Topic 5

BDD-Based Verification

系統晶片驗證

SoC Verification

2025.03.26

Outline

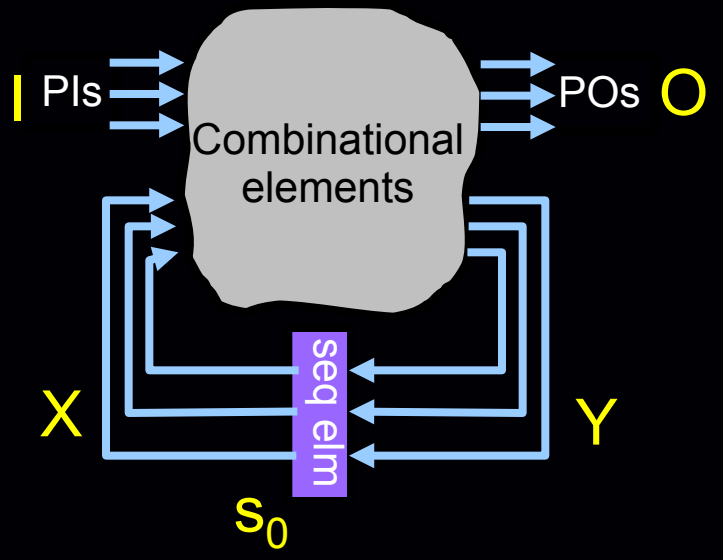
1. Introduction to Binary Decision Diagram (BDD)
2. Proving combinational properties using BDDs
   - Baseline algorithms
   - Other advanced techniques
3. Proving sequential properties using BDDs
   - Reachability analysis
   - Fixed point computation
   - Other advanced techniques

We have discussed how to use BDD to prove the combinational assertion properties.

Now let's see how to use it for sequential properties.

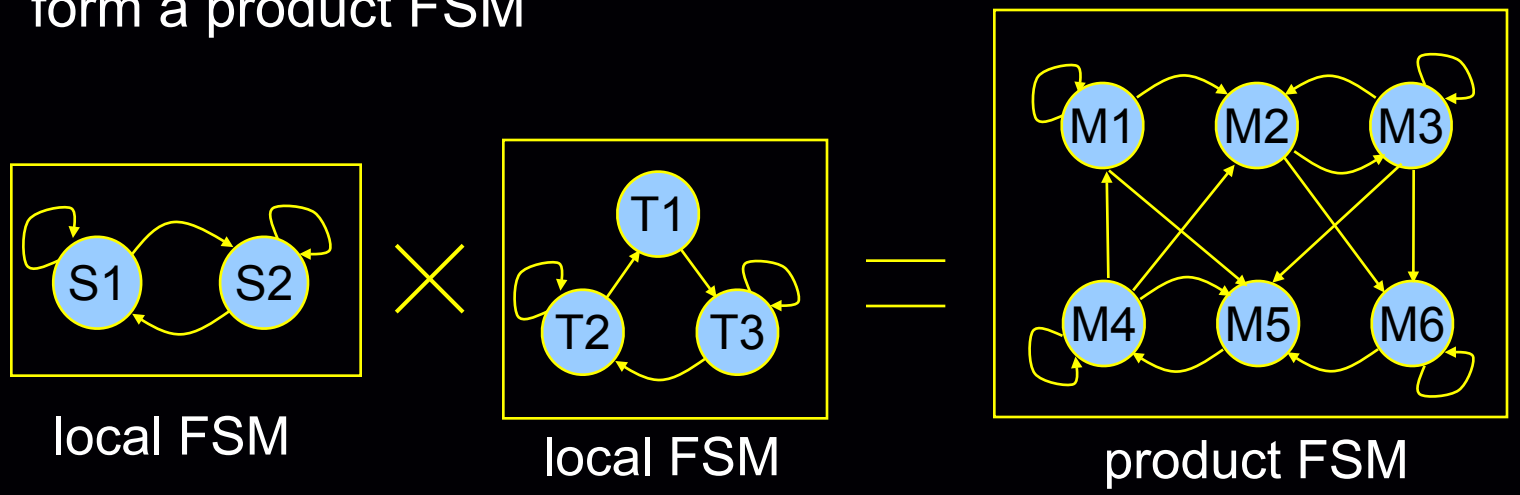
Refresh: Problem Modeling

- The DUV is in circuit Hoffman model
  - Circuit is divided into combinational and sequential portions
  - $\mathcal{M}$ = (I, X, Y, O, δ, λ, S_0)
    - I: vector of n input variables
    - X: vector of m current state vars
    - Y: vector of m next state vars
    - O: vector of k output variables
    - δ : vector of state transition functions ($2^n \times 2^m \rightarrow 2^m$)
    - λ : vector of output functions ($2^n \times 2^m \rightarrow 2^k$)
    - S_0: initial state of the circuit



Product Finite State Machine

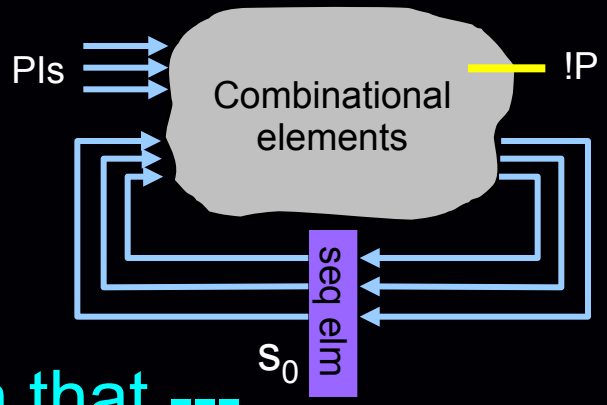
- A design usually contains several modules in certain hierarchy
  - Control logic
    - Enabling signals, conditional statements, operator selection, etc
  - Datapath
    - Arithmetic units, data flow, storage, etc
  - ➔ Sequential control logic is usually represented as a FSM
- From the top-level circuit point of view, these "local" FSMs form a product FSM

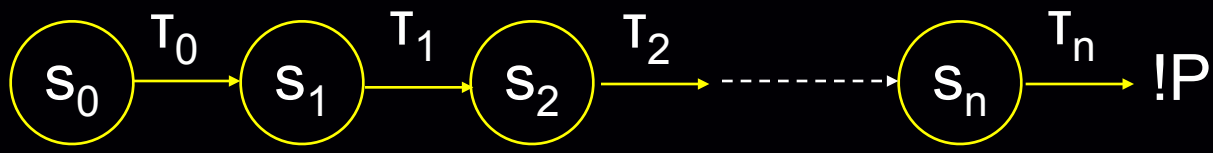


local FSM　　local FSM　　product FSM

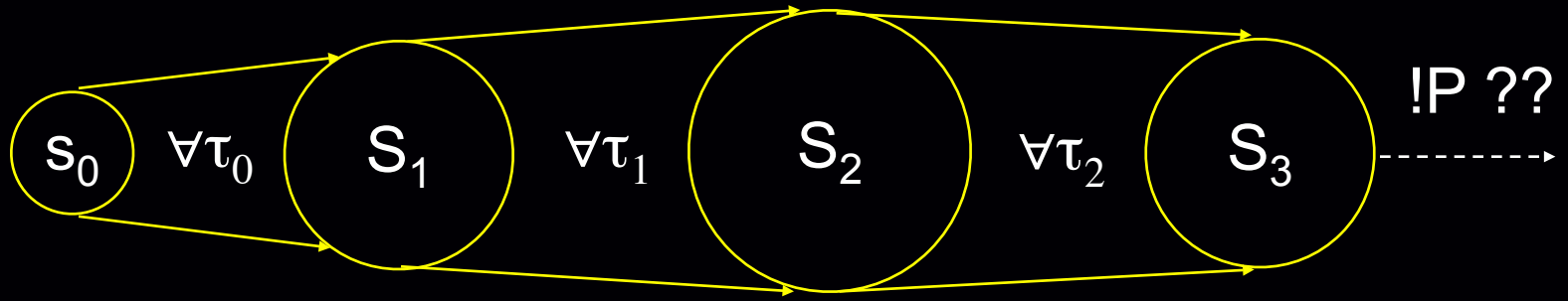
BDD to Prove Assertion Property

- Assert_always(P) = AG(P)
- AG(P) = ¬EF(¬P)
  - Try to witness ¬P
  - Find a input sequence such that ---

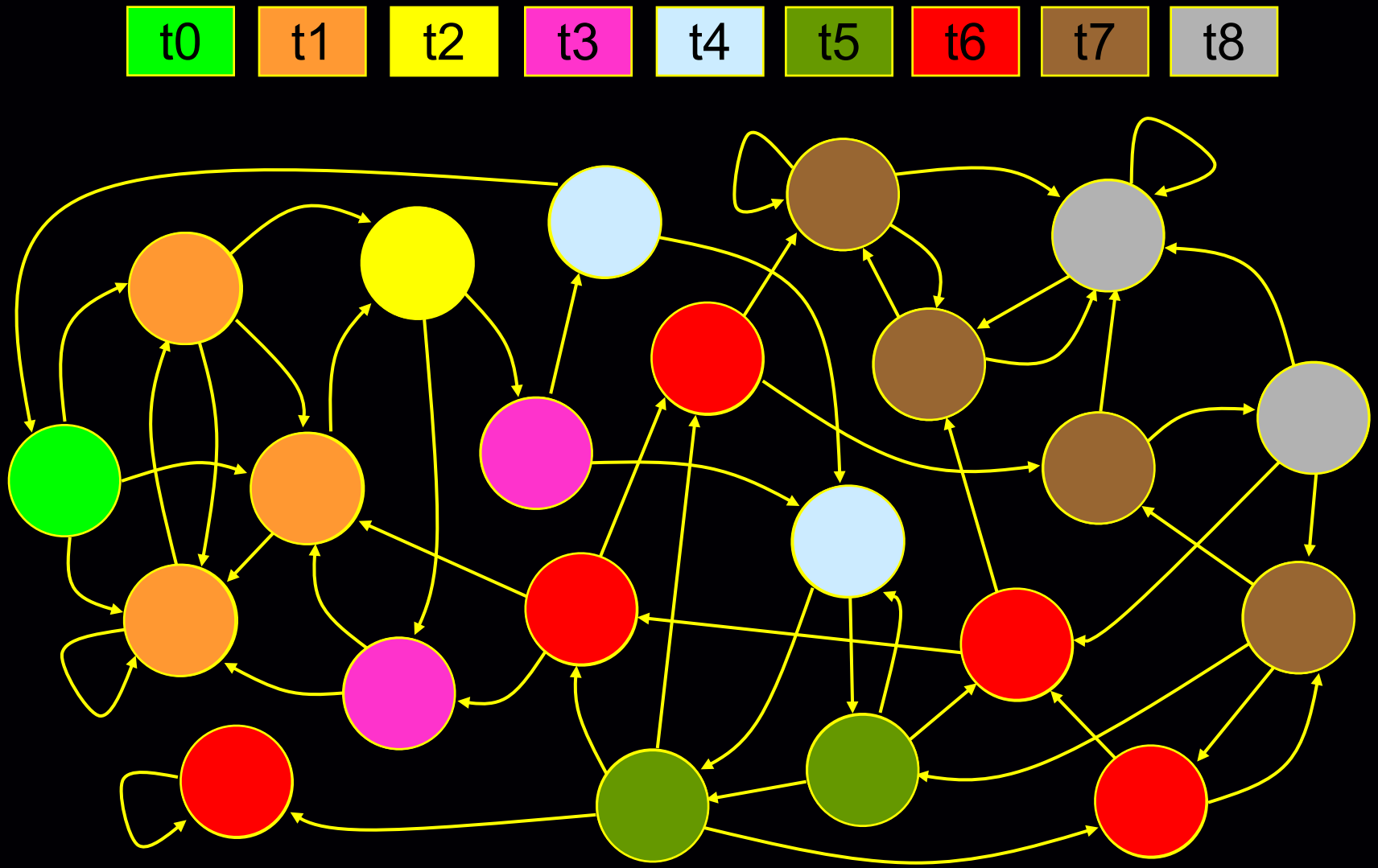




  - To prove AG(P), we need to compute ---

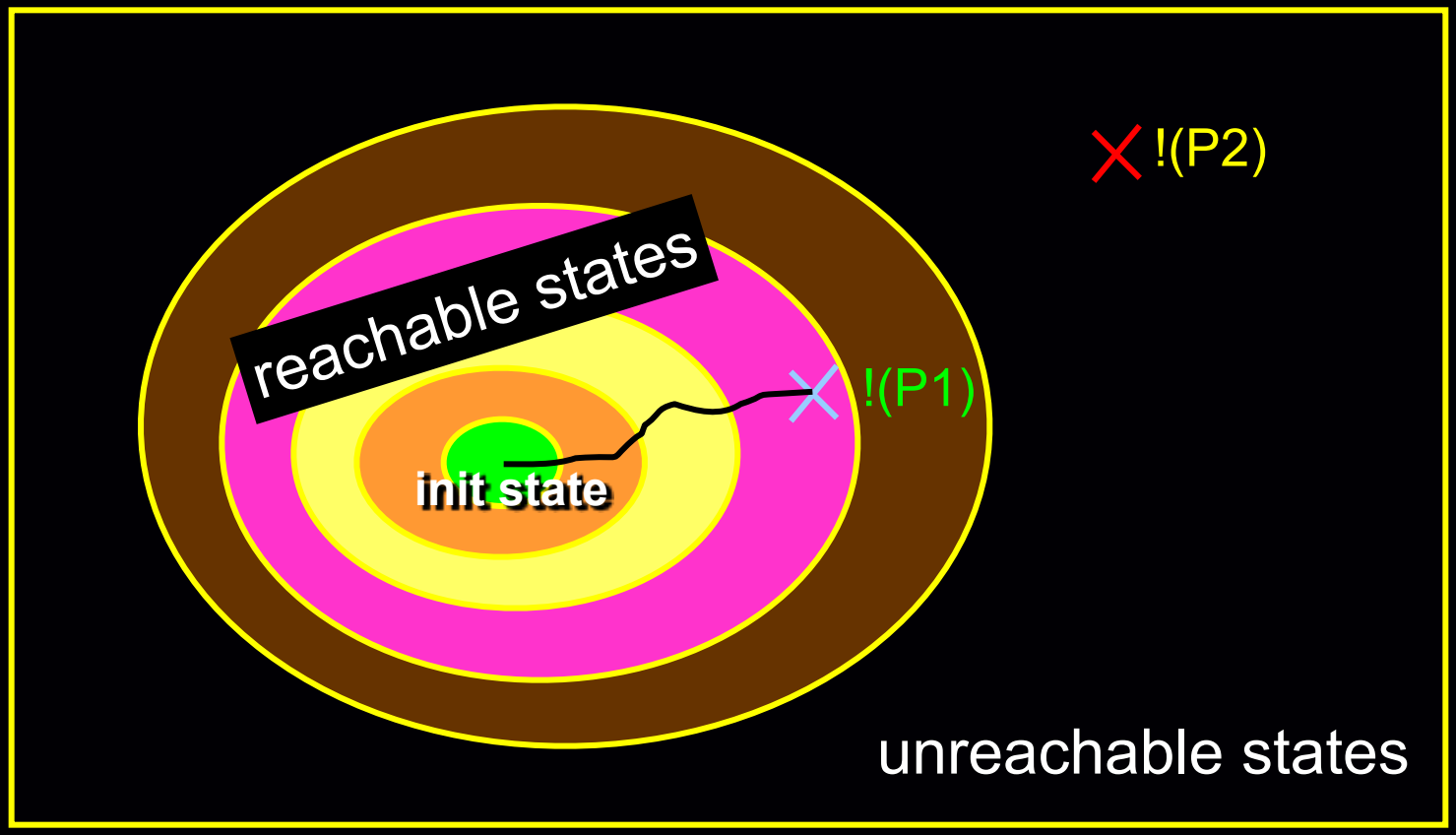


Set of Reachable States



Boolean State Space

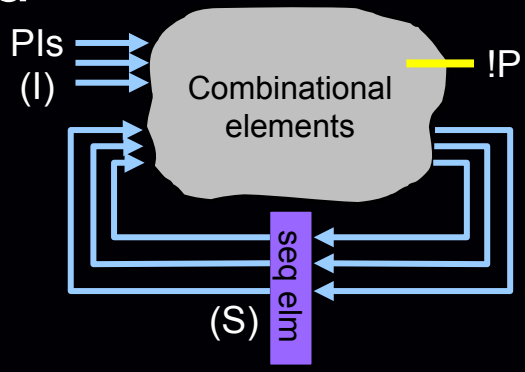
Boolean space of n state variables (2^n)



