clusters	16	32	27	17
zChaff (sec.) (Existing Approach)	3275.07	3266.73	3241.00	3323.34
Our Method (sec.)	957.42	879.19	754.58	751.36
Speedup	3.42	3.72	4.33	4.42

zChaff is a state-of-the-art SAT Solver.

Case Study 2 : OSES



OSES Results

- This case study is a on-line stock exchange system. The activity diagram consists of 27 activities, 29 transitions and 18 key paths. There are totally 51 properties.

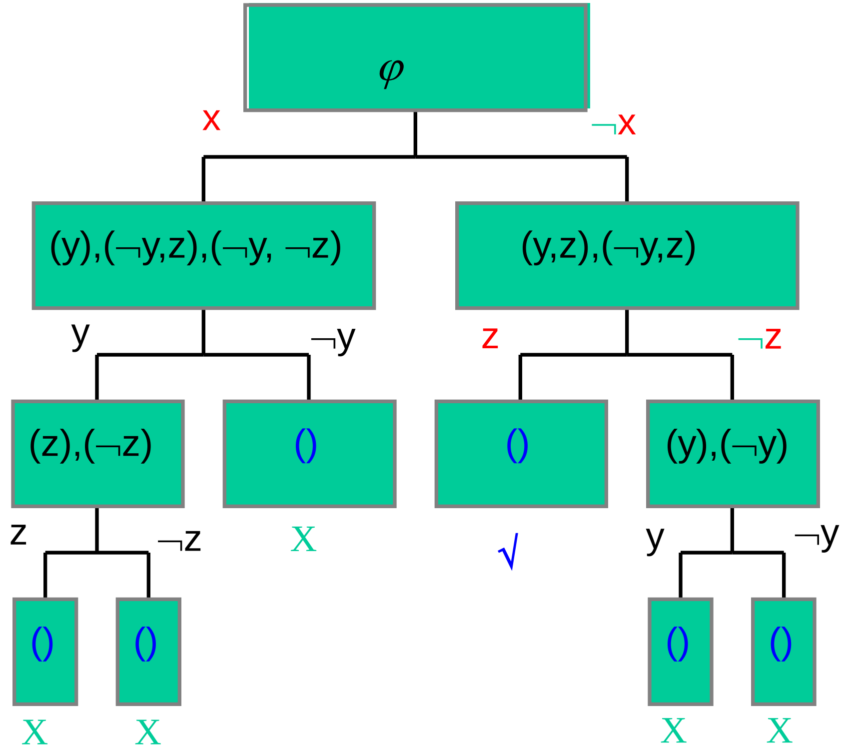
Methods	*Structure*	*Textual*	*Influence*	*Intersection*
Num. of Clusters	18	9	12	13
zChaff (sec.) (Existing Approach)	2119.16	2159.92	2311.47	2134.26
Our Method (sec.)	939.25	926.98	966.19	794.48
Speedup	2.26	2.33	2.44	2.69

Outline

- Introduction
  - Model Checking Based Test Generation
  - SAT-based Bounded Model Checking
    - DPLL algorithm
    - Conflict clause
- Efficient Test Generation Approaches
  - Conflict clause forwarding based approaches
  - Decision ordering based techniques
  - Property decomposition based methods
- Conclusion

Decision Ordering Problem

Given a φ in CNF: (x+y+z)(¬x+y)(¬y+z)(¬x+¬ y+¬ z)

