Synthesis for Verification

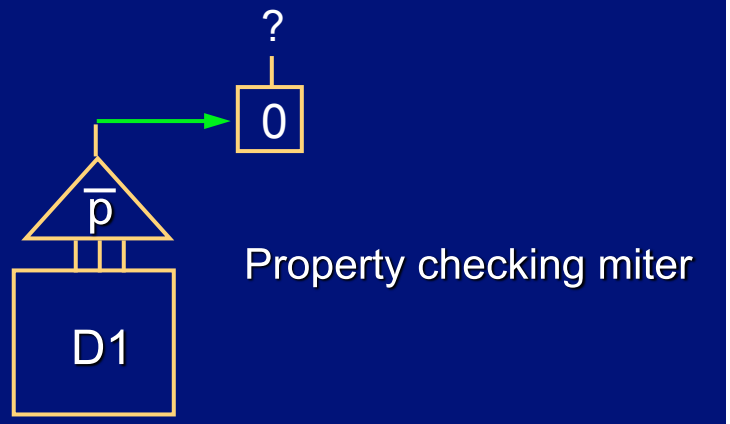
Robert Brayton, A. Mishchenko

BVSRC
UC Berkeley

Sequential Verification

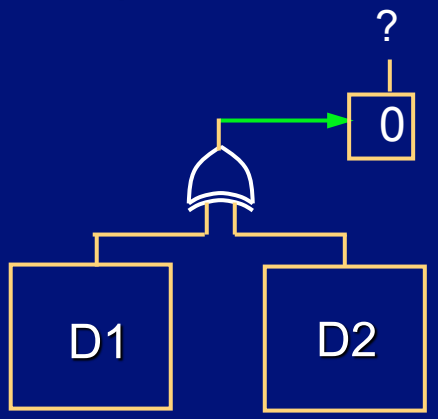
- Property checking
  - Create miter from the design and the safety property
  - Special construction for liveness
    - Biere, Artho, Schuppan
- Equivalence checking
  - Create miter from two versions of the same design

- Assuming the initial state is given
  - the goal is to prove that the output of the miter is 0, for all states reachable from the initial state.



Property checking miter

Equivalence checking miter



Idealy

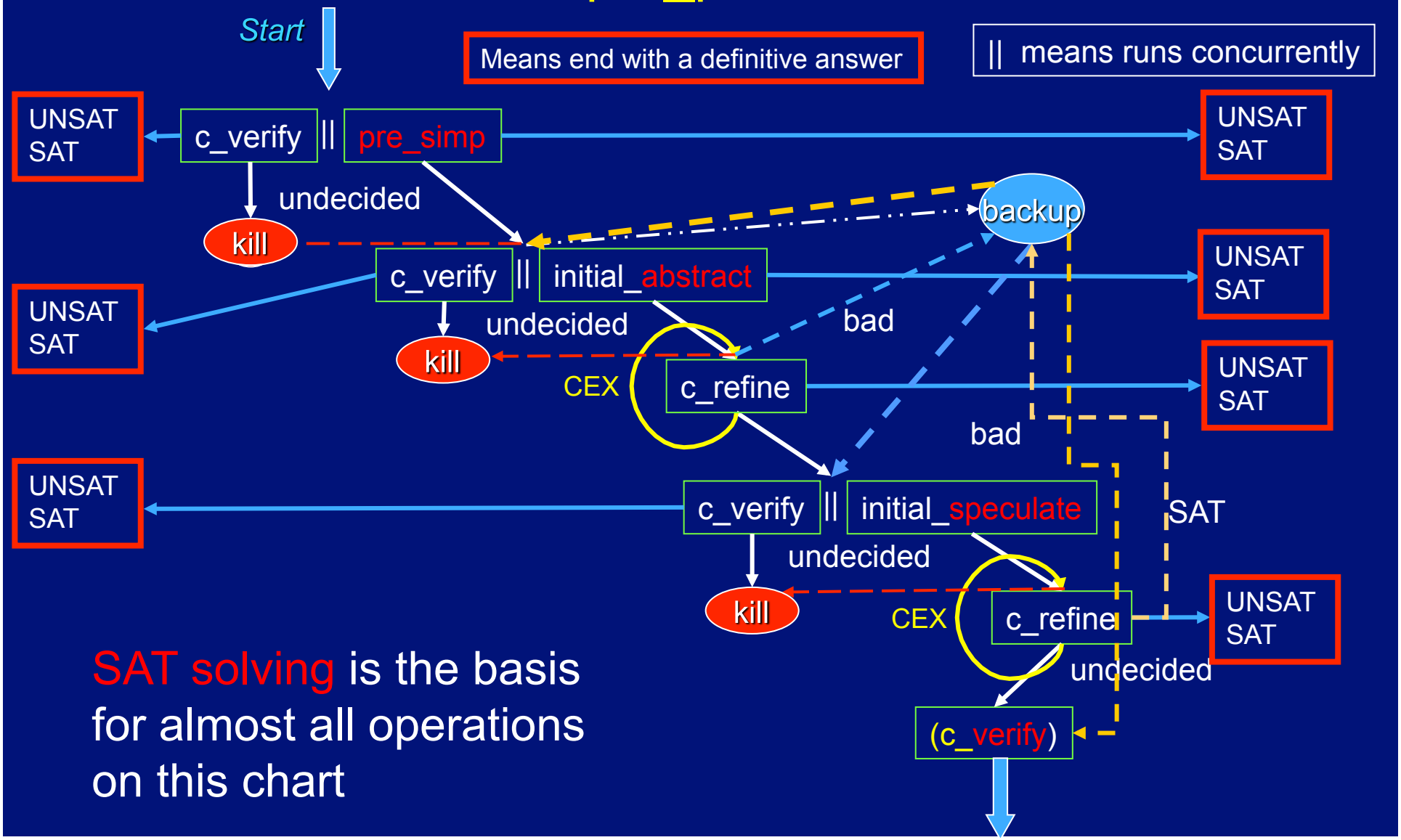
- If simplifiers were perfect
  - Could simplify output to constant 0
  - Need to sequential synthesis
    - PSPACE-complete
  - Sometimes it works
- However,
  - It is a good heuristic to simplify at first as much as can be afforded
  - Down-stream engines work better on smaller circuits.

