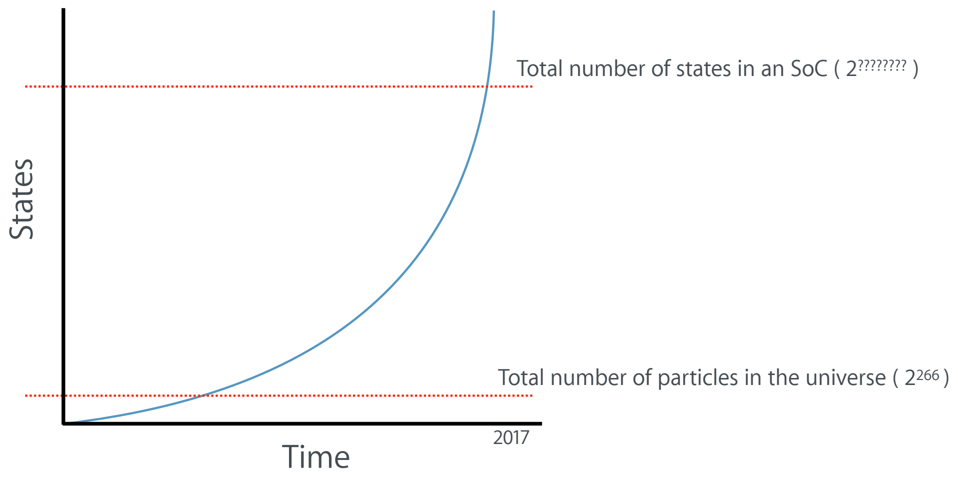
Model Checking

Amit Goel

amitgoel@apple.com

Verification complexity

Exponential



Simulation

Emulation

FPGA's

Formal Verification

Model Checking
Introduction

- Does a given state machine *M* satisfy a property *P*?
  - Check for **all** possible behaviors of the state machine
  - If not, produce a trace showing the violation

Properties

- Safety properties: something **bad** will **never** happen
  - e.g. we should never write to a full buffer

- Liveness properties: something **good** will **eventually** happen
  - e.g. all requests to an arbiter will eventually be granted

Model Checking

Graph Reachability
Symbolic Model Checking
Proof by Induction

Model Checking

Graph Reachability
Symbolic Model Checking
Proof by Induction

State Machines

- A state machine $M = (S, I, T)$
  - S is a set of states
  - $I \subseteq S$ is the set of initial states
  - $T \subseteq S \times S$ is a transition relation

State Machines
Example

$S = \{s_0, s_1, s_2, s_3\}$

$I = \{s_2\}$

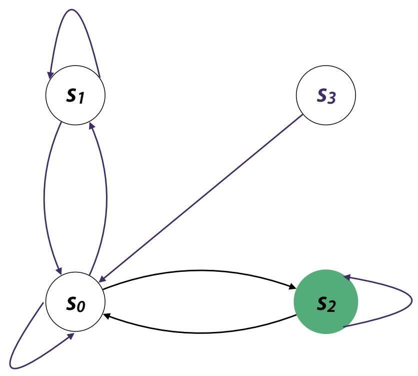
$T = \{(s_0,s_0), (s_0,s_1), (s_0,s_2), (s_1,s_0), (s_1,s_1), (s_2,s_0), (s_2,s_2), (s_3,s_0)\}$

State Machines
Example

$S = \{s_0, s_1, s_2, s_3\}$

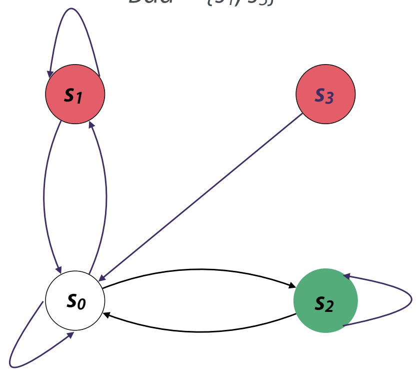
$I = \{s_2\}$

$T = \{(s_0,s_0), (s_0,s_1), (s_0,s_2), (s_1,s_0), (s_1,s_1), (s_2,s_0), (s_2,s_2), (s_3,s_0)\}$

