Efficient Self-Learning Techniques for SAT-Based Test Generation

Ang Li and Mingsong Chen

Shanghai Key Lab of Trustworthy Computing

East China Normal University, China

October 9, 2012

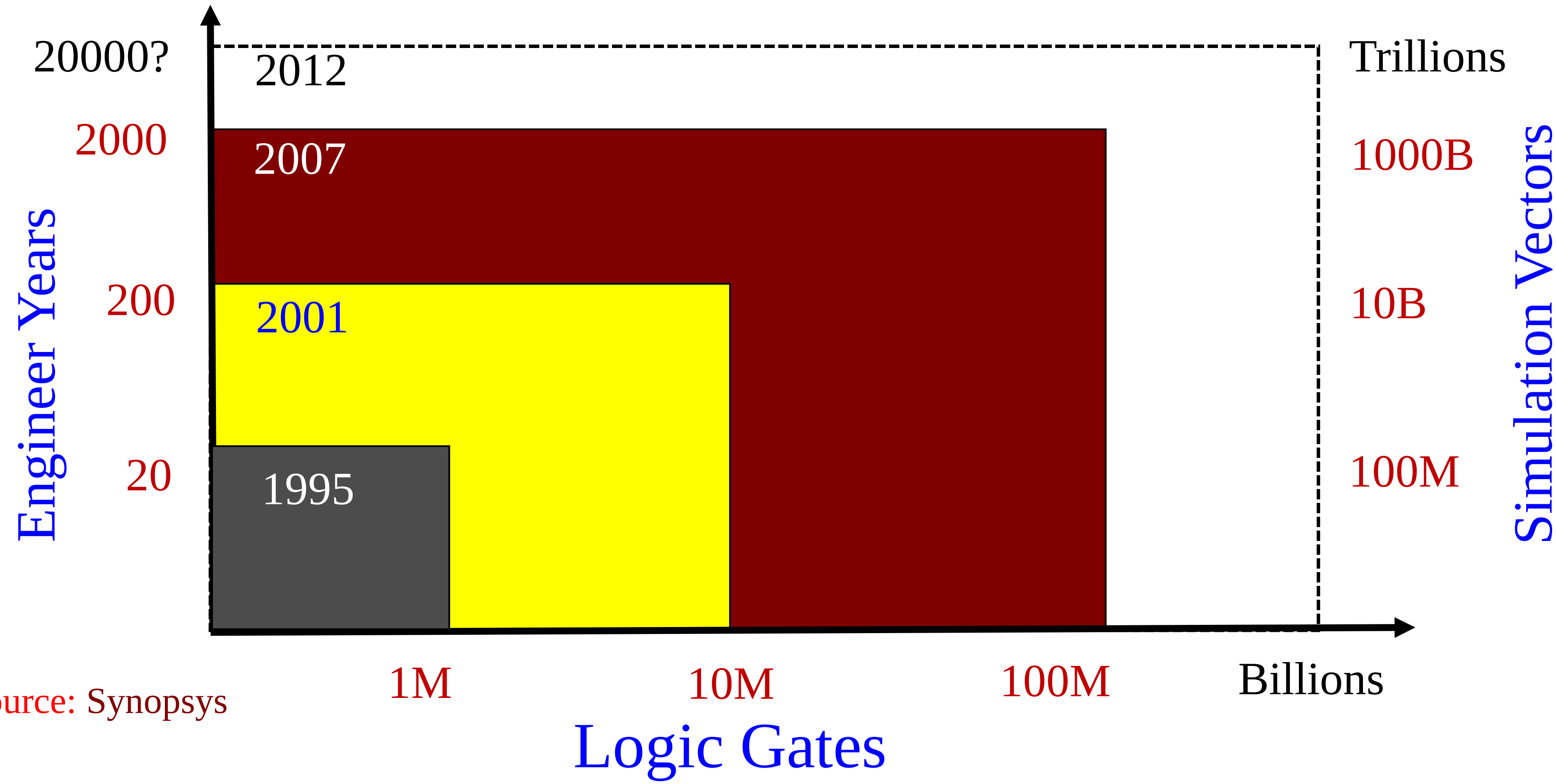
Outline

- Introduction
- SAT-Based Directed Test Generation
  - Test Generation using Model Checking
  - Test Generation using SAT-Based BMC
- Our Self-Learning Techniques
  - Motivation and Overview
  - Learning Oriented Partitioning Heuristics
  - Self-Learning Based Test Generation
- Experiments
- Conclusions

Outline

- Introduction
- SAT-Based Directed Test Generation
  - Test Generation using Model Checking
  - Test Generation using SAT-Based BMC
- Our Self-Learning Techniques
  - Motivation and Overview
  - Learning Oriented Partitioning Heuristics
  - Self-Learning Based Test Generation
- Experiments
- Conclusions

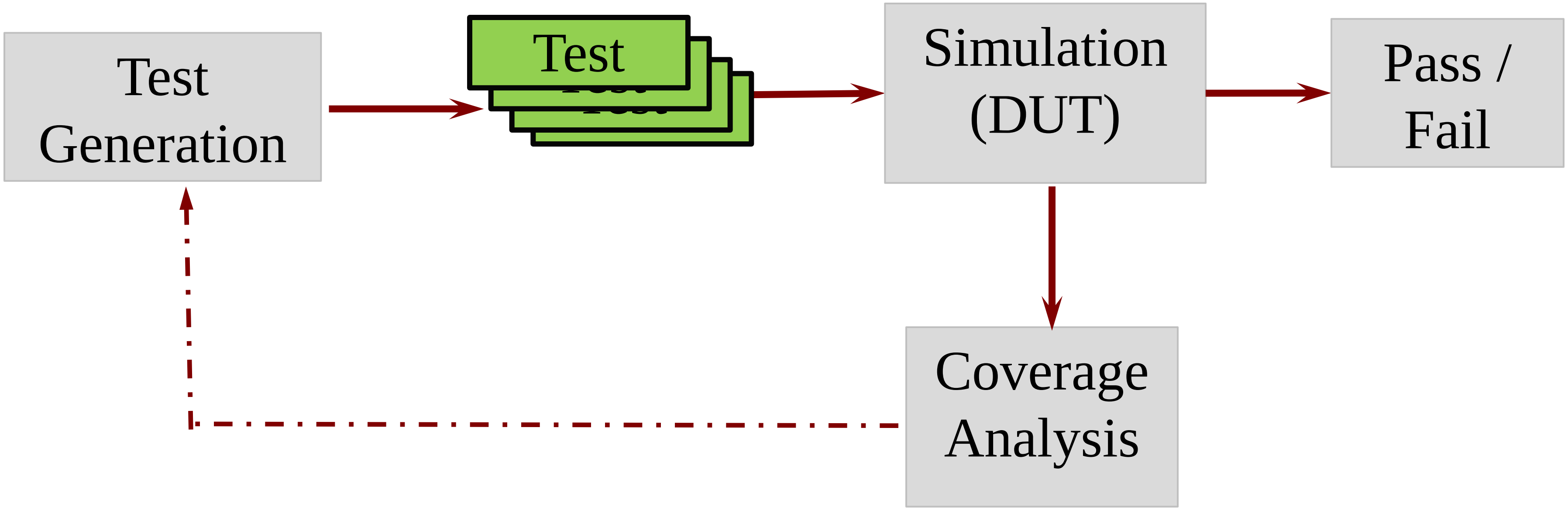
Functional Validation of SoC Designs



Source: Synopsys

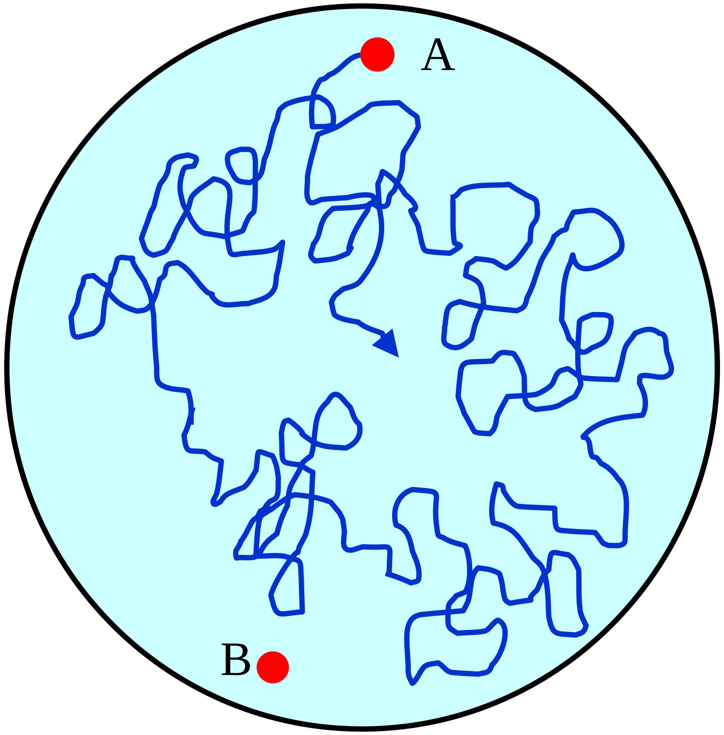
- SoC Validation is a major bottleneck
  - Up to 70% time and resources are used!