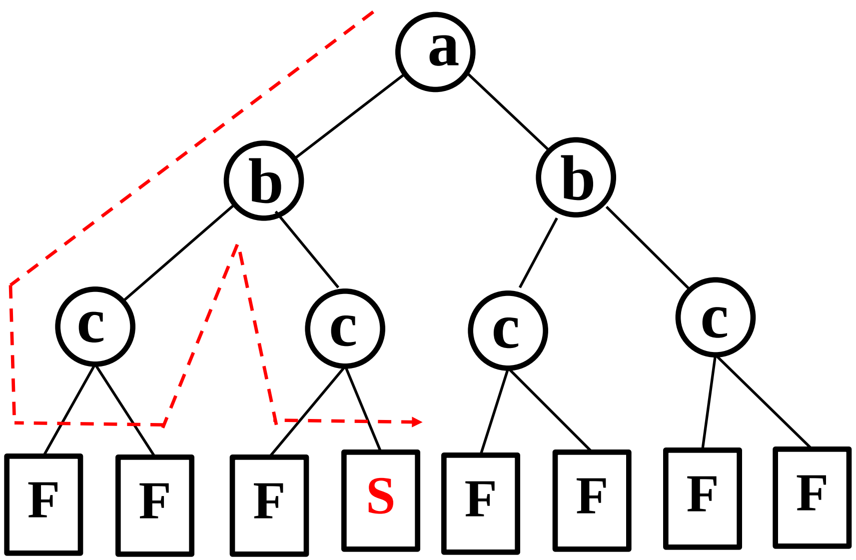


- **A wise decision ordering can quickly locate the true assignment.**
  - **Bit value ordering**
  - **Variable Orderinig**

Best decision: ¬ x, z

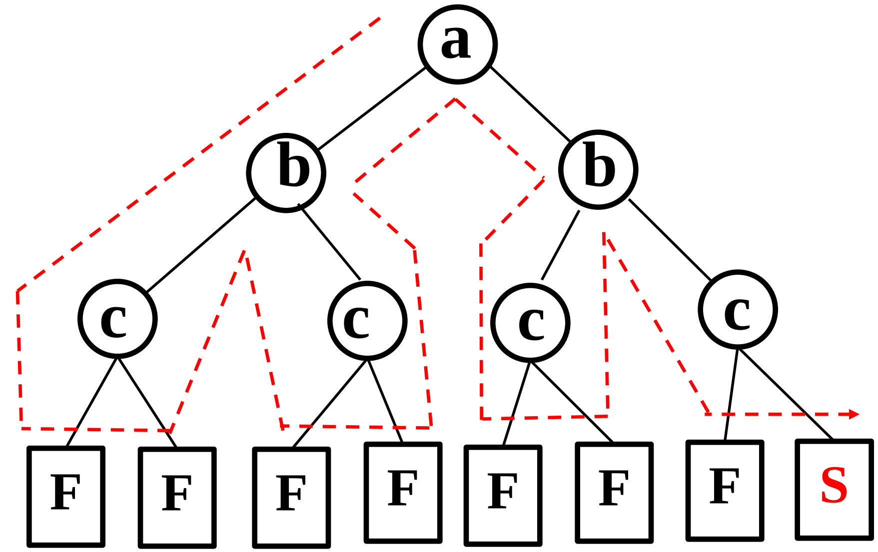
Two Similar SAT Problems

SAT 1



Ordering: a, a’, b, b’, c, c’

SAT 2

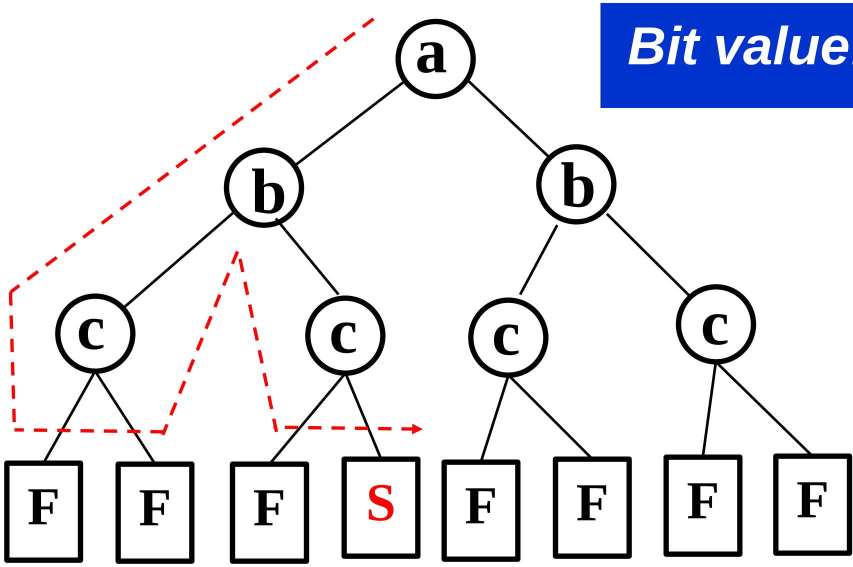


Ordering: a, a’, b, b’, c, c’

Without Learning, 7 conflicts in SAT2.

