State-of-The-Art Formal Property Verification Technologies for IC Designs

Pei-Hsin Ho
Advanced Technology Group
Synopsys, Inc.

Outline

- Formal property verification basics
- Modern formal property verification engines
- Hybrid proof and disproof methods
- State-of-the-art formal property verification tools

Outline

- *Formal property verification basics*
  - Methodology
  - Terminology
  - Formalism
- Modern formal property verification engines
- Hybrid proof and disproof methods
- State-of-the-art formal property verification tools

Formal Property Verification

- Inputs:
  - Design under verification (DUV)
    - IC design in RTL Verilog or VHDL
    - Initial states or initialization sequences
  - Assertion about the DUV
    - Monitor in RTL Verilog or VHDL
    - Property in property languages like OVA, PSL or SVA
  - Assumption about the environment of the DUV and/or the DUV
    - Monitor in RTL Verilog or VHDL
    - Property in property languages like OVA, PSL or SVA

OVA: Open Vera Assertion language; PSL: Property Specification Language; SVA: SystemVerilog Assertion language

Formal Property Verification (cont.)

- Outputs:
  - Falsified (bug is found)
    - Counter example
      - Error trace for debugging
      - Best outcome for verification engineers
    - Simulator ready or artificial VCD file
  - Proven
    - DUV satisfies the assertion against all input stimuli starting from the initial states under the assumptions
  - Inconclusive
    - Bounded proof of finite length
    - Some coverage results

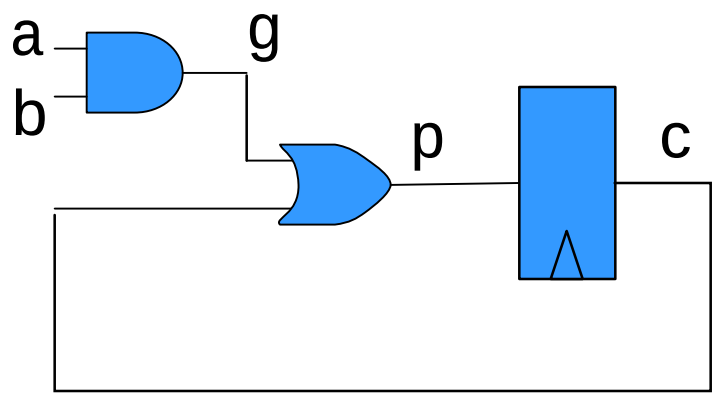
Design: RTL to Gate

- Verilog or VHDL RTL

- input a,b; output c;
  reg c; wire p, g;

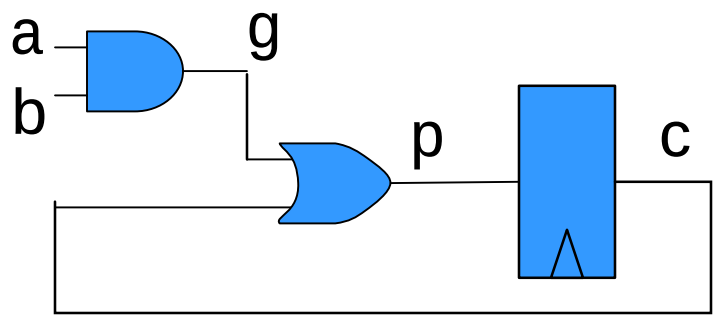
  assign g = a && b;
  assign p = g || c;
  always @ (posedge clk)
  c <= p;

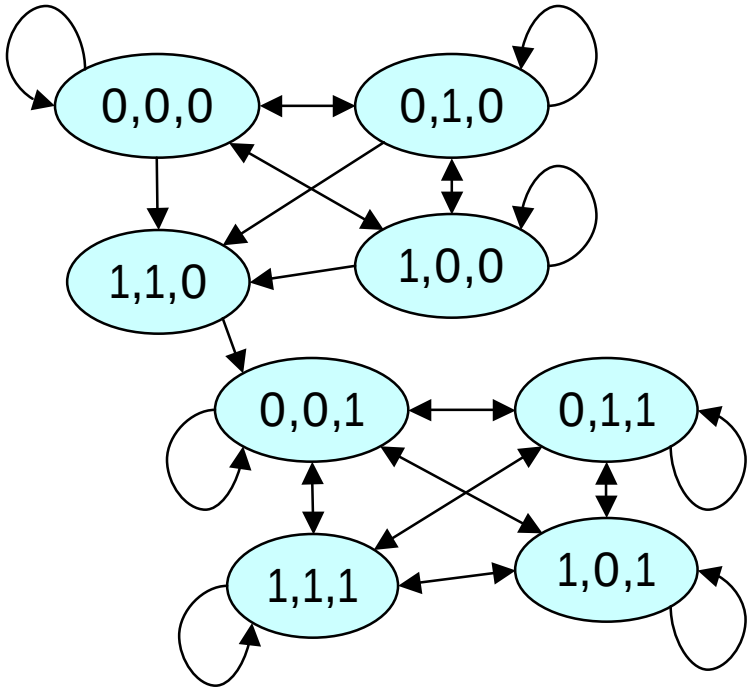
- Logic synthesis converts RTL to gate-level netlist



Design: Gate to State Transition Graph

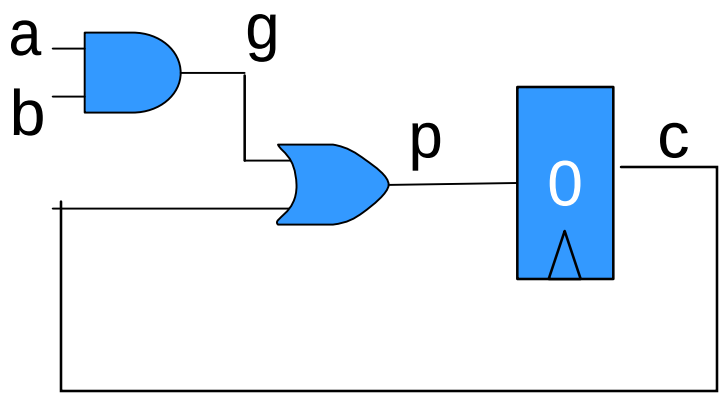
- State: assignment to the inputs and registers
  - (a=0, b=0, c=1), or (0,0,1)
- Transition: from state s1 to state s2 iff the design can go from state s1 to state s2 in 1 clock cycle
  - (1,1,0) -> (0,0,1)
- Finite state machine
- State transition graph:
  - Vertices: states
  - Arcs: transitions





Design: Initial States and Traces

- Initial states I: set of states from which the design start normal execution
  - Reset states
- User provides initialization sequences (reset sequences) or initial states in HDL, HVL or VCD dump
- Semantics of design
  - Set of traces
  - All finite and infinite paths (traces) from the initial states in the state transition graph



Properties

- Property
  - Statement about the design
    - Example: Whenever there is a request, there will be a grant in the next clock cycle
  - Set of traces
- Design satisfies the property if
  - Traces of design, T(D), is a subset of the traces of the property, T(p)
- Counter example (error traces)
  - Traces that are in T(D) but not in T(p)
  - Minimal error traces: error trace t is minimal if no prefix of t is an error trace

Properties (cont.)

- Safety properties
  - Properties whose minimal error traces are finite
  - Example: Whenever there is a request, there will be a grant in the next clock cycle
- Liveness properties
  - Properties whose minimal error traces are infinite
  - Example: Whenever there is a request, there will be a grant eventually
- Every property is a safety property, a liveness property or a conjunction of the two
- Will focus on safety properties in this tutorial

Safety Properties

- Properties in property languages
  - Temporal formulas
    - if request then #1 grant
- Monitors in RTL Verilog or VHDL
  - Designs that monitor the behavior of the DUV and assert a "bad" signal if and only if the DUV violates the property
  - always @(posedge clk or negedge rst)

```
if (!rst) begin   pre_req <= 0;
                  bad <= 0;
end
else begin        pre_req <= request;
                  bad <= pre_req && !grant;
end
```

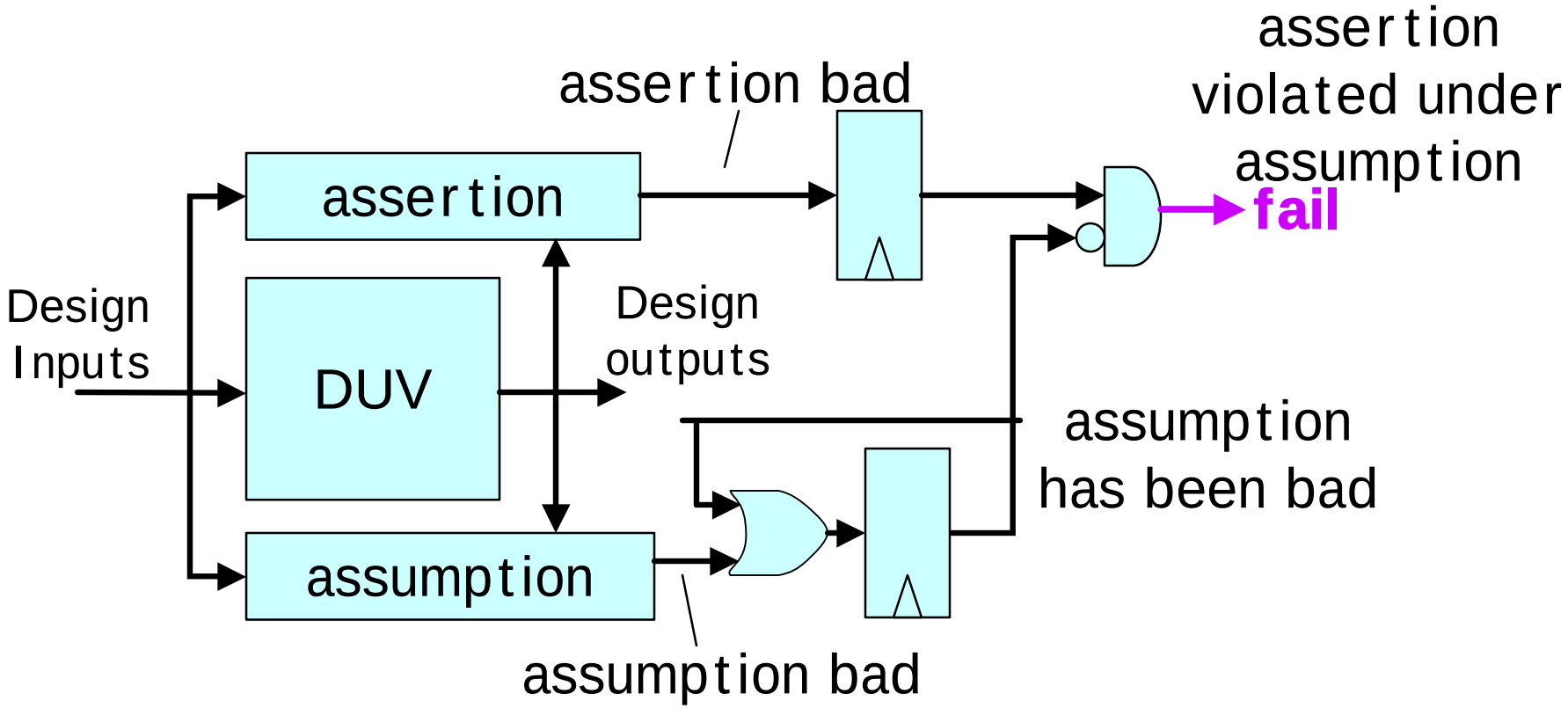
- Safety properties can be automatically converted into RTL monitors

Assertions and Assumptions

- Assertion
  - Properties that we check to see if the DUV would satisfy during the verification
- Assumption
  - Properties that we want to assume to be true for the DUV or the environment of the DUV during the verification

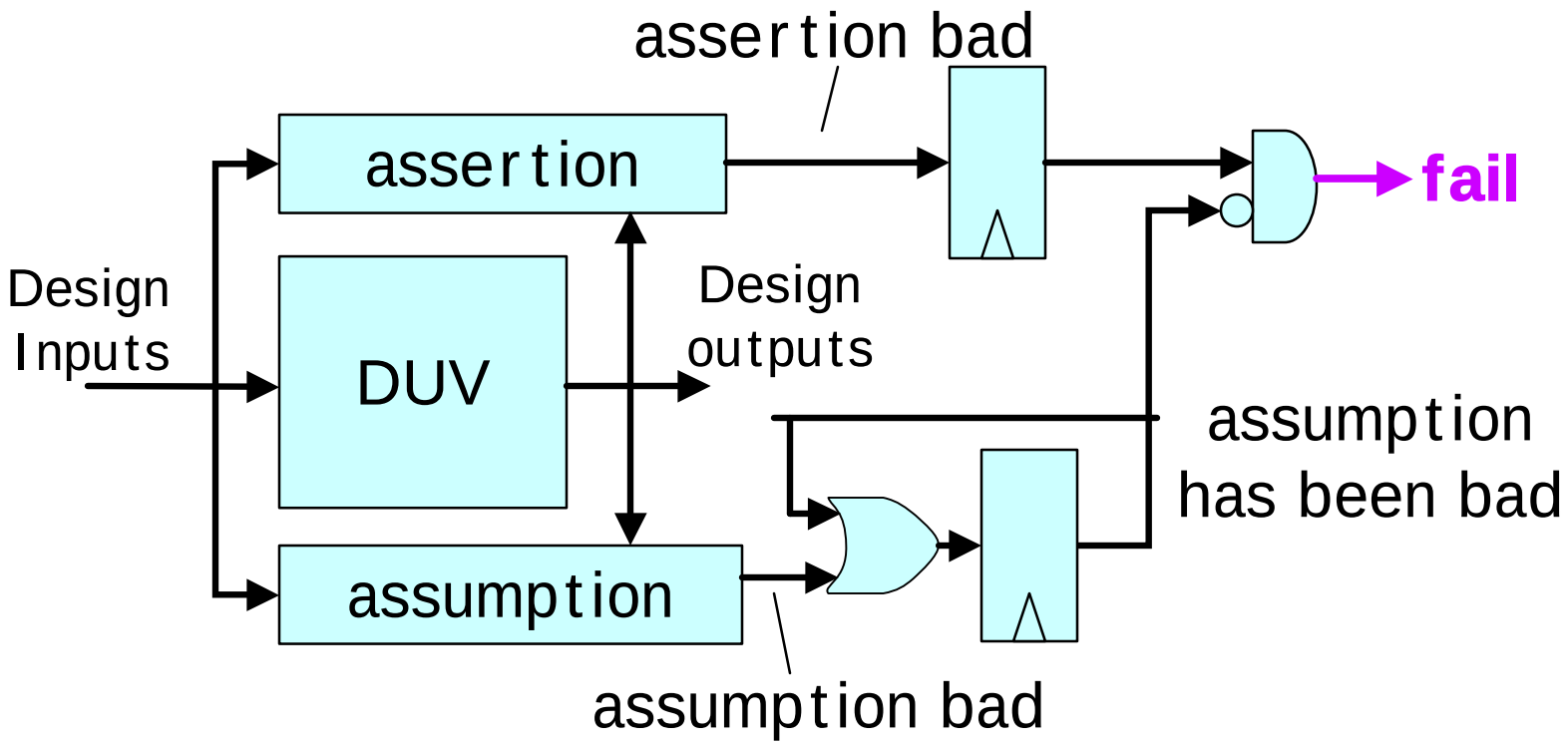
FV Model Under Verification

- DUV, assumption and assertion constitute the model under verification with an output that asserts iff the assertion is violated under the assumption



