Intro. to Formal Verification

Prof. Chien-Nan Liu
Institute of Electronics
National Chiao-Tung Univ.

Tel: (03)5712121 ext:31211
E-mail: jimmyliu@nctu.edu.tw
http://www.ee.ncu.edu.tw/~jimmy

Courtesy: Prof. Jing-Yang Jou

Outline

- Formal Verification Overview
- Equivalence Checking
  - Combinational equivalence checking
  - Sequential equivalence checking
- Model Checking

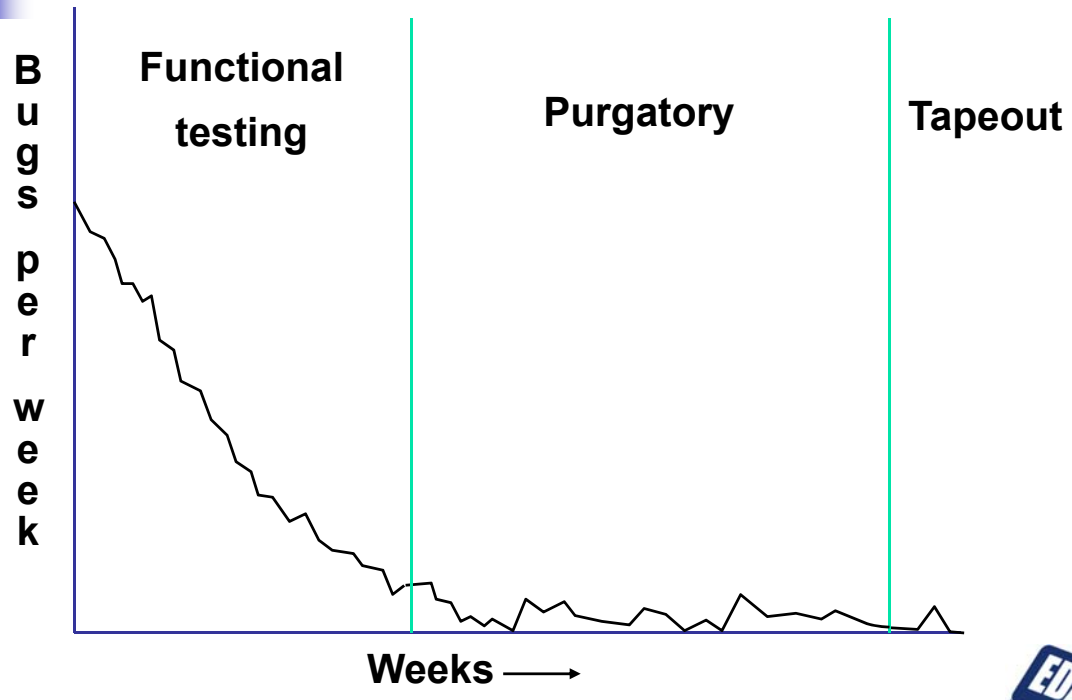
Specification V.S. Verification

- Specification: describe the behavior (property) of the system or circuits
- Verification: verify the system (circuit) against the specification
- Milestones of formal verification:
  - Software: begin around 1960
  - Hardware: late 1980
  - Hardware/software co-verification: ???

Current Design Practices

- Engineers write "reactive testbenches" in HDL
- Input generation 通常都是透過寫input pattern來進行驗證，只是coverage並不能被保證
  - Manual (verification engineers think of test cases)
  - Pseudo-random
  - Mixed (some random parameters)
- *These methods cannot get enough "coverage" to find all the bugs*

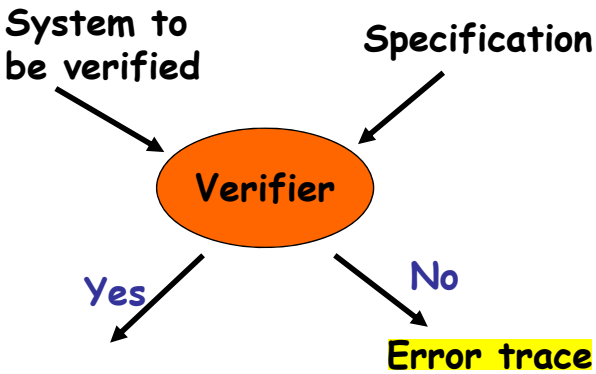
Typical Verification Experience



Formal Verification

使用理論"證明"design的function是對的

- Ensures consistency with specification for *all* possible inputs (100% coverage)
- Methods
  - Equivalence checking
  - Model checking
  - …



在驗證出錯誤時要dump出相關資訊

Valuable, but not a general solution

Simulation v.s. Formal Verification

- Simulation:
  - Exhaustive simulation is not possible
  - Increasingly difficult to handle the subtle interactions between separated systems
- Formal verification:
  - Design Verification:
    - Model checking: Deadlock, Mutual Exclusion, etc.
  - Implementation Verification: 給予兩個design，檢查是否功能相同
    - Equivalence checking: Ensure a correct translation from the specification to the implementation

ex: RTL轉gate level、加入gated clock後都是用equivalence checking的時機

風險: equivalence checking只能用來確認轉換是否正確，如果原本的deisgn就有錯誤，則該種驗證無法察覺

Limitations of Verification Methods

無法用在大型design，記憶體吃不下

- Simulation
  - CPU intensive
    - Have to run billions of cycles
  - Can handle large systems
- Formal verification
  - Memory intensive
    - Internal data structures (BDDs)
  - Memory usage is strongly related with the size of systems to be verified

Outline

- Formal Verification Overview
- Equivalence Checking
  - Combinational equivalence checking
  - Sequential equivalence checking
- Model Checking

Equivalence Checking

- Checks for mismatches between
  - Two gate-level circuits
  - HDL and gate-level designs
- *"Formal"*, because it checks for *all* input values (solves SAT problem)
- Gaining acceptance in practice
- **Limitation**: targets *implementation errors*, not *design errors*
  - Similar to check C v.s. assembly language

Example: Equivalence Checking

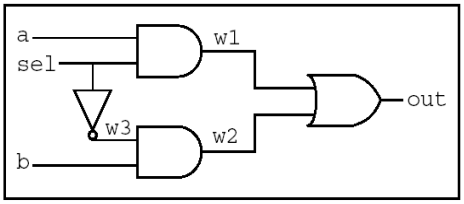
```
out = sel ? a : b ;
```

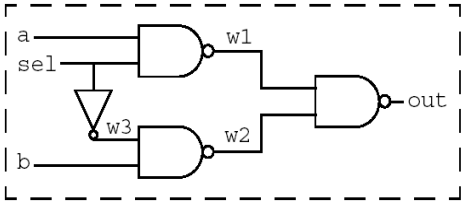
```
always @ (sell or a or b)
    if (sel) out = a;
    else out = b;
```

雖然RTL code看起來不一樣，gate level看起來也不一樣，但是他們功能卻是相同的，這就是equivalence checking難的地方

```
module des1 (out,a,sel,b)
    output out;
    input a,sel,b;
    wire w1,w2,w3,
      and u1(w1,a,sel)
      and u2(w2,w3,b);
      not u3(w3,sel);
      or u4(out,w1,w2)
endmodule
```

```
module des1 (out,a,sel,b)
    output out;
    input a,sel,b;
    wire w1,w2,w3,
      nand u1(w1,a,sel)
      nand u2(w2,w3,b);
      not u3(w3,sel);
      nand u4(out,w1,w2)
endmodule
```