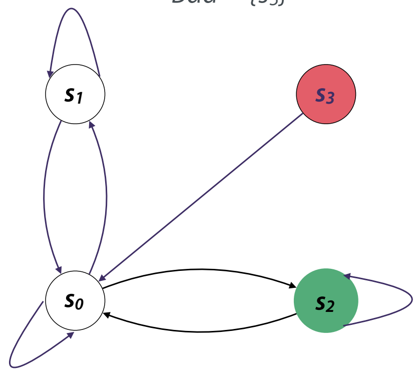


Checking Safety Properties
Examples

$Bad = \{s_1, s_3\}$



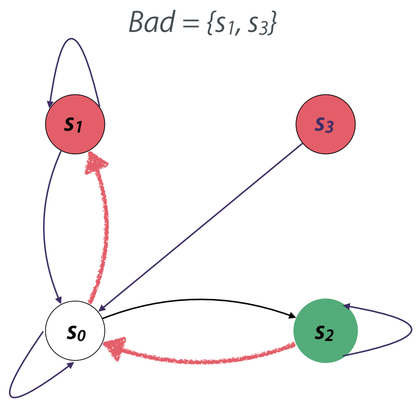
$Bad = \{s_3\}$

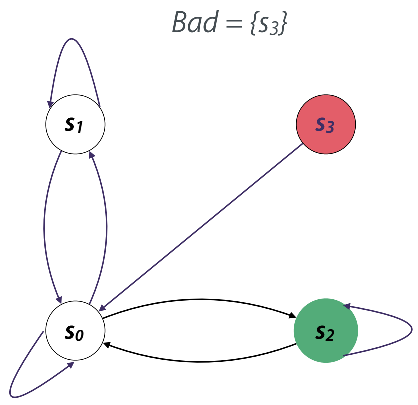


Can we reach a bad state from an initial state?

Checking Safety Properties

Examples



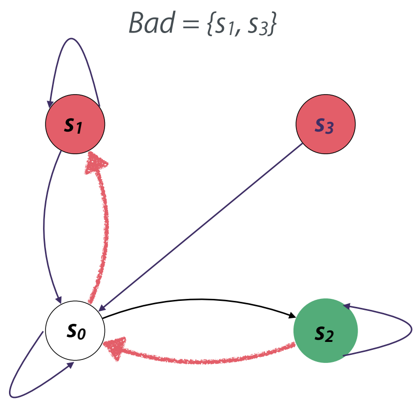


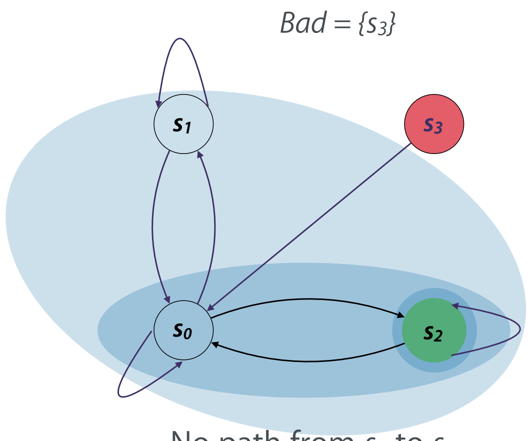
Yes: Path s_2, s_0, s_1

Can we reach a bad state from an initial state?

Checking Safety Properties

Examples





Yes: Path s_2, s_0, s_1

No path from s_2 to s_3

Can we reach a bad state from an initial state?

Explicit-state Model Checking

```
CHECK $(M, Bad)$                                    // $M = (S, I, T), Bad \subseteq S$

  if $(\exists s \in I.\ s \in Bad)$                // Is there a bad initial state?
     return Fail

  $Seen \longleftarrow I$                           // Mark the Initial states as Seen
  while $(\exists (s,s') \in T.\ s \in Seen$ and $s' \notin Seen)$   // Find a reachable unseen state $s'$?
     if $(s' \in Bad)$                              // Is $s'$ a bad state?
        return Fail
     $Seen \longleftarrow Seen \cup \{s'\}$         // Mark $s'$ as seen
  end
  return Pass                                       // Cannot reach a bad state
```

Checking Liveness properties
Example

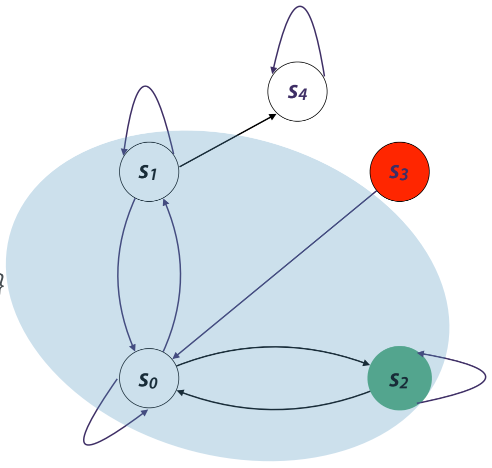
- Infinite loops (Deadlocks and Livelocks) in state machines

$S = \{s_0, s_1, s_2, s_3, s_4\}$

$I = \{s_2\}$

$T = \{(s_0,s_0), (s_0,s_1), (s_0,s_2), (s_1,s_0), (s_1,s_1), (s_1,s_4), (s_2,s_0), (s_2,s_2), (s_3,s_0), (s_4,s_4)\}$

$Good = \{s_0, s_1, s_2\}$



Can you reach a state where you cannot exit from and return to *any* good state?

