Bounded Model Checking

Bounded Model Checking

- Broad methodology
  - We construct a Boolean formula that is satisfiable iff the underlying state transition system can realize a finite sequence of state transitions that satisfy the temporal property we are trying to validate
  - We use powerful SAT solvers to determine the satisfiability of the Boolean formula
  - The bound may be increased incrementally until we reach the diameter of the state transition graph
    - Find the shortest path between each pair of vertices. The greatest length of any of these paths is the diameter of the graph.

Requirements

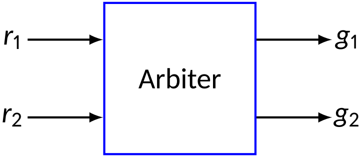
- Specification in temporal logic, f
- System as a finite state machine (kripke structure), M
- A bound on path length k
  - In bounded model checking, only path of bounded length k or less are considered

Requirements

- Specification in temporal logic, *f*
- System as a finite state machine (kripke structure), *M*
- A bound on path length *k*
  - In bounded model checking, only path of bounded length *k* or less are considered

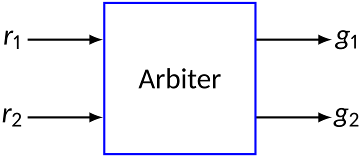
- Translation to SAT
  - We unfold the property into Boolean clauses over different time steps
  - We unfold the state machine into Boolean clauses over the same number of time steps
  - We check whether the clauses are together satisfiable

Unrolling



If r_1 is true in a cycle then g_1 has to be true for the next two cycles: $r_1 \rightarrow \boldsymbol{X} g_1 \wedge \boldsymbol{XX} g_1$

Unrolling



If r_1 is true in a cycle then g_1 has to be true for the next two cycles: $r_1 \rightarrow \mathbf{X} g_1 \wedge \mathbf{XX} g_1$

r_1^0

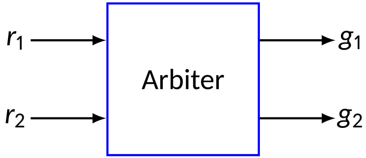
r_2^0

g_1^0

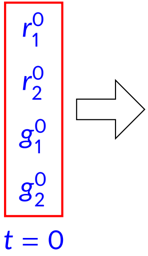
g_2^0

$t = 0$

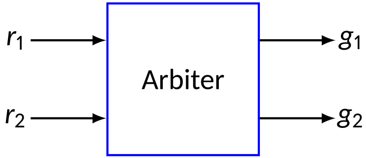
Unrolling



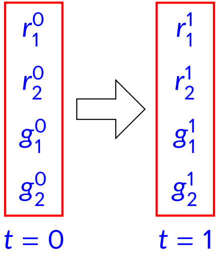
If r_1 is true in a cycle then g_1 has to be true for the next two cycles: $r_1 \longrightarrow \mathbf{X} g_1 \wedge \mathbf{XX} g_1$



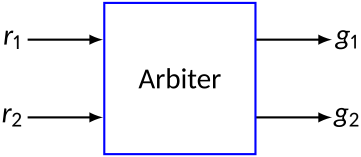
Unrolling



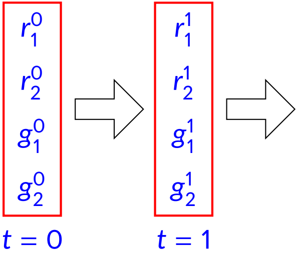
If r_1 is true in a cycle then g_1 has to be true for the next two cycles: $r_1 \longrightarrow \mathbf{X} g_1 \wedge \mathbf{XX} g_1$



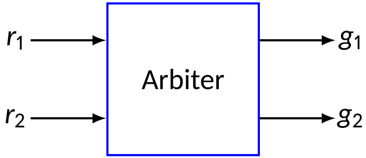
Unrolling



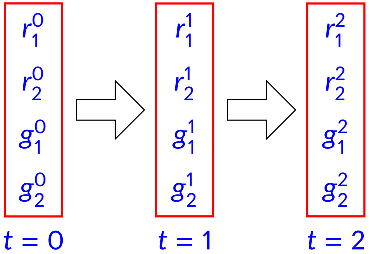
If r_1 is true in a cycle then g_1 has to be true for the next two cycles: $r_1 \longrightarrow \boldsymbol{X} g_1 \wedge \boldsymbol{XX} g_1$