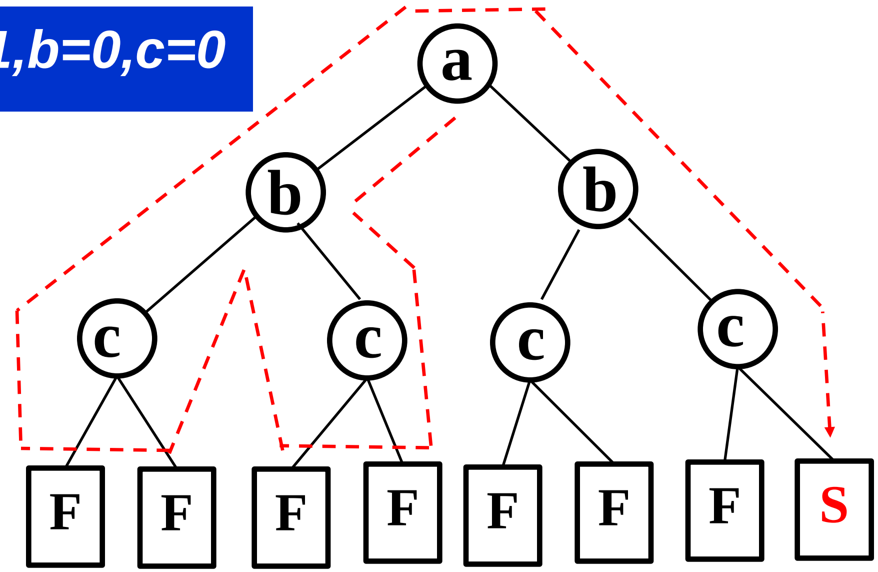
Learning: Bit Value Ordering

SAT 1



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

SAT 2



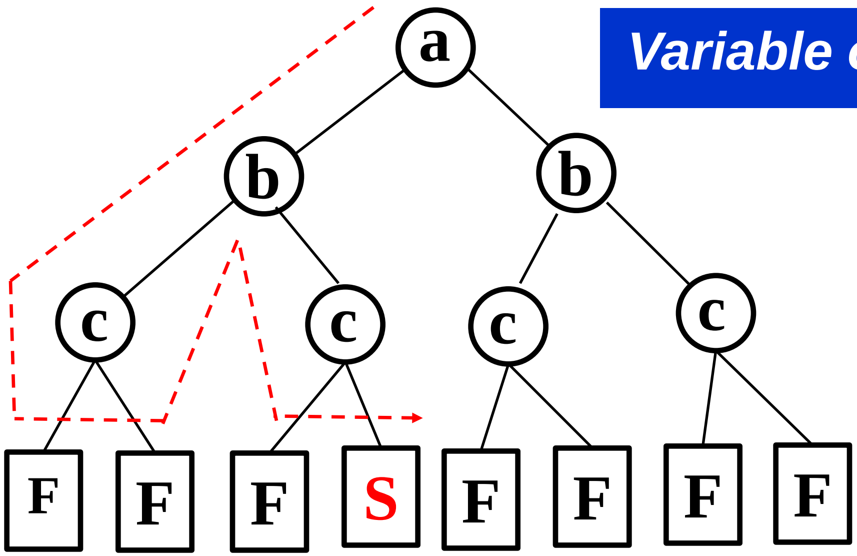
Ordering: a, a’, b, b’, c, c’

Ordering: a, a’, b’, b, c’, c

With bit value learning, 4 conflicts in SAT2.