State-space Explosion
Example

- The number of reachable states in systems is often too large to enumerate
- Consider a system which orders *n* things
  - e.g. Arbitration, Out-of-order processing, …
- Number of orderings is given by *n!*

n	***n!***
4	24
8	40,320
16	20,922,789,888,000

Model Checking

Graph Reachability

Symbolic Model Checking

Proof by Induction

Symbolic Representation of States

- States can be encoded using Boolean variables *V*

State	Encoding with *V= {x,y}*
s_0	*00*
s_1	*01*
s_2	*10*
s_3	*11*

Symbolic Representation of States

- States can be encoded using Boolean variables *V*
- State sets can be represented by Boolean functions over V

Boolean Function	State Set
true	*{00, 01, 10, 11}*
false	*{}*
$x \cdot \neg y$	*{10}*
$x \cdot y$	*{11}*
y	*{01, 11}*

Symbolic Representation of States

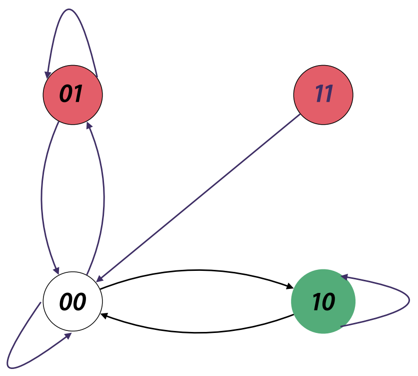
- States can be encoded using Boolean variables *V*
- State sets can be represented by Boolean functions over V
- State relations can be encoded by Boolean functions over two sets of variables, V and V′
  - *V* for current state
  - *V′* for next state

Boolean Function	Relation
$\neg x \cdot \neg y \cdot x' \cdot \neg y'$	$\{00 \rightarrow 10\}$
$\neg x \cdot \neg y \cdot (\neg x' \mid \neg y')$	$\{00 \rightarrow 00, 00 \rightarrow 10, 00 \rightarrow 01\}$

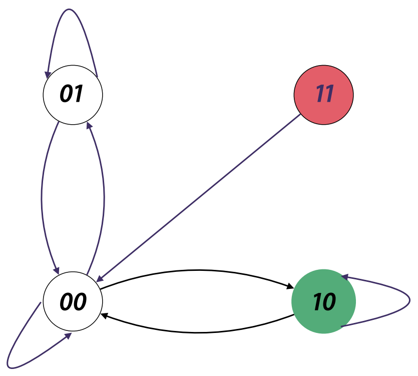
Symbolic Encoding for FSMs

Example

$I = x \cdot \neg y$
$T = \neg x \cdot \neg x' \mid \neg y \cdot \neg y' \mid \neg x' \cdot \neg y'$
$Bad = y$



$Bad = x \cdot y$



Symbolic Representation for Circuits

```
module toy (input clock,  input reset,
                input cx,  input cy,
                output x, output y);

  logic x; logic y;
  always @(posedge clock)
    if (reset) begin
      x <= 1'b1;
      y <= 1'b0;
    end else begin
      x <= !y && cx;
      y <= !x && !cx && cy;
    end
   // Mutex property
  assert property (!(x && y));
endmodule
```

State Variables:
$V = \{x, y\}$

Initial (reset) State:
$I(V) = x \cdot \neg y$

Transition Functions:
$x' = \neg y \cdot cx$
$y' = \neg x \cdot \neg cx \cdot cy$

Property:
$Bad = x \cdot y$

Synthesized Circuit

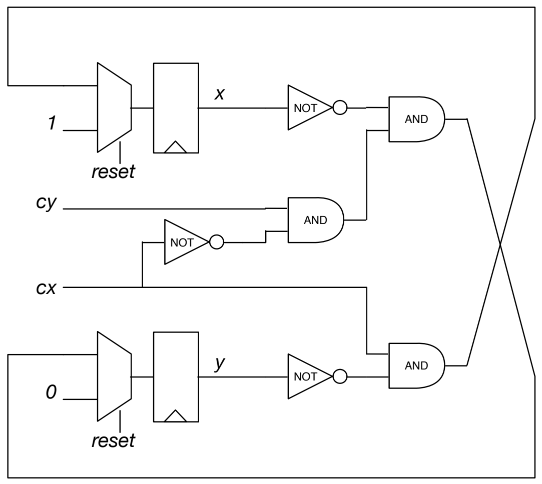
```
module toy (input clock,  input reset,
                input cx,  input cy,
                output x, output y);

  logic x; logic y;
  always @(posedge clock)
    if (reset) begin
      x <= 1'b1;
      y <= 1'b0;
    end else begin
      x <= !y && cx;
      y <= !x && !cx && cy;
    end
   // Mutex property
  assert property (!(x && y));
endmodule
```