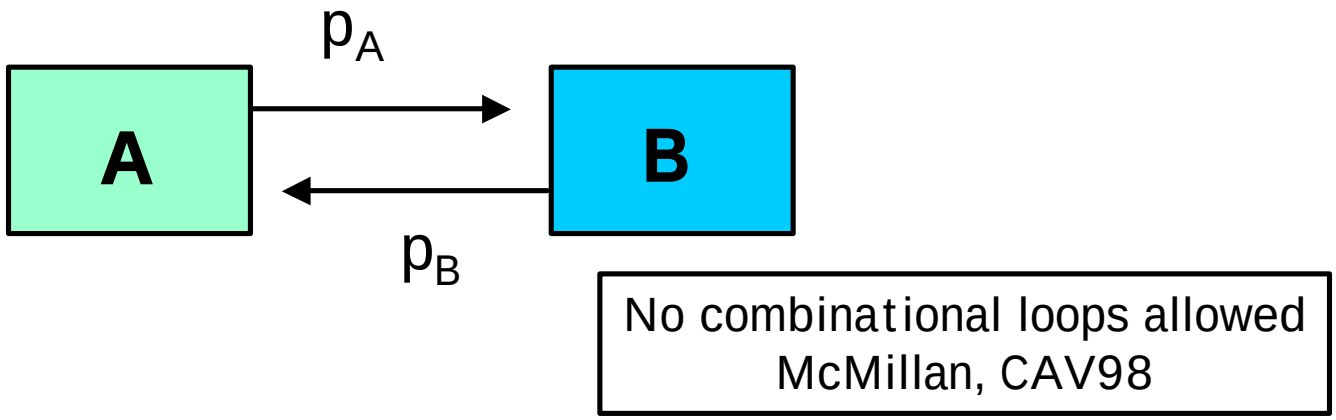
Reduction to Reachability Problem

- Formal property verification problem is reduced to a reachability problem on the state transition graph:
  - Are the fail states (F) reachable from the initial states (I)?



Assume Guarantee Reasoning

- Properties can be used as either assertions or assumptions
- Assume guarantee reasoning
  - DUV is block A
    - assert p_A; assume p_B
  - DUV is block B
    - assert p_B; assume p_A



Bibliography (1/3)

- K.L. McMillan, "Verification of an implementation of Tomasulo's algorithm by compositional model checking," CAV98
- OpenVera Assertions (OVA), http://www.open-vera.com/
- PSL/Sugar, http://www.haifa.il.ibm.com/projects/verification/sugar/psl.html
- SystemVerilog Assertions (SVA), http://www.eda.org/sv-ac/

Outline

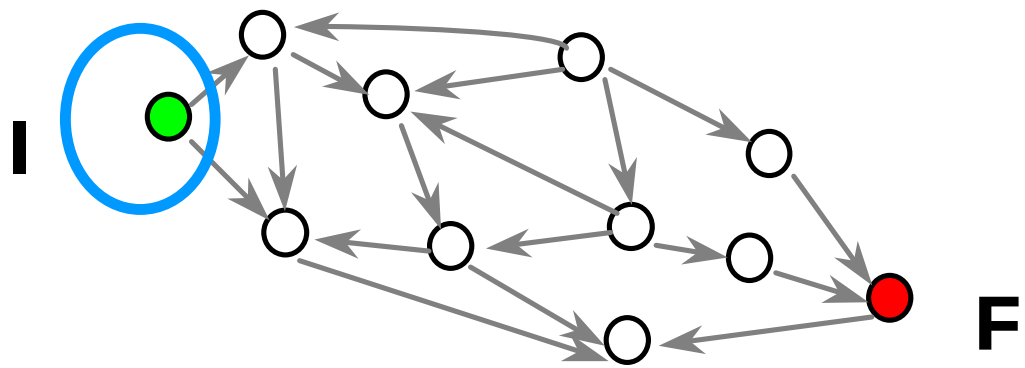
- Formal property verification basics
- *Modern formal property verification engines*
  - Random simulation
  - Reachable state set approximation
  - Symbolic simulation
  - SAT
  - ATPG and ATPG-SAT hybrid
  - Inductive proof
  - Structural subset-based abstraction
- Hybrid proof and disproof methods
- State-of-the-art formal property verification tools

Random Simulation

- Most effective engine for “simple” assertions
  - Quickly find mistakes in model under verification
    - Assertions, assumptions, initialization
  - Early design phase
    - Assumption/assertion-based random simulation sometimes can be built faster than conventional simulation testbenches in HDL/HVL
- Random simulation with assumptions
  - At each clock cycle, find an input vector that satisfies all assumptions
  - Combinational assumptions
    - Yuan,Shultz,Pixley,Miller,Aziz, ICCAD99
  - Sequential assumptions
    - Paper WIP

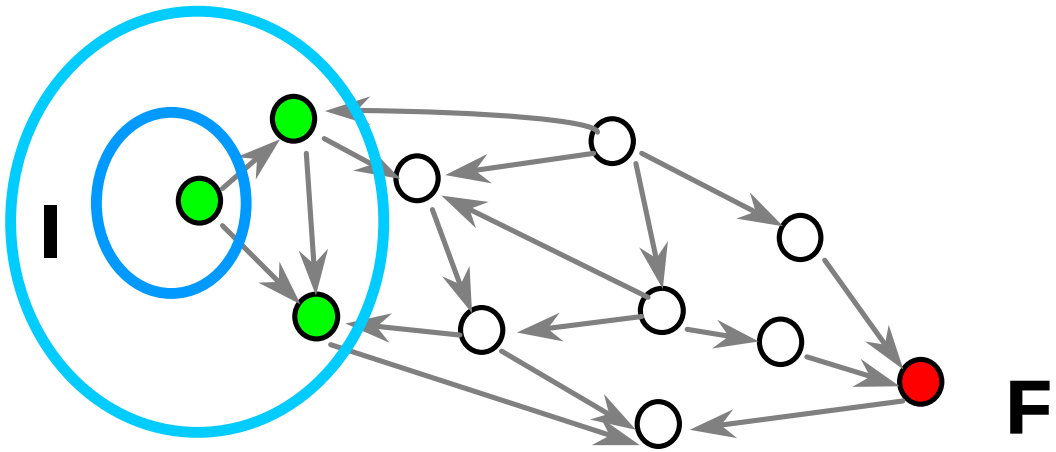
Symbolic Reachability Analysis

- Fixpoint computation
  - State transition graph
  - Starting from the initial states I



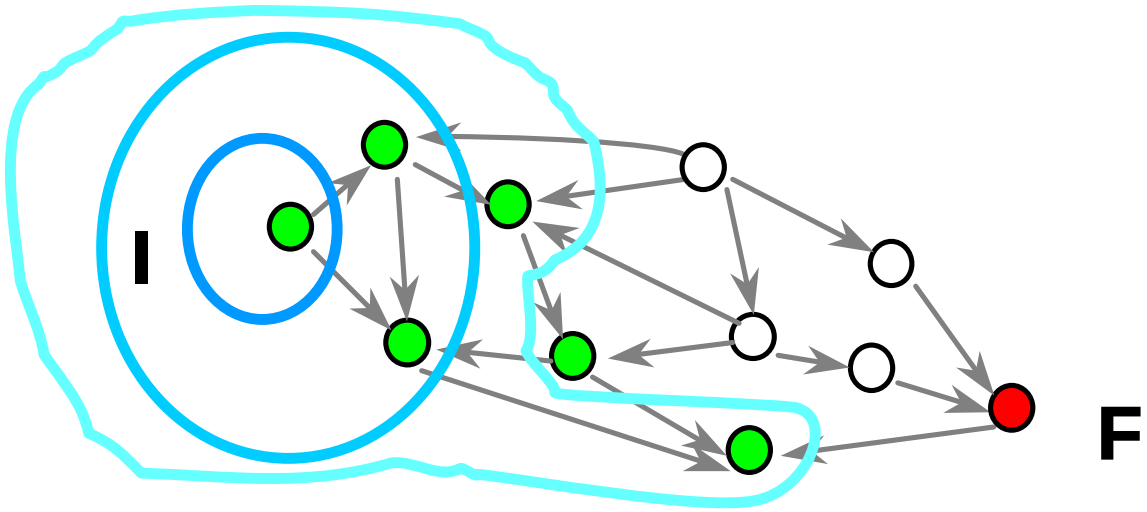
Symbolic Reachability Analysis

- Fixpoint computation
  - Compute all states that are reachable in 1 clock cycle
  - Image computation (forward)



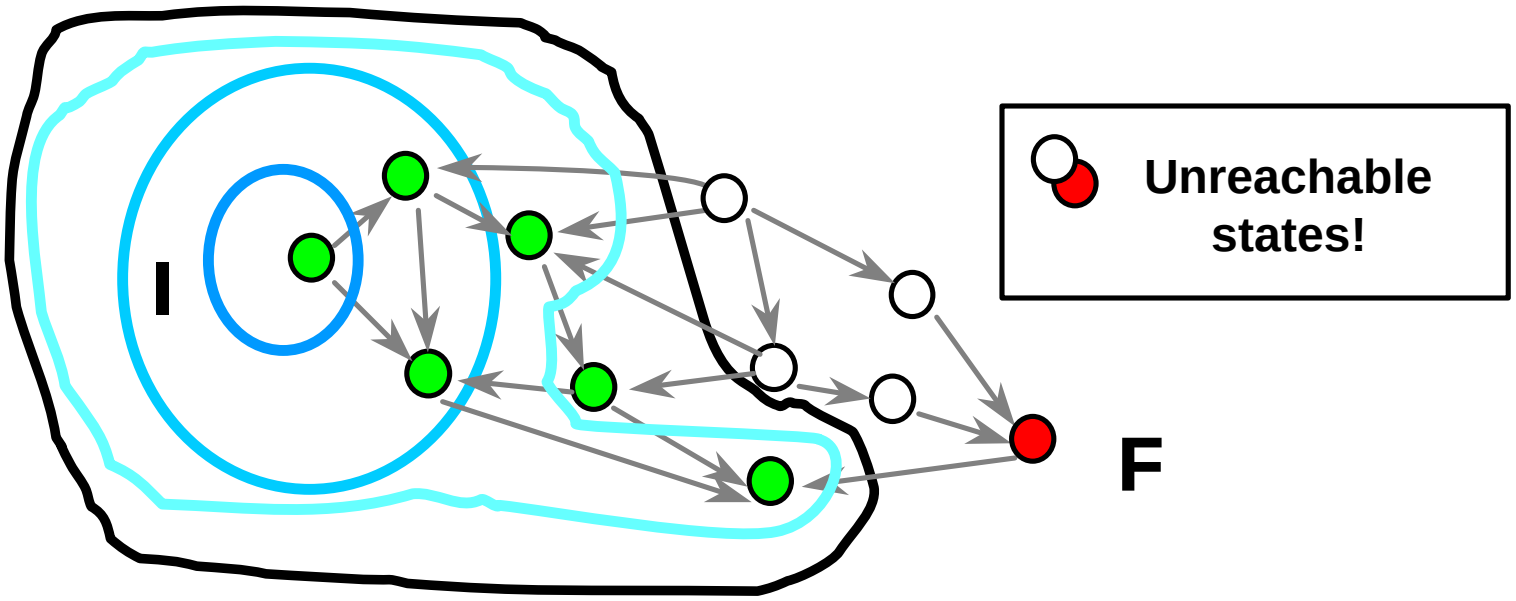
Symbolic Reachability Analysis

- Fixpoint computation
  - Compute all states that are reachable in 1 clock cycle
  - Image computation (forward)



Symbolic Reachability Analysis

- Fixpoint computation
  - Reached a fixpoint!
  - Identified unreachable states



Reachable States in Characteristic Representation

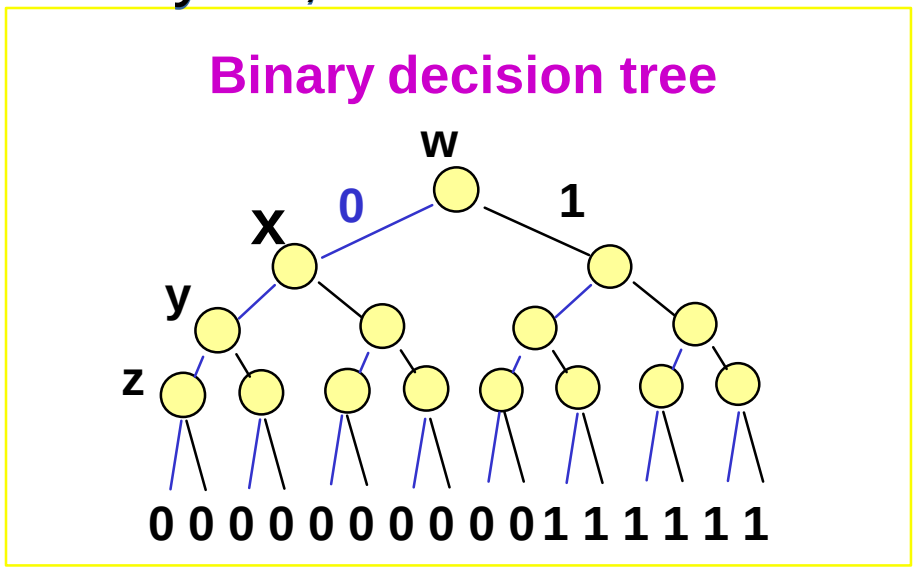
- Characteristic representation of sets of states and the model under verification
  - State set S is represented as a Boolean function R: $2^{|V|}$ -> B such that
    - State s is in the set S iff R(s)==1
    - V: variables (inputs and registers)
- Example:
  - State set {(x=0,y=1,z=0), (0,1,1), (1,0,1), (1,0,0)}
  - Characteristic representation: !(x && y) && (x||y)

