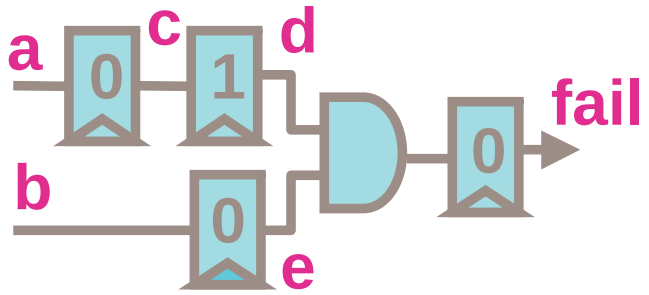
Abstraction Refinement

Pei-Hsin Ho
Advanced Technology Group
Synopsys, Inc.

Formal Verification of Safety Properties

- Problem:
  - A gate-level design with an initial state and an output signal fail
  - Prove that fail is always 0
- Answer:
  - True
  - False, input sequence (error trace) that asserts fail
  - Inconclusive

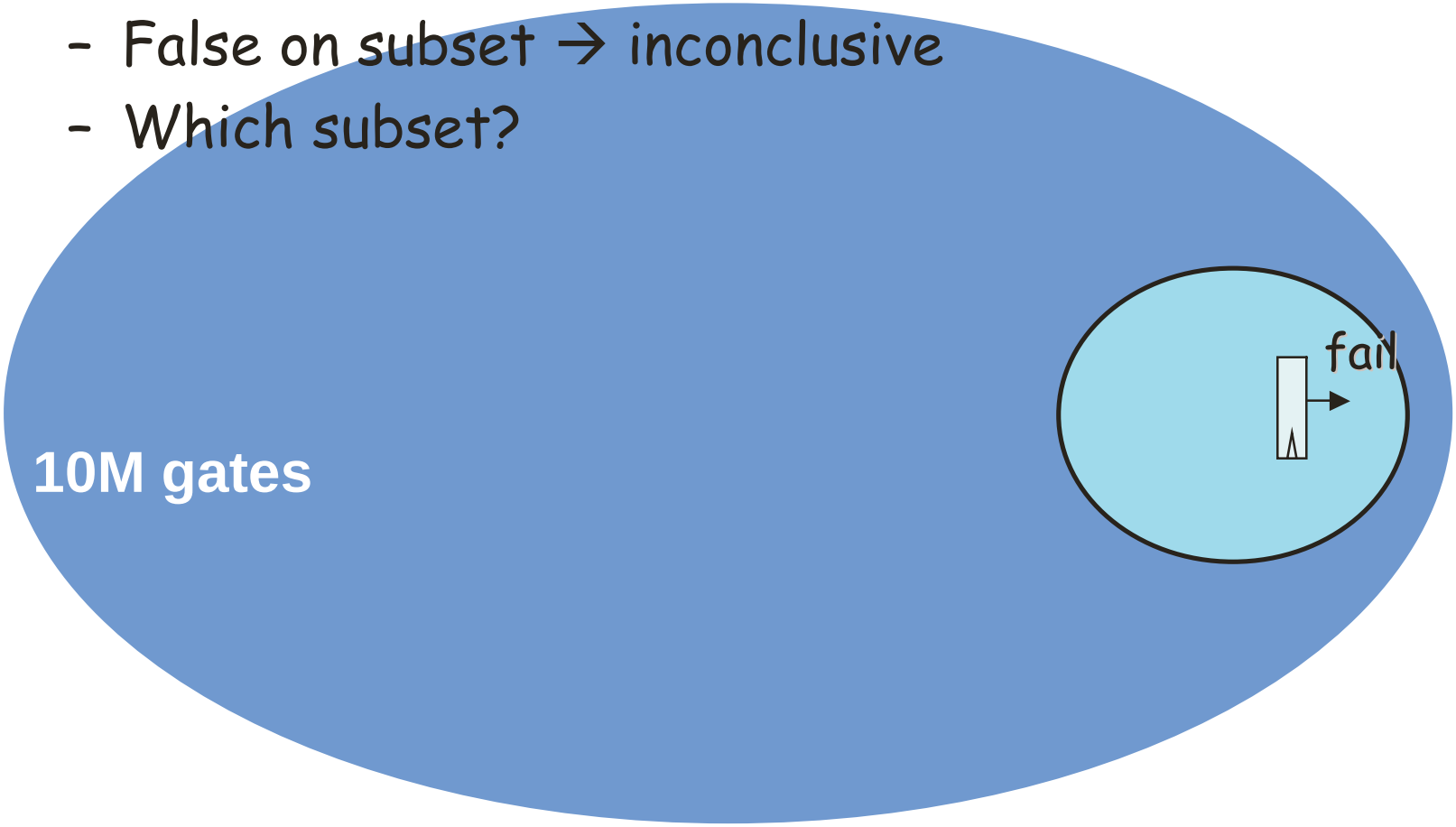


Challenge

- Design may have 10M gates (~1M registers)
- Formal proof engines cannot handle
  - BDD: ~200 registers
  - Clause [McMillan02]: ~10K gates
  - Interpolant [McMillan03]: ~100K gates
- How?
  - Abstraction

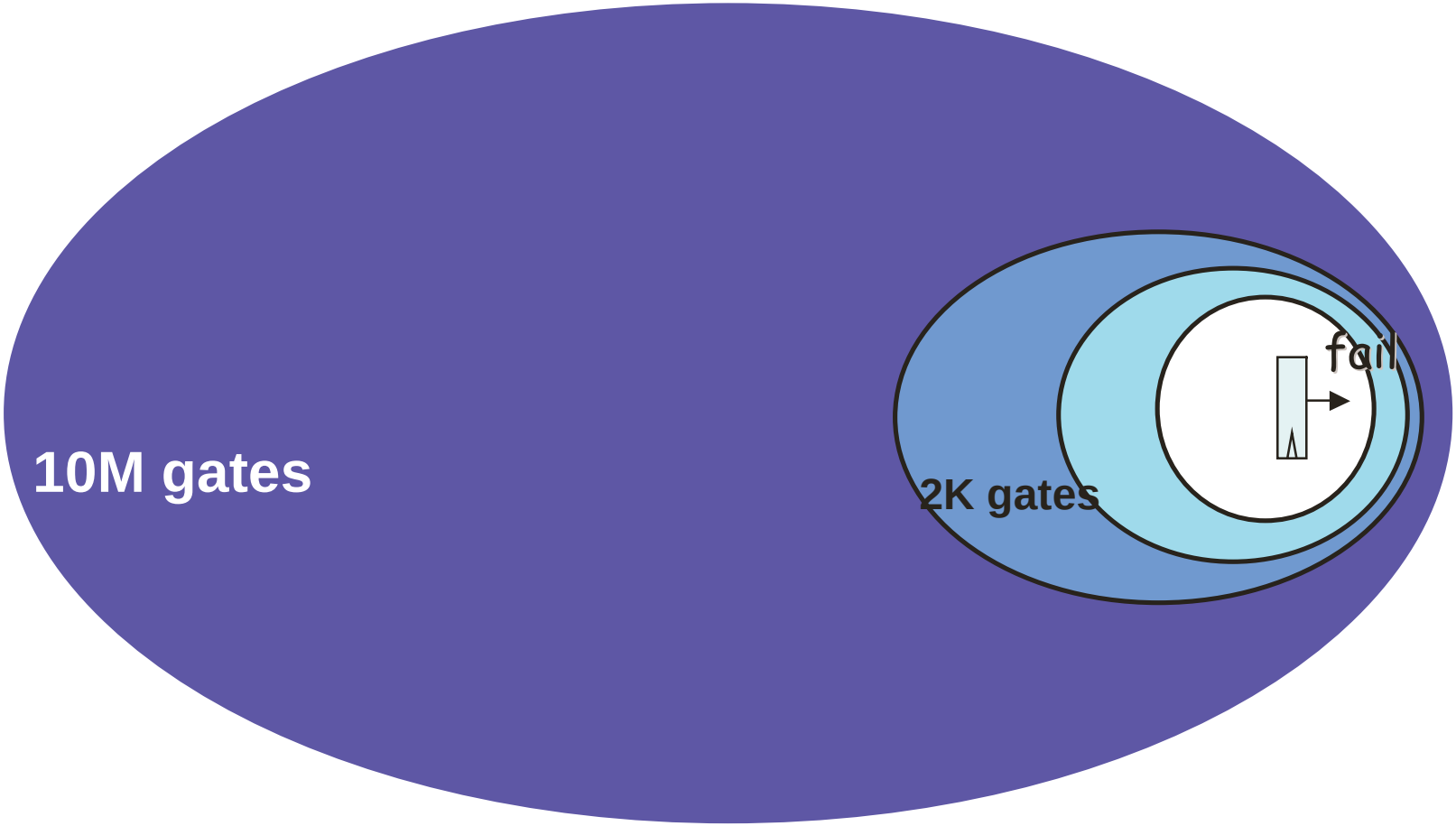
Subset Abstraction

- Prove the property on a subset
  - True on subset → True
  - False on subset → inconclusive
  - Which subset?