Simulation-based Validation

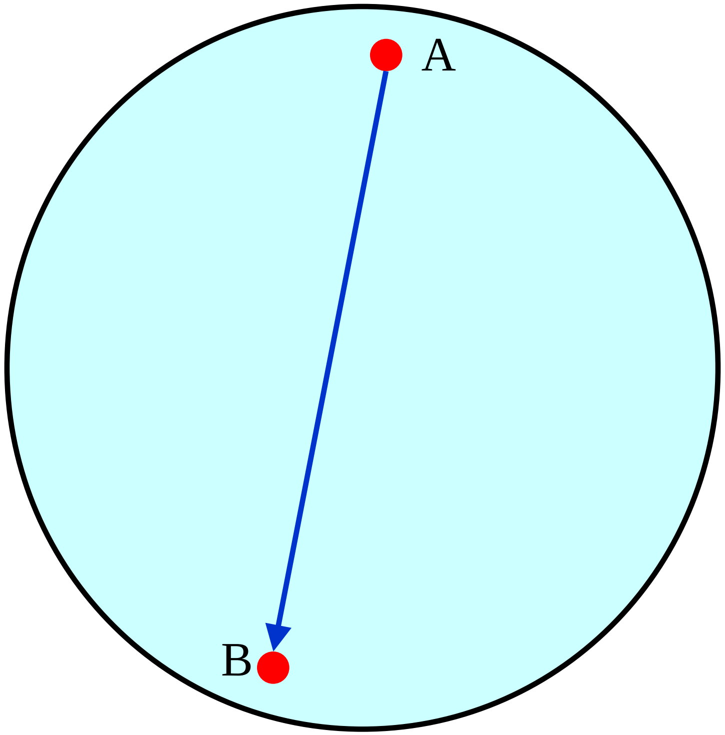


Why Directed Test Generation

- **Activate desired behavior with efficient tests**



Random Test



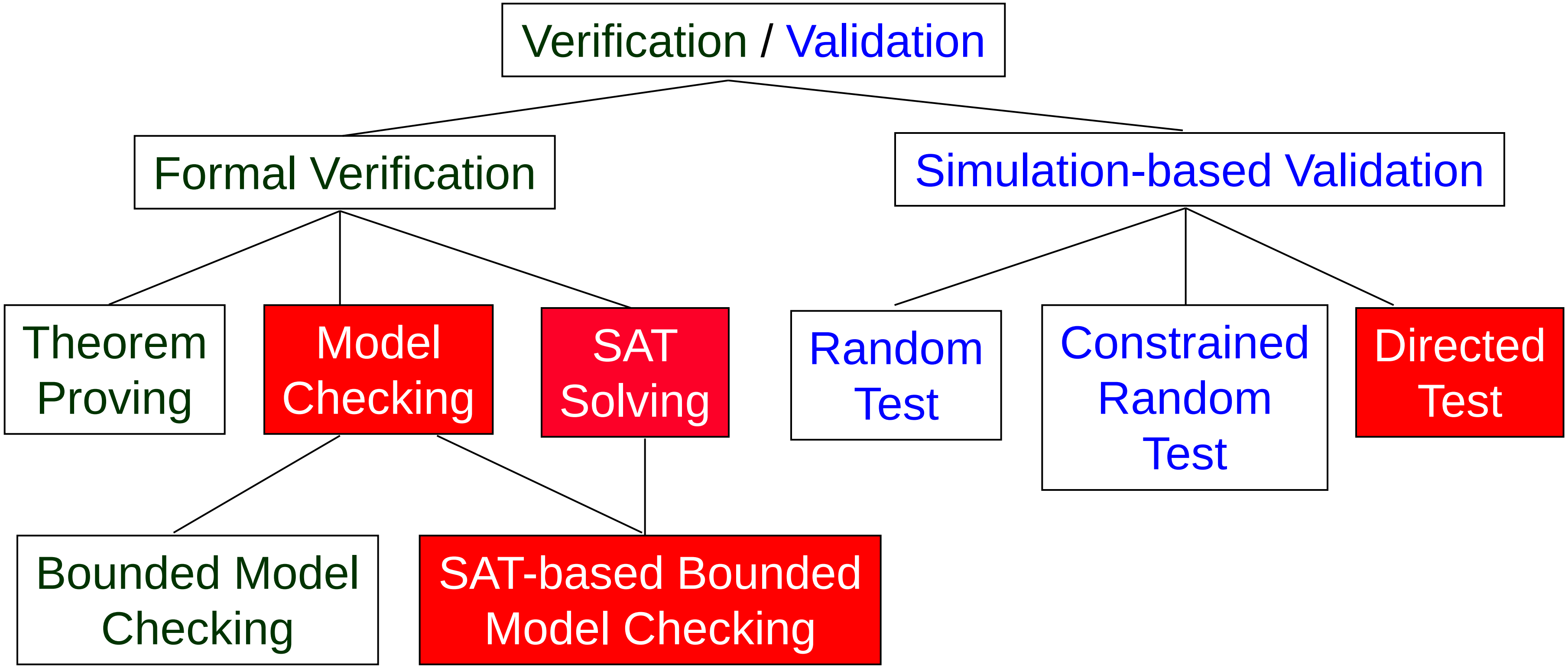
Directed Test

- **Need for automated generation of directed tests**

Outline

- Introduction
- SAT-Based Directed Test Generation
  - Test Generation using Model Checking
  - Test Generation using SAT-Based BMC
- Our Self-Learning Techniques
  - Motivation and Overview
  - Learning Oriented Partitioning Heuristics
  - Self-Learning Based Test Generation
- Experiments
- Conclusions

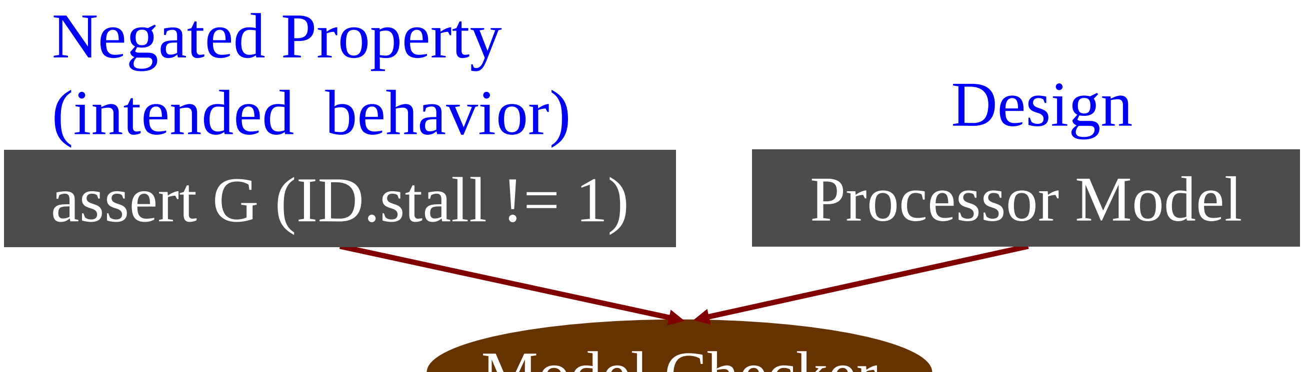
Automated Directed Test Generation



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

Test Generation using Model Checking

Example: Generate a directed test to stall a decode unit (ID)



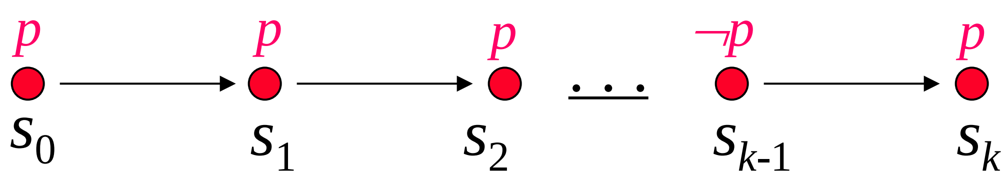
Problem: Test generation is very costly or not possible in many scenarios in the presence of complex SoCs and/or complex properties.