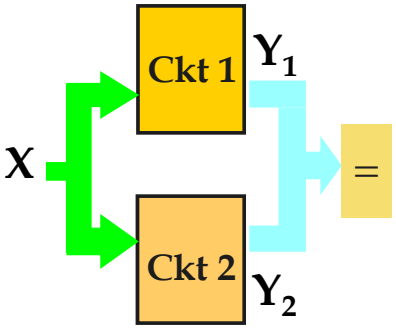




Equivalence Checking

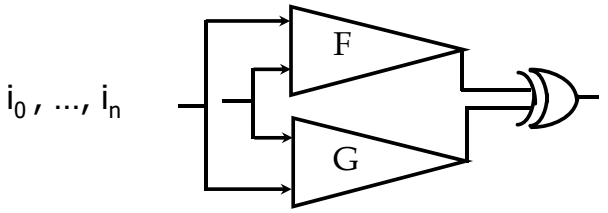
- Combinational Equivalence Checking
  - Outputs depend only on present inputs
- Sequential Equivalence Checking
  - Outputs depend on present inputs as well as past sequence of inputs

考慮combinational circuit Ckt1與Ckt2，只要送同樣的pattern進去，他們的產出一樣
--> equivalence



Combinational Equivalence Checking

- Problem formulation:
  - **Given:** two combinational Boolean netlists **F** , **G**
  - **Goal:** check if the corresponding outputs of the two circuits are equal for all possible inputs
- **e = F(i_0, …, i_n) ⊕ G(i_0, …, i_n)**
- **F is equivalent to G ⇔ e = 0 for all possible combination of input patterns**



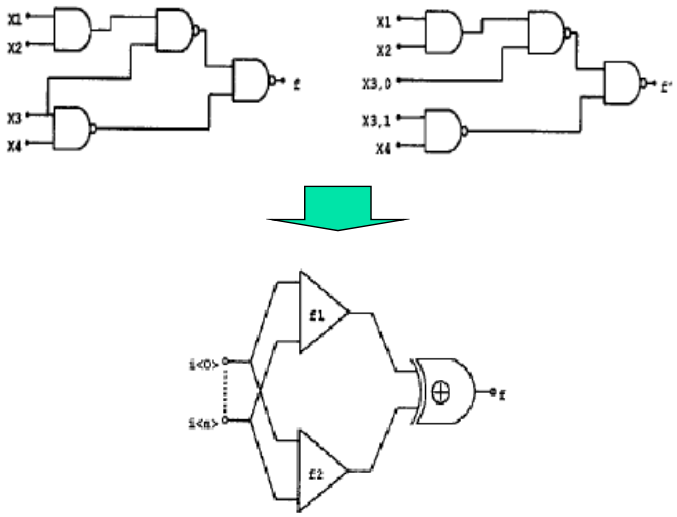
將input pattern送入兩個design後，對兩個design的output做XOR 如果全部都是0的話就代表equivalence

Approaches for Combinational Ckts

- Functional methods: Transform output functions into a canonical representation 將電路轉換為"唯一"的表達式
  - Based on BDDs 建立ROBDD，然後比對兩個BDD是否相等
  - Canonical BDD variants
- Structural methods:
  - Based on internal correspondence 抓出電路中一些容易確認的點
  - Learning techniques for identifying implications 運用推理證明這些點相等
  - Techniques for exploiting implications

Functional Methods

- Key idea: check the satisfiability of equation $f_1 \oplus f_2$ 解SAT



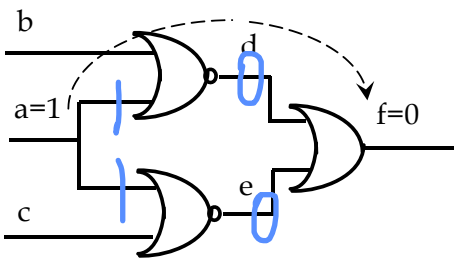
Functional Methods

- Given two circuits:
  - Build the ROBDDs of the outputs in terms of the primary inputs
  - Two circuits are equivalent if and only if the ROBDDs are isomorphic
- Complexity of verification depends on the size of ROBDDs
  - Compact in many cases

Structural Methods

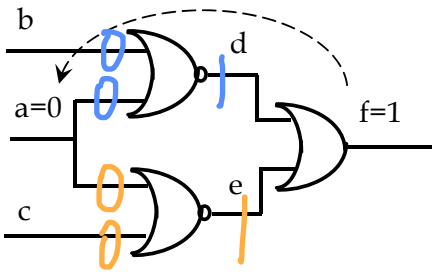
- Based on internal correspondences
  - Learning techniques for identifying implications
  - Techniques for exploiting implications
- Basic idea: 由初始的幾個internal equivalent points藉由推導得到更多的points
  - Two networks to be verified have many internal equivalent points and implications
  - Identify these equivalences and implications to simplify verification problem

Implications



Direct Implication

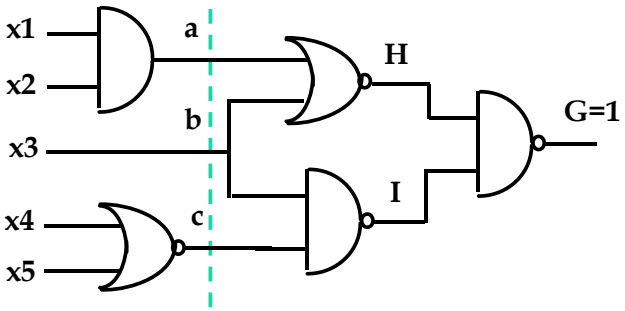
可以發現只要a=0，f必為0



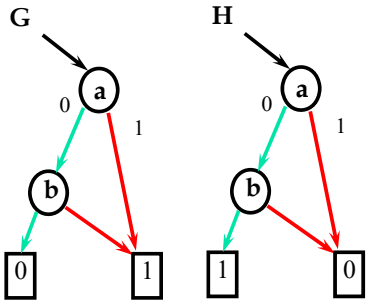
Indirect Implication (Learning)

逆推，可以發現若f=1，則d、e中其中一個必為1，代表(a,b),(a,c)其中一組必全為0 --> a必為0

Learning: Identifying implications



Functional Learning



$G = 1 \Rightarrow H = 0$

Recursive Learning

$G = 1$

Case 1: $H = 0$

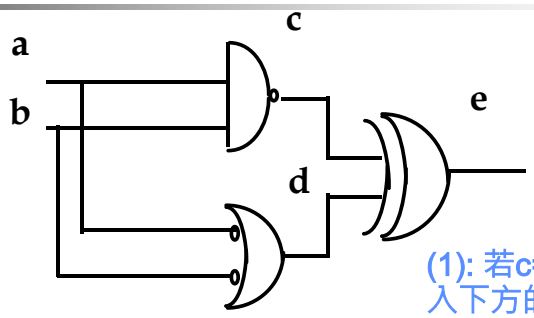
Case 2: $I = 0 \Rightarrow b = 1$

$\Rightarrow H = 0$

$G = 1 \Rightarrow H = 0$

由G=1逆推，可以發現不管是哪個case，H都為0

Learning for Verification



Learn: $c = 1 \Rightarrow e = 0$ (1)

$d = 1 \Rightarrow e = 0$ (2)