➔ AG (p1) ≡ false; AG (p2) ≡ true

BDD-based Sequential Proof

- Property (P) can be represented in terms of state variables (S) and PIs (I)
  - Build BDD for P(S, I)
  - or, build BDD for ¬P(S, I)



- Set of reachable states (R) is a function of state variables (S)
  - Build BDD for R(S)
- If $R(S) \wedge \neg P(S, I) = \varnothing$

  → No counter sequence can be found for P

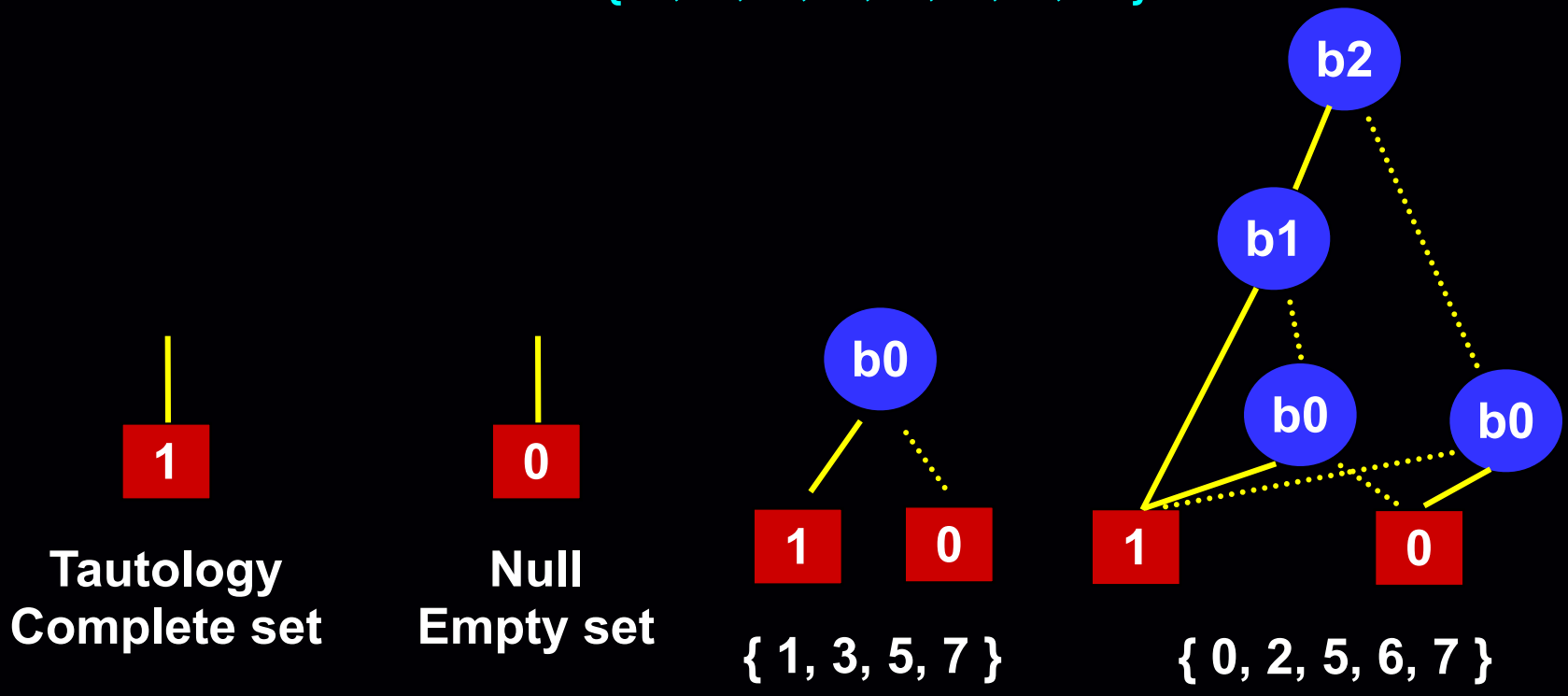
  → P is proven TRUE!!

But, the problem is...

How to represent and compute the set of reachable states R(S) in BDD?

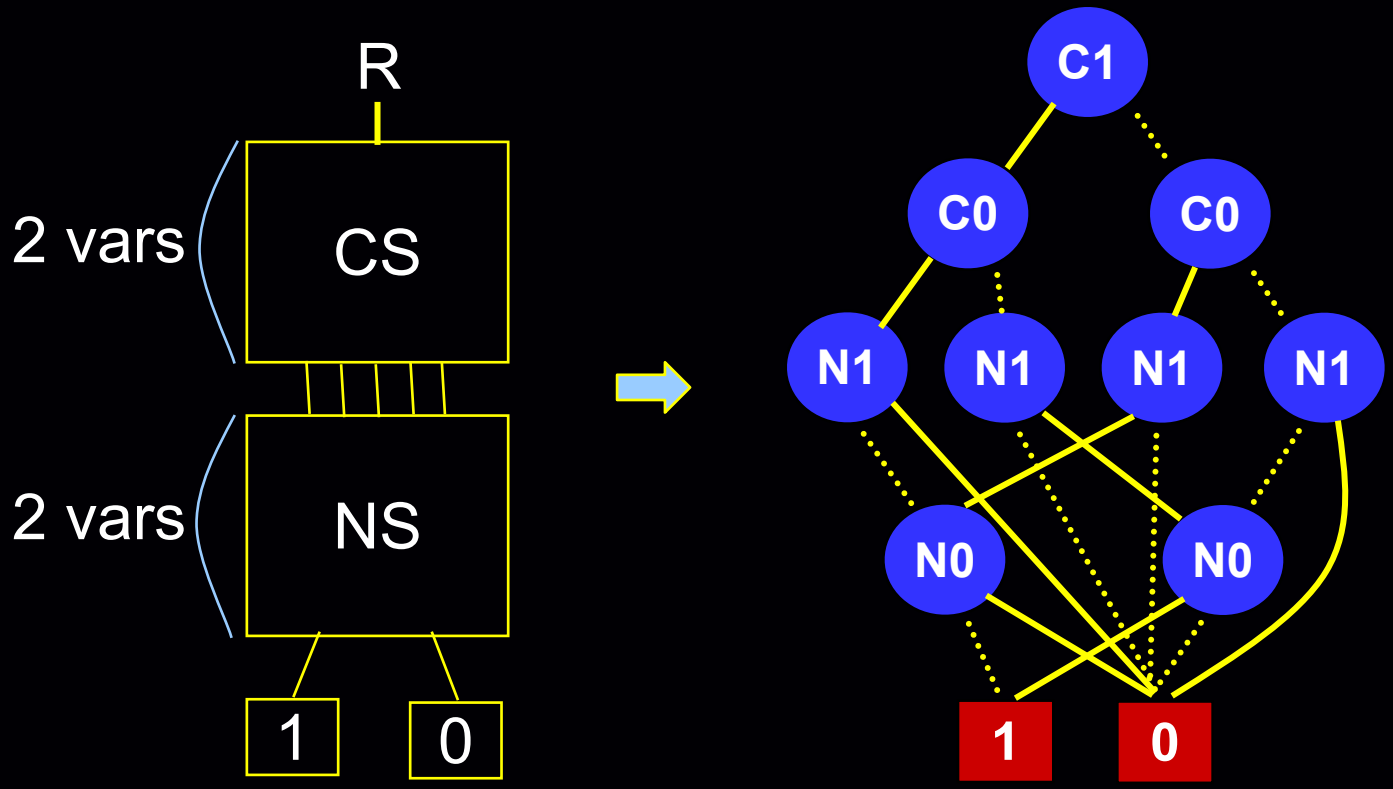
Recall: BDD to Represent a Set

- Other than Boolean function, BDD can also represent sets over Boolean variables
  - e.g. Use 3-variable BDDs to represent any subset of the numbers in { 0, 1, 2, 3, 4, 5, 6, 7 }



BDD can also represent relations

- e.g. 2-bit ring counter: NS = (CS + 1) % 4
  R: { (0→1), (1→2), (2→3), (3→0) }

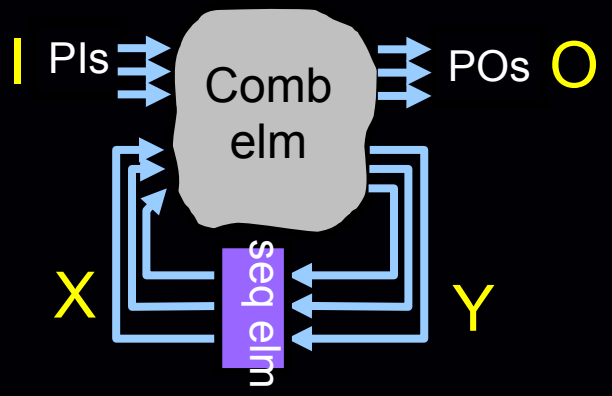


For BDD to represent
the set of reachable states R(S),

we need to introduce the concept of
"(state) transition relationship (TR)",
and how to use BDD to represent TR.

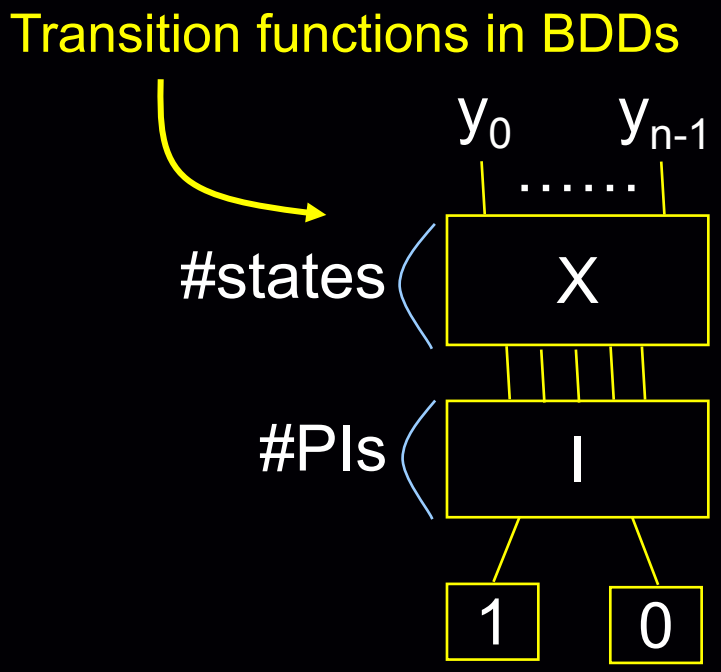
From Transition Functions to Transition Relationship, to Set of Reachable States

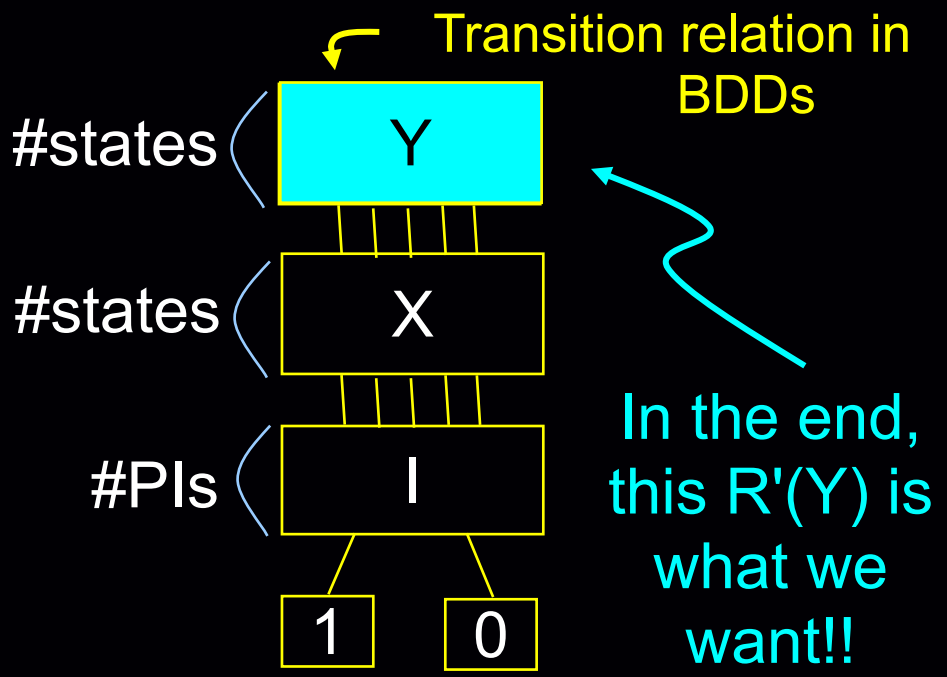
- Assume current set of reachable states are represented as R(X) // Init states = ?
- We would like to compute next set of reachable states as R'(Y)
  - What's the relationship between R(X) and R'(Y)?
  - What's the relationship between X and Y?
- Recall: Transition Functions: Y = δ(X, I)
  - For each state variable, $y_i = \delta_i(X, I)$
  - → We can build BDDs for y_i's
- Define: Transition Relationship: TR(Y, X, I)
  - For each legal transition (X, I) →Y, the relation TR(Y, X, I) = 1
  - → How to build the BDD for TR(Y, X, I)?
  - → If we have R(X), how to compute the image on Y?? and represent it as BDD: R'(Y) ??