Learning: Bit Value + Variable Ordering

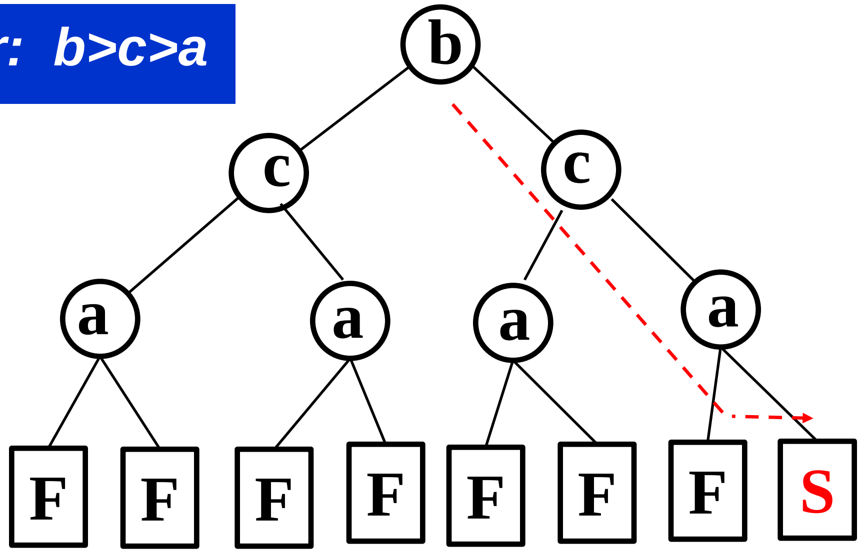
SAT 1



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.

Our method – An 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

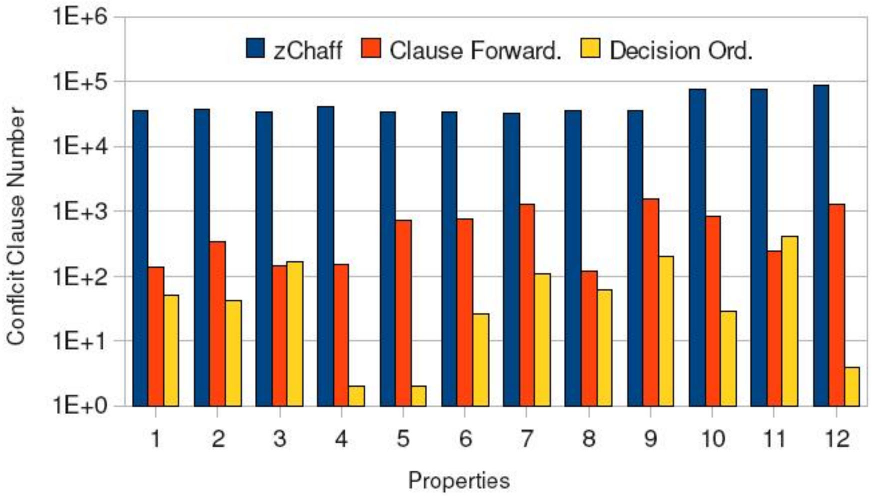
- **Count of values → bit value ordering**
- **Variance of counts of two literals → variable ordering**

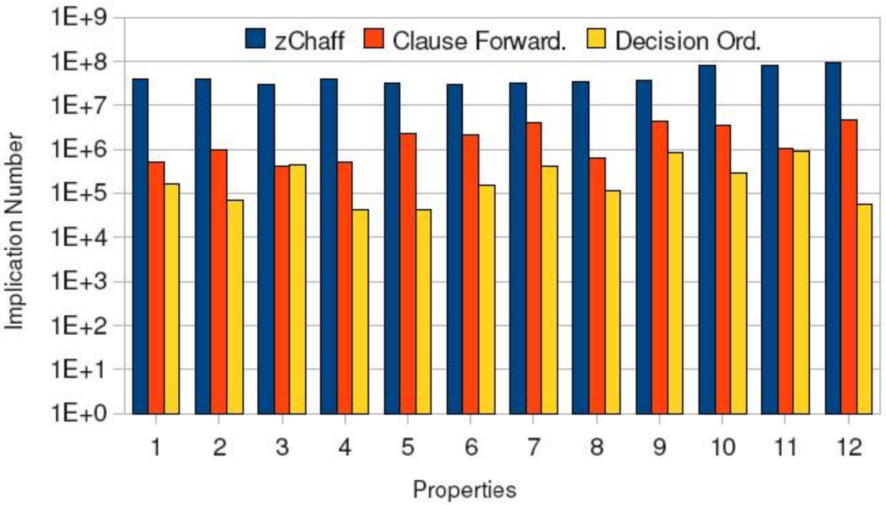
Case Study 1 : MIPS Processor

- For each function unit (ALU, DIV, FADD and MUL) in the pipelined processor. We generate 4 properties.

