Topic X

Formal Hardware Verification (III) Sequential Verification Techniques

系統晶片驗證

SoC Verification

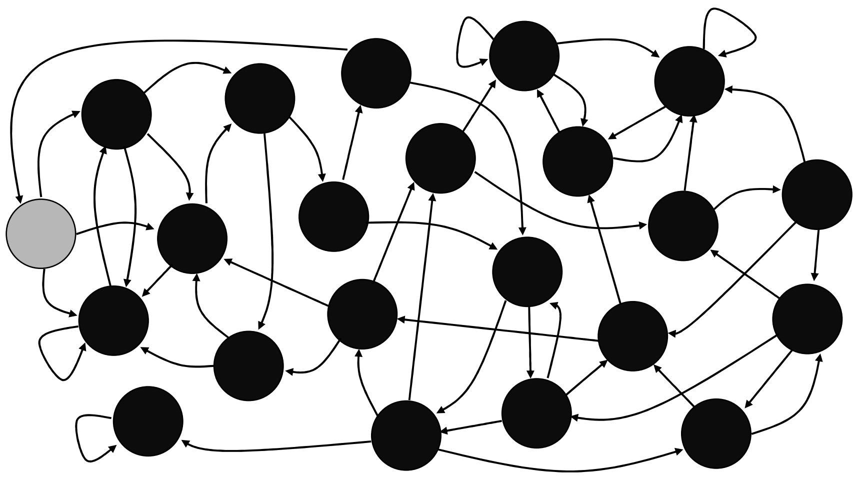
Sep, 2004

What we will cover in this topic ---

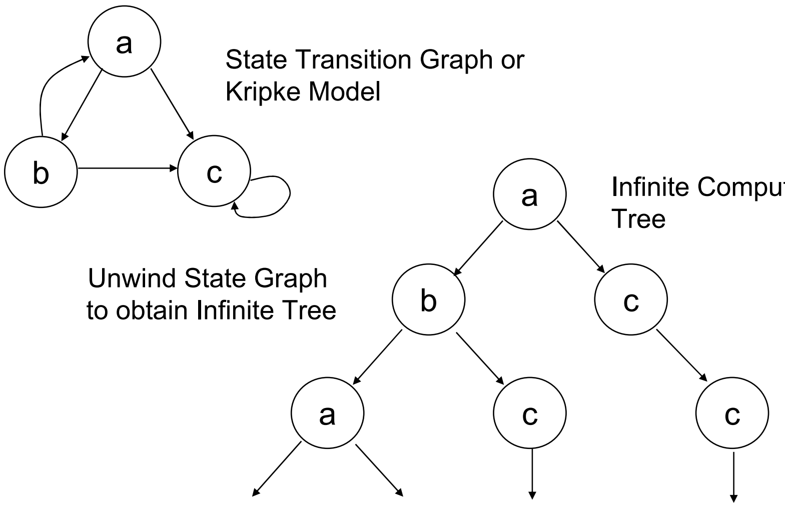
1. Introduction to temporal logic
   - CTL*, CTL, LTL
2. CTL Symbolic model checking using BDDs
   - Fixpoint Theorems
   - Basic CTL operators
   - Limitation of BDDs
3. Symbolic modeling checking using ATPG/SAT
   - Sequential circuit modeling
   - Bounded model checking
   - Unbounded model checking

Circuit Modeling

Finite State Machine (State Transition Graph)



Model of Computation



In general,

functional verification is dealing with "sequential constraint satisfaction problem"

Sequential constraint

= function (time, logic)

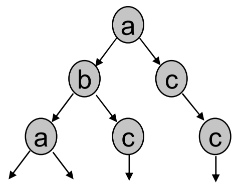
Temporal Logic

How to describe formula in computation tree

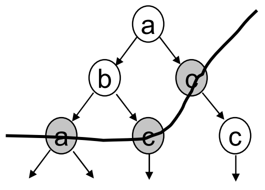
1. Path quantifier
   - A --- "for every path"
   - E --- "there exists a path"

2. Linear-time operators (State Quantifier)
   - Xp --- p holds next time
   - Fp --- p holds sometime in the future
   - Gp --- p holds globally in the future
   - pUq --- p holds until q holds

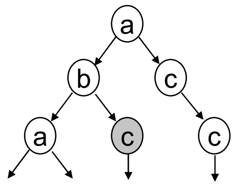
For example…



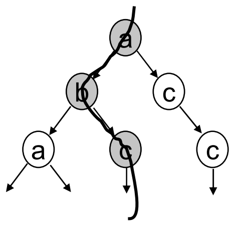
AGp



AFp



EFp



EGp

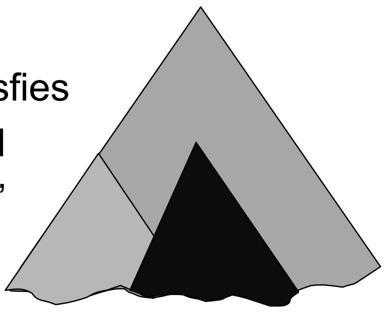
Recursive Definition

◆In an infinite computation tree, any sub-tree is also an infinite computation tree

◆Let Φ1, Φ2 be temporal formulae

➔ "Φ1 (Φ2)" means ---

"For any state that satisfies Φ1, its sub-tree should satisfy the formula Φ2"



More examples…

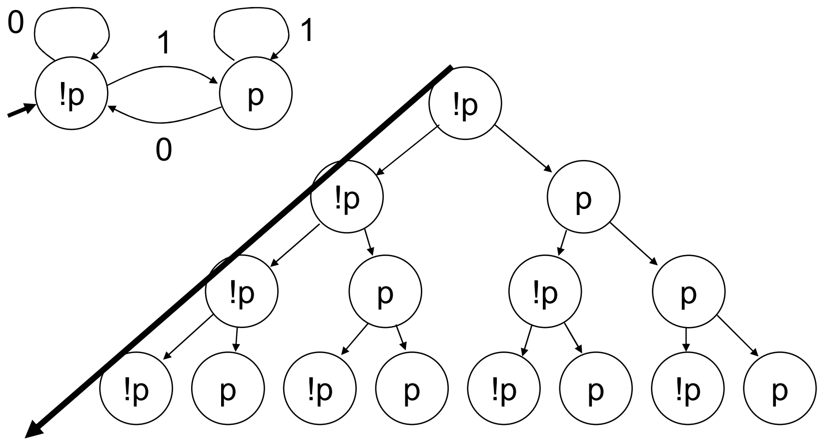
◆ AG(EF p)
- For any state in the computation tree,
  its sub-tree should at least contain a state that satisfies p
- e.g. AG(EF Restart) ➔ !deadlock
  - From any state it is possible to get to the Restart state

◆ AG(AF p)
- For any state in the computation tree,
  its sub-tree should have a "cut" that satisfies p
- e.g. AG(AF DeviceEnabled)
  - From any state, any of its future computation path must see a DeviceEnabled
  - DeviceEnabled holds infinitely often on every computation path

Is AG(EF p) the same as AG(AF p)??

