Symbolic Trajectory Evaluation (STE):
Automatic Refinement and Vacuity Detection

Orna Grumberg
Technion, Israel

Marktoberdort 2007

Agenda

- Model checking
- Symbolic Trajectory Evaluation
- Basic Concepts
- Automatic Refinement for STE
- Vacuity in STE

System Verification

Given a (hardware or software) system
and a specification,
does the system satisfy the specification?

Not decidable!

We restrict the problem to a decidable one:

- **Finite-state** reactive systems
- **Propositional** temporal logics

Finite state systems

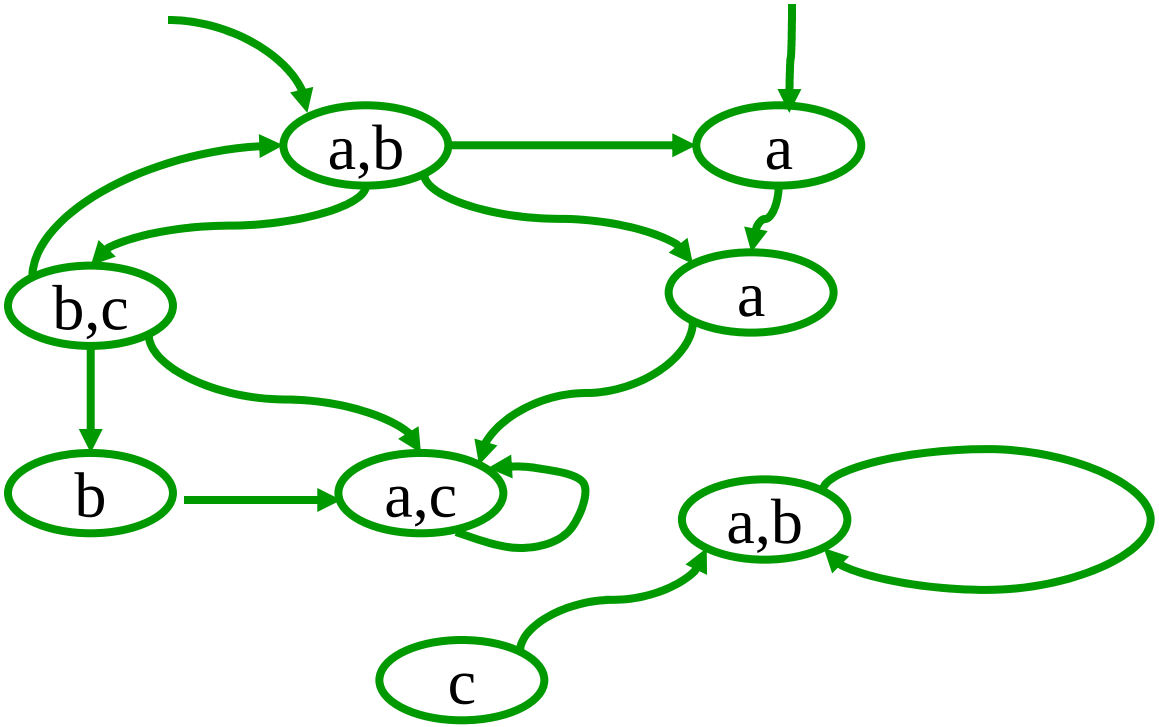
- hardware designs
- Communication protocols
- High level (abstract) description of non finite state systems

Properties in temporal logic

- **mutual exclusion:**
  always $\neg(\ cs_1 \wedge cs_2)$
- **non starvation:**
  always (request $\Rightarrow$ **eventually** grant)
- **communication protocols:**
  ($\neg$ get-message) **until** send-message

Model of a system

Kripke structure / transition system



Model of systems

M=<S, I, R, L>

- **S** - Set of states.
- **I** $\subseteq$ S - Initial states.
- **R** $\subseteq$ S x S - **Total** transition relation.
- **L:** $S \rightarrow 2^{AP}$ - Labeling function.

AP – Set of **atomic propositions**

$\pi = s_0 s_1 s_2 \ldots$ is a **path** in M from s iff

$s = s_0$ and

for every $i \geq 0$: $(s_i, s_{i+1}) \in R$

Propositional temporal logic

AP – a set of atomic propositions

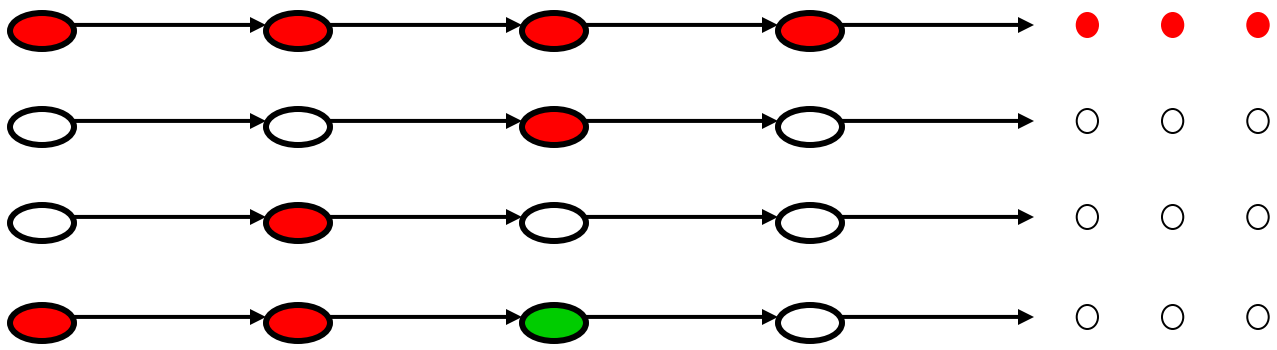
Temporal operators:

Gp

Fp

Xp

pUq



Path quantifiers: A for all path

E there exists a path

Model Checking

An efficient procedure that receives:

- A finite-state model describing a system
- A temporal logic formula describing a property

It returns

yes, if the system has the property

no + Counterexample, otherwise

Model Checking

- Emerging as an industrial standard tool for hardware design: Intel, IBM, Cadence, Synopsys,…
- Recently applied successfully also for software verification: NASA, Microsoft, ETH, CMU, …

Model checking

A basic operation: Image computation

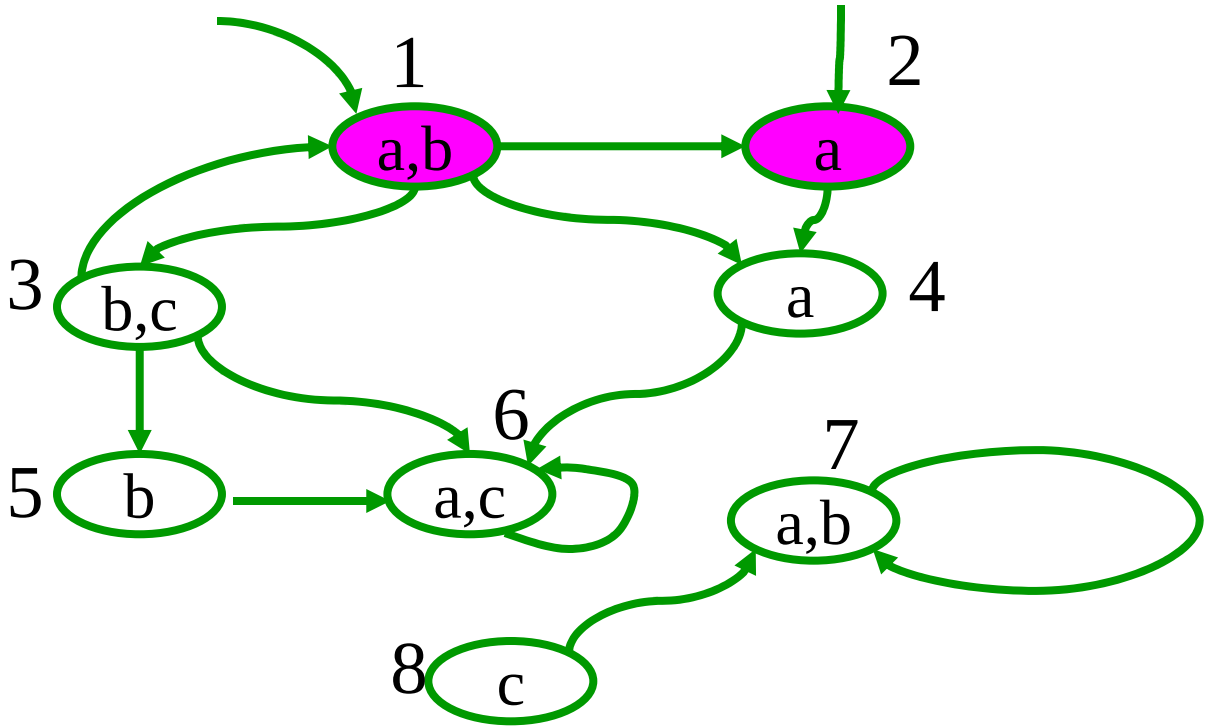
Given a set of states Q, Image(Q)
returns the set of successors of Q