Output: $e = 1$

Case 1: c=0 and d=1

$\Rightarrow e = 0$ (from 2) *Conflict*

Case 2: c=1 and d=0

$\Rightarrow e = 0$ (from 1) *Conflict*

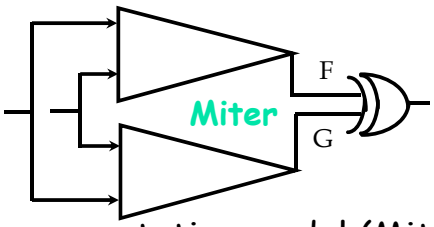
Conclusion: $e = 0$ i.e. *circuits are equal*

(1): 若c=1，代表a=b=0，將其代入下方的電路，得到d=1。因此c=d=1-->e=0

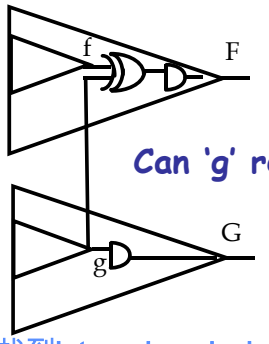
(2):d=1，代表a,b其中一個為0，因此c=1 --> e=0

再利用剛剛learning所得的(1), (2)證明e必為0

Implications for Verification

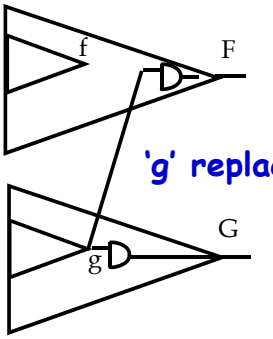


The computation model (Miter)



Can 'g' replace 'f' ?

找到internal equivalence point也可以縮小需要驗證的電路(切割電路)
-->將全部points接到其中一個equivalence point上，其他point的電路就不用理他

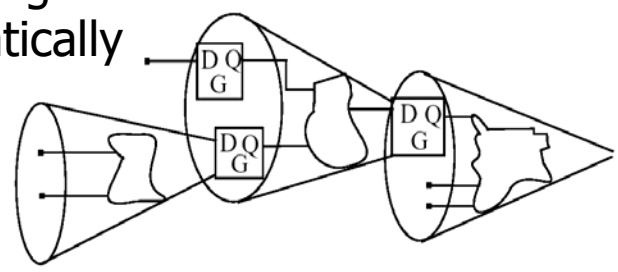


'g' replaces 'f'

Key idea: using internal equivalent points

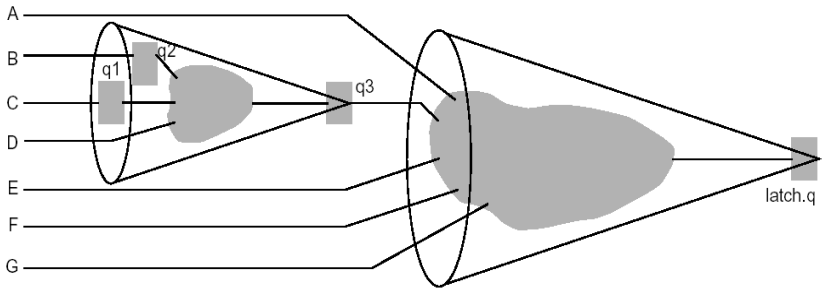
Compare (Key) Points

- A design object used as a combinational logic endpoint during verification
- FEC tools verify a compare point by comparing the **logic cone** of two matching points 只會比較會影響當前point的電路
- FEC Tools use the following design objects to automatically create compare points:
  - Primary outputs
  - Sequential elements
  - Black box input pins
  - Nets driven by multiple drivers, where at least one driver is a port or black box



Logic Cones

- A logic cone consists of all logic that funnels down to, and drives, a key point
- A logic cone can have any number of inputs, but only one output



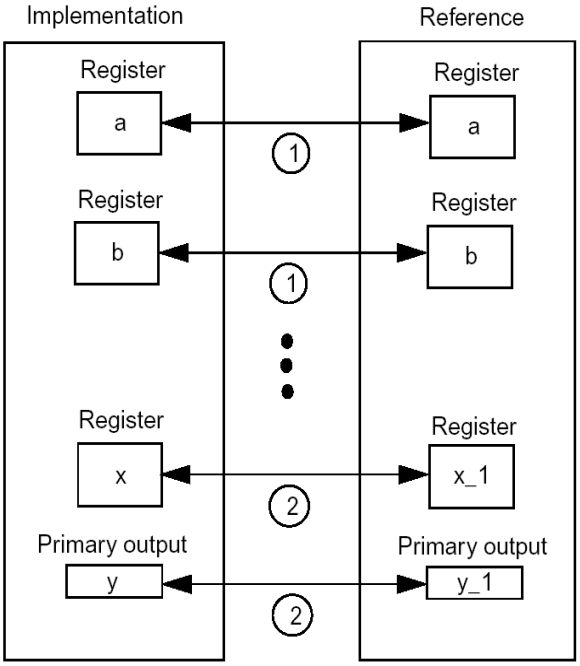
q3 = f (q1, q2, D)

latch.q = f (q3, A, E, F, G)

Constructing Compare Points

① Automatically defined compare points

② User-defined compare points



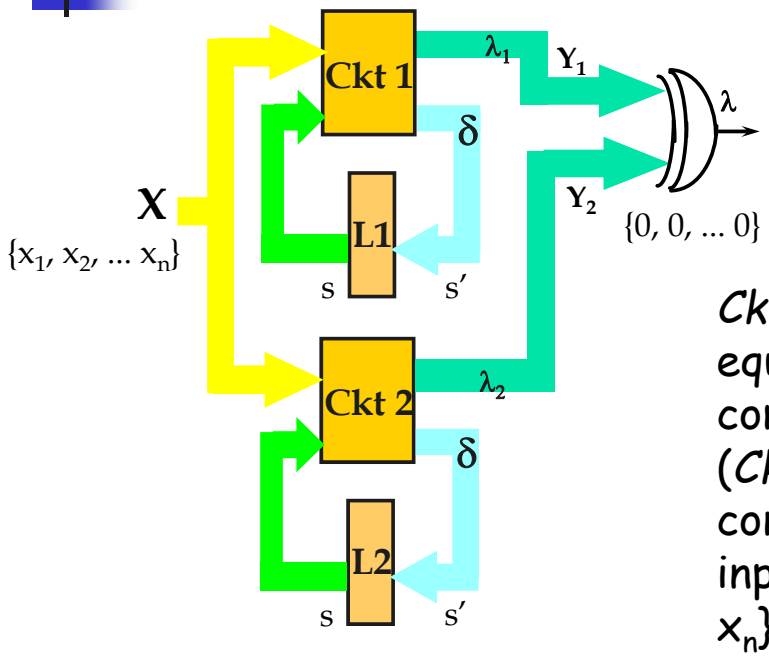
Summary

- Two basic approaches:
  - BDD based
    - General but suffer from memory explosion problem
  - Learning based
    - Require structural similarity
    - Fast but not as general 不一定能夠找到所有的internal equivalence point
- Recently attempt to consolidate different approaches in a single environment
  - Identify equivalent nodes
  - If not enough equivalences then use implications
  - Finally BDD based approach such as partitioning

Outline

- Formal Verification Overview
- Equivalence Checking
  - Combinational equivalence checking
  - Sequential equivalence checking
- Model Checking

Sequential Equivalence Checking



驗證sequential circuit時需要將其所有state的combinational 去做equivalence checking
--> 複雜度為C * S
C為檢驗combinational的時間
S為state數量