No, a counter-example is…

◆Satisfies AG(EF p), but not AG(AF p)



Computation Tree Logic (CTL)

◆ A restricted subset of CTL* that permits only branching-time operators --- each of the linear-time operators G, F, X, and U must be immediately preceeded by a path quantifier

◆ Formula

<Path_quantifier> <linear-time operator> [(CTL formula)]

e.g. AG(p → EF q) ….. OK

e.g. AGF p ….. Not OK, no A/E between GF

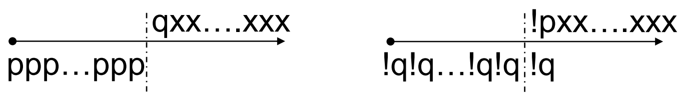
e.g. AG(p → E q) ….. Not OK, missing state quantifier

CTL Formula Reduction

◆ All CTL formulas can be expressed in terms of EX, EG, and EU

- AX p = ! EX (!p)
- AG p = ! EF (!p)
- AF p = ! EG (!p)
- EF p = E (true U p)
- A (p U q) = (! E (!q U (!p ^ !q)))
  & (! EG (!q))

2 types of counter-examples



We will talk about different types of temporal logic later…

Now, let’s see how we verify a CTL property

Model Checking Problem (for Functional Verification)

1. Let *M* be the state-transition graph obtained from the circuit modeling
2. Let *f* be the specification (property) expressed in temporal logic
3. Find all states s of *M*, such that

$$M, s \models f$$

CTL Model Checking by Explicit State Enumeration

◆Explicit state transition graph is required

◆The target formula is decomposed into smaller subformulas for ease of checking
  - e.g. (! AF !a ^ AF b)
    ➔ (AF !a), (AF b)

◆Starting from the initial state, evaluate the subformulas on each state it reaches

◆Continue until all the states are reached

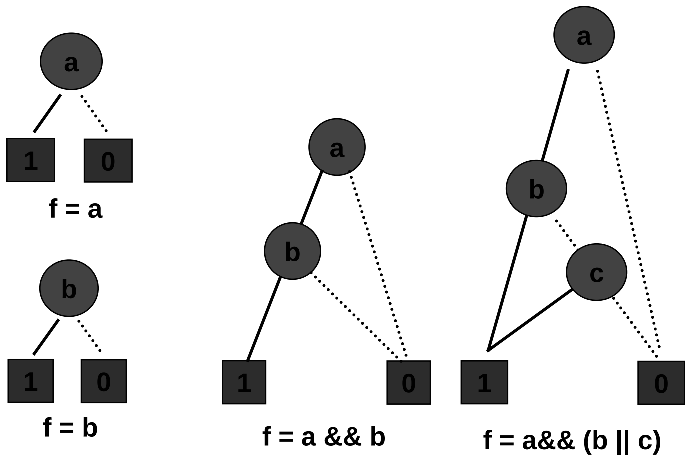
State Explosion Problem

Symbolic Modeling Checking

◆States and state graph are implicitly represented by certain compact data structure
  - e.g. Using BDDs

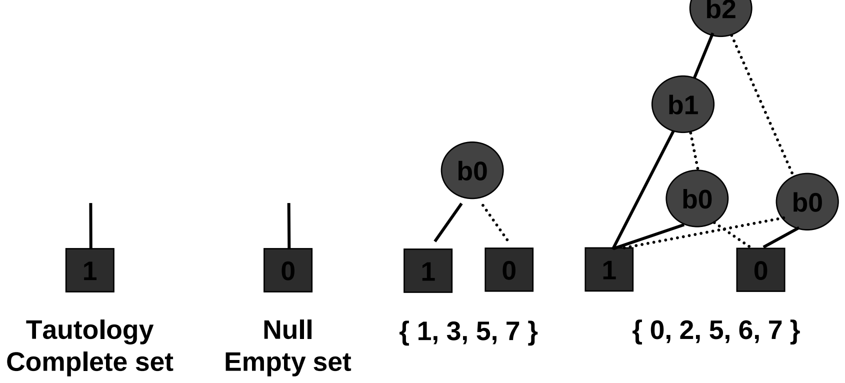
◆Temporal formulas are evaluated based on operations of the above data structure

Remember: BDD to Represent Logic Functions



BDD to Represent a Set

◆ e.g. Use 3-variable BDDs to represent any subset of the numbers in { 0, 1, 2, 3, 4, 5, 6, 7 }



Note: Don’t Use BDDs as Container

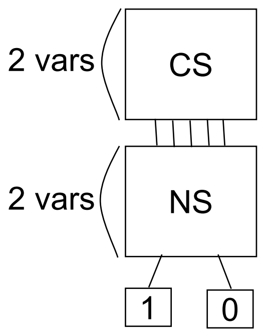
- When deposit “minterms” or “cubes” into a BDD ---
  - OK to query “membership”
  - NOT OK to retrieve the originally deposited minterms or cubes

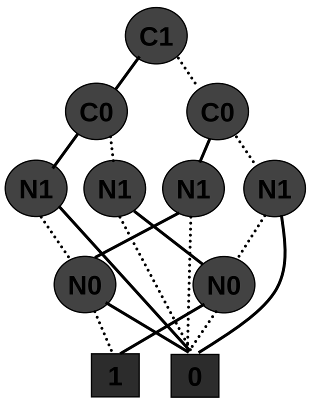
Why??

BDD to Represent Relations

- e.g. 2-bit ring counter

  R: { (0 → 1), (1 → 2), (2 → 3), (3 → 0) }

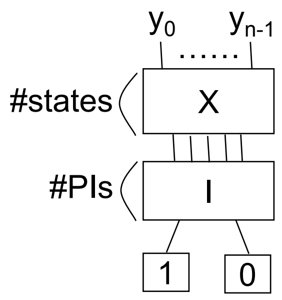


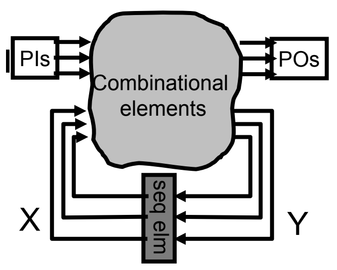


BDD to Represent State Transitions

◆ I : Set of PI variables
  X: Set of current state variables
  Y: Set of next state variables

◆ Transition Function: Y = T(X, I)
- For each state variable, $y_i = T_i(X, I)$

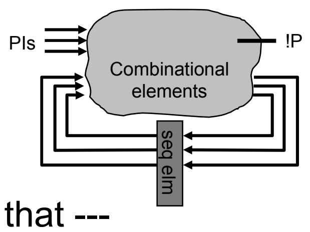




BDD to Prove Assertion Property

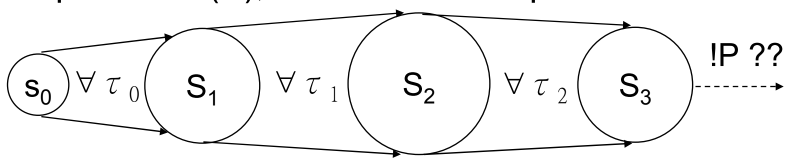
◆ Assert_always(P) = AG(P)

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

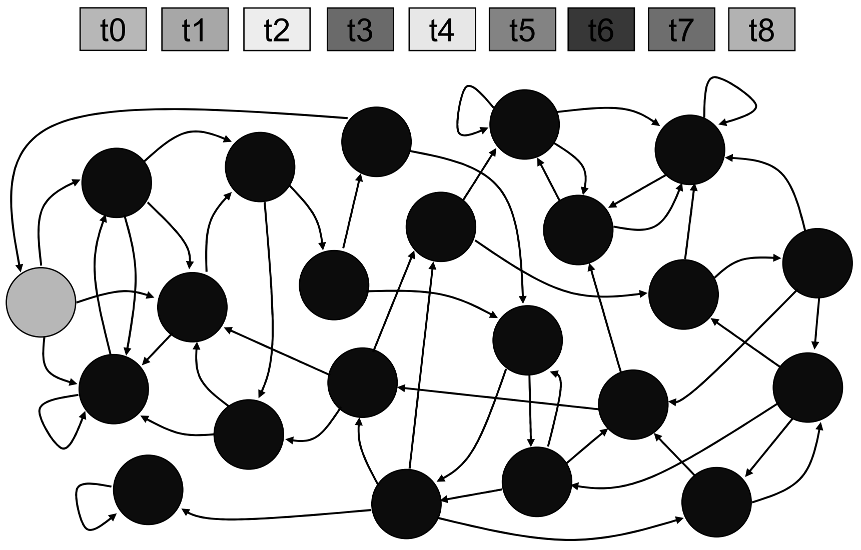


$$s_0 \xrightarrow{\tau_0} s_1 \xrightarrow{\tau_1} s_2 \xrightarrow{\tau_2} \cdots\cdots \rightarrow s_n \xrightarrow{\tau_n} !P$$