Approach: Exploit some learning to reduce complexity
- Reduce TG time & memory requirements
- Enable test generation in complex scenarios

SAT-based Bounded 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.**

DPLL Search Procedure

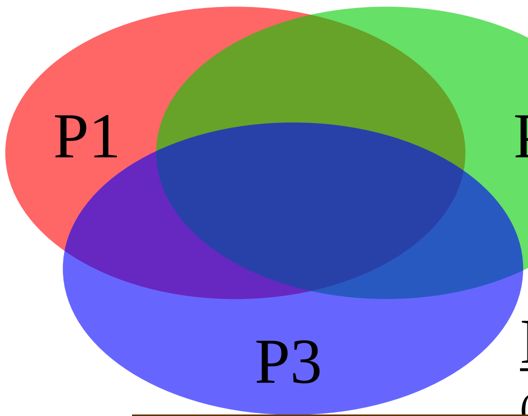
```
while (1){
    run_periodic_function();
    if( Decision ){
        while ( Deduction == CONFLICT) {
                blevel = Diagnosis ();
                if( blevel<0 )
                    return UNSAT;
        }
    } else return SAT;
}
```

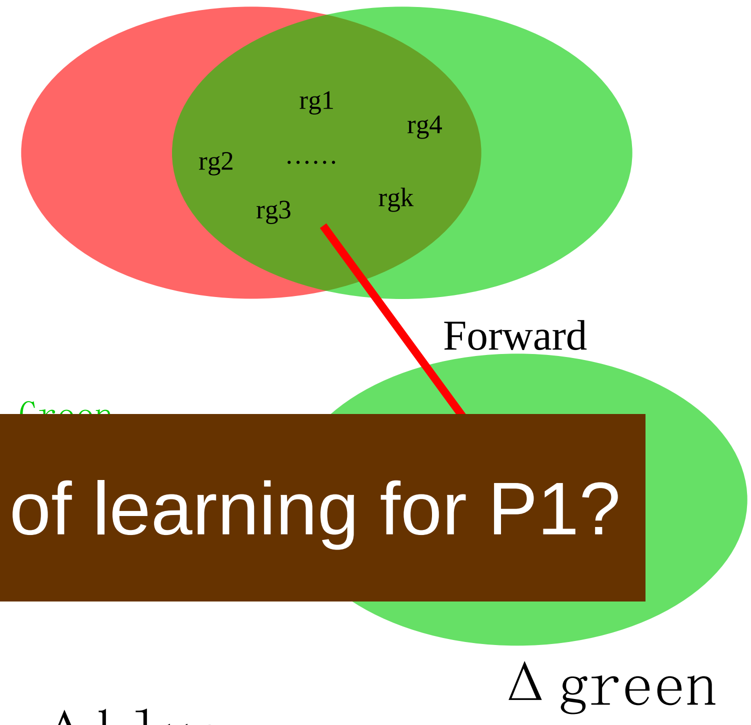
SAT search time can be improved if we can reduce:
- the number of bad decisions at early stage, and
- the number of long distance backtracks.

Outline

- Introduction
- SAT-Based Directed Test Generation
  - Test Generation using Model Checking
  - Test Generation using SAT-Based BMC
- **Our Self-Learning Techniques**
  - **Motivation and Overview**
  - **Learning Oriented Partitioning Heuristics**
  - **Self-Learning Based Test Generation**
- Experiments
- Conclusions

Property Learning Techniques

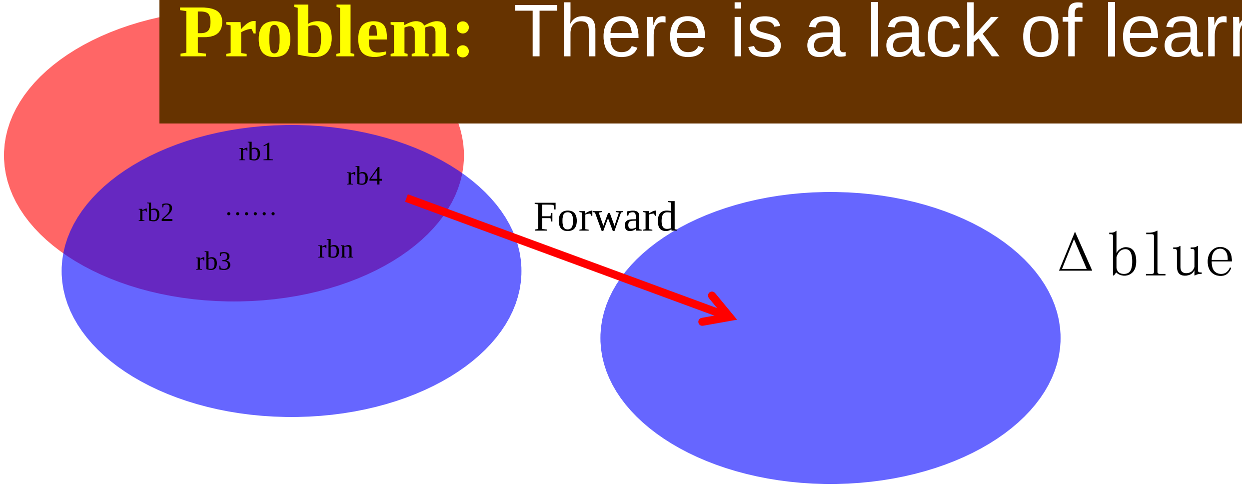




<u>Benefit:</u>

Original: Red + Blue + Green

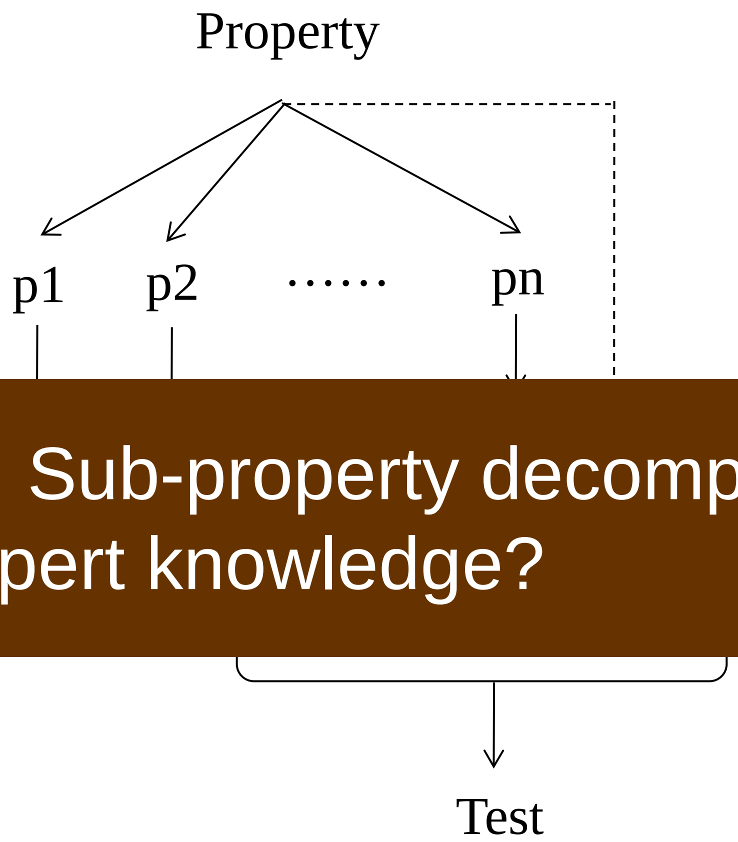
Problem: There is a lack of learning for P1?



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 Techniques

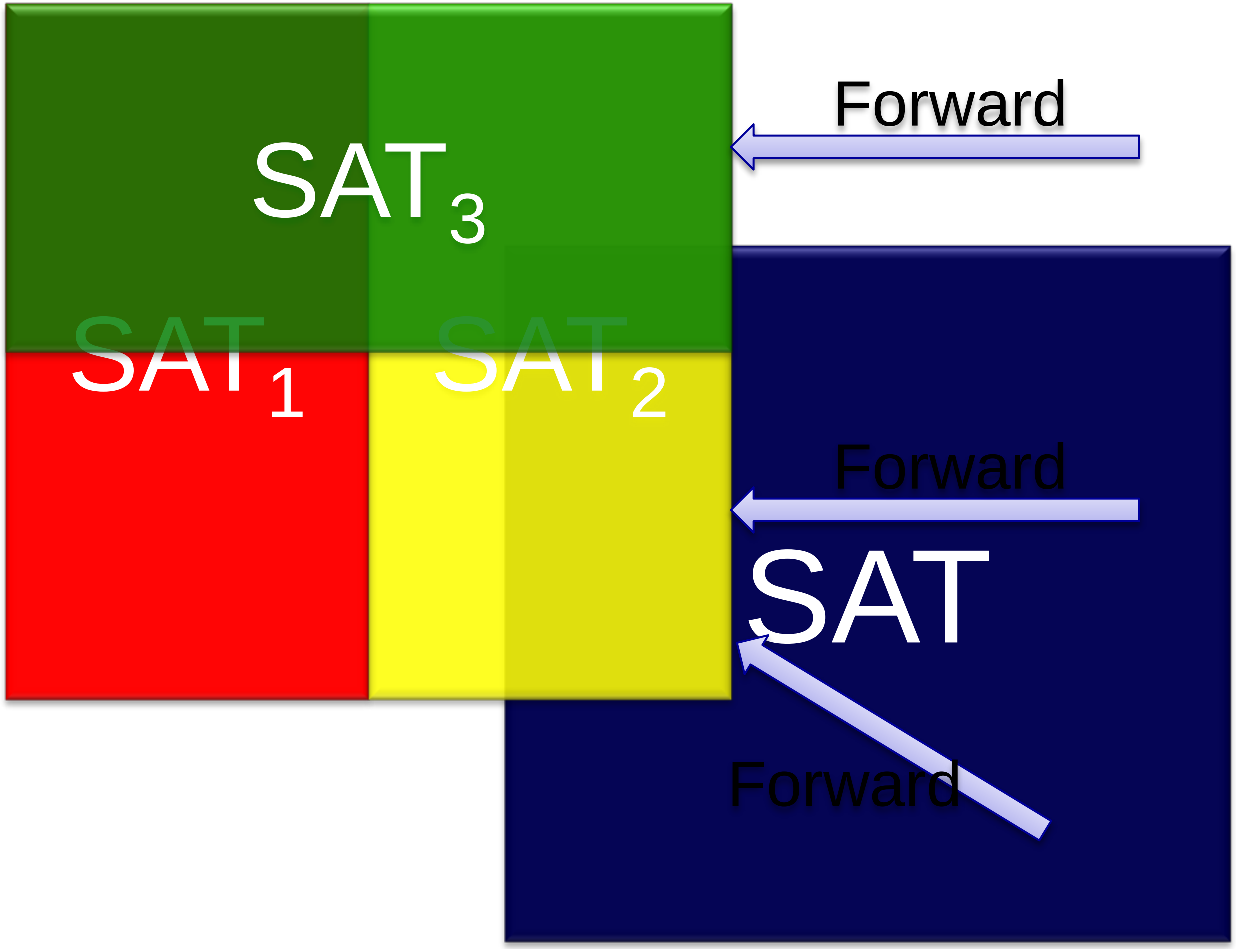


Problem: Sub-property decomposition needs expert knowledge?