State Variables:
$V = \{x, y\}$

Initial (reset) State:
$I(V) = x \cdot \neg y$

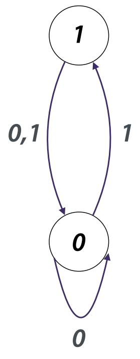
Transition Functions:
$x' = \neg y \cdot cx$
$y' = \neg x \cdot \neg cx \cdot cy$

Property:
$Bad = x \cdot y$



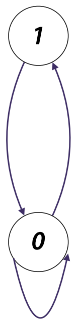
Transition Function and Transition Relation
Example

- Transition function:
  $x' = \neg x \cdot cx$
- Transition relation (with input variables):
  $\tilde{T}(x, cx, x') = (x' \longleftrightarrow \neg x \cdot cx)$



Transition Function and Transition Relation
Example

- Transition function:
  $x' = \neg x \cdot cx$
- Transition relation (with input variables):
  $\tilde{T}(x, cx, x') = (x' \longleftrightarrow \neg x \cdot cx)$
- Transition Relation (without input variables):
  $T(x, x') = \exists cx.(x' \longleftrightarrow \neg x \cdot cx)$
  $= (\neg x \mid \neg x')$



Transition Functions and Transition Relation

Example

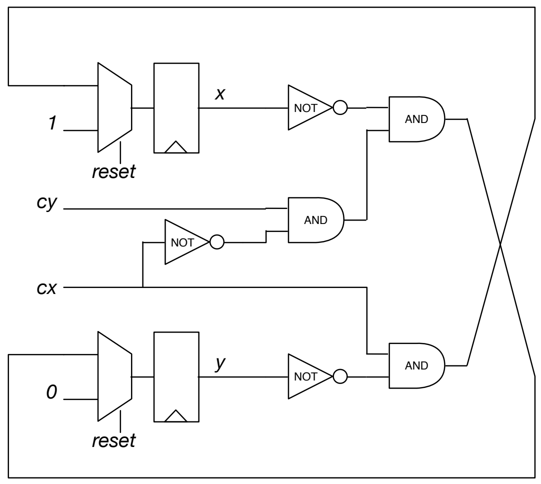
- Transition function:
  $x' = \neg y \cdot cx$
  $y' = \neg x \cdot \neg cx \cdot cy$
- Transition relation (with input variables):
  $\tilde{T}(V, cx, cy, V') = (x' \leftrightarrow \neg y \cdot cx) \wedge (y' \leftrightarrow \neg x \cdot \neg cx \cdot cy)$
- Transition Relation:
  $T(V, V') = \exists cx, cy.\, \tilde{T}(V, cx, cy, V')$
  $\quad = (\neg x \cdot \neg x' \mid \neg y \cdot \neg y' \mid \neg x' \cdot \neg y')$

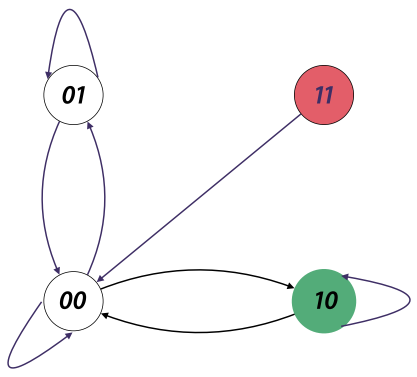
Symbolic Representation

Verilog, Circuit and FSM

$V = \{x,y\}$
$I(V) = x \cdot \neg y$
$T(V,V') = \neg x \cdot \neg x' \mid \neg y \cdot \neg y' \mid \neg x' \cdot \neg y'$
$Bad(V) = x \cdot y$

```
module toy (input clock,  input reset,
               input cx,  input cy,
               output x, output y);

  logic x; logic y;
  always @(posedge clock)
    if (reset) begin
      x <= 1'b1;
      y <= 1'b0;
    end else begin
      x <= !y && cx;
      y <= !x && !cx && cy;
    end

   // Mutex property
  assert property (!(x && y));
endmodule
```





Symbolic Set Operations

Symbolic Expression	Boolean Formula	Corresponding Set	Set Expression
	true	*{00, 01, 10, 11}*	S
A	x	*{10, 11}*	S_A
B	y	*{01, 11}*	S_B
$A \vee B$	$x \mid y$	*{01, 10, 11}*	$S_A \cup S_B$
$A \wedge B$	$x \cdot y$	*{11}*	$S_A \cap S_B$
$\neg A$	$\neg x$	*{00, 01}*	$S \setminus S_A$