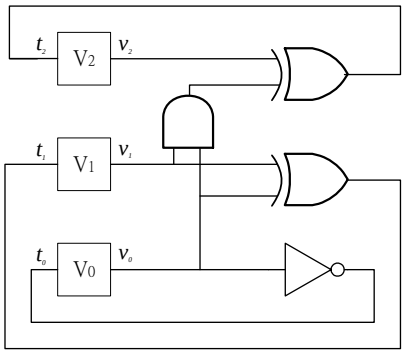
Ckt1 and *Ckt2* are equivalent iff the computational model (*Ckt1* ⊕ *Ckt2*) produces constant 0 for all valid input sequences {x_1, x_2, ... x_n}

FSM Model

- Finite state machine: FSM **(i, x, y, o, t, f, x^0)**
  - **i**: set of **input** variables
  - **x**: set of **current state** variables
  - **y**: set of **next state** variables
  - **o**: set of **output** variables
  - **t**: **state transition** functions: ***y = t(x, i)***
  - **f**: **output transition** functions: ***o = f(x, i)***
  - **x^0**: set of **initial states**

Represent a FSM using BDDs

- Representing a FSM includes:
  - the current/next states in the FSM
  - the transition/output relations



A modulo 8 counter

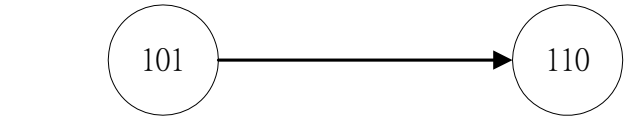
curState = v;
nextState = t;

$t_0 = !v_0$

$t_1 = v_0 \oplus v_1$

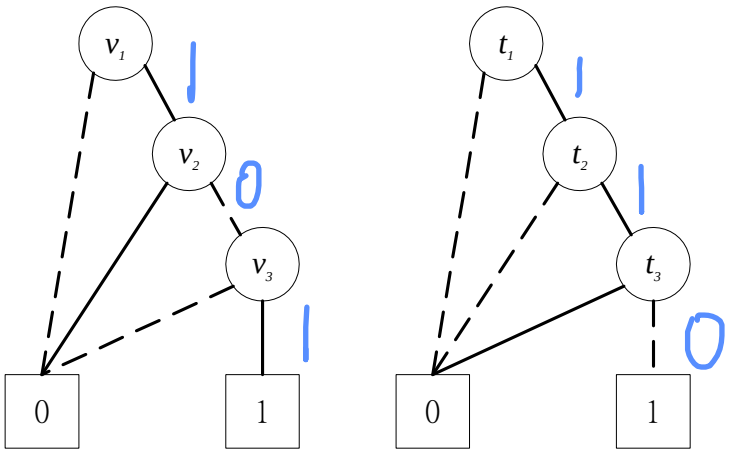
$t_2 = (v_0 \wedge v_1) \oplus v_2$

Represent Current/Next States



--> v1=1, v2=0, v3=1

--> v1=1, v2=1, v3=0



Current-State

Next-State

Characteristic Function

- Given a state transition function $y_k = t(x, i)$
- The *characteristic function* $z = \chi_{y_k}(y_k, x, i)$ of the function y_k is a **query** function:
  - $z = 1$ **(true)** means the values of y_k, x, i **satisfy** the equation: $y_k = t(x, i)$
  - $z = 0$ **(false)** otherwise

Ex: Consider OR gate

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

只要是合理的搭配組合，output就是1
(a,b,y) = (0,1,1) --> z=1
(a,b,y) = (0,0,1) --> z=0

- The equation of z is :

$$z = \chi_{y_k}(y_k, x, i) = (y_k \Leftrightarrow t(x, i)) = XNOR(y_k, t(x, i))$$

input的三個變數，只要有兩個已知，就可利用XOR求出剩餘的那個變數

State Transition Relation

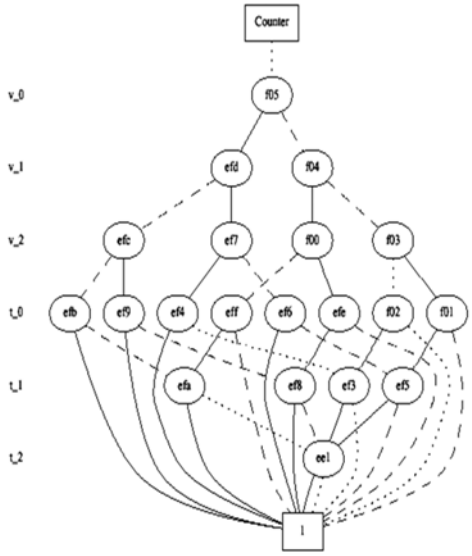
因為每個FF都要符合，因此將每個state transition function 作AND

- Given the state transition functions:
  $t(x, i) = [\ t_1(x, i), t_2(x, i), \ldots, t_m(x, i)\ ]$
  - $t_k(X, i)$ corresponds to one FF
- The state transition relation $T(x, i, y)$ of an FSM is defined as follows:

$$T(x,i,y) = \prod_{k=1}^{m} (\chi_{y_k}(y_k, x, i)) = \prod_{k=1}^{m} (y_k \Leftrightarrow t_k(x,i))$$

characteristic function of each state transition function

Represent Transition Relation



BDDs of State Transition Relation

$N_0(V, T) = (t_0 \Leftrightarrow !v_0)$

$N_1(V, T) = (t_1 \Leftrightarrow v_0 \oplus v_1)$

$N_2(V, T) = (t_2 \Leftrightarrow (v_0 \wedge v_1) \oplus v_2)$

$N(V, T) = N_0(V, T) \wedge N_1(V, T) \wedge N_2(V, T)$

Existential Quantification

- Given a state transition relation $T(x, i, y)$, the **existential quantification for x** is defined by the operator $\exists x$:

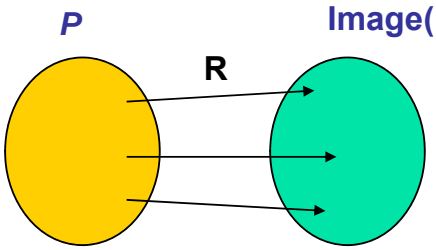
Shannon expansion

$$F = \exists x\, T(x, i, y) = T_{x=1}(i, y) + T_{x=0}(i, y)$$

- ***F is still a characteristic function***
  - ***F*** is true → there exists the assignments for x such that ***T*** is true

Forward and Reverse Images

- Forward image



一次對一整個set進行運算:
curState set 與 transition relation set
進行BDD AND
就可以得到nextState set

$$\text{Image}(P,R) = \{\mathbf{v}' : \text{ for some } \mathbf{v},\ \mathbf{v} \in P \text{ and } (\mathbf{v},\mathbf{v}') \in R\}$$

$$\chi_{\text{Image}(P,R)}(\mathbf{v}') = \exists \mathbf{v}.(\chi_P(\mathbf{v}) \wedge \chi_R(\mathbf{v},\mathbf{v}'))$$

