Decision Ordering Based Property Decomposition for Functional Test Generation

Mingsong Chen

Software Engineering Institute

East China Normal University, China

Prabhat Mishra

CISE Department

University of Florida, USA

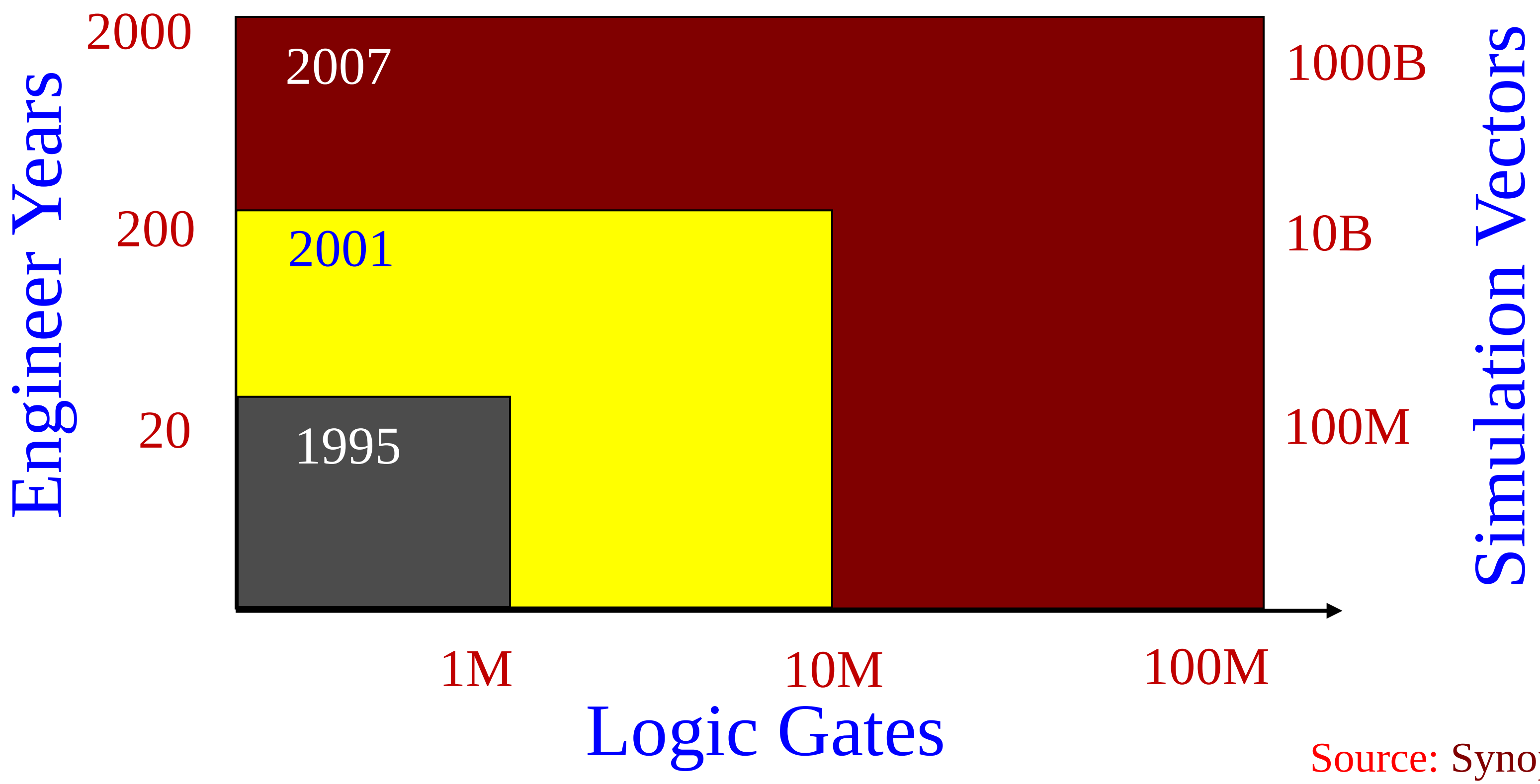
Outline

- Introduction
- Simulation-based Functional Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Learning-oriented property decomposition
  - Decision ordering based learning techniques
  - Test generation using our methodology
- Experiments
- Conclusion

Outline

- Introduction
- Simulation-based Functional Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Learning-oriented property decomposition
  - Decision ordering based learning techniques
  - Test generation using our methodology
- Experiments
- Conclusion

Functional Validation of SoC Designs



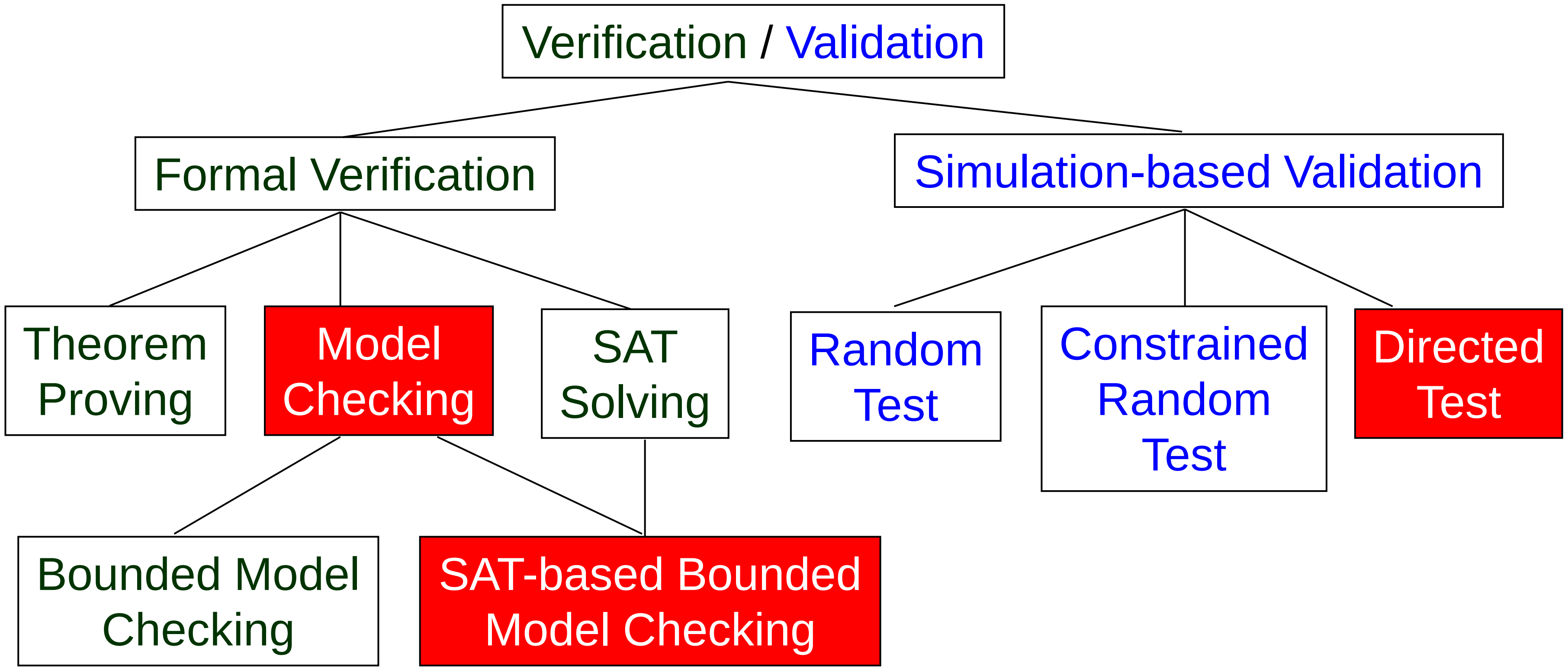
Source: Synopsys

- 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 Functional Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Learning-oriented property decomposition
  - Decision ordering based learning techniques
  - Test generation using our methodology
- Experiments
- Conclusion

Automated Directed Test Generation



Directed test generation based on the automation of model checking techniques.