Integrated Verification Flow

1. Simplifications
2. Abstractions
   - Localization Abstraction
   - Speculation
3. High effort verification

Concurrent Prover Flow - hybrid

super_prove



Synthesis for SAT
minimizing CNF

- SAT is the basis for ~95% of all operations in current verification methods
- Important to map circuits into CNF so that it is easier for SAT
  - (fewer clauses, less variables)
- The best method for this is done by "technology" mapping
  - Map into 8 input LUTs
  - Get truth table, canonicize, hash CNF
- Can make SAT solving 1.5-3 times faster.

Simplification
pre_simp

Sequential transformation

- sequential cleanup, (*scl*)
- rewriting, (*dc2, syn2*)
- retiming (minimum area and most forward),
- reparametrization, (*reparam*)
- phase abstraction,
- temporal decomposition,
- constraint extraction,
- signal correspondence, (*scorr*)

Simplification
pre_simp

- sequential cleanup, (*scl*)
- rewriting, (*dc2, syn2*)
- retiming (minimum area and most forward),
- reparametrization, (*reparam*)
- phase abstraction,
- temporal decomposition,
- constraint extraction,
- signal correspondence, (*scorr*)

Sequential cleanup

- structural hashing (*strash*)
  - Works on AIG
  - Hash on input IDs
- remove dangling logic
- ternary simulation
  - ternary simulate until fixed point
  - insert and propagate constants
  - strash

Very fast – ~1M aigs in a ~sec.