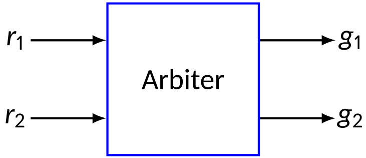
Unrolling



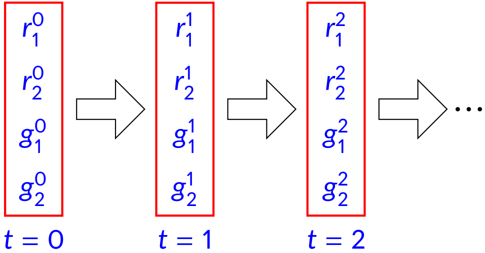
If r_1 is true in a cycle then g_1 has to be true for the next two cycles: $r_1 \rightarrow \mathbf{X} g_1 \wedge \mathbf{XX} g_1$



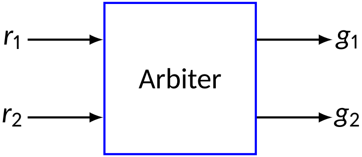
Unrolling



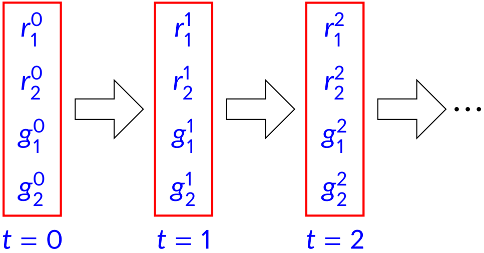
If r_1 is true in a cycle then g_1 has to be true for the next two cycles: $r_1 \rightarrow \mathbf{X} g_1 \wedge \mathbf{XX} g_1$



Unrolling



If r_1 is true in a cycle then g_1 has to be true for the next two cycles: $r_1 \rightarrow Xg_1 \wedge XXg_1$



$$r_1^0 \rightarrow g_1^1 \wedge g_1^2$$

$$\forall t \ \left[r_1^t \rightarrow g_1^{t+1} \wedge g_1^{t+2} \right]$$

Recap: CTL

- ***A*** **– for every path**
- ***E*** **– there exist a path**

Recap: CTL

- ***A*** – for every path
- ***E*** – there exist a path

- ***X****p* – *p* holds in the next time step
- ***F****p* – *p* holds at some time in the future
- ***G****p* – *p* holds at every time in the future
- *p***U***q* – *p* holds until q holds

Recap: CTL

- A – for every path
- E – there exist a path

- Xp – p holds in the next time step
- Fp – p holds at some time in the future
- Gp – p holds at every time in the future
- pUq – p holds until q holds

- $EF(Started \wedge \neg Ready)$ – it is possible to get to a state where *Started* holds but *Ready* does not
- $AG(Req \implies AFgr)$ – if a *Req* comes then it will eventually be *granted*

Symbolic model checking

- **We allocate Boolean variables for present state of latches, next state of latches and primary input variables**

Symbolic model checking

- **We allocate Boolean variables for present state of latches, next state of latches and primary input variables**
- **We compute Boolean expression f_j for the input function for each latch, j, in the circuit. Let $\boldsymbol{x}$, $\boldsymbol{i}$ are the present state vector and the primary input vector respectively**

Symbolic model checking

- **We allocate Boolean variables for present state of latches, next state of latches and primary input variables**
- **We compute Boolean expression f_j for the input function for each latch, j, in the circuit. Let $\boldsymbol{x}$, $\boldsymbol{i}$ are the present state vector and the primary input vector respectively**
- **For each latch, Boolean expression for transition relation is formed, $x'_j \leftrightarrow f_j(\boldsymbol{x}, \boldsymbol{i})$ where x'_j is the next state variable for the jth latch. Symbol $\leftrightarrow$ means *if and only if* (i.e., XNOR)**