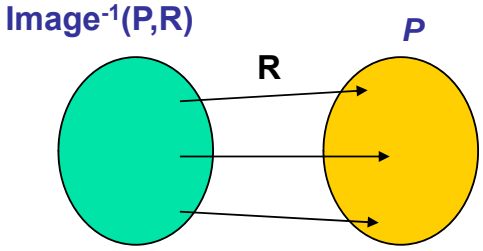
BDD for transition relation

BDD for next state

BDD for current state

Forward and Reverse Images

- Reverse image



將nextState set 與 transition
relation state 進行 BDD AND
就可以得到curState set

= *EX P*

$$\text{Image}^{-1}(P,R) = \{\mathbf{v} : \text{ for some } \mathbf{v}',\ \mathbf{v}' \in P \text{ and } (\mathbf{v},\mathbf{v}') \in R\}$$

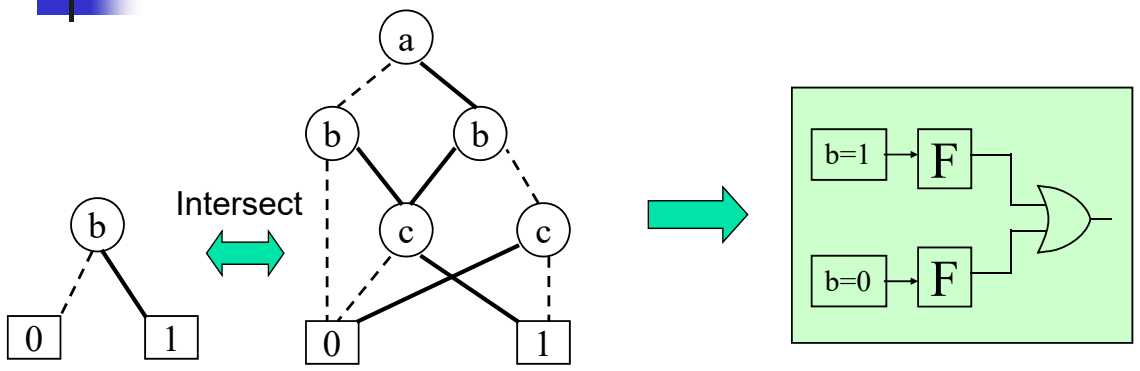
$$\chi_{\text{Image}(P,R)}(\mathbf{v}) = \exists \mathbf{v}'.(\chi_P(\mathbf{v}') \wedge \chi_R(\mathbf{v},\mathbf{v}'))$$

BDD for transition relation

BDD for current state

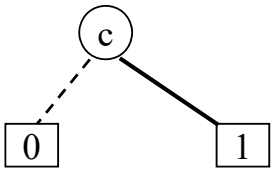
BDD for next state

Example: Image Computation



$\chi_C(b) = b$ $\quad f = ab'c' + abc + a'bc$

要將小tree代入右邊的大tree中，最後的結果就是兩棵BDD tree AND的結果



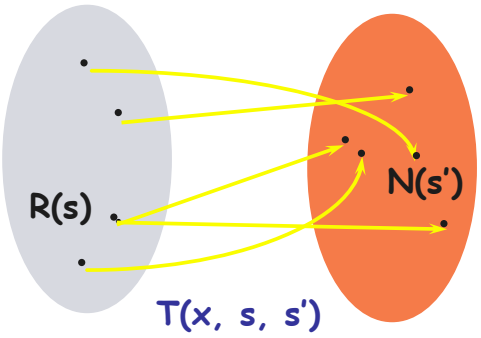
$\exists b\,[\,f\,(a, b, c) \wedge \chi_C(b)] = c$

Basic Approach of SEC

- Check if any state where the outputs are not equal is reachable from the initial state 只檢查reachable的state，其他不用理他
- Reachability analysis
  - To determine that a set of states can be reached from initial states in a system
  - Implemented by BDDs

Reachability Analysis

s: current state, s': next state, x: prime input
R(s): set of current state
N(s'): set of next state
T(x, s, s'): set of state transition relation



首先將curState代入state transition 方程式得到nextState，這些nextState就是我們reachable的state，因此再將它們加回curState中不斷重複，因此會像漣漪一樣不斷擴大，直到覆蓋所有reachable state (類似BFS)

N(s') can be obtained by the equation:

$$N(s') = \exists_{s,\,x} (T(x,\ s,\ s') \wedge R^i\ (s))$$

Algorithm of Reachability Analysis

Algorithm: do_reachability (I(s), T(x, s, s'))

```
i = -1
R^0(s) = I(s)
repeat
    i = i +1
    N(s') = ∃_{s,x} ( T(x, s, s') ∧ R^i (s) )
    N(s) = N(s' ← s)
    R^{i+1} (s) = R^i (s) + N(s)
until (R^{i+1} (s) = R^i (s))
return  (R^{i+1} (s))
```

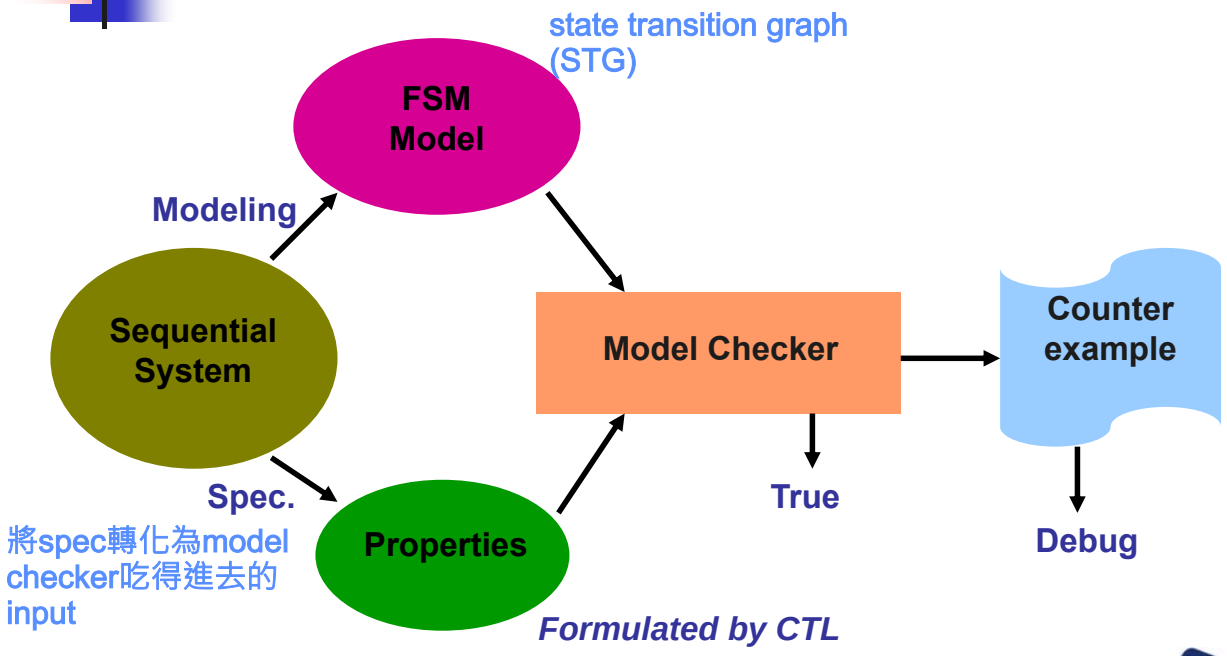
Debugging

- When any mismatch is found, a **counter example** that illustrates the difference will be generated
  - We can find the bugs through the example
- The counter example typically consists of
  - Comparison points that differ
  - Inputs of the logic cone that drive the comparison points
  - Intermediate nodes inside the logic cones
- Determine the exact location of the errors and how to correct them requires user's intervention

Outline

- Formal Verification Overview
- Equivalence Checking
  - Combinational equivalence checking
  - Sequential equivalence checking
- Model Checking

What is Model Checking ?