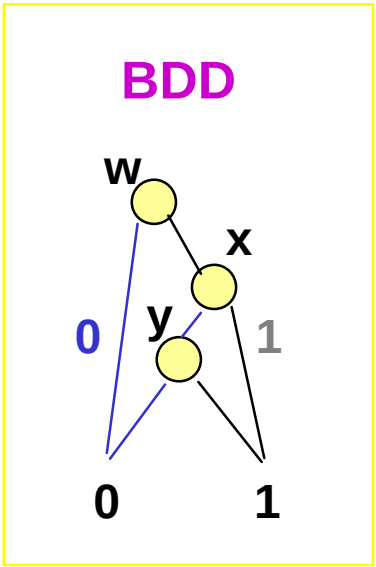
Characteristic Reachability Analysis Methods

- Methods
  - BDD represents both the state set and model
    - Burch,Clarke,McMillan,Dill,Hwang, LICS1990
  - Clauses represent both the state set and model
    - McMillan, CAV02 ✍ details in Ken's session
  - State set: interpolants; Model: clauses
    - McMillan, CAV03 ✍ details in Ken's session

Binary Decision Diagram (BDD)

- Compact and canonical data structure for manipulating Boolean functions
  - Example: 4 variables w, x, y and z
    Boolean function: w & (x | y) & (z | !z)
  - Bryant, TCAD86





Characteristic Fixpoint Computation

- Fixpoint computation
  - C(V,V'): set of state transitions
  - $R_0 = I$; $R_{i+1} = R_i \cup Img(R_i,C)$
  - When $R_{i+1}==R_i$; F is unreachable iff $(F \cap R_i)==\emptyset$
- Boolean operators for characteristic representation
  - Union == disjunction $\vee$
  - Intersection == conjunction $\wedge$
  - $Img(R_i,C) == \exists V'. \exists V. (R_i \wedge C(V,V'))$
    - V: variables (registers and inputs)
    - C(V,V'): the set of state transitions (transition relation)
    - $R_i(V)$: the set of states
    - Existential quantification: $\exists V$

Reachable States in Parametric Representation

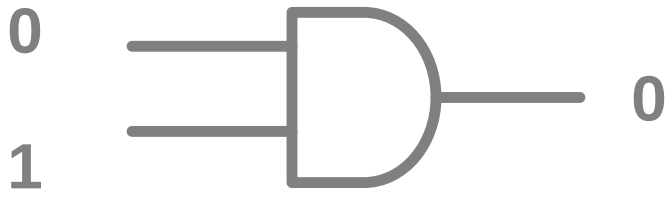
- Parametric representation of sets of states
  - State set S is represented as an array of Boolean functions $(f_0,\ldots,f_n)$ such that
    - Each f_i: $2^{|u|}$ -> B
    - State $s=(s_0,\ldots,s_n)$ is in the set S iff there exists an assignment to the variables in U such that $(s_0,\ldots,s_n) == (f_0,\ldots,f_n)$
  - Example:
    - State set {(x=0,y=1,z=0), (0,1,1), (1,0,1), (1,0,0)}
    - Parametric representation: (a, !a, b)

Parametric Reachability Analysis Methods

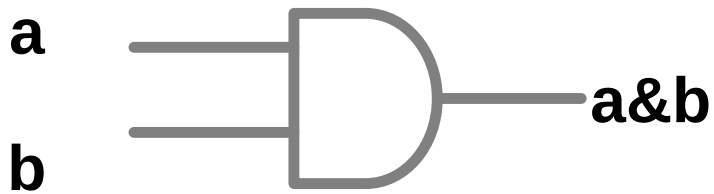
- Methods
  - State set: BDD; Model: gate-level netlist
    - Goel,Bryant, DATE03
    - Parametric union, intersection, quantification

Symbolic Simulation

- Conventional (scalar) simulation
  - Propagate constants

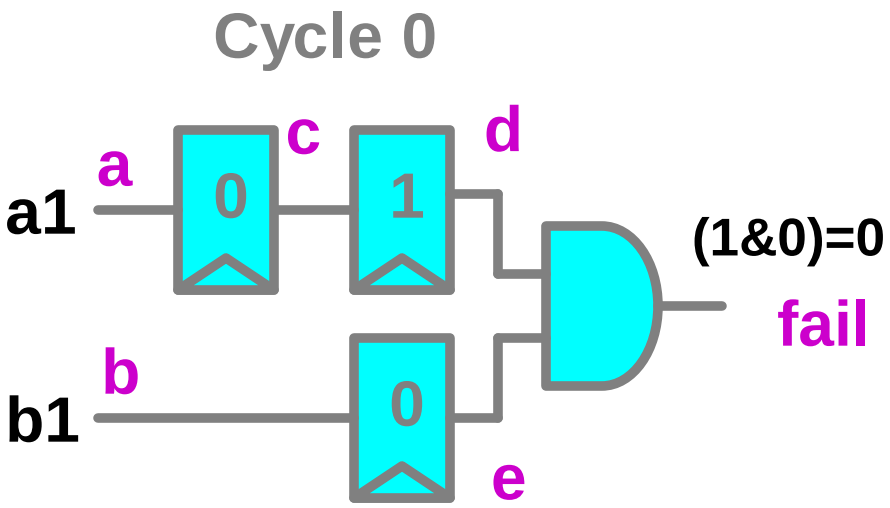


- Symbolic simulation
  - Propagate parametric symbolic functions



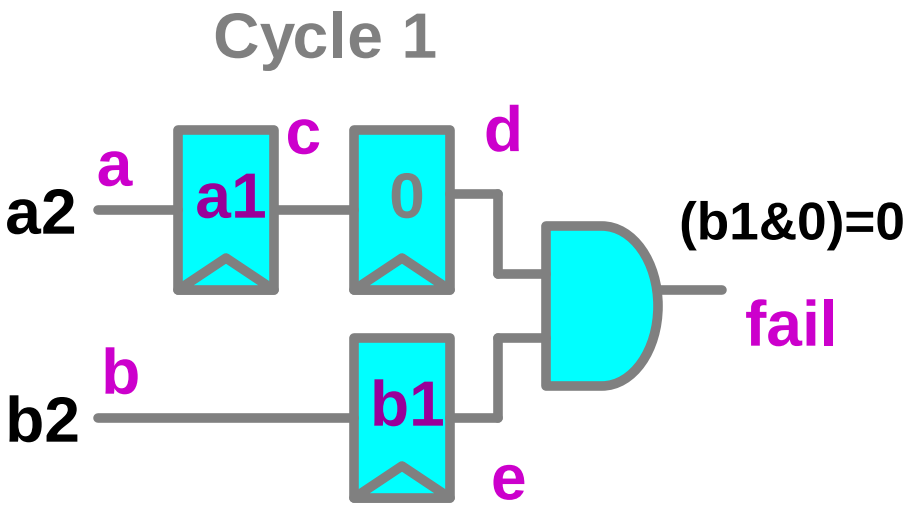
Symbolic Simulation: Example

- Initial state: !c & d & !e
- Bad states: fail ==1



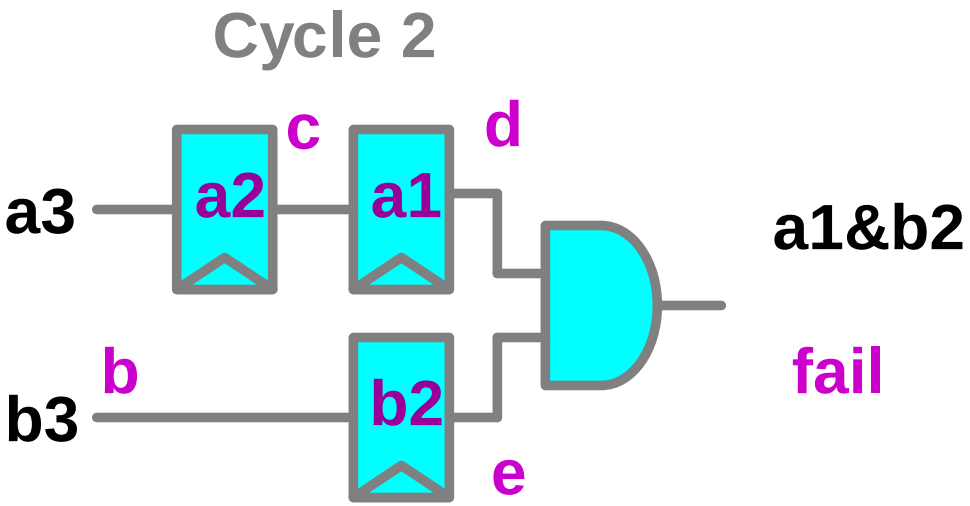
Symbolic Simulation: Example

- Symbolic simulation after 1st cycle
  - Verified that the fail state not reachable in 1 cycle



Symbolic Simulation: Example

- Hit the fail state (fail==1 iff a1&b2 is 1)!
  - Generated input sequence
    - @0, a=1
    - @1, b=1



Symbolic Simulation

- Applications
  - Disproof
  - Proof
- Recent work on disproof
  - Set input variables to constants when BDD is getting too big
    - Which input variable to under-approximate?
  - Re-parameterization (to make the BDD smaller)
    - Bertacco,Olukotun, DAC02
    - Kwak,Moon,Kukula,Shiple, ICCAD02
  - Approximate values and case splitting
    - Wilson,Dill,Bryant, FMCAD02
  - Handle embedded memories efficiently
    - Kölbl,Kukula,Antreich,Damiano, DAC02

Symbolic Simulation

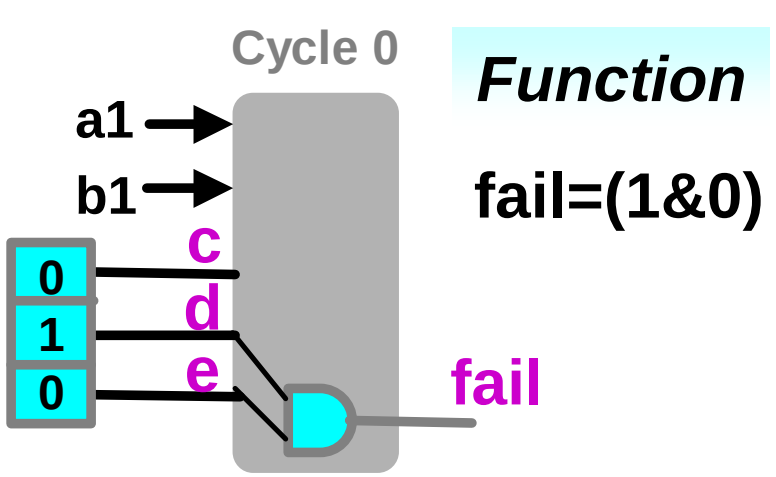
- Recent work on proof
  - Parametric symbolic reachability analysis as mentioned earlier
    - Goel,Bryant, DATE03
  - Generalized Symbolic Trajectory Evaluation (STE)
    - Yang,Seger, FMCAD02
    - Manual approximation/refinement is required

SAT (Satisfiability)

- Given Boolean function $f(x_1, x_2, \ldots, x_n)$
- If it is possible to find assignment $(a_1, a_2, \ldots, a_n)$ such that
  - $f(a_1, a_2, \ldots, a_n) = 1$?
- Ken's session will focus on this area of many recent breakthroughs

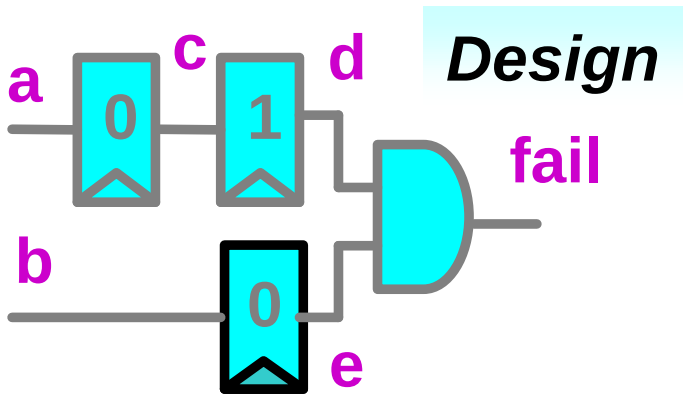
SAT: Example

- Initial state: !c & d & !e
- Bad (goal) states: fail ==1
  - Convert problem into a function: fail
  - Find input assignment so that fail is 1
    - No BDDs



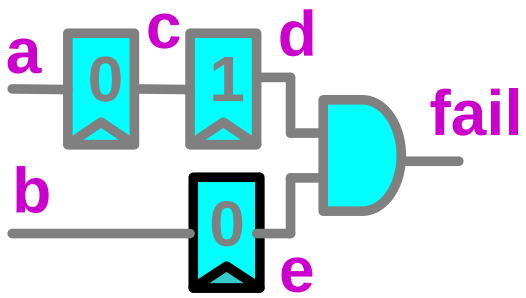
Function

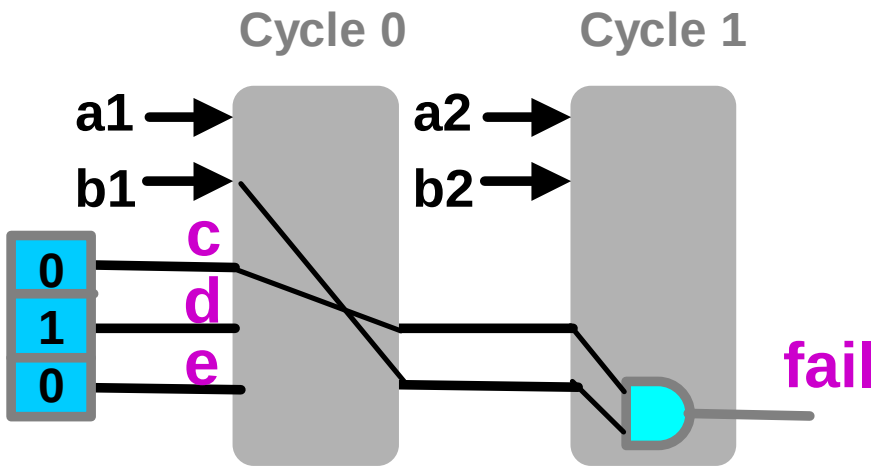
fail=(1&0)