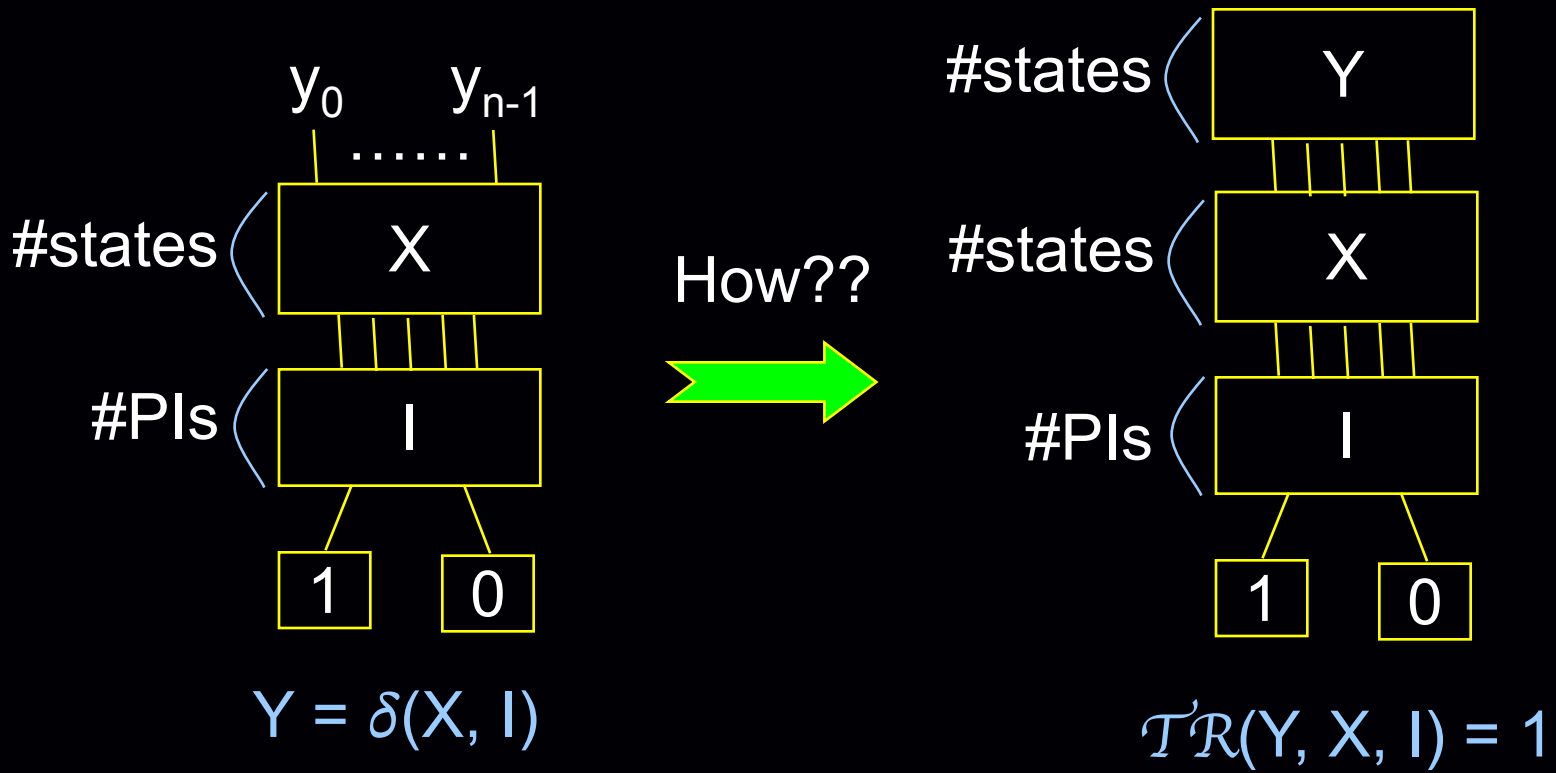
From Transition Functions to Transition Relationship, to Set of Reachable States

- In other words, given ---
- We want to compute ---





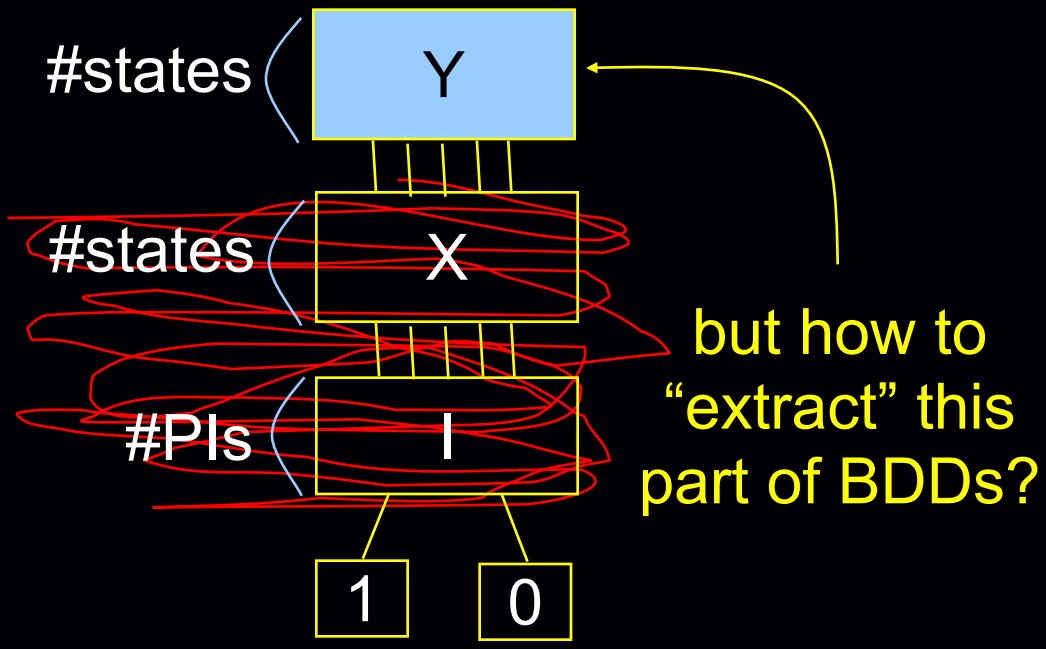
From Transition Function to Transition Relationship



- $\mathcal{TR}(Y, X, I) = \prod(y_i = \delta_i(X, I))$

  $= (y_0 = \delta_0(X, I)) \wedge (y_1 = \delta_1(X, I)) \wedge \ldots$

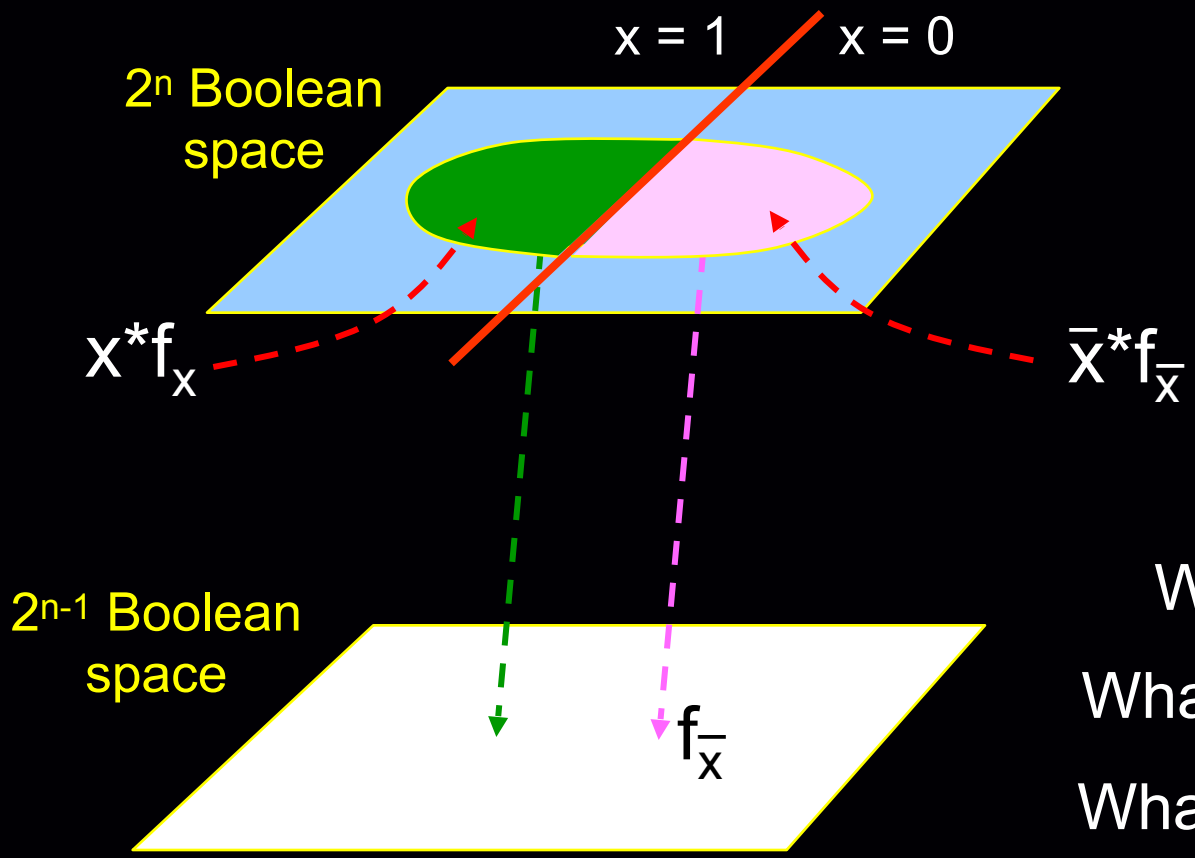
The Question is…



Erase BDD variables ➔ Existential Quantification

Remember: co-factors...

- Fallacy: $f_x \wedge f_{\bar{x}} = \varnothing$



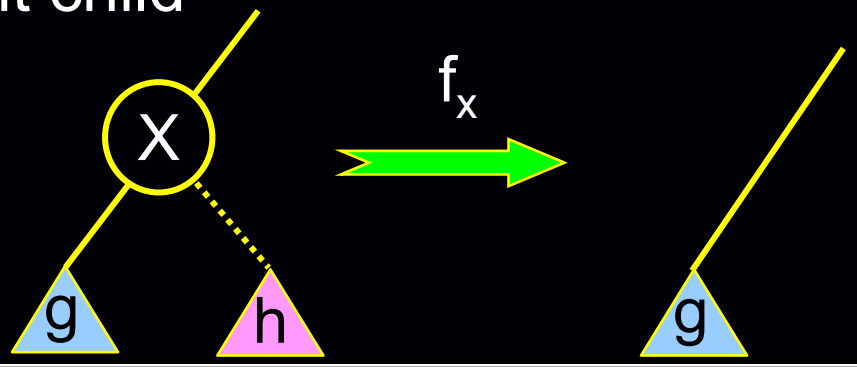
What is $x*f_x$?

What is $(f_x \wedge f_{\bar{x}})$?

What is $(f_x \oplus f_{\bar{x}})$?

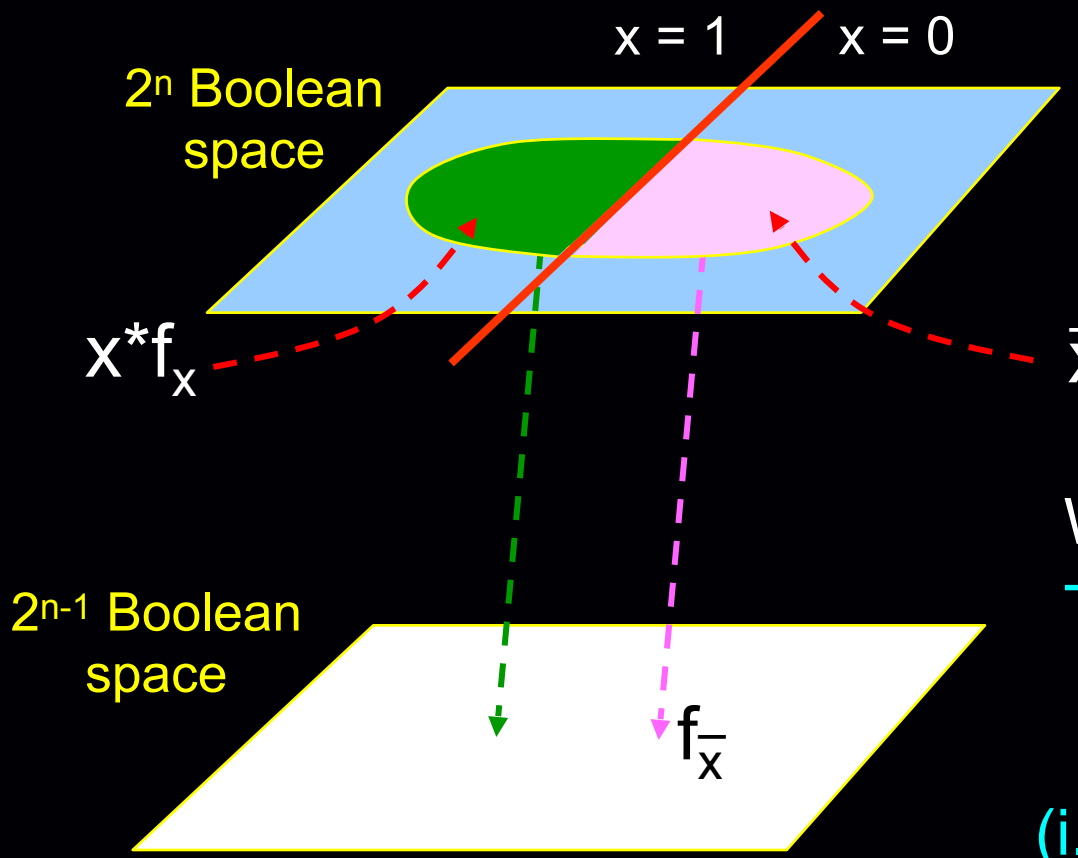
Computing Cofactors on BDD

- Given a function f
  ➔find its positive/negative cofactor f_x / $f_{\bar{x}}$
  - e.g. Let f = a $\bar{c}$ + b c
    - ➔ f_c = b
    - ➔ $f_{\bar{c}}$ = a
    - ➔ $f_{\bar{a}}$ = b c
- If x is top variable
  ➔ cofactor = left or right child
- Otherwise,
  for each node in level X



Existential Quantification

- $\exists x.f = f_x + f_{\bar{x}}$ (⬅ What does this mean?)

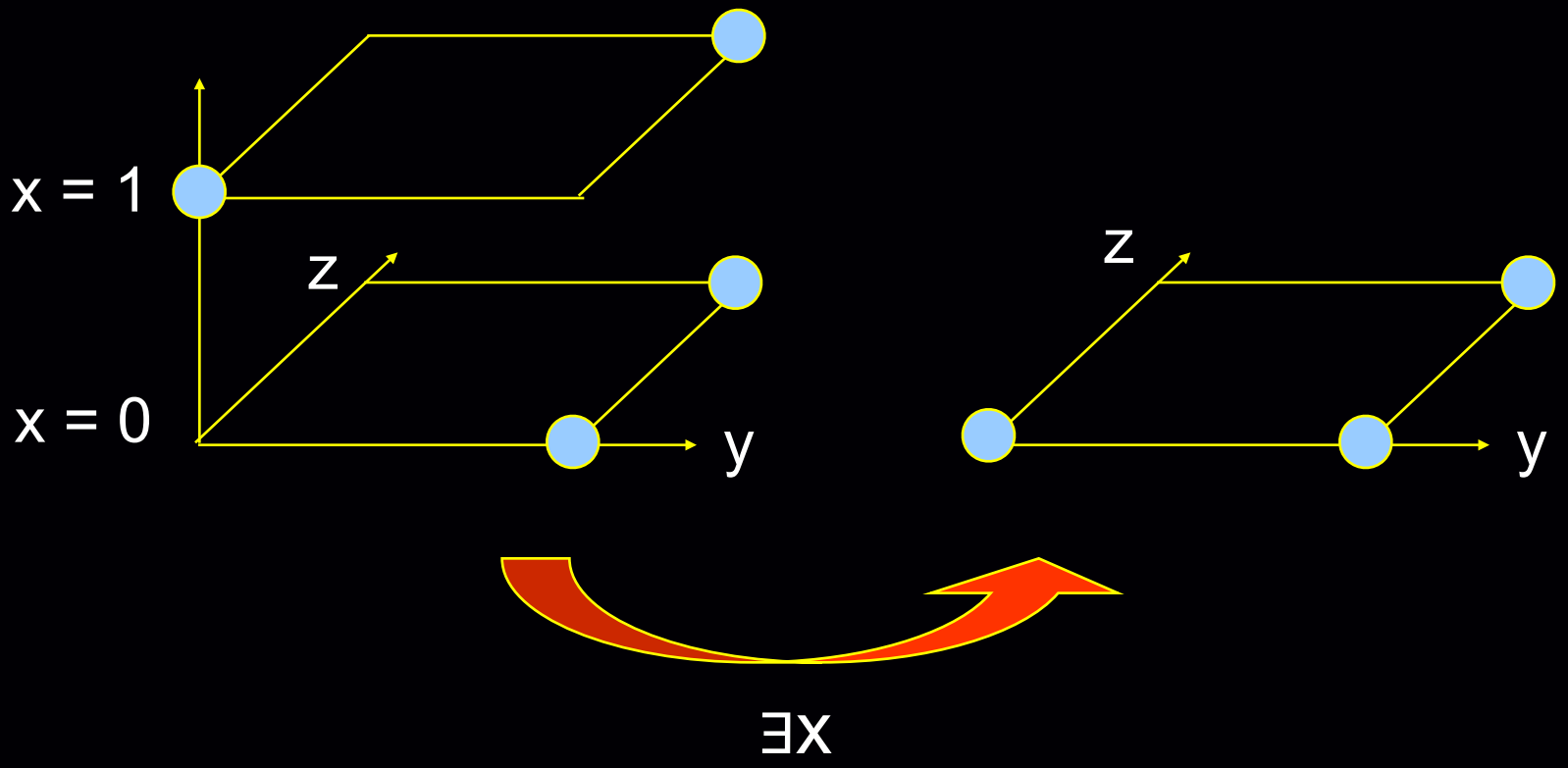


What is ($f_x + f_{\bar{x}}$) ?