Test Generation using Model Checking

- Model Checking
  - Designs are in formal specifications, e.g., SMV
  - Desired behaviors in temporal logic properties
  - Property holds, or fails with a counterexample

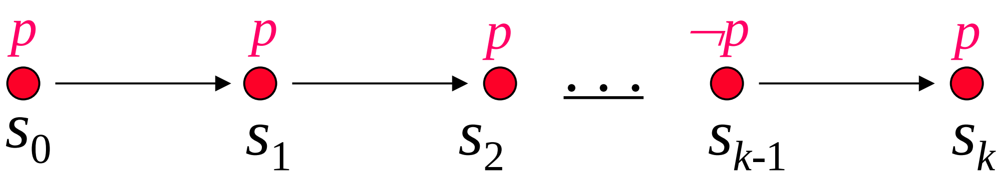
Problem: Test generation is very costly or not possible in many scenarios in the presence of complex SoCs and/or complex properties.

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

SAT-based Bound Model Checking

- **Test generation needs to consider safety properties**
- **The safety property *P* is valid up to cycle *k* iff Ω(*k*) is not satisfiable.**

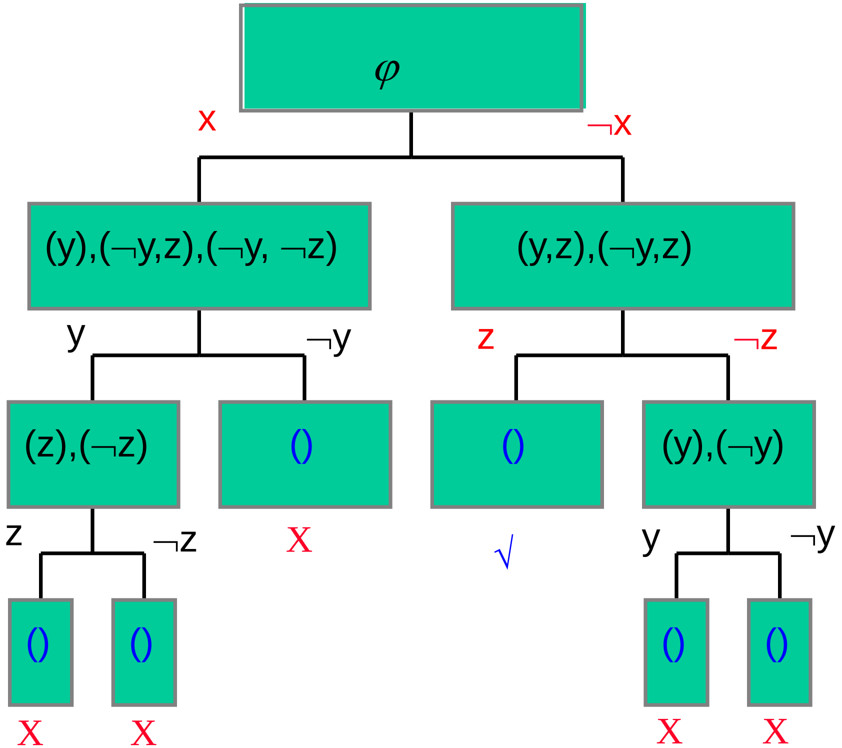
$$\Omega(k) = I(S_0) \wedge \bigwedge_{i=0}^{k-1} R(S_i, S_{i+1}) \wedge \bigvee_{i=0}^{k} \neg P(s_i)$$



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

Decision Ordering Problem

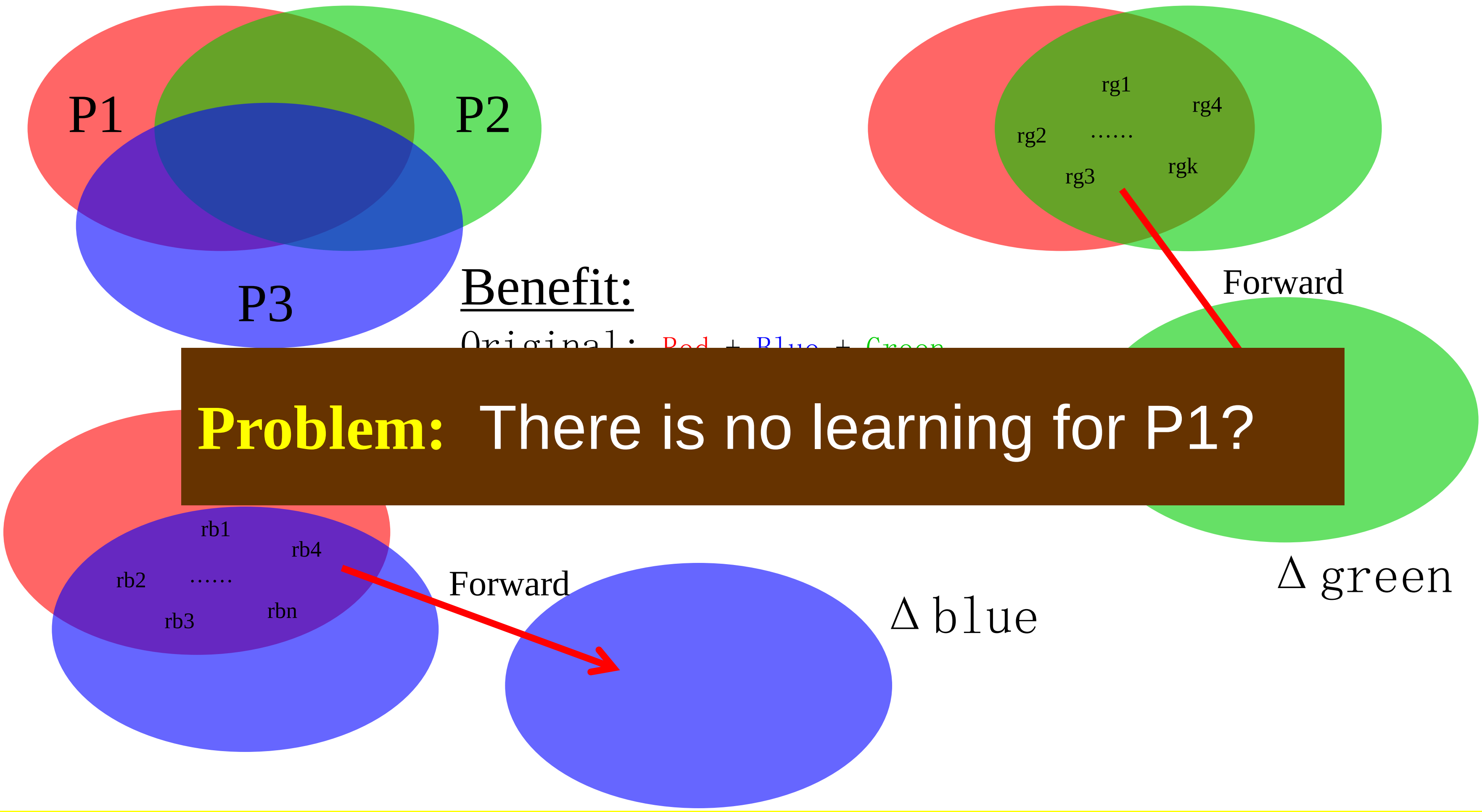
Given a φ in CNF: (x+y+z)(¬x+y)(¬y+z)(¬x+¬ y+¬ z)