SAT: Example

- SAT after 1st unroll
  - Verified goal state not reachable in 1 cycle

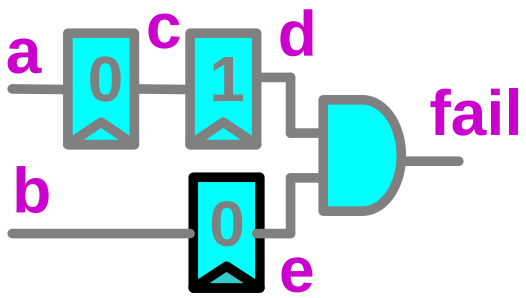


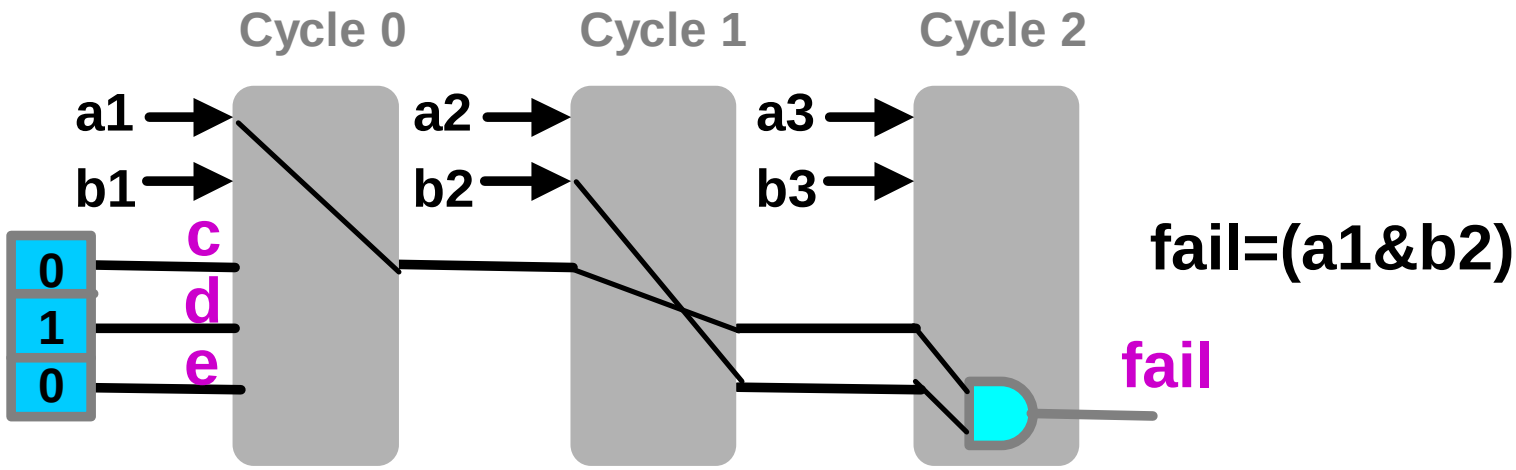


fail=(0&b1)

SAT: Example

- SAT after 2nd unroll
  - Find assignment to inputs a1=1 and b2=1 make fail=1
  - Find input sequence



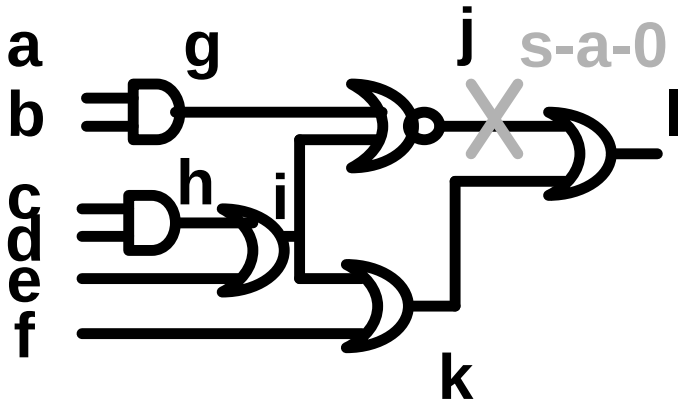


SAT (Satisfiability)

- Applications
  - Proof
  - Disproof
  - Ken's session

ATPG (Automatic Test Pattern Generation)

- Original problem statement
  - Wire of a circuit is stuck at 0 or 1
  - Generate test so that
    - Outputs of the good circuit and the faulty circuit differ
  - Justification
    - Generate a test that causes an opposite value at stuck-at wire
  - Propagation
    - Propagate the difference at the stuck-at wire to the output

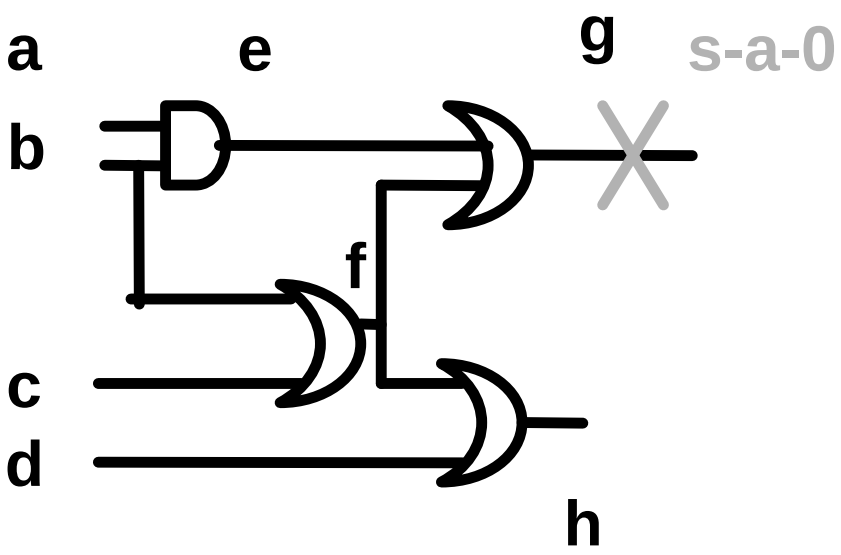


ATPG (cont.)

- Formal property verification only needs justification of the bad value
  - If it is possible to find assignment $(a_1,a_2,\ldots,a_n)$ to the inputs $x_1,x_2,\ldots,x_n$ such that y = 1?
- Sequential ATPG
  - Does not explicitly unroll netlist
- Circuit topology determines decision-making schedule

ATPG Justification: Example

- ATPG value systems
  - FV 3 value: 0, 1, X



a	b	c	d	e	f	g
						1
				1	X	1
1	1			1	1	1

Comparison of Conventional ATPG and SAT technologies

- Iyer,Parthasarathy,Cheng, ICCAD03:

Feature	SAT	ATPG	Winner
Conflict-based learning	Yes	Minimal	SAT
Efficient implications	Yes	No	SAT
Structural information	Some	Yes	ATPG
Decision strategy	Appearance in clauses	Topology	sat: ATPG unsat: SAT
Algorithm complexity	Low	High	SAT
Number of sat. assignments	High	Low	ATPG
Unrolling for sequential problems	Explicit	Implicit	ATPG

Hybrid ATPG-SAT Solver

- SATORI [Iyer,Parthasarathy,Cheng, ICCAD03]
  - ATPG decision strategy
  - ATPG implicit unrolling
  - ATPG partial assignments
  - SAT conflict-based learning
  - SAT implication

Word-Level ATPG Solver

- RACE [Mahesh Iyer, ITC03]
- Solves word-level arithmetic (datapath) and Boolean constraints (control)

Inductive Prover

- Induction proof
  - Use SAT or ATPG
  - Will be covered in Ken's session

Bibliography (2/3)

- V. Bertacco, K. Olukotun, “Efficient state representation for symbolic simulation,” DAC2002, pp.99-110
- R.E. Bryant, “Graph-based algorithms for Boolean function manipulation,” IEEE TCAD, C-35(8), 1986
- J.R. Burch,E.M. Clarke,K.L. McMillan,D.L. Dill,J. Hwang, “Symbolic model checking:10^{20} states and beyond,” LICS1990
- A. Goel, R.E. Bryant “Set manipulation with Boolean functional vectors for symbolic reachability analysis,'' DATE2003
- M.K. Iyer, G. Parthasarathy, K.T. Cheng, “SATORI-a fast sequential SAT engine for circuits,” ICCAD03
- M. Iyer, “RACE: a word-level ATPG-based constraint solver system for smart random simulation,” ITC03
- A. KÖlbl, J. Kukula, K. Antreich, R. Damiano, “Handling special constructs in symbolic simulation,” DAC2002, pp.105-110
- H.H. Kwak, I.-H. Moon, J. Kukula, T.R. Shiple, “Combinational equivalence checking through function transformation,” ICCAD02, pp.526-533
- K.L. McMillan, “Interpolation and SAT-based Model Checking,” CAV2003
- K.L. McMillan, “Applying SAT methods in unbounded symbolic model checking,” CAV2003, pp.250-264
- C. Wilson, D.L. Dill, R.E. Bryant, “Symbolic simulation with approximate values,” *FMCAD 2000*, pp. 486-504
- J. Yuan,K. Shultz,C. Pixley, H.Miller, A. Aziz “Modeling design constraints and biasing in simulation using BDDs,” ICCAD1999, pp.584-589
- J. Yang, C.-J. Seger, “Generalized symbolic trajectory evaluation – abstraction in action,” FMCAD2002, pp. 70-87

Outline

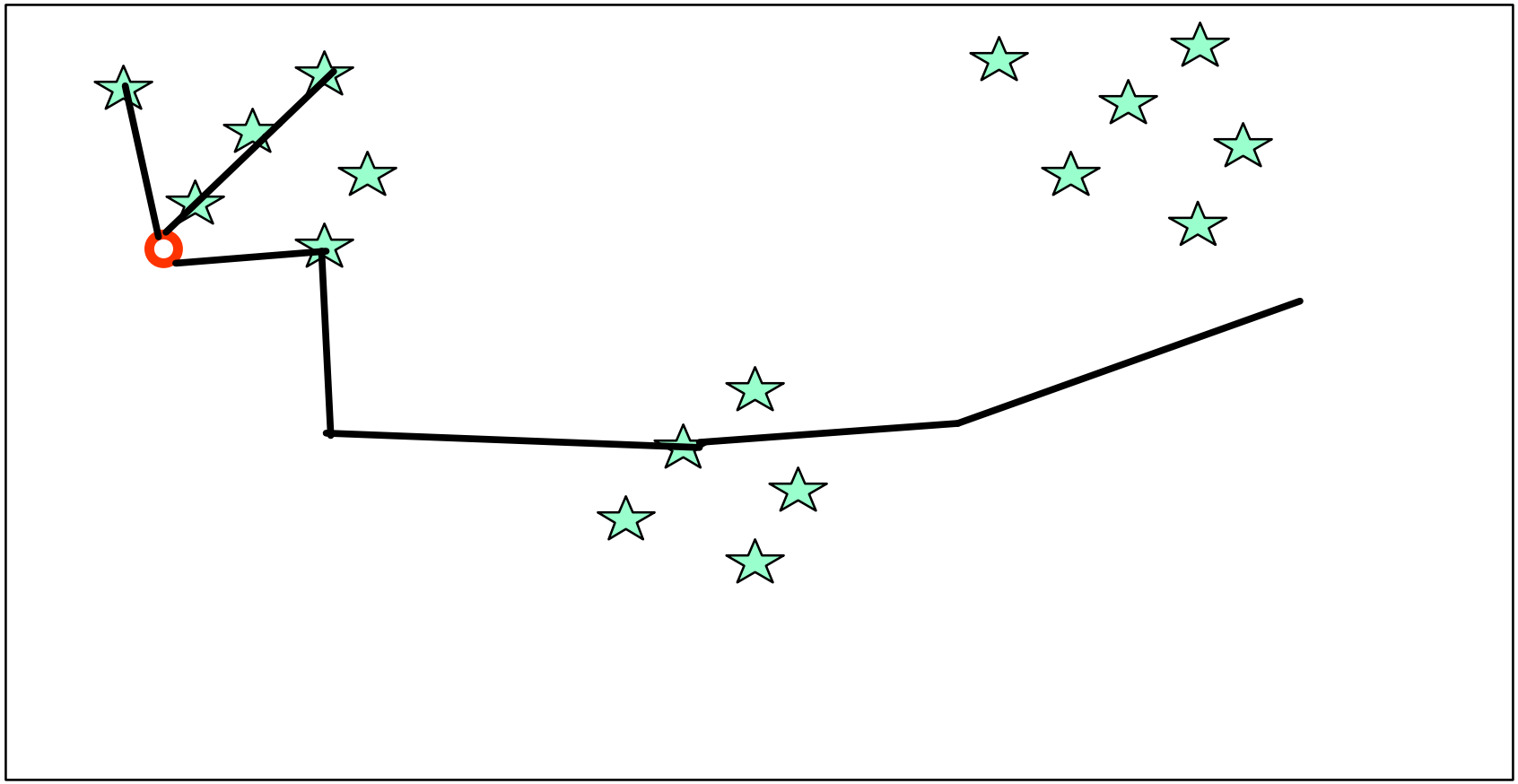
- Formal property verification basics
- Modern formal property verification engines
- *Hybrid proof and disproof methods*
  - Disproof
  - Proof
- Key ingredients for practical formal property verification tools

Hybrid Disproof Methods

- Orchestrate simulation and multiple formal disproof engines
- Iteratively run random simulation and formal engines from “deep” states
  - Ho,Shiple,Harer,Kukula,Damiano,Bertacco,Taylor,Long, ICCAD00