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

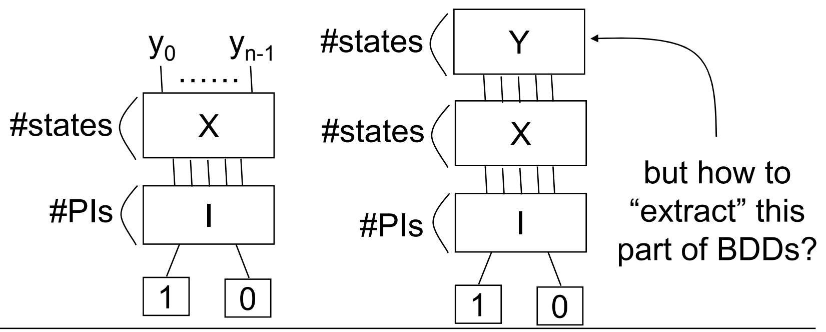


Set of Reachable States

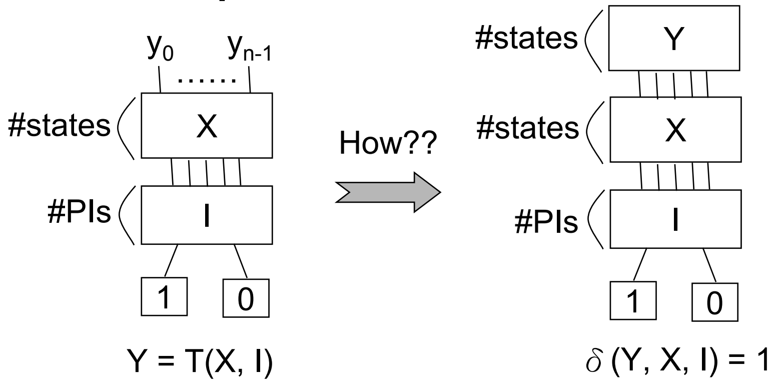


BDD to Compute Set of Reachable States

- ◆Transition Function: Y = T(X, I)
  - ● For each state variable, $y_i = T_i(X, I)$
  - ➔ Use BDDs to record reachable states (Y)
- ◆Transition Relationship: δ(Y, X, I) = 1

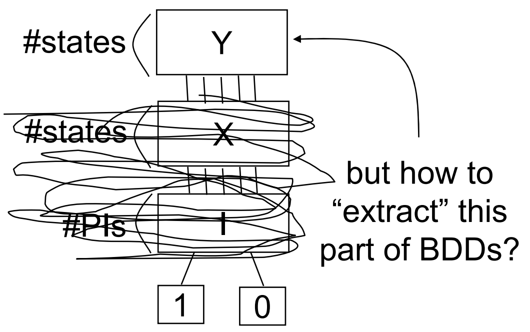


From Transition Function to Transition Relationship



◆ $\delta(Y, X, I) = \prod (y_i = T_i(X, I))$
$= (y_0 = T_0(X, I)) \,\&\, (y_1 = T_1(X, I)) \,\&\, \ldots$

The Question is…



Erase BDD variables ➔ Existential Quantification

Remember…

◆Shannon expansion of f

- $f = x * f_x + \overline{x} * f_{\overline{x}}$

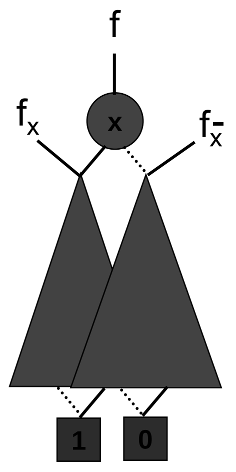
◆$f * g$

$= (x * f_x + \overline{x} * f_{\overline{x}}) *$
$(x * g_x + \overline{x} * g_{\overline{x}})$
$= x * (f_x * g_x) + \overline{x} * (f_{\overline{x}} * g_{\overline{x}})$

◆$f + g$

$= x * (f_x + g_x) + \overline{x} * (f_{\overline{x}} + g_{\overline{x}})$

◆Operation: perform on <u>cofactors</u> individually



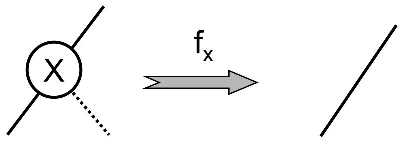
BDD Cofactor

◆Given a function f, find its positive/negative cofactor f_x / $f_{\overline{x}}$

- e.g. Let $f = a\,\overline{c} + b\,c$
  - ➔ $f_c = b$
  - ➔ $f_{\overline{c}} = a$
  - ➔ $f_{\overline{a}} = b\,c$

◆If x is top variable ➔ Left and right children

◆Otherwise,



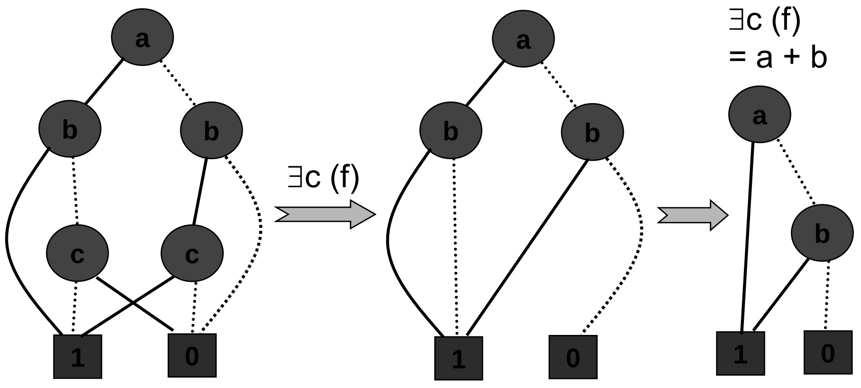
Existential Quantification

- ◆ $\exists x\,(f) = f_x + f_{\overline{x}}$
- ◆ If x is top variable
  - ➔ Perform an "OR" on its cofactors
- ◆ If x is bottom variable
  - ➔ Replace it with '1'
- ◆ If x is middle variable
  - ➔ ???

Which one is better??

Existential Quantification

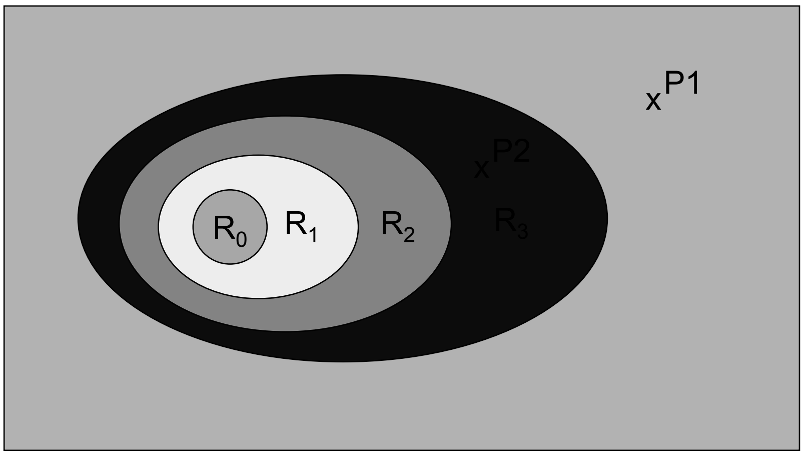
- ◆ $\exists x\,(f) = f_x + f_{\overline{x}}$
  - ● e.g. $f = a\,\overline{c} + b\,c$



BDD to Compute Set of Reachable States

- Let S_0 be the set of initial states
  - ➔ The set of states in time 1 (S_1) can be computed by ---
    - $S_1(Y) = \exists X,I\ (\delta(Y, X, I)\ \&\ S_0(X))$
- Let R_n be the set reachable states in time n
  - ➔ $R_n = \bigcup_{i=0}^{n} (S_i)$
- If $R_{n+1} = R_n$, no new state can be reached
  - ➔ Fixed point condition

Reachability Analysis for Property Checking



State space

The above reachability analysis is usually called “forward image computation”

➔ May suffer from state space explosion problems (for #variable > 200)

➔ i.e. The set of forward reachable states could be very large

Alternative: Can we compute the “backward reachable states from !P”?