Image Computation

- Given states $R(V)$ and transition relation $T(V,V')$
- Let $F(V') = \exists V.\ R(V) \wedge T(V,V')$,
- Let $F(V)$ be obtained by renaming V' to V in $F(V')$
- Then $F(V)$ is the set of all states reachable in one step from states in R

Image Computation

- Given states *R(V)* and transition relation *T(V,V′)*
- Let *F(V′)*= ∃*V. R(V)* ∧ *T(V,V′),*
- Then *F(V)* is the set of all states reachable in one step from states in *R*

$R(V) = I(V) = x \cdot \neg y$

$T(V,V') = \neg x \cdot \neg x' \mid \neg y \cdot \neg y' \mid \neg x' \cdot \neg y'$

$F(V') = \neg y'$

$F(V) \;= \neg y$

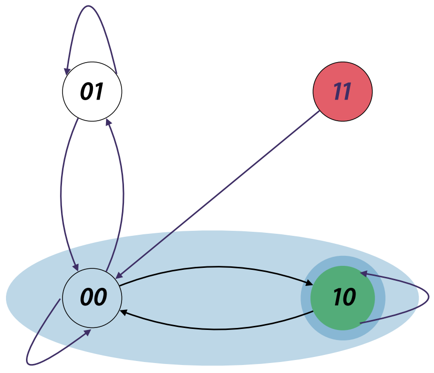


Image Computation

- Given states *R(V)* and transition relation *T(V,V′)*
- Let *F(V′)*= ∃*V. R(V)* ∧ *T(V,V′),*
- Then *F(V)* is the set of all states reachable in one step from states in *R*

$R(V) = \neg y$

$T(V,V') = \neg x \cdot \neg x' \mid \neg y \cdot \neg y' \mid \neg x' \cdot \neg y'$

$F(V) = \neg x \mid \neg y$

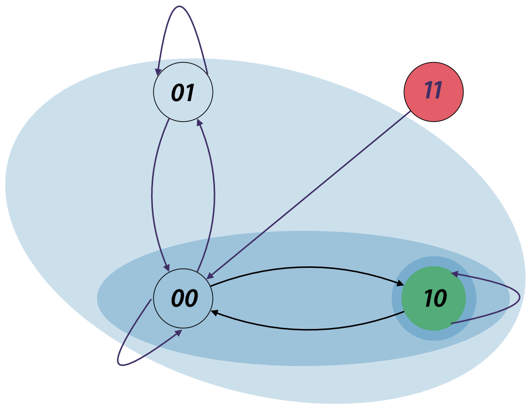


Image Computation

- Given states *R(V)* and transition relation *T(V,V')*
- Let $F(V') = \exists V.\ R(V) \wedge T(V,V')$,
- Then *F(V)* is the set of all states reachable in one step from states in *R*

- We write *Img(R,T)* for the above image computation

Symbolic Model Checking

Forward Reachability Algorithm

```
CHECK (M, Bad)                          // M = (S, I, T), Bad ⊆ S

  Prev ⟵ false                          // No states have been seen as yet
  Seen ⟵ I                              // Mark Initial States as Seen
  while (Seen ≠ Prev)                   // Have we seen any new states?
     if (Seen ∧ Bad ≠ false)            // Have we seen a bad state?
        return Fail
     Prev ⟵ Seen                        // Update previously seen states
     Seen ⟵ Prev ∨ Img(Prev,T)          // Mark states in the image of Prev as seen
  end
  return Pass                           // No Bad state reachable
```

Satisfiability Solvers

- Given a Boolean formula *Q*,
- Is there a satisfying assignment to the variables in Q?

Satisfiability Solvers
Example

- SAT$((x \mid y) \cdot (x \mid z) \cdot (\neg x \mid \neg z) \cdot (y \mid z))$?

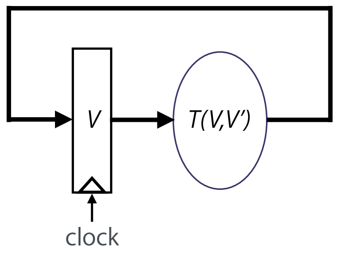
  Yes.
  Satisfying assignment: $x=1, y=1, z=0$

  Satisfying assignment: $x=0, y=1, z=1$
- SAT$((x \mid \neg y) \cdot (\neg x \mid y) \cdot (x \mid z) \cdot (\neg x \mid \neg z) \cdot (y \mid \neg z) \cdot (\neg y \mid z))$?

  No.