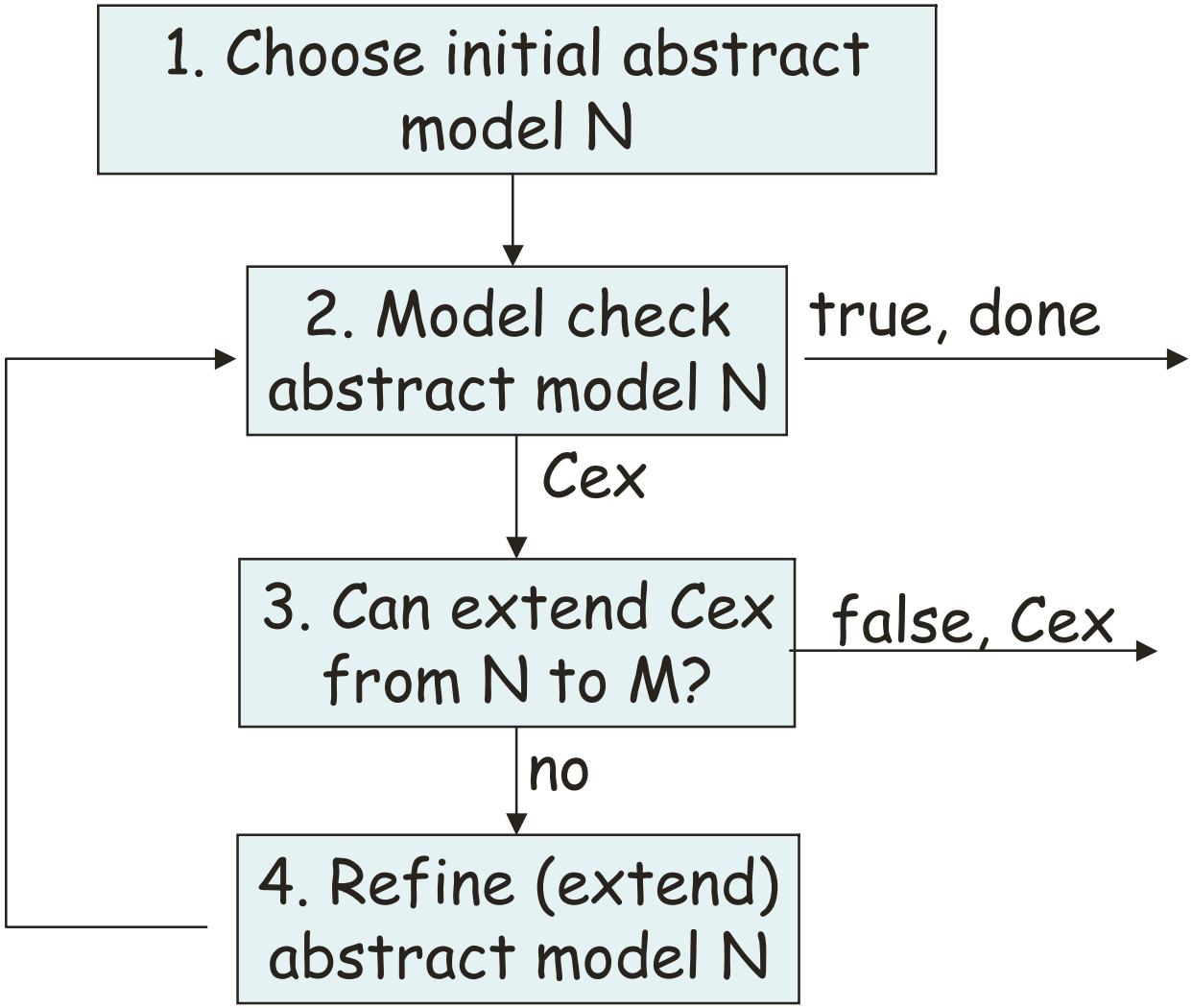


Abstraction Refinement

- Incrementally expand the abstract model (subset) by analyzing the error traces



Abstraction Refinement Algorithm



History (As Far As I Know)

- Kurshan introduced abstraction refinement
  - Localization reduction
  - R. Kurshan. Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach. Princeton University Press, 1994.
    - High level

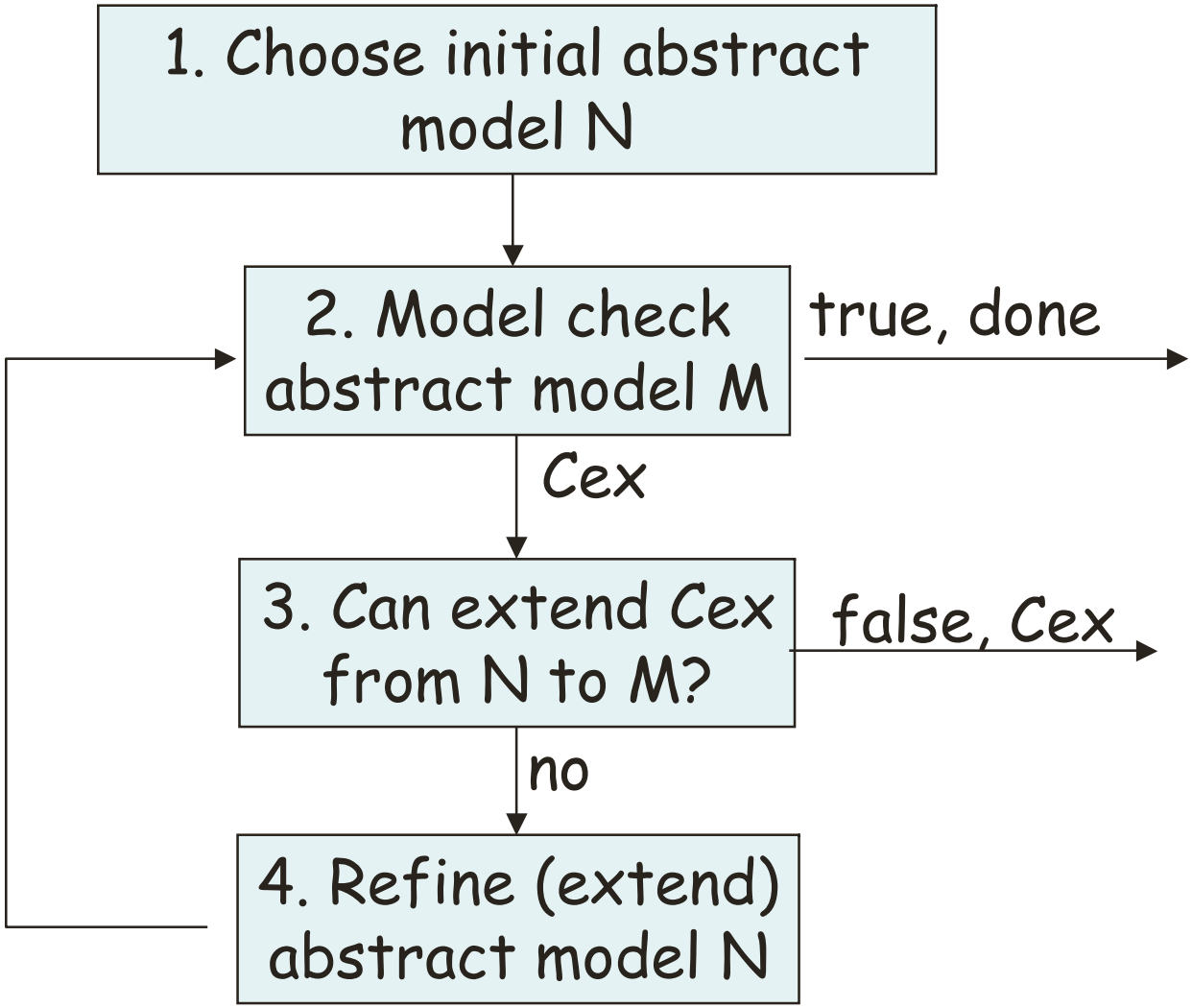
History (cont.)