Image(Q) = $\{ s' \mid \exists s \, [R(s,s') \wedge Q(s)] \}$

Model checking AGp on M

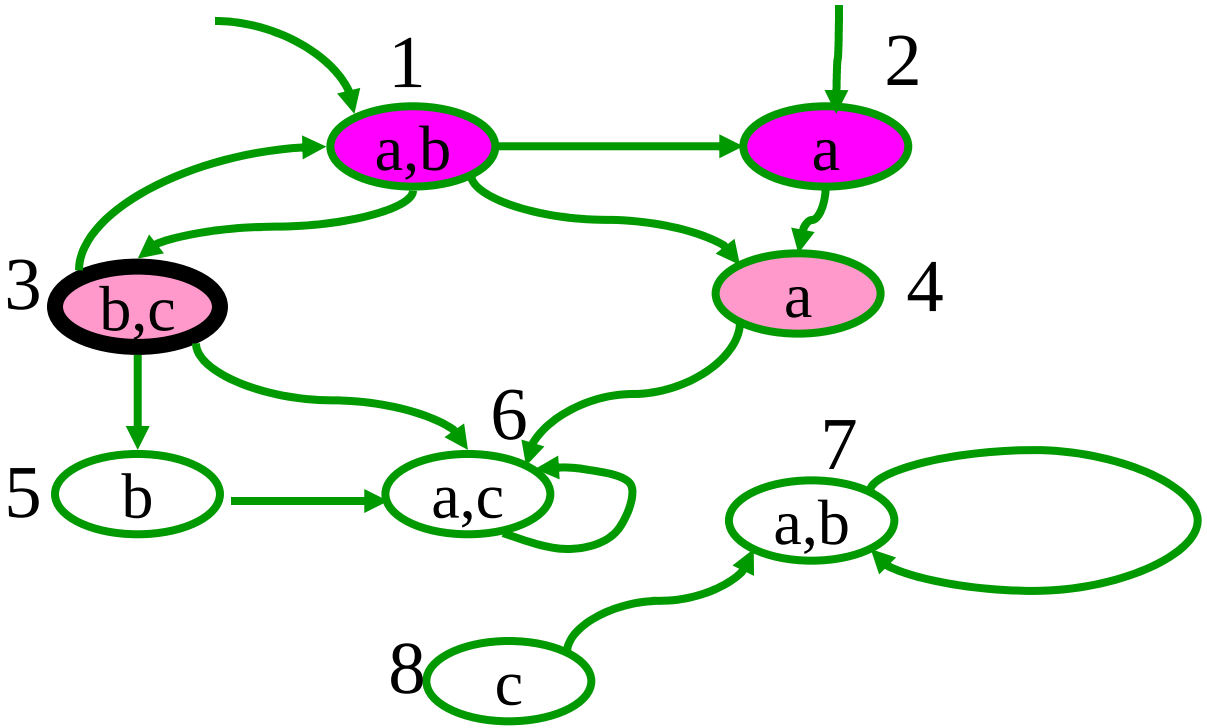
- Starting from the initial states of M, iteratively compute the set of successors.
- At each iteration check whether it reached a state which satisfies $\neg p$.
  - If so, declare a failure.
- Stop when no new states are found.
  - Result: the set of reachable states.

Reachability + checking AG a



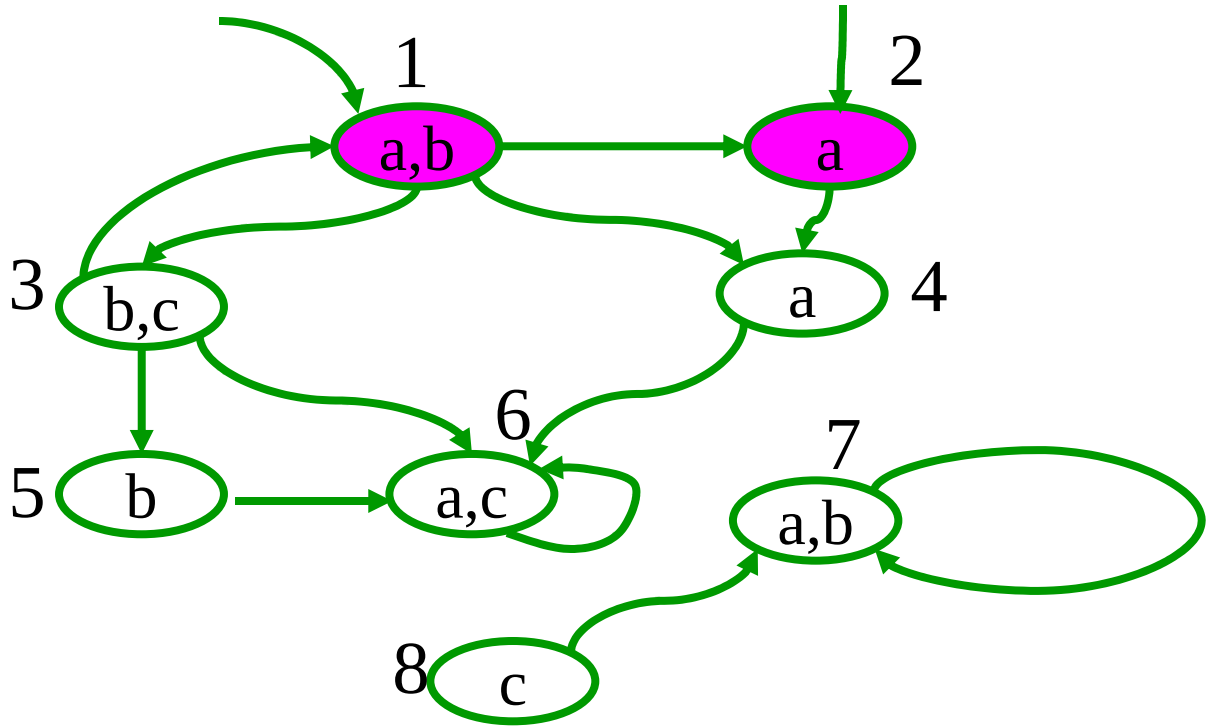
Reach = New = I = { 1, 2 }

Return: M $\not\models$ AG a



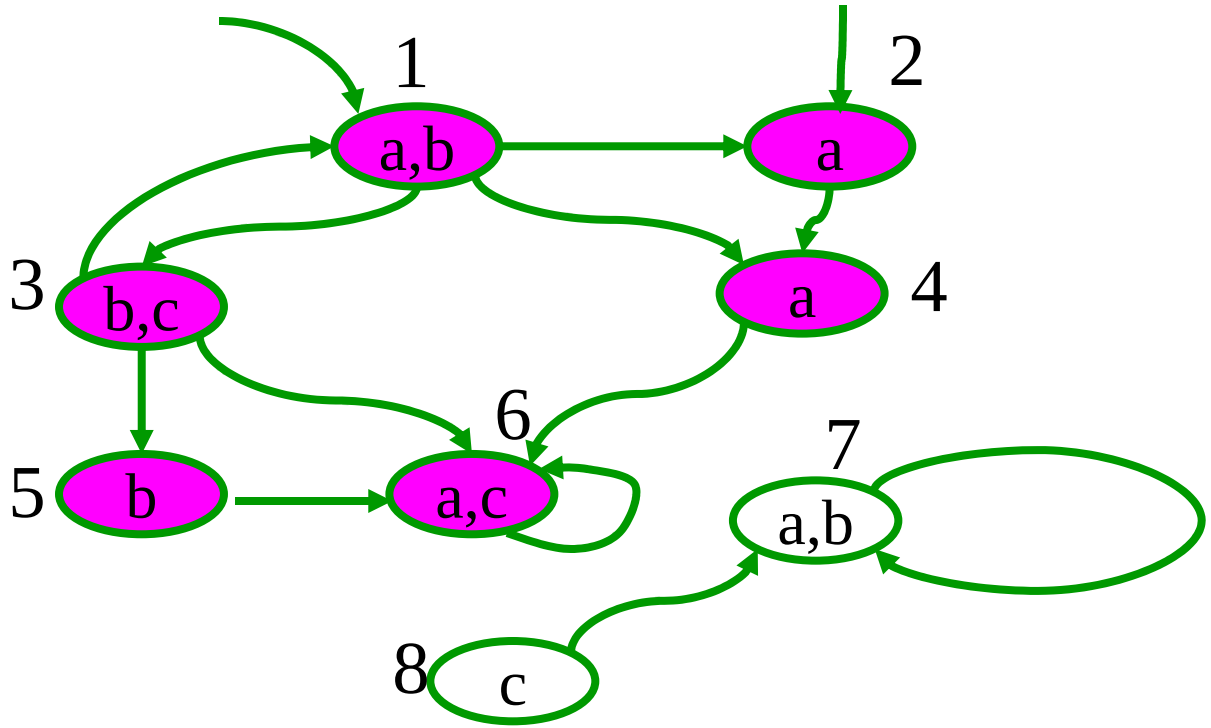
Failure: **New $\not\subset$ S_a**

Reachability + checking AG (a∨b)



Reach = New = I = { 1, 2 }

Return: Reach, M |= AG (a∨b)



Reach = {1, 2, 3, 4, 5, 6} **New = emptyset**

Main limitation:

The state explosion problem:

Model checking is efficient in time but suffers from high space requirements:

The number of states in the system model grows exponentially with

- the number of variables
- the number of components in the system

Symbolic model checking

A solution to the state explosion problem which uses **Binary Decision Diagrams (BDDs)** to represent the **model** and **sets of states**.