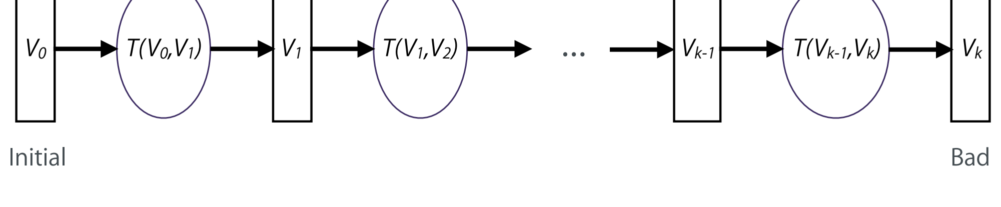
Bounded Model Checking

- Bounded Model Checking: Can we reach a bad state in *k* cycles?



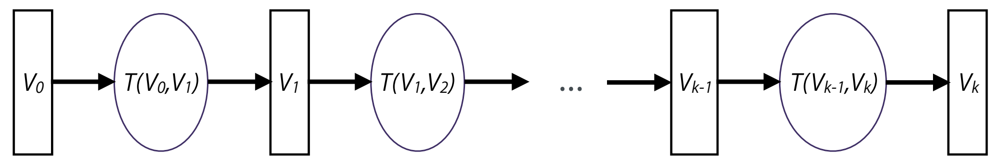
Bounded Model Checking

- Bounded Model Checking: Can we reach a bad state in *k* cycles?
- Unroll the circuit *k* times



Bounded Model Checking

- Bounded Model Checking: Can we reach a bad state in *k* cycles?
- Unroll the circuit *k* times



Initial Bad

- $Bad_k = I(V_0) \wedge T(V_0,V_1) \wedge \ldots \wedge T(V_{k-1}, V_k) \wedge Bad(V_k)$
- Unsatisfiable? No failure in *k* cycles
- Satisfiable? Satisfying assignment is a *k*-length counterexample

Bounded Model Checking
Iterative BMC

```
CHECK (M, Bad)                  // M = (S, I, T), Bad ⊆ S

  k ⟵ 0
  while (true)
     if SAT(Bad_k)              // k-BMC
       return Fail
     k ⟵ k+1                    // Increment k
  end
```

When does the loop terminate for an N-bit state machine?

Model Checking

Graph Reachability
Symbolic Model Checking
Proof by Induction

Natural Induction

- To Prove:
  $1 + 2 + \ldots + n = (n * (n+1)) / 2$
- Base Step:
  Show that the equation holds for $n = 1$
  $1 = (1 * 2) / 2 = 1$
- Induction:
  Assume equation holds for $n = i$,
  then show that it holds for $n = (i+1)$

$$\begin{aligned}
1 + 2 + \ldots + i &= (i*(i+1))/2 && \text{// Assumption} \\
1 + 2 + \ldots + i + (i+1) &= (i*(i+1))/2 + (i+1) \\
&= (i*(i+1) + 2*(i+1))/2 \\
&= ((i+1) * (i+2))/2
\end{aligned}$$

Induction for FSM Properties

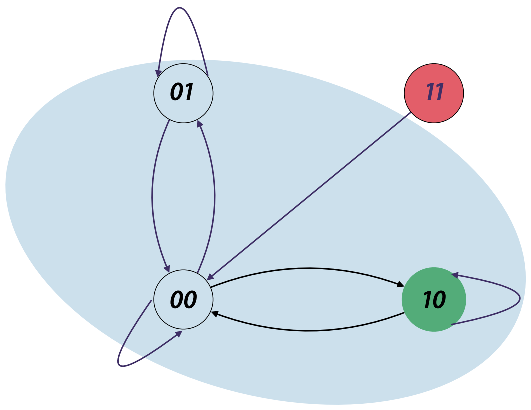
- Given *Bad* states, all the other states are *Good*
  $Good(V) = \neg Bad(V)$
- To show that an FSM never reaches a *Bad* state
- Prove that the FSM always stays in a *Good* state
  i.e. *Good* is an invariant for the system

Induction for FSM Properties

- Given *Bad* states, all the other states are *Good*
  $Good(V) = \neg Bad(V)$
- To show that an FSM never reaches a *Bad* state
- Prove that the FSM always stays in a *Good* state

- Show that the following are valid:
  $I(V) \Rightarrow Good(V)$ (Base step)

  $Good(V) \wedge T(V,V') \Rightarrow Good(V')$ (Induction)



