SAT-based Diagnosis in an Abstraction-Refinement Framework

Cécile Braunstein
Group of Computer Architecture
University of Bremen

Grenoble, May 5th 2008

Arbeitsgruppe Rechnerarchitektur

Team - Rolf Drechsler

- Start in 2001
- 19 workers
- 15 scientists : 1 professor,2 associate professors, 2 post-doc, 10 PhD students

Research area : Formal Methods

- Circuit and system descriptions : Hardware description languages, System level description languages (e.g. SystemC)
- Verification : Model checking, Equivalence checking, Debugging
- Test : Automatic test pattern generation, Design for testability
- Logic-synthesis : Integration of technology dependent information, Multi-valued logic

Partners

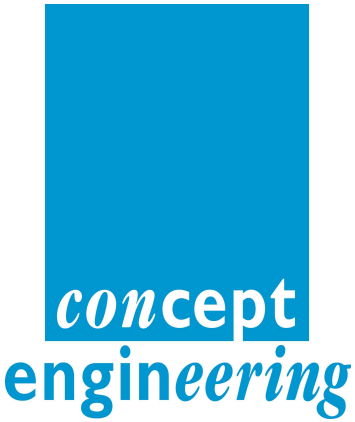

OneSpin
SOLUTIONS


NXP
founded by Philips


AMD


PHILIPS


infineon


concept
engineering


SIEMENS

SAT-based Diagnosis in an Abstraction-Refinement Framework

1. Motivation
2. SAT techniques
   - SAT Solver
   - Bounded Model checking
3. Counter-Example Guided Abstraction Refinement loop
   - Localization Reduction
   - Counter-example analysis
4. Diagnosis with a CEGAR loop
   - Classical SAT-based diagnosis
   - CEGAR-based diagnosis
5. Conclusion and perspectives