- Clarke,Grumberg,Jha,Lu,Veith, CAV00
  - 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
  - Require building transition relation of the design in BDD
    - Not effective in real world

This Talk

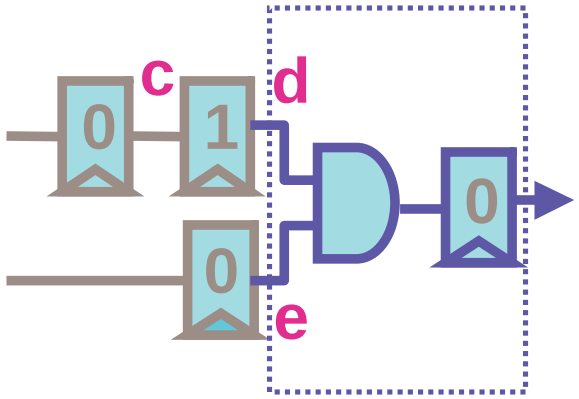
- Mang,Ho, DAC04
- Wang,Ho,Kukula,Zhu,Ma,Damiano, DAC01
- Goal: Make the size of the design almost irrelevant; Only proof complexity matters
  - 10M gates
  - Algorithm avoids building or analyzing whole design
    - 3-value simulation
    - ATPG in limited fashion
      - ATPG model size is linear to the netlist, not depth*netlist
  - Use hybrid engines to model check abstract model
    - BDD-based symbolic reachability analysis
    - ATPG

Abstraction Refinement Algorithm



Step 1: Create Abstract Model

- Task
  - Create abstract model N
- Abstract model
  - A subset of registers and their combinational fanin cones
  - Initially → "fail" and its combinational fanin cone



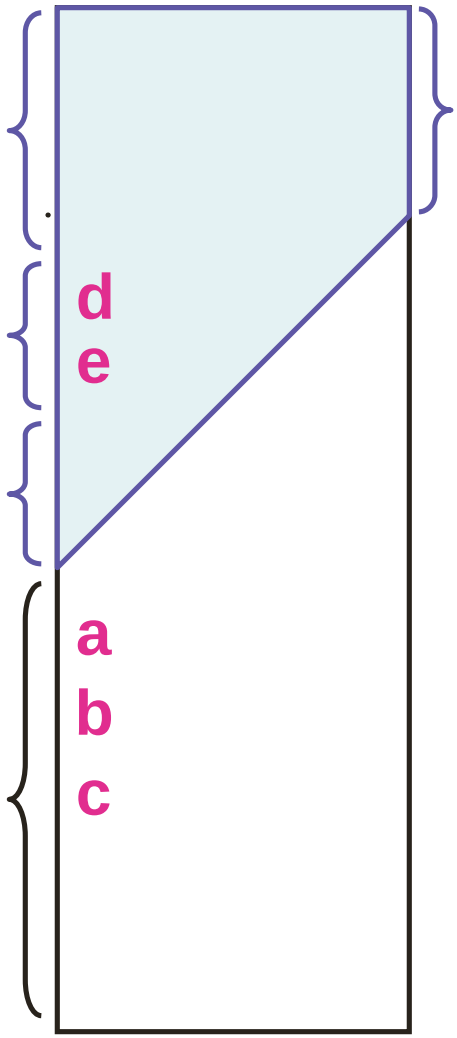
The Abstract Model

Outputs of the registers of N

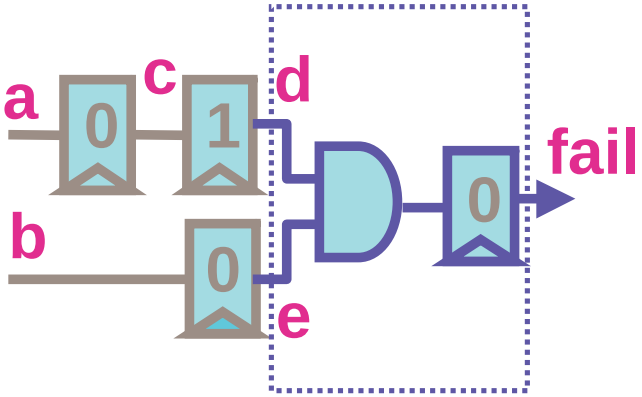
Primary inputs of N but register outputs of M

Primary inputs of N and M

Rest of registers and inputs of M



Inputs of the registers of N



M: concrete model
N: abstract model

Step 2: Model Check Abstract Model

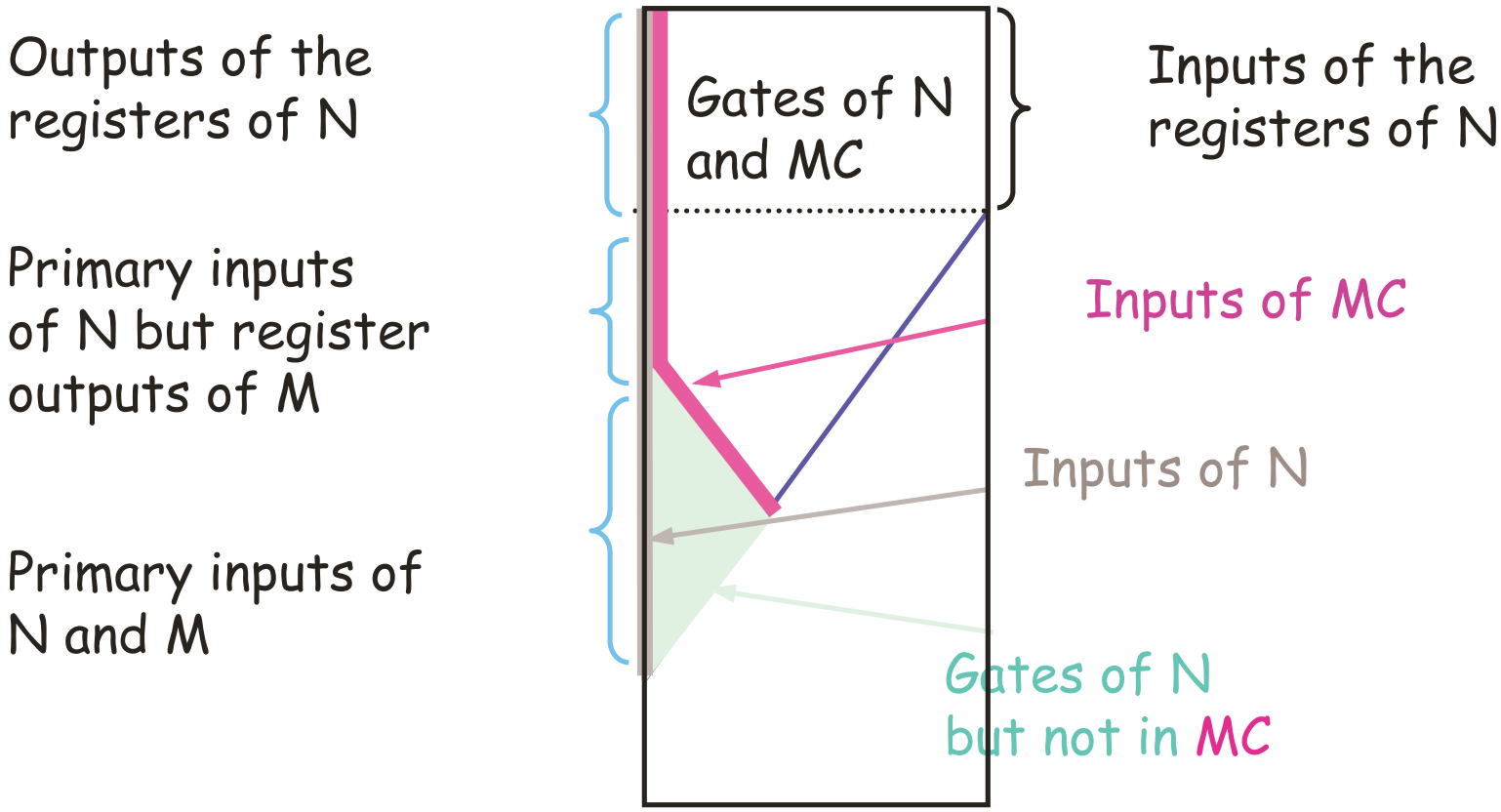
- Task
  - Find an abstract error trace to assert fail
  - Or declare that fail is always 0 (proven)
- Find an error trace on the abstract model
  - BDD based image computation
    - Forward image computation
    - Number of input variables is often an issue for backward image computation
  - SAT/ATPG based search
    - Length of the error trace sometimes is an issue

Find Abstract Error Trace

- Hybrid BDD-ATPG Simulation algorithm for abstract error trace:
  - Forward image to reach the fail state
  - Backward image to find an abstract error trace
    - Computes a min-cut abstract model with less number of inputs
    - … (next 2 slides)

