Efficient Techniques for Directed Test Generation Using Incremental Satisfiablility

Prabhat Mishra and Mingsong Chen

Embedded Systems Lab

Computer and Information Science and Engineering

University of Florida, USA

Outline

- Introduction
- Related Work
- Test Generation Using Incremental SAT
  - Clustering of similar properties
  - Name substitution for computation of intersection
  - Identify and reuse of common conflict clauses
- Experiments
- Conclusion

Introduction

- Functional verification is a major bottleneck
  - Increasing design complexity
  - Decreasing time-to-market
- Directed tests can reduce validation effort
  - Same coverage goal can be reached using small number of directed tests
- Model checking based test generation
  - Automated generation of directed tests
  - Unsuitable for large designs
    - State space explosion
  - Need to reduce test generation time (complexity)

Motivation

- SAT-based bounded model checking (BMC) can address state space explosion
  - Searches within a bound
  - CNF can be smaller than BDD
  - SAT has many heuristic decision algorithms
  - Exploit the similarity of SAT instances
- Existing approaches exploit similarity for the same test generation instance
  - Same property with different bounds
- We extend incremental SAT to exploit test generation involving multiple properties

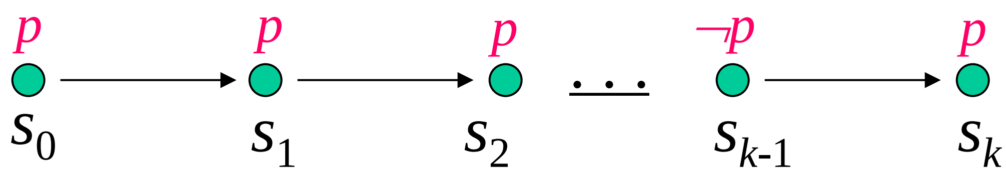
SAT-based Bound Model Checking

- 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 $\Omega(k)$ 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 $\Omega(k)$ is satisfiable, then we can get an assignment which can be translated to a test.

Implication Graph, Conflict Clause

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

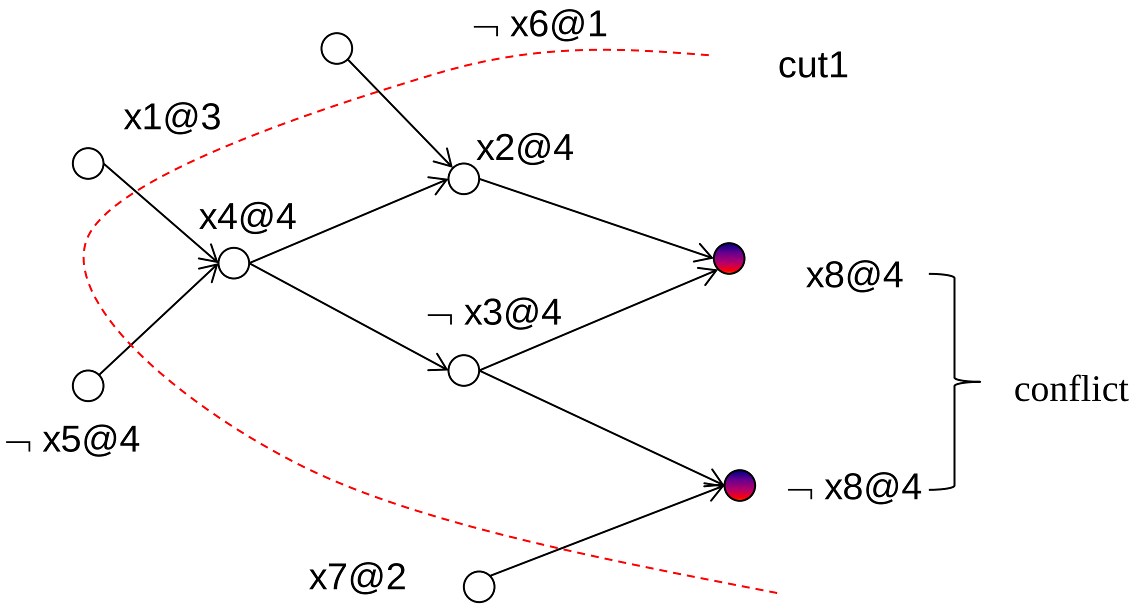
$$\omega_2 = (\neg x_8 \vee x_3 \vee \neg x_7)$$

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

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

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

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



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

Incremental SAT

- **Given two CNF formulas (sets of clauses) S1 and S2, the following statement holds.**

(1) Let π be the conflict clause learned from S1, then:

S1 is satisfiable iff S1 $\wedge$ π is satisfiable

(2) Let $\varphi_0 \equiv S1 \cap S2$, if π **is a conflict clause learned from** φ_0 then:

$S1$ is satisfiable iff $S1 \wedge \pi$ is satisfiable.

$S2$ is satisfiable iff $S2 \wedge \pi$ is satisfiable.