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

Best decision: ¬ x, z

Same Design, Different Properties



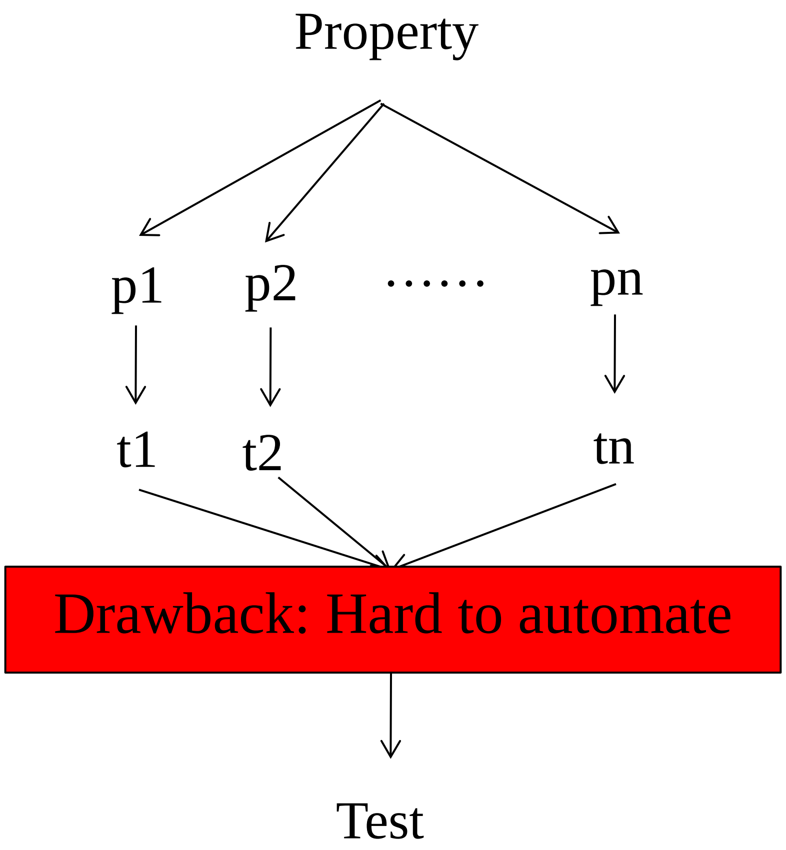
Benefit:

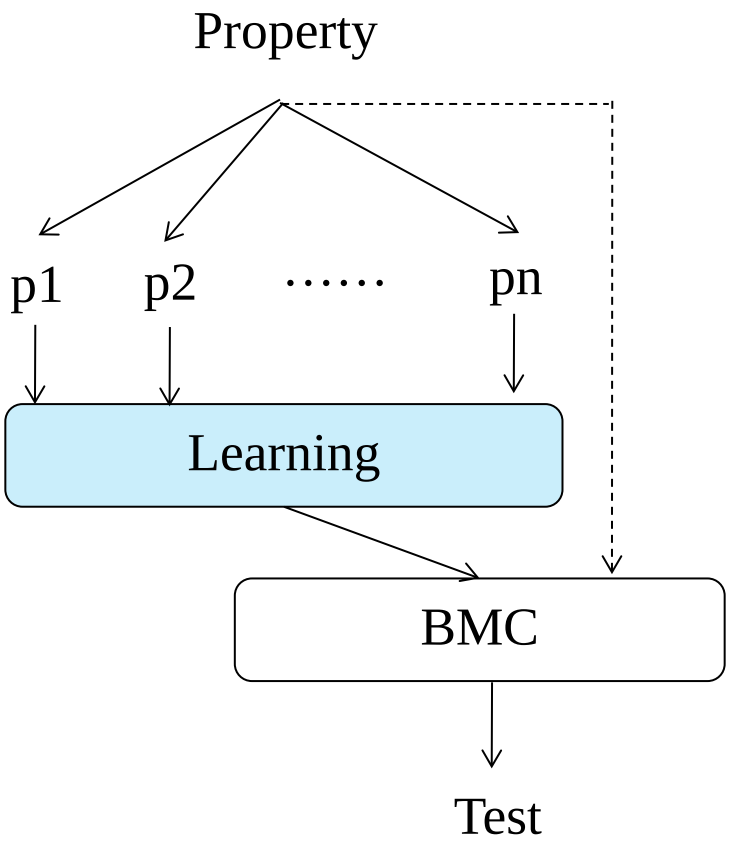
Problem: There is no learning for P1?

M. Chen and P. Mishra. **Functional Test Generation using Efficient Property Clustering and Learning Techniques.** ***TCAD 2010.***

M. Chen and P. Mishra. **Efficient Decision Ordering Techniques for SAT-based Test Generation.** ***DATE 2010.***

Property Decomposition Technique





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

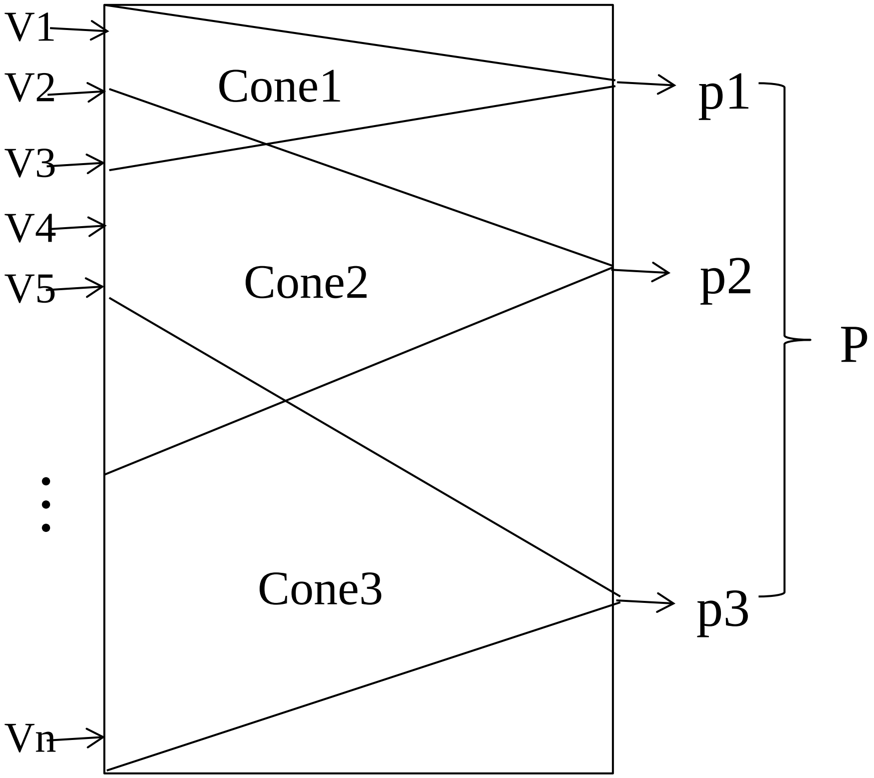
Promising Observations

- Sub-properties may have a large overlap in counter-examples (variable assignments) with original property.
  - Such important information can be reused as a kind of decision ordering.
- The learning from sub-properties can drastically reduce the overall test generation time.
  - The SAT instance for sub-properties can be much smaller than that of original property
  - The learning from sub-properties can drastically accelerate the falsification of original property.

Outline

- Introduction
- Simulation-based Functional Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Learning-oriented property decomposition
  - Decision ordering based learning techniques
  - Test generation using our methodology
- Experiments
- Conclusion