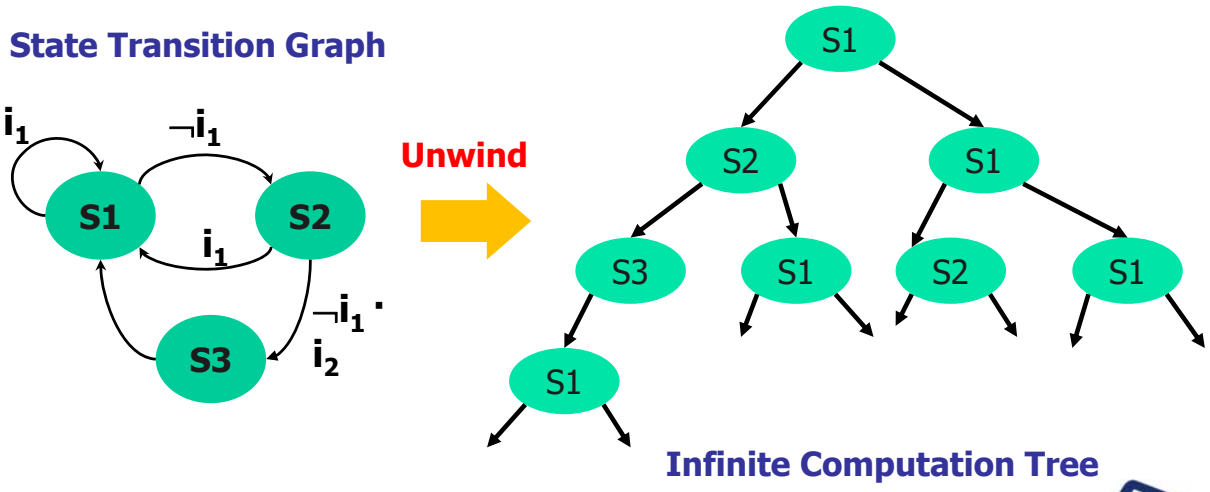


Specification & Temporal Logic

- Specification: describing the behaviors (**properties**) of the circuit
- Temporal logic: a formulism for **describing the temporal properties of a system**
  - Check whether the model satisfies those rules
- Features of temporal logic:
  - "**time**" is not mentioned explicitly 描述時間先後順序，而非確切時間
  - formulas might specify the concept of "**eventually**", "**never**", "**always**", or "**until**" for some designated states in the circuit

Computation Tree

- The computation tree shows all of the possible executions starting from the initial state of an FSM

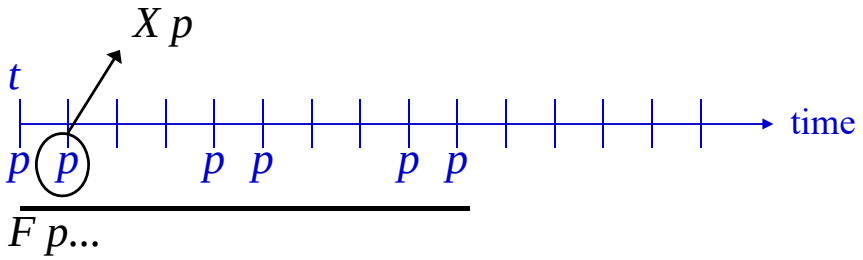


Computation Tree Logic (CTL)

- Formulas are constructed from ***logic operators***, ***temporal operators***, and ***path quantifiers*** :
  - Formal representations for the properties of a design
- Logic operator:
  - ¬ (not) , • (and) , + (or) , → (imply) , ↔ (if and only if)
- Temporal operator:
  - **Xp** — property **p** holds ***next time***
  - **Fp** — property **p** holds ***sometime in the future***
  - **Gp** — property **p** holds ***globally in the future***
  - **pUq** — property **p** holds ***until*** property q holds
- Path quantifier:
  - **A** — "***for every path***" in the computation tree
  - **E** — "***there exists a path***" in the computation tree

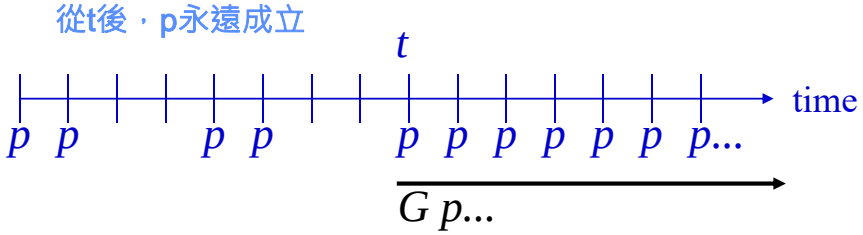
Temporal Operators

- **X**: "Next-time", $X\,p$ at t iff p at $t+1$
  代表下一個cycle一定會發生p
- **F**: "Future", $F\,p$ at t iff p for some $t' \geq t$
  在將來(包含當前)某個cycle一定會發生p

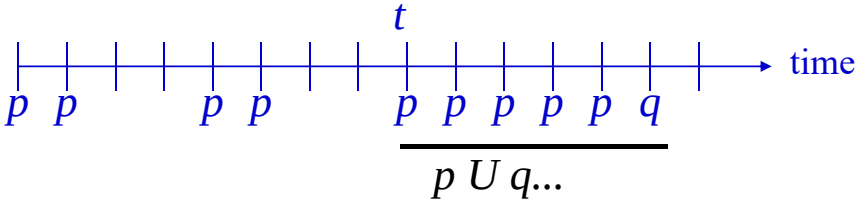


Temporal Operators

- **G**: "Globally", $G\,p$ at t iff p for all $t' \geq t$
  從t後，p永遠成立



- **U**: "Until", $p\,U\,q$ at t iff q for some $t' \geq t$ and p in the range $[\,t, t'\,)$
  q在 [t,t') 區間成立

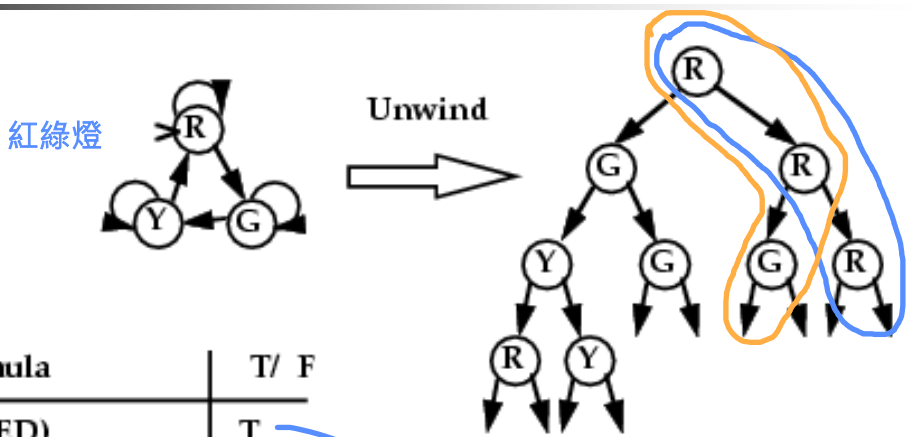


CTL Formula

path quantifier加在最前面

- Every operator *F, G, X, U* preceded by *A* or *E*
- Any CTL operator applied to a CTL formula gives another CTL formula
- Any Boolean combination of CTL formula is a CTL formula
- Propositions represented by small case alphabet (e.g. p, q, r)
- Examples of CTL formulas:
  - AG p
  - E (p U q)
  - AG EF p
  - AG (not(AX p) + EF q)