- **So when checking S2, we can reuse the knowledge π learned during checking *S1*.**
- **Currently, the incremental SAT is used for checking the *same property with different bounds*.**

Test Generation Using Incremental SAT

- The goal of our approach is to reduce the overall functional validation effort by reducing the test generation time for directed tests.
- The basic idea is to learn from solving one property and sharing learning (through conflict clauses) for solving the similar properties in the cluster.
- This paper focuses on test generation for safety properties. We assume that the bound for each property can be pre-determined based on the structure of the model.

Workflow of Our Method

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≠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

   ③ **Solve** CNF^i_1 to get the conflict clauses CC^i_1 and $test^i_1$

   ④ **for** each CNF^i_j (j≠1)

      a) $CNF^i_j = CNF^i_j$ + Filter (CNF_i , j)

      b) Solve CNF^i_j and get the $test^i_j$

      endfor

   endfor

Property Clustering

- More intersections imply more conflict clause forwarding. However, for n properties, clustering based on intersection need n(n-1)/2 comparisons.

- A simple and natural way to cluster properties is to exploit the structural and behavior similarity.

- Rules used for base property selection.
  - Variable and/or sub-expression overlap
  - Small bound.

Name Substitution

- **The DIMACS file contains the mapping between the CNF variable and the variables of the model.**
  - **E.g.** **C 8 => V1_var [6]**

    Variable 8 is used to refer to 7th bit of variable *var* in the specification in time step 1.
- **Name substitution can get more intersection.**

```
C 1 => a_1
C 2 => b_1
C 3 => a_2

p  cnf  3  3
 -1   2   0
  3   2   0
  1   3   0
```

DIMACS f1

```
……
C 4 => a_1
C 5 => b_1
C 6 => a_2

p  cnf   6   4
  5   -4    0
  5    6    0
  1    4    0
  2   -3    0
```

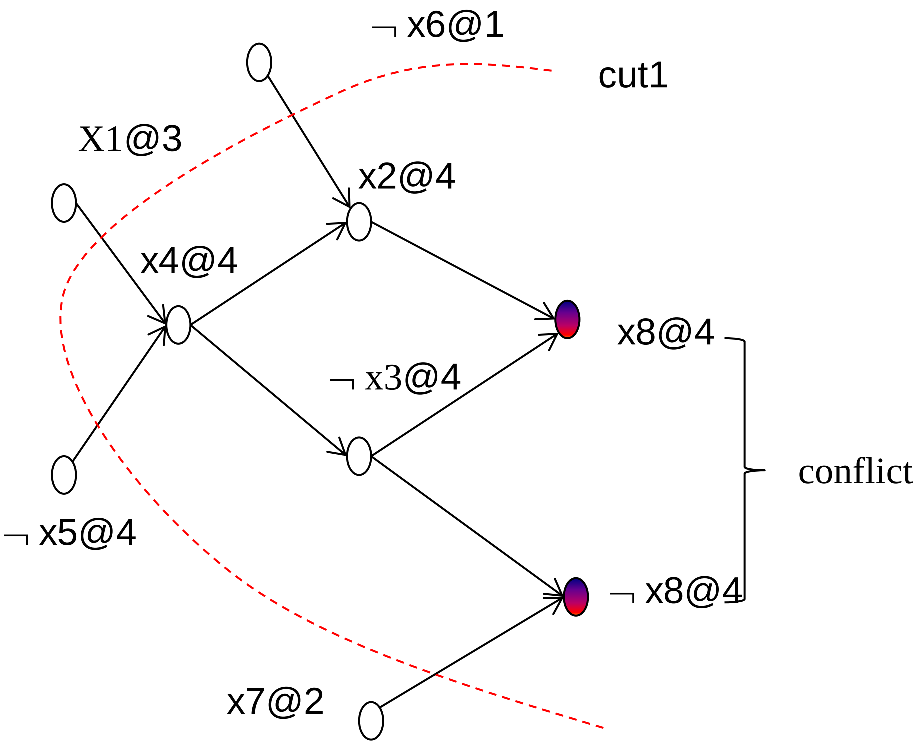
DIMACS f2

```
C 1 => a_1
C 2 => b_1
C 3 => a_2

P  cnf   6   4
  2    -1     0
  2     3     0
  4     1     0
  5    -6    0
```

DIMACS f2’