- Can handle systems with **hundreds** of Boolean variables

Binary decision diagrams (BDDs)

- Data structure for representing Boolean functions
- Often **concise** in memory
- **Canonical** representation
- Most **Boolean operations** on BDDs can be done in **polynomial time** in the BDD size

BDDs in model checking

- Every set A can be represented by its **characteristic function**

$$f_A(u) = \begin{cases} 1 & \text{if } u \in A \\ 0 & \text{if } u \notin A \end{cases}$$

- If the elements of A are encoded by sequences over $\{0,1\}^n$ then f_A is a **Boolean function** and can be represented by a BDD

Representing a model with BDDs

- Assume that **states** in model M are **encoded by $\{0,1\}^n$** and described by Boolean variables $v_1 \ldots v_n$
- **Reach, New** can be represented by BDDs over $v_1 \ldots v_n$
- **R** (a set of pairs of states **(s,s')**) can be represented by a BDD over $v_1 \ldots v_n \; v_1' \ldots v_n'$

Example: representing a model with BDDs

$S = \{ s_1, s_2, s_3 \}$

$R = \{ (s_1,s_2), (s_2,s_2), (s_3,s_1) \}$

State encoding:

s_1: v_1v_2=00 s_2: v_1v_2=01 s_3: v_1v_2=11

For $A = \{s_1, s_2\}$ the Boolean formula representing A:

$f_A(v_1,v_2) = (\neg v_1 \wedge \neg v_2) \vee (\neg v_1 \wedge v_2) = \neg v_1$

$f_R(v_1, v_2, v'_1, v'_2) =$

$(\neg v_1 \wedge \neg v_2 \wedge \neg v'_1 \wedge v'_2) \vee$

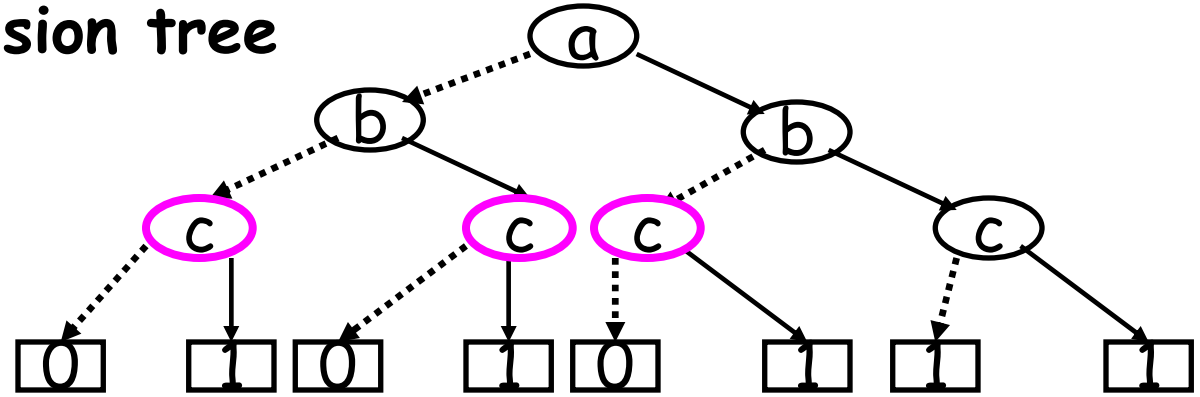
$(\neg v_1 \wedge v_2 \wedge \neg v'_1 \wedge v'_2) \vee$

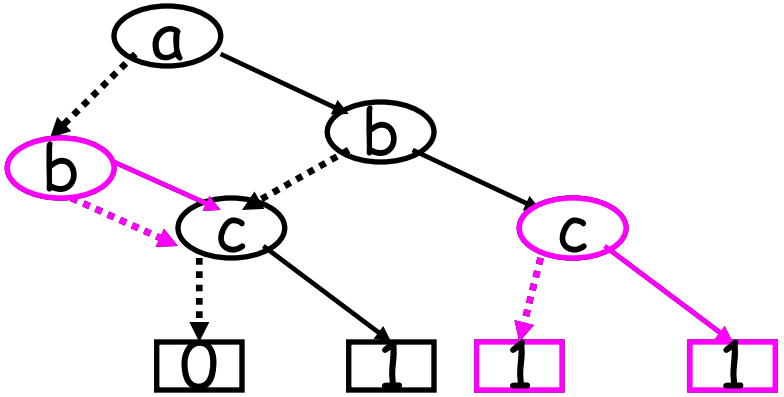
$(v_1 \wedge v_2 \wedge \neg v'_1 \wedge \neg v'_2)$

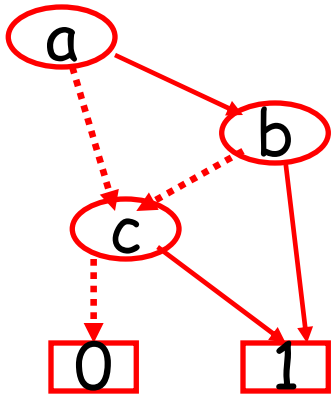
f_A and f_R can be represented by **BDDs**.

BDD for f(a,b,c) = (a ∧ b) ∨ c

Decision tree







BDD

SAT-based model checking

Another solution to the state explosion problem

- Translates the model and the specification to a propositional formula
- Uses efficient tools for solving the satisfiability problem

Since the satisfiability problem is **NP-complete**, SAT solvers are based on **heuristics**.

SAT solvers

- Using heuristics, SAT tools can solve very large problems fast
- They can handle systems with thousands variables

Bounded model checking

Most commonly used SAT-based model checking

For checking AGp:

- Unwind the model for k levels, i.e., construct all computation of length k
- If a state satisfying $\neg$p is encountered, then produce a counter example

The method is suitable for **falsification**, not verification

SAT-based model checking

- Can also handle general **temporal logic** specifications
- Can be used for **verification** by using methods such as induction and interpolation.

Bounded model checking in detail

- Construct a formula $f_{M,k}$ describing all possible computations of M of length k
- Construct a formula $f_{\varphi,k}$ expressing that $\varphi = EF\neg p$ holds within k computation steps
- Check whether $f = f_{M,k} \wedge f_{\varphi,k}$ is satisfiable

If f is satisfiable then $M \not\models AGp$

The satisfying assignment is a **counterexample**

Example – shift register

Shift register of 3 bits: <x, y, z>

Transition relation:

$R(x,y,z,x',y',z') = \quad x'=y \ \wedge \ y'=z \ \wedge \ z'=1$

error

Initial condition:

$I(x,y,z) = \ x=0 \vee y=0 \vee z=0$

Specification: $AG\ (\ x=0 \vee y=0 \vee z=0)$

Propositional formula for k=2

$$f_M = (x_0=0 \vee y_0=0 \vee z_0=0) \wedge$$
$$(x_1=y_0 \wedge y_1=z_0 \wedge z_1=1) \wedge$$
$$(x_2=y_1 \wedge y_2=z_1 \wedge z_2=1)$$

$$f_\varphi = V_{i=0,..2} (x_i=1 \wedge y_i=1 \wedge z_i=1)$$

Satisfying assignment: 101 011 111

This is a counter example!

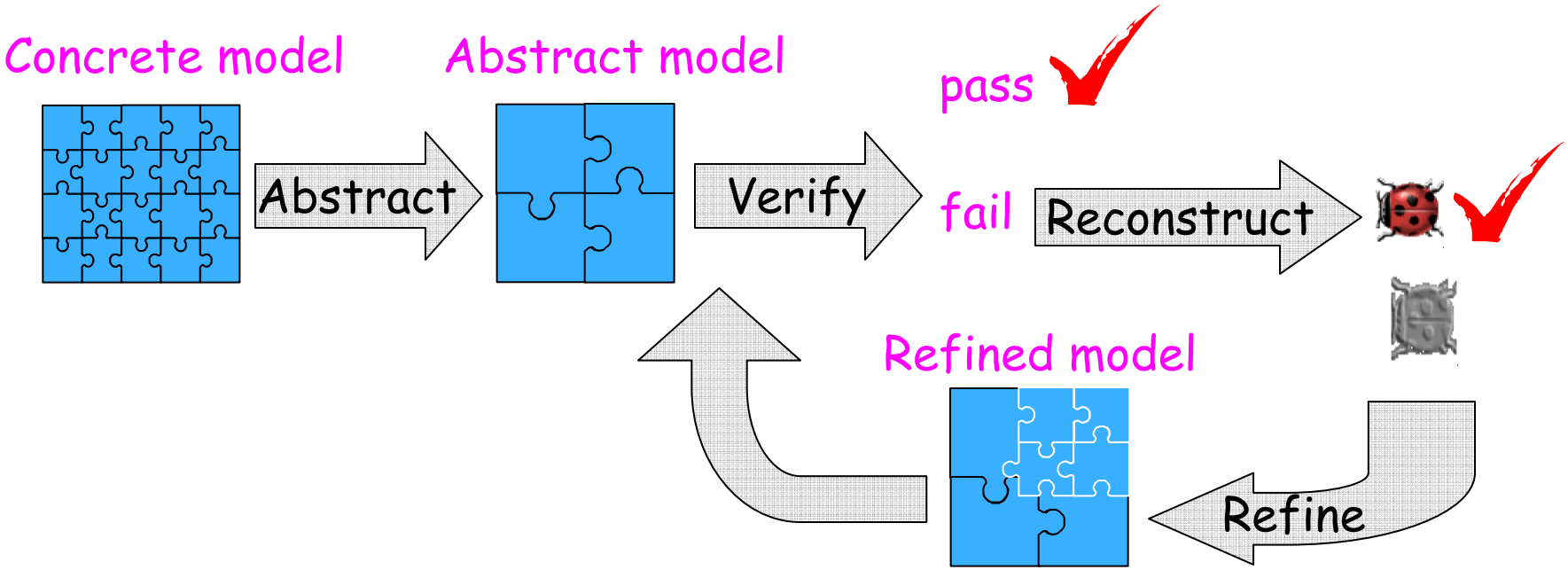
A remark

In order to describe a computation of length k by a propositional formula we need k copies of the state variables.