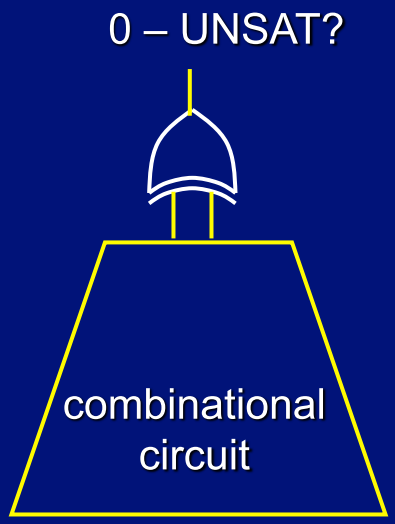
Signal Correspondence

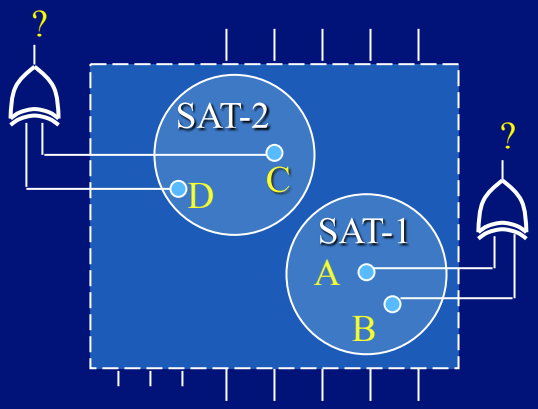
Two kinds

- *scorr*
  - *k*-step induction
    - slower, but
    - better results
  - for medium size circuits (upto ~50K aig nodes)
- *&scorr*
  - modified for large circuits (upto ~1M aig nodes)
    - faster, but
    - less quality results

Similarity with combinational SAT sweeping

Combinational SAT Sweeping





Proving internal equivalences
in a topological order

Naïve approach

- build output miter – call SAT
- works well for many easy problems

Better approach - SAT sweeping

- based on *incremental* SAT solving
  - detects possibly equivalent nodes using simulation
    - candidate constant nodes
    - candidate equivalent nodes
  - create “miter” circuit
  - runs SAT on the intermediate miters in a topological order
    - refines candidates using counterexamples
    - merges nodes if proved

Sequential SAT Sweeping *(signal correspondence)*

Similar to combinational SAT sweeping

- detects node equivalences
- But the equivalences are sequential
  - guaranteed to hold *only* on the reachable state space

- Every combinational equivalence is a sequential one
  - ➔ run combinational SAT sweeping first

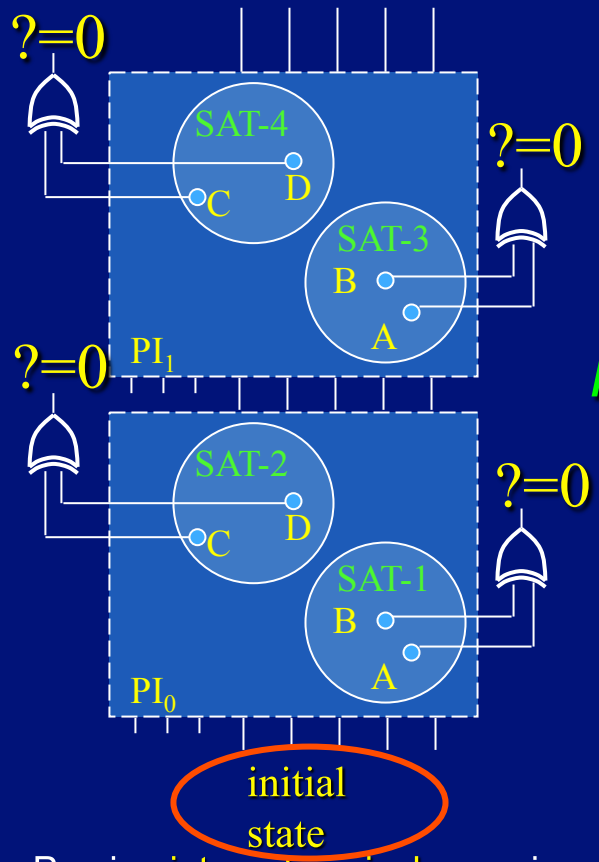
A *set* of sequential equivalences are proved by *k*-step induction

- Base case
- Inductive case
- iteration until fixed point set is proved
- Efficient implementation of induction is key!

k-step Induction (*scorr*)

Base Case
(just BMC for k cycles)