Spatial Property Decomposition



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

⇩

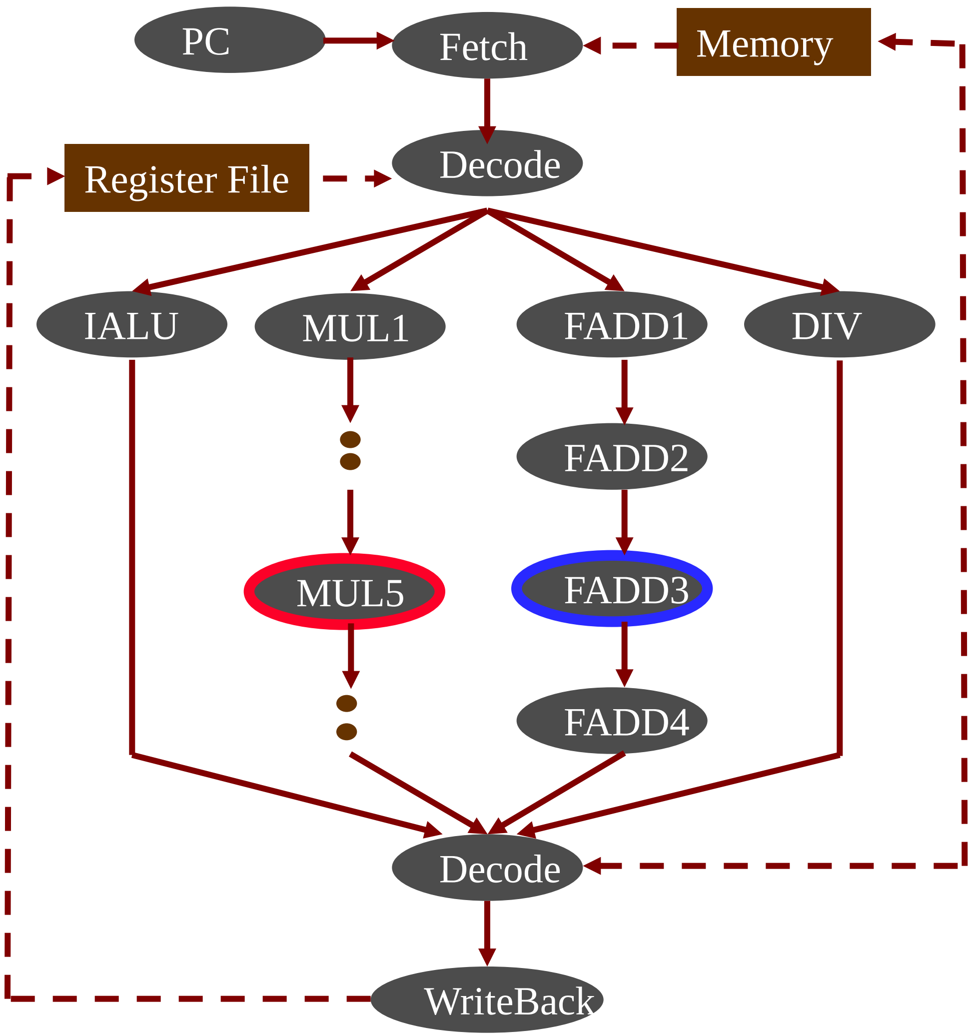
Time(p1) < Time(p2) < Time(p3) <Time(P)

⇩

Learning from P1 can reduce the Time(P) ?

Learn from the sub-properties with smaller COI.

A MIPS Processor Example

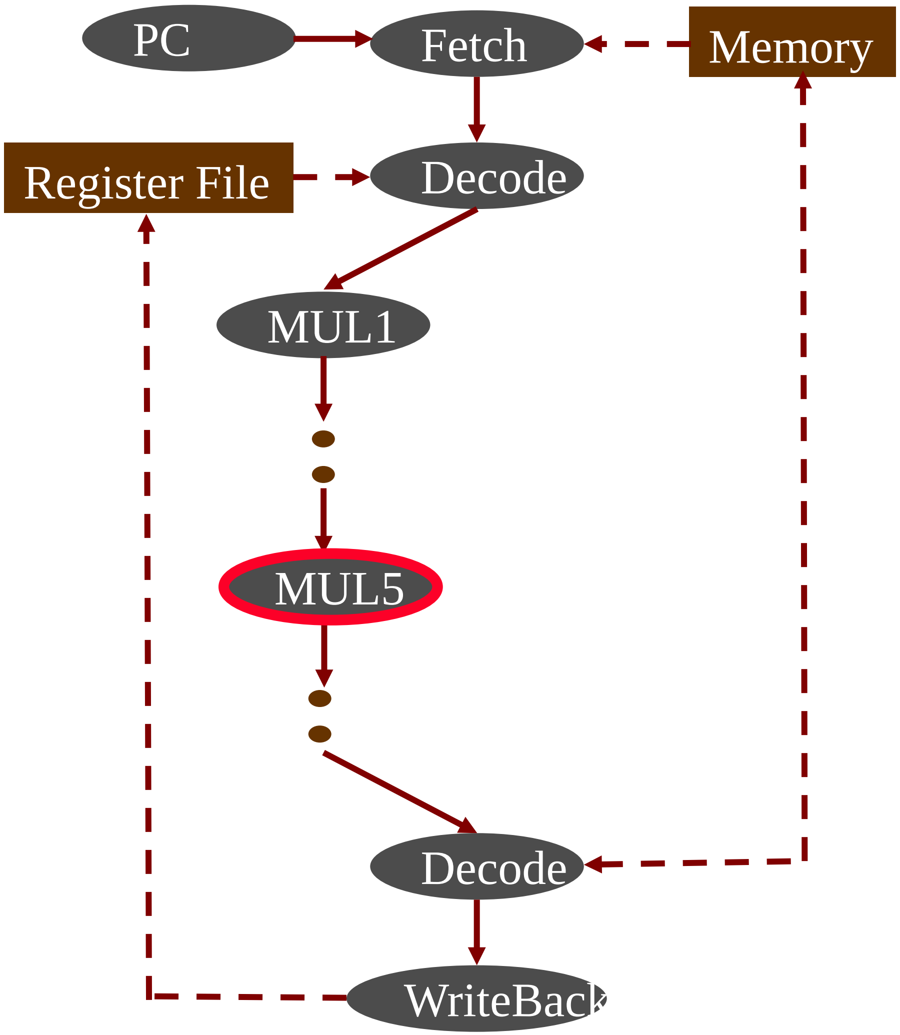


Checked Property

P: The units MUL5 and FADD3 can be activated together at 8th clock cycle.

LTL: ! F(MUL5=active & FADD3=active & clk=8)

Spatial Property Decomposition



Checked sub Property

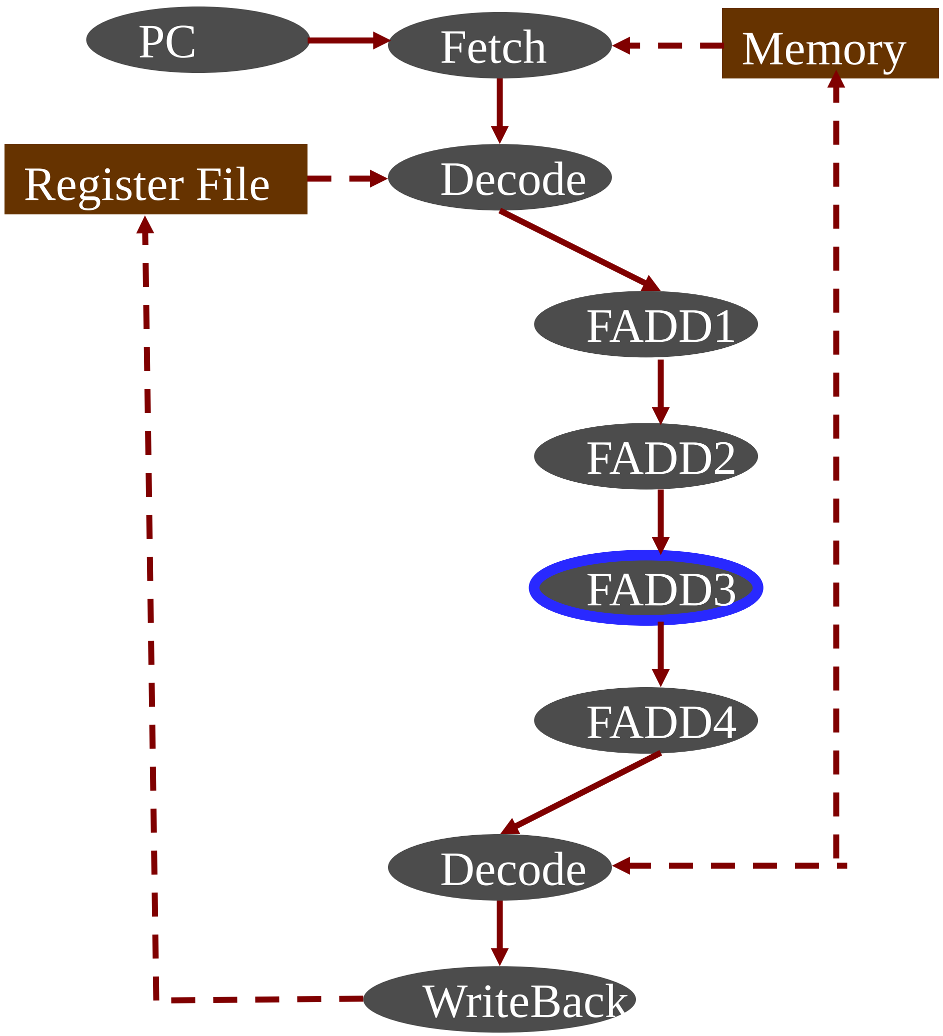
P1: The units MUL5 can be activated at 8th clock cycle.