Motivation

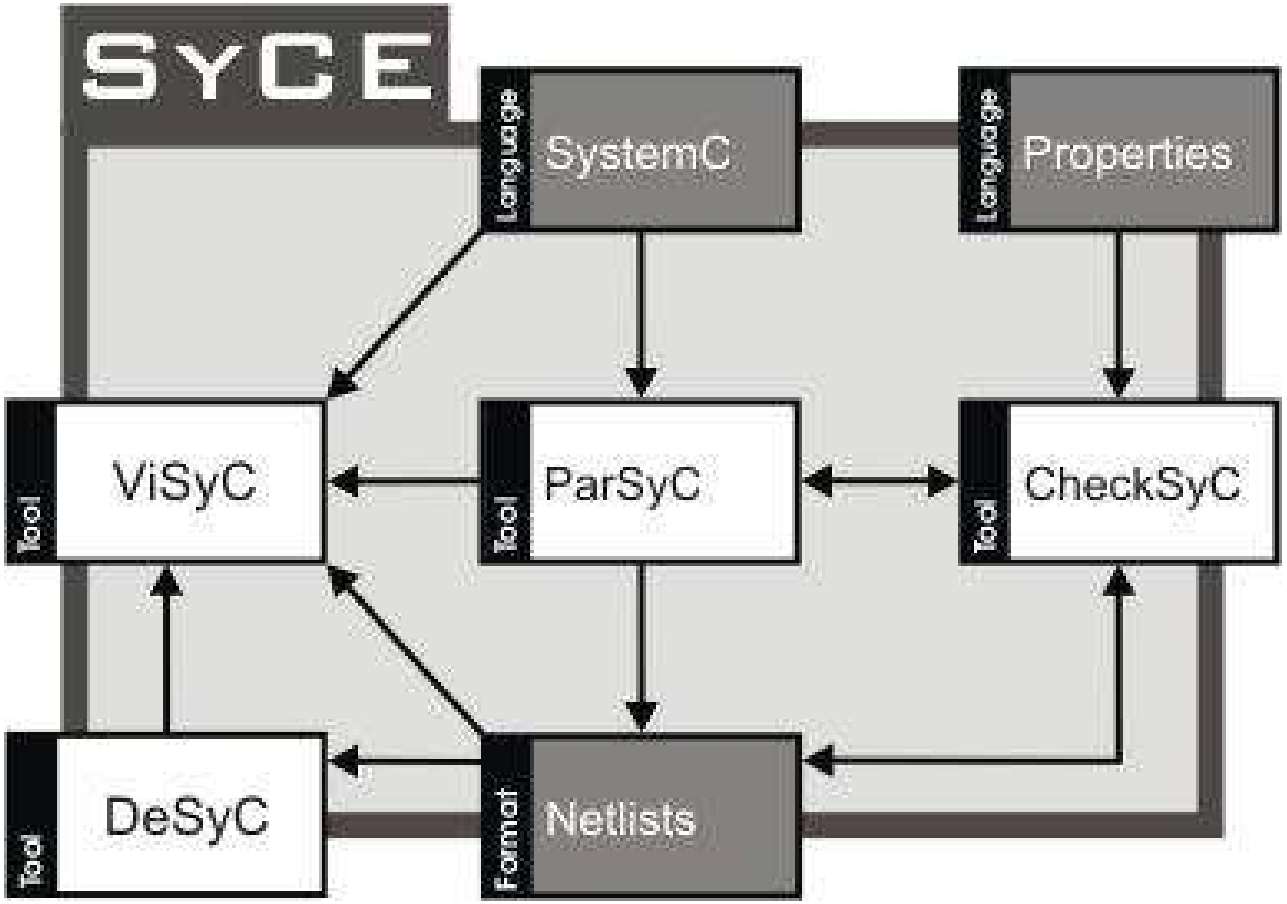
Verification process

- Verification becomes very important :
  → Up to 80% of the design costs

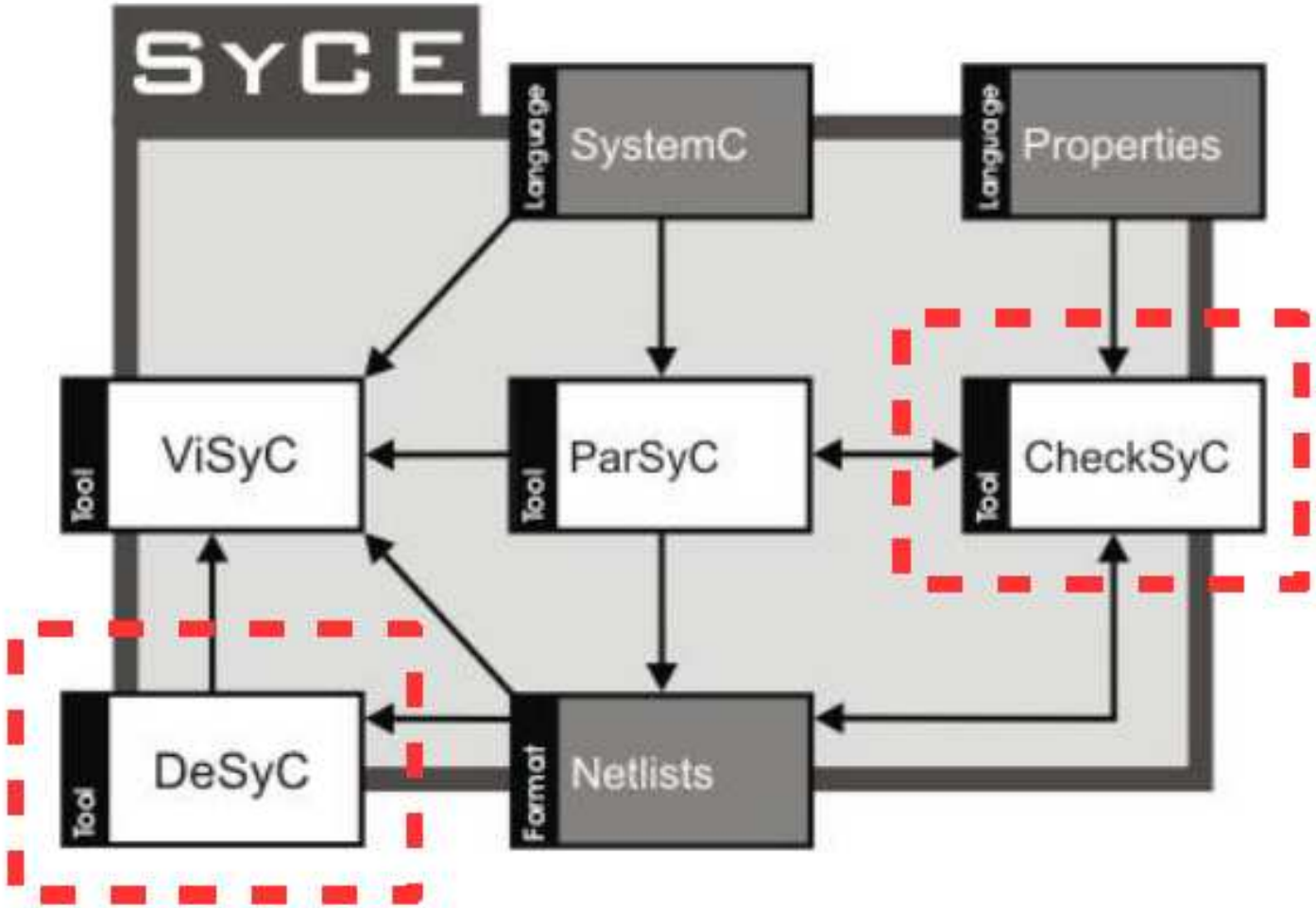
- Need for :
  - Efficient push button tools
  - Fast algorithms

- System level
  - Prototyping
  - Hardware-Software Co-Design

An Integrated Environment for System Design [2001]



An Integrated Environment for System Design [2001]



My Contributions

Goal

Reducing the memory needs for the checker and the debugger

Counter-Example guided Abstraction Refinement framework

- Implementation of a CEGAR loop for Bounded Model Checking
- Implementation of a CEGAR loop for SAT-based diagnosis

SAT Solver

Satisfability

- Boolean formula : $f(v_1, v_2, \ldots)$
  - Find an assignment such that $f(a_1, a_2, \ldots) =$ **true** ; f is SAT
  - Or show that there exists no such assignment ; f is UNSAT
- NP-complete problem

Tools

GRASP, MiniSat, Zchaff ...

→ They can handle 10 thousand variables and millions of constraints

Applications

- Artificial intelligence
- Electronic Design Automaton : ATPG, Model checking ...

Bounded Model Checking

BMC [Biere and al. 1999]

- Can handle larger model than BDD model checking
- Limited to a bounded path
  - ➢ More suitable to find bug than to prove a property

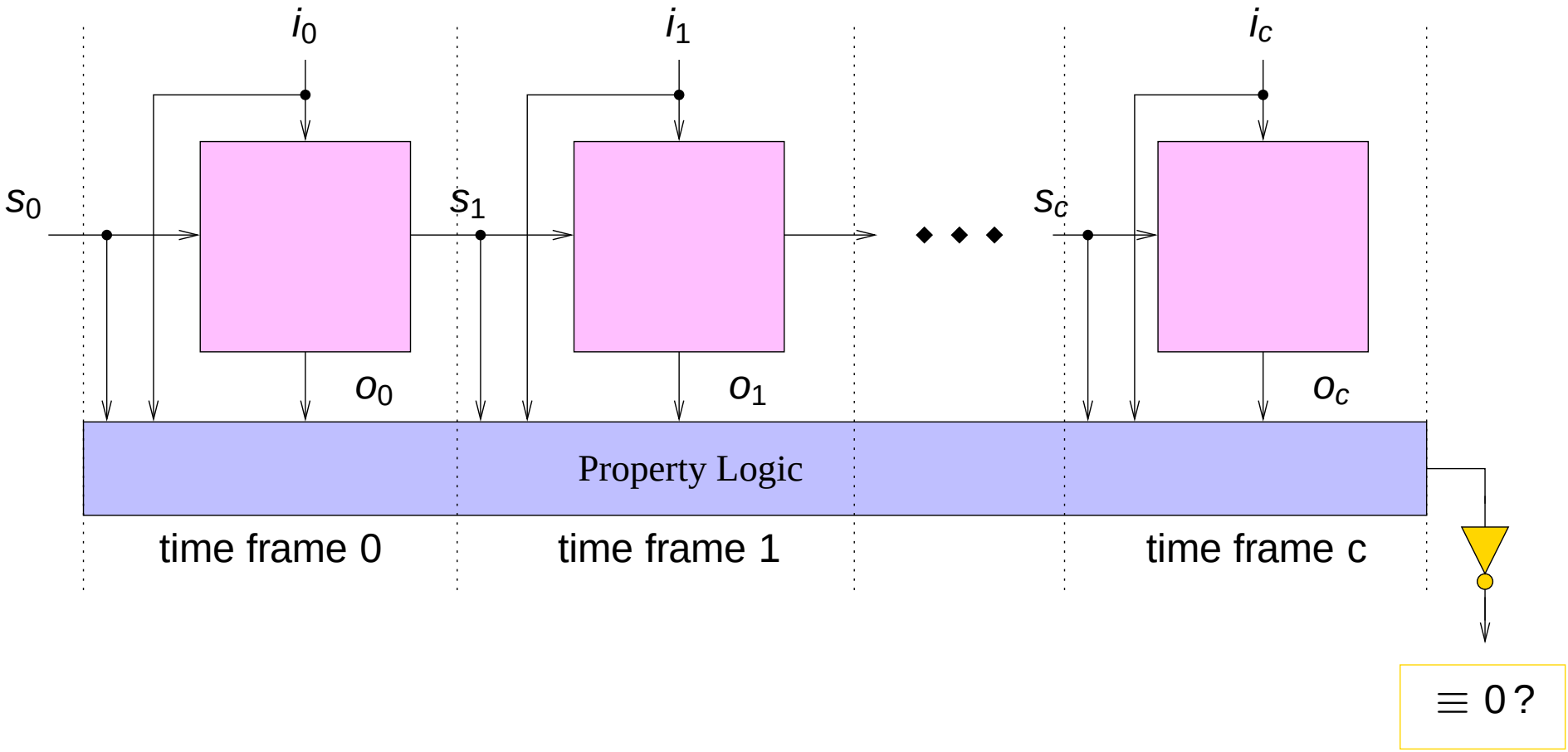
Principle

- A property φ argues over a finite time interval [0,c]
- BMC SAT instance :

$$I(s) \wedge \bigwedge_{i=0}^{c-1} T(s_i, s_{i+1}) \wedge \neg\varphi$$

  - ➢ UNSAT : property holds
  - ➢ SAT : property fails, the assignment provides a counter-example

BMC-Unrolling



Reducing the memory needs

Abstraction

Find small subsets of the original model where the property can be proved or refuted :