Symbolic model checking

- We allocate Boolean variables for present state of latches, next state of latches and primary input variables
- We compute Boolean expression f_j for the input function for each latch, j, in the circuit. Let $\mathbf{x}, \mathbf{i}$ are the present state vector and the primary input vector respectively
- For each latch, Boolean expression for transition relation is formed, $x_j' \leftrightarrow f_j(\mathbf{x}, \mathbf{i})$ where x_j' is the next state variable for the jth latch. Symbol $\leftrightarrow$ means *if and only if* (i.e., XNOR)
- Transition relation $T(s, s')$, where s, s' denote present and next state, can be expressed as

$$T(s, s') = \bigwedge_{j=1}^{n} x_j' \leftrightarrow f_j(\mathbf{x}, \mathbf{i})$$

Traversal

- **Once the Boolean expression for transition relation is computed, it can be used for traversal the underlying transition system**
- **Traversals are done by computing** *images* **and** *preimages* **of set of states. These denote** *successor* **or** *predecessor* **states respectively.**

Traversal

- **Once the Boolean expression for transition relation is computed, it can be used for traversal the underlying transition system**
- **Traversals are done by computing** *images* **and** *preimages* **of set of states. These denote** *successor* **or** *predecessor* **states respectively.**

- **Let $P(s)$ be the set of states then**

$$\begin{aligned} Image_P &= \exists s\ [T(s,s') \wedge P(s)] \\ Preimage_P &= \exists s'\ [T(s,s') \wedge P(s')] \end{aligned}$$

- **Existential abstraction**

$$\exists x_i\ [f(x_0,\ldots,x_i,\ldots,x_n) = f(x_0,\ldots,0,\ldots,x_n) \vee f(x_0,\ldots,1,\ldots,x_n)$$

Algorithm for checking *EFp*

1. Let Q be Boolean expression that represents the set of states in which p is true. So is R
2. Pre = preimage of Q
3. $Union\_Reached = Pre \vee R$
4. If $Union\_Reached = R$, go to 8
5. $Q = Pre \wedge \neg R$
6. $R = Union\_Reached$
7. Go to 2
8. If $(R \wedge I)$ is satisfiable (initial state intersection), *EFp* holds. Otherwise, it does not hold

Creation of propositional formulas

- **Need to construct propositional formula for** $[\![M, f]\!]_k$

Creation of propositional formulas

- **Need to construct propositional formula for** $[\![M, f]\!]_k$
- **Unrolled transition relation**

Creation of propositional formulas

- **Need to construct propositional formula for** $[\![M, f]\!]_k$
- **Unrolled transition relation** $[\![M]\!]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$

Creation of propositional formulas

- **Need to construct propositional formula for** $[\![M, f]\!]_k$
- **Unrolled transition relation** $[\![M]\!]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$
- **Need to find** $[\![f]\!]_k$ **which is true if the formula** f **is valid along a path of length** k

Creation of propositional formulas

- **Need to construct propositional formula for** $[\![M, f]\!]_k$
- **Unrolled transition relation** $[\![M]\!]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$
- **Need to find** $[\![f]\!]_k$ **which is true if the formula** f **is valid along a path of length** k
- **Finally we need to form the conjunction of** $[\![M]\!]_k$ **and** $[\![f]\!]_k$

Creation of propositional formulas

- Need to construct propositional formula for $[\![M, f]\!]_k$
- Unrolled transition relation $[\![M]\!]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$
- Need to find $[\![f]\!]_k$ which is true if the formula f is valid along a path of length k
- Finally we need to form the conjunction of $[\![M]\!]_k$ and $[\![f]\!]_k$

- Consider CTL formula EFp

Creation of propositional formulas

- Need to construct propositional formula for $[\![M, f]\!]_k$
- Unrolled transition relation $[\![M]\!]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$
- Need to find $[\![f]\!]_k$ which is true if the formula f is valid along a path of length k
- Finally we need to form the conjunction of $[\![M]\!]_k$ and $[\![f]\!]_k$

- Consider CTL formula ***EF****p*
- We wish to check whether ***EF****p* can be verified in two time steps i.e., $k = 2$