LTL: !F(MUL5=active & clk=8)

Counterexample for P1

Cycles	P1's test
1	NOP
2	MUL R2, R2, R0
3	NOP
4	NOP

Spatial Property Decomposition



Checked sub Property

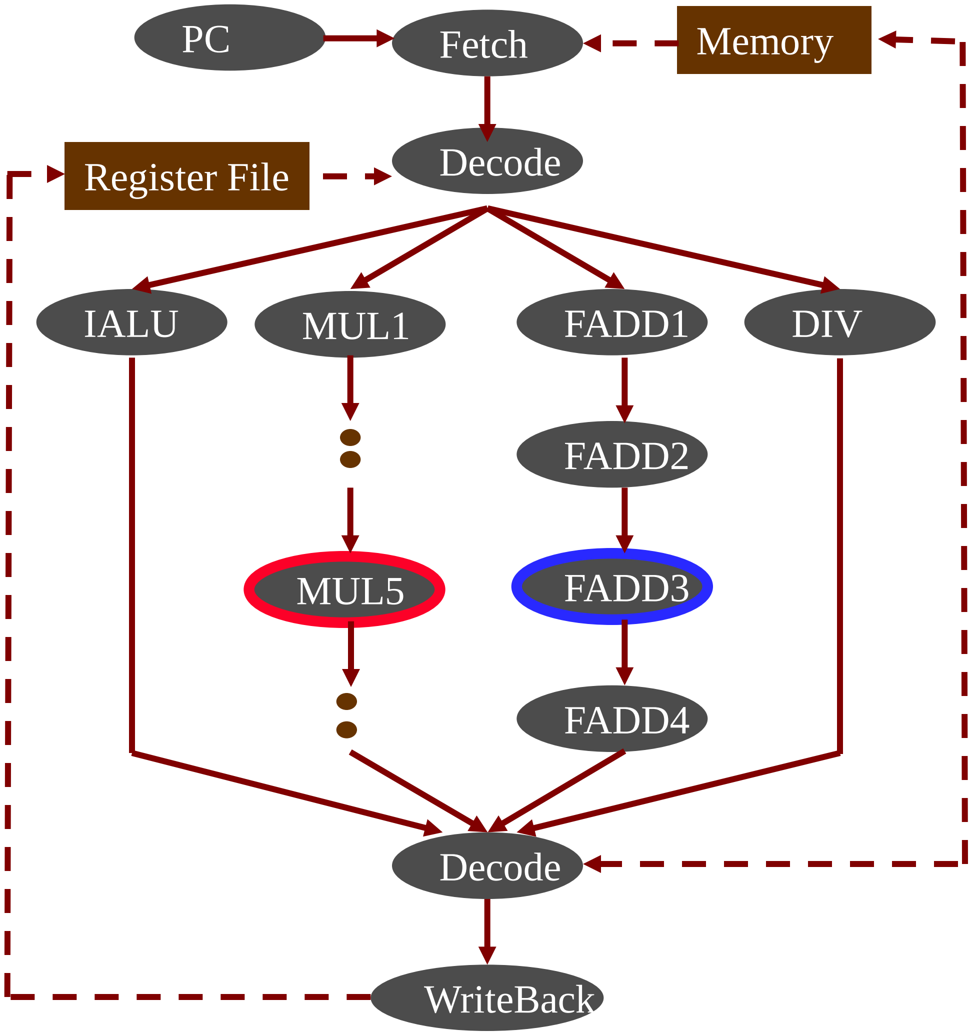
P2: The units FADD3 can be activated at 8th clock cycle.

LTL: !F(FADD3=active & clk=8)