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

M. Chen and P. Mishra. Decision ordering based property decomposition for functional test generation. *DATE*, 2011

Key Observations



Problems: How to achieve the beneficial learning efficiently?

- Which kind of learning can be forwarded?
- How can we achieve and utilize such kinds of learnings?

Conflict Clause Based Learning

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

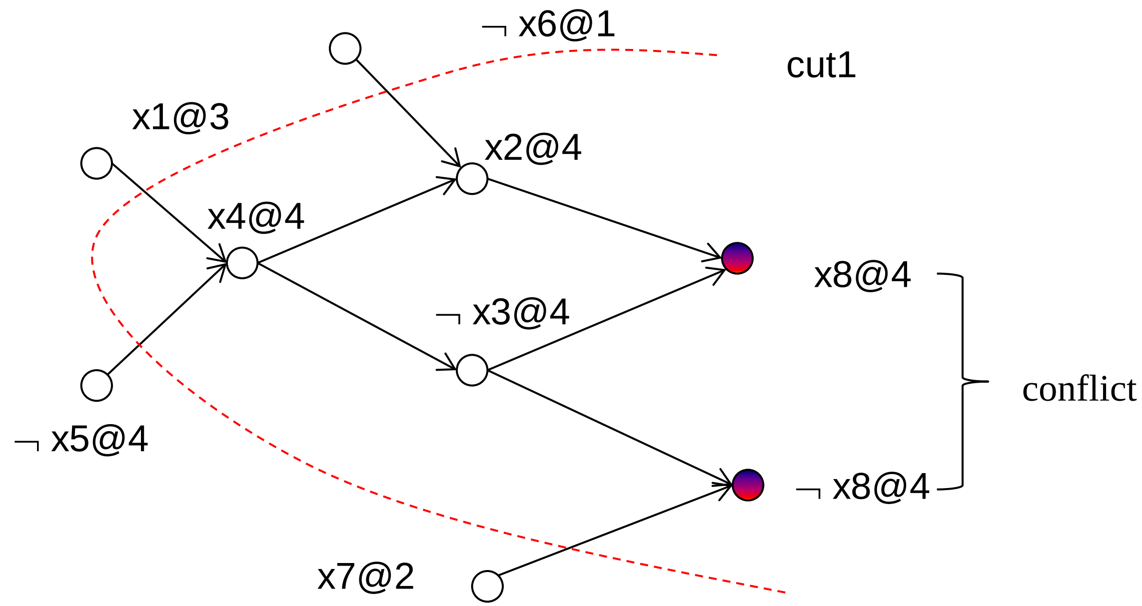
$\omega_2 = (\neg x_8 \vee x_3 \vee \neg x_7)$

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

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

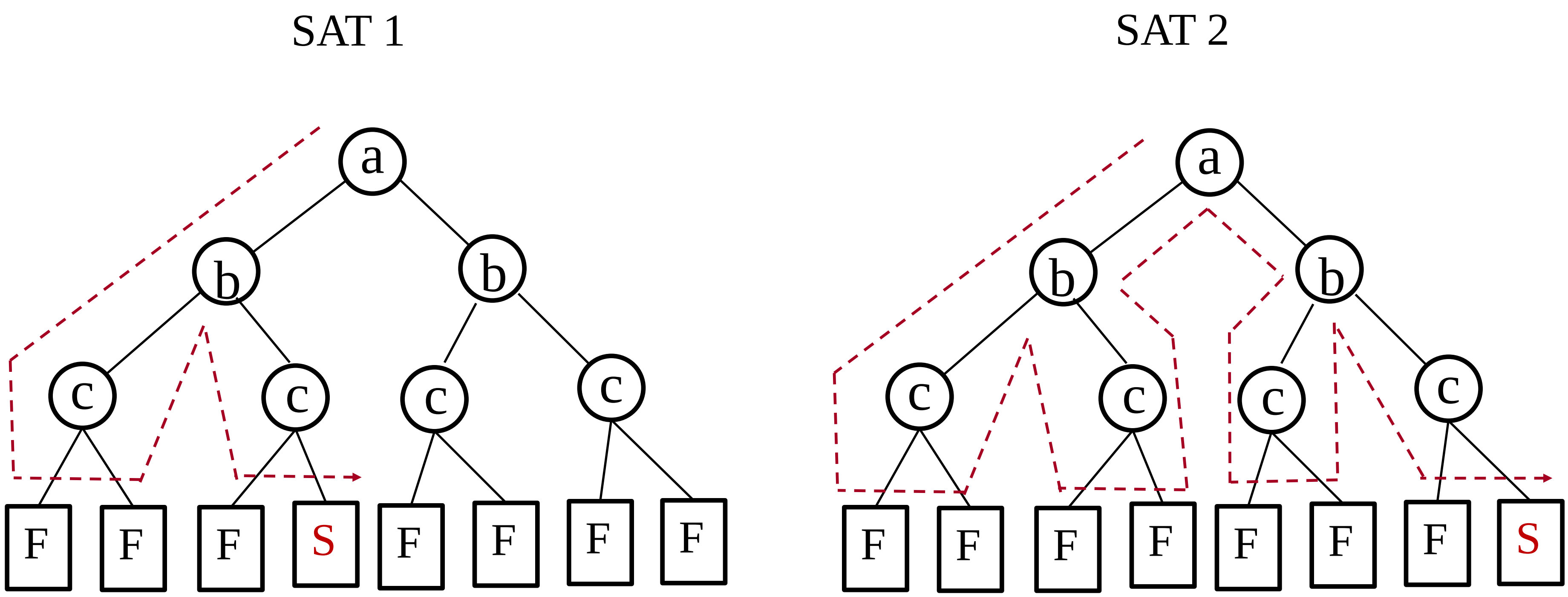
$\omega_5 = (\neg x_2 \vee x_3 \vee x_8)$

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



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

Decision Ordering Based Learning



Ordering: a, a’, b, b’, c, c’

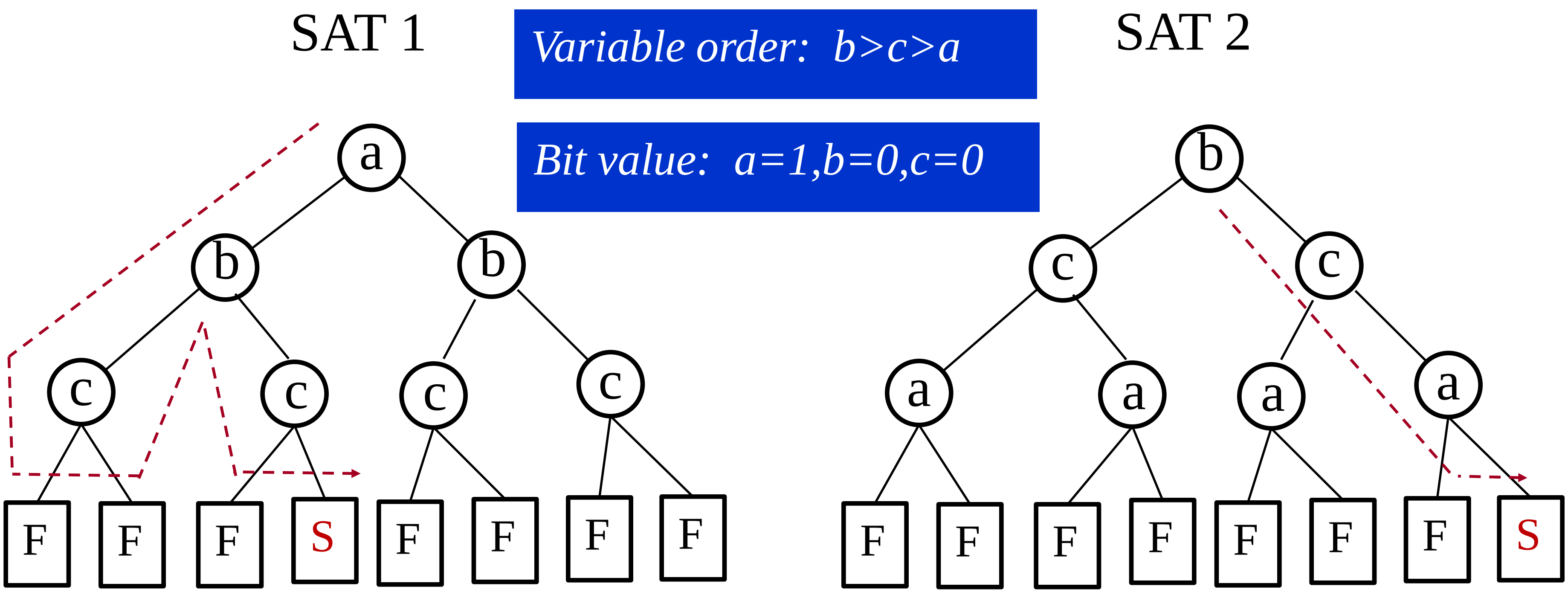
Ordering: a, a’, b, b’, c, c’

Without Learning, 7 conflicts in SAT2.