$Bad = x \cdot y$
$Good = \neg x \mid \neg y$

Induction for FSM Properties

Not all valid properties are inductive

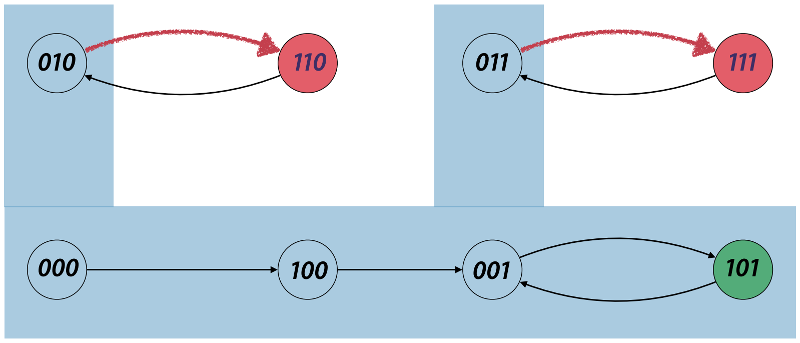
$V = \{x,y,z\}$

$I = x \cdot \neg y \cdot z$

$T = (x' \leftrightarrow \neg x) \cdot (y' \leftrightarrow y) \cdot (z \leftrightarrow (x \mid z))$

$Bad = x \cdot y$

$Good = \neg(x \cdot y)$ Not inductive



Inductive Invariants

- An inductive invariant for a state machine is any property $\Phi(V)$ such that:

  $I(V) \Rightarrow \Phi(V)$

  $\Phi(V) \wedge T(V,V') \Rightarrow \Phi(V')$

- To prove that FSM always stays in *Good:*

  Find an inductive invariant $\Phi(V)$

  Show that $\Phi(V) \Rightarrow Good(V)$

Model Checking and Induction

- The set *Reach* of all reachable states is, by definition, inductive
  - The initial states are in *Reach*
  - From *Reach*, you can only reach states in *Reach*

Model Checking and Induction

- The set *Reach* of all reachable states is, by definition, inductive
  - The initial states are in *Reach*
  - From *Reach*, you can only reach states in *Reach*
- *Reach* is the strongest invariant for the state machine
  - Given any other invariant $\Phi(V)$,
    $Reach(V) \Rightarrow \Phi(V)$

Model Checking and Induction

- The set *Reach* of all reachable states is, by definition, inductive
  - The initial states are in *Reach*
  - From *Reach*, you can only reach states in *Reach*
- *Reach* is the strongest invariant for the state machine
  - Given any invariant $\Phi(V)$,
    $Reach(V) \Rightarrow \Phi(V)$
- Alternate algorithms attempt to find weaker invariants:
  - Interpolation
  - Property Directed Reachability (PDR)

Model Checking and Induction

The set of reachable states is inductive

$V = \{x,y,z\}$

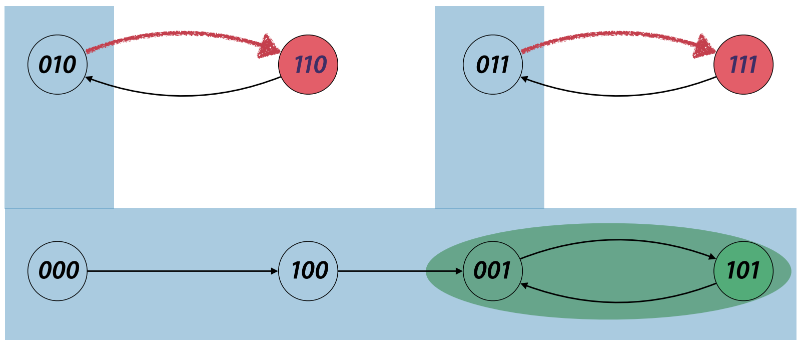
$I = x \cdot \neg y \cdot z$

$T = (x' \leftrightarrow \neg x) \cdot (y' \leftrightarrow y) \cdot (z \leftrightarrow (x \mid z))$

$Bad = x \cdot y$

$Good = \neg(x \cdot y)$ Not inductive

$Reach = \neg y \cdot z$ Inductive



Model Checking and Induction

Alternate Inductive Invariant

$V = \{x,y,z\}$

$I = x \cdot \neg y \cdot z$

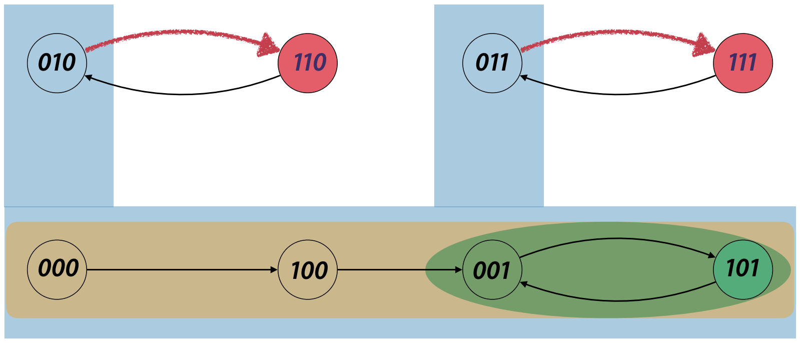
$T = (x' \leftrightarrow \neg x) \cdot (y' \leftrightarrow y) \cdot (z \leftrightarrow (x \mid z))$