- Contain enough information
- Can be handled by a model checker

Abstraction combined with automatic refinement

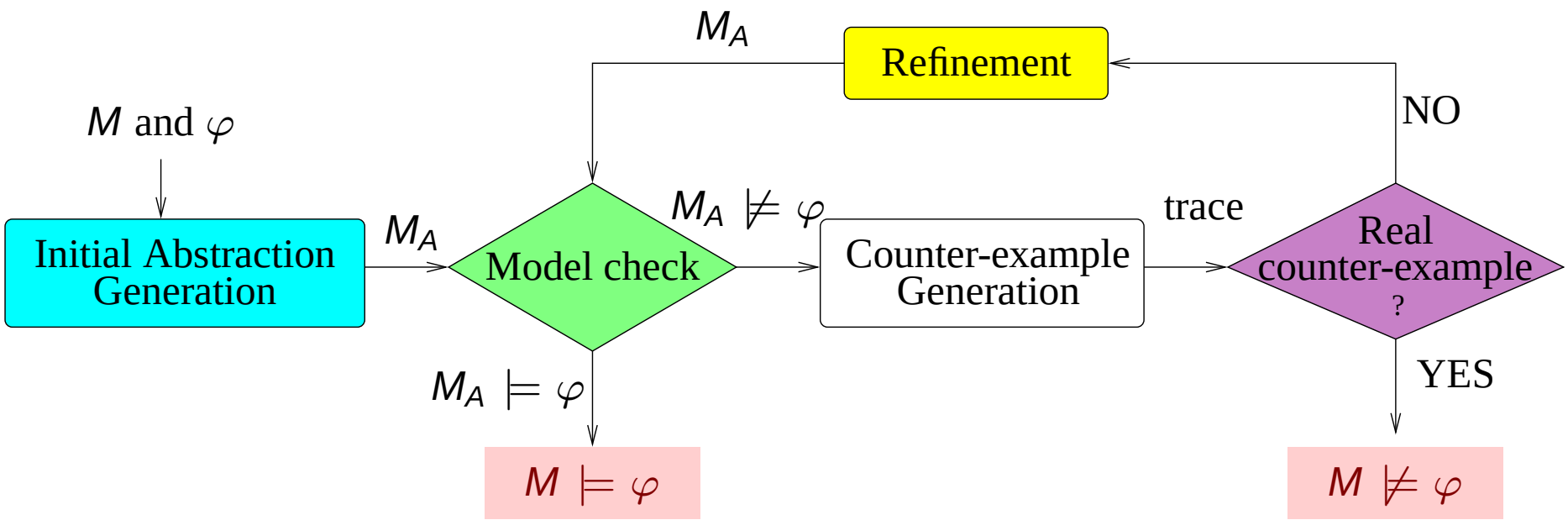
Conservative abstraction :

- Preserve all behavior of the original model
- May introduce additional behaviors in the abstract model

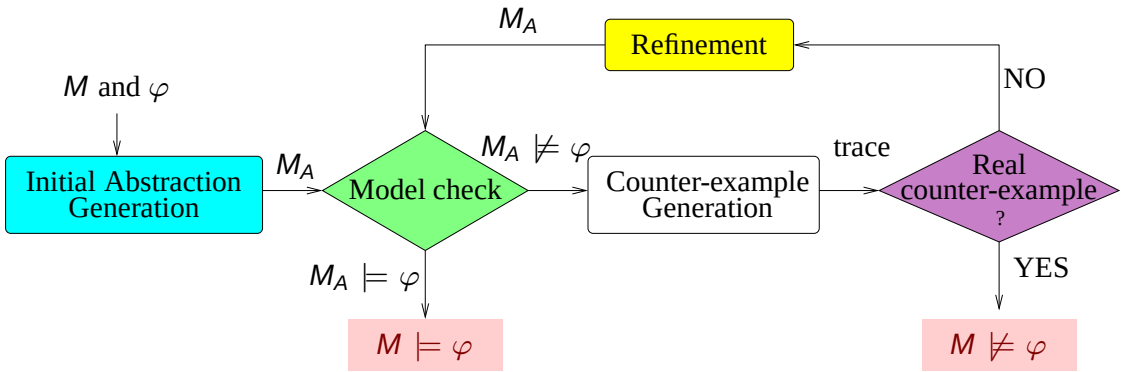
→ Spurious counter-examples may exist

We need to refine the abstraction

Counter-example guided abstraction refinement loop (CEGAR [Clarke/Grumberg and al. 00])



Counter-example guided abstraction refinement loop (CEGAR [Clarke/Grumberg and al. 00])



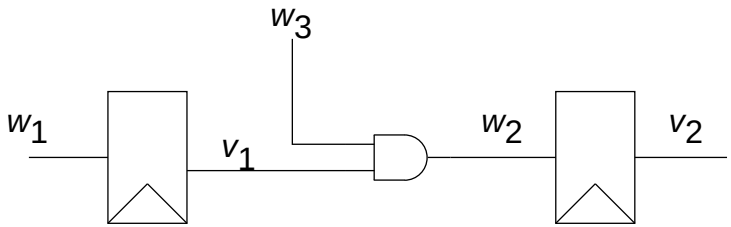
3 Major issues

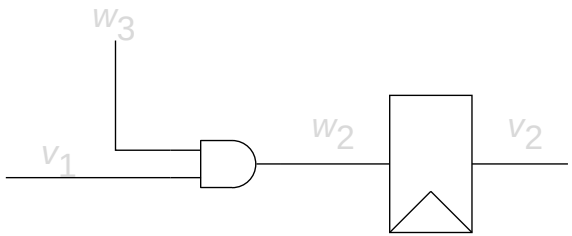
1. Choose abstraction
2. Determine spurious counter-example (concretization test)
3. Perform the refinement

Localization Reduction

[Kurshan 1994]

Partitioning variables into visible and invisible set.





Concrete $\langle V, W, I, T \rangle$

- $V = v_1, v_2$
- $W = w_1, w_2, w_3$
- $I(V) = \neg v_1 \wedge \neg v_2$
- $T(V, W, V') = (v_1' \leftrightarrow w_1) \wedge (v_2' \leftrightarrow w_2) \wedge (w_2 \leftrightarrow (w_3 \wedge v_1))$

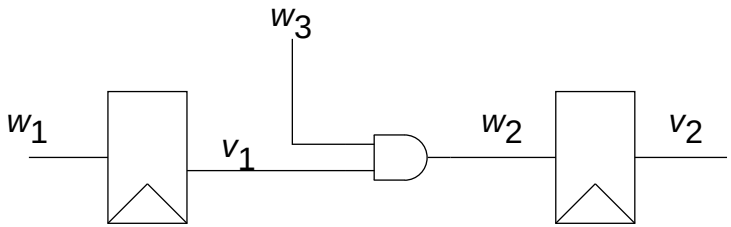
Abstract $\langle \hat{V}, \hat{W}, \hat{I}, \hat{T} \rangle$

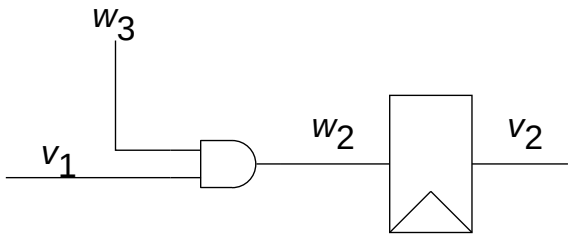
- $\hat{V} = v_2$
- $\hat{W} = w_2, w_3, v_1$
- $\hat{I}(\hat{V}) = \neg v_2$
- $\hat{T}(\hat{V}, \hat{W}, \hat{V}') = (v_2' \leftrightarrow w_2) \wedge (w_2 \leftrightarrow (w_3 \wedge v_1))$

Localization Reduction

[Kurshan 1994]

Partitioning variables into visible and invisible set.