➔ i.e. which states can lead to !P

Backward Reachability Analysis

◆ Image
- $S_{n+1}(Y) = \exists X,I\ (\delta(Y, X, I)\ \&\ S_n(X))$

◆ Pre-image
- $S_{n-1}(X) = \exists Y,I\ (\delta(Y, X, I)\ \&\ S_n(Y))$
- To check the property P
  - Compute the backward reachable states from !P
  - If intersect with initial states
    ➔ A counter-example is found
  - Otherwise, if reach a backward fixed point
    ➔ The property is always true

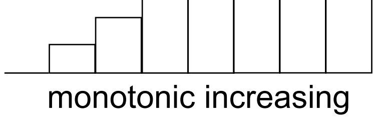
Remember,

All CTL formulas can be expressed in terms of EX, EG, and EU

How do we prove EGp??

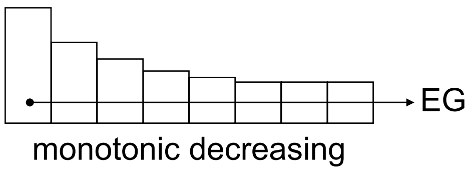
Least Fixpoint Theorem

```
leastFixPoint()
{
   R = False;
   R′ = δ(R);
   while (R != R′) {
      R = R′
      R′ = δ(R′);
   }
   return R;
}
```


monotonic increasing

Greatest Fixpoint Theorem

```
GreatestFixPoint()
{
   R = True;
   R′ = δ(R);
   while (R != R′) {
      R = R′
      R′ = δ(R′);
   }
   return R;
}
```



In short,

1. All the CTL formulae can be converted to the combinations of EX, EG, or EU
2. We can use BDDs to prove the EX, EG and EU properties
3. We can use BDDs to prove all kinds of CTL properties

But Be Aware of the State Explosion Problem

Other Types of Temporal Logic

- Temporal logics may differ according to how they handle branching in the underlying computation tree
- In a linear temporal logic, operators are provided for describing events along a single computation path
- In a branching-time logic, the temporal operators quantify over the paths that are possible from a given state

Linear Tree Logic (LTL)

- Consists of formulas that have the form Af, where f is a path formula in which the only state subformulas permitted are atomic propositions (i.e. no path quantifier)
- Formula

  A (<linear-time operator>…)

  - e.g. A(FGp)

The Logic CTL*

◆The computation tree logic CTL* (CTL-star) combines both branching-time and linear time operators

◆Formula

<Path_quantifier> <linear-time operator>… [(CTL* formula)]

e.g. A(FG(p → EF q))

Expressive Power

◆CTL*, LTL, and CTL have different expressive powers

- No CTL formula that is equivalent to the LTL formula A(FG p)
- No LTL formula that is equivalent to the CTL formula AG (EF p)
- The disjunction (A (FG p) V AG(EF p)) is a CTL* formula that is not expressible in either CTL nor LTL

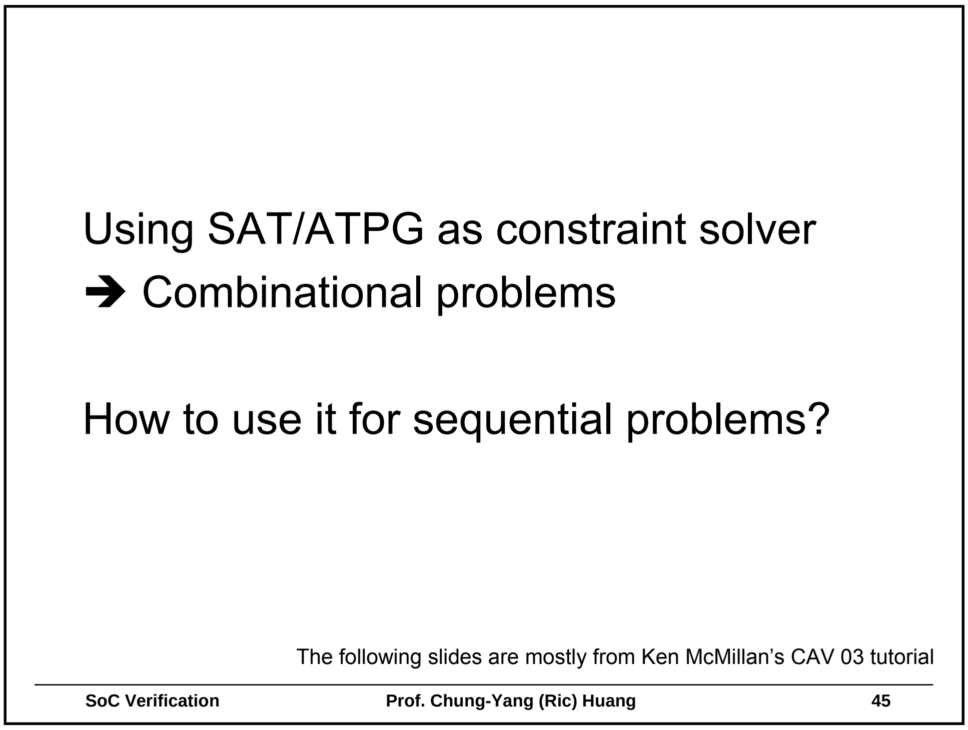
◆Can you come up with a formula that cannot be expressed by neither CTL*, CTL, nor LTL?

BDD vs. SAT/ATPG

- ◆BDD
  - Tend to solve all the solutions at once
  - Memory explosion problems
  - Note: intermediate memory usage may be larger than the end memory
- ◆SAT/ATPG
  - Find one solution at a time
  - Time complexity
  - Note: decision order matters

BDD Solver Techniques

- ◆Combinational problems
  - Global BDDs
  - Local cutting (may have false negatives)
- ◆AG (or EF problem)
  - Reachability analysis (least fixpoint)
- ◆AF (or EG problem)
  - Greatest fixpoint

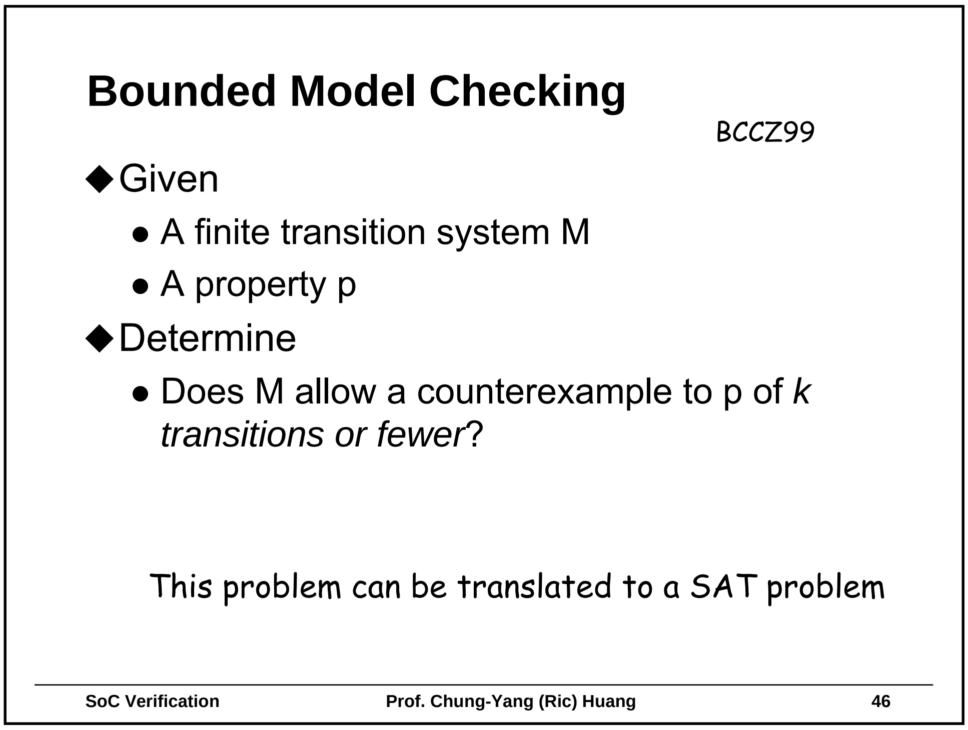


Using SAT/ATPG as constraint solver

➔ Combinational problems

How to use it for sequential problems?