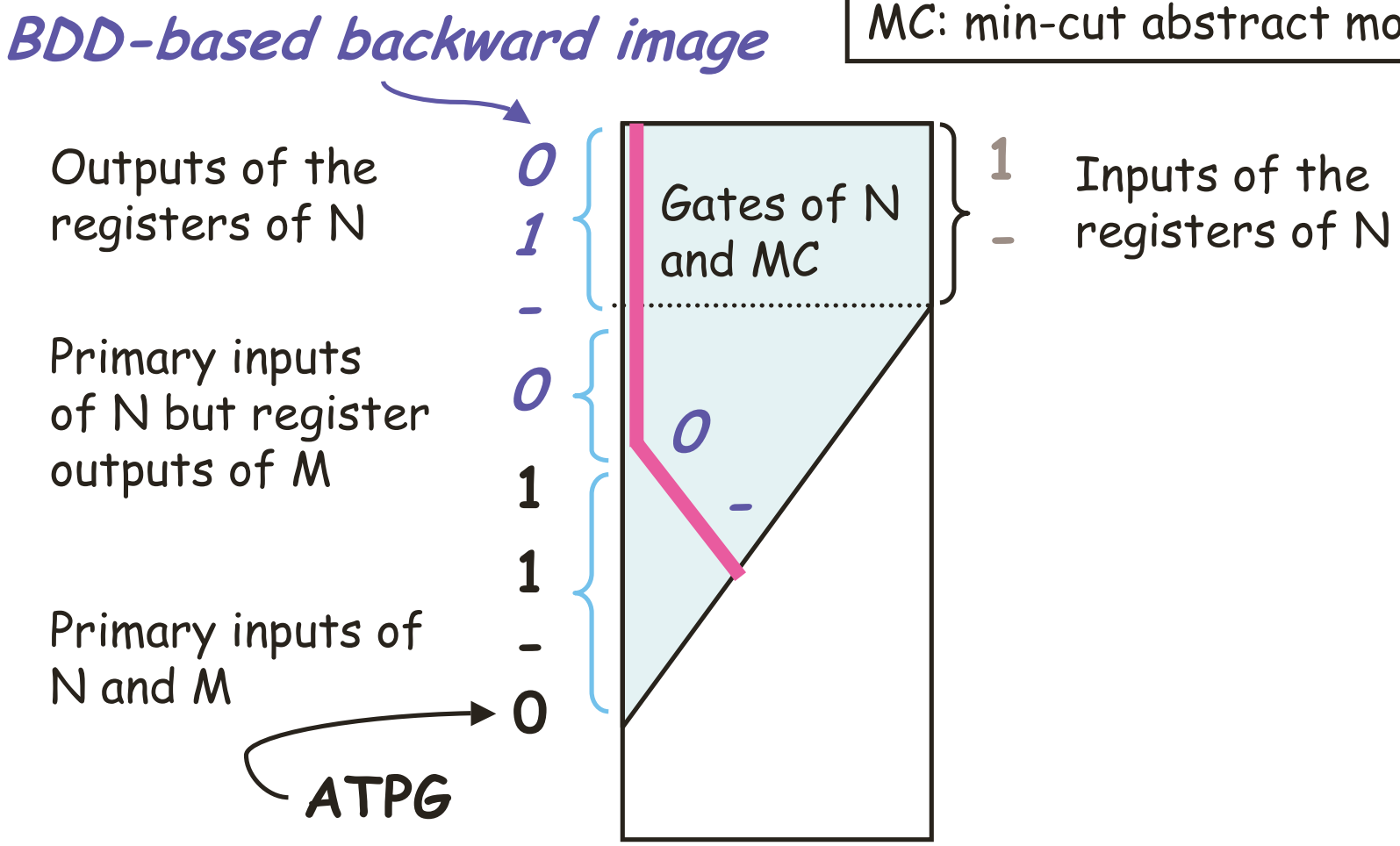
Min-Cut and Original Abstract Models

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 The Property

- If error trace cannot be found on the abstract model (after reaching resource limit)
  - Apply symbolic reachability analysis to prove the property on the abstract model
    - BDD-based forward fixpoint
    - Or SAT[McMillan02] or interpolant[McMillan03]

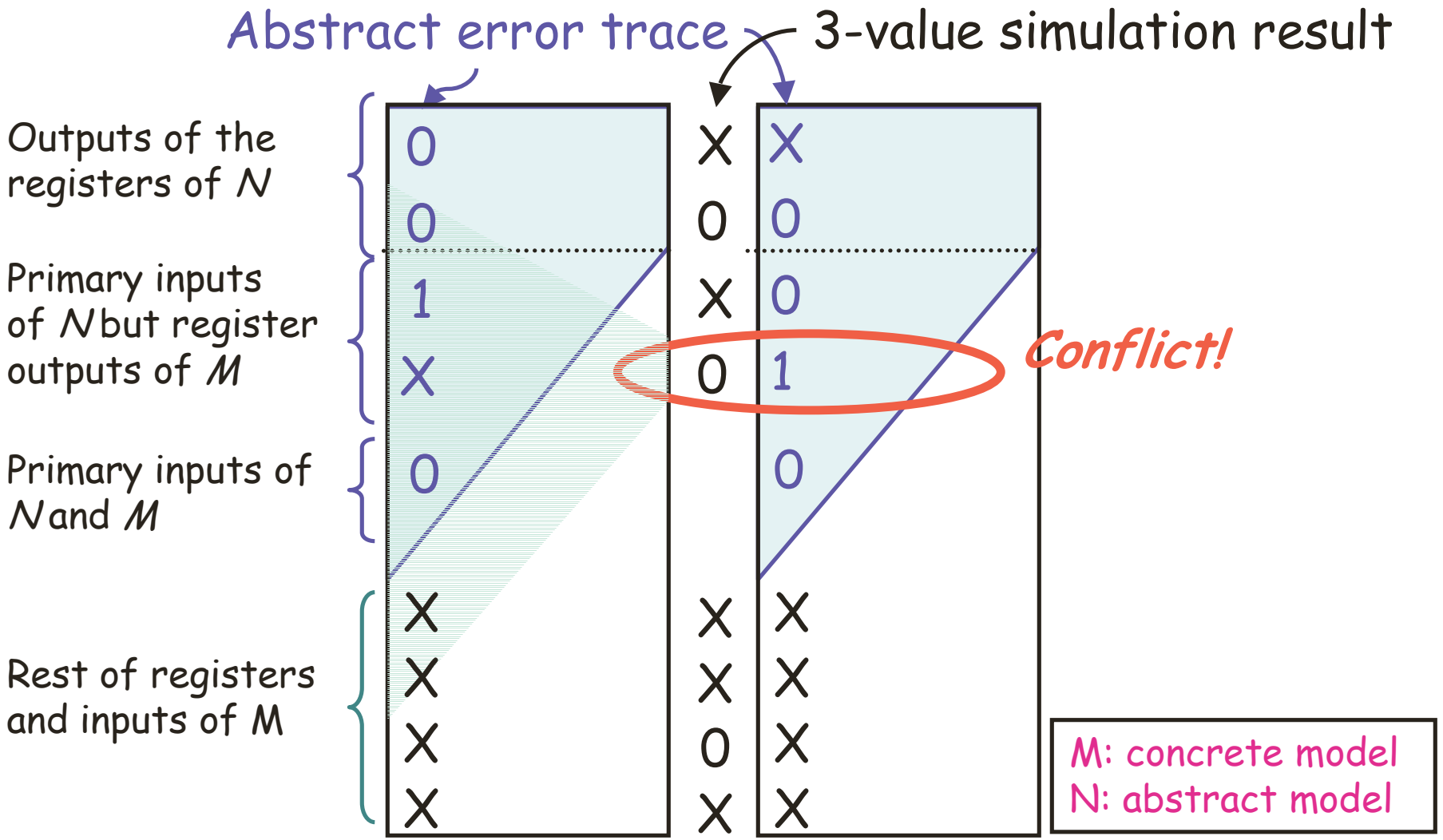
Step 3: Try to Concretize Abstract Error Trace