With BDDs we use only two copies of current and next states.

Abstraction-Refinement

A successful approach to deal with the state explosion problem in model checking



Abstraction-refinement (cont.)

M_A - abstract model M_C - concrete model

- 2-valued abstraction

  $M_A \models \varphi \Rightarrow M_C \models \varphi$

  $M_A \not\models \varphi \Rightarrow M_C$?

- 3-valued abstraction

  $M_A \models \varphi \Rightarrow M_C \models \varphi$

  $M_A \not\models \varphi \Rightarrow M_C \not\models \varphi$

  M_A ? $\Rightarrow$ M_C ?

Agenda

- Model checking
- Symbolic Trajectory Evaluation
- Basic Concepts
- Automatic Refinement for STE
- Vacuity in STE

Symbolic Trajectory Evaluation (STE)

A powerful technique for hardware model checking that can handle

- much larger hardware designs
- relatively simple specification language

Widely used in industry, e.g., Intel, Freescale

STE is given

- A circuit **M**
- A specification **A** $\Rightarrow$ **C**, where
  - **Antecedent A** imposes constraints on M
  - **Consequent C** imposes requirements on M

A and **C** are formulas in a restricted temporal logic (called **TEL**)

STE

- Works on the gate-level representation of the circuit
- Combines symbolic simulation and abstraction

Current STE

- **Automatically** constructs an abstract model for M, based on A ($M \times A$)
- **Checks** whether $M \times A \models C$
  Return:
  - **Pass:** $M \models A \Rightarrow C$
  - **Fail** + counterexample
  - **Undecided:** refinement is needed

  This is a form of 3-valued abstraction

- **Manually** refines A (and thus also $M \times A$)

Agenda

- Model checking
- Symbolic Trajectory Evaluation
- Basic Concepts
- Automatic Refinement for STE
- Vacuity in STE

Modeling a circuit

- A Circuit **M** is described as a graph whose nodes **n** are **inputs**, **gates**, and **latches**
- We refer to node **n** at different times **t**

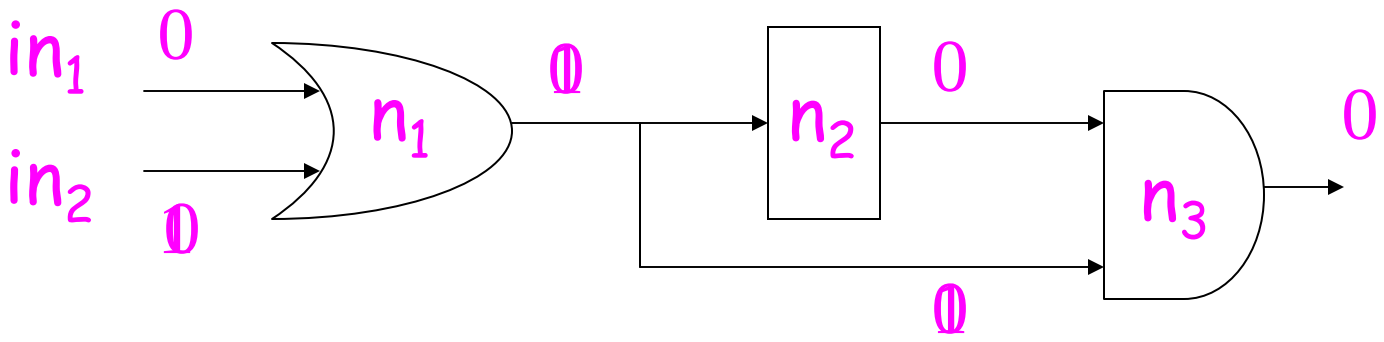
In fact, we look at an **unwinding** of the circuit for **k** times

- **k** is determined by $A \Rightarrow C$

Modeling a circuit (cont.)

- The value of an **input** node at time t is nondeterministic: 0 or 1
- The value of a **gate** node at time t depends on the values of its **source nodes** at time **t**
- The value of a **latch** node at time t depends on the values of its **source nodes** at time **t** and **t-1**

Example: a circuit



Time=0

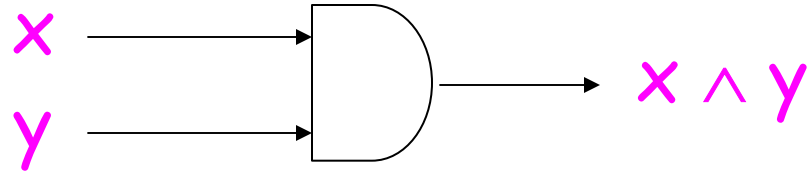
Simulation Based Verification

- Assigns values to the inputs of the model over time (as in the example)
- Compares the output values to the expected ones according to the specification

- Main drawback: the model is verified only for those specific combinations of inputs that were tested

Symbolic Simulation

- Assigns the inputs of the model with Symbolic Variables over {0,1}



- Checks all possible combinations of inputs at once
- Main drawback: representations of such Boolean expressions (e.g. by **BDDs**) are **exponential in the number of inputs**

STE solution

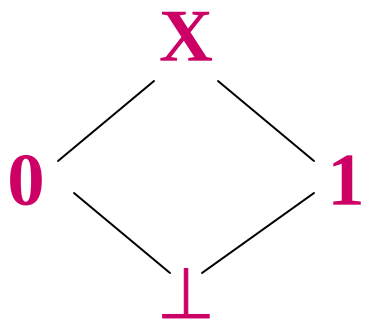
- Adds an "unknown" value **X**, in addition to 0, 1, and symbolic variables
- Needs also an "over-constrained" value $\perp$

4-valued lattice

To describe values of nodes, STE uses:

0,**1**, **X**, and ⊥

- (n,t) has value **X** when the value of n at time t is **unknown**
- (n,t) has value ⊥ when the value of n at time t is **over-constrained**

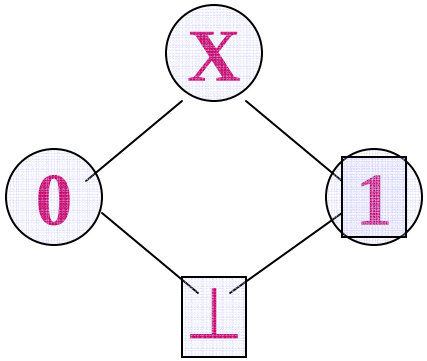


$0 \sqsubseteq x \quad 1 \sqsubseteq x \quad \bot \sqsubseteq 0 \quad \bot \sqsubseteq 1$

Operations on lattice elements

- **Meet**: a ⊓ b is the greatest lower bound of a and b

  X⊓1=1 X⊓0=0 0⊓1=⊥ …



- **Join**: a ⊔ b is the least upper bound

Lattice Semantics

- **X** is used to obtain **abstraction**
- ⊥ is used to denote a **contradiction** between a circuit behavior and the constraints imposed by the antecedent *A*
- Note: the values of concrete circuit node are only 0 and 1.