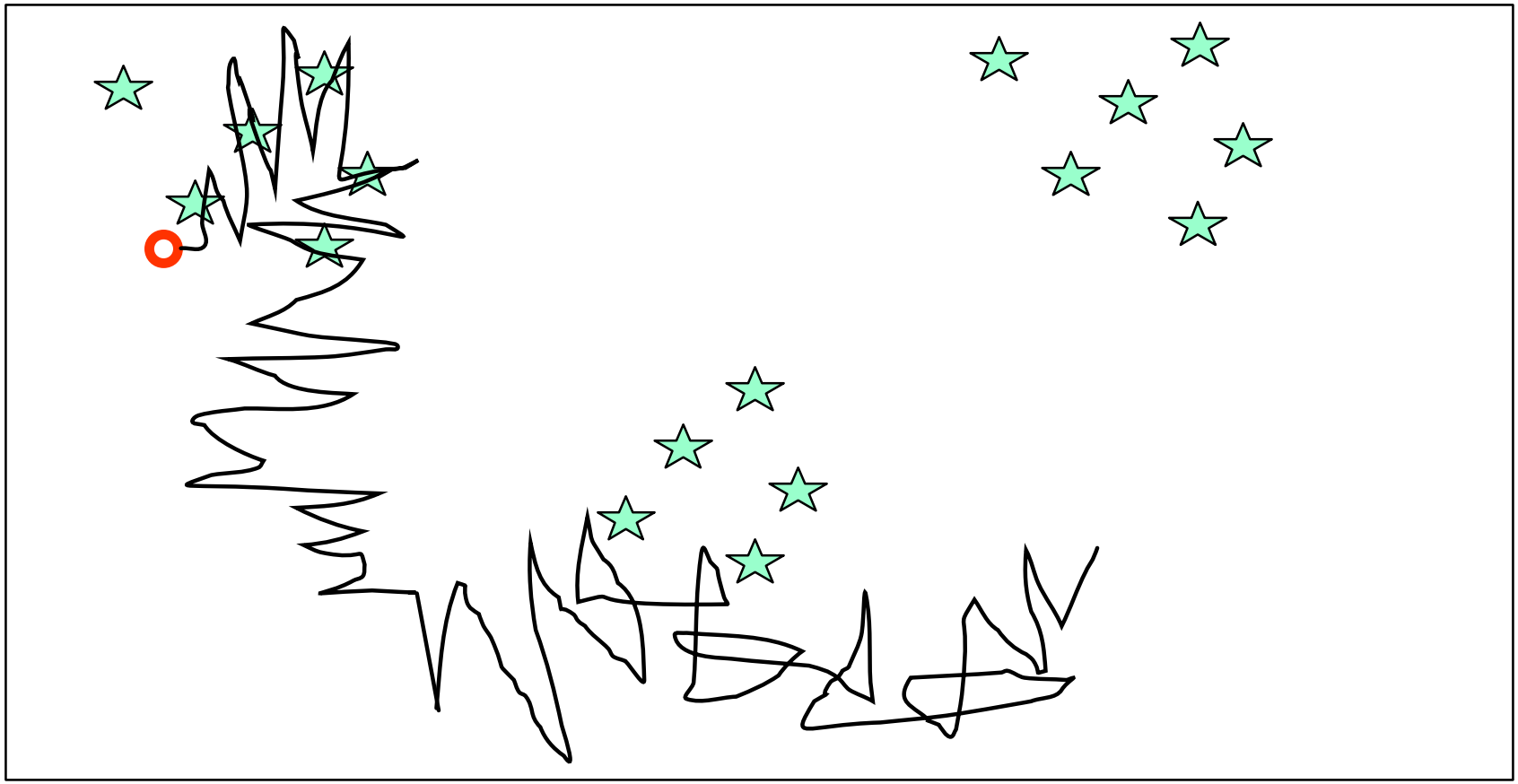
Directed Simulation

Manually create input vectors to drive the design --- SLOW, LOW COVERAGE, MISS BUGS



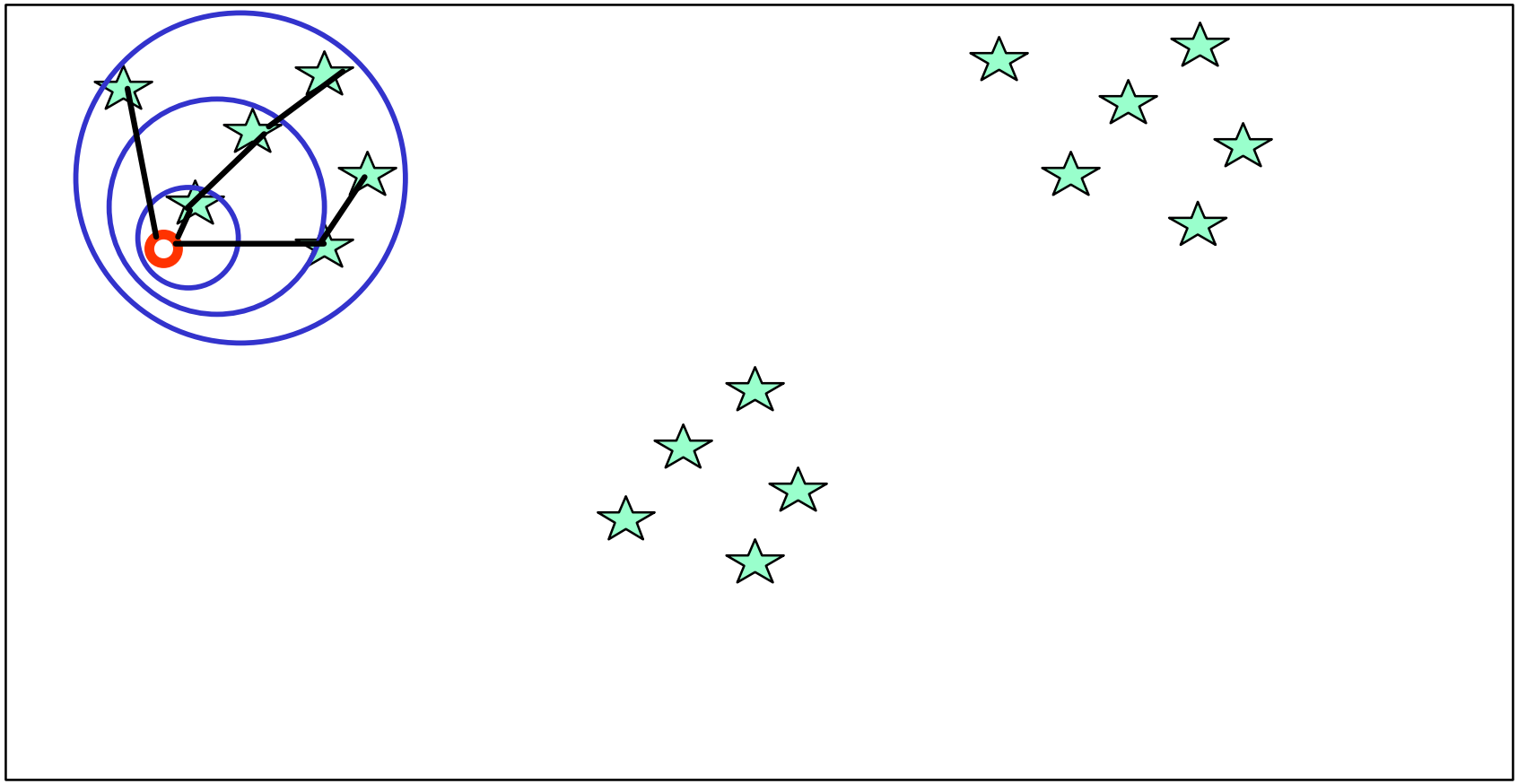
Random Simulation

Drive the design with random but legal input vectors --- MISS HARD-TO-FIND BUGS



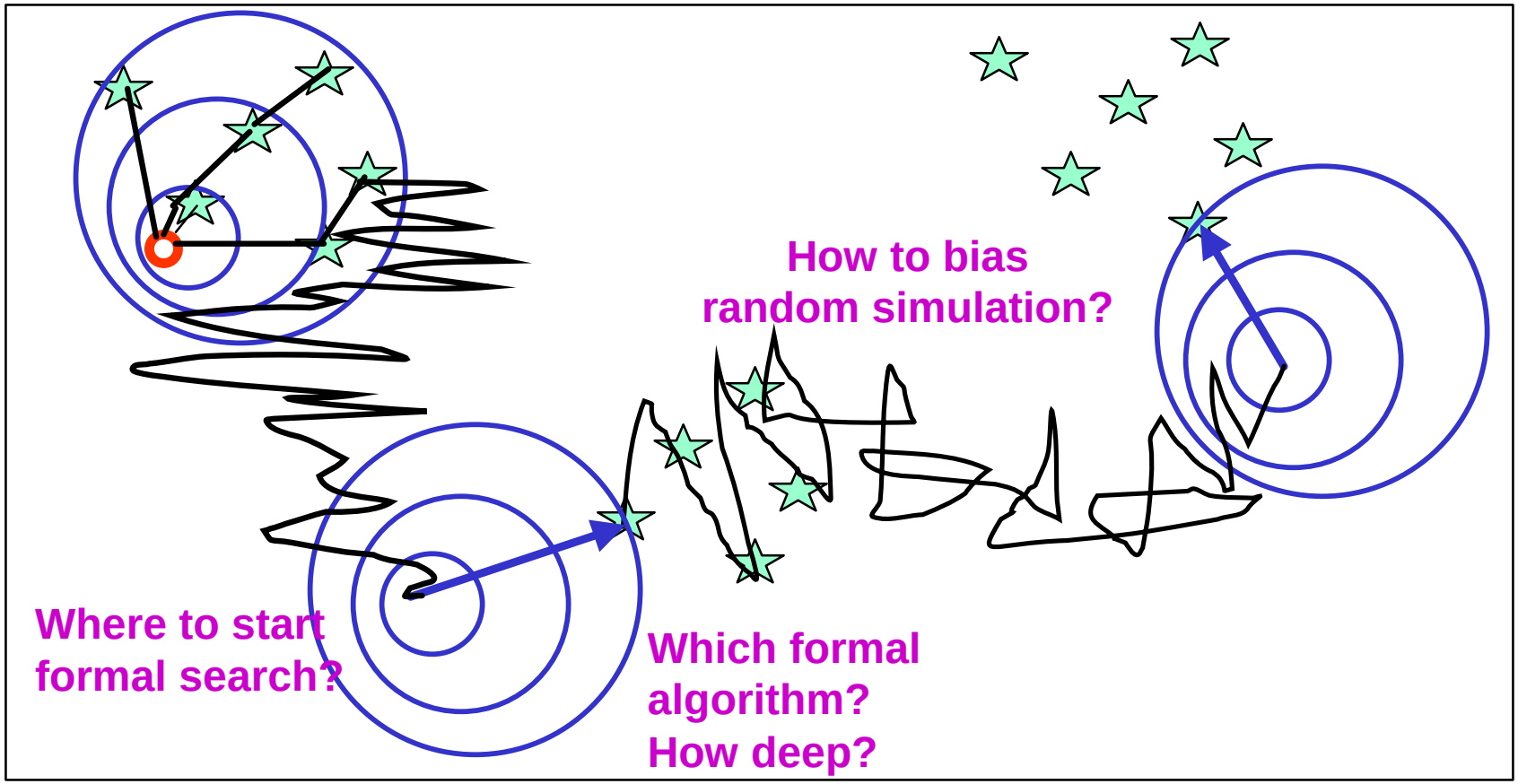
Formal Disproof

Short range semi-exhaustive search --- MISS DEEP BUGS



Hybrid Disproof [ICCAD00]

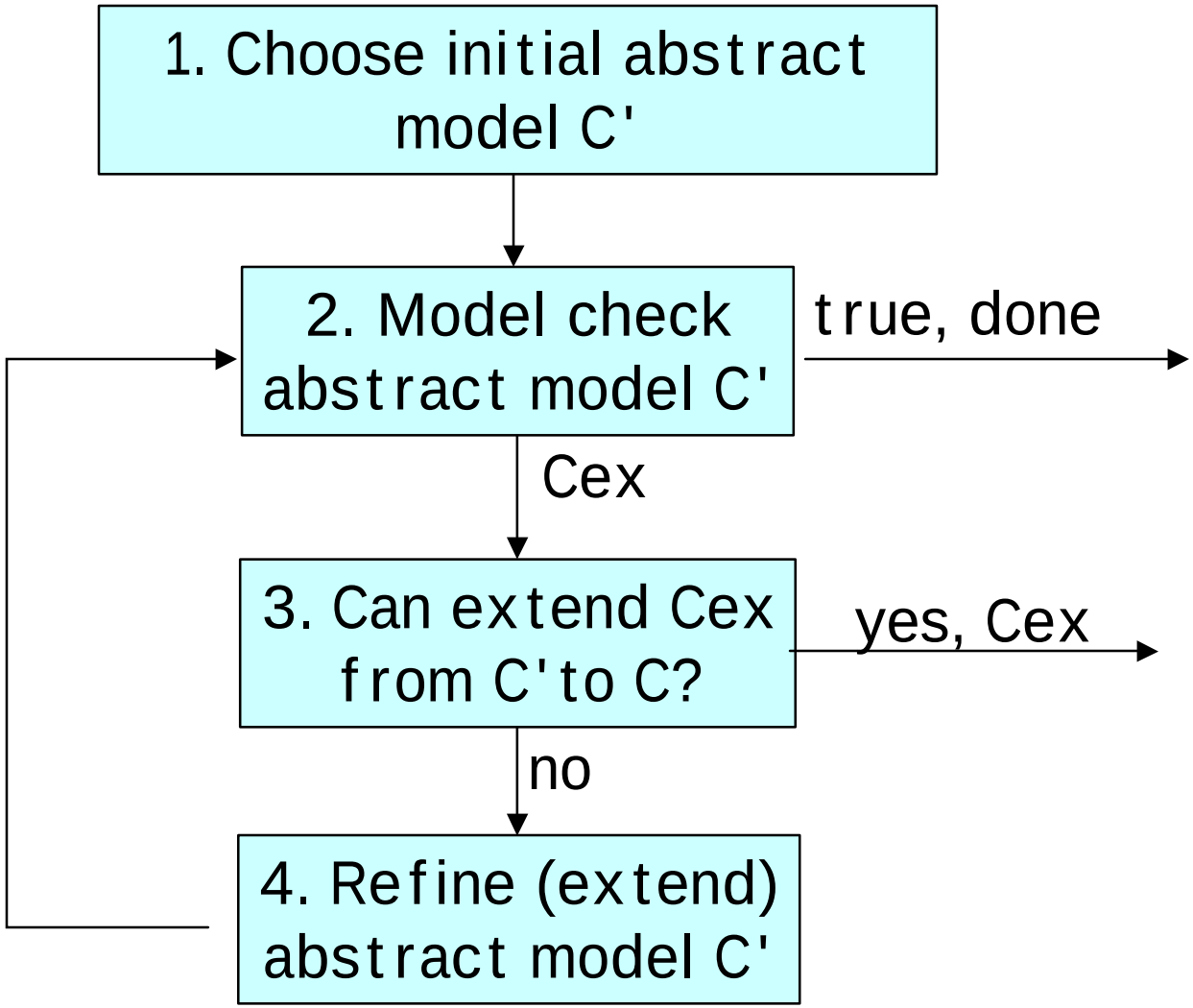
Collaborative simulation and formal engines searching far and wide