Concrete $\langle V, W, I, T \rangle$

- $V = v_1, v_2$
- $W = w_1, w_2, w_3$
- $I(V) = \neg v_1 \wedge \neg v_2$
- $T(V, W, V') = (v_1' \leftrightarrow w_1) \wedge (v_2' \leftrightarrow w_2) \wedge (w_2 \leftrightarrow (w_3 \wedge v_1))$

Abstract $\langle \hat{V}, \hat{W}, \hat{I}, \hat{T} \rangle$

- $\hat{V} = v_2$
- $\hat{W} = w_2, w_3, v_1$
- $\hat{I}(\hat{V}) = \neg v_2$
- $\hat{T}(\hat{V}, \hat{W}, \hat{V}') = (v_2' \leftrightarrow w_2) \wedge (w_2 \leftrightarrow (w_3 \wedge v_1))$

Counter-example analysis

Principle

- Model Check the abstract Model
  - Yes : property holds
  - No : Abstract counter-example (ACE) : $(c_1, \ldots, c_k)$
    each c_i is an assignment to a variable of the abstract model
- Simulate the abstract counter-example on the concrete model
  - Symbolic simulation with SAT solver

Counter-example analysis

Concretization test

Let k be the length of the counter-example,
SAT formula =

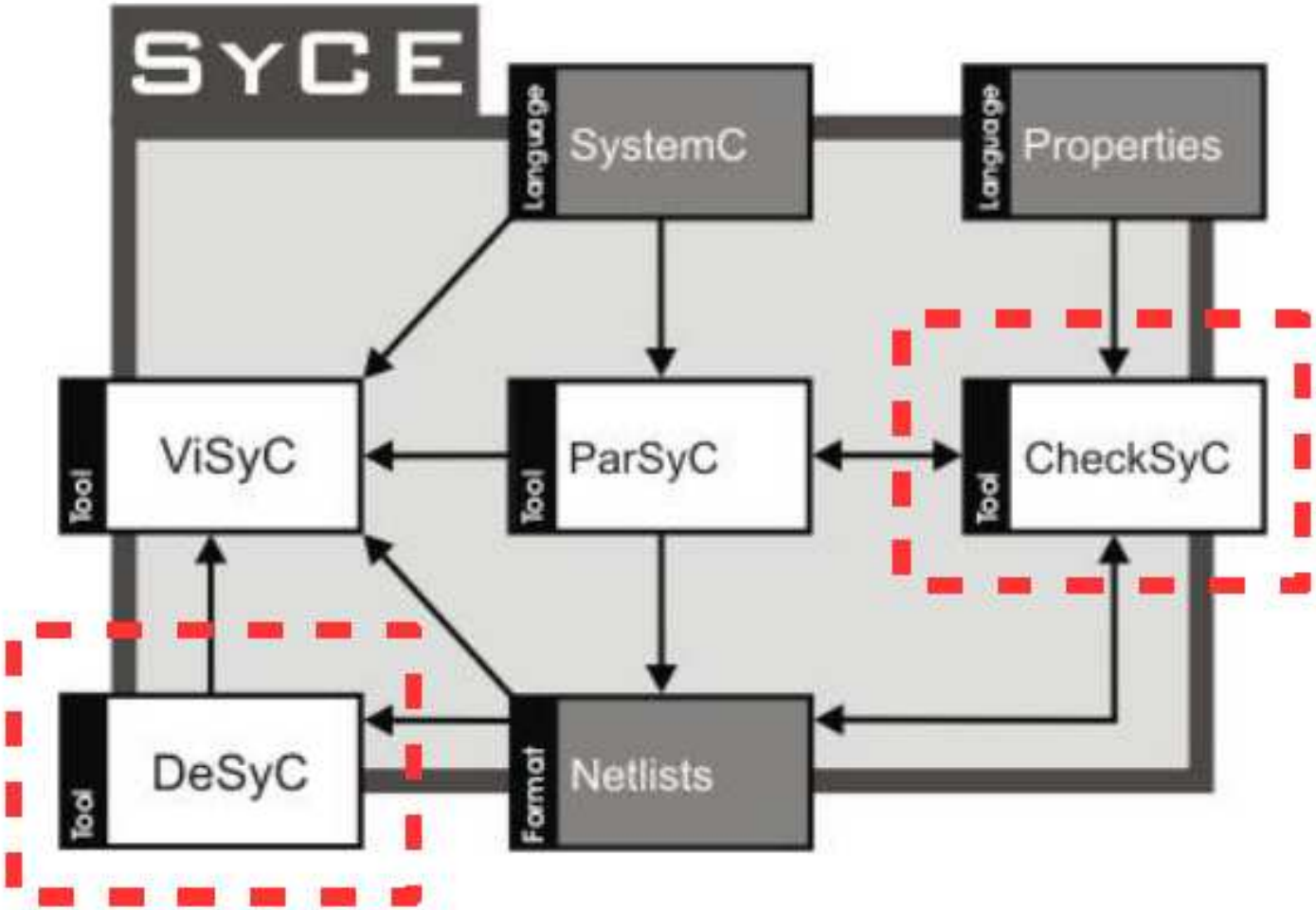
$\bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge$ (Unrolled transition relation)

$\bigwedge_{i=0}^{k-1} visible(s_i) = c_i$ (Restriction of visible variables to ACE)

If the formula is

- SAT : the abstract counter-example is a real one ; the formula failed
- UNSAT : the abstract counter-example is spurious ; refine abstraction

An Integrated Environment for System Design [2001]



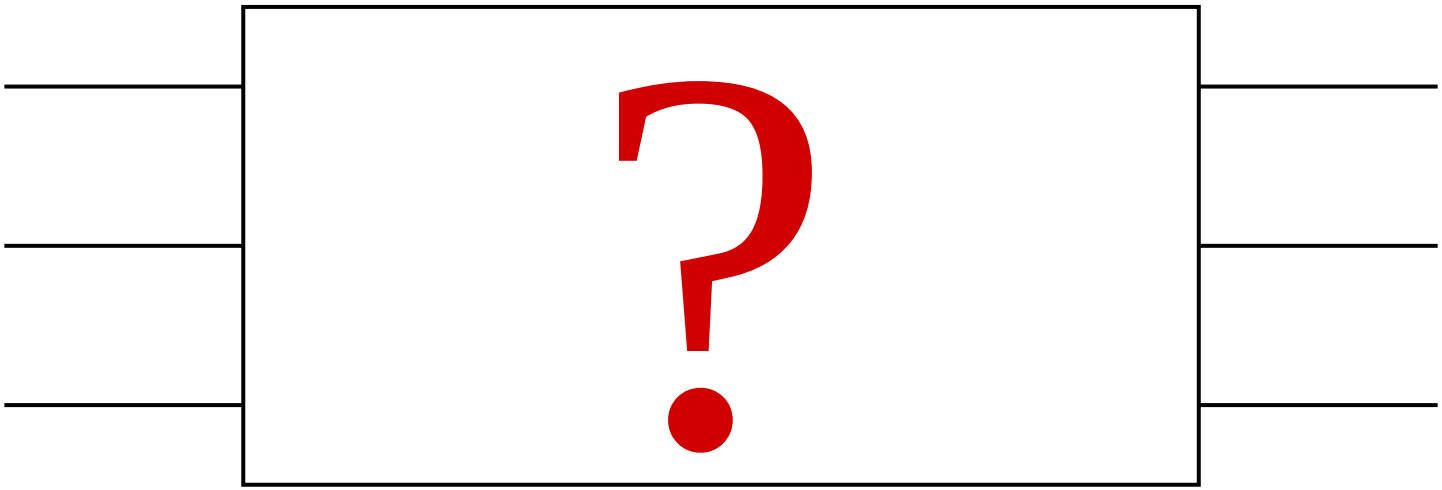
Diagnosis Problem

Design Error Detection and Correction

Given :

- A set of failure traces : test vectors or counter-examples
- An implementation : gate or system level

Question : Where is the error ?