Branching View of Time



Formula	T/ F
E G (RED)	T
E (RED U GREEN)	T
AF (GREEN)	F
AGEF(GREEN)	T
A(RED U GREEN)	F

存在一條path永遠都是紅燈

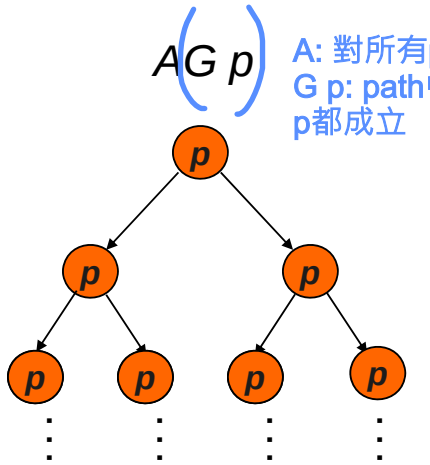
存在一條path在綠燈前一直都是紅燈

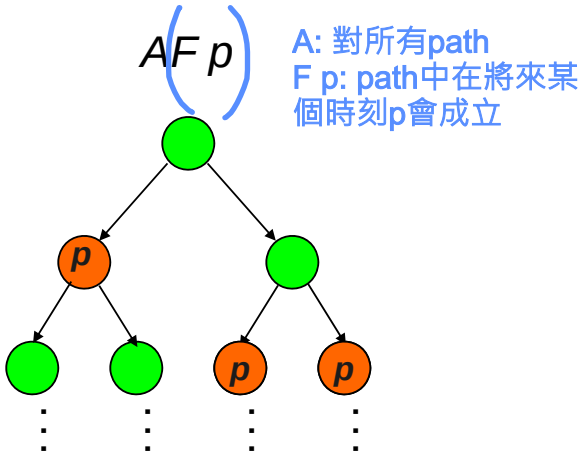
所有path在將來某個時刻皆會變為綠燈(與藍色衝突)

對所有的path (A) 中的每一個點 (G)，都存在一條path在將來會變成綠燈 (EF)

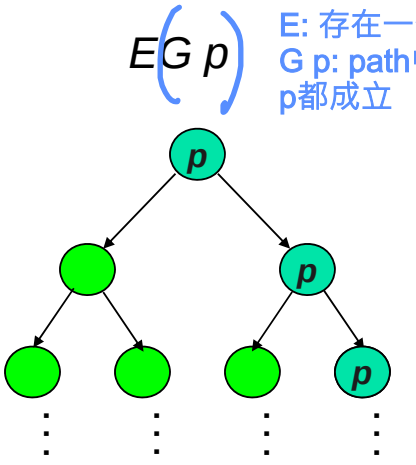
所有path中在綠燈前一直都是紅燈

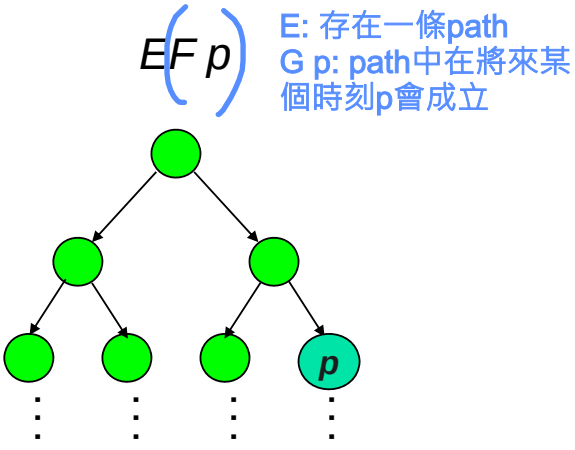
Computation Tree Logic



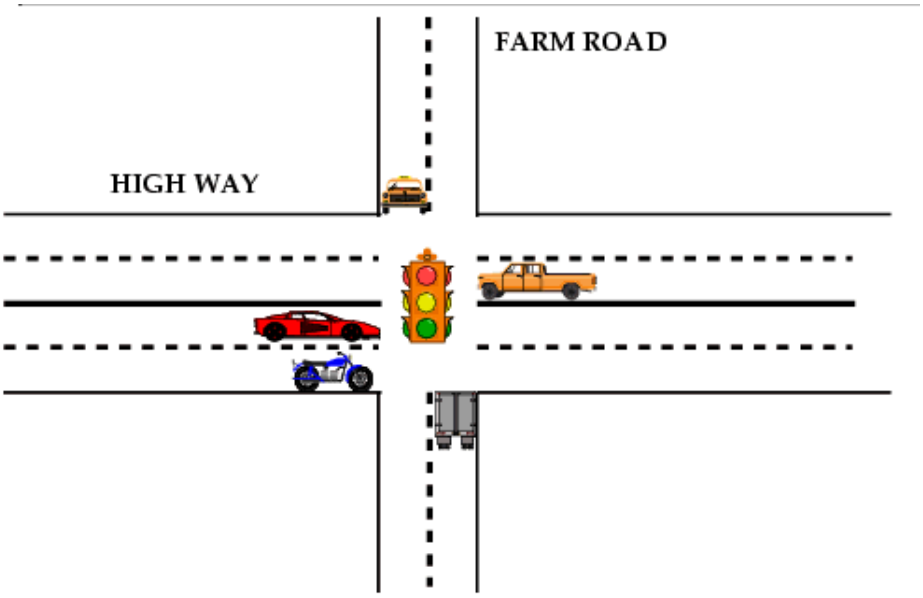


Computation Tree Logic

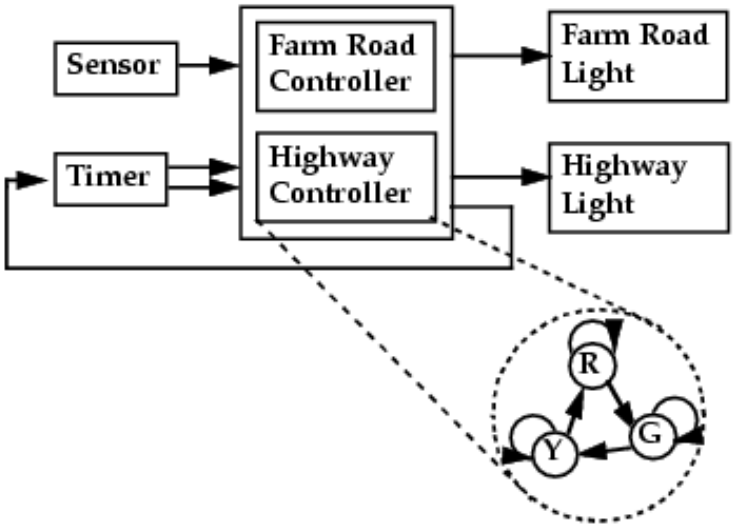




Example System: Traffic Light Controller



Block Diagram of TLC



Example Properties for TLC

- Invariant: It is never the case that both the highway and farm road have green simultaneously
- The CTL formula saying this is:

AG(!((hwy_light=green) * (farm_light=green)))