This is exactly the projection of f on the 2^{n-1} plane (i.e. regardless of x)

Existential Quantification

- $\exists x.f = f_x + f_{\bar{x}}$ (⇐ What does this mean?)



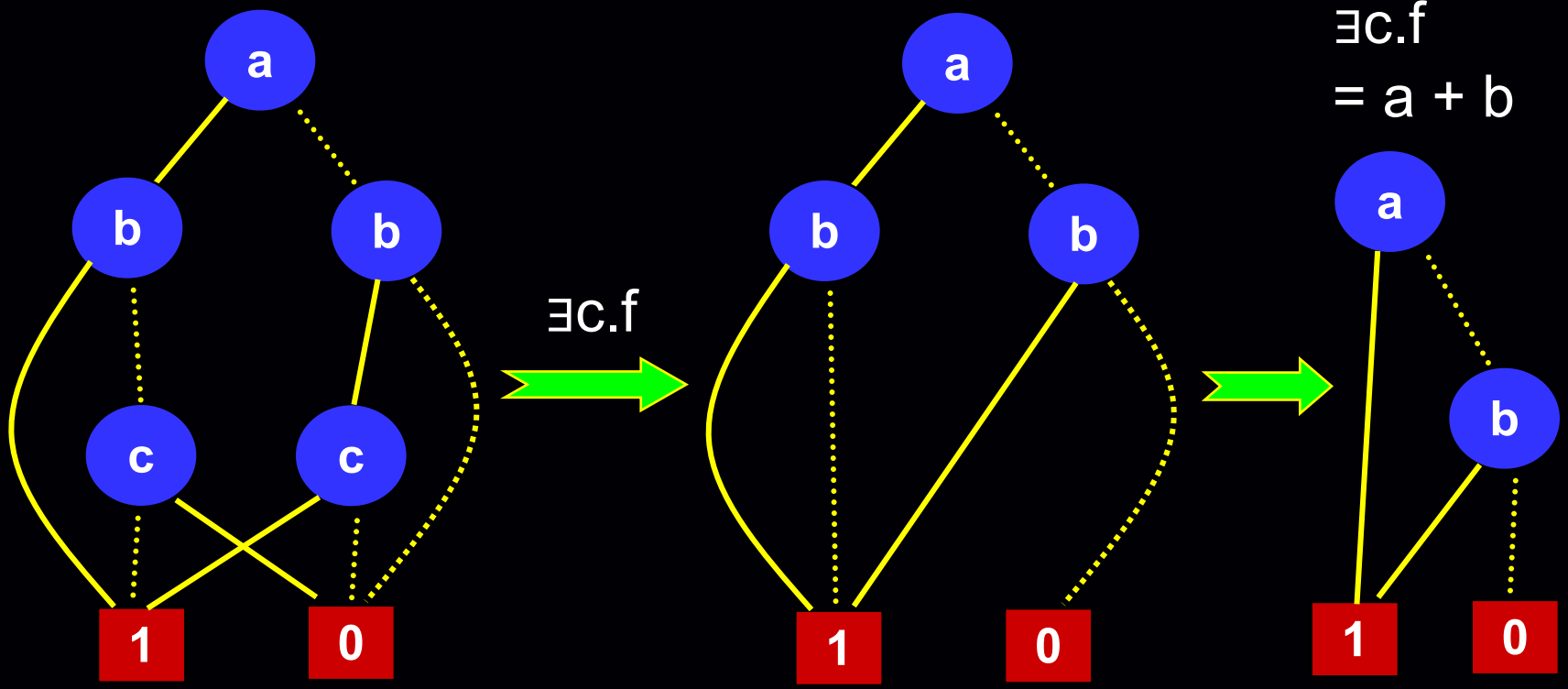
Existential Quantification on BDD

- $\exists x.f = f_x + f_{\bar{x}}$ (🡸 How to perform it on BDD?)
- If x is top variable
  - 🡺 Perform an "OR" on its cofactors
- If x is bottom variable
  - 🡺 Replace non-zero nodes with '1' (why?)
- If x is middle variable
  - 🡺 ???

Which one is better??

Existential Quantification

- $\exists x.f = f_x + f_{\bar{x}}$
  - e.g. $f = a\,b + a\,\bar{b}\,\bar{c} + \bar{a}\,b\,c$



Cofactors, Boolean Difference, Existential Quantification

- Let $F = x \cdot A + \bar{x} \cdot B + C$

1. Cofactors
   - $F_x = A + C$
   - $F_{\bar{x}} = B + C$
2. Boolean difference
   - $F_d = (A \oplus B) \cdot \bar{C}$
3. Existential quantification
   - $\exists x.F = A + B + C$

- What if the formula is represented in PoS form?

BDD to Compute Set of Reachable States

- Let $S_0(X)$ be the set of initial states
  (how to represent init states in BDD?)
  - $R_0(X) = S_0(X)$: set of reachable states up to time 0
- The set of states reached in time 1 can be computed by
  - $S_1(Y) = \exists X,I\ (TR\ (Y, X, I) \wedge S_0(X))$
  - Let $S_1(X) = S_1(Y) \mid_{Y \to X}$ // Same BDD; replace X by Y
  - $R_1(X) = R_0(X) \vee S_1(X)$
- In general,
  - $S_{n+1}(Y) = \exists X,I\ (TR\ (Y, X, I) \wedge S_n(X))$
  - $S_{n+1}(X) = S_{n+1}(Y) \mid_{Y \to X}$
  - $R_{n+1}(X) = R_n(X) \vee S_{n+1}(X)$
- If $R_{n+1} = R_n$, no new state can be reached
  ➔ Fixed point condition (set of all reachable states)

Early Quantification

independent of 'I'

- $S_{n+1}(Y) = \exists X,I\ (TR\ (Y, X, I) \wedge S_n(X))$

  $= \exists X\ (\ \exists I\ (TR(Y, X, I))\ \wedge S_n(X))$

  → Let: $\mathcal{TR}'(Y, X) = \exists I\ (\mathcal{TR}(Y, X, I))$

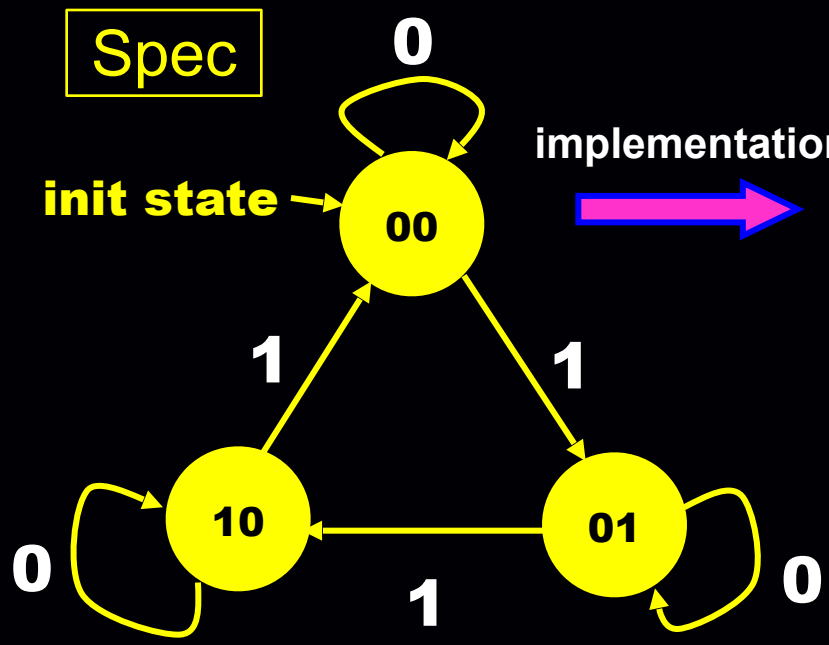
- $S_{n+1}(Y) = \exists X\ (\mathcal{TR}'(Y, X) \wedge S_n(X))$

  → We will use $\mathcal{TR}'(Y, X)$ instead of $\mathcal{TR}(Y, X, I)$

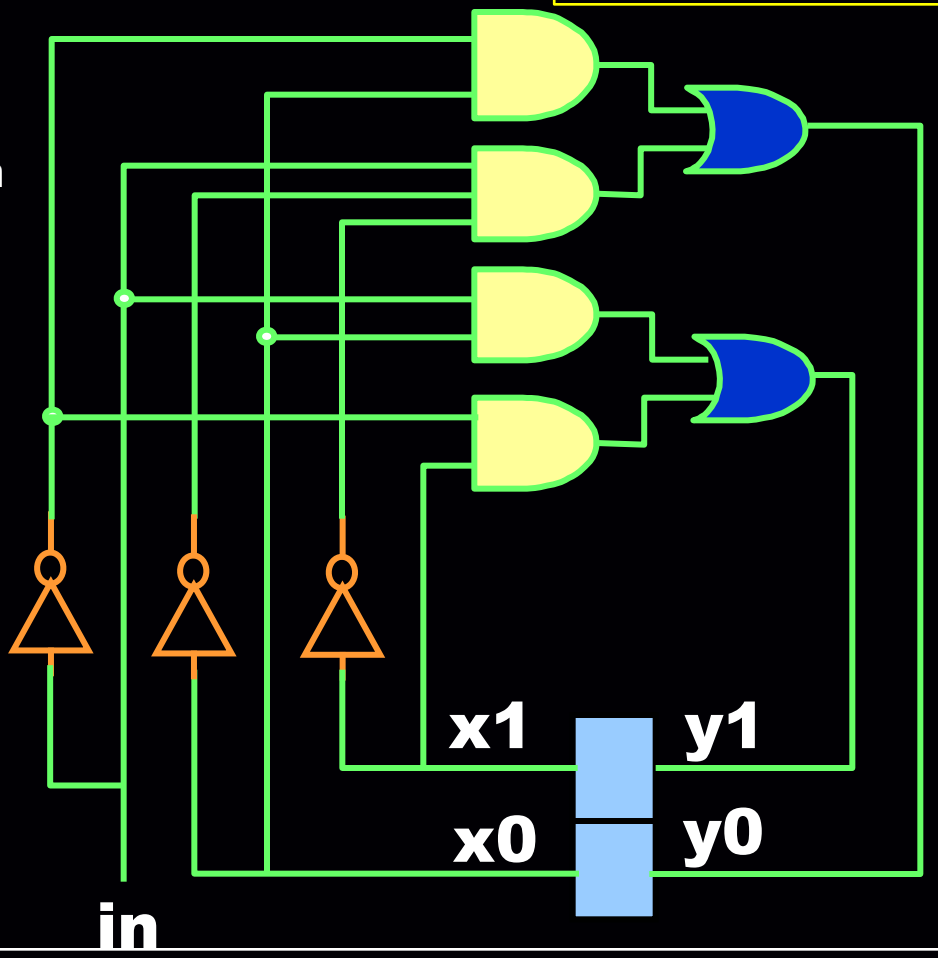
Example: Property Checking Using BDD

- Given a 2-bit counter as follows:

Is this implementation correct??



implementation



Assert: P

State <= 2

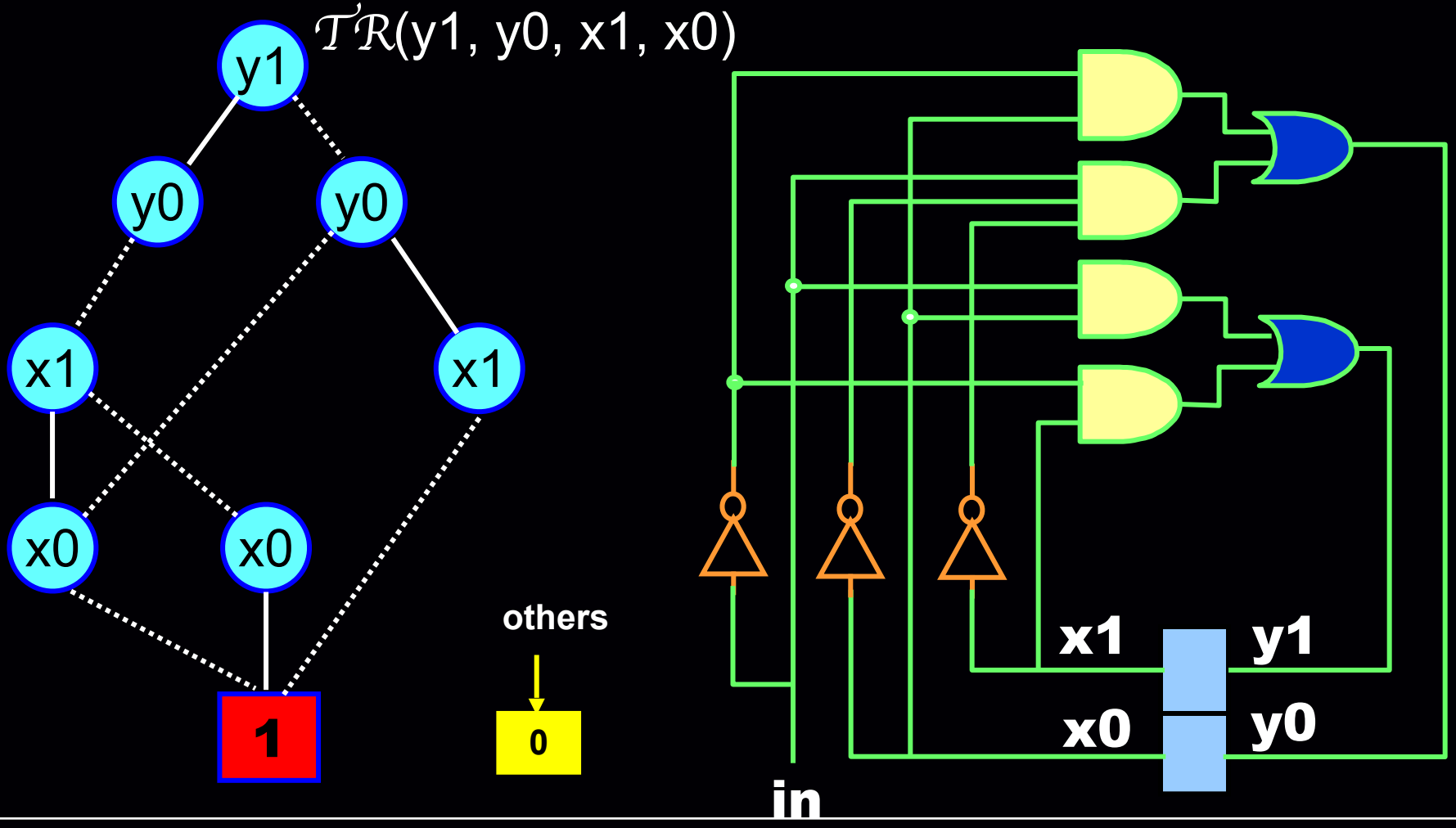
(i.e. {y1, y0} <= 2)

Proved by BDD

1. Determine the variable ordering
   - e.g. y1, y0, x1, x0
2. Compute state transition relationship ($\mathcal{TR}$) of current states and next states
   - $\mathcal{TR}$(y1, y0, x1, x0)
3. Starting from initial state R = $S_0(X)$ = { (x1, x0) = 00 }, compute the set of reachable states at time i
   - R += $S_i(X)$ until R_i = R;
   - Otherwise, R ← R_i
4. Check if !P (i.e. (y1, y0) = 11) intersects with the set of reachable states
   - Check (R && !P)
   - If yes (intersects), property is false
     else go to 3