- 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

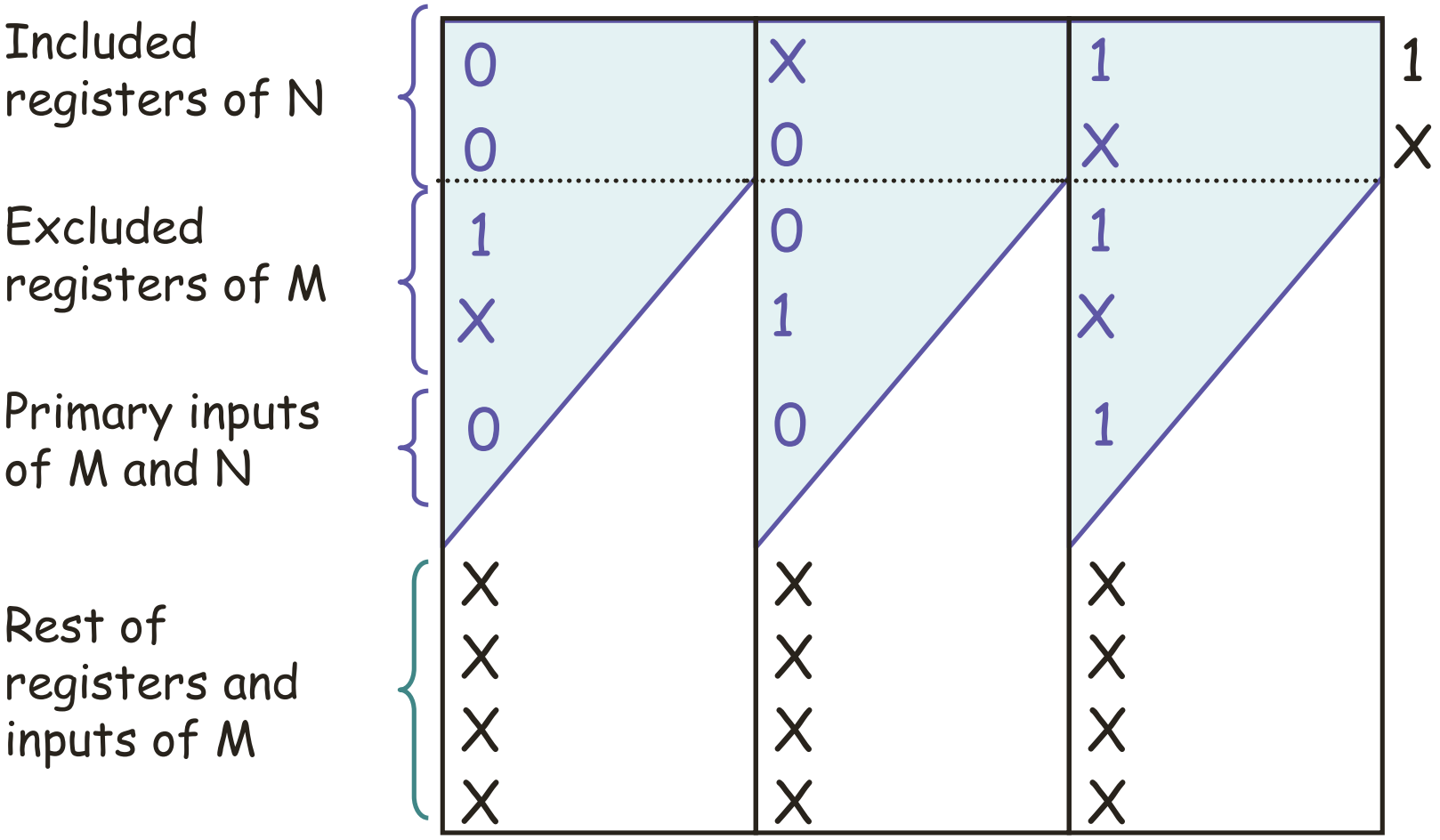
Identify Conflicts Using 3-Value Simulation



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

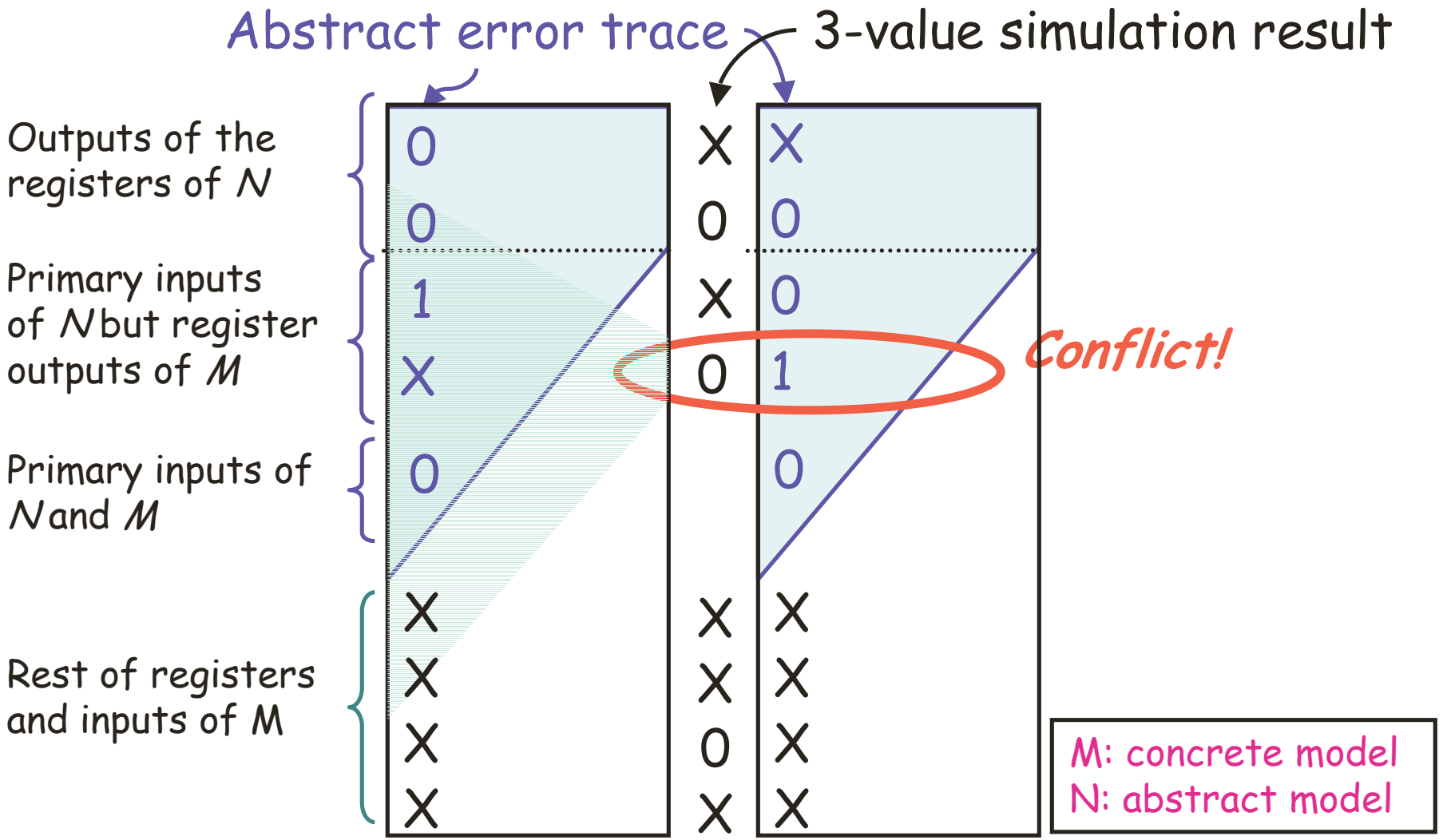
Abstract Error Trace Guided ATPG



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
  - Concrete model does not permit the assignments (conflicts)

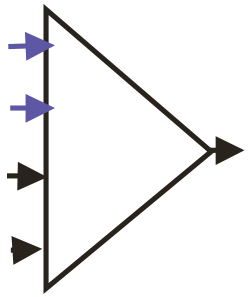
Identify Conflicts Using 3-Value Simulation (Recall)



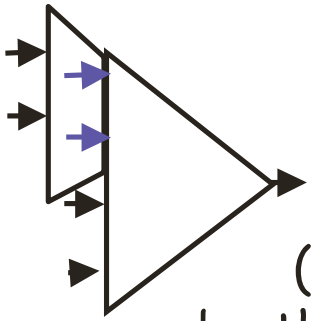
Rank Conflict Registers

- Conflict registers are ranked:
  - Frequencies of conflict (high)
  - Persistence (beg to be selected)
  - Sequential Distances (close)
  - Input Widths (small)
    - Number of primary inputs in the support of transition function
- Can we do better?
  - Game based register selection