Example Properties for TLC

- If a car is waiting on the farm road, then eventually the farm road light turns green
- The CTL formula saying this is:

AG(car_waiting → AF(farm_light=green))

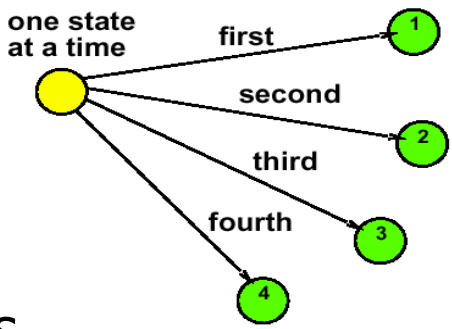
注意要補上這個AG，後面的敘述才會永遠成立

Symbolic Model Checking

- State explosion problem
  - State graph exponential in program size
- Symbolic model checking approach
  - Boolean formulas represent sets and relations
    - Often represented using BDD
  - Use **fixed point** characterizations of CTL operators
    - No more states can be reached from current state
  - Model checking without building state graph

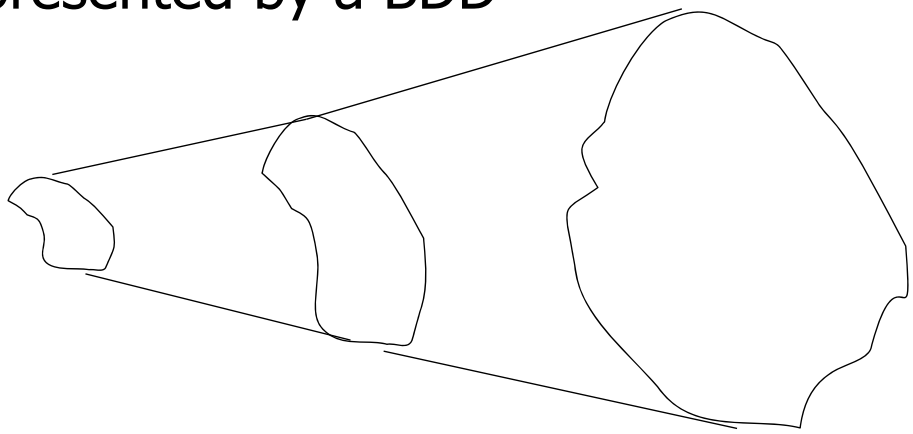
Explicit State Traversal

- For each state, its next state are enumerated one by one
- Process one state at a time
- The complexity depends on the number of states and the number of input combinations
  - Often a huge number for modern designs



Implicit State Traversal (1/2)

- Each layer of breadth-first search is represented by a BDD



- No explicit STG is built

Implicit State Traversal (2/2)

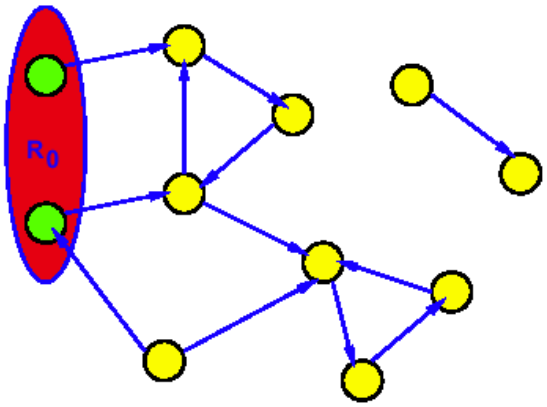
- Step 1: represent the set of states by **BDDs**
- Step 2: calculate the **set of next states *y*** from the set of current states ***x***
- Step 3: add the set of next states ***y*** to the **set of reachable states *R***
- Step 4: let the set of reachable states ***R*** be the set of current states, and repeat step 2 and step 3 ***until R is saturated***

Implicit State Traversal: Algorithm

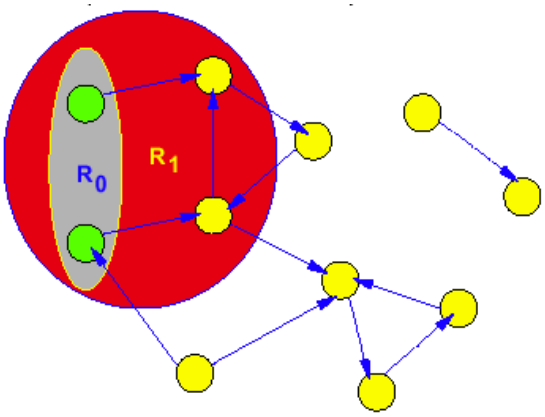
- **Input: set of initial states $\chi_x(x^0)$;**
  state transition relation $T(x, i, y)$;

```
1   k = -1
2   R_0(x) = χ_x(x^0);
3   do
4       k = k + 1;
5       χ_y(y) = ∃x,i ( R_k(x) • T(x, i, y) )      → Step 2
6       χ_x(x) = χ_y(x ← y)
7       R_{k+1}(x) = R_k(x) + χ_x(x)               → Step 3
8   while ( R_{k+1}(x) != R_k(x) )
9   return R_{k+1}(x)
```


Implicit State Traversal: Example

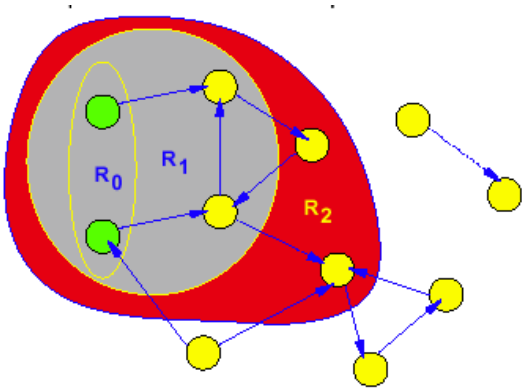


(1) R_0 is the set of initial states

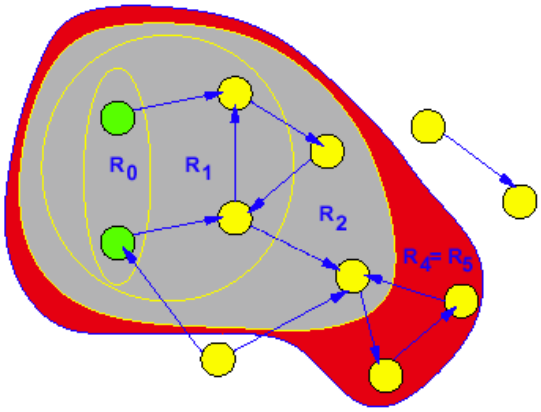


(2) R_1 is the set of states reachable from R_0 in less than or equal to one step

Implicit State Traversal: Example



(3) R_2 is the set of states reachable from R_0 in less than or equal to two step



(4) The iteration terminates after finding $R_5 = R_4$ and the resultant set is the set of reachable states

Acceptance of SMC

Acceptance:

- There have been major successes on some industrial projects
- Use on particular projects in huge companies (e.g. IBM, Intel)
- Commercially supported products
- But <1% use overall

Limitations of SMC

Limitations:

- State explosion problems limits to small submodules of hardware

  .... but interface is not specified

- Changing design may cause unpredictable blowup

Future Directions

- Only "**partial**" or "**bounded**" model checking

限縮檢查的範圍(BFS的n個step內才檢查)，
可以搭配simulation使用