Property (test)	zChaff (sec)	Clustering	Speedup (over zChaff)	Decision Ordering	Speedup (over Clustering)
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 1 : MIPS Processor





Test generation time is significantly improved

- Drastic reduction of conflict clauses

- Drastic reduction in number of implications

Case Study 2 : OSES

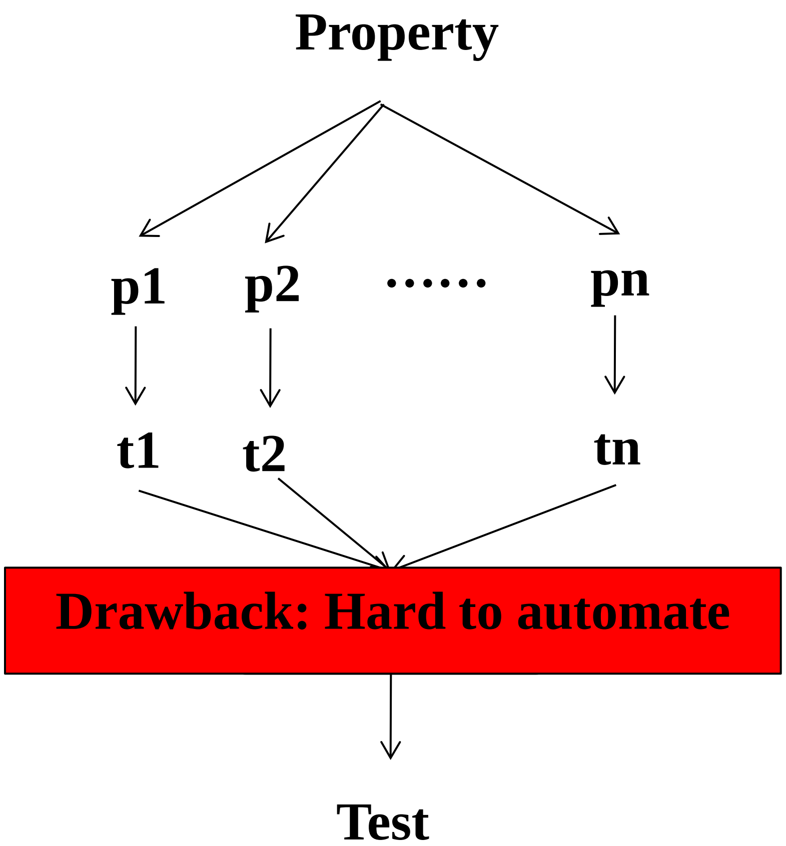
- This case study is a on-line stock exchange system. The activity diagram consists of 27 activities, 29 transitions and 18 key paths.

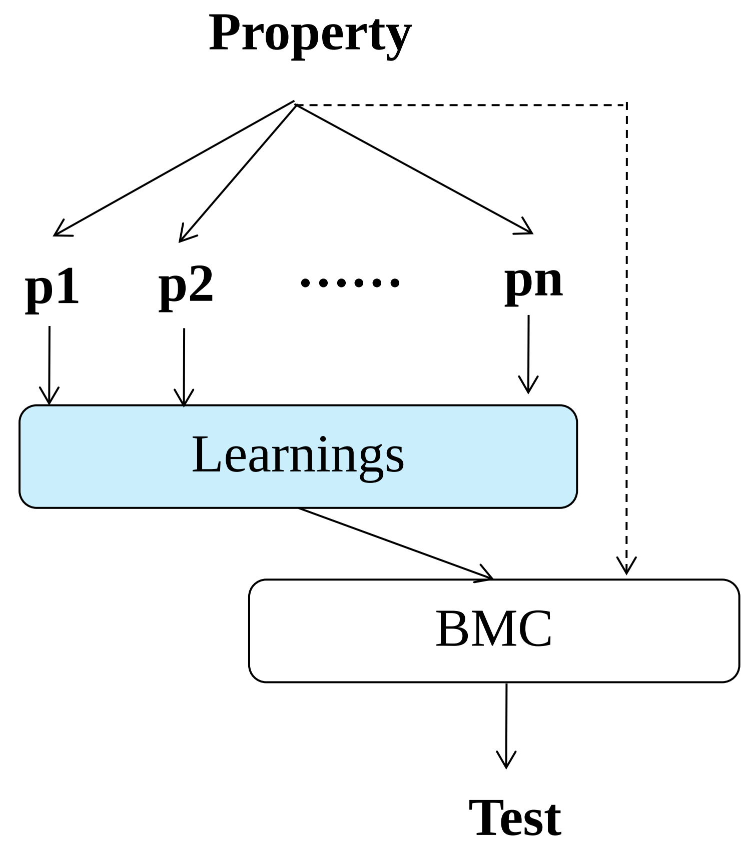
Cluster	Size	zChaff	Clustering	Speedup (over zChaff)	Decision Ordering	Speedup (over Clustering)
C1	3	1.18	2.18	**0.54**	0.70	**3.11**
C2	4	14.53	9.53	**1.52**	0.78	**12.22**
C3	8	375.91	170.06	**2.21**	36.19	**4.70**
C4	4	12.98	8.33	**1.56**	1.24	**6.72**
C5	4	7.13	16.88	**0.42**	1.02	**16.55**
C6	8	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**