Counterexample for P2

Cycles	P2's test
1	NOP
2	NOP
3	NOP
4	FADD R1, R1, R0

Learning from Spatial Property Decomposition



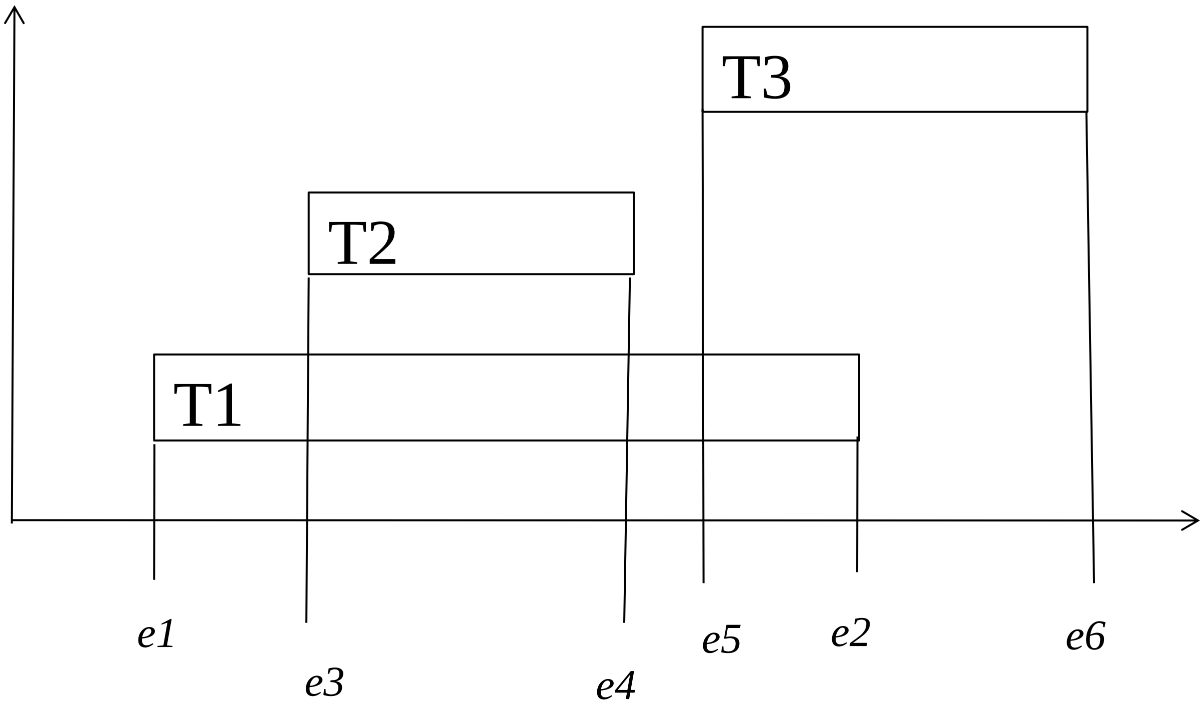
Countereample for P2 guided by P1

Cycles	Learning
1	NOP
2	MUL R2, R2, R0
3	NOP
4	FADD R1, R1, R0

Counterexample for P

Cycles	Learnings
1	NOP
2	MUL R2, R2, R0
3	NOP
4	FADD R1, R1, R0

Temporal Decomposition

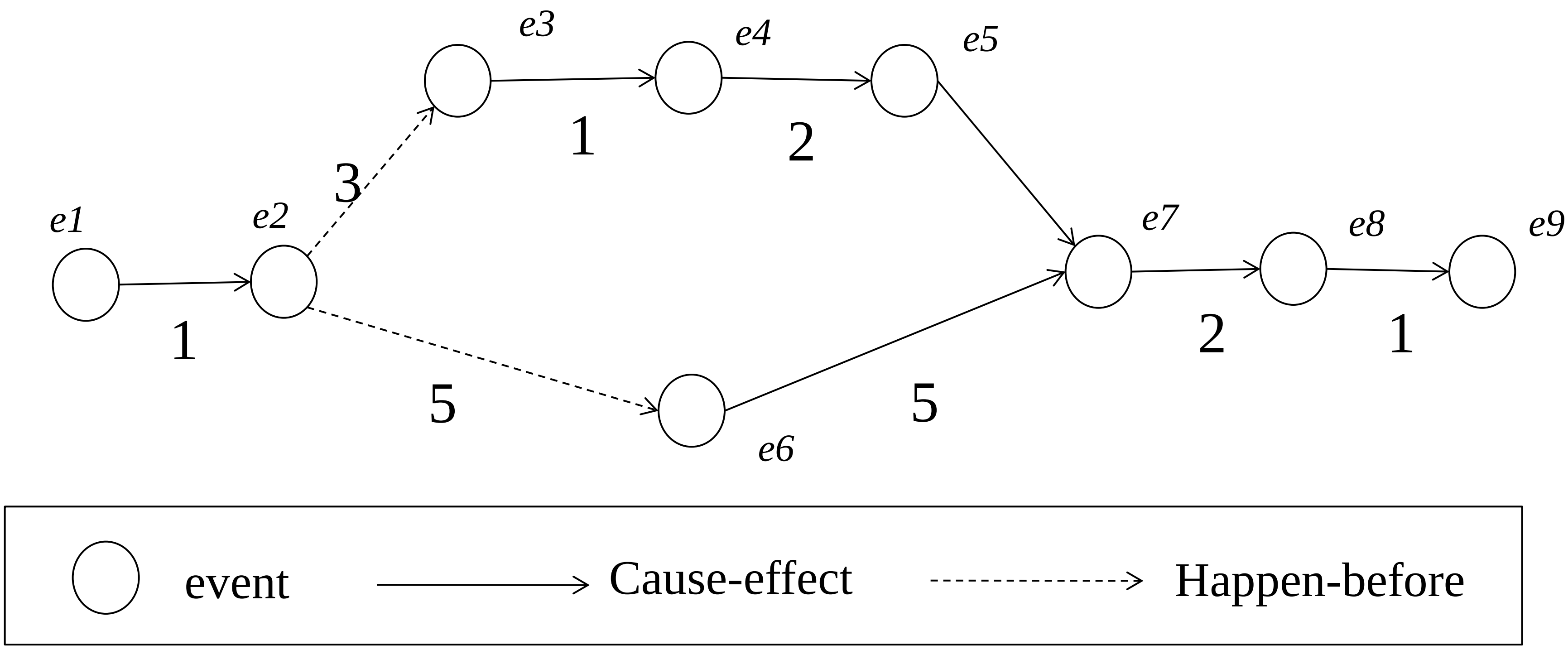


Cause effect relation: e1→e2 e3→e4 e5→e6

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

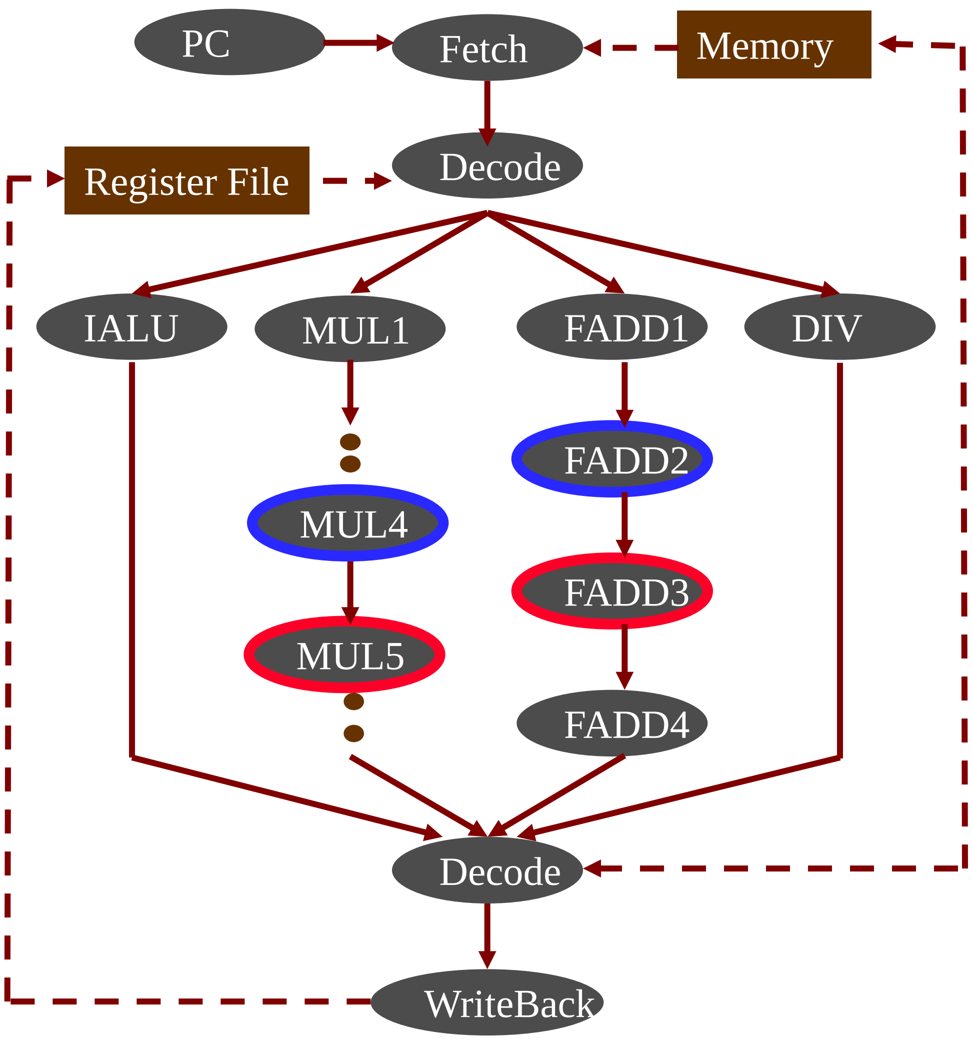
Learn from the sub-properties with smaller bound.

Event Relation Analysis



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

A MIPS Processor Example



<u>Checked Property</u>

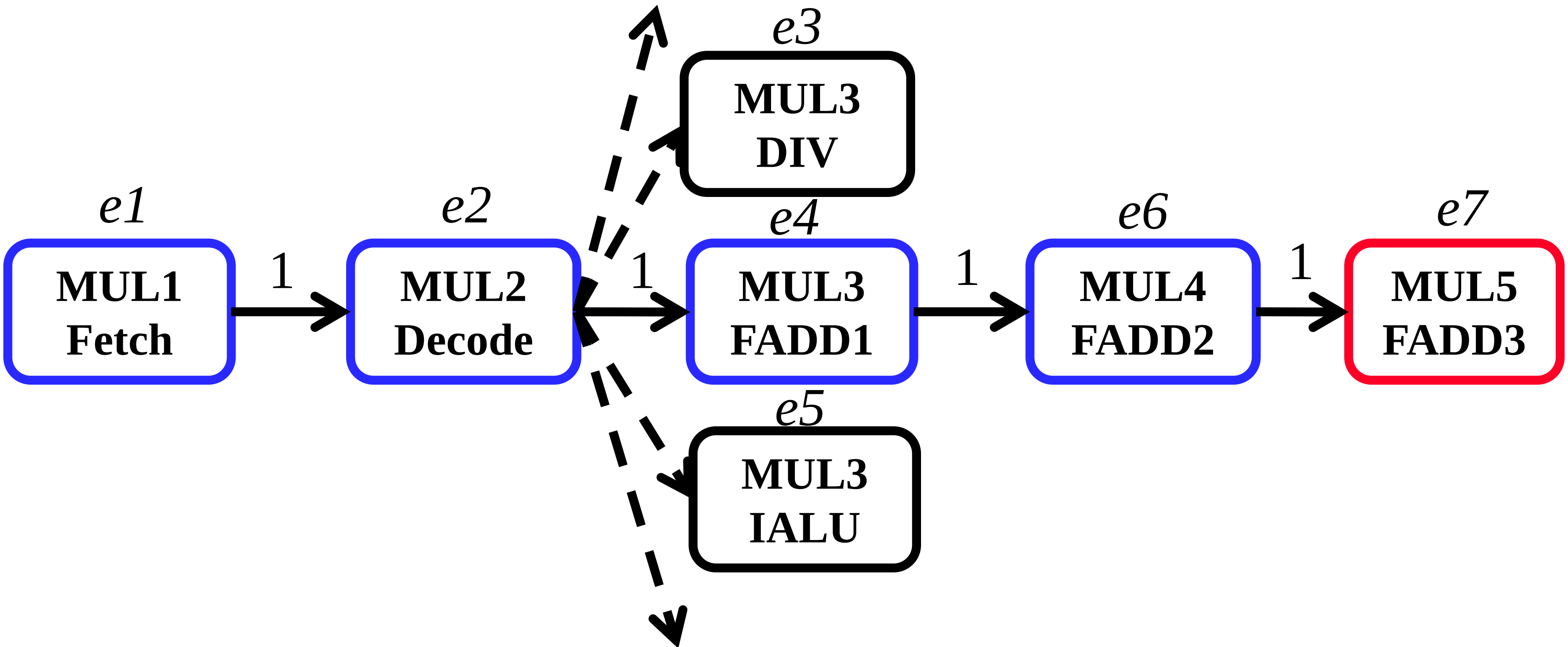
P: The units MUL5 and FADD3 can be activated together at 8th clock cycle.