Hybrid Iterative Abstraction Refinement

- Goal: Make the size of the design almost irrelevant; Only proof complexity matters
  - Model under verification with 10M gates
  - Algorithm avoids building or analyzing whole design
    - 3-value simulation
    - ATPG in limited fashion
      - ATPG model size is linear to the netlist, not depth*netlist
  - Use hybrid engines to model check abstract model
    - BDD-based symbolic reachability analysis
    - ATPG
  - Wang,Ho,Long,Kukula,Zhu,Ma,Damiano, DAC01

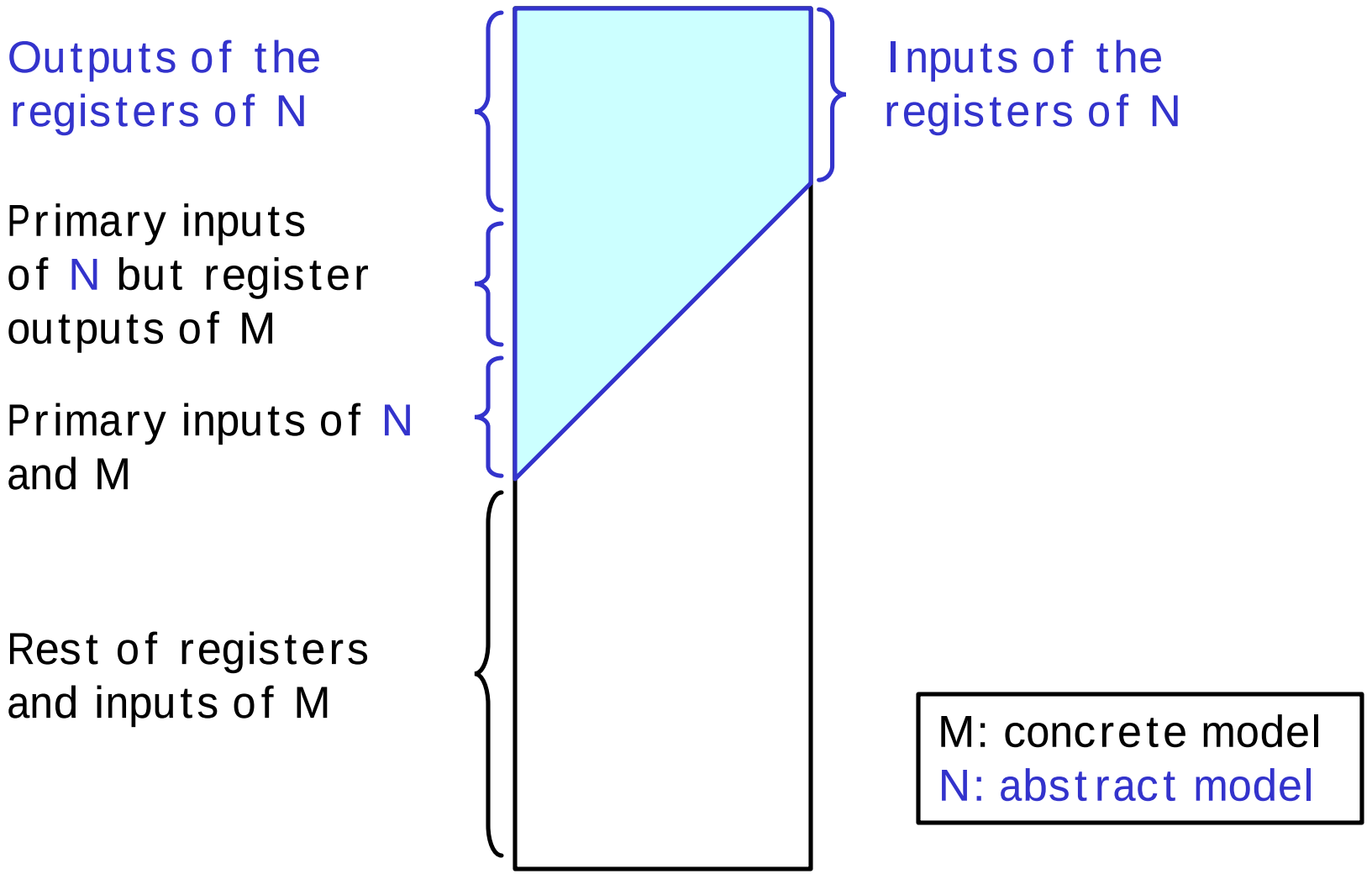
Counter Example Guided Abstraction Refinement



Step 1: Create Abstract Model

- Task
  - Create an abstract model
- Abstract model
  - A subset of registers
    - Included registers, excluded registers
  - Combinational fanin cones of included registers
- Included registers
  - Initially include registers in the assertion
  - Later refinement adds more registers

Abstract model



Step 2: Model Check Abstract Model

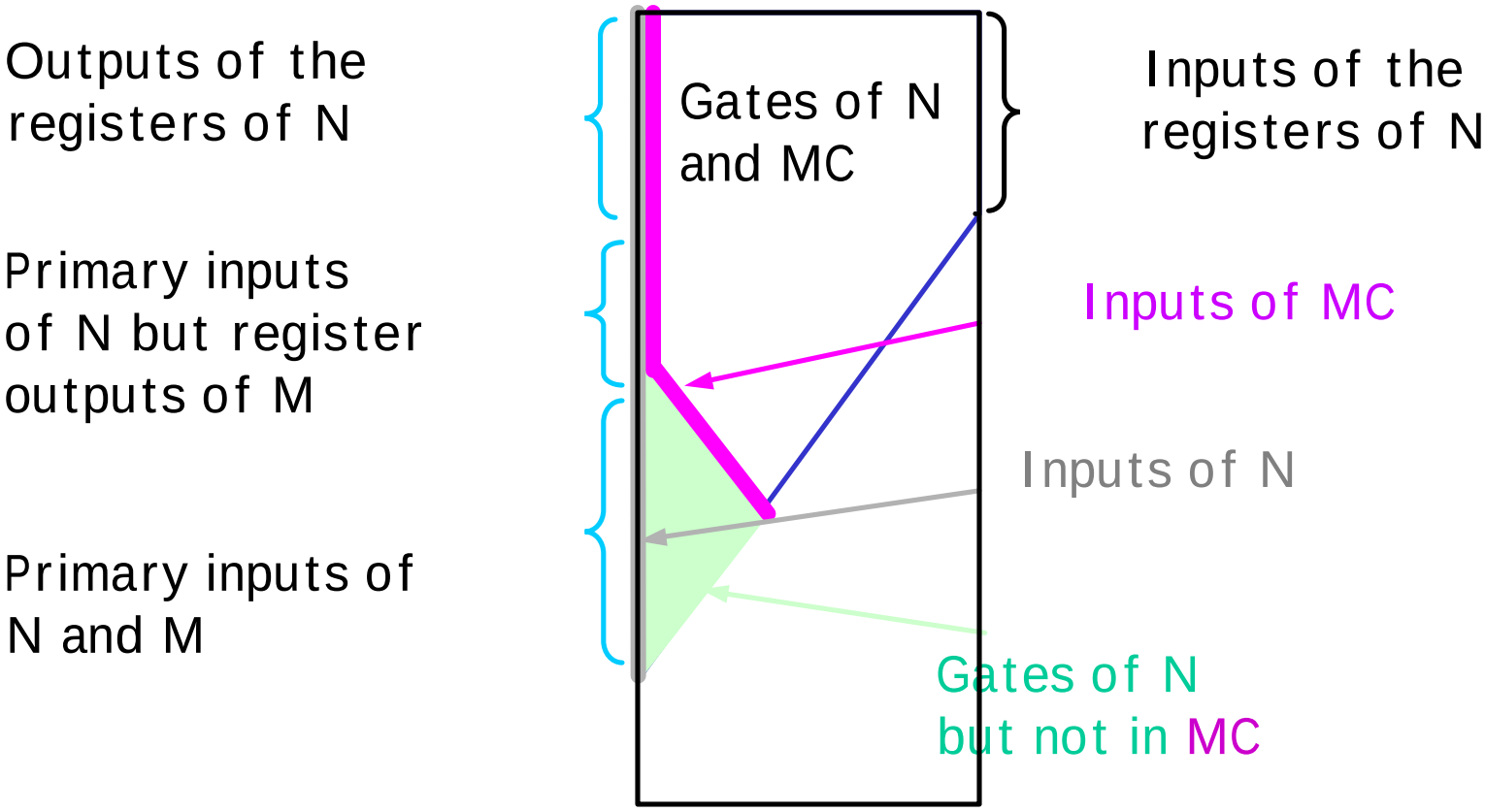
- Task
  - Find an abstract error trace to reach the fail state
  - Or declare the fail state unreachable (assertion proven)
- Find an error trace on the abstract model
  - BDD based image computation
    - Number of input variables is often an issue for backward image computation
  - ATPG based search
    - Length of the error trace sometimes is an issue

Find Abstract Error Trace

- Hybrid BDD-ATPG algorithm for abstract error trace:
  - Forward image to reach the fail state
    - Input variables are existentially quantified out, not an issue
  - Backward image to find an abstract error trace
    - Computes a min-cut abstract model with less number of inputs
    - Backward image to get an assignment to the state variables and min-cut inputs with as many dashes as possible; e.g., (-,0,-,1,-,-,-)
    - Use ATPG to generate an assignment to the original input variables with as many dashes as possible
- ATPG alone as the fall-back solution

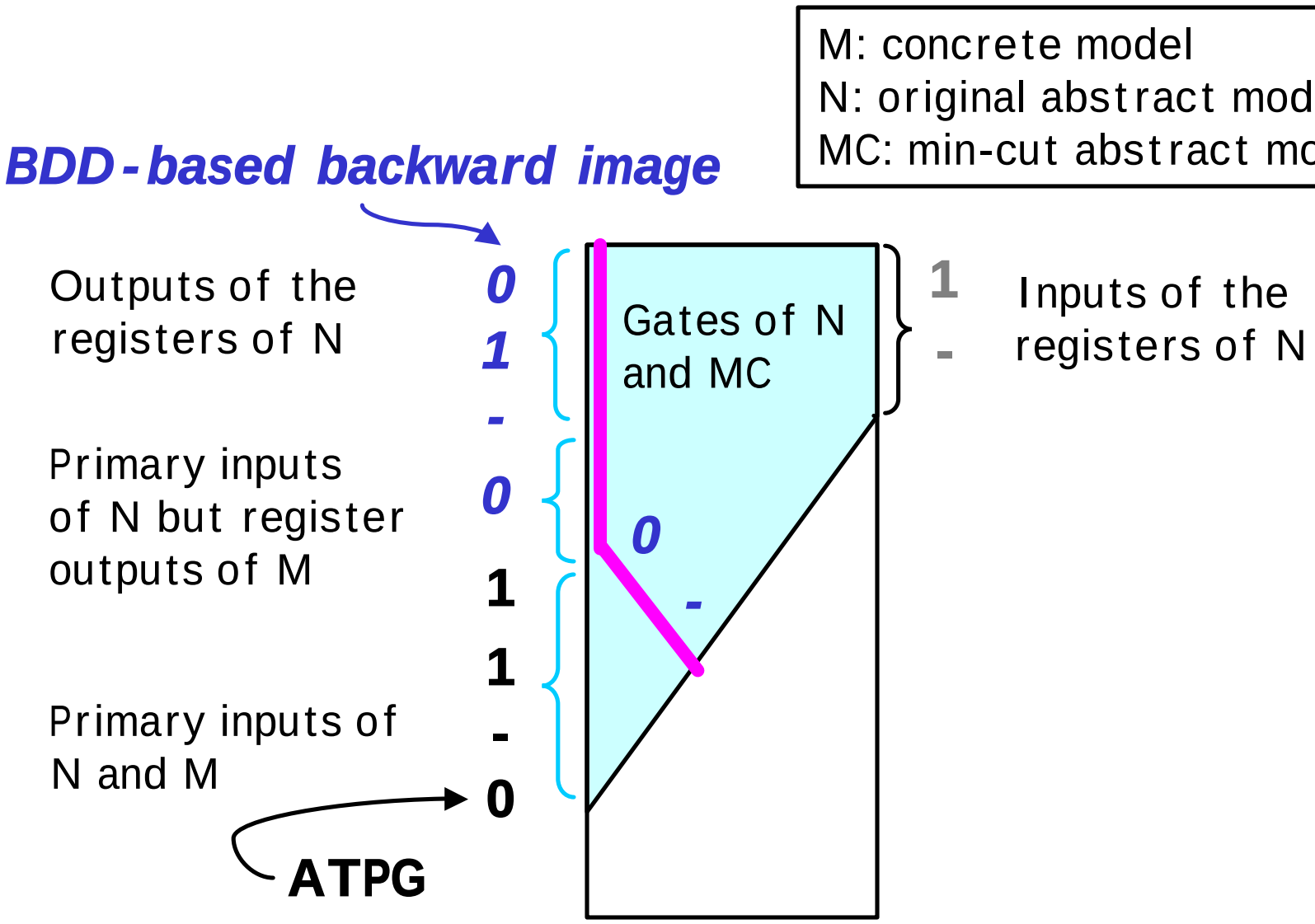
Min-Cut and Original Abstract Models

M: concrete model
N: original abstract model
MC: min-cut abstract model



Hybrid Algorithm

M: concrete model
N: original abstract model
MC: min-cut abstract model



Prove Assertion

- If error trace cannot be found on the abstract model (aborted after reaching resource limit)
  - Apply symbolic reachability analysis to prove the assertion on the abstract model
    - BDD
    - Interpolant ✍ Ken's session
- If proof is also aborted
  - Increase the resource limit and resume error trace finding