The following slides are mostly from Ken McMillan's CAV 03 tutorial



Bounded Model Checking

BCCZ99

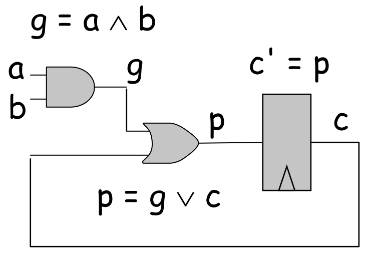
◆ Given
- A finite transition system M
- A property p

◆ Determine
- Does M allow a counterexample to p of *k transitions or fewer*?

This problem can be translated to a SAT problem

Models

Transition system described by a set of constraints



Model:

$C = \{$
$\quad g = a \wedge b,$
$\quad p = g \vee c,$
$\quad c' = p$
$\}$

Each circuit element is a constraint
note: $a = a_t$ and $a' = a_{t+1}$

Properties

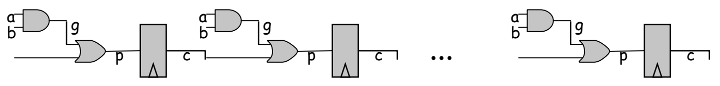
- We restrict our attention to safety (AG) properties.
- Characterized by:
  - Initial condition I
  - Final condition F (representing "bad" states)
- A counterexample is a path from a state satisfying I to state satisfying F, where every transition satisfies C.

Unfolding

◆Unfold the model k times:

$U_k = C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1}$

I_0



F_k

- Use SAT solver to check satisfiability of

  $I_0 \wedge U_k \wedge F_k$

- A satisfying assignment is a counterexample of k steps

K is unknown…

```
1. for (k = 0 to infinity)
2.     T = $I_0 \wedge C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1} \wedge F_k$
3.     if (solve(T = 1) == true)
4.         return HAS_SOLUTION;
5.     if (effort exceeds limit)
6.         return ABORT;
7. endfor
```

Cannot prove "NO_SOLUTION"

BMC applications

- Debugging:
  - Can find counterexamples using a SAT solver
- Proving properties:
  - Only possible if a bound on the length of the shortest counterexample is known.
    - I.e., we need a *diameter* bound. The diameter is the maximum lenth of the shortest path between any two states.
  - Worst case is exponential. Obtaining better bounds is sometimes possible, but generally intractable.

Unbounded Model Checking

- We consider a variety of methods to explioit SAT and BMC for unbounded model checking:
  - K-step induction
  - Abstraction
    - Counterexample-based
    - Non-counterexample-based
  - Exact image computations
    - SAT solver tests for fixed point
    - SAT solver computes image
  - Over-approximate image computations

K-induction

SSS2000

- ◆ Induction:

$$\frac{P(s_0) \qquad \forall i: P(s_i) \Rightarrow P(s_{i+1})}{\forall i: P(s_i)}$$

- k-step induction:

$$\frac{P(s_{0..k-1}) \qquad \forall i: P(s_{i..i+k-1}) \Rightarrow P(s_{i+k})}{\forall i: P(s_i)}$$

K-induction with a SAT solver

- ◆ Recall:

  $U_k = C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1}$
- ◆ Two formulas to check:
  - ● Base case:

    $I_0 \wedge U_{k-1} \Rightarrow P_0 \ldots P_{k-1}$
  - ● Induction step:

    $U_k \wedge P_0 \ldots P_{k-1} \Rightarrow P_k$
- ◆ If both are valid, then P always holds.
- ◆ If not, increase k and try again.

Induction SAT

```
for (k = 0 to infinity)
    S = $U_k \wedge F_k$
    T = $I_0 \wedge S$
    // induciton step
    if (solve(S = 1) == false)
       return NO_SOLUTION;
    // normal proof: base case for next k
    if (solve(T = 1) == true)
       return HAS_SOLUTION;
    if (effort exceeds limit)
       return ABORT;
endfor
```

Does “Induction SAT” guarantee convergence?

i.e. We will either

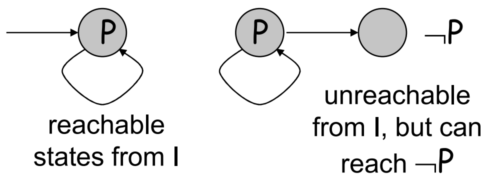
1. conclude no solution in induction step

or 2. find a counter-example in normal proof with a finite number k ???

Simple path assumption

◆Unfortunately, k-induction is not complete.

- Some properties not k-inductive for any k.



◆Simple path restriction:

- There is a path to ¬P iff there is a *simple* path to ¬P (path with no repeated states).

Induction over simple paths

◆Let simple($s_{0..k}$) be defined as:

- $\forall i,j$ in $0..k$: $(i \neq j) \Rightarrow s_i \neq s_j$

◆k-induction over simple paths:

$$\frac{P(s_{0..k-1}) \qquad \forall i: \text{simple}(s_{0..k}) \wedge P(s_{i..i+k-1}) \Rightarrow P(s_{i+k})}{\forall i: P(s_i)}$$

Must hold for k large enough, since a simple path cannot be unboundedly long. Length of longest simple path is called *recurrence diameter*.

...with a SAT solver

◆ For simple path restriction, let:

$S_k = \forall t=0..k,\ t'=t+1..k:\ \neg \forall v \text{ in } V : v_t = v_{t'}$

(where V is the set of state variables).

◆ Two formulas to check:

- Base case:

$$I_0 \wedge U_{k-1} \Rightarrow P_0 ... P_{k-1}$$

- Induction step:

$$S_k \wedge U_k \wedge P_0 ... P_{k-1} \Rightarrow P_k$$

◆ If both are valid, then P always holds.

◆ If not, increase k and try again.

Is the recurrence diameter the same as the diameter (the max of the shortest path between any 2 states)??

Termination

◆ Termination condition:

k is the length of the longest simple path of the form

$P^* \ \neg P$

◆ This can be exponentially longer than the diameter.

- example:
  - loadable mod 2^N counter where P is (count $\neq 2^N-1$)
  - diameter = 1
  - longest simple path = 2^N

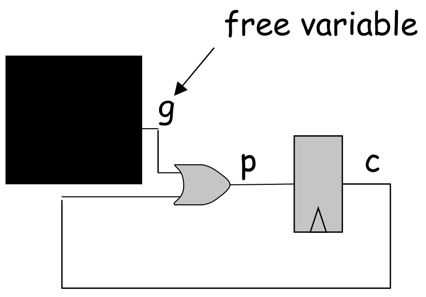
◆ Nice special cases:

- P is a tautology (k=0)
- P is inductive invariant (k=1)

Localization abstraction

Kurshan

◆Property: G (c $\Rightarrow$ X c)



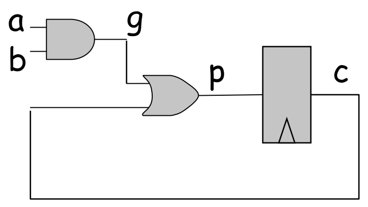
Model:

C'= {
[redacted]
p = g ∨ c,
c' = p
}

$$\frac{C' \Rightarrow \text{property},\ C \Rightarrow C'}{C \Rightarrow \text{property}}$$

Constraint granularity

Most authors use constraints at "latch" granularity...