Creation of propositional formulas

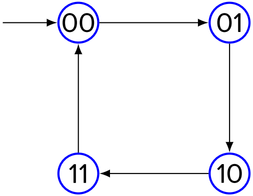
- **Need to construct propositional formula for** $[\![M, f]\!]_k$
- **Unrolled transition relation** $[\![M]\!]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$
- **Need to find** $[\![f]\!]_k$ **which is true if the formula** f **is valid along a path of length** k
- **Finally we need to form the conjunction of** $[\![M]\!]_k$ **and** $[\![f]\!]_k$

- **Consider CTL formula** $\boldsymbol{EF}p$
- **We wish to check whether** $\boldsymbol{EF}p$ **can be verified in two time steps i.e.,** $k = 2$

  $[\![M, f]\!]_2 := I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge (p(s_0) \vee p(s_1) \vee p(s_2))$

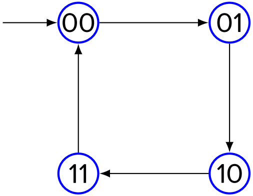
Safety property - *EFp*

- **Let us assume a 2-bit counter with least significant bit represented by Boolean variable *a* and most significant by *b***



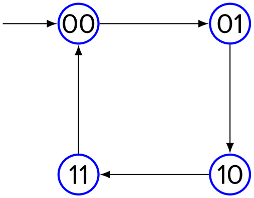
Safety property - *EFp*

- Let us assume a 2-bit counter with least significant bit represented by Boolean variable a and most significant by b
- Transition relation of the counter is $(a' \leftrightarrow \neg a) \wedge (b' \leftrightarrow (a \oplus b))$



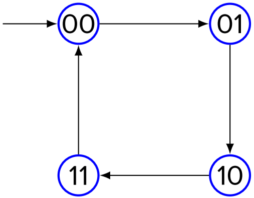
Safety property - *EFp*

- **Let us assume a 2-bit counter with least significant bit represented by Boolean variable a and most significant by b**
- **Transition relation of the counter is** $(a' \leftrightarrow \neg a) \wedge (b' \leftrightarrow (a \oplus b))$
- **Let us assume initially both bits are at 0 and we wish to check if** (a, b) **can transition to** $(1, 1)$
  - $\mathbf{EF}p = \mathbf{EF}(a \wedge b)$



Safety property - *EFp*

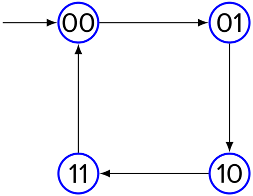
- Let us assume a 2-bit counter with least significant bit represented by Boolean variable a and most significant by b
- Transition relation of the counter is $(a' \leftrightarrow \neg a) \wedge (b' \leftrightarrow (a \oplus b))$
- Let us assume initially both bits are at 0 and we wish to check if (a, b) can transition to $(1, 1)$
  - $\boldsymbol{EF}p = \boldsymbol{EF}(a \wedge b)$
- Unfolding the transition relation and applying reachability check, we get



Safety property - *EFp*

- **Let us assume a 2-bit counter with least significant bit represented by Boolean variable a and most significant by b**
- **Transition relation of the counter is** $(a' \leftrightarrow \neg a) \wedge (b' \leftrightarrow (a \oplus b))$
- **Let us assume initially both bits are at 0 and we wish to check if** (a, b) **can transition to** $(1, 1)$
  - $\boldsymbol{EF}p = \boldsymbol{EF}(a \wedge b)$
- **Unfolding the transition relation and applying reachability check, we get**

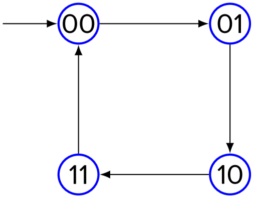
$$I(s_0): \quad (\quad \neg a_0 \wedge \neg b_0 \quad) \wedge$$