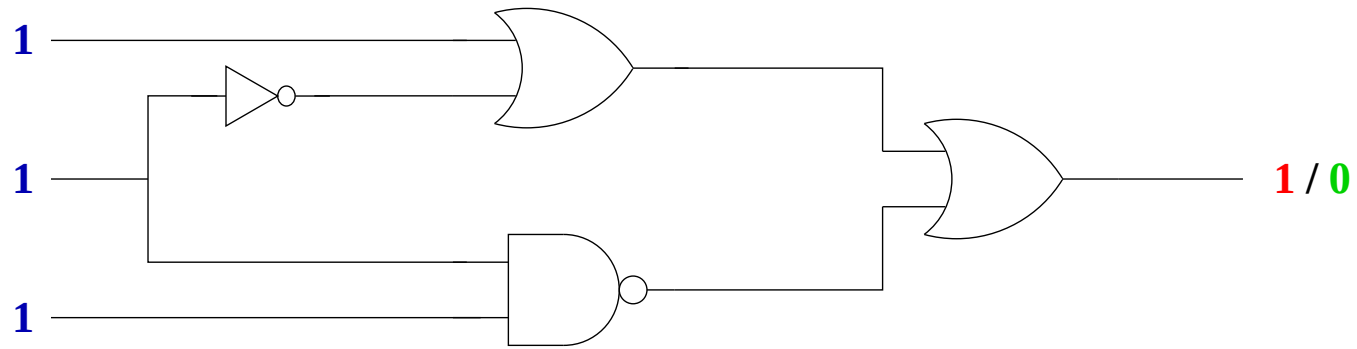


Classical SAT-based Diagnosis

Process

Given a model *M* and a set of failure traces

- Add multiplexers to force a gate to behave abnormally
- Use a SAT solver to solve the problem

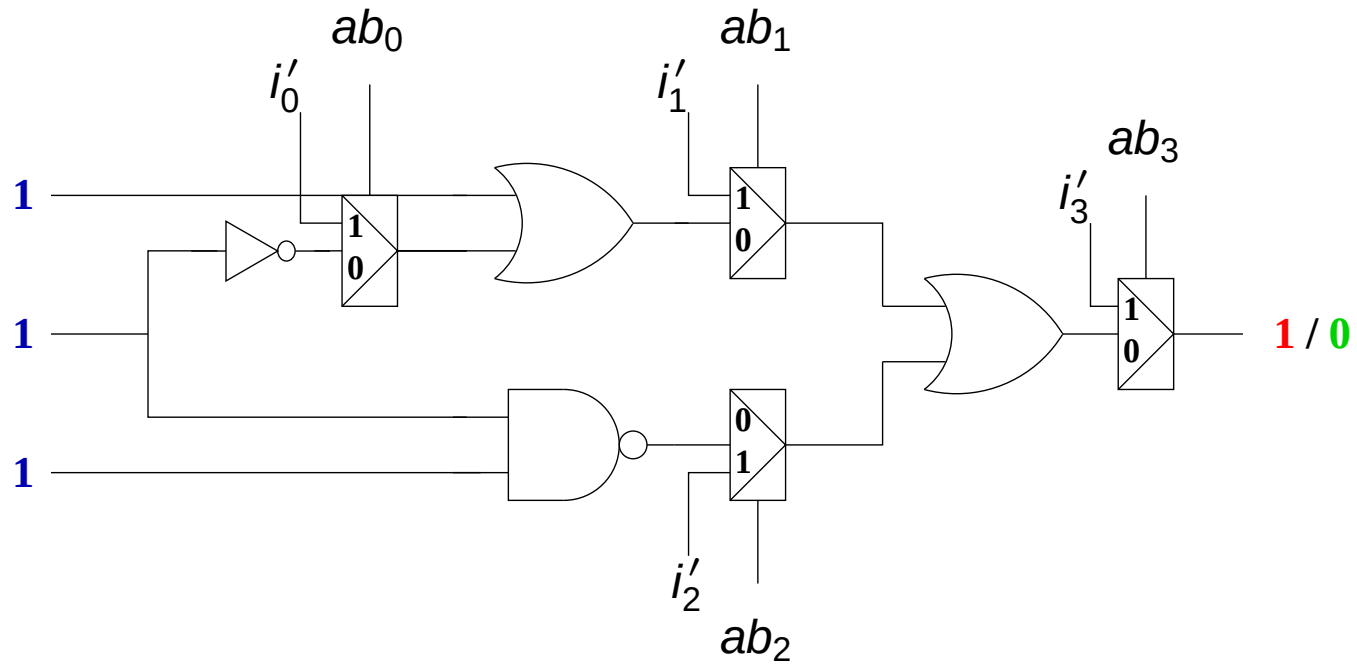


Classical SAT-based Diagnosis

Process

Given a model *M* and a set of failure traces

- Add multiplexers to force a gate to behave abnormally
- Use a SAT solver to solve the problem