Decision Ordering Based Learning

Variable order: b>c>a

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



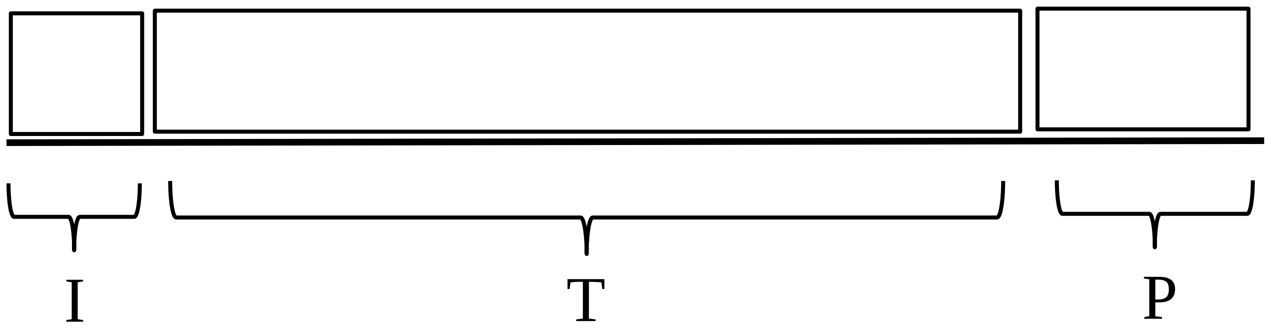
Ordering: a, a’, b, b’, c, c’

Ordering: b’, b, c’, c, a, a’

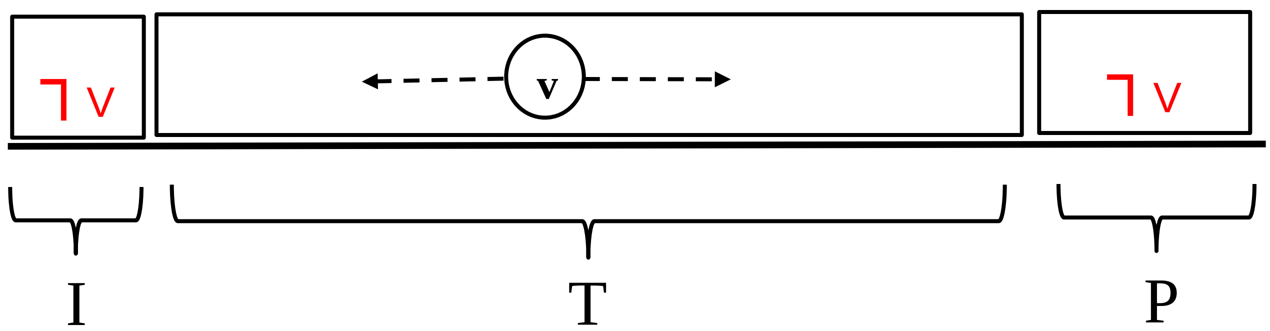
With bit value+ variable order learning, 1 conflict in SAT2.

Learn from the Structure of BMC Formulas

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



Learn from the Structure of BMC Formulas



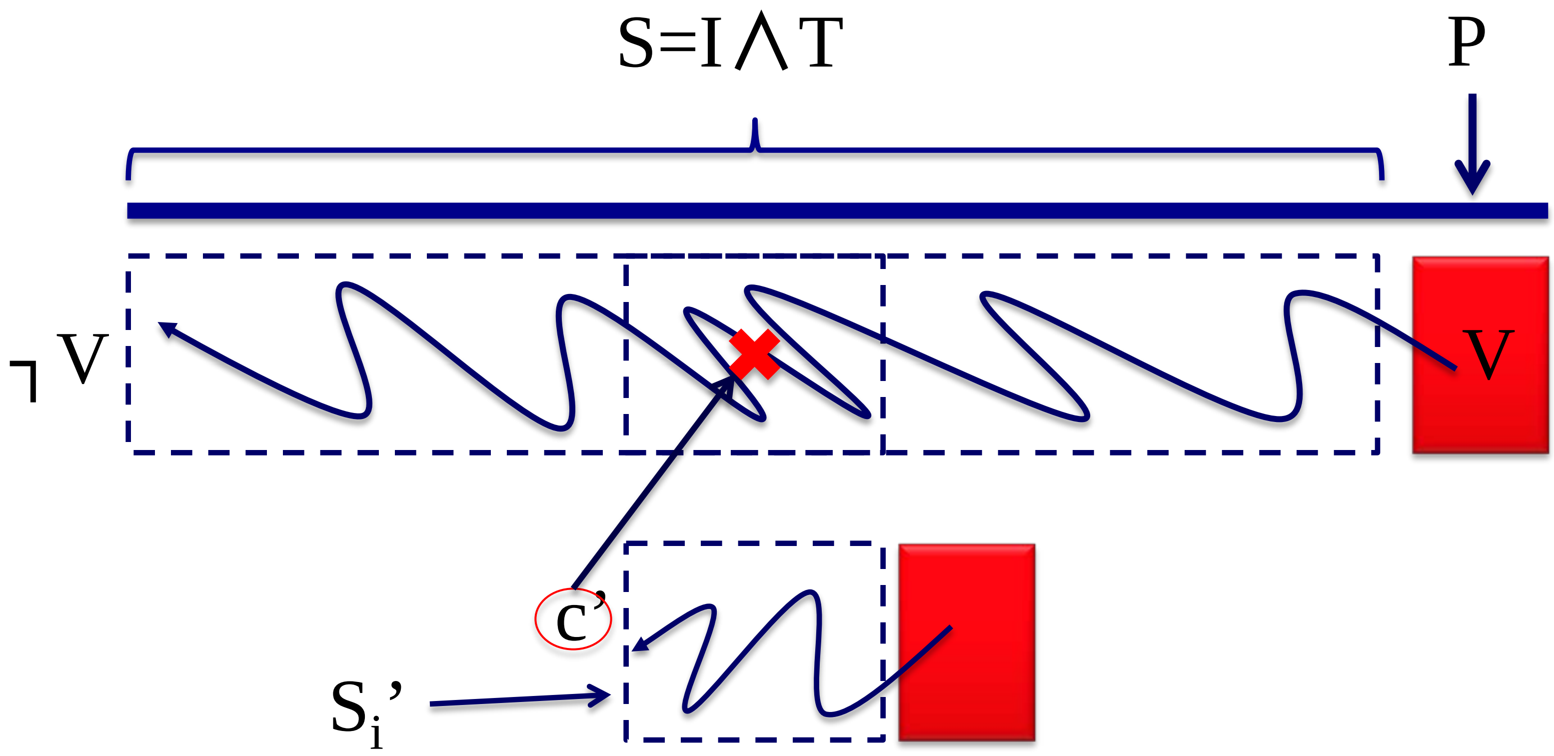
a) A deduction for a transition variable

Reasons for long-distance backtracks:

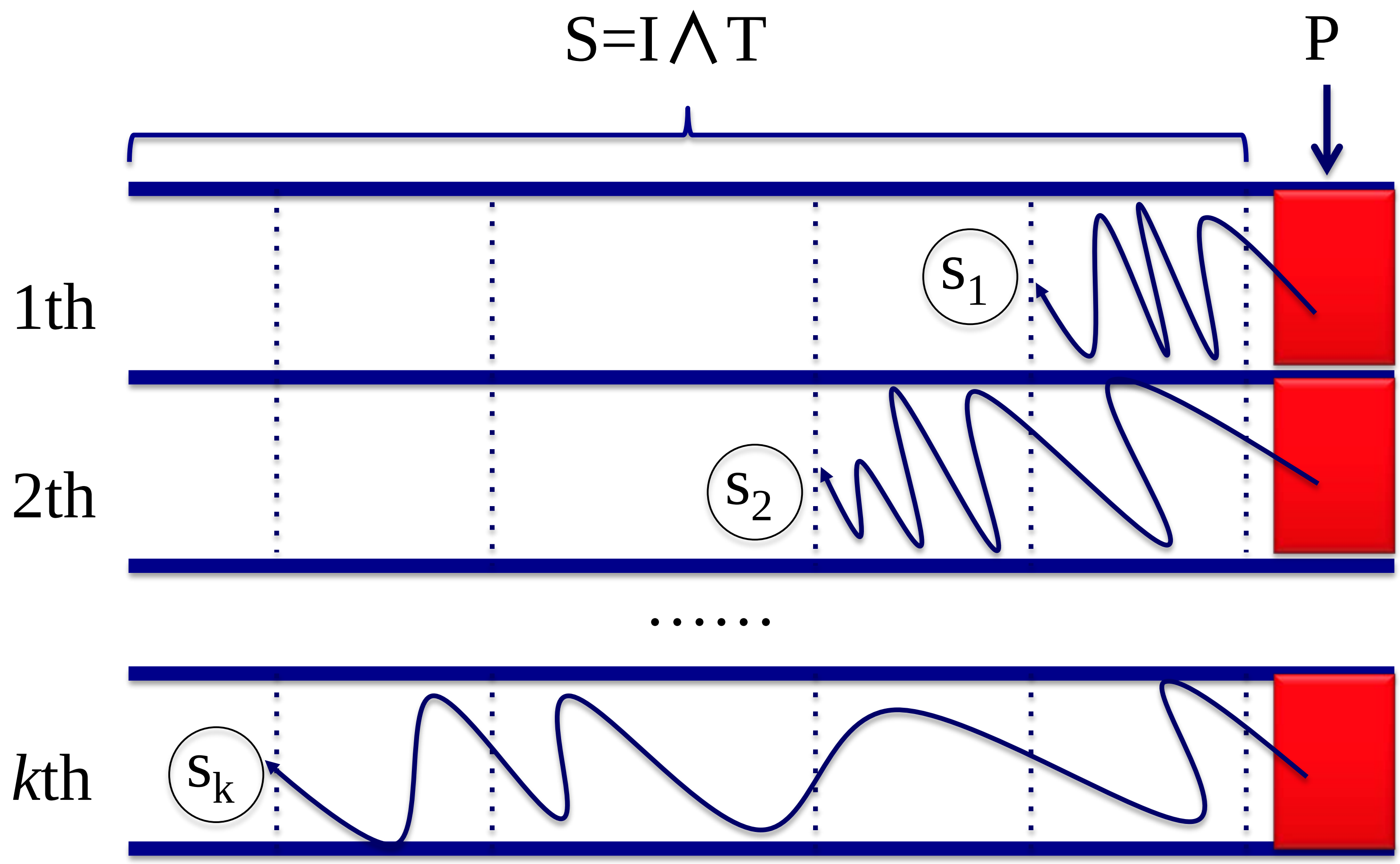
- The constraint imposed by the transition part T is weak.
- A bad decision is made too early but detected too late.

By our observation, conflict clause and decision ordering are two promising self-learning candidates to address the above two problems.

Learning from Segmental Partitioning



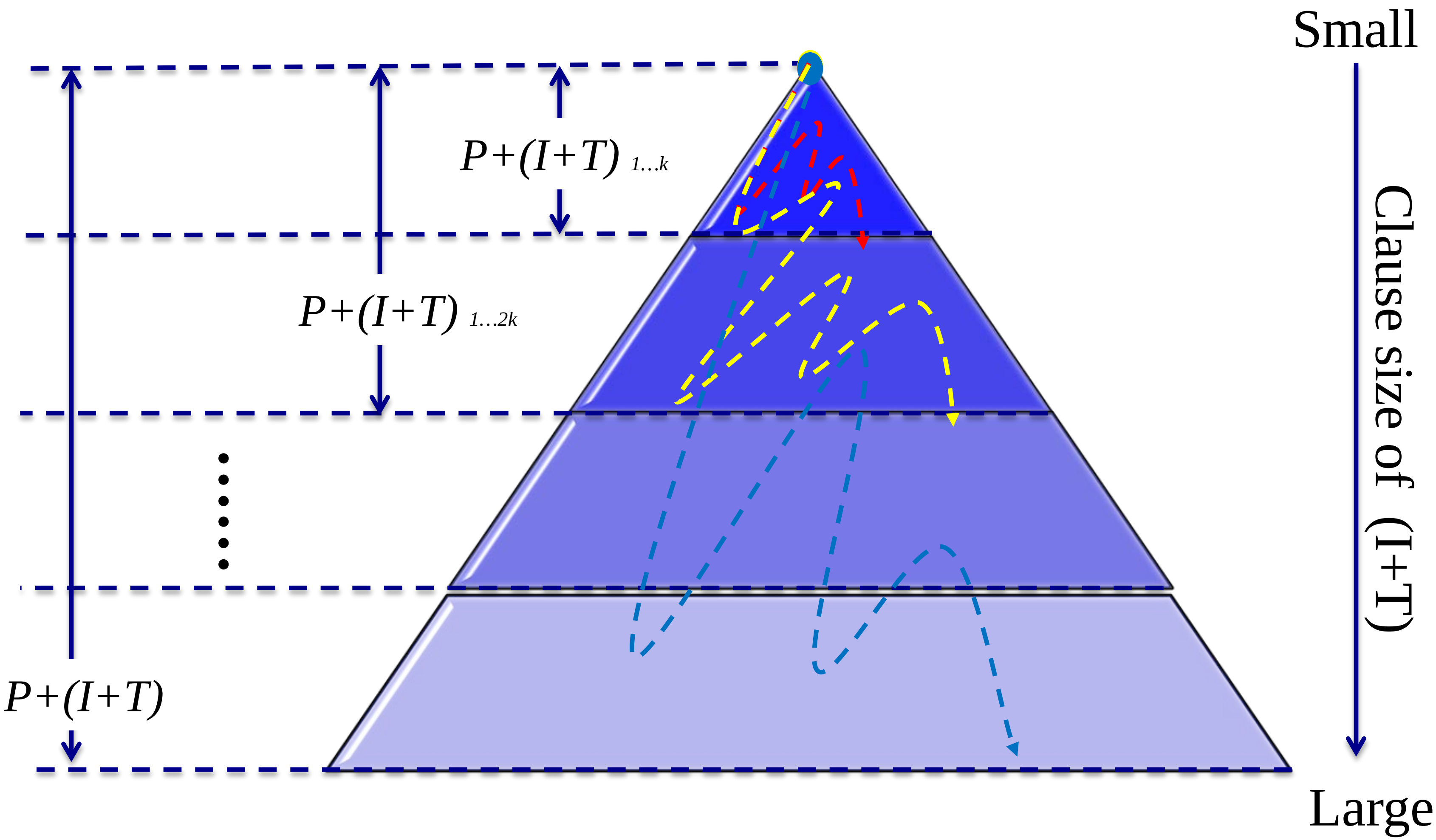
Learning from Incremental Partitioning



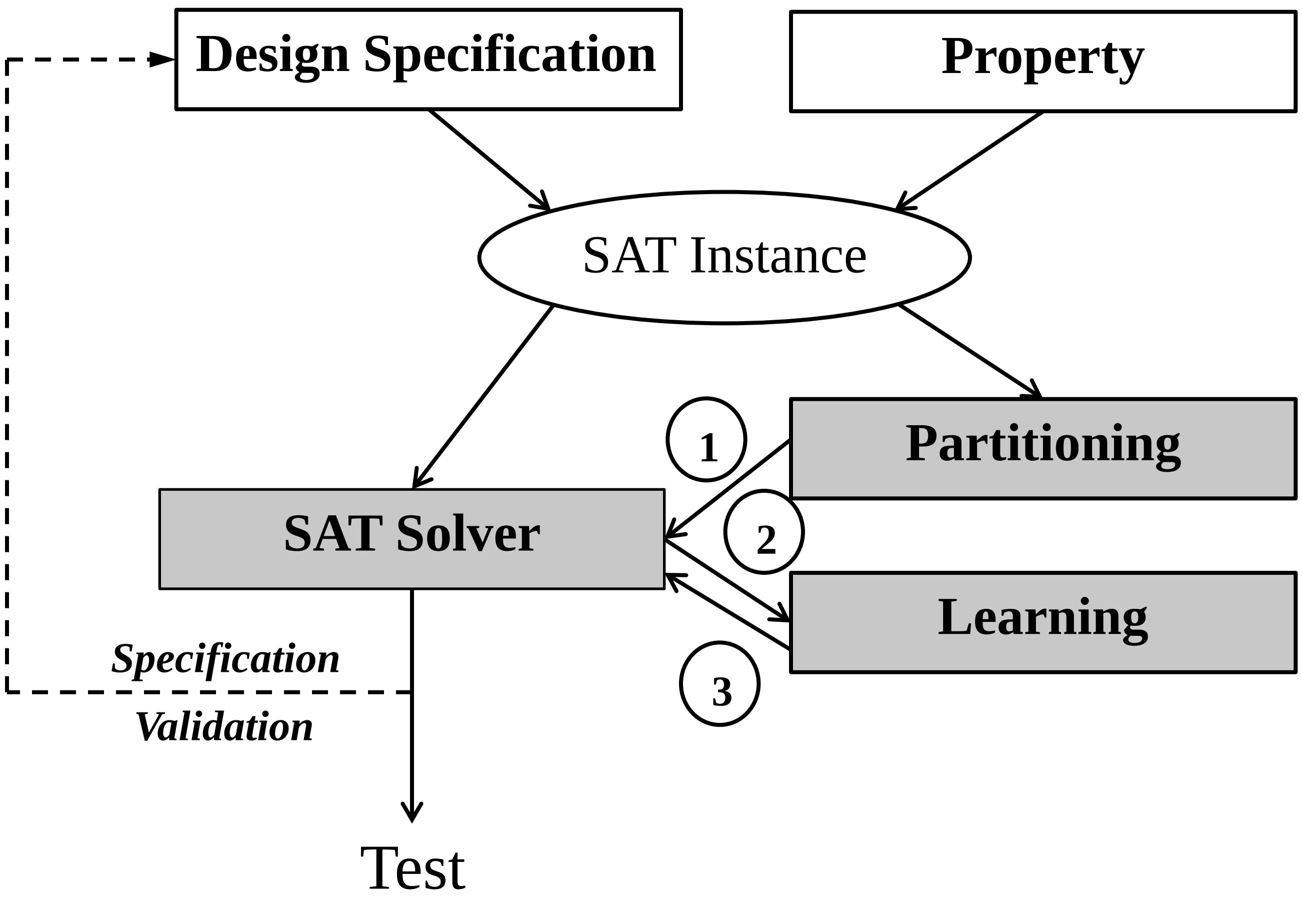
Learn from the Structure of CNF clauses

- For a set of small CNF clauses, the impact of a variable decision can be propagate instantly. Only a few decision levels are involved in the searching (avoid long-distance backtracking).
- We take clause size into account to derive high-quality conflict clauses.
  - A set of small CNF clauses has a high chance to derive small-size conflict clauses.
  - Smaller conflict clauses can prune more search space.