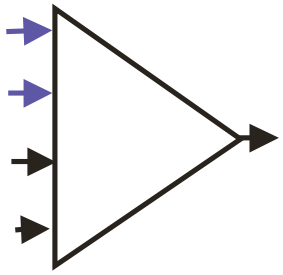
K-Cooperativeness



Input trace
$c^0, c^1 \ldots c^k$
is a counterexample



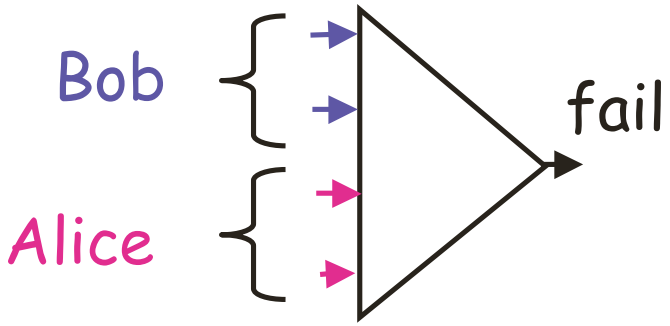
Trace
$c^0, c^1 \ldots c^k$
not valid
(not producible
by the additional logic)



Blue inputs are called k-cooperative

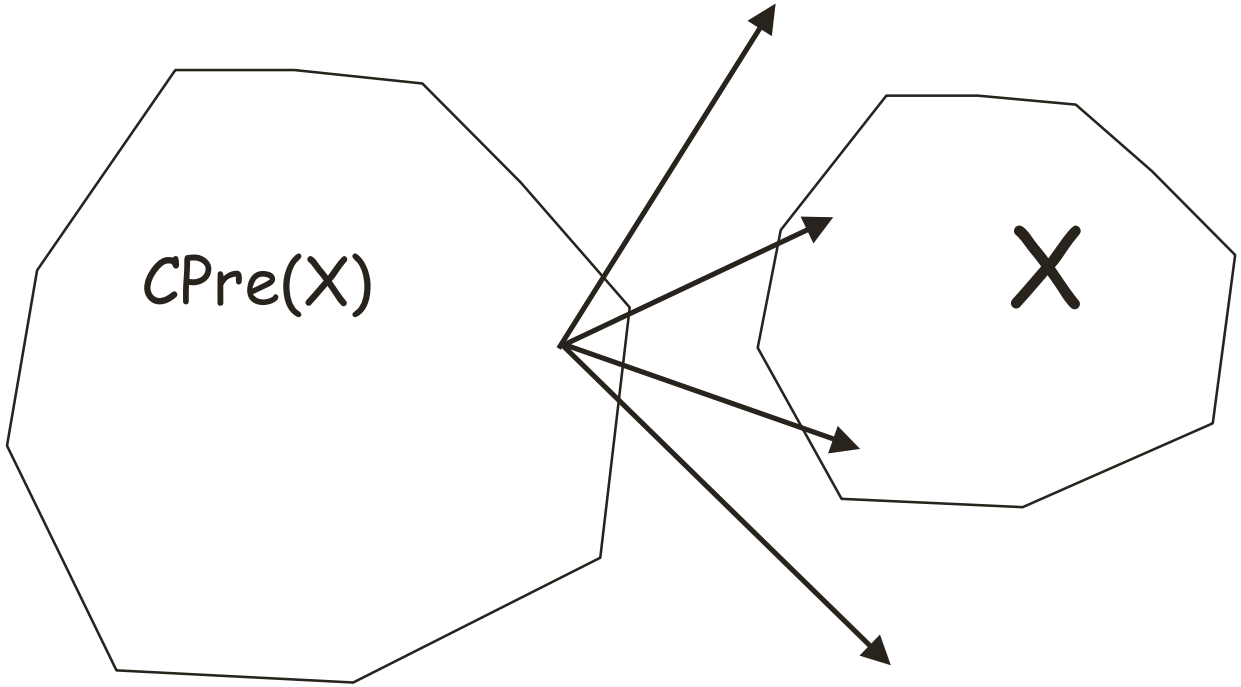
K-controllability

Given an abstraction and a partition {Alice, Bob} of the inputs of the abstraction, the abstraction is k-controllable by Bob if no matter what input Alice chooses, Bob is able to choose an input such that the fail signal is low for k cycles.