Safety property - *EFp*

- Let us assume a 2-bit counter with least significant bit represented by Boolean variable a and most significant by b
- Transition relation of the counter is $(a' \leftrightarrow \neg a) \wedge (b' \leftrightarrow (a \oplus b))$
- Let us assume initially both bits are at 0 and we wish to check if (a, b) can transition to $(1, 1)$
  - $EFp = EF(a \wedge b)$
- Unfolding the transition relation and applying reachability check, we get

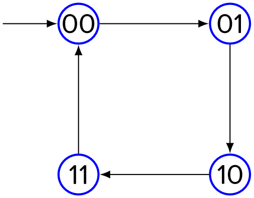
$$\begin{aligned}
I(s_0) &: \quad (\quad \neg a_0 \wedge \neg b_0 \quad) \wedge \\
T(s_0, s_1) &: \quad (\quad ((a_1 \leftrightarrow \neg a_0) \wedge (b_1 \leftrightarrow (a_0 \oplus b_0)) \quad) \wedge \\
T(s_1, s_2) &: \quad (\quad ((a_2 \leftrightarrow \neg a_1) \wedge (b_2 \leftrightarrow (a_1 \oplus b_1)) \quad) \wedge
\end{aligned}$$



Safety property - *EFp*

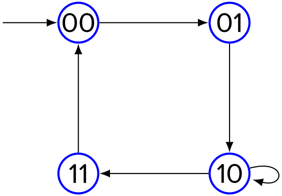
- **Let us assume a 2-bit counter with least significant bit represented by Boolean variable a and most significant by b**
- **Transition relation of the counter is** $(a' \leftrightarrow \neg a) \wedge (b' \leftrightarrow (a \oplus b))$
- **Let us assume initially both bits are at 0 and we wish to check if** (a, b) **can transition to** $(1, 1)$
  - $\mathbf{EF}p = \mathbf{EF}(a \wedge b)$
- **Unfolding the transition relation and applying reachability check, we get**

$$
\begin{array}{rlcl}
I(s_0): & (& \neg a_0 \wedge \neg b_0 &) \wedge \\
T(s_0, s_1): & (& ((a_1 \leftrightarrow \neg a_0) \wedge (b_1 \leftrightarrow (a_0 \oplus b_0)) &) \wedge \\
T(s_1, s_2): & (& ((a_2 \leftrightarrow \neg a_1) \wedge (b_2 \leftrightarrow (a_1 \oplus b_1)) &) \wedge \\
p(s_0): & (& a_0 \wedge b_0 \ \vee & \\
p(s_1): & & a_1 \wedge b_1 \ \vee & \\
p(s_2): & & a_2 \wedge b_2 &)
\end{array}
$$



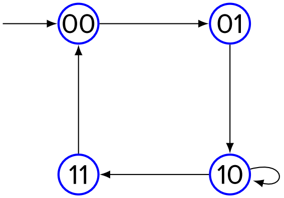
Liveness property - *AFp*

- Consider a 2-bit counter with least significant bit represented by Boolean variable *a* and most significant by *b*



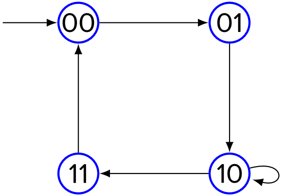
Liveness property - *AFp*

- Consider a 2-bit counter with least significant bit represented by Boolean variable a and most significant by b
- Transition relation is different from previous. An extra transition is added from state $(1, 0)$ back to itself
  - Let $\mathbb{T}(s, s') = (a' \leftrightarrow \neg a) \wedge (b' \leftrightarrow (a \oplus b))$
  - Hence transition relation will be $T(s, s') = \mathbb{T}(s, s') \wedge (b \wedge \neg a \wedge b' \wedge \neg a')$



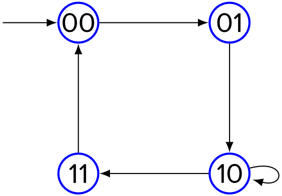
Liveness property - *AFp*

- Consider a 2-bit counter with least significant bit represented by Boolean variable a and most significant by b
- Transition relation is different from previous. An extra transition is added from state $(1, 0)$ back to itself
  - Let $\mathbb{T}(s, s') = (a' \leftrightarrow \neg a) \wedge (b' \leftrightarrow (a \oplus b))$
  - Hence transition relation will be $T(s, s') = \mathbb{T}(s, s') \wedge (b \wedge \neg a \wedge b' \wedge \neg a')$
- Suppose we claim this new counter must eventually reach state $(1, 1)$
  - The property can be expressed as $\boldsymbol{AF}(b \wedge a)$
  - This can expressed as $\boldsymbol{EG}p$ where $p = \neg b \vee \neg a$