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**

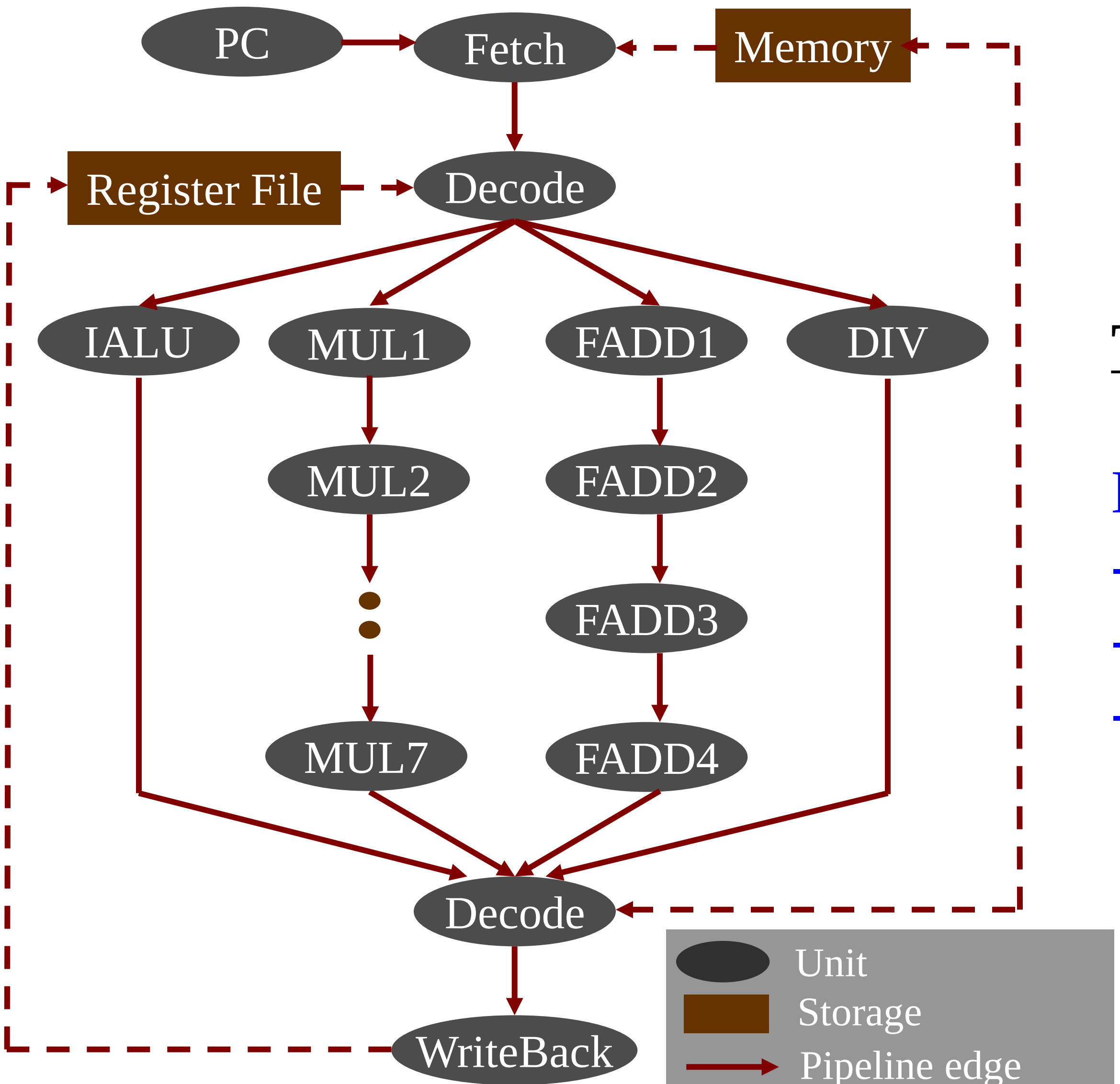
Let ∧ be the bit "AND" operation. (0111 ∧ 1010 ∧ 1111 ∧ 1010 ∧ 1110) = 0010.
So the conflict clause (¬X1 ∨X5 ∨X6 ∨¬X7) can be reused for property 2.

On-line Stock Exchange System

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

Clusters (properties)	Preprocess Time	zChaff (sec.)	Our method (sec.)	Improv. Factor
Cluster 1 (2)	**3.79**	**59.82**	**4.43**	**13.50**
Cluster 2 (4)	**11.98**	**78.13**	**13.68**	**5.72**
Cluster 3 (4)	**11.81**	**161.91**	**40.50**	**4.00**
Cluster 4 (4)	**12.70**	**144.12**	**51.80**	**2.78**
Cluster 5 (4)	**12.76**	**426.09**	**75.34**	**5.66**
Average	**4.08**	**48.33**	**10.32**	**4.68**

Case Study 2: MIPS Processor



The Architecture

MIPS Processor
- 20 nodes
- 24 edges
- 91 instructions

Case Study 2: MIPS Processor

- The processor has five pipeline stages: fetch, decode, execute, memory and writeback. The execute stage has four execution path, 1 stage integer ALU, 7 stages multiplier, 4 stage floating point adder and one multi-cycle divider.

Clusters	CNF Clauses	Intersection Size	zChaff	Our Method	Improv. Factor
CLALU	460994	457168	19.35	5.10	3.79
CLFADD	592119	67894	61.61	42.46	1.45
CLMUL	854368	522283	718.85	159.21	4.51
CLDIV	526517	457160	35.07	8.19	4.28
Average	608504	376126	208.72	53.74	3.88

Conclusions

- Functional validation is a major bottleneck
- Test generation using SAT-based BMC
  - Incremental SAT involving one property (test)
- Directed test generation using Incremental SAT
  - Share learning across multiple properties
    - Clustering of similar properties
    - Name substitution for computation of intersection
    - Identify and reuse of common conflict clauses
  - Reduces test generation time and complexity
    - **Four times** improvement in test generation time for both software and hardware designs

Thank you !