Learning from Clause Size Aware Partitioning



Test Generation using Self-Learning Techniques



Self-Learning Based Test Generation

Input: i) Formal model ***D*** and property ***P*** with the bound ***$Bound_p$***

ii) Partitioning type ***type*** and the number of patitions ***n***

Output: A test ***t_p*** to satisfy the property ***P***

1. Initialize lBucket[1..n] and vStat[1..sz][ ];
2. CNF=BMC(D, P, $Bound_p$);
3. {p_1,p_2,..., p_n} = Partition(CNF, type, n);
4. **for** i is from 1 to n do
   ① ($Assign_i$, $conf_i$) = SAT(p_i, lBucket[1..i-1], vStat);
   ② lBucket[i] = $conf_i$;
   ③ **for** j is from 1 to sz do
      a) **if** (assigni[j]==0) vStat[j][2]++;
      b) **else if** (assigni[j]==1) vStat[j][1]++;
5. (t_p,) = SAT(CNF, lBucket[1..n] , vStat);
6. Return t_p.

Outline

- Introduction
- SAT-Based Directed Test Generation
  - Test Generation using Model Checking
  - Test Generation using SAT-Based BMC
- Our Self-Learning Techniques
  - Motivation and Overview
  - Learning Oriented Partitioning Heuristics
  - Self-Learning Based Test Generation
- **Experiments**
- Conclusions

Experiments

- The experiments were conducted on a Linux PC with a 3.3GHz CPU and 4G RAM.

- We modified the SAT solvers MiniSAT2.2 and zChaff to incorporate our paritioning and self-learning techniques.

- In our experiments, all the SAT instances are divided into 4 partitions by default. For conflict clauses based self-learning, we only forward the conflict clauses whose size is smaller than 9.

Case Study 1: DLX+OSES

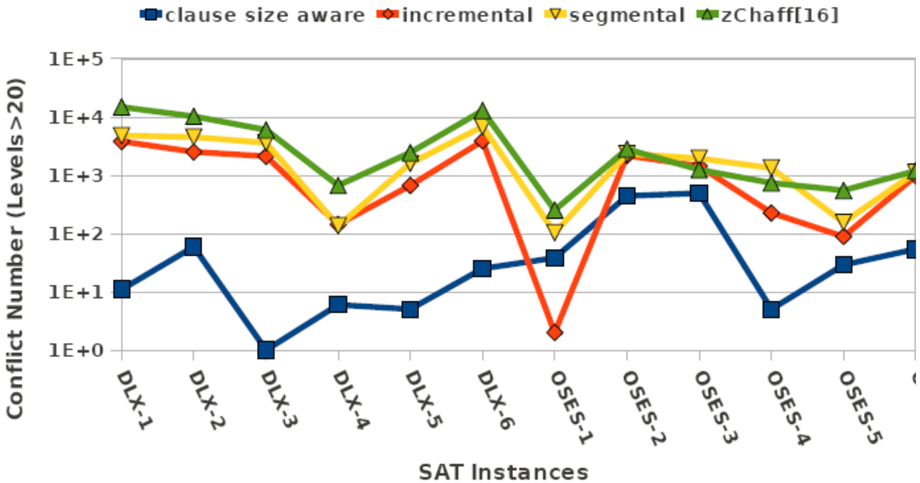
- **The benchmark is a set of SAT instances derived from a MIPS processor (DLX), and an online stock exchance system (OSES). All the SAT instances are generated using the tool NuSMV.**

SAT Instance	MiniSAT[15] /zChaff[16]	Segmental Partitioning ([15]/[16])			Incremental Partitioning ([15]/[16])			C. S. A. Partitioning ([15]/[16])			Max Speedup
		cls	var	cls+var	cls	var	cls+var	cls	var	cls+var	
DLX-1	1.9/104.5	3.6/72.8	4.3/22.2	**3.3**/23.0	4.6/133.4	4.9/28.6	3.7/24.6	3.5/23.1	4.3/6.7	3.4/**6.2**	0.6/16.9
DLX-2	1.5/58.0	3.5/58.5	3.3/20.8	**3.1**/19.0	4.3/90.0	4.5/19.1	3.8/13.9	3.8/35.3	4.1/9.5	3.2/**5.7**	0.5/10.2
DLX-3	0.8/32.4	1.7/26.3	2.1/12.7	**1.6**/15.2	2.1/189.2	2.0/8.7	2.0/11.8	1.9/16.9	2.1/5.0	1.8/**3.2**	0.5/10.1
DLX-4	0.2/2.1	0.5/2.4	0.6/0.8	0.5/**0.5**	0.7/1.5	0.6/0.7	**0.5**/0.6	0.5/1.4	0.5/0.8	0.5/0.7	0.4/4.2
DLX-5	0.6/7.9	1.3/6.1	1.3/4.2	1.3/3.8	1.4/9.6	1.5/4.9	1.2/2.8	1.1/6.6	1.3/2.9	**1.0/2.7**	0.6/2.9
DLX-6	1.1/101.3	2.6/105.0	3.0/35.5	2.7/31.0	3.2/106.1	2.8/27.2	2.9/25.6	2.6/71.7	2.6/3.7	**2.3/2.9**	0.5/34.9
OSES-1	0.6/5.5	**0.3**/0.6	0.4/18.5	0.4/4.2	0.3/6.3	0.4/1.7	0.3/0.8	0.4/1.0	0.4/**0.5**	0.4/0.7	2.0/11.0
OSES-2	0.6/209.3	**0.5**/80.2	2.1/200.1	0.9/290.6	2.2/131.1	1.6/177.6	1.1/249.0	1.1/246.6	2.3/73.1	1.0/**24.8**	1.2/8.4
OSES-3	4.7/121.7	1.7/89.8	9.5/181.5	1.0/238.9	**0.9**/89.5	2.0/140.8	9.2/269.5	1.9/74.5	1.3/**6.7**	5.4/35.3	5.2/18.2
OSES-4	0.9/28.8	3.5/47.3	7.9/91.4	4.0/99.6	4.0/39.0	3.4/45.2	**1.0**/8.7	5.1/36.4	18.6/**0.8**	5.0/0.9	0.9/32.0
OSES-5	0.1/21.8	**0.2**/20.2	0.3/8.8	0.3/5.2	0.6/12.0	0.3/3.5	0.4/2.7	0.4/21.1	0.3/**0.6**	0.3/0.7	0.5/36.3
OSES-6	0.9/53.8	**0.3**/74.4	0.4/76.5	0.4/67.4	0.7/122.8	0.5/15.2	0.7/78.8	1.2/136.4	1.4/**1.7**	1.5/2.3	3.0/31.6

- **Since the test generation time using MiniSAT without any leanring is small, the extra self-learning does not show significant improvement.**

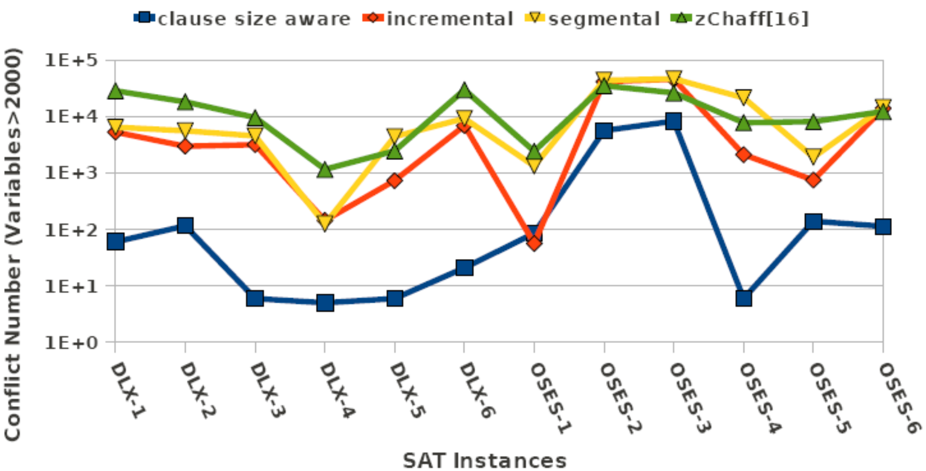
Case Study 1: DLX+OSES

The conflict which requires a backtrack with >20 decision levels



(a) The statistics of canceled levels

The conflict which requires a cancel of >2000 variable assignments



(b) The statistics of canceled variable assignments

Indications: Test generation complexity is significantly reduced in zChaff.