LTL: ! F(MUL5=active & FADD3=active & clk=8)

<u>A sub-property example</u>

LTL: ! F(MUL4=active & FADD2=active & clk=7)

Event Relation Construction



<u>Original Property</u>

P_e7: ! F(MUL5=active & FADD3=active & clk=8)

<u>Temporally Decomposed Properties</u>

P_e1: ! F(MUL1=active & Fetch=active & clk=4)

P_e4: ! F(MUL3=active & FADD1=active & clk=6)

Decision Ordering Heuristics

- **Let *vstat[sz][2]* be a 2-dimension array to record the statistics of sub-property results. It is used to indicate the decision ordering of unchecked properties.**

- **The term *bias(vi)* is used to indicate the variable assignment variance of *vi*.**

$$bias\ (vi) = \frac{\mathrm{Max}(\ vstat[i][0],\ vstat[i][1]) + 1}{\mathrm{Min}(\ vstat[i][0],\ vstat[i][1]) + 1}$$

Decision Ordering Heuristics (cont.)

- **Our decision ordering is based on VSIDS but our method considers decision ordering learned from sub-properites.**

Initialization

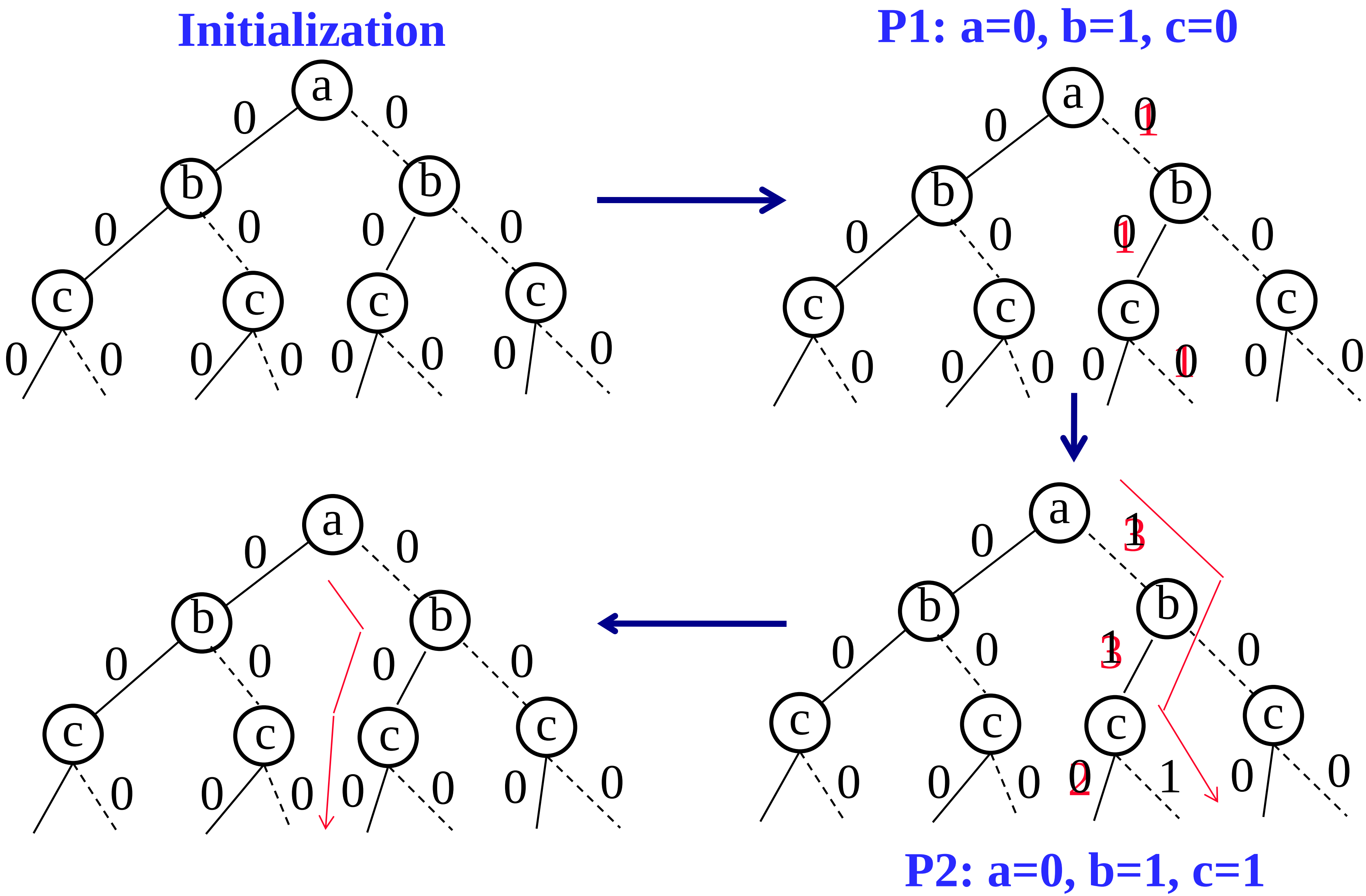
$$score(li) = \text{literal count of } li \text{ in CNF clauses}$$

Periodical update (include initialization)

$$score(li) = \begin{cases} max(vi) * bias(vi) & (\ varStat[i][1] > varStat[i][0] \ \& \ li = vi) \\ & \text{or } (\ varStat[i][0] > varStat[i][1] \ \& \ li = vi') \\ score(li) & \text{Otherwise} \end{cases}$$

where $\max(vi) = \text{MAX}(\ score(vi), score(vi')\ ) + 1.$

An Example of Learning



Test Generation Using Our Method

Inputs: a) Formal model of the Design, *D*

b) Property *P* and satisfiable bound $bound_P$

c) Decomposed properties *prop and* satisfiable *bounds*

Output: A test $test_p$ for P

1. *CNFs = BMC(D, props, bounds);*
2. *(CNF1,CNF2, …,CNFn)*=Sort CNF using increasing file size
3. Initialize *vstat*;
4. for *i* is from *1* to *n* do
   a) $test_i = SAT(CNF_i, vsat);$
   b) $Update(vstat, test_i, bounds[i]);$

   endfor
5. Generate $CNF = BMC(D, P, bound_P);$
6. return *testp = SAT(CNF, vstat);*

Outline

- Introduction
- Simulation-based Functional Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Learning-oriented property decomposition
  - Decision ordering based learning techniques
  - Test generation using our methodology
- Experiments
- Conclusion