Outline

- Introduction
  - Model Checking Based Test Generation
  - SAT-based Bounded Model Checking
    - Implication graph
    - SAT decision procedure – DPLL algorithm
- Efficient Test Generation Approaches
  - Conflict clause forwarding based approaches
  - Decision ordering based techniques
  - Property decomposition based methods
- Conclusion

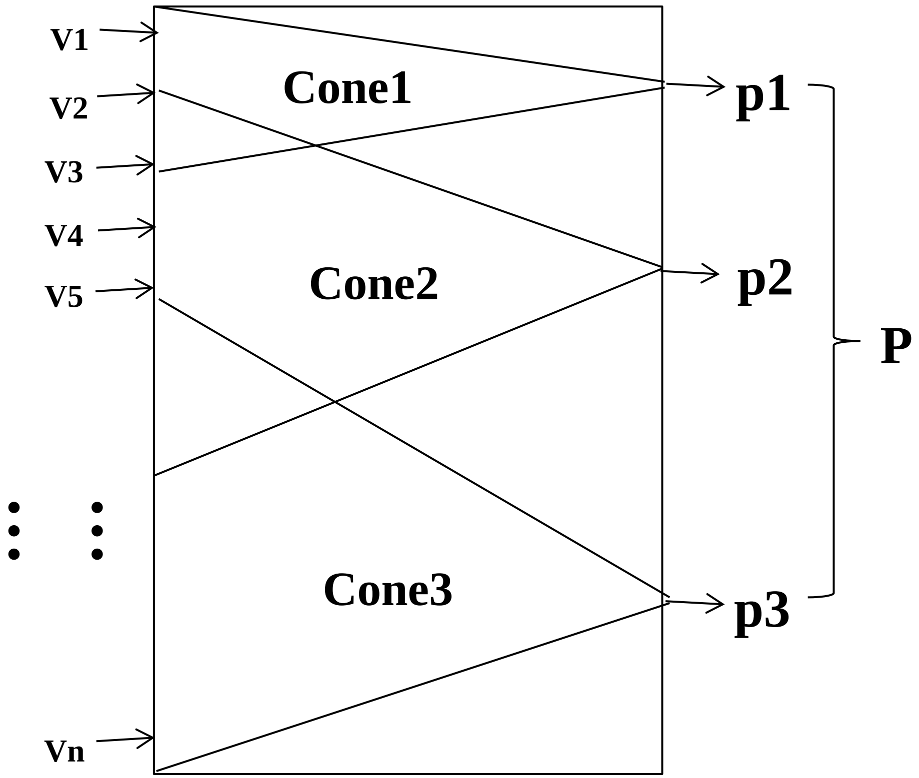
Property Decomposition Techniques





Koo et al. Functional Test Generation using Property Decomposition Techniques. ACM *TECS, 2009*