Quaternary operations

- $X \vee 1 = 1 \quad X \vee 0 = X \quad X \vee X = X$
- $X \wedge 1 = X \quad X \wedge 0 = 0 \quad X \wedge X = X$
- $\neg X = X$

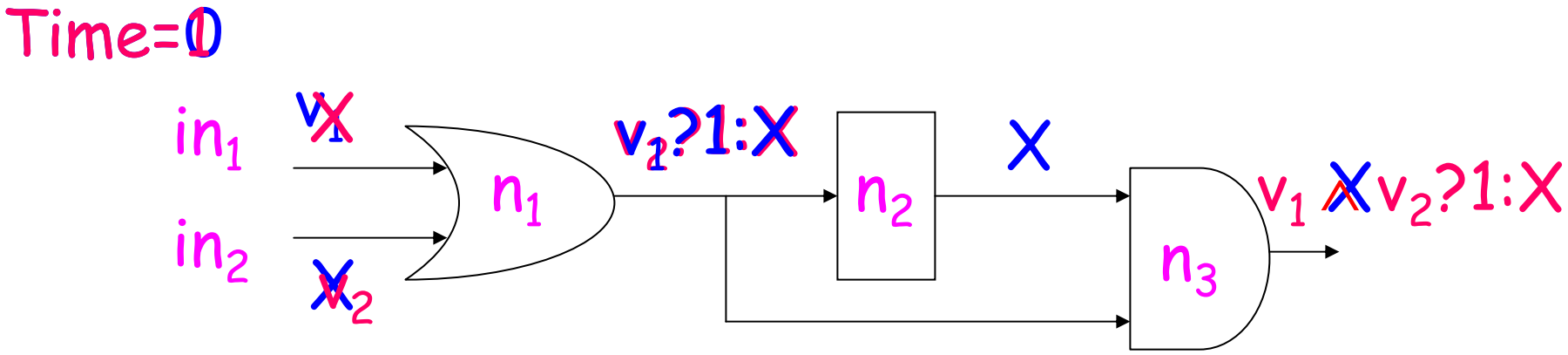
- Any Boolean expression containing $\perp$ has the value $\perp$

Symbolic execution

- STE combines abstraction with symbolic simulation to represent multiple executions at once

- Given a set of symbolic variables V, the nodes of the circuit are mapped to symbolic expressions over $V \cup \{0,1,X,\perp\}$

Example: symbolic abstract execution



Time	in_1	in_2	n_1	n_2	n_3
0	v_1	X	$v_1?1:X$	X	X
1	X	v_2	$v_2?1:X$	$v_1?1:X$	$v_1 \wedge v_2?1:X$

The difference between X and $v \in V$

- **$X \wedge \neg X = X$**
- **$v \wedge \neg v$ = false**

- Different occurrences of X do not necessarily represent the same value ("unknow")

- All occurrences of **v** represent the same value

- Each line is a symbolic state
- Trajectory: sequence of states, compatible with the behavior of the circuit

Time	in_1	in_2	n_1	n_2	n_3
0	v_1	X	v_1?1:X	X	X
1	X	v_2	v_2?1:X	v_1?1:X	$v_1 \wedge v_2$?1:X

Implementation issues

- The value of each node (n,t) is a function from V to {**0**,**1** , **X**, $\perp$ }
- BDD representation – **Dual rail**

  Two Boolean functions:

  $f_{n,t}^{1} : V \rightarrow \{ 0,1 \}$

  $f_{n,t}^{0} : V \rightarrow \{ 0,1 \}$

Dual rail

For a specific assignment to V

- $f_{n,t}^{1}(V) \wedge \neg f_{n,t}^{0}(V)$ represents **1** for (n,t)

$(f_{n,t}^{1} , f_{n,t}^{0})$	(n,t)
(1,0)	1
(0,1)	0
(0,0)	X
(1,1)	$\perp$

STE / model checking

- STE holds **local view** of the system:
  for each (n,t) separately

- Model checking holds **global view**:
  A state – values of all nodes at time t

Trajectory Evaluation Logic (TEL)

Defined recursively over **V**, where
p is a Boolean expression over **V**
n is a node
f, f_1, f_2 are TEL formulas
N is the **next-time** operator

(n is p) **(p → f)**

$(f_1 \wedge f_2)$ **(N f)**

Example: TEL formula

$$f = (\text{in1 is } v_1) \wedge N\,(\text{in}_2 \text{ is } v_2) \wedge N^2\,(v_1 \wedge v_2 \rightarrow (\text{n3 is 0}))$$

Semantics of TEL formulas

TEL formulas are interpreted over

- **Symbolic execution** σ over **V**, and
- **assignment** $\phi : V \rightarrow \{0,1\}$

- $[\phi,\sigma \models f] \in \{1, 0, X, \perp\}$

Note: (ϕ,σ) **represents an (abstract) execution,**
i.e., a series of expressions, each over $\{0,1,X,\perp\}$

Example: TEL semantics

The same ϕ is applied to f and to σ

$f = N(v_1 \wedge v_2 \rightarrow (n_3 \text{ is } 1))$

Time	in_1	in_2	n_1	n_2	n_3
0	v_1	X	$v_1?1:X$	X	X
1	X	v_2	$v_2?1:X$	$v_1?1:X$	$v_1 \wedge v_2?1:X$

For every ϕ, $[\phi,\sigma \models f\ ] = 1$

Example: TEL semantics

$f = N\ (n_3\ \text{is}\ \ (v_1 \wedge v_2 ? 1 : 0))$

Time	in_1	in_2	n_1	n_2	n_3
0	v_1	X	$v_1?1:X$	X	X
1	X	v_2	$v_2?1:X$	$v_1?1:X$	$v_1 \wedge v_2 ? 1 : X$

For $\phi(v_1 \wedge v_2)=0$, $[\phi,\sigma \models f\ ] = X$

TEL Semantics

- For every TEL formula f,
  $[\phi,\sigma \models \mathbf{f}] = \perp$ **iff** $\exists i,n: \phi(\sigma)(i)(n) = \perp$

A sequence that contains ⊥ does not satisfy any formula

TEL semantics (cont.)
(σ does not contains $\perp$)

Note: $\phi(p) \in \{0,1\}$

- $[\phi,\sigma \models (n \text{ is } p)] = 1$ **iff** $\phi(\sigma)(0)(n) = \phi(p)$

- $[\phi,\sigma \models (n \text{ is } p)] = 0$ **iff**
  $\phi(\sigma)(0)(n) \in \{0,1\}$ **and** $\phi(\sigma)(0)(n) \neq \phi(p)$

- $[\phi,\sigma \models (n \text{ is } p)] = X$ **iff** $\phi(\sigma)(0)(n) = X$

TEL semantics (cont.)

- $[\phi,\sigma \models (f_1 \wedge f_2)] = [\phi,\sigma \models f_1] \wedge [\phi,\sigma \models f_2]$
- $[\phi,\sigma \models (p \rightarrow f)] = \phi(\neg p) \vee [\phi,\sigma \models f]$
- $[\phi,\sigma \models (\mathbf{N}\ f)] = [\phi,\sigma^1 \models f]$

TEL semantics (cont.)

$[\sigma \models f\ ] = 0$ iff for some ϕ, $[\phi,\sigma \models f]=0$

$[\sigma \models f\ ] = X$ iff for all ϕ, $[\phi,\sigma \models f] \neq 0$ and

for some ϕ, $[\phi,\sigma \models f]=X$

TEL semantics (cont.)

$[\sigma \models f\ ] = 1$ iff for all ϕ, $[\phi,\sigma \models f] \notin \{0,X\}$

and for some ϕ, $[\phi,\sigma \models f]=1$

$[\sigma \models f\ ] = \perp$ iff for all ϕ, $[\phi,\sigma \models f]= \perp$

Back to STE…

Recall that our goal is to check whether

$$M \models A \Rightarrow C$$

where A imposes constraints on M and C imposes requirements

$M \times A$: Abstraction of M derived by A

The defining trajectory of M and A, denoted $M \times A$, is defined as follows:

- $M \times A$ is a symbolic execution of M that satisfies A
- For every symbolic execution σ of M

  $[\sigma \models A]=1 \quad \leftrightarrow \quad \sigma \sqsubseteq M \times A$

	n_1,t	n_2,t	n_3,t	n_4,t
$M \times A$	1	X	0	X
σ	$1/\perp$		$0/\perp$	

$M \times A$ (cont.)

- [Seger&Bryant] show that every circuit M and TEL formula f has such $M \times f$

$M \times A$ (cont.)

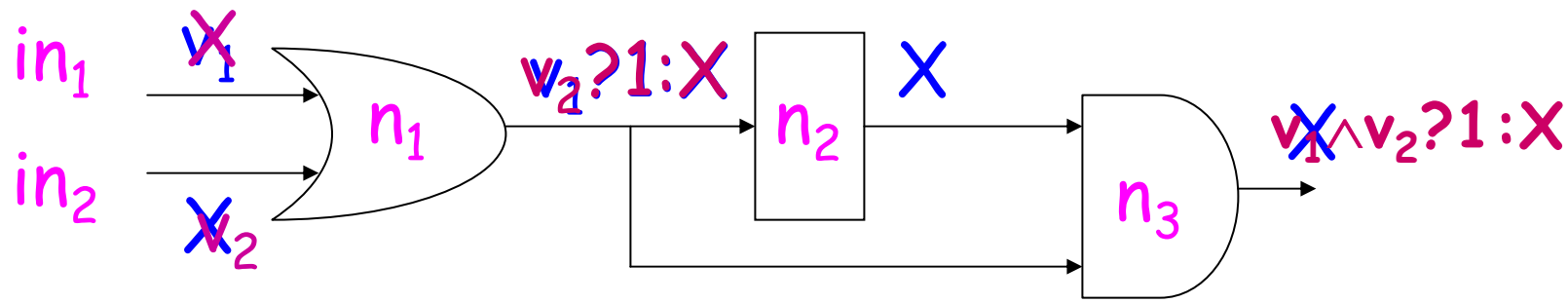
- $M \times A$ is the abstraction of all executions of M that satisfy A and therefore should also satisfy C
- If $M \times A$ satisfies C then all executions that satisfy A also satisfy C

Checking $M \models A \Rightarrow C$ with STE

- Compute the defining trajectory $M \times A$ of M and A
- Compute the truth value of $[M \times A \models C]$
  - $[M \times A \models C] = 1 \rightarrow$ Pass
  - $[M \times A \models C] = 0 \rightarrow$ Fail
  - $[M \times A \models C] = X \rightarrow$ Undecided
- The size of $M \times A$ (as described with BDDs) is proportional to A, not to M !