Candidate equivalences: {A = B}, {C = D}



k = 2

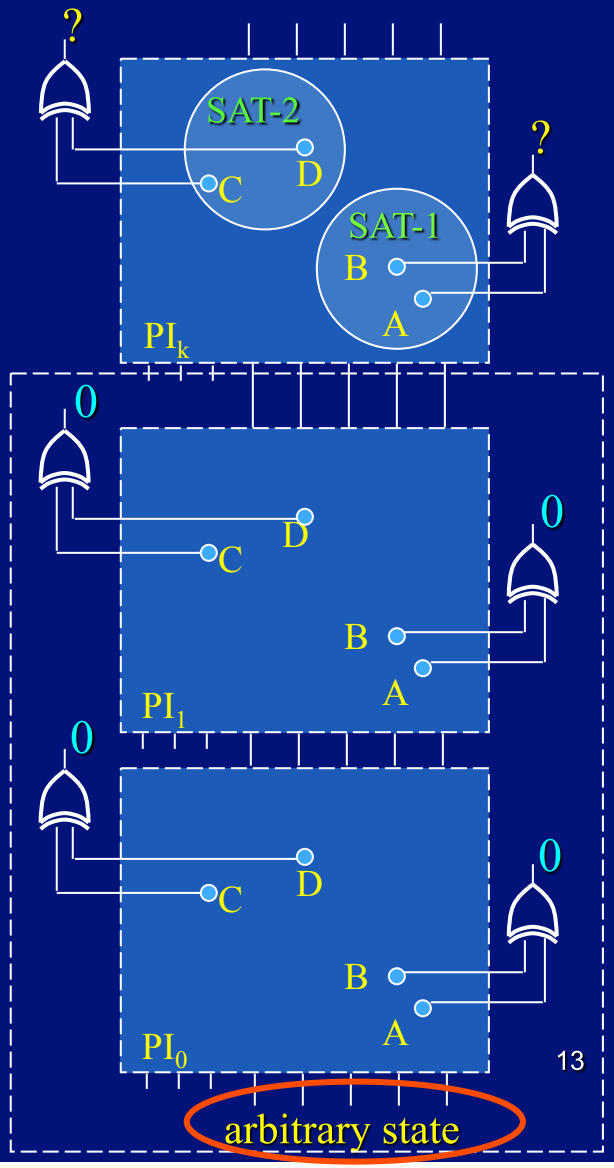
Proving internal equivalences in initialized frames 1 through k

Proving internal equivalences in a topological order in frame *k* +1

Assuming internal equivalences in uninitialized frames 1 through *k*

If proof of any equivalence fail, remove and restart

Inductive Case



Efficient Implementation

Two observations:

1. Both base and inductive cases of *k*-step induction are combinational SAT sweeping problems
   - Tricks and know-how from the above are applicable
     - base case is just BMC
   - The same integrated package can be used
     - starts with simulation
     - performs node checking in a topological order
     - benefits from the counter-example simulation
2. Speculative reduction
   - Deals with how assumptions are used in the inductive case

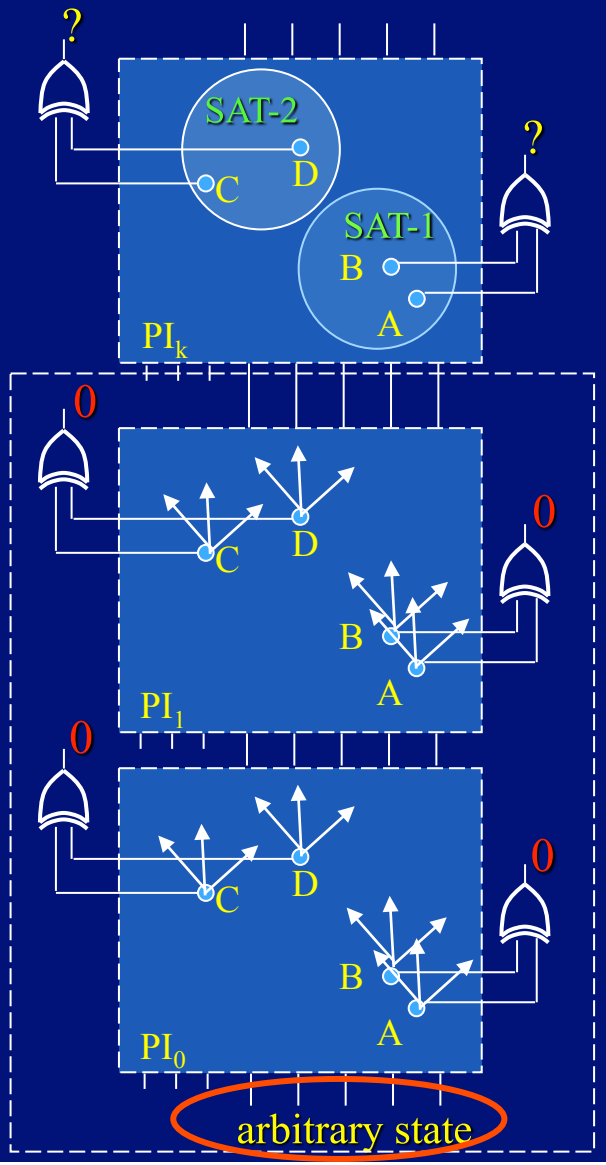
k-step Induction (*scorr*)