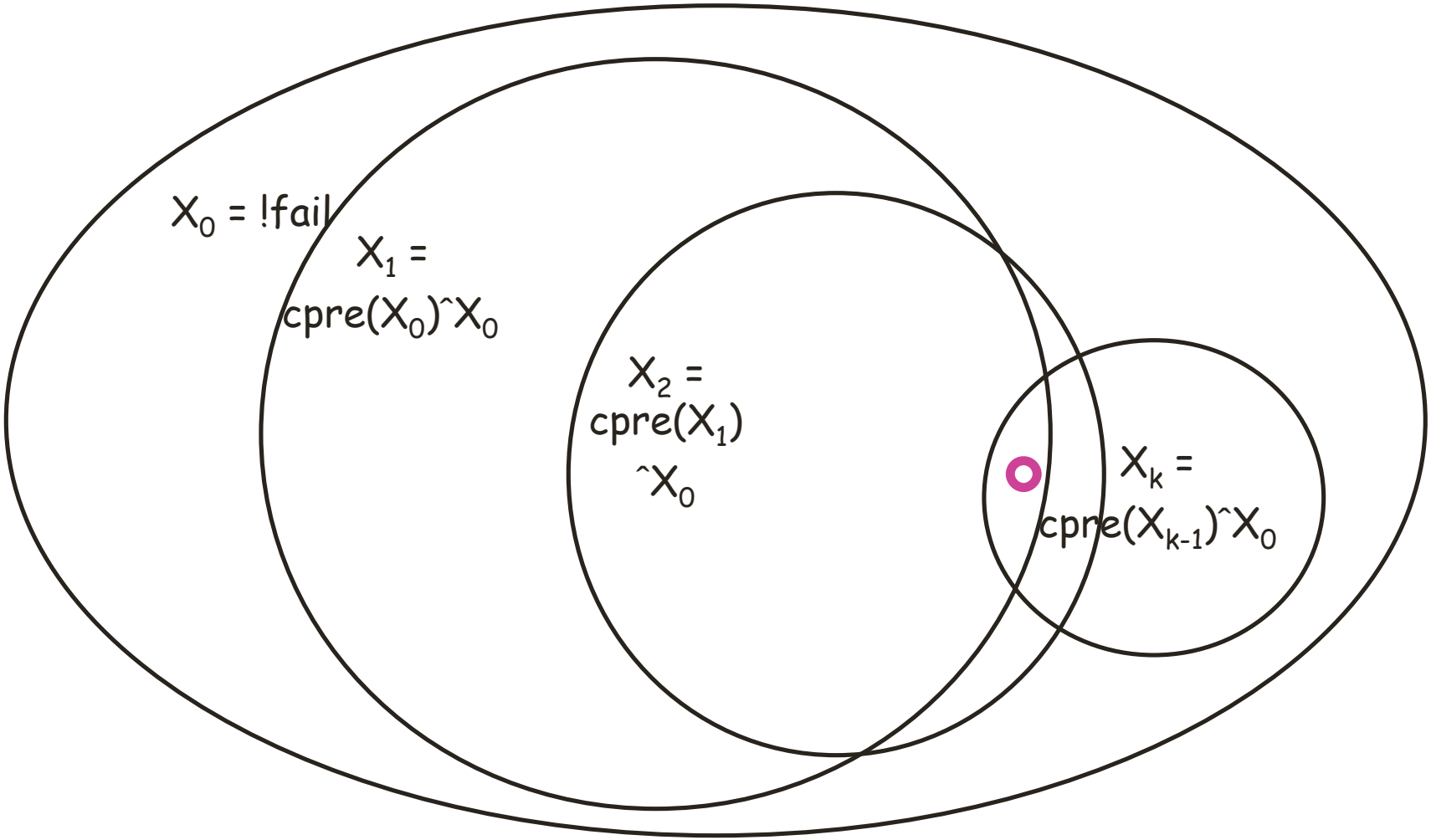


Controllable Predecessor

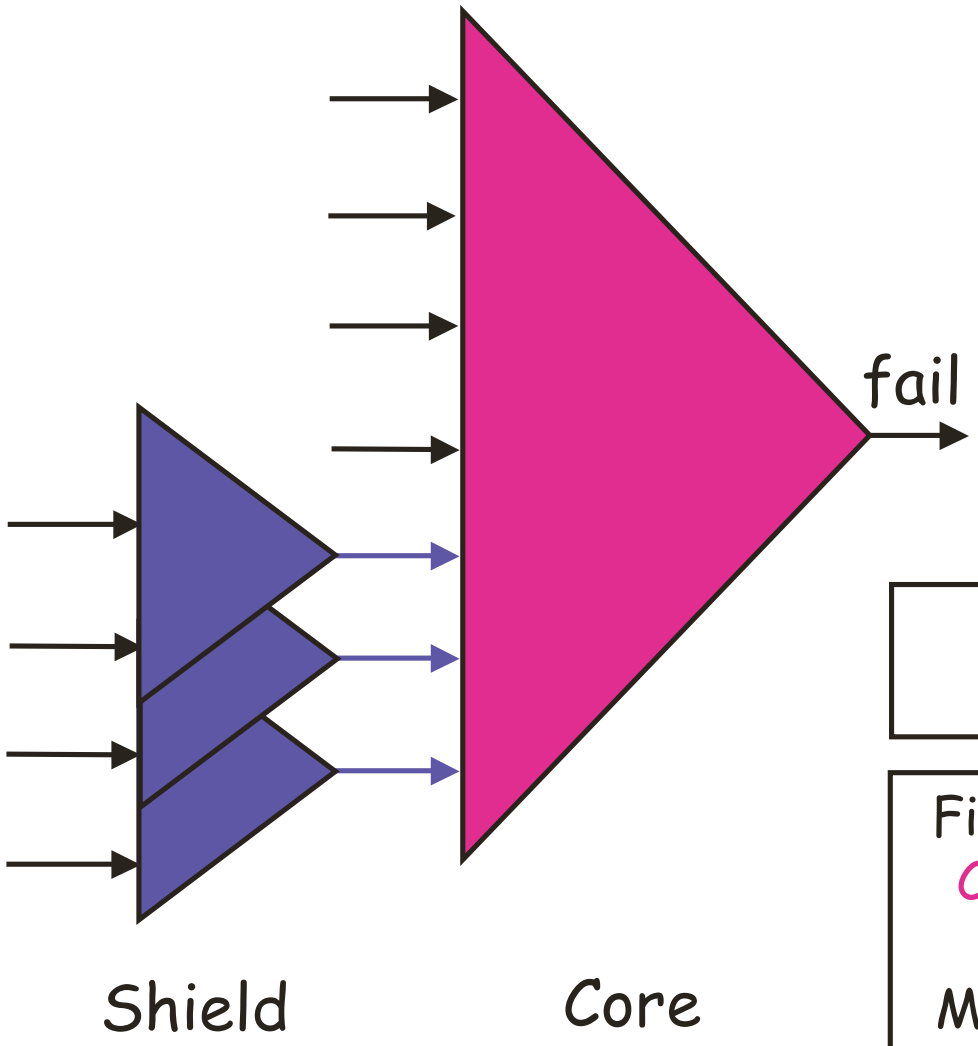
Alice/Bob



$$CPre(X) = \forall A \; \exists B \; . \; (T_{abs} \wedge next(X))$$



K-controllable by Bob if initial state is in X_k

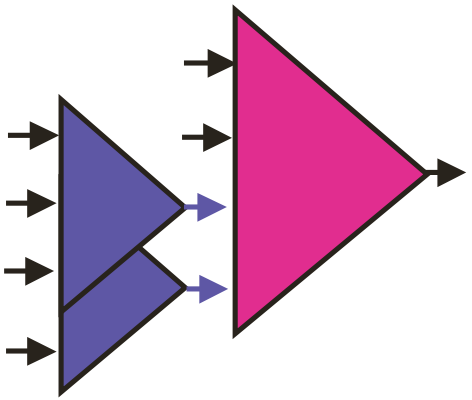


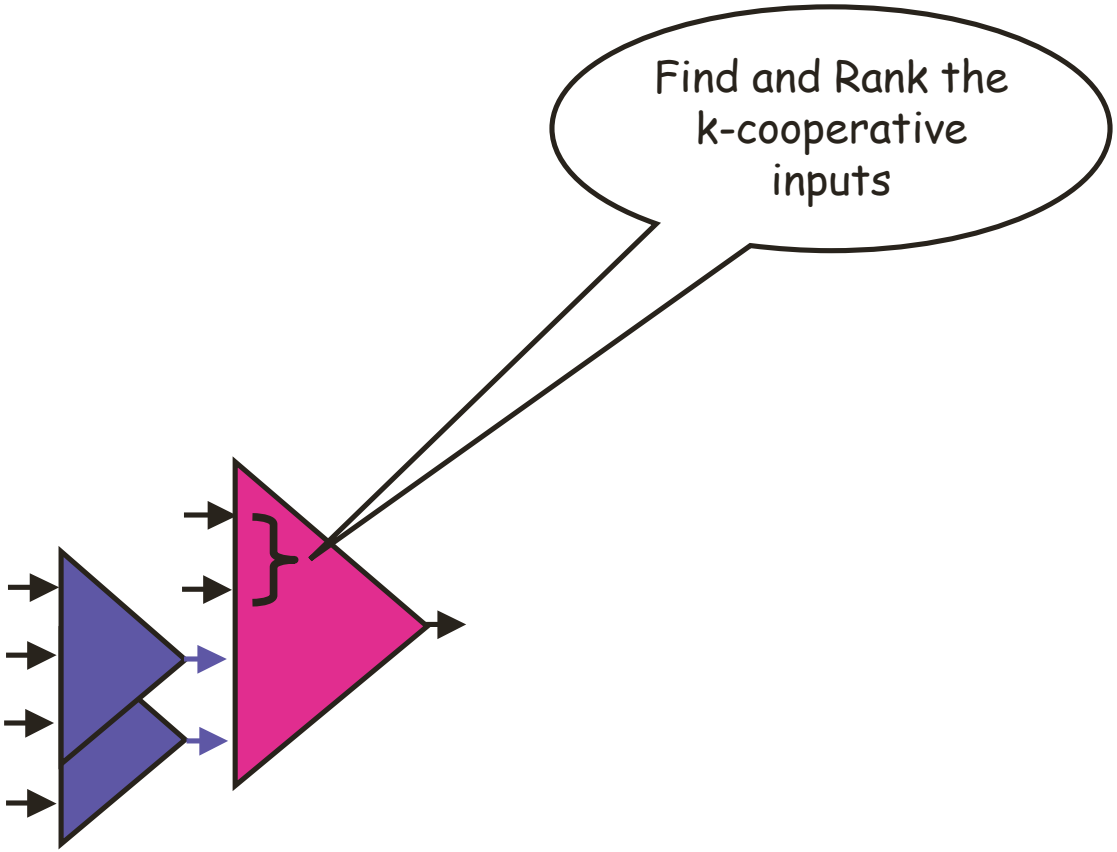
Abstract Model = (Shield, Core)

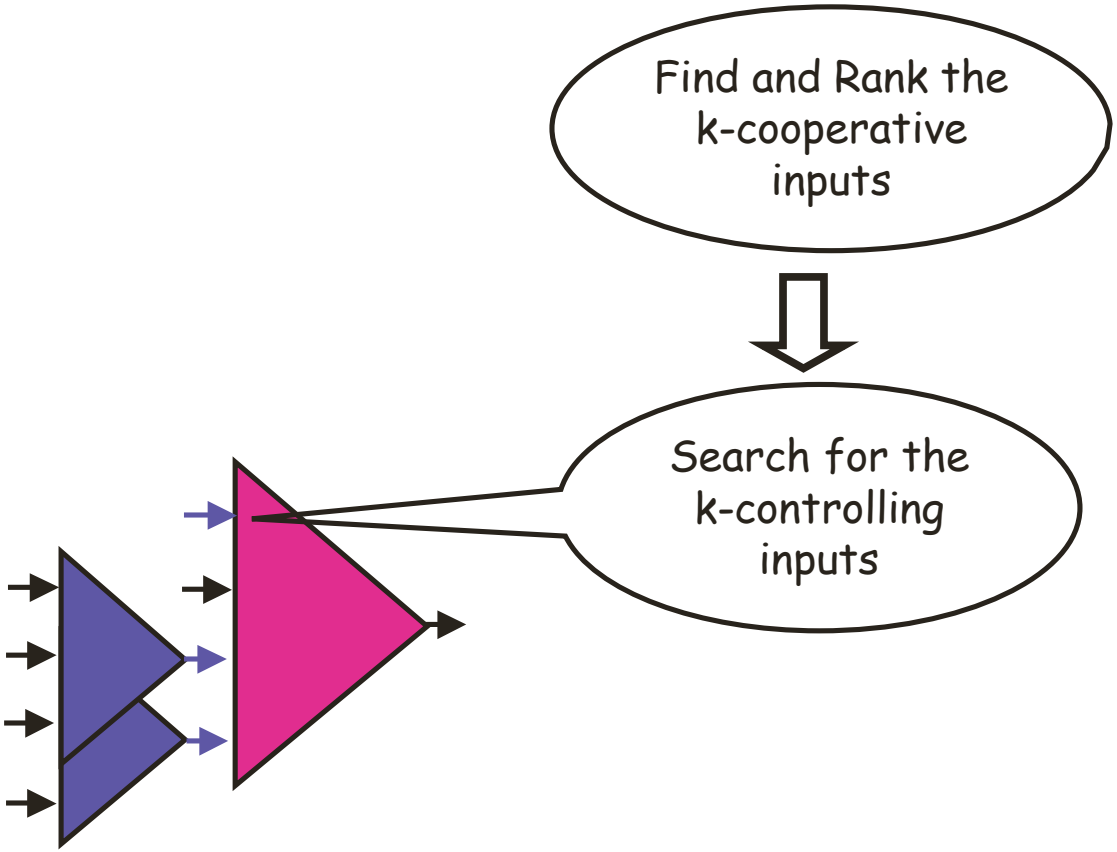
Loop invariant: Core is not controllable by Shield