Transition Relationship in BDD

- We first build TR from the circuit netlist as follows --



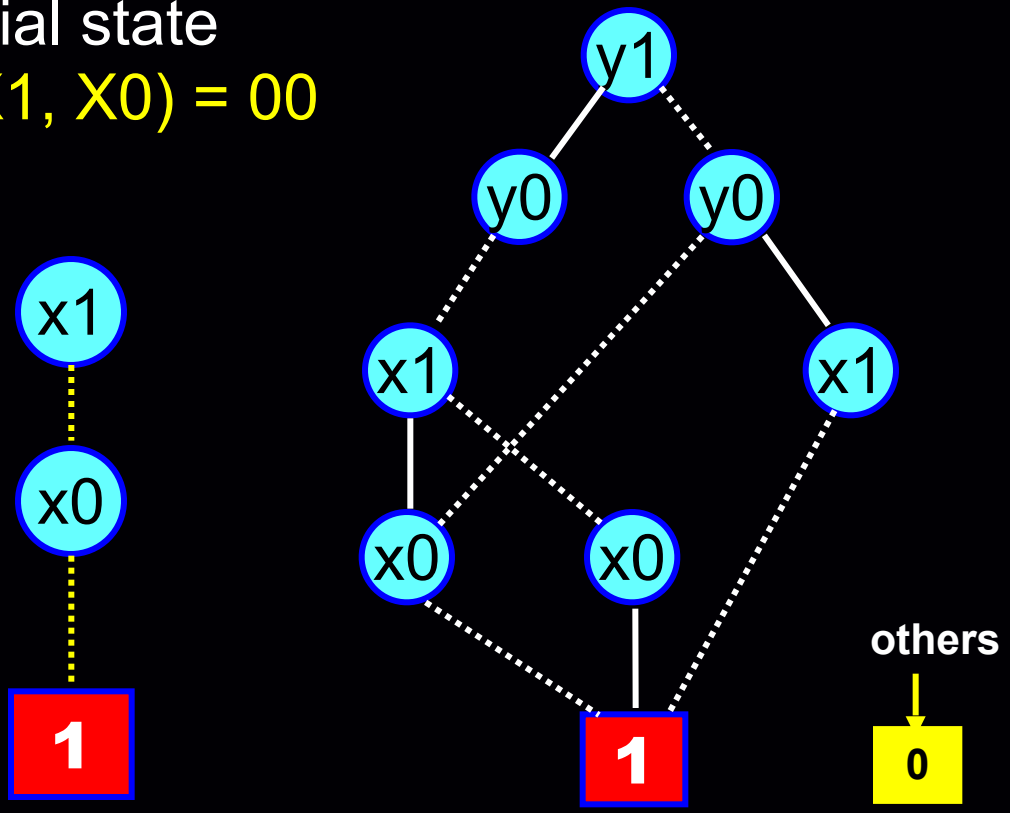
Compute Set of Reachable States (Time 1)

Transition Relationship

$\mathcal{TR}$(y1, y0, x1, x0)

initial state

R_0 : (X1, X0) = 00



What's the set of reachable state S1 for time 1?

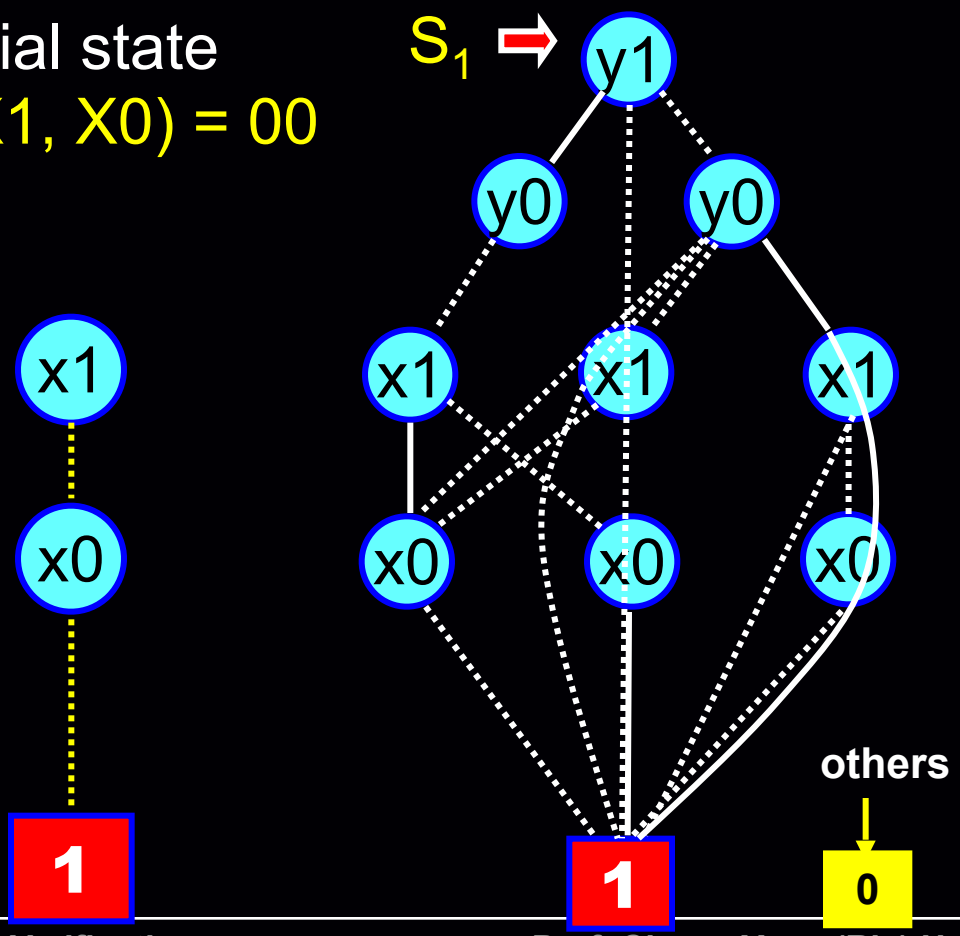
Compute Set of Reachable States (Time 1)

Transition Relationship

$\mathcal{TR}$(y1, y0, x1, x0)

initial state

R_0 : (X1, X0) = 00



To compute the set of reachable states in the next cycle ---

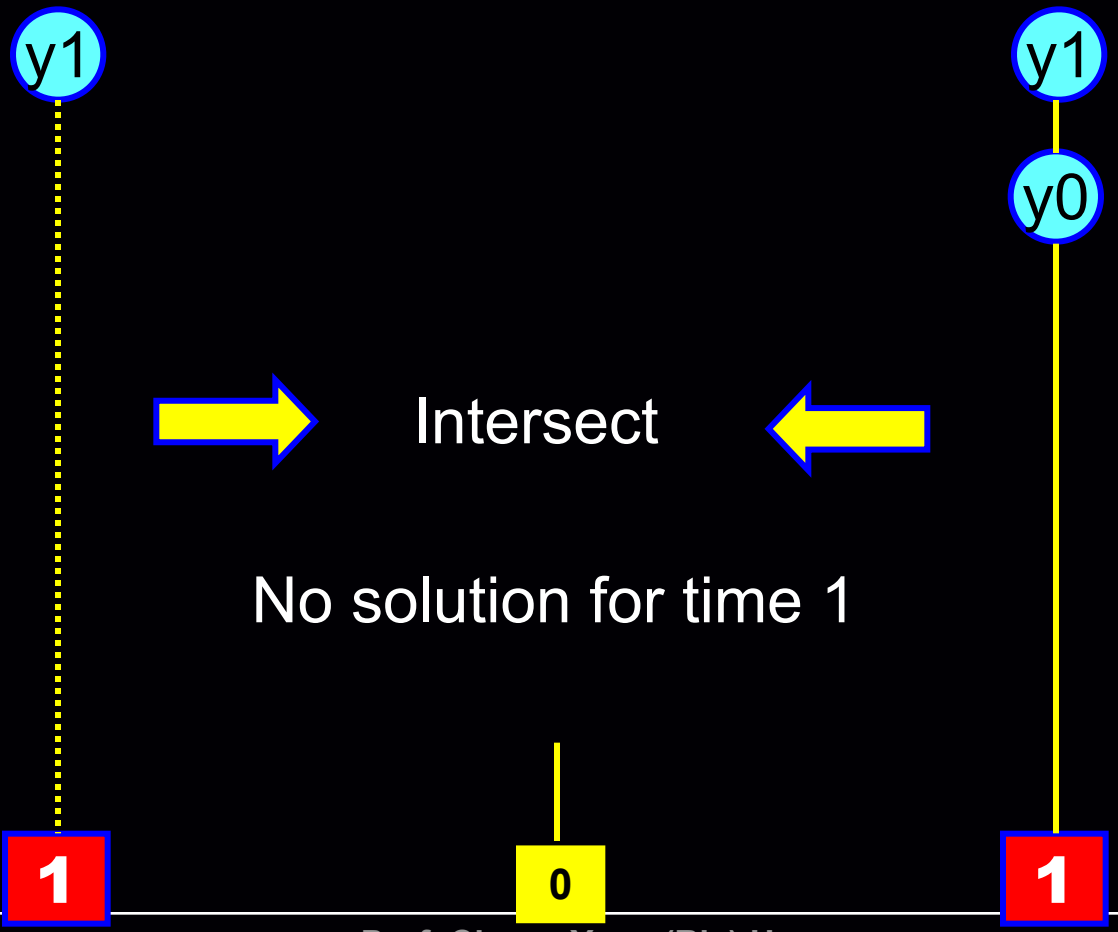
1. Build the TR BDD
2. Apply $\mathcal{TR}(X = R_0)$
3. Existential quantification

   $S_1 = \exists_{x1,x0} \mathcal{TR}(R0)$
4. Check intersection with !P

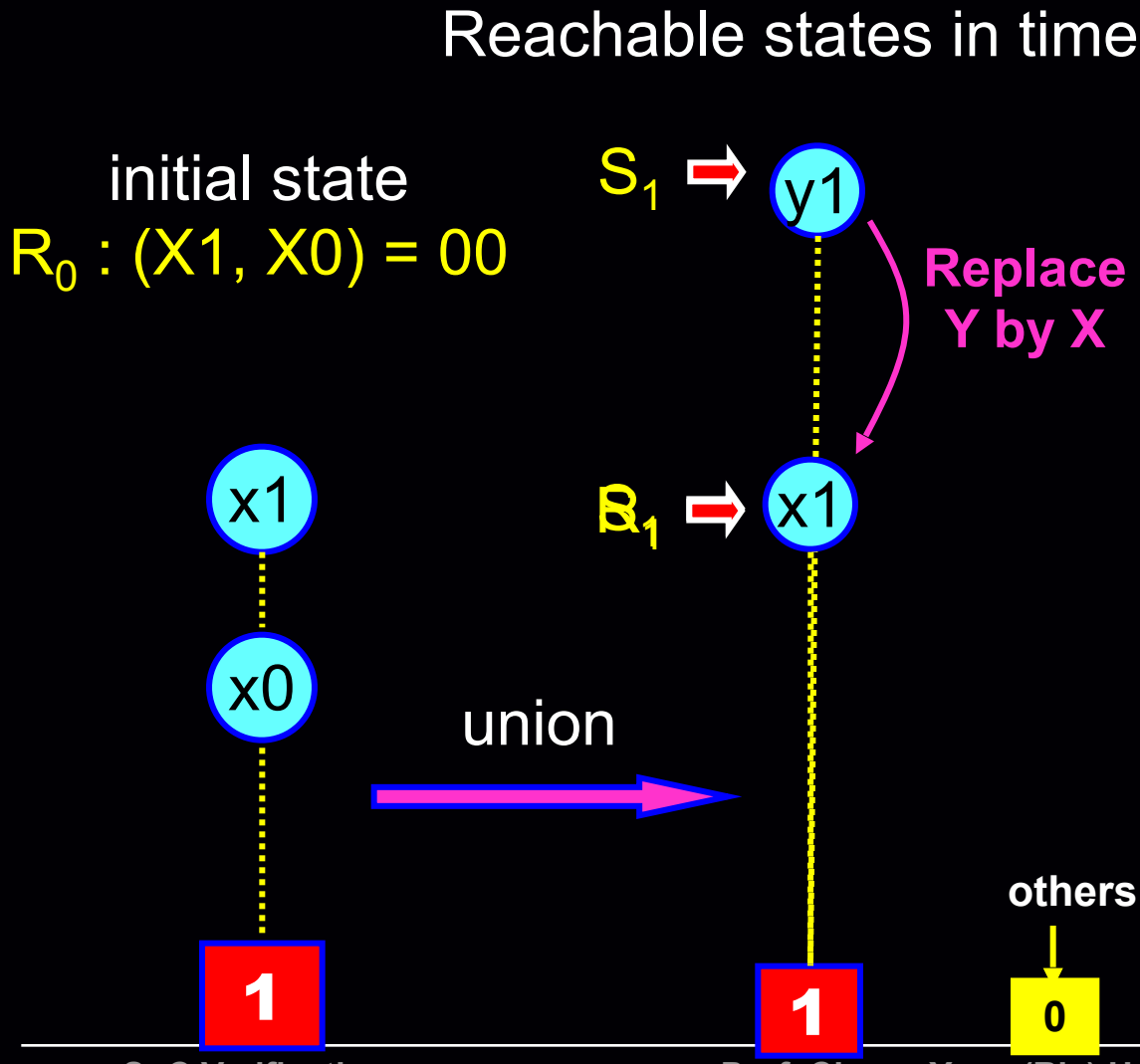
Check Intersection with !P

S_1 : (y1, y0) = 0 -
states reachable in time 1

P : (y1, y0) <= 2
➔!P : (y1, y0) = 11



Compute Set of Reachable States (Time 1)



To compute the set of reachable states in the next cycle ---

1. Build the TR BDD
2. Apply $\mathcal{TR}(R_0)$
3. Existential quantification
   $S_1 = \exists_{x1,x0} \mathcal{TR}(R0)$
4. Check intersection with !P
5. $R_1 = S_1(Y \rightarrow X) \cup R_0$

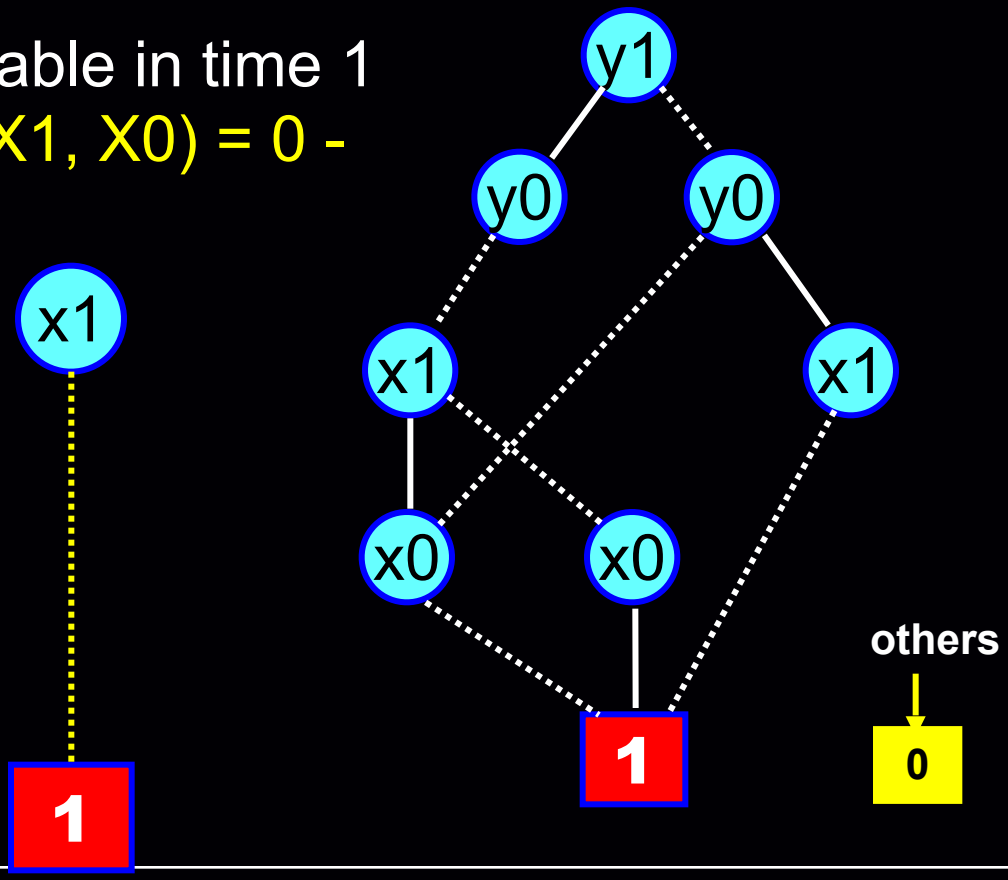
Compute Set of Reachable States (Time 2)