Inductive Case

Candidate equivalences:

{A = B}, {C = D}

k = 2



Proving internal equivalences in a topological order in frame *k*+1

Assuming internal equivalences in uninitialized frames 1 through *k*

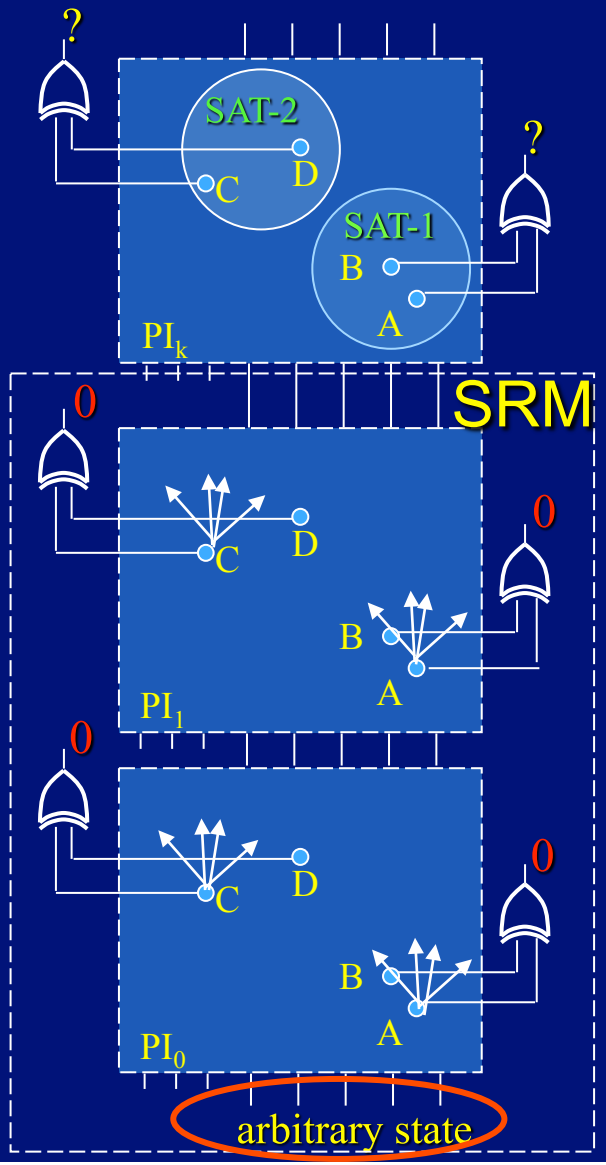
k-step Induction (*scorr*)

Inductive Case

Candidate equivalences:

{A = B}, {C = D}

k = 2



Combine fanouts

Strash

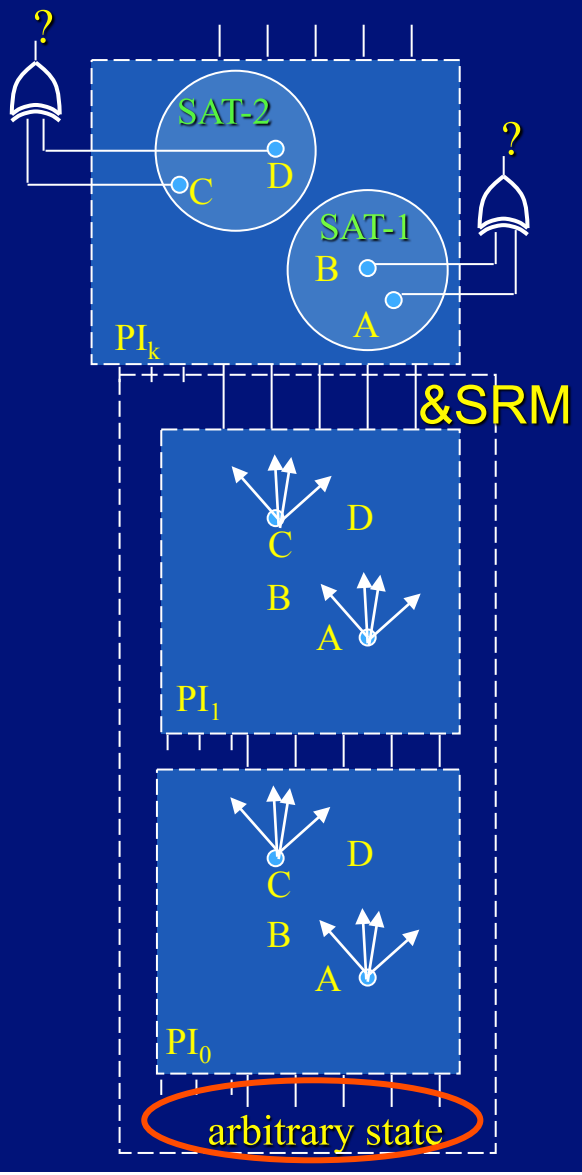
k-step Induction (*&scorr*)