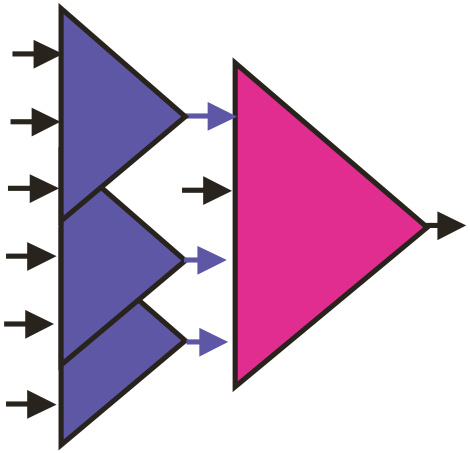
Find a K-controlling input of the Core and give it to the Shield;

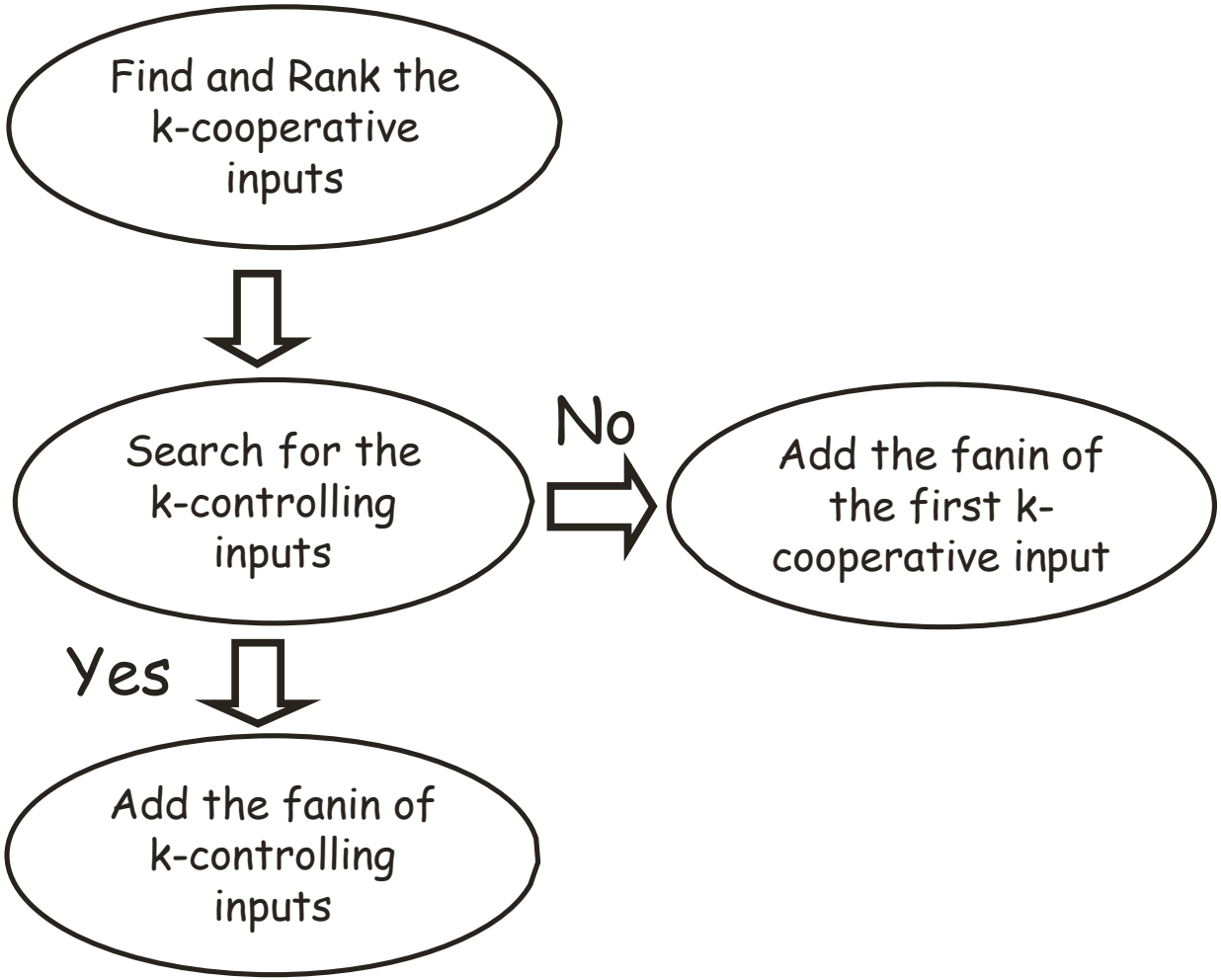
Move registers from the Shield to the Core one-by-one until the loop invariant holds again

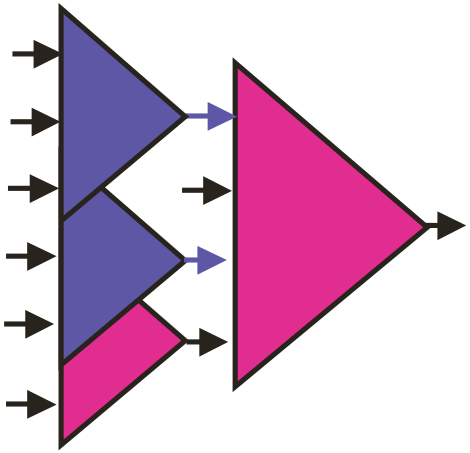



Find and Rank the
k-cooperative
inputs


Find and Rank the k-cooperative inputs
Search for the k-controlling inputs




Find and Rank the k-cooperative inputs
Search for the k-controlling inputs
No
Add the fanin of the first k-cooperative input
Yes
Add the fanin of k-controlling inputs



Find and Rank the k-cooperative inputs

Search for the k-controlling inputs

No

Add the fanin of the first k-cooperative input

Yes

Add the fanin of k-controlling inputs

Move some fanins of the shield inputs into the core

Rank k-Cooperative Variables

- Conflict registers that are inputs of shield are removed from the list
- Remaining conflict registers are ranked
  - Frequencies of conflict (high)
  - Persistence (beg to be selected)
  - Sequential Distances (close)
  - Input Widths (small)
    - Number of primary inputs in the support of transition function
- Try to find k-controlling variable in the top 3 ranked registers
  - Found → Use it
  - Not found → Use the top register

Outline

- What is functional verification?
- What is formal property verification (FPV)?
- FPV techniques
- Abstraction refinement for FPV
- *Experimental results*

Comparisons

- Technologies
  - Old RFN [WHL+01]
  - GRAB [WLJ+03]
    - All the inputs are scored according to a game-theoretic formula
      - Highest scored does not imply k-controlling
      - k-controlling does not imply highest scored
  - Interpolant [McM03]
    - SAT-based
- Testcases
  - 7 properties for 6 industrial designs
  - 750 MHz SPARC, 4GB memory, Solaris 5.8

Results – Runtime

	#gates/#rgtrs	Game RFN	RFN	mGrab	INT
P1	481/60	2246.3s	3826.6s	3333.9s	>10hr
P2	8372/697	1091.9s	13555.6s	>10hr	65.1s
P3	61552/4986	737.0s	310.2s	>10hr	194.2s
P4	77545/2122	202.8s	1049.0s	>10hr	>10hr
P5	127229/4891	10004.9s	10027.2s	>10hr	>10hr
P6	127261/4895	8311.6s	10920.5s	>10hr	>10hr
P7	137365/4494	230.3s	340.7s	>10hr	>10hr

Number of Registers in Abstract Model

	Game RFN	RFN	K-cntrl & k-coop
P1	51	57	38
P2	62	75	28
P3	21	17	7
P4	10	34	10
P5	51	51	34
P6	54	60	32
p7	13	23	13

Conclusion

- Complexity is less correlated to design complexity
  - Only perform expensive computation on abstract model
  - More scalable than interpolant
  - Much more efficient than GRAB
- Can use interpolant or SAT as proof engine for abstract model
- Apply similar scheme to verify timed or hybrid systems?