Liveness property - I

- **Let us assume $k = 2$ for checking EGp**
- **All candidate paths will have 3 states, initial state (s_0) and two other states $\{s_1, s_2\}$ reached upon successive transition**
- **Unrolled transition relation is $[\![M]\!]_2 = I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2)$**
- **For a valid path s_0, s_1, s_2 must be part of a loop**
  - **There must be a transition from the last state, s_2, back to either s_0, s_1, or itself**
  - **This is $T(s_2, s_3) \wedge (s_3 = s_0 \vee s_3 = s_1 \vee s_3 = s_2)$**
- **We need to further constrain that the p must hold on every state of the path**



Liveness property - II

- **Set of constraints**

$$I(s_0): \quad (\qquad \neg a_0 \wedge \neg b_0 \qquad) \wedge$$

Liveness property - II

- **Set of constraints**

$$
\begin{array}{rlcl}
I(s_0): & (& \neg a_0 \wedge \neg b_0 &)\wedge \\
T(s_0,s_1): & (& ((a_1 \leftrightarrow \neg a_0) \wedge (b_1 \leftrightarrow (a_0 \oplus b_0)) \vee b_1 \wedge \neg a_1 \wedge b_0 \wedge \neg a_0 &)\wedge \\
T(s_1,s_2): & (& ((a_2 \leftrightarrow \neg a_1) \wedge (b_2 \leftrightarrow (a_1 \oplus b_1)) \vee b_2 \wedge \neg a_2 \wedge b_1 \wedge \neg a_1 &)\wedge \\
T(s_2,s_3): & (& ((a_3 \leftrightarrow \neg a_2) \wedge (b_3 \leftrightarrow (a_2 \oplus b_2)) \vee b_3 \wedge \neg a_3 \wedge b_2 \wedge \neg a_2 &)\wedge
\end{array}
$$

Liveness property - II

- **Set of constraints**

$$
\begin{array}{rll}
I(s_0): & (& \neg a_0 \wedge \neg b_0 \quad)\wedge \\
T(s_0,s_1): & (& ((a_1 \leftrightarrow \neg a_0) \wedge (b_1 \leftrightarrow (a_0 \oplus b_0)) \vee b_1 \wedge \neg a_1 \wedge b_0 \wedge \neg a_0 \quad)\wedge \\
T(s_1,s_2): & (& ((a_2 \leftrightarrow \neg a_1) \wedge (b_2 \leftrightarrow (a_1 \oplus b_1)) \vee b_2 \wedge \neg a_2 \wedge b_1 \wedge \neg a_1 \quad)\wedge \\
T(s_2,s_3): & (& ((a_3 \leftrightarrow \neg a_2) \wedge (b_3 \leftrightarrow (a_2 \oplus b_2)) \vee b_3 \wedge \neg a_3 \wedge b_2 \wedge \neg a_2 \quad)\wedge \\
s_3 = s_0: & (& (a_3 \leftrightarrow a_0) \wedge (b_3 \leftrightarrow b_0) \vee \\
s_3 = s_1: & & (a_3 \leftrightarrow a_1) \wedge (b_3 \leftrightarrow b_1) \vee \\
s_3 = s_2: & & (a_3 \leftrightarrow a_2) \wedge (b_3 \leftrightarrow b_2) \quad)\wedge
\end{array}
$$