Spatial Decomposition

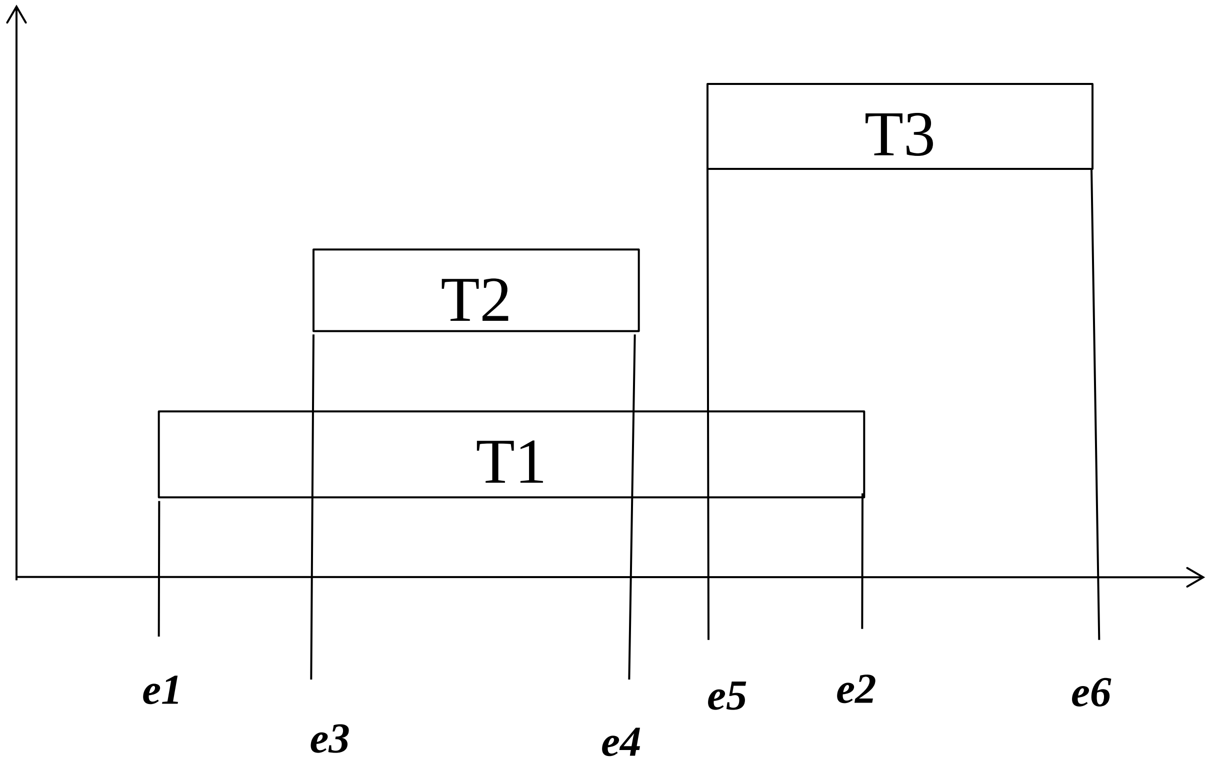


COI(p1) < COI(p2) < COI(p3) <COI(P)

Time(p1) < Time(p2) < Time(p3) <Time(P)

Learning from P1 can reduce the Time(P) ?