Transition Relationship

$\mathcal{TR}$(y1, y0, x1, x0)

Reachable in time 1

R_1 : (X1, X0) = 0 -

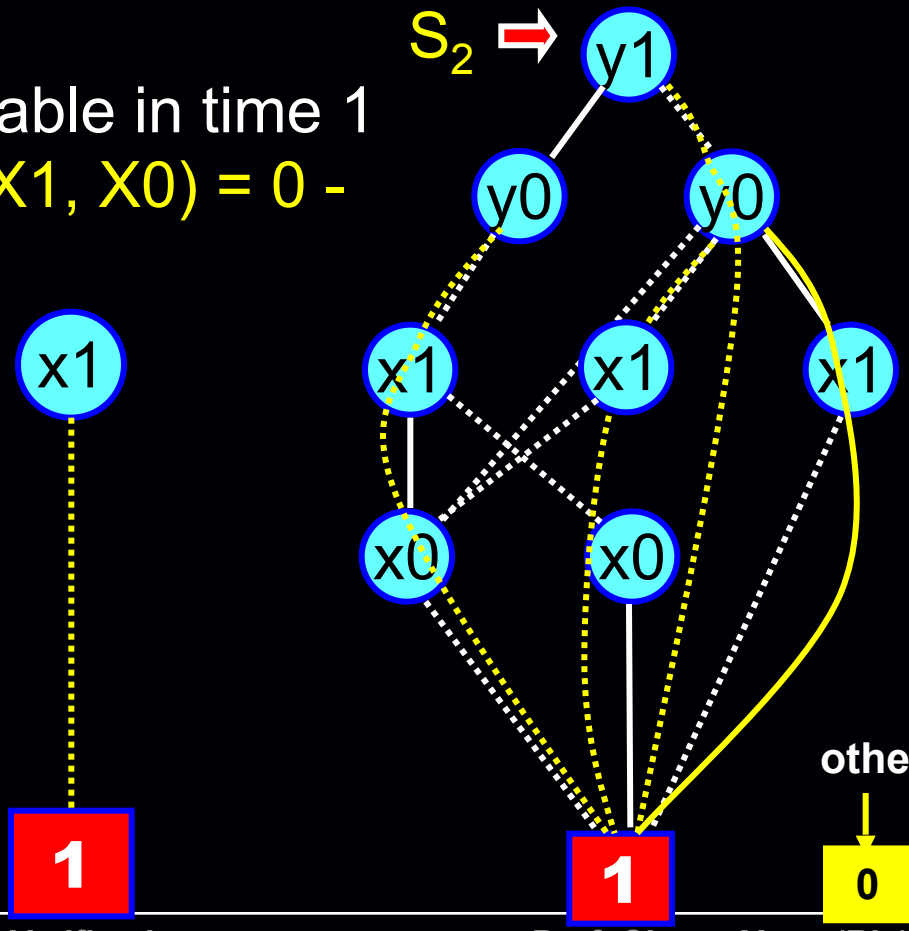


What's the set of reachable state S2 for time 2?

Compute Set of Reachable States (Time 2)

Transition Relationship

$\mathcal{TR}$(y1, y0, x1, x0)



Reachable in time 1

R_1 : (X1, X0) = 0 -

To compute the set of reachable states in the next cycle ---

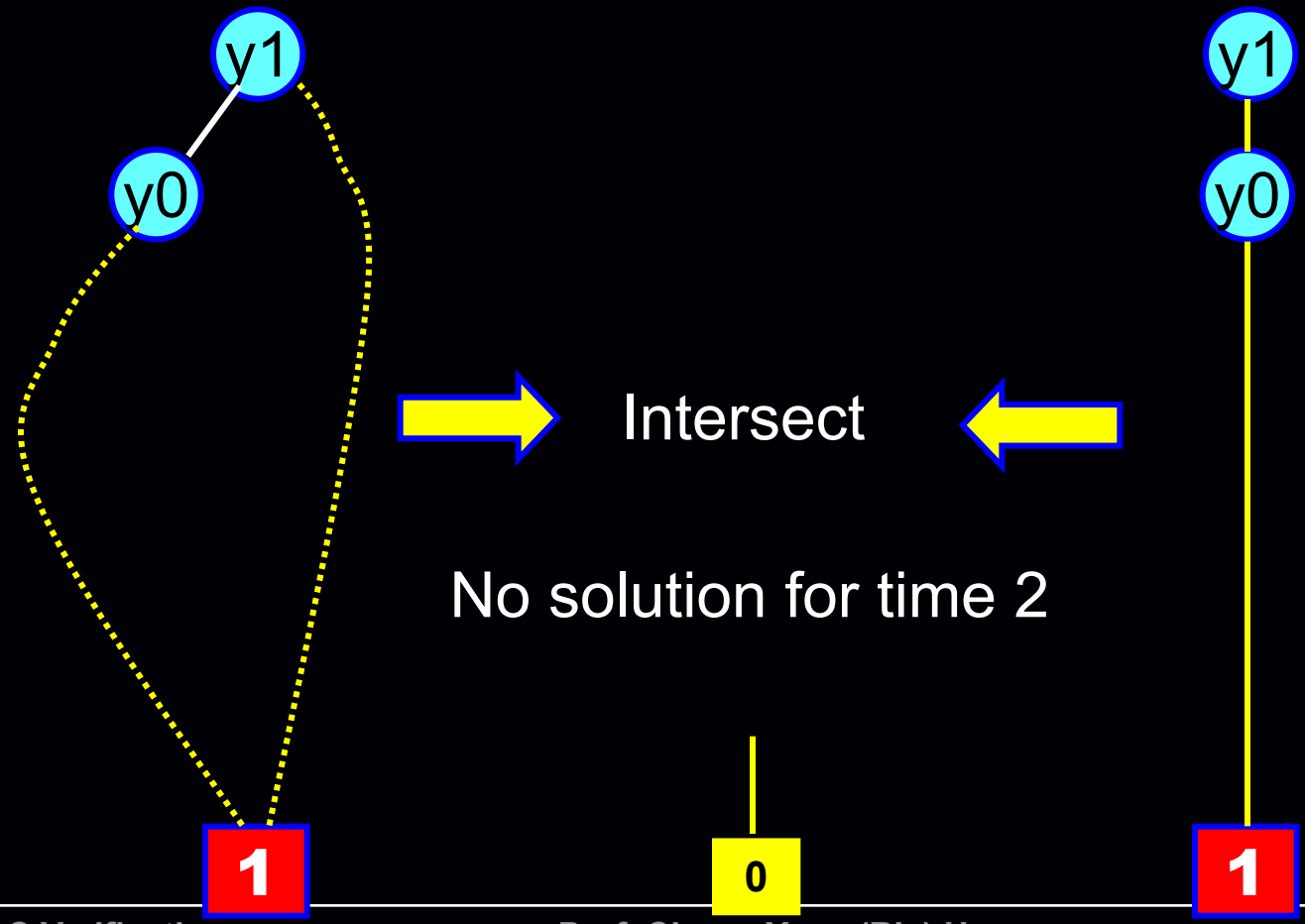
1. Apply TR(X = R_1)
2. Existential quantification

   $S_2 = \exists_{x1,x0} \mathcal{TR}(R_1)$
3. Check intersection with !P
4. $R_2 = S_2(Y \rightarrow X)\ U\ R_1$

Check Intersection

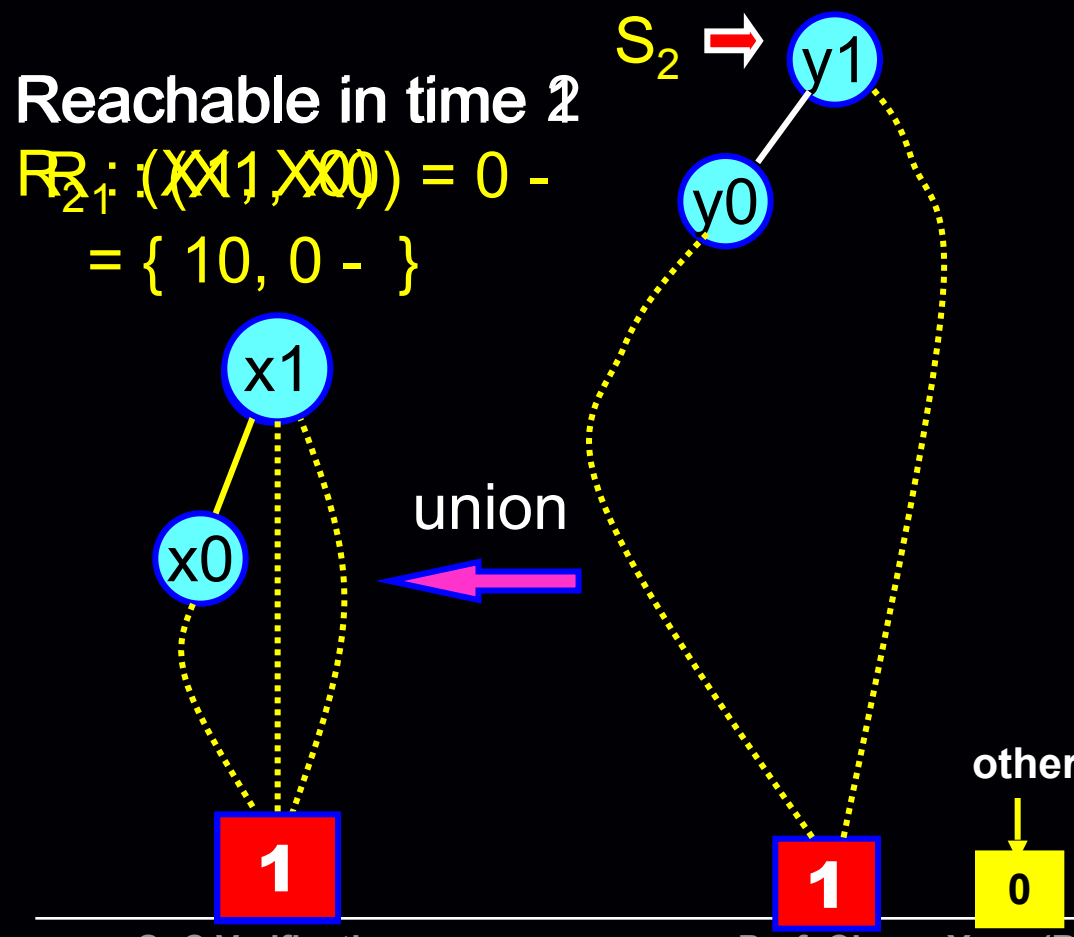
S_2 : (y1, y0) = { 10, 0 - }
states reachable in time 2

P : (y1, y0) <= 2
➔!P : (y1, y0) = 11



Compute Set of Reachable States (Time 2)

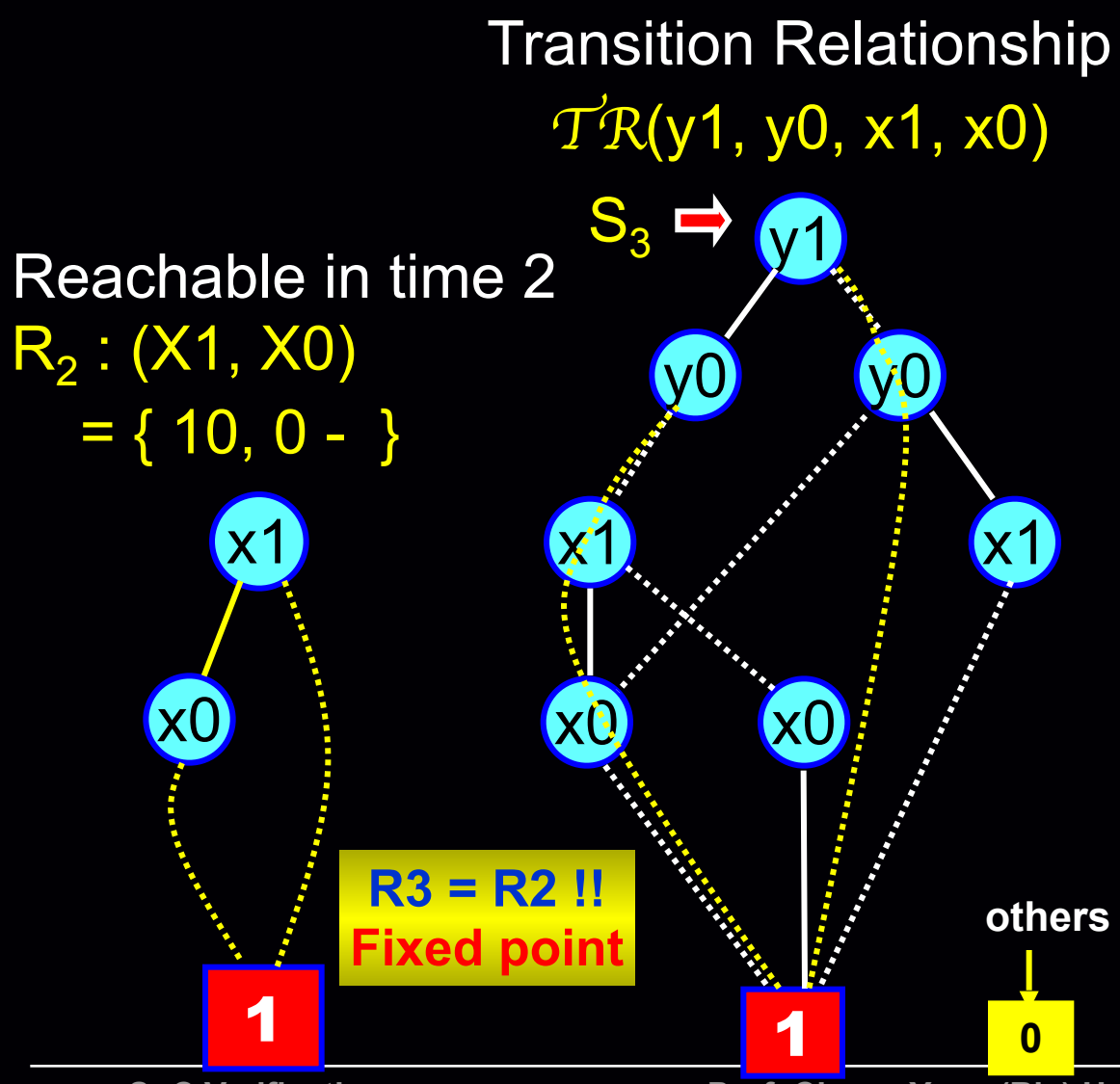
S_2 : (y1, y0) = { 10, 0 - }
states reachable in time 2



To compute the set of reachable states in the next cycle ---

1. Apply TR(R_1)
2. Existential quantification
   $S_2 = \exists_{x1,x0} \mathcal{TR}(R_1)$
3. Check intersection with !P
4. $R_2 = S_2(Y \rightarrow X)\ U\ R_1$

Compute Set of Reachable States (Time 3)



To compute the set of reachable states in the next cycle ---

1. Apply TR(R_2)
2. Existential quantification
   $S_3 = \exists_{x1,x0} \mathcal{TR}(R_2)$
3. Check intersection with !P
4. $R_3 = S_3(Y \rightarrow X) \cup R_2$

Fixed Point in State Reachability Analysis

- All the reachable states are in the set
  - Those not in the set are NOT reachable from the initial state
- If the intersection with !P is '0'
  - Property P is always true (for all input combinations in time infinity)

Property is formally proved!!