Example: M×A

$A = (in_1 \text{ is } v_1) \wedge N\,(in_2 \text{ is } v_2)$ $\quad C = N\,(n_3 \text{ is } 1)$



Time	in_1	in_2	n_1	n_2	n_3
0	v_1	X	$v_1?1:X$	X	X
1	X	v_2	$v_2?1:X$	$v_1?1:X$	$v_1 \wedge v_2?1:X$

Undecided results

A = (in_1 is v1) $\wedge$ N (in_2 is v2)

C = N (n_3 is 1)

In $M \times A$ the value of (n_3,1) is $v_1 \wedge v_2$?1:X

C requires (n_3,1) to be 1

For $\phi(v1 \wedge v2)=0$, $[\phi, M \times A \models C] = X$

When $v_1 \wedge v_2$ is 0, STE results in "undecided" for (n_3,1) and thus refinement of A is needed

Agenda

- Model checking
- Symbolic Trajectory Evaluation
- Basic Concepts
- Automatic Refinement for STE
- Vacuity in STE

Our Automatic Refinement Methodology

- Choose for refinement a set **Iref** of **inputs** at **specific times** that do not appear in A
- For each $(n,t) \in Iref$, $v_{n,t}$ is a **fresh** variable, not in V
- The **refined antecedent** is:

$$A_{new} = A \wedge \Lambda_{(n,t)\in Iref} N^t(n \text{ is } v_{n,t})$$

Refinement (cont.)

A_{new} has the property that:

$$M \models A \Rightarrow C \quad \Leftrightarrow \quad M \models A_{new} \Rightarrow C$$

Here we refer to the value of $A \Rightarrow C$ / $A_{new} \Rightarrow C$ over the concrete behaviors of M

Goal:

Add a **small** number of constraints to A, keeping $M \times A$ relatively small, while **eliminating** as many **undecided results** as possible

Remark: Eliminating only **some** of the undecided results may still reveal "**fail**". For "**pass**", **all** of them need to be eliminated

Choose a refinement goal

We choose one refinement goal (root,tt)

- A node that appears in the consequent C
- Truth value is X
- Has minimal t and depends on minimal number of inputs

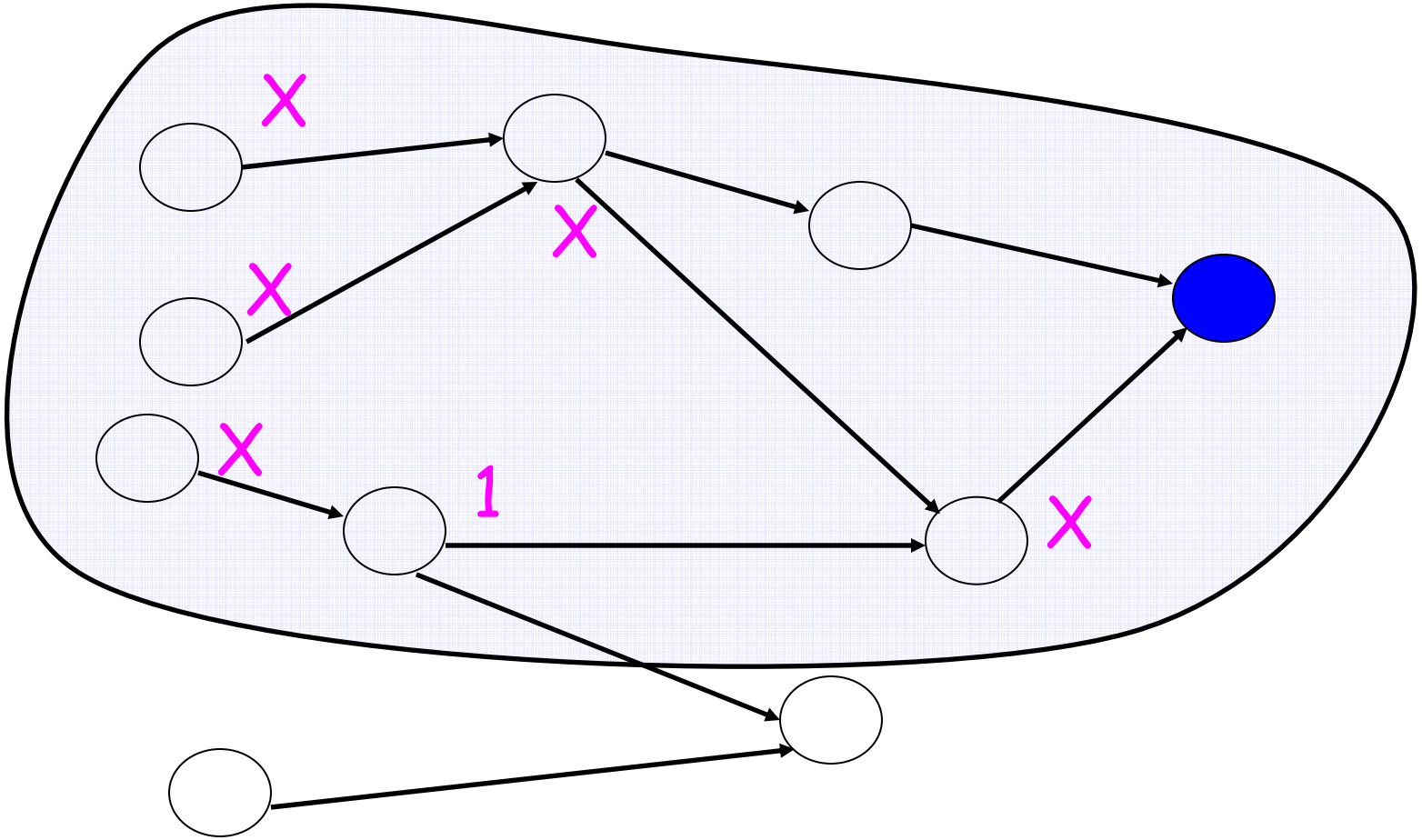
We will examine at once all executions in which (root,tt) is undecided

Choosing Iref for (root,tt)

Naïve (syntactic) solution:

Choose all (n,t) from which (root,tt) is reachable in the unwound graph of the circuit

Will guarantee elimination of all undecided results for (root,tt)


X
X
X
1
X

Better (semantic) solution

- Identify those **(n,t)** that for **some assignment** are on a **path** to **(root,tt)** along which all nodes are X
- **Iref** is the subset of the above, where n is an input
- Will still guarantee elimination of all undecided results for (root,tt)

Heuristics for smaller Iref

Choose a **subset** of Iref based on
circuit topology and **functionality**, such as:

- Prefer inputs that influence (root,tt) along several paths
- Give priority to control nodes over data nodes
- And more

Experimental Results for Automatic Refinement

We implemented our automatic refinement within the Intel's STE tool Forte.

We ran it on two nontrivial different circuits:

- Intel's Content Addressable Memory (CAM)
  - 1152 latches, 83 inputs and 5064 gates
- IBM's Calculator design
  - 2781 latches, 157 inputs and 56960 gates

We limited the number of added constraints at each refinement iteration to 1

Some more implementation issues

- Recall that the value of each node (n,t) is a function from V to {**0**,**1**, **X**, $\perp$ }
- BDD representation – **Dual rail**

  Two Boolean functions:

  $f_{n,t}^{1} : V \rightarrow \{ 0,1 \}$

  $f_{n,t}^{0} : V \rightarrow \{ 0,1$

Dual rail

$(f_{n,t}^1, f_{n,t}^0)$	(n,t)
(1,0)	1
(0,1)	0
(0,0)	X
(1,1)	$\perp$

Notation:

- $(f_{n,t}^{1} , f_{n,t}^{0})$ represents (n,t) in MxA
- $(g_{n,t}^{1} , g_{n,t}^{0})$ represents (n,t) in C

Symbolic counterexample

$$\bigvee_{(n,t)\in C} [(g_{n,t}^{1} \wedge \neg f_{n,t}^{1} \wedge f_{n,t}^{0}) \vee (g_{n,t}^{0} \wedge f_{n,t}^{1} \wedge \neg f_{n,t}^{0})]$$