Model:

$C = \{$
$c' = (a \wedge b) \vee c$
$\}$

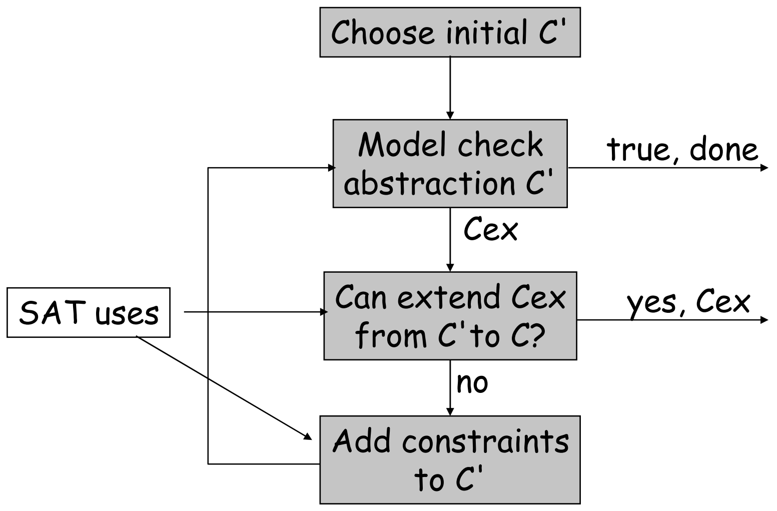
...however, techniques we will consider can be applied at both "gate" and "latch" granularity.

Localization, cont

- ◆ C' may refer to fewer state variables than C
  - reduction in the state explosion problem
- ◆ Key issue: how to choose constraints in C'
  - counterexample-based
  - proof-based

Algorithm

Kurshan



Abstract counterexamples

- Assume simple safety property:
  - initial condition I and final condition F
  - w.l.o.g., assume I and F are atomic formulas
    - to make this true, add constraints in C:

      $v_I \Leftrightarrow I \qquad v_F \Leftrightarrow F$
- Abstract variables V' = support(C',I,F)
- Abstract counterexample A' is a truth assignment to:

  $\{ v_t \mid v \text{ in } V',\ t \text{ in } 0..k \}$

  where k is the number of steps.

Counterexample extension

CGJLV 2000

◆ Abstract counterexample A' satisfies:

$I_0 \wedge U'_k \wedge F_k$ where $U'_k = C'_0 \wedge C'_1 \wedge \ldots \wedge C'_{k-1}$

◆ Find A consistent with A', satisfying:

$I_0 \wedge U_k \wedge F_k$ where $U_k = C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1}$

◆ That is, A is any satisfying assignment to:

$A' \wedge I_0 \wedge U_k \wedge F_k$

I.e., to extend an abstract counterexample, we just apply it as a constraint in BMC. If unsat, abstract counterexample is "false".

Abstraction refinement

◆ Refinement = adding constraints to C' to eliminate false counterexamples.

◆ Many heuristsics used for this.

- Too many to cover here.
- SAT solver can produce a resolution-based refutation in the UNSAT case....

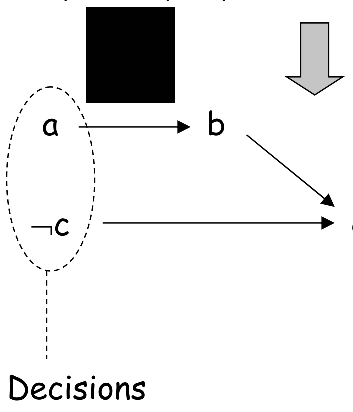
DPLL-style SAT solvers

SATO,GRASP,CHAFF,BERKMIN

- Objective:
  - Check satisfiability of a CNF formula
    - literal: v or ¬v
    - clause: disjunction of literals
    - CNF: conjunction of clauses
- Approach:
  - Branch: make arbitrary decisions
  - Propagate implication graph
  - Use conflicts to guide inference steps

The Implication Graph (BCP)

(¬a ∨ b) ∧ (¬b ∨ c ∨ d)



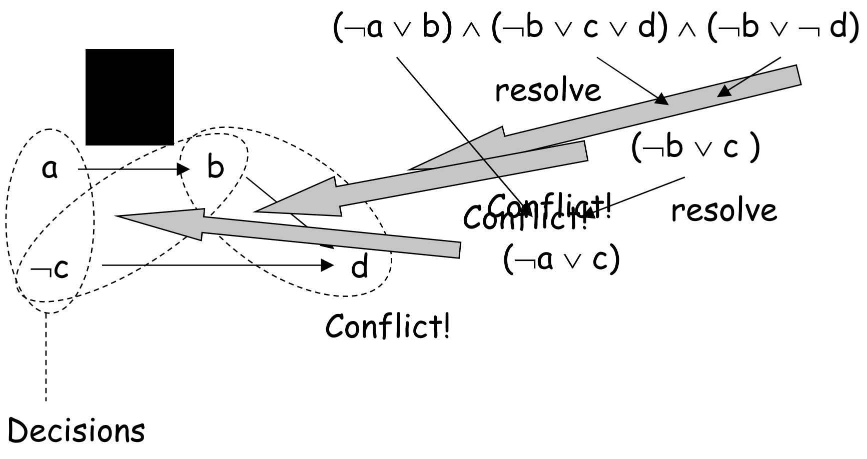
Assignment: a ∧ b ∧ ¬c ∧ d

Resolution

$$\begin{tikzcd}
a \lor b \lor \neg c \arrow[dr] & & \neg a \lor \neg c \lor d \arrow[dl] \\
& b \lor \neg c \lor d &
\end{tikzcd}$$

When a conflict occurs, the implication graph is used to guide the resolution of clauses, so that the same conflict will not occur again.

Conflict Clauses



Assignment: a ∧ b ∧ ¬c ∧ d

Conflict Clauses (cont.)

◆Conflict clauses:

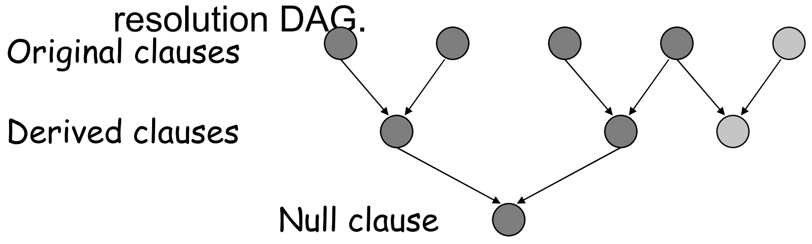
- Are generated by resolution
- Are implied by existing clauses
- Are in conflict in the current assignment
- Are safely added to the clause set

Many heuristics are available for determining when to terminate the resolution process.

Generating refutations

◆Refutation = a proof of the null clause

- Record a DAG containing all resolution steps performed during conflict clause generation.
- When null clause is generated, we can extract a proof of the null clause as a resolution DAG.



Proof-based refinement

- ◆ Recall, to extend abstract Cex A', we check:

  $A' \wedge I_0 \wedge U_k \wedge F_k$

- ◆ If UNSAT, we obtain refutation proof P
  - ● proof that A' cannot be extended to concrete Cex
- ◆ Let E be set of constraints used in proof P:

  $E = \{ c \in C \mid \text{some } c_i \text{ occurs in } P \}$

- ◆ A' cannot be extended to a Cex for E
  - ● P is the proof of this.

Thus, add E to C' and continue...

In other words...

The refutation of the formula:

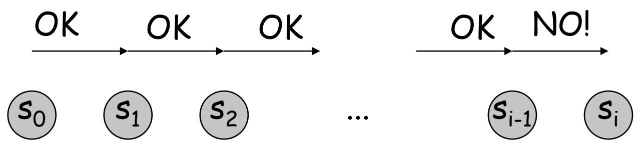
$A' \wedge I_0 \wedge U_k \wedge F_k$

gives us a sufficient set of constraints to rule out the abstract counterexample.

We continue ruling out counterexamples until either the abstraction C' proves the property or we can extend an abstract counterexample to a concrete one.

CCKSVW approach (FMCAD02)

- Find the shortest prefix of Cex A' that cannot be extended.



- That is,

  $A' \wedge I_0 \wedge U_k \wedge F_k$

  is feasible for all k < i, but not for k=i.