Candidate equivalences:

{A = B}, {C = D}

k = 2

Inductive Case



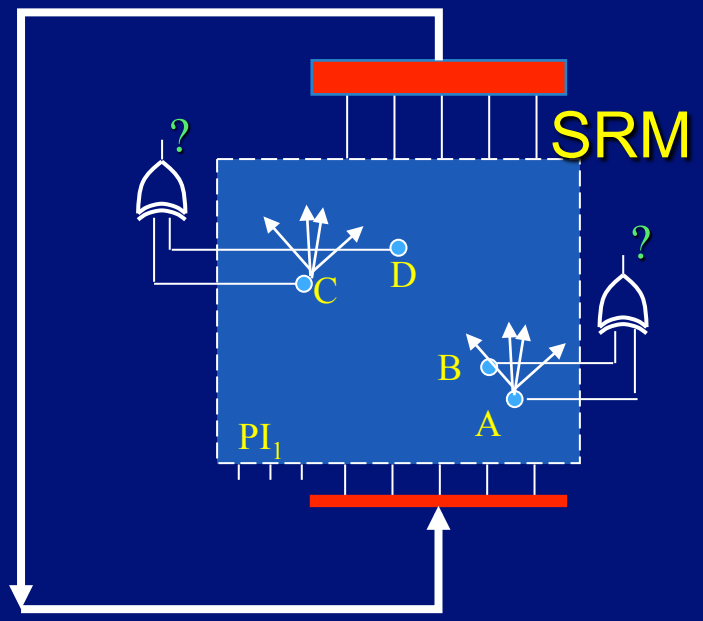
- &SRM is an abstraction of SRM
- If all ? proved UNSAT (=0)
  - all equivalences are proved
- Combine fanouts
- Keep only representative of each equivalence class (B and D removed)
  - no assumptions
- Strash and propagate constants

Verification for Synthesis

- For sequential synthesis we need scalable methods
- Three types fit the bill
  - Sequential cleanup (scl)
  - Signal correspondence (scorr, &scorr)
  - Retiming
  - Speculation

Not state minimization, state encoding, etc.

Speculation



- Speculate on equalities/constants
- Set up miters and create SRM
  - Multi-output verification problem
- Prove them for reachable states
  - Use any verification methods that work
  - Not necessarily based on k-step induction
- If any equalities/constants are disproved,
  - Eliminate, build new SRM, and start over.

Verification Engines (Summary)

- Simplifiers
  - Combinational synthesis
  - Sequential synthesis
    - Sequential cleanup
    - Retiming
    - Sequential SAT sweeping (k-step induction)
  - Re-parametrization
  - Retiming (most forward and minimum FF)
- Bug-hunters (*also part of abstraction methods*)
  - random simulation (sequential)
  - bounded model checking (BMC)
  - Property directed reachability (PDR)
  - BDD reachability
- Provers
  - *k*-step induction, with and without constraints
  - Interpolation (over-approximate reachability)
  - Property directed reachability (PDR)
  - BDDs (exact reachability)
  - Explicit state space enumeration (‘*era*’)

Conclusions

- Simplification is a major contributor to efficient verification
  - initially
  - during abstractions
  - for generating small CNF
- Needs a set of fast sequential synthesis methods
  - scl
  - scorr
  - &scorr
- Sequential synthesis is becoming of more interest to industry
  - Verification engines can be used

end