$Bad = x \cdot y$

$Good = \neg(x \cdot y)$ Not inductive

$Reach = \neg y \cdot z$ Inductive

$\Phi = \neg y$ Inductive



Model Checking and Induction

Invariant Strength

$V = \{x,y,z\}$

$I = x \cdot \neg y \cdot z$

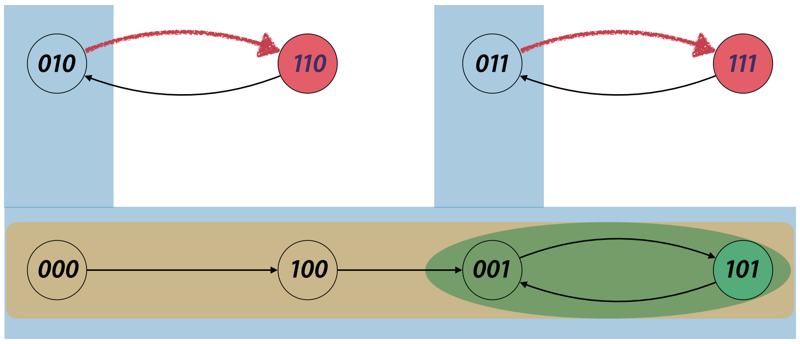
$T = (x' \leftrightarrow \neg x) \cdot (y' \leftrightarrow y) \cdot (z \leftrightarrow (x \mid z))$

$Bad = x \cdot y$

$Good = \neg(x \cdot y)$ Not inductive

$Reach = \neg y \cdot z$ Inductive

$\Phi = \neg y$ Inductive



Invariant Strength: $Reach \Rightarrow \Phi \Rightarrow Good$

Summary

Summary

- Model Checking is an effective static analysis method for verification
  - Enables validation of complex System-on-a-Chip designs (SoC's), including CPU's and GPU's
  - Results in robust design micro-architecture specifications and implementations
  - Unit-level formal analysis <u>must</u> seamlessly dovetail into product design methodologies
- Core Ideas
  - Graph Reachability
  - Symbolic representation
  - Induction
- Plenty of scope for creative work and careers in hardware verification
  - Tools, flows, and methodologies to tackle hard verification "puzzles" in industry

References

References

- Model Checking
  - E. A. Emerson and E. M. Clarke, "Characterizing Correctness Properties of Parallel Programs as Fixpoints", *Proceedings of the Seventh International Colloquium on Automata, Languages, and Programming*, *LNCS, Vol. 85*, 1981.
  - E.M. Clarke, E. A. Emerson and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems, Vol. 8, No. 2*, April 1986.
  - J-P. Queille and J. Sifakis, "Specification and Verification of Concurrent Systems," *CESAR International Symposium on Programming, LNCS, Vol. 137*, 1982
- BDDs and Symbolic Model Checking
  - R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Transactions on Computers, Vol. C - 35, No. 8, August, 1986
  - O. Coudert, C. Berthet and J.C. Madre, "Verification of Synchronous Sequential Machines Based on Symbolic Execution," *International Workshop on Automatic Verification Methods for Finite State Systems*, 1989
  - J.R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang, "Symbolic model checking: 10^{20} states and beyond," *5th Ann. Symposium on Logic in Computer Science*, June 1990

References

- Satisfiability and Bounded Model Checking
  - Niklas Eén and Niklas Sörensson, "An Extensible SAT-solver," *6th International Conference on Theory and Applications of Satisfiability Testing,* 2003
  - A. Biere, A. Cimatti, E. M. Clarke and Y. Zhu, "Symbolic Model Checking without BDDs," *5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, 1999
- Induction and other SAT-based methods
  - M. Sheeran, S. Singh and G. Stalmark, "Checking Safety Properties using Induction and a SAT-solver," *Formal Methods in Computer-Aided Design*, 2000
  - K.L. McMillan, "Craig Interpolation and Reachability Analysis," *10th International Symposium on Static Analysis*, 2003
  - A.R. Bradley, "SAT-based Model Checking without Unrolling," *12th International Conference on Verification, Model Checking, and Abstract Interpretation,* 2011

References

- Textbooks
  - E.M. Clarke, O. Grumberg and D.A. Peled, "Model Checking," MIT Press, 1999
  - T. Kropf, "Introduction to Formal Hardware Verification", *Springer-Verlag,* 1999