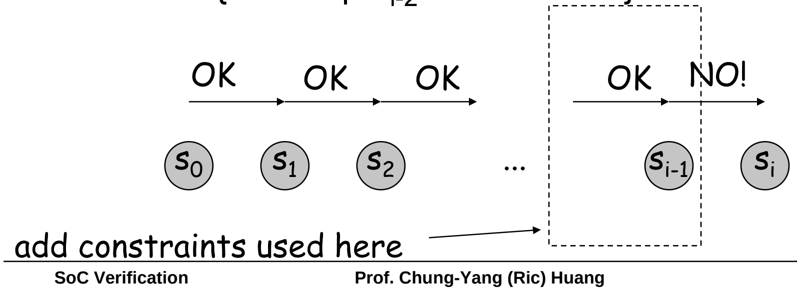
CCKSVW approach cont.

- Let P be a refutation of

  $A' \wedge I_0 \wedge U_i \wedge F_i$

- Let E be set of constraints used in proof P *only on state s_{i-1}:*

  E = { $c \in C$ | c_{i-2} occurs in P }



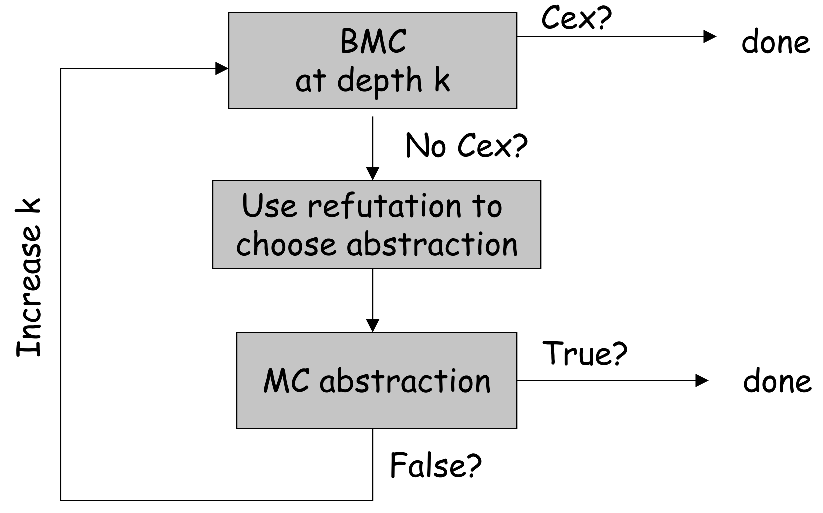
Weakness of Cex-based approach

◆ Arbitrarily chosen abstract Cex may be refutable for many reasons not related to property.
  - Thus, may add irrelevant constraints.
  - To remedy, may try to characterize a set of Cex's rather than just one (e.g., GKM-HFV,TACAS03).

Alternative: don't use counterexamples

Proof-based abstraction

MA,TACAS03



BMC phase

- ◆Unfold the model k times:

  $U = C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1}$

- Use SAT solver to check satisfiability of

  $I_0 \wedge U \wedge F_k$

- If unsatisfiable:
  - property has no Cex of length k
  - produce a refutation proof P

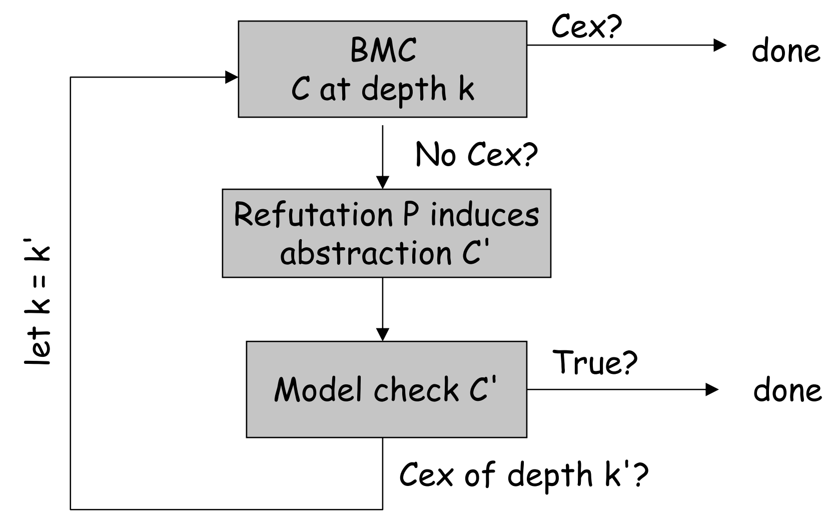
Abstraction phase

- ◆Let C' be set of constraints used in proof P:

  $C' = \{ c \in C \mid \text{ some } c_i \text{ occurs in } P \}$

- ◆C' admits no counterexample *of length k*
  - ● let $U' = C'_0 \wedge C'_1 \wedge \ldots \wedge C'_{k-1}$
  - ● P is a refutation of $I_0 \wedge U' \wedge F_k$
- ◆Model check property on C'
  - ● property true for C' implies true for C
  - ● else Cex of length k' > k (why?)
    - ▪ restart for k = k'

Algorithm



Notice: MC counterexample is thrown away!

Termination

- ◆Depth k increases at each iteration
- ◆Eventually k > d, diameter of C'
- ◆If k > d, no counterexample is possible

In practice, termination uses occurs when k ≈ d/2

Usually, diameter C' << diameter of C

Weakness of proof-based abs

- BMC must refute all counterexamples of length k, while in Cex-based, BMC must refute only one (partial) counterexample.

Inference

- SAT solver seems to be *very* effective at narrowing down the proof to relevant facts.

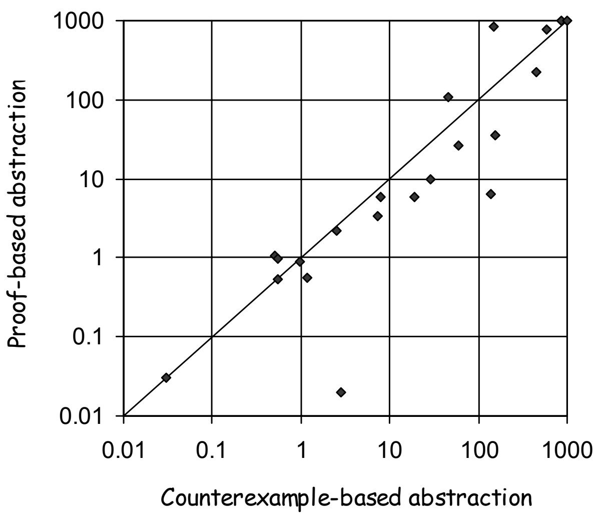
In most cases, it did better than manual abstraction.

Comparing CBA and PBA

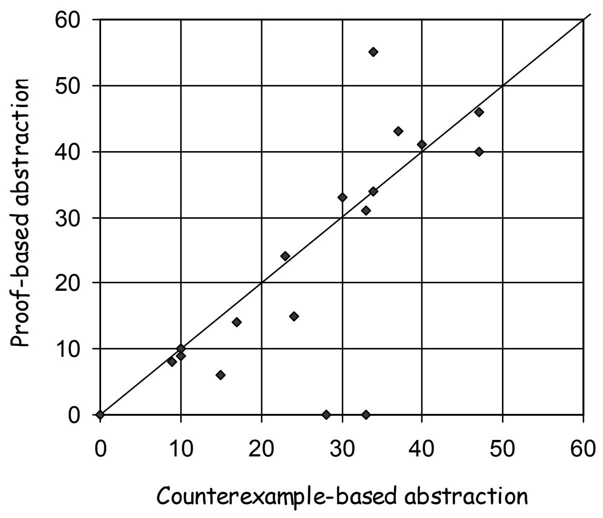
◆ Apples-apples comparison
- same SAT solver
- same model checker
- only differences are:
  - For CBA previous A' is kept as a constriaint for BMC, C' is cumulative.
  - For PBA previous A' and C' are thrown away each iteration.

Note these are my implementations. This says nothing about performance of specific tools!