Limitation of BDD-based Sequential Proof

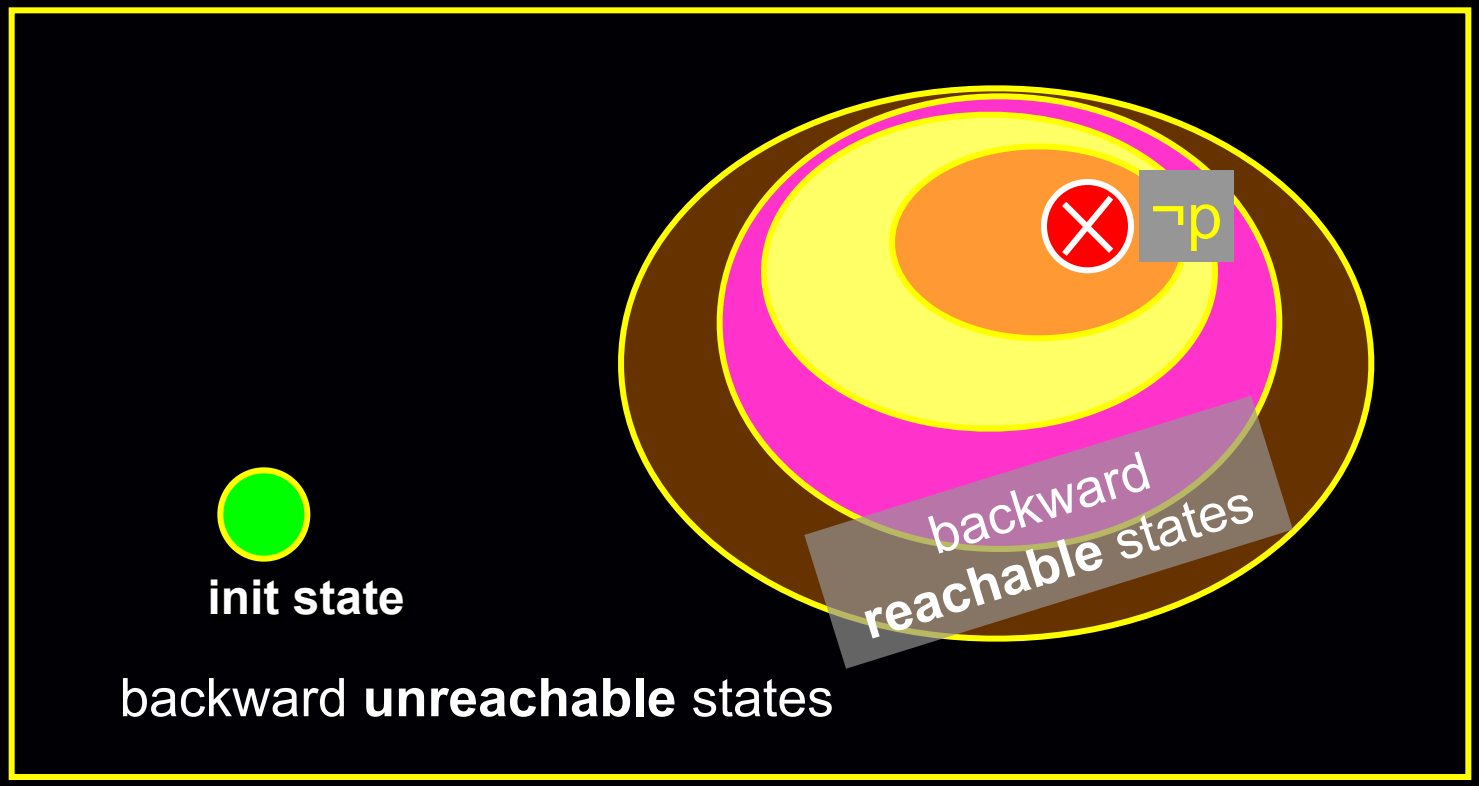
- Sounds good, if we can compute the set of reachable states, we can formally prove the properties...
- But the fact is that real design usually contains thousands of FFs, and the combinational cone may be as big as million gates....
- Any advanced techniques?

Advanced techniques for sequential BDD

1. Backward pre-image computation
2. Speeding up the image computation
   - Frontier set simplification
   - Iterative squaring
3. Simplify the transition relation BDD
   - Disjunctive partition
   - Conjunctive partition
4. Approximate BDD techniques

Boolean State Space

Boolean space of n state variables (2^n)



If (Backward reachable states) intersects (init state) = ∅

➔ Property p is true

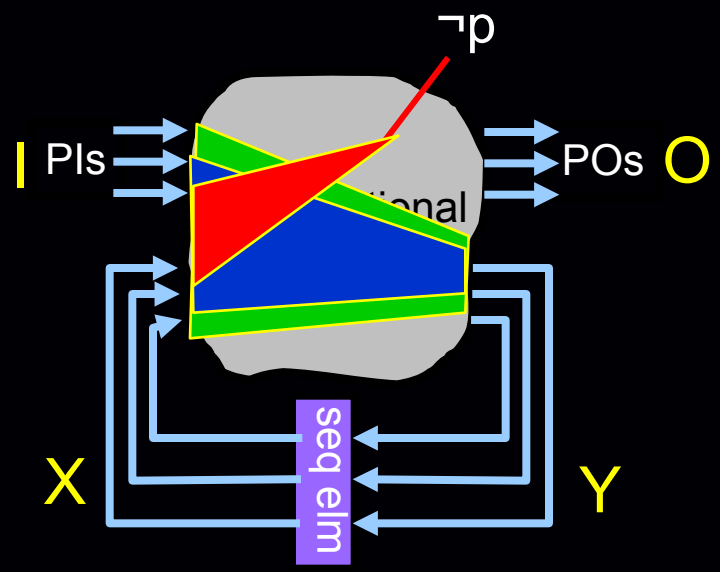
Backward Reachability Analysis

- Let the backward timeframe index be: 0 -1, -2, -3,...
  - 0: the last timeframe that satisfies ¬p
  - -n: n timeframes before the last timeframe
- Let $T_0(X)$ be the set of states that satisfies (¬p) at the last timeframe ➔
  - $B_0(X) = T_0(X)$: set of backward reachable states at the last timeframe
- The set of states in backward time -1 (T_{-1}) can be computed by ---
  - Let $T_0(Y) = T_0(X) \mid_{X \to Y}$
  - $T_{-1}(X) = \exists Y\ (\mathcal{TR}(Y, X) \wedge T_0(Y))$
  - $B_{-1}(X) = B_0(X) \vee T_{-1}(X)$
- In general,
  - $T_{-n}(Y) = T_{-n}(X) \mid_{X \to Y}$
  - $T_{-(n+1)}(X) = \exists Y\ (\mathcal{TR}(Y, X) \wedge T_{-n}(Y))$
  - $B_{-(n+1)}(X) = B_{-n}(X) \vee T_{-(n+1)}(X)$
- If $B_{-(n+1)} = B_{-n}$, no new state can be backward reached

  ➔ Fixed point condition

Can backward be faster than forward?

1. Set of backward reachable states “may be” smaller
2. Can start from a smaller set of state variables and then increase gradually

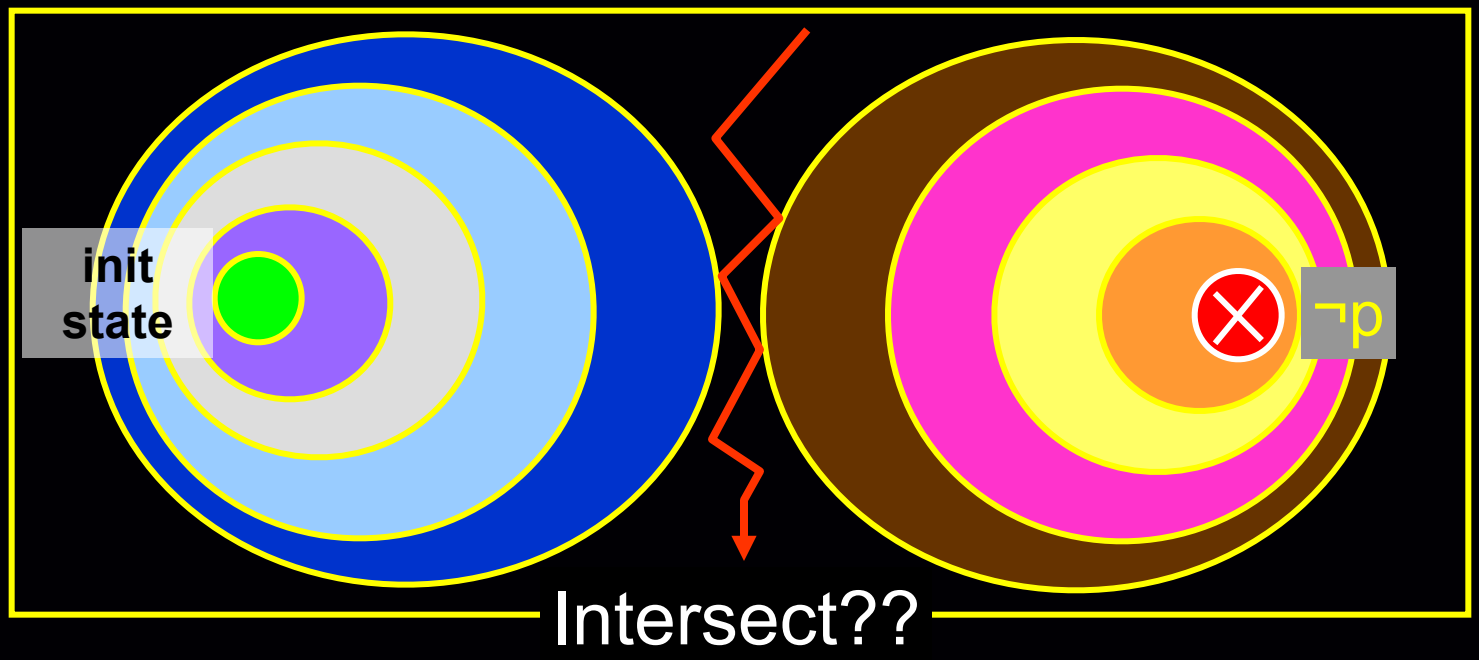


But experience shows that the effectiveness of backward approach varies from cases to cases

Combining Forward and Backward Approach

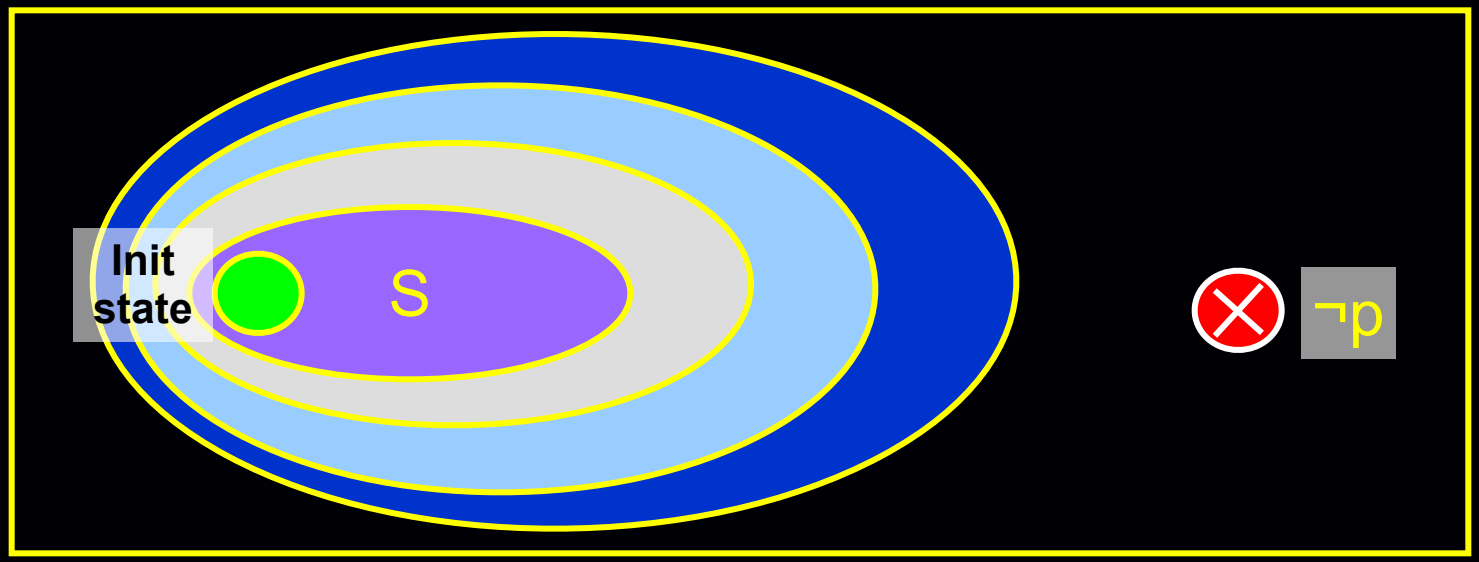
- Think: since it is difficult to predict whether forward or backward is more efficient...

  ➔ Can we use both of them together?



Or an approximate approach...

- Start with a set of states S that is a superset of init states



- If (FixPoint(S) ∧¬p = ∅)
  ➔ (FixPoint(Init) ∧¬p = ∅) ➔ Property p is true

- Alternatively, we can start from a set of states T that is a super set of (¬p), and perform backward pre-image computation

Advanced techniques for sequential BDD

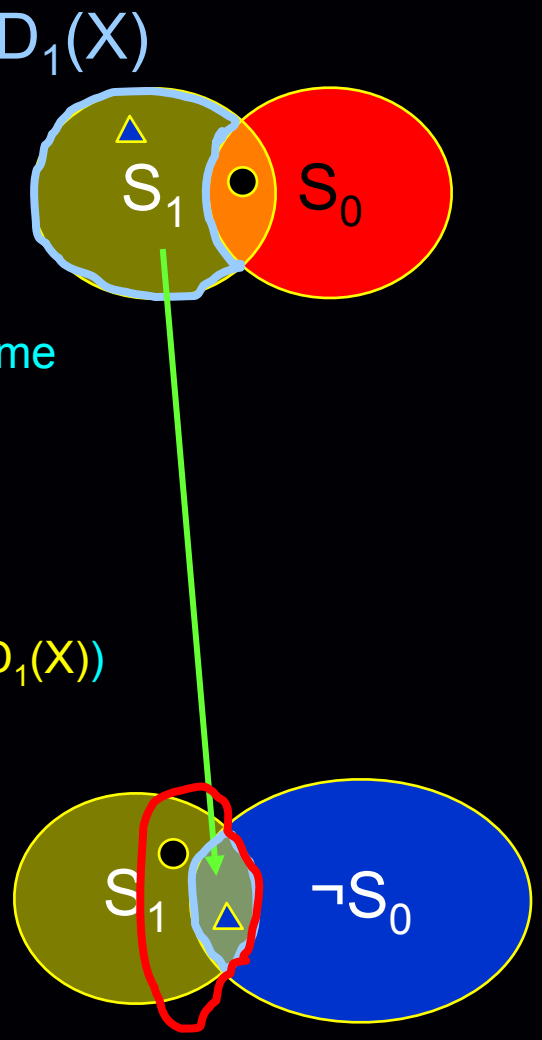
1. Backward pre-image computation
2. Speeding up the image computation
   - Frontier set simplification
   - Iterative squaring
3. Simplify the transition relation BDD
   - Disjunctive partition
   - Conjunctive partition
4. Approximate BDD techniques

Frontier set simplification

- In the image computation equation ---
  - $S_1(Y) = \exists X\ (\mathcal{TR}(Y, X) \wedge S_0(X))$
  - ➔ $S_1(Y)$ contains the newly reached states △, and some states in $S_0(Y)$ ○
- Let $D_1(Y) = S_1(Y) - S_0(Y) = S_1(Y) \wedge \neg S_0(Y)$
  - ➔ $D_1(Y)$ is the set of newly reached states in time 1
  - ➔ $S_2(Y)$ can be computed by $S_2(Y) = \exists X\ (\mathcal{TR}(Y, X) \wedge D_1(X))$
- A even better choice:

  Restrict (S_1, $\neg S_0(Y)$) ←(BDD smaller than D_1)
  - ➔ Guarantee to contain ($S_1(Y) \wedge \neg S_0(Y)$)

    and some part of $S_0(Y)$



In practice, this method usually results in insignificant increase in memory, and a significant factor decrease in time

Iterative squaring

- In some cases (e.g. counter), it may takes a large number of timeframes to reach the fix-point
- Let's consider:
  - $S_1(Y) = \exists X\ (\mathcal{TR}(Y, X) \wedge S_0(X))$
  - $S_2(Y) = \exists X\ (\mathcal{TR}(Y, X) \wedge S_1(X))$

  ➔ $S_2(Y) = \exists K\ (\mathcal{TR}(Y, K) \wedge \exists X\ (\mathcal{TR}(K, X) \wedge S_0(X))$

  $= \exists X\ (\exists K\ (\mathcal{TR}(Y, K) \wedge (\mathcal{TR}(K, X)) \wedge S_0(X))$
- Let: $F^2(Y, X) = \exists K\ (\mathcal{TR}(Y, K) \wedge (\mathcal{TR}(K, X))$

  ➔ $S_2(Y) = \exists X\ (F^2(Y, X) \wedge S_0(X))$
- Let: $F^4(Y, X) = \exists K\ (F^2(Y, K) \wedge F^2(K, X))$

  ➔ $S_6(Y) = \exists X\ (F^4(Y, X) \wedge S_2(X))$
- We can continue for (F^8, S_{14}), (F^{16}, S_{30}),... etc.
- Experience shows that "Iterative Squaring" is more efficient only for extremely simple examples like counter

Advanced techniques for sequential BDD

1. Backward pre-image computation
2. Speeding up the image computation
   - Frontier set simplification
   - Iterative squaring
3. Simplify the transition relation BDD
   - Disjunctive partition
   - Conjunctive partition
4. Approximate BDD techniques

Transition Relation BDD

- Let's look at the basic reachability analysis equation:
  - $S_1(Y) = \exists X\ (\mathcal{TR}(Y, X) \wedge S_0(X))$

  where ---
  - $\mathcal{TR}(Y, X) = \exists I\ (\prod(y_i = \delta_i(X, I)))$

    $= \exists I\ ((y_0 = \delta_0(X, I)) \wedge (y_1 = \delta_1(X, I)) \wedge \ldots))$

- $\mathcal{TR}(Y, X)$ is usually the most difficult part

  ➔ We may not be able to construct the BDD for it

Disjunctive partition

- If $\mathcal{TR}$ can be written in a disjunctive form ---

  ➔ $\mathcal{TR}(Y, X) = \exists I\ (D_1 \vee D_2 \vee ... \vee D_k)$

[Note] "Existential Quantification" can be distributed over disjunctions...