- Reduction of long-distance backtracking
- Reduction of bad decisions

Case Study 2: PIPE Processor Design

- **The benchmark is a set of SAT instances from 10 buggy variants of 12-pipelined processors named PIPE-SAT-1.1.**

Table 2: Test Generation Results for PIPE Processors

SAT Instance	MiniSAT[15] /zChaff[16]	Segmental Partitioning ([15]/[16])			Incremental Partitioning ([15]/[16])			C. S. A. Partitioning ([15]/[16])			Max Speedup
		cls	var	cls+var	cls	var	cls+var	cls	var	cls+var	
PIPE1	12/854	28/488	234/761	190/572	8/507	**1**/242	**1**/924	26/844	2/**7**	2/8	12.0/122.0
PIPE2	1/1090	**1**/2085	**1**/421	**1/1**	11/943	**1**/174	**1**/217	**1**/1124	**1**/11	**1**/11	1.0/1090.0
PIPE3	15/524	8/879	169/1185	144/1495	8/692	**1**/521	1/**142**	16/707	4/276	4/276	15.0/3.7
PIPE4	6/670	36/77	1055/162	**1**/12	5/311	NA/**4**	118/4	18/684	2/237	2/236	6.0/167.5
PIPE5	5/395	2/1345	3/2544	2/2613	9/90	2/595	10/1240	16/401	**2/55**	4/**55**	2.5/7.2
PIPE6	1/1	**1**/98	4/106	11/55	1/2	2/122	4/142	**1/1**	**1/1**	**1/1**	1.0/1.0
PIPE7	13/1117	284/505	322/2069	112/1184	450/681	6/1188	**4**/162	13/1059	4/**38**	4/39	3.3/29.4
PIPE8	2/73	**6/1**	44/69	91/61	7/110	NA/75	NA/76	9/73	**6**/30	11/30	0.3/73
PIPE9	30/3897	32/2522	2/404	2/**395**	43/421	3/1295	**1**/964	6/3874	1/1022	1/1020	30.0/9.9
PIPE10	174/907	112/647	672/973	4/497	111/446	**2**/139	**2/118**	17/661	4/148	4/147	87.0/7.7

- **Both the modified zChaff and MiniSAT shows that our self-learnings can achieve significant improvement.**

Case Study 3: UNSAT PIPE_OOO

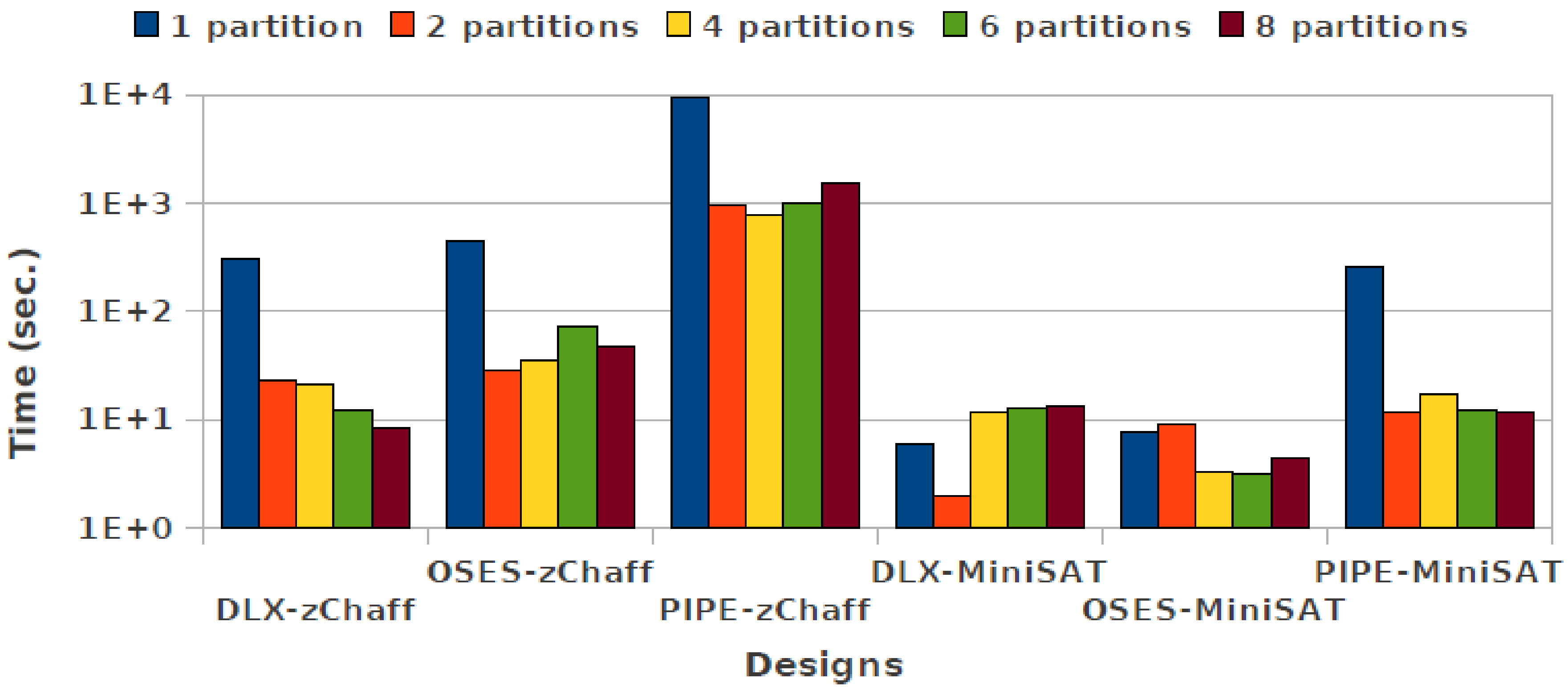
- **We also conducted the experiment on a set of bigger variants of the pipe_ooo benchmarks named PIPE-OOO-UNSAT-1.1.**

Table 3: SAT Solving Results for UNSAT Instances of PIPE Processors

SAT Instance	MiniSAT[15] /zChaff[16]	Segmental Partitioning ([15]/[16])			Incremental Partitioning ([15]/[16])			C. S. A. Partitioning ([15]/[16])			Max Speedup
		cls	var	cls+var	cls	var	cls+var	cls	var	cls+var	
pipe2-ooo	0.3/0.06	0.1/**0.05**	**0.04**/0.08	0.05/0.09	0.1/0.06	0.1/0.10	0.1/0.10	0.3/0.08	0.1/0.19	0.1/0.13	7.5/1.2
pipe3-ooo	10.7/0.8	12.5/0.9	6.7/0.9	5.9/1.1	11.6/1.1	12.5/1.0	8.0/0.9	8.4/**0.8**	**0.8**/1.5	1.2/1.1	13.4/1.0
pipe4-ooo	94/6	142/4	118/3	111/4	114/**3**	77/4	197/4	50/6	13/5	**10**/5	9.4/2.0
pipe5-ooo	280/18	356/15	64/15	**47**/14	498/14	NA/12	1065/**11**	476/13	51/18	62/21	6.0/1.6
pipe6-ooo	731/69	NA/65	NA/64	NA/61	279/67	NA/50	NA/**48**	645/66	74/78	**70**/70	10.4/1.4
pipe7-ooo	1016/266	**702**/278	NA/285	NA/294	NA/303	NA/**199**	NA/208	NA/285	NA/304	NA/295	1.4/1.3
pipe8-ooo	NA/1107	NA/1330	NA/1083	NA/1104	NA/1387	NA/772	NA/**684**	NA/1326	NA/1354	NA/1334	1.4/1.6
pipe9-ooo	NA/1697	NA/1716	NA/1565	NA/1656	NA/1688	NA/**940**	NA/989	NA/1497	NA/2005	NA/1913	NA/1.8

- **Interestingly, our approach can also benifit the UNSAT instance checking. We can find that zChaff outperforms MiniSAT in this benchmark, and our self-learning techniques can achieve an up to 13.4 times improvement using MiniSAT and 2 times improvement using zChaff.**

Evaluation of the number of partitions



Indications:

-For zChaff, the methods using 4 partitions show a better performance for these three benchmarks.

- For MiniSAT, the approaches using 2 partitions or 4 partitions are good enough for test generation.

Outline

- Introduction
- SAT-Based Directed Test Generation
  - Test Generation using Model Checking
  - Test Generation using SAT-Based BMC
- Our Self-Learning Techniques
  - Motivation and Overview
  - Learning Oriented Partitioning Heuristics
  - Self-Learning Based Test Generation
- Experiments
- **Conclusions**

Conclusions

- Functional validation is a major bottleneck
  - SAT-based approaches are promising for automated test generation.
- Proposed efficient self-learning techniques for automated generation of directed tests
  - Investigated two kinds of learning objects.
  - Developed three learning-oriented partitioning heuristic methods.
- Successfully applied on both hardware and software designs
  - Significant reduction in test generation time.

Thank you !