Liveness property - II

- **Set of constraints**

$$
\begin{array}{rlcl}
I(s_0): & (& \neg a_0 \wedge \neg b_0 &)\wedge \\
T(s_0, s_1): & (& ((a_1 \leftrightarrow \neg a_0) \wedge (b_1 \leftrightarrow (a_0 \oplus b_0)) \vee b_1 \wedge \neg a_1 \wedge b_0 \wedge \neg a_0 &)\wedge \\
T(s_1, s_2): & (& ((a_2 \leftrightarrow \neg a_1) \wedge (b_2 \leftrightarrow (a_1 \oplus b_1)) \vee b_2 \wedge \neg a_2 \wedge b_1 \wedge \neg a_1 &)\wedge \\
T(s_2, s_3): & (& ((a_3 \leftrightarrow \neg a_2) \wedge (b_3 \leftrightarrow (a_2 \oplus b_2)) \vee b_3 \wedge \neg a_3 \wedge b_2 \wedge \neg a_2 &)\wedge \\
s_3 = s_0: & (& (a_3 \leftrightarrow a_0) \wedge (b_3 \leftrightarrow b_0) \vee & \\
s_3 = s_1: & & (a_3 \leftrightarrow a_1) \wedge (b_3 \leftrightarrow b_1) \vee & \\
s_3 = s_2: & & (a_3 \leftrightarrow a_2) \wedge (b_3 \leftrightarrow b_2) &)\wedge \\
p(s_0): & (& a_0 \vee b_0 &)\wedge \\
p(s_1): & (& a_1 \vee b_1 &)\wedge \\
p(s_2): & (& a_2 \vee b_2 &)
\end{array}
$$

- **The SAT instance is satisfiable. Satisfying assignment corresponds to a path from initial state** $(0, 0)$ **to** $(0, 1)$ **and then to** $(1, 0)$**, followed by self-loop at** $(1, 0)$
  - **This is a counterexample to** $AF(b \wedge a)$

Recap: SAT

- A formula f in CNF is represented as a set of clauses
- Each clause is a set of literals and each literal is either +ve or -ve propositional variable
- A formula is a conjunction of clauses and clause is a disjunction of literals
- Example
  - $((a \vee \neg b \vee c) \wedge (d \vee \neg e))$ is represented as $\{\{a, \neg b, c\}, \{d, \neg e\}\}$

Conversion to CNF

- Given a Boolean formula f, Boolean operators in f may be replaced with $\neg, \vee, \wedge$ and apply the distributive rule and De Morgan's law to convert f in CNF

- Brute force approach
  - Build the truth table of the formula
  - For each row that gives F, generate a conjunction of literals and then negate it, obtain a clause
  - Take the conjunction of all clauses generated in the previous step
- Complexity – exponential
- There exist better approach

Equisatisfiability

- **It preserves satisfiability of the original formula by adding extra variables**
  - **It results in equisatisfiable formula**

- **Example**
  - $ab \vee cd \rightsquigarrow (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$ **– standard translation, logically equivalent**

  - **Using additional variables**

$$
\begin{aligned}
(a \wedge b) \vee (c \wedge d) \quad &\rightsquigarrow \quad (e \leftrightarrow a \wedge b) \wedge (f \leftrightarrow c \wedge d) \wedge (e \vee f) \\
&= \quad (e \vee f) \wedge (e \rightarrow a \wedge b) \wedge (a \wedge b \rightarrow e)(f \rightarrow c \wedge d) \wedge (c \wedge d \rightarrow f) \\
&= \quad (e \vee f) \wedge (\bar{e} \vee a) \wedge (\bar{e} \vee b) \wedge (\bar{a} \vee \bar{b} \vee e) \wedge (\bar{f} \vee c) \wedge (\bar{f} \vee d) \wedge (\bar{c} \vee \bar{d} \vee f)
\end{aligned}
$$

Advantages of BMC

- **Able to handle larger state spaces as compared to Binary Decision Diagrams**
- **Takes advantage of several decades of research on efficient SAT solvers.**
- **The witness/counterexample produced are usually of minimum possible length, making them easier to understand and analyze**

Limitations

- Sound but not complete
  - Works for a bounded depth
  - In order to have a complete procedure, we need to run it at least up to the diameter (unknown) of the transition system
- For larger depths the number of clauses can grow rapidly, thereby raising capacity issues
- Nevertheless, SAT-based FPV tools can handle much larger designs as compared to BDD-based tools

Thank you!