➔ $S_1(Y) = \exists X\ (\mathcal{TR}(Y, X) \wedge S_0(X))$

$= \exists X\ (\exists I\ (D_1 \vee D_2 \vee ... \vee D_k) \wedge S_0(X))$

$= \exists X, I\ (D_1 \wedge S_0(X)) \vee \exists X, I\ (D_2 \wedge S_0(X)) \vee ...$

➔ Each "$\exists X, I\ (D_i \wedge S_0(X))$" could be much smaller, and thus $\mathcal{TR}$ can be much easier computed.

- But the problem is:

  It is usually not trivial to represent $\mathcal{TR}$ in a disjunctive form...

Conjunctive Partitioned Transition Relations

- $\mathcal{TR}(Y, X) = \exists I\ (\prod(y_i = \delta_i(X, I)))$

  $= \exists I\ (C_1(Y, X, I) \wedge ... \wedge C_n(Y, X, I))$

[Note] "Existential Quantification" CANNOT be distributed over conjunctions...

➔ $S_1(Y) = \exists X\ (\mathcal{TR}(Y, X) \wedge S_0(X))$

$= \exists X, I\ (C_1(Y, X, I) \wedge ... \wedge C_n(Y, X, I) \wedge S_0(X))$

➔ Cannot be easily simplified as "disjunctive form" does

(What can we do???)

Early Quantification for Conjunctive Partition

[Observations]

- Circuits exhibit locality
  - Most of the "$C_i(Y, X, I)$" may only depend on a small portion of X variables
- If some $C_j(Y, X, I)$ does NOT depend on a variable x_k

  ➔ We can do:

  $\exists_{X - \{x_k\}} (\exists x_k(\text{all C's except Cj})) \wedge C_j(Y, X, I)$

- In general, we can rearrange and group C_i's into G_1, G_2,..., G_k, such that ---

  ➔ $\exists x_1 (\exists x_2 ...(\exists x_k\ G_k\ ) \wedge G_{k-1}) \wedge ...) \wedge G_2) \wedge G_1$

  (G_{k-1} does not depend on x_k)

03/26

Advanced techniques for sequential BDD

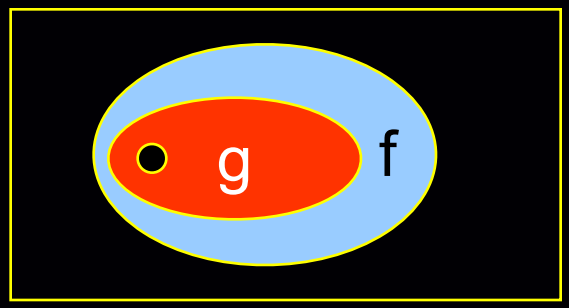
1. Backward pre-image computation
2. Speeding up the image computation
   - Frontier set simplification
   - Iterative squaring
3. Simplify the transition relation BDD
   - Disjunctive partition
   - Conjunctive partition
4. Approximate BDD techniques

Exact vs. Approximate Approaches

- (BDD) The exact approach usually suffers from memory explosion problem for real designs
  - What do you get if the TR cannot be built?

- Given a function f, the approximate approach tries to derive another function g such that
  1. "Close" enough to the original f (usually in terms of Hamming distance)
  2. BDD(g) < BDD(f) // BDD size

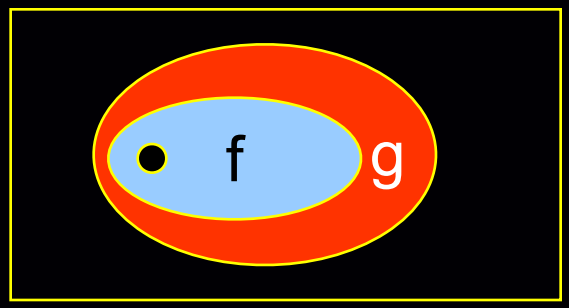
Under Approximation



- Ref: Ravi DAC 98
- Construct g that contains less minterms than f, while maximize the BDD density function

  → minterm(g) / BDD(g) > minterm(f) / BDD(f)
- Replace a node v in f by ---
  1. One of its children
  2. Shared grandchild
  3. Constant '0'
- If ¬p is reached → a real bug
  If ¬p cannot be reached → may be false positive

Over Approximation



- Ref: Cho TCAD 96, Dill DAC 99
- Function g contains more minterms than f
  1. Replace some arcs in BDD with '1'
  2. Partition state variables and decompose into several BDDs (sometimes with auxiliary variables)
  3. Abstraction --- removing part of the netlist away
- If ¬p is reached → may be false negative

  If ¬p cannot be reached → property is proven

Conclusions on BDD... so far...

- BDD is a compact and canonical data structure for function, set, and relation representations
- Combinational assertion checking is straightforward
  - Tautology checking
  - Dynamic variable ordering, local cut, various DDs, etc
- Sequential assertion checking = state reachability analysis
  - Forward vs. backward
  - Need to avoid memory explosion problem

BDD and its applications have been among the most researched EDA areas in 90's

- ➔ Had great impact on every EDA fields
- ➔ Became a must-understand knowledge for EDA researchers

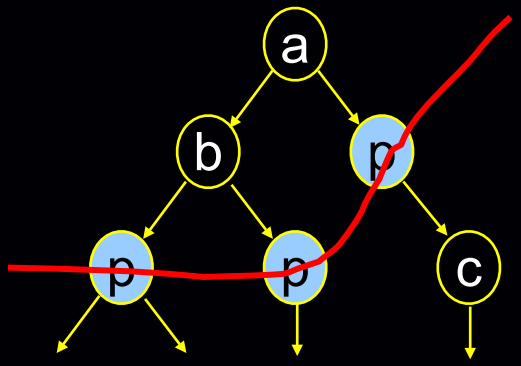
However, the power of exponential growth is formidable

- ➔ Design size: exponential growth (Moore's Law)
- ➔ BDD size: exponential to the design size
- ➔ But BDD is still much more efficient than most of the data structures for functional representation

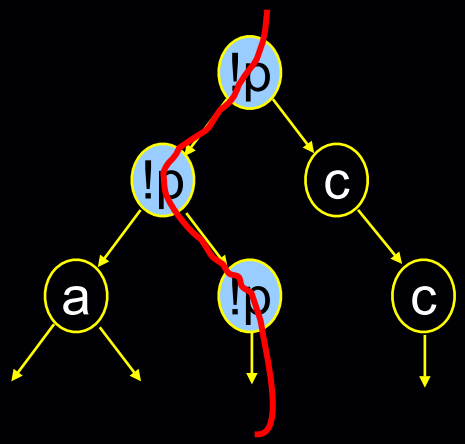
How to use it at the right time
for the right application is the key.

Appendix I: Greatest Fixed-Point Computation

- We know how to prove AG(p)
  - ➔ Find a solution for EF(¬p) as a counter-example
  - ➔ Reachability analysis (least fixed point computation)
- Another important type of property is “Eventuality” AF(p)
  - ➔ How to find its counter-example EG(¬p) ?



AFp

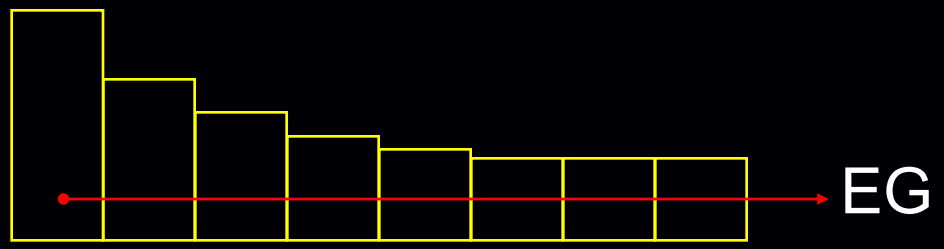


EG ¬p

➔ Need to find a loop in STG !! (How?)

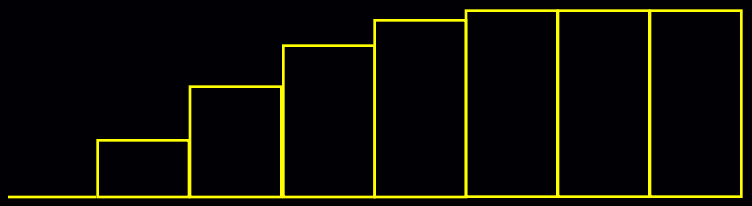
Greatest Fixed-Point Theorem

```
greatestFixedPoint()
{
    R = True;
    R′ = δ(R); // i.e. δ(True)
    while (R != R′) {
        R = R′
        R′ = δ(R) ∧ R;
    }
    return R;
}
```



Least Fixed-Point Theorem

```
leastFixedPoint() // reachability
{
    R = I0;
    R′ = δ(R);
    while (R != R′) {
        R = R′
        R′ = δ(R) + R;
    }
    return R;
}
```

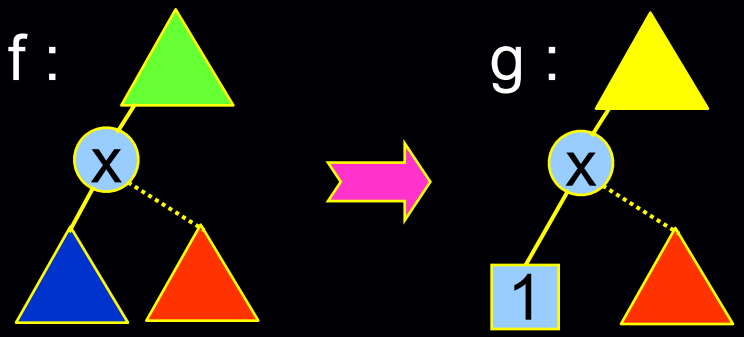


monotonic increasing

Appendix II: Decompose BDD

- Motivation
  - If BDD is too large, can we decompose it into several conjunctive or disjunctive BDDs
  - e.g. $f = f_1 \cdot f_2 \cdot f_3 \cdot ...$
- Decomposed by BDD variables
  - Given a variable x, decompose f into g · h, where

    $g = x + f_{\bar{x}}$ $h = \bar{x} + f_x$



  - We can decompose with a set of variables
- How to guarantee the results BDDs are smaller?

Appendix III: Extracting counter-example

- During the sequential BDD verification, after you found a bug, how do you extract the counter-example?
- Recall: the proof process
  - $R_0(X) = S_0(X)$
  - $S_{n+1}(Y) = \exists X\ (TR(Y, X) \wedge S_n(X))$
  - $S_{n+1}(X) = S_{n+1}(Y) \mid_{Y \rightarrow X}$
  - $R_{n+1}(X) = R_n(X) \vee S_{n+1}(X)$
  - If ($S_{n+1}(X) \wedge !P_{n+1}(X, I)$!= null), a bug is found!
  - If $R_{n+1} = R_n$, fixed point!!

  → What is missing to derive the counter-example?

Extracting counter-example { V_i }

- What is "$S_{n+1}(X) \wedge !P_{n+1}(X, I)$" ?
  - How to derive input vector $V_{n+1}(I)$?
  - $V_{n+1}(I)$ =
  - What is $S_{n+1}(X)$? A set of states or a state?
  - How to get a state $s_{n+1}(X)$ for which when $V_{n+1}(I)$ is applied, !P (the bug) can be reached?
- How about $V_n(I)$?
  - What is missing in "$S_{n+1}(Y) = \exists X\ (TR(Y, X) \wedge S_n(X))$" ?
  - $V_n(I)$ =
- Always converged to $S_0(X)$?

References

1. (Original BDD paper) R. E. Bryant, "Graph-based algorithms for Boolean function manipulation", TCOMP 1986.
2. (BDD implementation classic) K. S. Brace, R. L. Rudell, and R. E. Bryant. "Efficient implementation of a BDD package", DAC 1990.
3. (An exemplar BDD package) F. Somenzi, "CUDD: CU Decision Diagram Package", http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html
4. (Sifting algorithm) R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams", ICCAD 1993

References

5. (Various DDs rerview) R. Stankovic and T. Sasao, "Decision Diagrams for Discrete Functions: Classification and United Interpretation", ASPDAC 1998
6. (ZDD) S. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems", DAC 1993
7. (OFDD) U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel logic synthesis based on functional decision diagrams", EDAC 1992
8. (OKFDD) R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski, "Efficient Representation and Manipulation of Switching Functions Based on Ordered Kronecker Functional Decision Diagrams", DAC 1994
9. (BMD) R. Bryant and Y. Chen, "Verification of Arithmetic Circuits with Binary Moment Diagrams", DAC 1995

References

10. (Reachability analysis classic) O. Coudert and J. C. Madre, "A unified framework for the formal verification of sequential circuits", ICCAD 1990
11. (A comprehensive paper for sequential verification) J. Burch, E. Clarke, D. Long, K. McMillan, and D. Dill, "Symbolic Model Checking for Sequential Circuit Verification", TCAD 1994
12. (Advanced TR) G. Cabodi, P. Camurati, L. Lavagno, and S. Quer, "Disjunctive partitionining and partial iterative squaring: An effective approach for symbolic traversal of large circuits", DAC 1997
13. (Partition TR summary) I. Moon, J. Kukula, K. Ravi, and F. Somenzi, "To Split or to Conjoin: The Question in Image Computation", DAC 2000

References

14. (Under approx) K. Ravi, K. McMillan, T. Shiple, and F. Somenzi, “Approximation and Decomposition of Binary Decision Diagram”, DAC 1998
15. (Over approx) S. Govindaraju, D. Dill and J. Bergmann, “Improved Approximate Reachability using Auxiliary State Variables”, DAC 1999
16. (Over approx) H. Cho, G. D. Hachtel, E. Macii, M. Poncino, and F. Somenzi, “Automatic state space decomposition for approximate FSM traversal based on circuit analysis”, TCAD 1996.