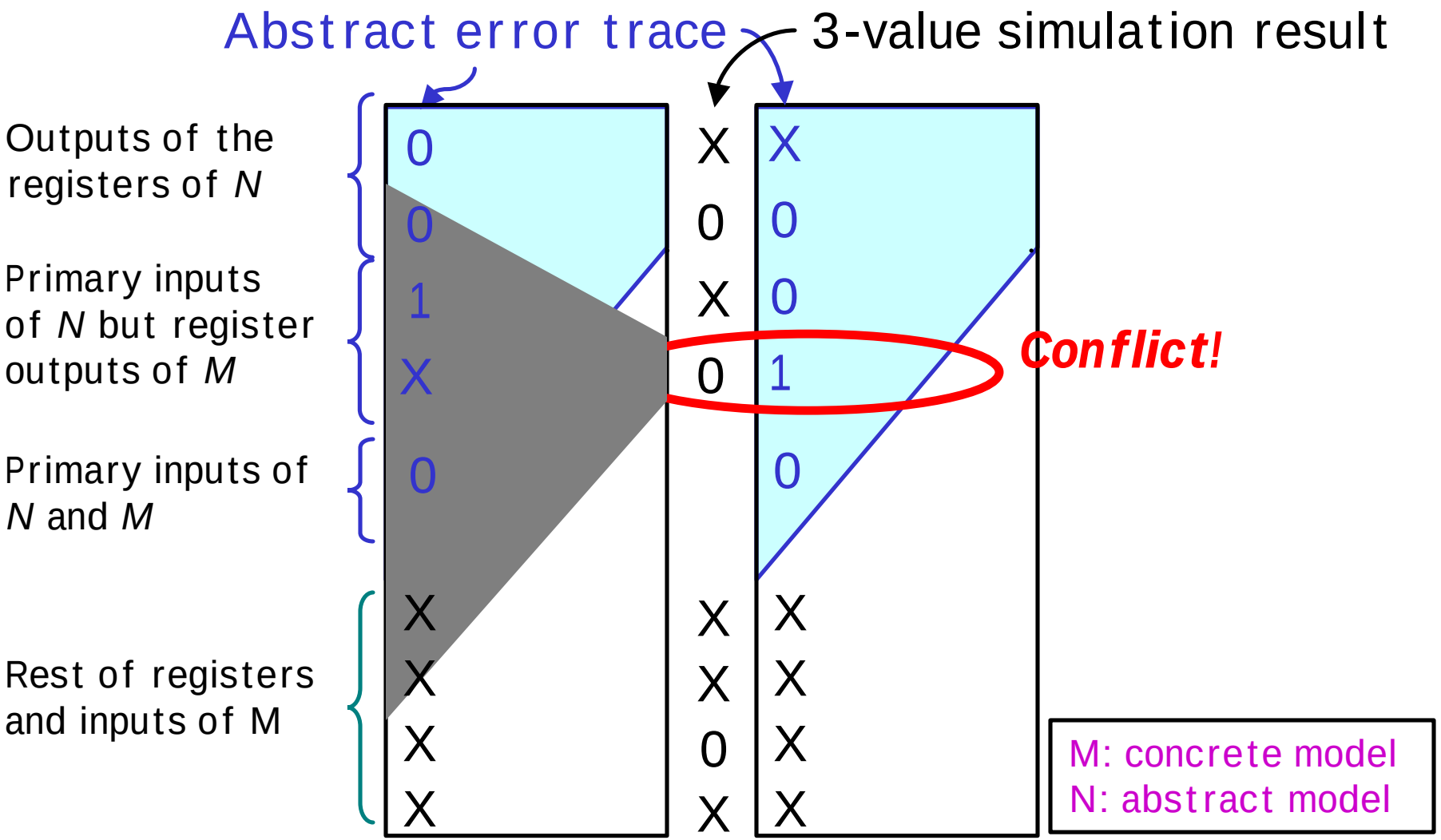
Step 3: Try to Concretize Abstract Error Trace

- Task
  - Check the validity of abstract error trace on concrete model
  - Discover concrete error trace
- Challenge
  - Must analyze the whole design
- Solution
  - Use 3-value simulation to quickly identify abstract error traces that cannot be concretized
  - Use guided ATPG to concretize the error trace

Check Abstract Error Trace Using 3-Value Simulation

- 3-valued simulation
  - Simulate the abstract error trace on concrete model to see if there are conflicts on excluded registers
  - Conflicts ✍ candidates to be included in the refined abstract model
  - No conflicts ✍ Try to concretize the abstract error trace using ATPG

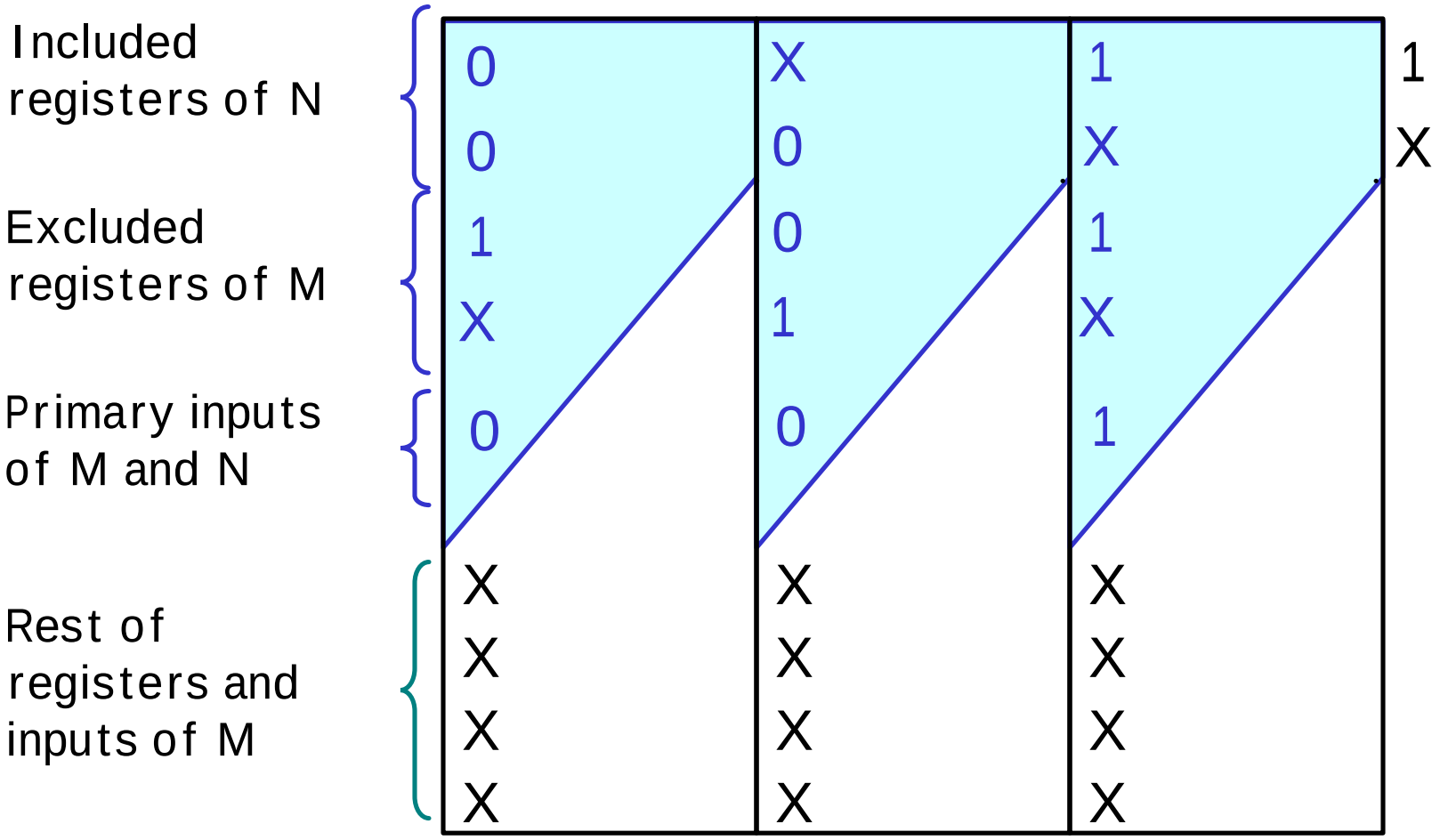
Identify Conflicts Using 3-Value Simulation



Step 3: Try to Concretize the Abstract Error Trace

- Conflict
  - Yes ✍ conflict variables are good candidates to be included to refine the abstract model (in Step 4)
  - No ✍ guided ATPG to find concrete error trace
- Guided ATPG
  - Runs faster than unguided
  - Gradually impose more constraints
    - Increases the chance to find real error traces

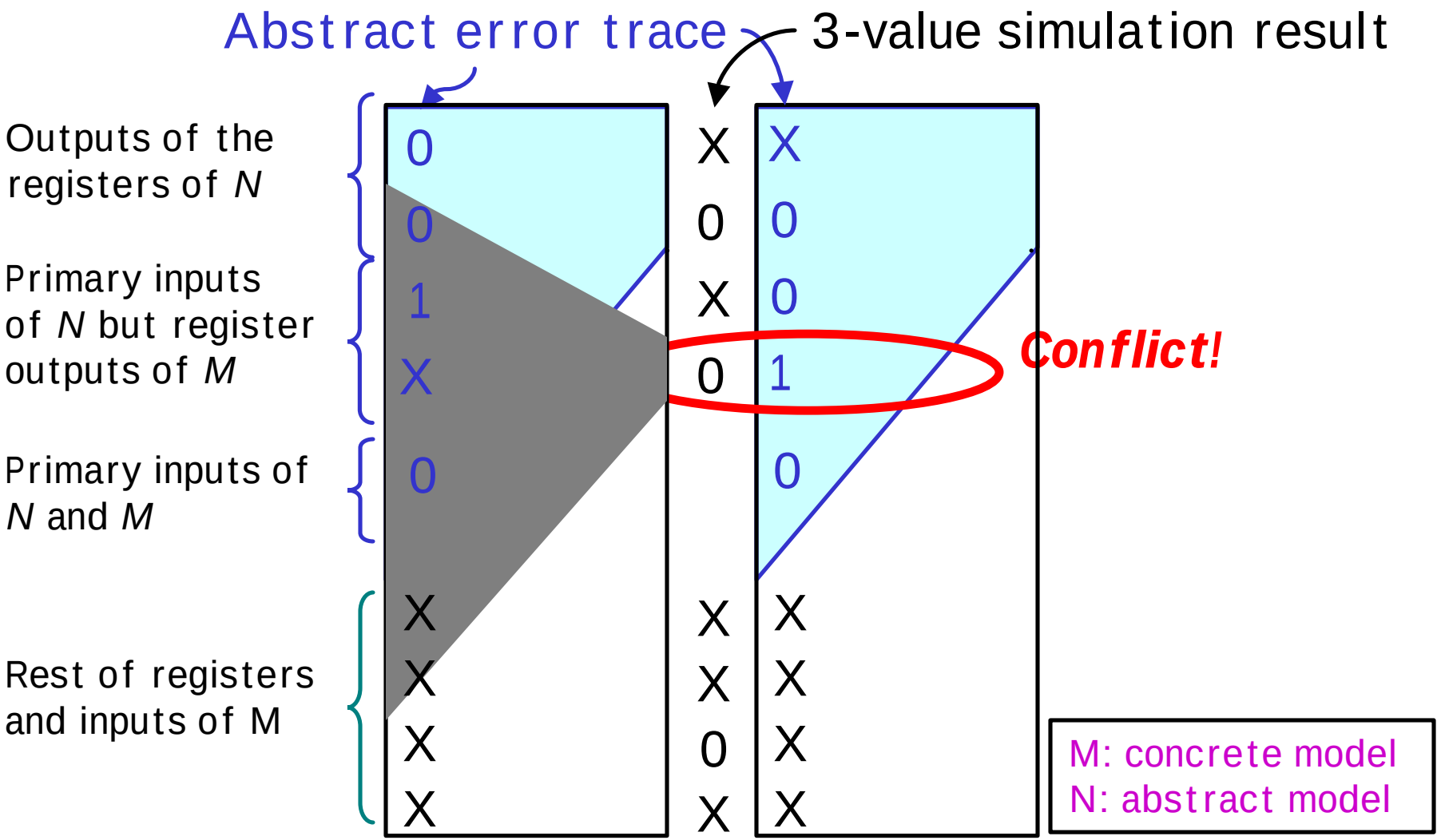
Abstract Error Trace Guided ATPG



Step 4: Refinement

- Task
  - Add “important” registers to refine the abstract model
  - Intuition: add registers that invalidate the spurious error trace
- Key idea: 3-value simulation conflicts are good candidates
  - Assignments required by the spurious error trace if the trace is minimal (true for BDD, not always true for ATPG, SAT is bad for this)
  - Concrete model does not permit the assignments (conflicts)