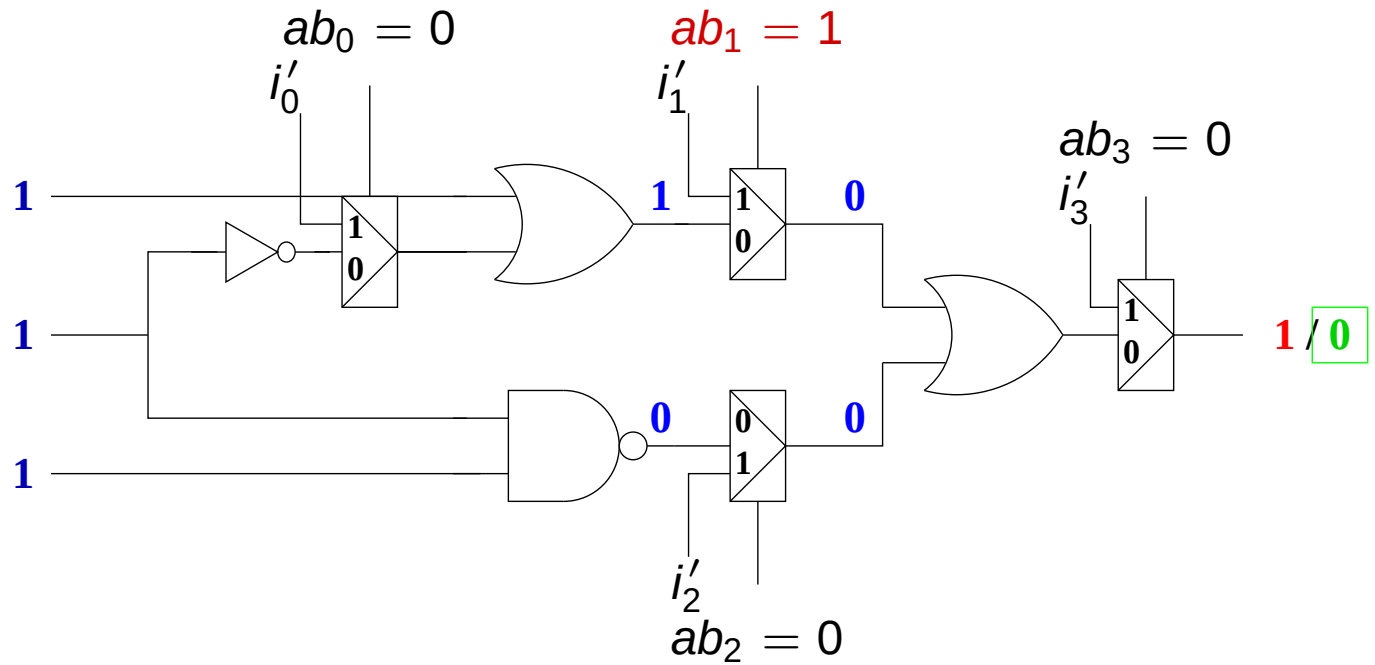


Classical SAT-based Diagnosis

Process

Given a model M and a set of failure traces

- Add multiplexers to force a gate to behave abnormally
- Use a SAT solver to solve the problem

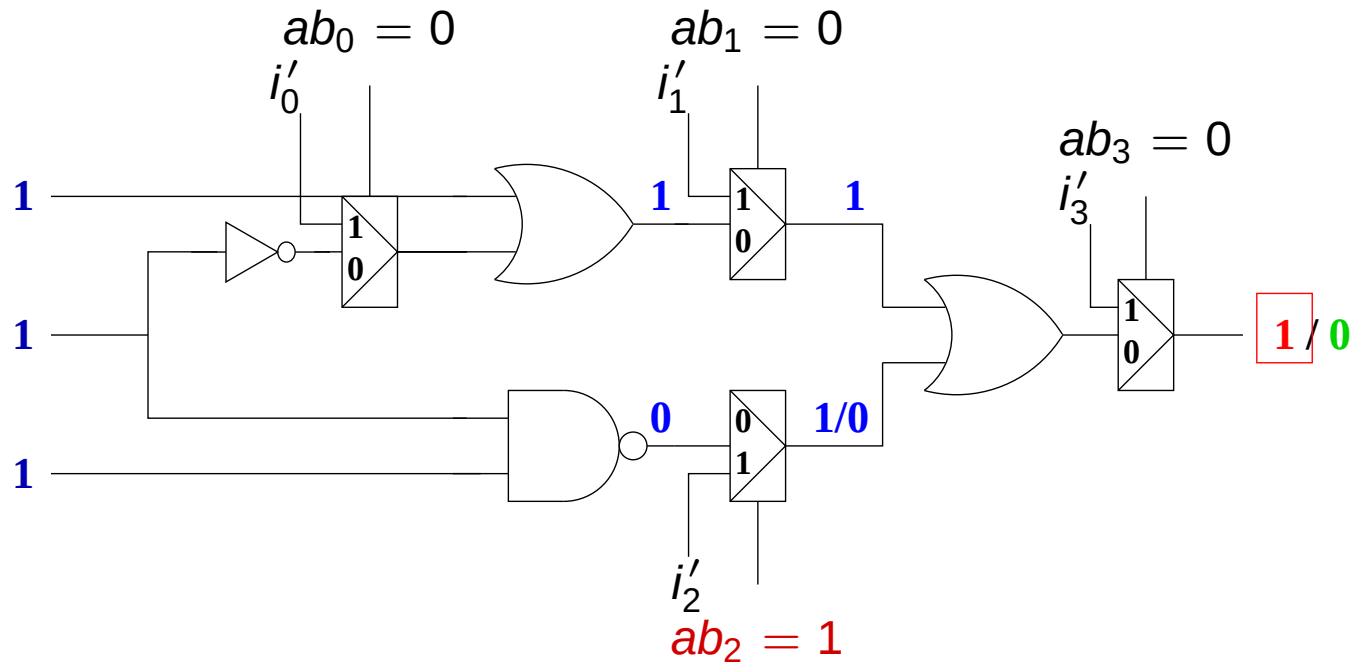


Classical SAT-based Diagnosis

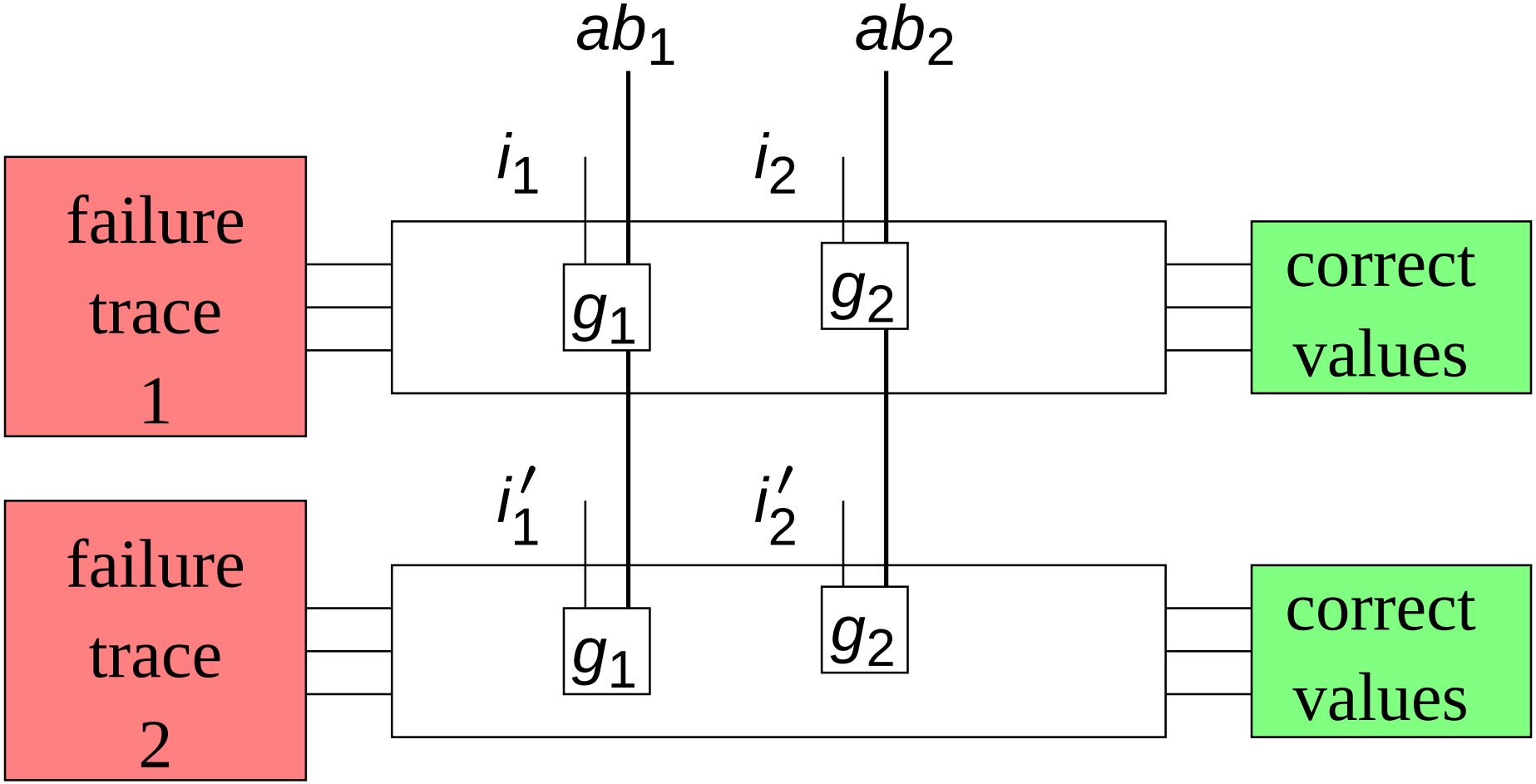
Process

Given a model M and a set of failure traces

- Add multiplexers to force a gate to behave abnormally
- Use a SAT solver to solve the problem