Case Study 1: MIPS Processor

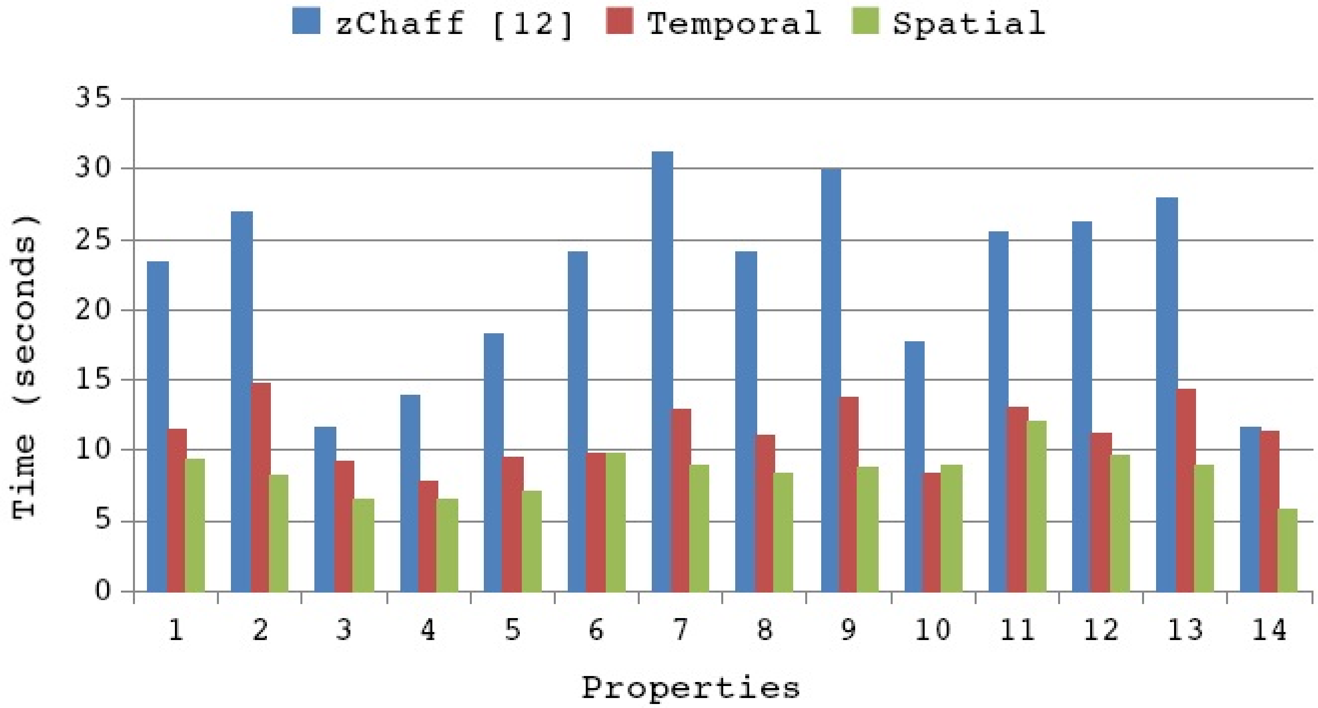
- **We generated 20 properties based on interaction faults onvarious function unit (ALU, DIV, FADD and MUL). 6 of them cannot handled by temporal decomposition.**

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 1: MIPS Processor

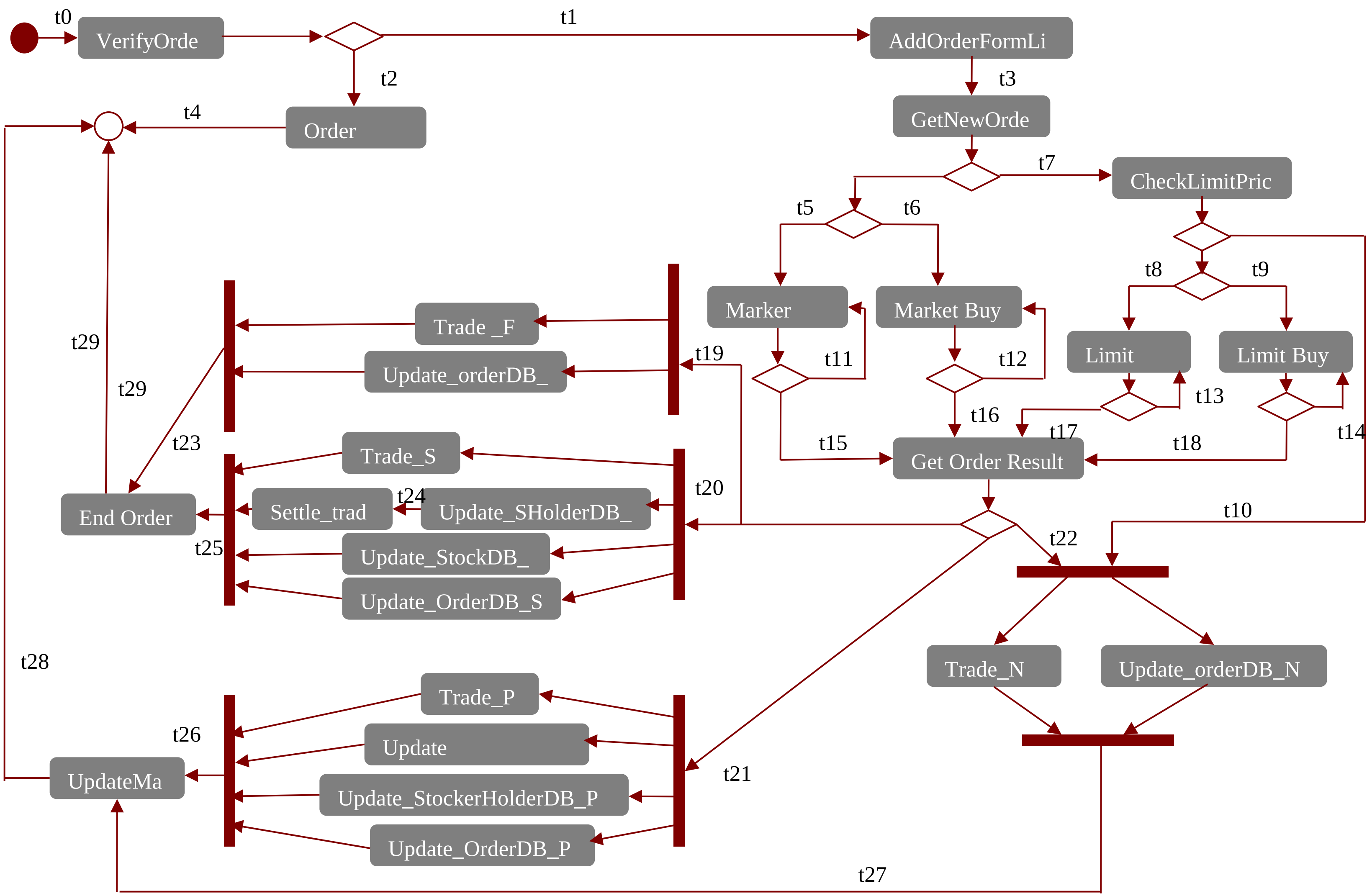
- **For the remaining 14 properties, we adopts both spatial and temporal decompositions.**



Indications: Test generation complexity is significantly improved

- Spatial decomposition is better in this example
- Temporal decomposition can still get 2.5X speedup

Case Study 2 : OSES



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: 3-63 times

Outline

- Introduction
- Simulation-based Functional Validation
  - Test Generation using Model Checking
  - Test Generation using SAT-based BMC
- Test Generation using Decision Ordering
  - Learning-oriented property decomposition
  - Decision ordering based learning techniques
  - Test generation using our methodology
- Experiments
- Conclusion

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 two novel property decomposition techniques based on decision ordering learning.
- **Successfully applied on both hardware and software designs**
  - Significant reduction in overall validation effort

Thank you !