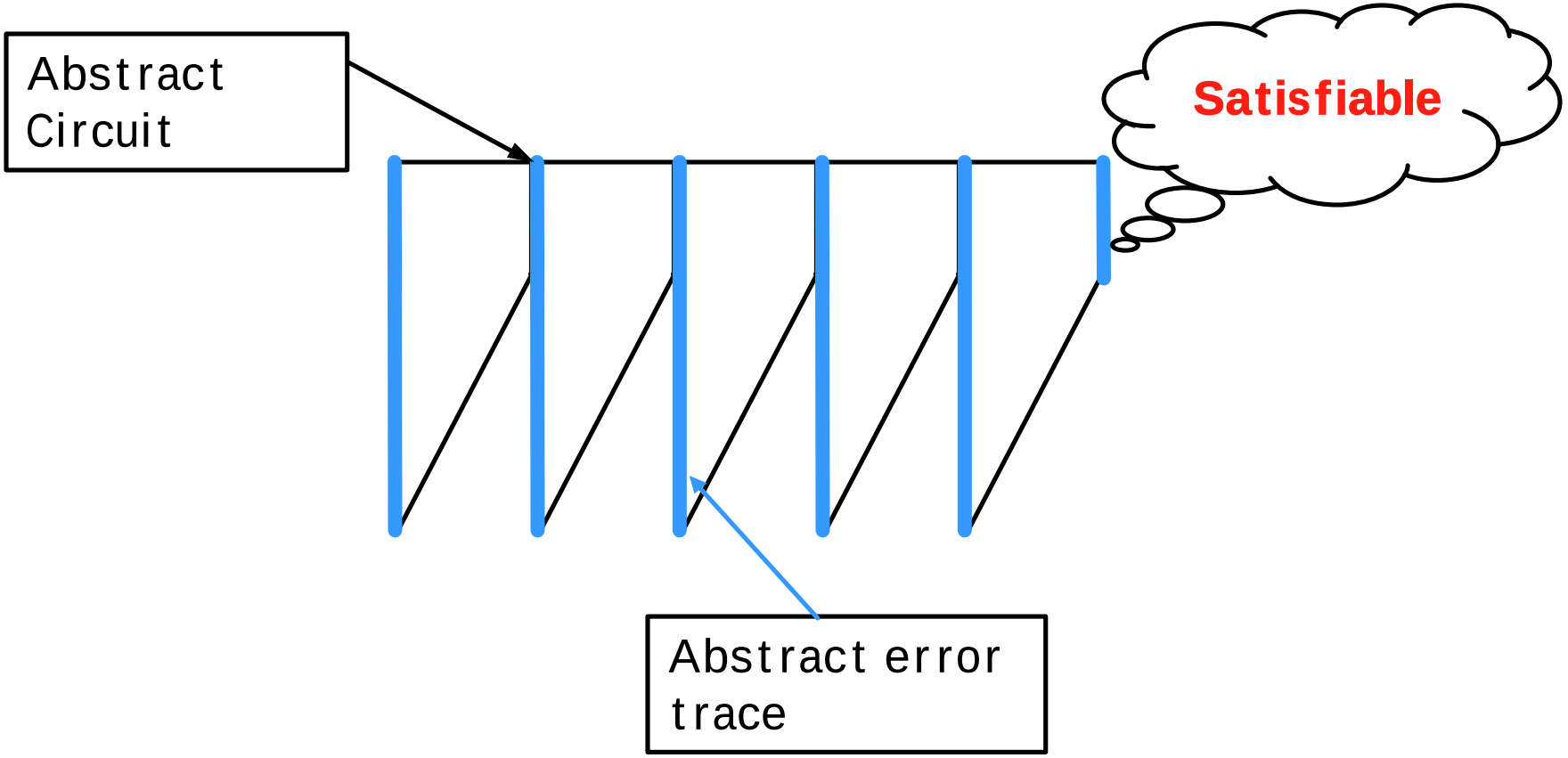
Identify Conflicts Using 3-Value Simulation (Recall)



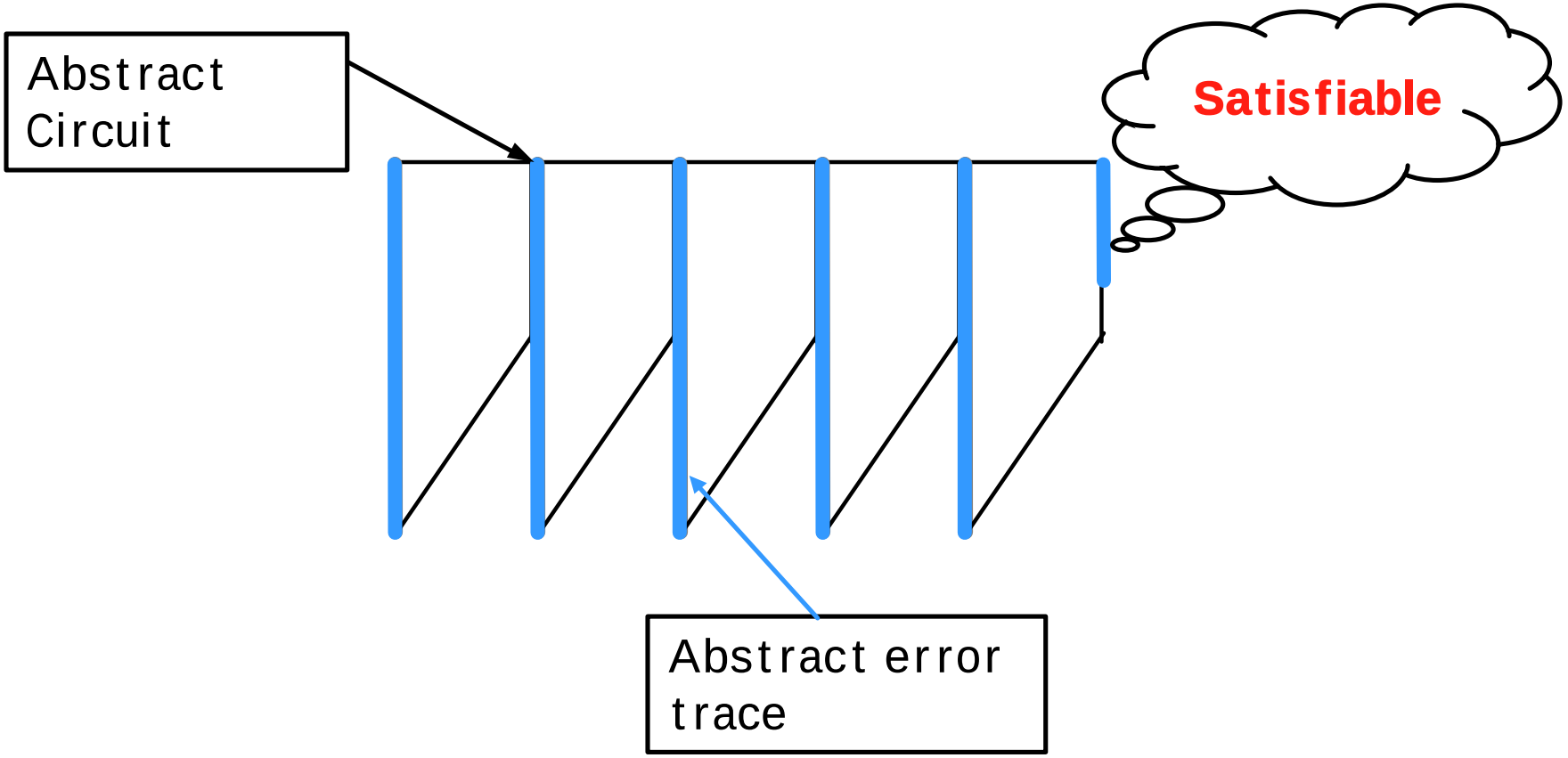
Candidate Minimization

- Find a smaller set of registers to be included
- Greedy minimization algorithm using SAT
  - Order all conflict registers according to heuristics
  - Add one conflict register at a time to the abstract model
    - Until the augmented abstract model and the error trace become unsatisfiable
  - Try to remove previously added conflict registers one at a time
    - A register is removed if the minimized model is still unsatisfiable

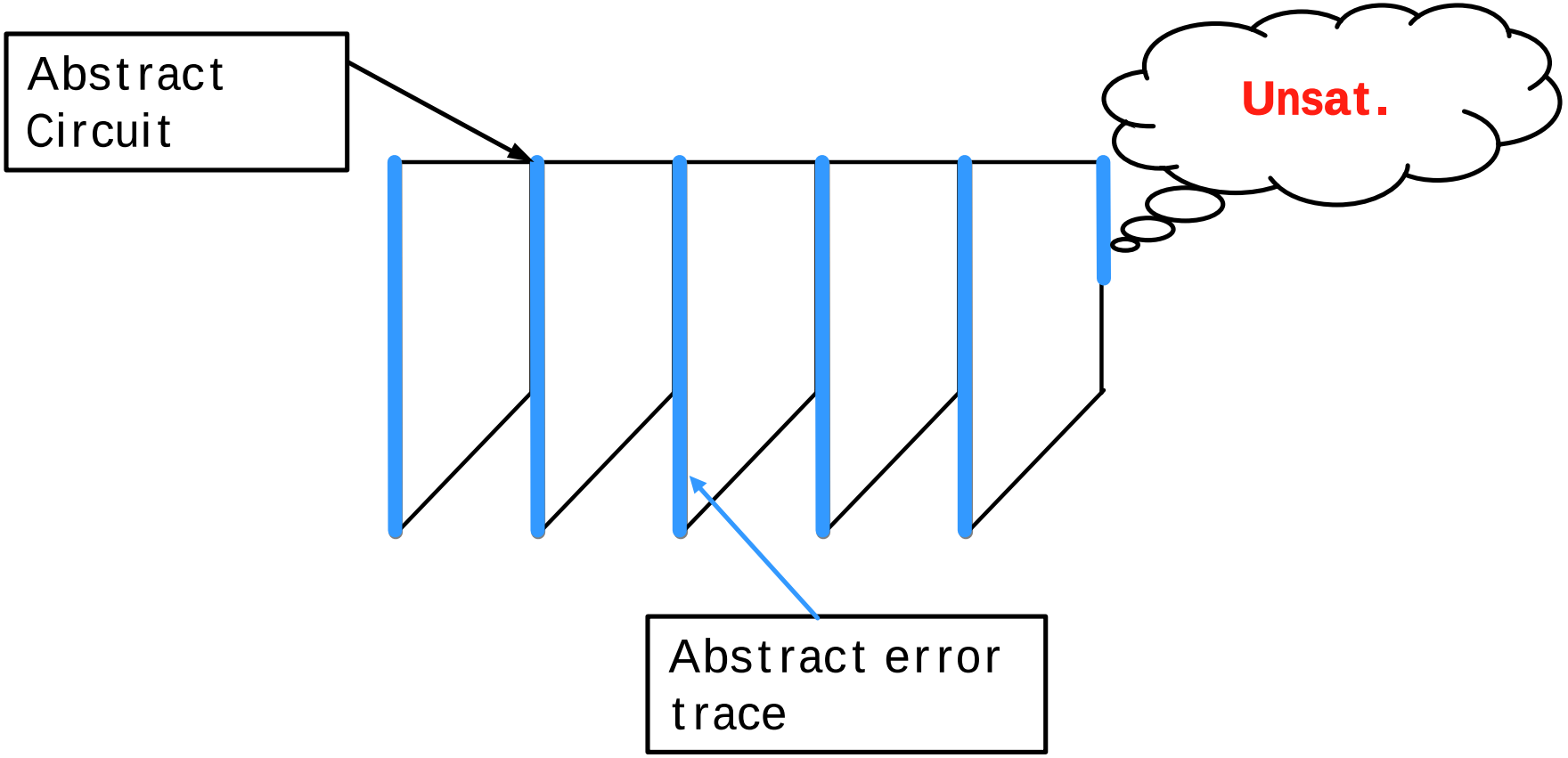
Minimization by SAT



Minimization by SAT



Minimization by SAT



Related Work

- Clarke,Grumberg,Jha,Lu,Veith, CAV00
  - Require building transition relation of whole design in BDD
  - Refinement based on investigation of deadend states
    - States in abstract error trace
      - Can be reached by concrete error trace
      - Cannot reach fail states by concrete error trace
      - Closest to fail states
    - Different than the deadend states that we mentioned about random simulation with assumptions
    - But some states before the deadend states might reveal the key register

Related Work (cont.)

- Glusman,Kamhi,Mador-Haim,Fraer,Vardi, TACAS03
  - Consider a sequence of BDDs to find refinement
    - More expensive backward image computation than a sequence of cubes (multiple traces in both cases)
  - Add gates or registers to refine the abstract model
    - More expensive refinement process

Related Work (cont.)

- Wang,Li,Jin,Hachtel,Somenzi, ICCAD03
  - Build synchronous onion rings (SORs) to represent all shortest error traces at once
  - Score each input of the abstract model by the number of transitions that the input can potentially kill (if it is on our side) in the SORs
    - Pick the register that drives the input with the highest score to be included in the abstract model
    - Register may not be on our side
    - Register that can kill all shortest error traces may not get the highest score
  - Expensive computations are only done on the abstract model
    - Scalable
- Many SAT-based abstraction refinement work
  - Ken's session

Bibliography (3/3)

- E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, “Counterexample-guided abstraction refinement,” CAV2000, pp. 154-169
- M. Glusman, G. Kamhi, S. Mador-Haim, R. Fraer, M. Vardi, “Multiple-counterexample guided iterative abstraction refinement: an industrial evaluation,” TACAS2003
- P.-H. Ho, T. Shiple, K. Harer, J. Kukula, R. Damiano, V. Bertacco, J. Taylor, “Smart simulation using collaborative formal and simulation engines,” ICCAD2000, pp.120-126
- C. Wang, B. Li, H. Jin, G.D. Hachtel, F. Somenzi, “Improving Ariadne’s bundle by following multiple threads in abstraction refinement,” ICCAD2003, pp.408-415
- D. Wang, P.-H. Ho, J. Long, J. Kukula, Y. Zhu, T. Ma, R. Damiano, “Formal property verification by abstraction refinement with formal, simulation and hybrid engines,” DAC 2001, pp. 35-40

Outline

- Formal property verification basics
- Modern formal property verification engines
- Hybrid proof and disproof methods
- *State-of-the-art formal property verification tools*

Make Formal Property Verification Main-Stream

- Higher return for the user
  - Handle bigger designs more efficiently
    - Never choke on it; always do something
  - Produce definitive results on more designs
- Lower investment from the user
  - Reduce the effort and turnaround time for setting up the DUV, the assertions and assumptions
  - Tool better be launch-and-forget
  - Reduce debugging effort

Challenges

- Problem is intrinsically hard
  - PSPACE complete, most likely exponentially harder than NP complete problems (place&route and logic synthesis)
- Assertions and assumptions have to be written in RTL or property languages
  - Cannot accept high-level verification languages like Vera, e or C++
- Key ingredients that help

Key 1: Abstraction

- Goal: successful verification depends on the complexity of the proof/disproof rather than the size of the design
  - Abstraction throughout
  - Can produce definitive results for some assertions of designs with 10M gates
    - May fail to do so for some assertions of designs with 1K gates (it's all right)

Key 2: Tight Integration with Simulation Environments

- Handle initialization sequences in HVL, behavior HDL, C++ or VCD
- Help debugging and regression
- Most effective tool for refining the FV model (DUV, assertions, assumptions and reset)
- Reuse assertions and assumptions between simulation and formal property verification
- Identify and reduce synthesis-simulation mismatches

Key 3: Multiple Formal Technologies

- Already a must for main-stream formal equivalence checking tools
- Different technologies/tricks excel at
  - Different applications (proof vs. disproof)
  - Different assertions (easy vs. hard)
  - Different design constructs (control vs. data)
- No silver bullets
  - The more technologies the more successes

Key 4: Proof vs. Disproof

- Prove and disprove deserve separate considerations
  - Prove:
    - Most effective abstraction is over-approximation of the model
      - More traces than original model; no false positive
      - False error traces
  - Disprove:
    - Most effective abstraction is under-approximation of the input stimuli
      - Less traces than original model; no false negative
      - Cannot give valid (unbounded) proof
      - Bounded proof