Diagnosis with multiple failure traces

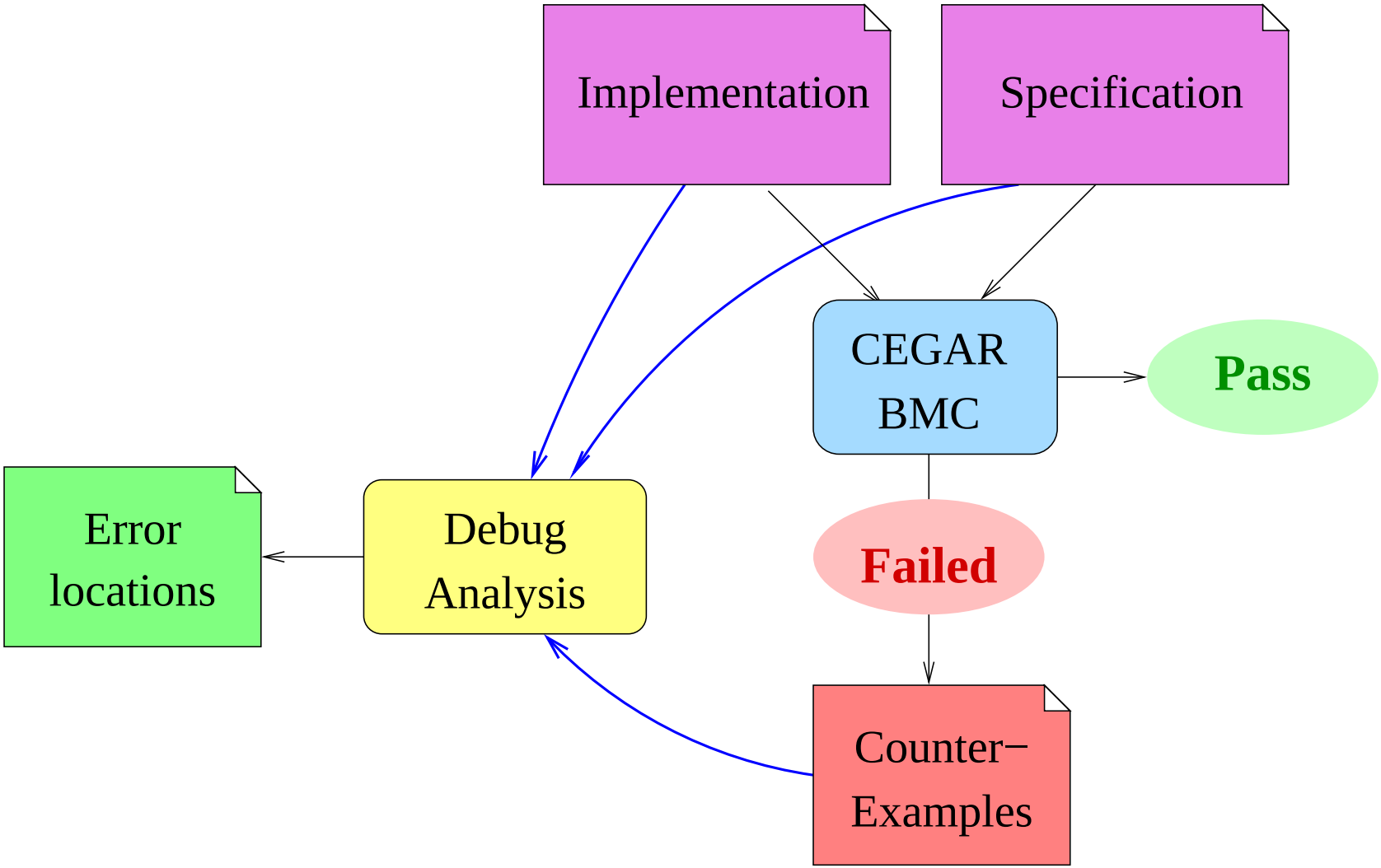


Properties of SAT-based Diagnosis

Properties

- Efficient SAT-solvers
- Handle multiple error locations
- Ensure the minimal cardinality of the solutions
- More efficient with many test vectors
- SAT-solution $\leftrightarrow$ Values to correct the test vectors

Our Debug flow



CEGAR SAT-based Diagnosis

Problem

Memory for large sequential design and long test vectors

➢ Abstraction-Refinement framework [Safarpour and Veneris 2007]

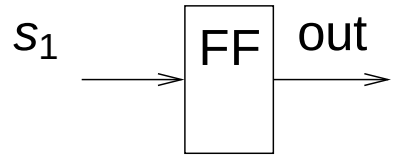
Principle

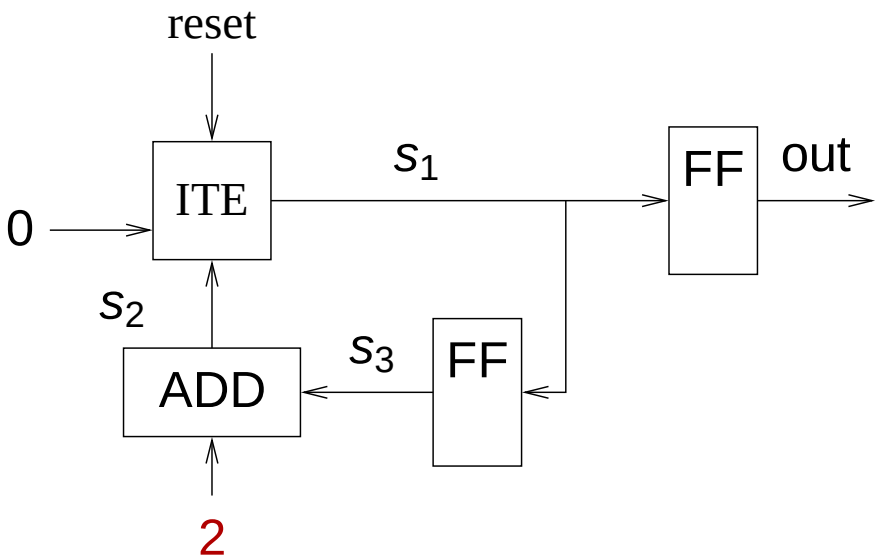
- Start with an abstract model (localization reduction)
- if abstracted elements are a part of the solutions : refine

Properties

- Correctness : solutions on the abstract model are solutions on the concrete one
- Completeness : no error sources remain undetected for a given set of test vectors