Note: C is never $\perp$

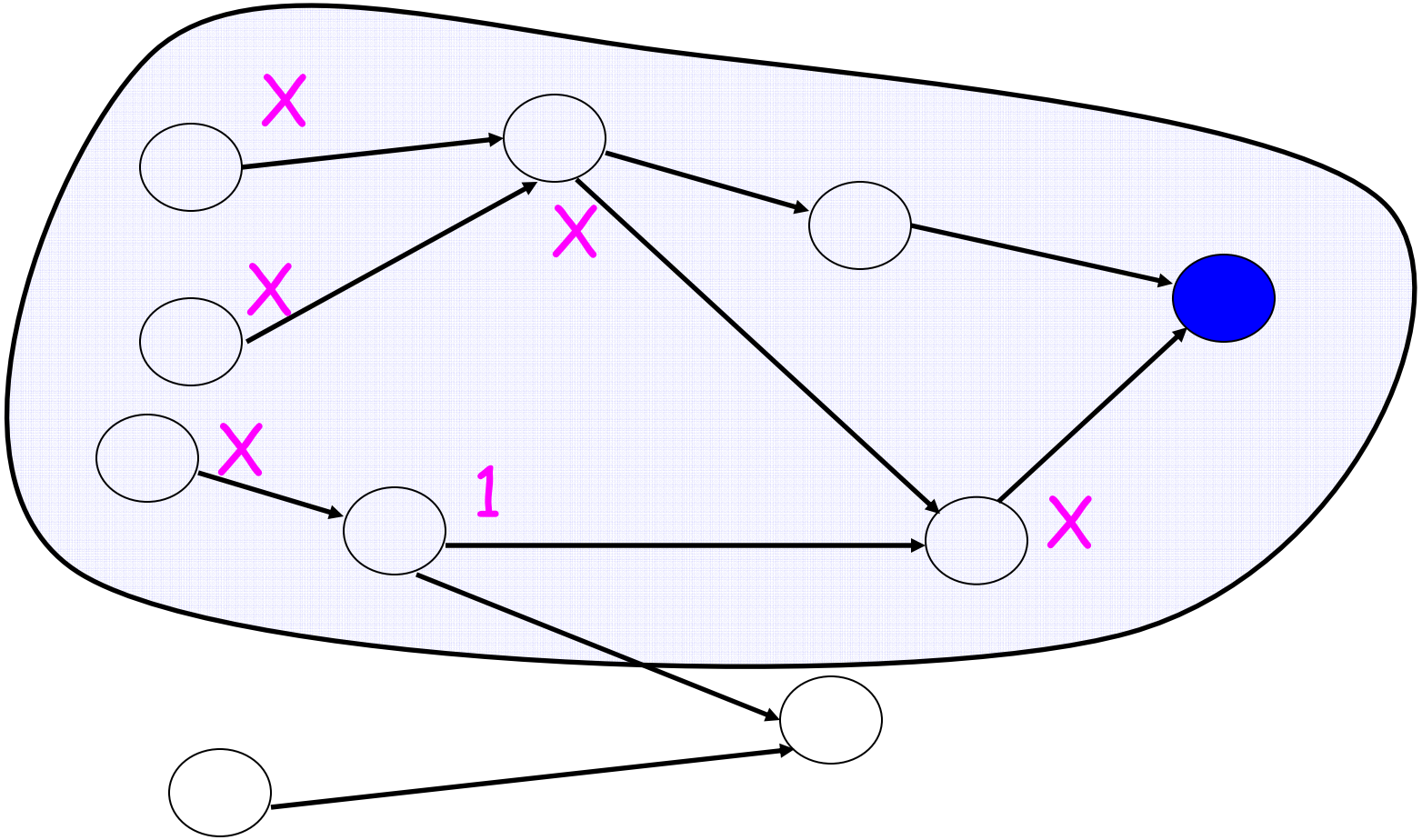
- Represents all assignments to V in which for some node (n,t), MxA and C do not agree on 0/1
- User needs to correct either the circuit or the specification

Symbolic incomplete trace

$$\bigvee_{(n,t)\in C} [(g_{n,t}^{1} \vee g_{n,t}^{0}) \wedge (\neg f_{n,t}^{1} \wedge \neg f_{n,t}^{0})]$$

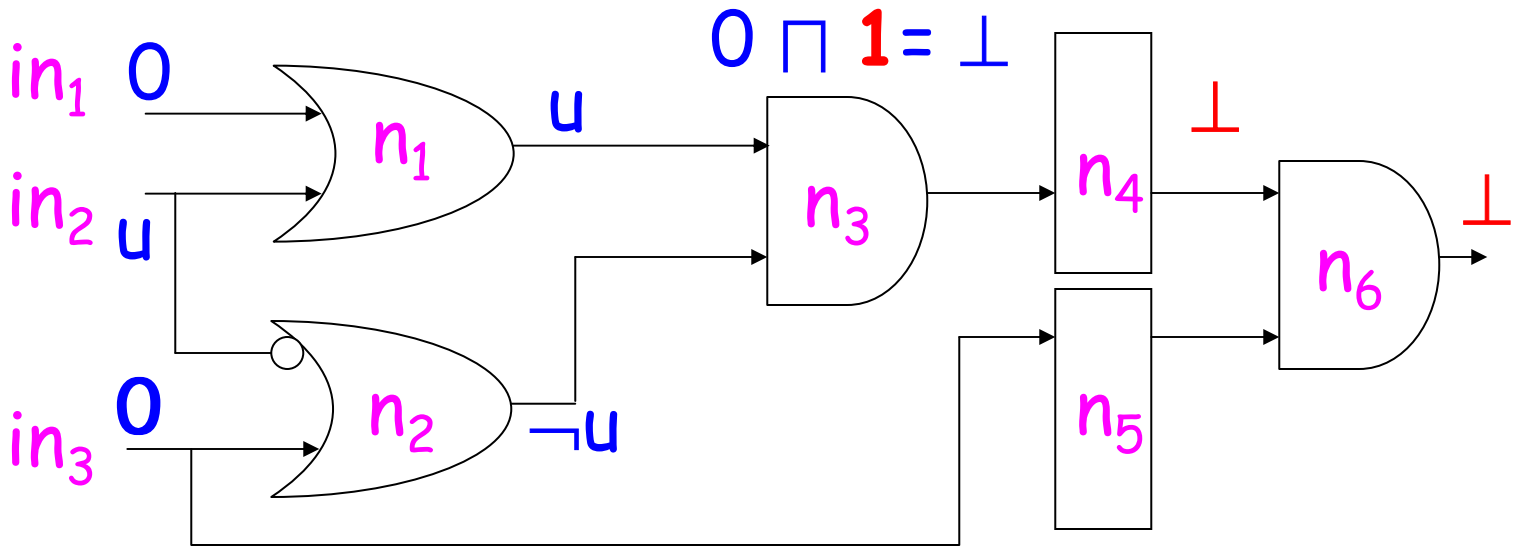
- Represents all assignments to V in which for some node (n,t), C imposes some requirement (0 or 1) but MxA is X
- Automatic/manual refinement is needed

Semantic I_{ref} can be computed in a similar manner



How do we get ⊥ in STE ?

A = in_1 is 0 ∧ in_2 is u ∧ in_3 is 0 ∧ n_3 is 1



Antecedent failure

Antecedent failure is the case in which, for some assignment, MxA contains $\perp$

- Can only occur when the antecedent imposes a constraint on internal node
- Reflects contradiction between
  - Antecedent constraints
  - Circuit execution

- In our work, such assignments are ignored during verification

Agenda

- Model checking
- Symbolic Trajectory Evaluation
- Basic Concepts
- Automatic Refinement for STE
- Vacuity in STE

Vacuity in model checking

Example:

$M \models AG(request \rightarrow F\ granted)$

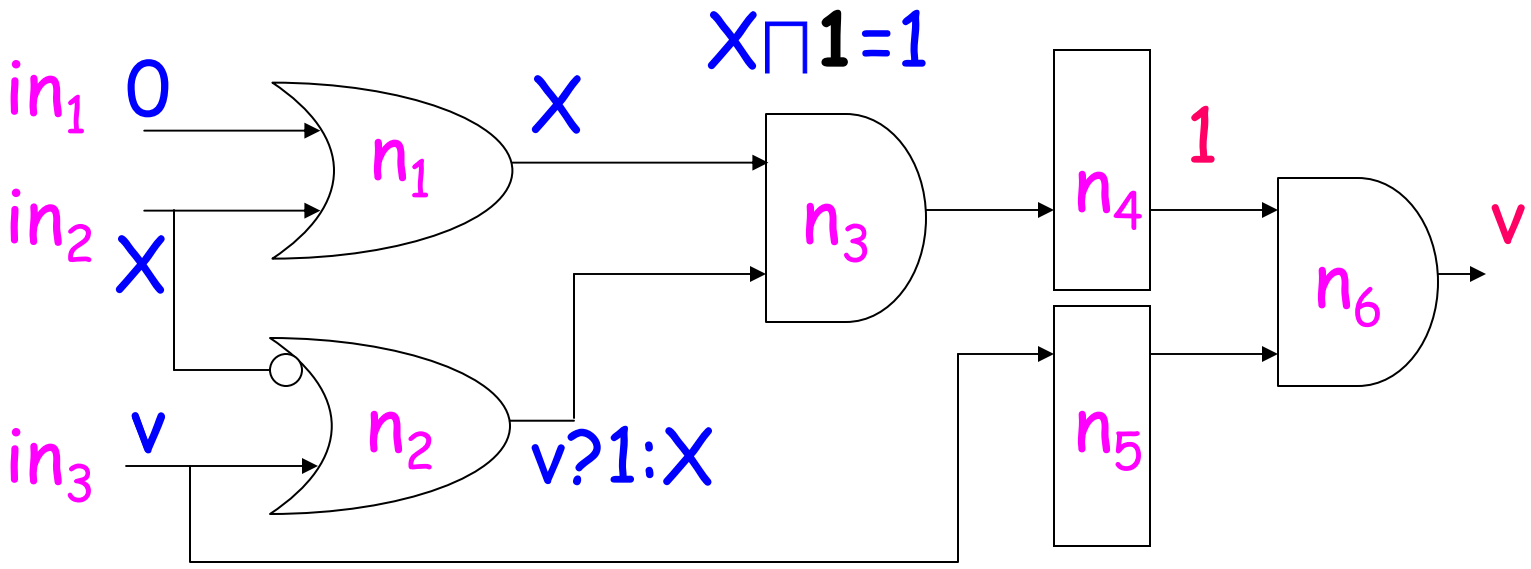
holds vacuously if

- request is always false or
- granted is always true

Vacuous Results

$A = in_1$ is 0 $\wedge$ in_3 is v $\wedge$ n_3 is 1

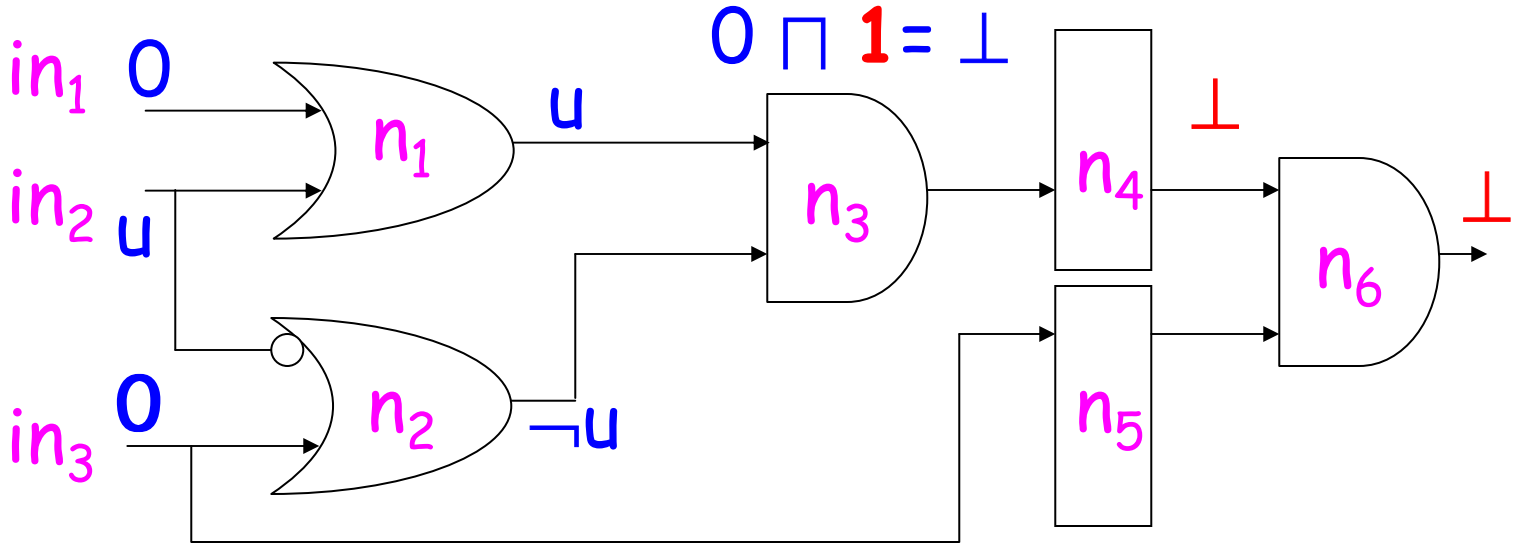
$C = N(n_6 \text{ is } 1)$



Counterexample for v=0. Spurious?

Vacuous Results - Refined

$A = in_1 \text{ is } 0 \wedge in_2 \text{ is } u \wedge in_3 \text{ is } 0 \wedge n_3 \text{ is } 1$



The counterexample is spurious!

The Vacuity Problem

Given an STE assertion $A \Rightarrow C$, an assignment ϕ to V and a circuit M:

- $A \Rightarrow C$ is **vacuous** in M under ϕ if
  - there is no concrete execution of M that satisfies $\phi(A)$

  OR
  - C under ϕ imposes no requirements.

  For example, if $C=(v_1 \rightarrow (n \text{ is } v_2))$ then for assignments in which $v_1=0$, C imposes no requirement

The Vacuity Problem (cont.)

- $A \Rightarrow C$ fails vacuously in M if
  - $[M \times A \models C] = 0$

  AND

  - for all assignments ϕ so that

    $[\phi, M \times A \models C] = 0$, $A \Rightarrow C$ is vacuous in M under ϕ

The Vacuity Problem (cont.)

- $A \Rightarrow C$ passes vacuously in M if
  - $[M \times A \models C] = 1$

  AND
  - for all assignments ϕ so that $[\phi, M \times A \models C] = 1$, $A \Rightarrow C$ is vacuous in M under ϕ

Observation

- Vacuity can only occur when A contains constraints on internal nodes (gates, latches)

- Antecedent failure is an explicit vacuity. Our goal is to reveal **hidden vacuity**.

Detecting (non-)vacuity

Given a circuit M, an STE assertion $A \Rightarrow C$ and an STE result (either fail or pass), our purpose is to find an assignment ϕ to V and an execution of M that satisfies all the constraints in $\phi(A)$

Detecting (non-)vacuity

In Addition:

- In case of **pass**, ϕ should also impose requirements in C

- In case of **fail**, the execution should constitute a counterexample

Detecting (non-)vacuity

We developed two different algorithms for detecting vacuity / non-vacuity:

- An algorithm that uses BMC and runs on the concrete circuit.
- An algorithm that uses STE and automatic refinement.

Detecting (non-)vacuity using BMC

1. Transform *A* into an LTL formula
2. Encode M and *A* as a BMC formula
3. In case of fail STE result, add the counterexample as a constraint to the BMC formula
4. In case of pass STE result, add constraints to enforce at least one requirement in *C*
5. Return "vacuous" if and only if the resulting formula is unsatisfiable

Detecting (non-)vacuity using BMC

Main drawback: no abstraction is used

We would like to detect vacuity while utilizing STE abstraction

Detecting (non-)vacuity using STE

- $A^{in} \Rightarrow A^{out}$ is a new STE assertion, where
  - A^{in} includes all constraints on inputs in A, and
  - A^{out} includes the constraints on internal nodes in A

- Run STE on $A^{in} \Rightarrow A^{out}$. Let Φ denote the set of assignments to V for which $[M \times A^{in} \models A^{out}]=1$

Detecting (non-)vacuity using STE (cont.)

1. In case $[M \times A \models C]=1$: If there is an assignment in Φ that imposes a requirement in C, return "pass non vacuously"

2. In case $[M \times A \models C]=0$: If there exists $\phi \in \Phi$ and ϕ' so that $[\phi', M \times A \models C]=0$ and ($\phi \wedge \phi'$ is satisfiable) , return "fail non vacuously"

Detecting (non-)vacuity using STE (cont.)

3. If there is no ϕ so that $[\phi, M \times A^{in} \models A^{out}]=X$, return "vacuous"

4. Refine $A^{in} \Rightarrow A^{out}$ and return to step 2

Summary

What makes STE successful?

The combination of:

- Symbolic simulation
- Abstraction
- **Local** (dual rail) BDD implementation

Conclusion and future work

Generalized STE (GSTE) extends STE by providing a specification language which is as expressive as ω-regular languages.

Other directions:

- automatic refinement for GSTE (FMCAD'07)
- Vacuity definition and detection for GSTE
- SAT-based STE (ATVA 2007)
- New specification language for GSTE (FMCAD'07)

References

Model Checking

- Model checking
  E. Clarke, O. Grumberg, D. Peled, MIT Press, 1999.

Abstraction-refinement in model checking

- Counterexample-guided abstraction refinement for symbolic model checking
  E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith,
  JACM 50(5): 752-794 (2003)

Vacuity in model checking

- Efficient detection of vacuity in temporal model checking
  I. Beer, S. Ben-David, C. Eisner, Y. Rodeh, Formal Methods in System Design, 18, 2001.

References

STE

- Formal verification by symbolic evaluation of partially-ordered trajectories
  C-J. Seger and R. Bryant, Formal Methods in System Design, 6(2), 1995.

FORTE

- An industrially effective environment for formal hardware verification
  C-J Seger, R. Jones, J. O'Leary, T. Melham, M. Aagaard, C. Barrett, D. Syme, IEEE transactions on Computer-Aided Design of Integrated Circuits and Systems, 24(9), 2005
- FORTE
  http://www.intel.com/software/products/opensource/tools1/verification

References

Refinement in STE

- Automatic refinement and vacuity detection for symbolic trajectory evaluation
  - R. Tzoref and O. Grumberg, CAV'06
  - R. Tzoref, Master thesis, Technion, Haifa, 2006
- SAT-based assistance in abstraction refinement for symbolic trajectory evaluation
  J-W. Roorda and K. Claessen, CAV'06

GSTE

- Introduction to generalized symbolic trajectory evaluation
  J. Yang and C-J. Seger, IEEE transactions on very large scale integrated systems, 11(3), 2003.

THE END