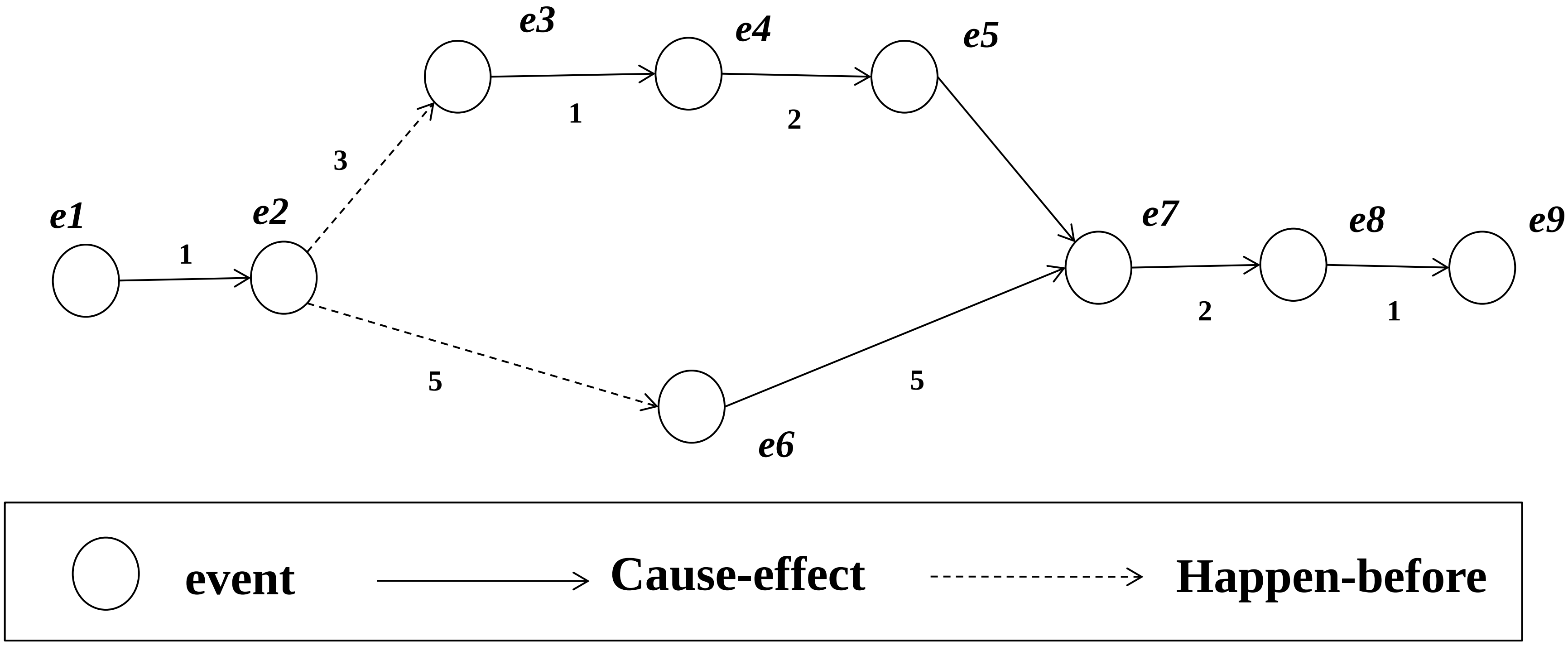
Temporal Decomposition



Cause effect relation: **e1→e2 e3→e4 e5→e6**

Happen before relation: **e1<e3<e4 <e5<e2<e6**

Temporal Decomposition



$$!F(e1) \rightarrow !F(e3) \rightarrow !F(e7) \rightarrow !F(e9)$$

Case Study 1: MIPS Processor

- We generated 6 complex properties based on interaction faults on various function unit (ALU, DIV, FADD and MUL), which cannot handled by temporal decomposition.

Property (test)	zChaff (sec)	Num. of Clusters	Num. of Sub-props	Spatial (sec)	Speedup
P1	127.52	3	**2**	49.41	**2.58**
P2	49.24	3	**2**	15.73	**3.13**
P3	9.18	2	**1**	4.99	**1.84**
P4	13.78	2	**1**	7.28	**1.89**
P5	31.63	3	**2**	12.74	**2.48**
P6	120.72	3	**2**	54.21	**2.23**

Speedup: 1.84-3.13 times

Case Study 2 : OSES

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

Property	zChaff (sec)	Bound	Num. of Sub-properties	Temporal (sec)	Speedup
P1	25.99	8	3	0.78	**33.32**
P2	48.99	10	4	2.69	**18.21**
P3	39.67	11	5	3.45	**11.50**
P4	247.26	11	5	22.46	**11.01**
P5	160.73	11	5	15.68	**10.25**
P6	97.54	11	4	1.56	**62.53**
P7	31.39	10	4	12.31	**2.55**
P8	161.74	11	4	12.62	**12.82**
P9	142.91	10	4	17.57	**8.13**
P10	33.77	10	4	1.76	**19.19**

Speedup: 3-62 times

Outline

- **Introduction**
  - Model Checking Based Test Generation
  - SAT-based Bounded Model Checking
    - DPLL algorithm
    - Conflict clause
- **Efficient Test Generation Approaches**
  - Conflict clause forwarding based approaches
  - Decision ordering based techniques
  - Property decomposition based methods
- **Conclusion**

Conclusion

- Validation is a major bottleneck in HW/SW designs
- This presentation discusses how to reduce the overall validation effort for directed test generation from models.
  1. Conflict clause forwarding and property clustering methods
  2. Efficient decision ordering approaches
  3. Property decomposition techniques
- Successfully applied on both HW/SW designs
  - **Several orders of magnitude reduction in overall validation effort**

Thank you !