CEGAR SAT-based Diagnosis





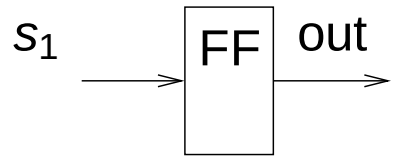
Property :

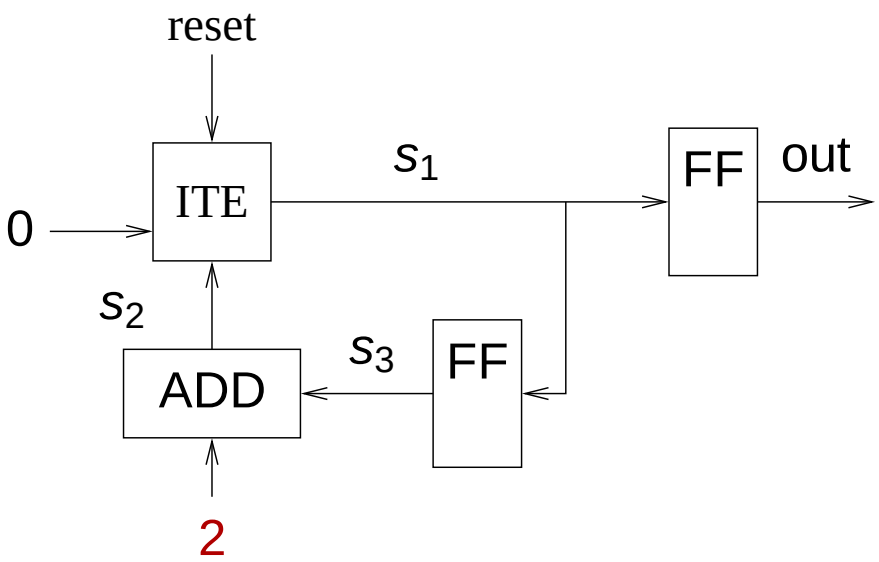
$$\text{out} = 3 \rightarrow \text{next(out)} = 0$$

Failure trace :

t	reset	s_1	s_2	s_3	out
0	0	1	1	3	3
1	0	1	3	3	1

CEGAR SAT-based Diagnosis





Spurious solution

(s_1) is an abstracted signal

➢ Refine !

Property :

$$out = 3 \rightarrow next(out) = 0$$

Failure trace :

t	reset	s_1	s_2	s_3	out
0	0	1	1	3	3
1	0	1	3	3	1

First Experiments

Model

Small processor : 128K data memory,4k instruction memory,16 bit data-path,8 registers,32 instructions

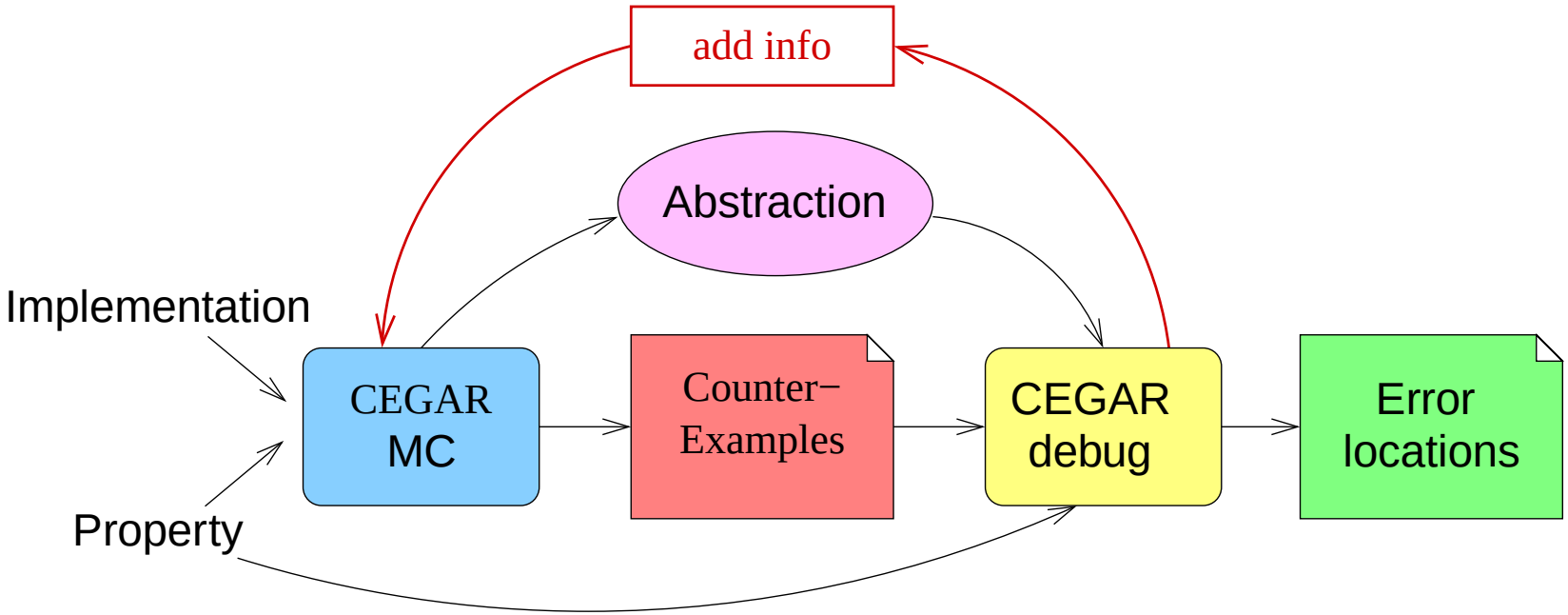
Property : G($reset \rightarrow \forall reg_i$, next($reg_i$) = 0)

		Classical Debug	CEGAR Debug		
# err	# loc.	time	abst.	time	reduc. mem
1	3	1.26s	85%	0.29s	60%
2	9	2.11s	62%	1.05s	40%

Next Step in Diagnosis for Property Checking

Diagnosis for property checking

- Find a complete correction for a given property
- Do not limit to a set of counter-examples
- Use the diagnosis result to find new counter-examples



Problem remains

Major issues

- Too many counter-examples
- Not always enough difference between counter-examples (bit vectors)

→ Find a more compact representation for a set of counter-examples (multi-valued)

→ Block more counter-example at each step

Conclusion : Post-Doc

Implementations

- CEGAR loop for bounded Model checking
- CEGAR loop for SAT-based diagnosis

→ Larger designs can be handled

Future work

CEGAR-based fault localization

- Improvement of the new CEGAR-diagnosis framework
- Start a cooperation with the University of Bremen

Conclusion : PhD thesis

Abstraction Algorithm

Transformation of CTL formulas into Kripke structure for the verification by BDD-Model checking

- Abstract components by a subset of their specification
- Obtain an abstraction directly from a syntactic analysis of the formulas

Results :

- → Reduction of the explored tree depth
- → Reduction of the number of BDD nodes

Research project

CEGAR framework for mixed abstraction

- Many different types of abstraction possible (predicate abstraction, localization reduction, property abstraction ...)
- Some abstractions are more suitable for :
  - hardware or software
  - a given architecture (memory, pipeline ...)
  - a given description level (bit, word ...)

→ Unified abstraction-refinement framework for mixed abstraction

Issues

- Counter-example analysis with mixed abstraction
- Refinement process
- Level of automation

Research project

Applications

- Property Checking
- Fault Localization
- Source code annotation ...

- Embeded Design
- Communication protocols
- Automotive units, Railway systems ...