Run time comparison

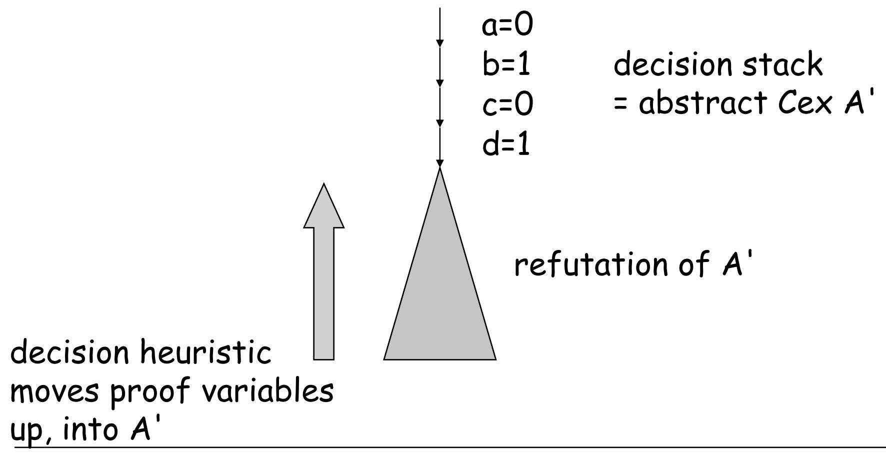


Abstraction comparison



Possible explanation

◆Internally, SAT solver is really doing CBA



A (fuzzy) hypothesis

SAT-based BMC "succeeds" when number of relevant variables is small, and fails otherwise.

"success" is BMC for k = diameter of relevant logic

- Parameterized models allowing no abstraction

Model	Max state vars
German protocol	42
"swap"	21

Implications

- Most of the time if bounded model checking succeeds, unbounded model checking also succeeds using abstraction.
- No need to settle for time bounded result
- Bounded model checking may be applicable only to localizable properties

Image computation methods

- ◆ Symbolic model checking without BDD's
  - ● Use SAT solver just for fixed-point detection
    - ▪ Abdulla, Bjesse and Een 2000
    - ▪ Williams, Biere, Clarke and Gupta 2000
  - ● Adapt SAT solver to compute image directly
    - ▪ McMillan, 2002

Symbolic model checking

- ◆ Recall: Fixed point characterizaion of CTL:

$$EFp = \mu . Q\ p \vee EX\ Q$$

- ◆ Reverse image:

$$EXp = \exists \underbrace{W.}_{\text{input variables}}\ p < \underbrace{\delta_i}_{\text{transition function}} / \underbrace{s_i}_{\text{state variable}} >$$

Syntactic expansion of quantifiers

- By definition:
  - $\exists w.\ p = p\langle 0/w \rangle \vee p\langle 1/w \rangle$
- Thus, we can compute reverse image by syntactic expansion and simplification.
  - note: expontential in number of inputs.
- Fixed-point series:

$$R_0 = \text{false}$$
$$R_{i+1} = p \vee \text{EX } R_i$$

Terminates when $R_{i+1} \Rightarrow R_i$
(SAT problem)

Limitations

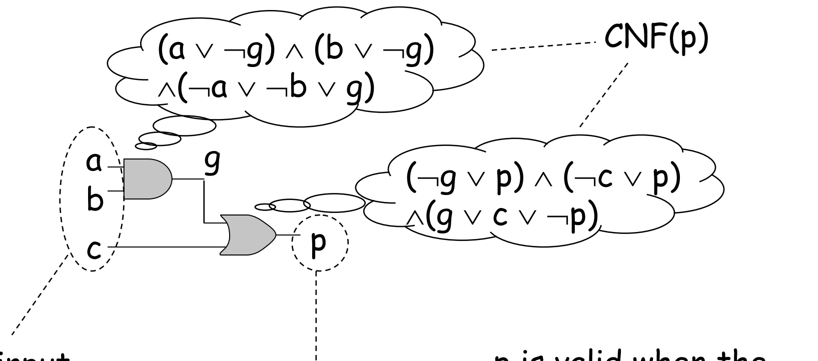
- Syntactic quantifier elimination is exponential
  - Method limited to circuits with very few inputs
  - E.g., sequential arithmetic circuits

Direct image computation

- ◆ Adapt SAT methods for image computation in symbolic model checking
  - ● Recall: this is essentially quantifier elimination
- ◆ Idea: reduce formula to CNF or DNF
  - ● Make quantifier elimination easy
  - ● Essentially, enumerate all satisfying assignments, but in an efficient way (i.e., by covering them with clauses or cubes).

Circuit Validity

Can the circuit output be 0?


$(a \vee \neg g) \wedge (b \vee \neg g) \wedge(\neg a \vee \neg b \vee g)$
CNF(p)
$(\neg g \vee p) \wedge (\neg c \vee p) \wedge(g \vee c \vee \neg p)$
a
b
c
g
p

input variables V_I

output variable

p is valid when the formula CNF(p) $\wedge$ $\neg$p is unsatisfiable

CNF Characterization

Instead of checking validity of p, we now want to derive a CNF formula over the input variables V_I that is logically equivalent to the circuit.

Idea: each time a satisfying assignment is found, add a new "blocking clause" that rules out this satisfying assignment.

The blocking clauses form our characterization of p.

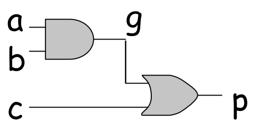
Blocking clauses

- Blocking clauses must:
  - be implied by p
  - be in conflict in the current assignment
  - involve only input variables (in V_I)

Can we use conflict clauses as blocking clauses?

Not quite...

An example



Want to characterize p in CNF:

•Test satisfiability of CNF(p) $\wedge \neg p$

Guess the assignment *A* = a

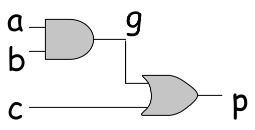
Implication graph:

$$\begin{tikzcd}
a \arrow[rr] & & \neg b \\
& \neg g \arrow[ur] & \\
\neg p \arrow[ur] \arrow[dr] & & \\
& \neg c &
\end{tikzcd}$$

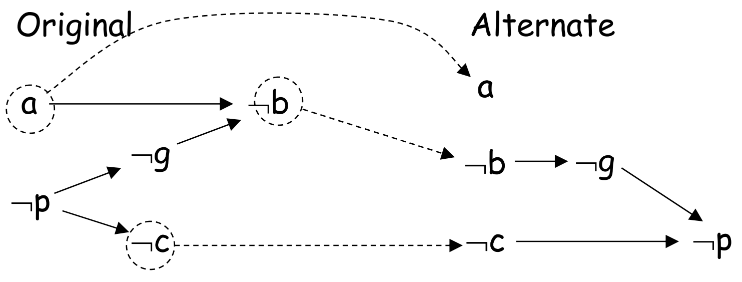
Satisfying!

Problem:
We can't infer anything from p, because $\neg p$ is already a root of the graph.

Alternate implication graph

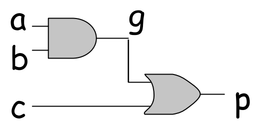


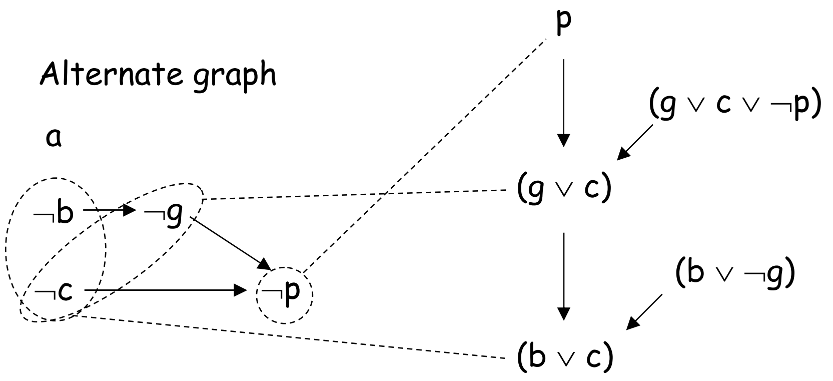
Construct a new implication graph rooted at the input variables.



Now we can always generate a conflict clause from p using only input variables.

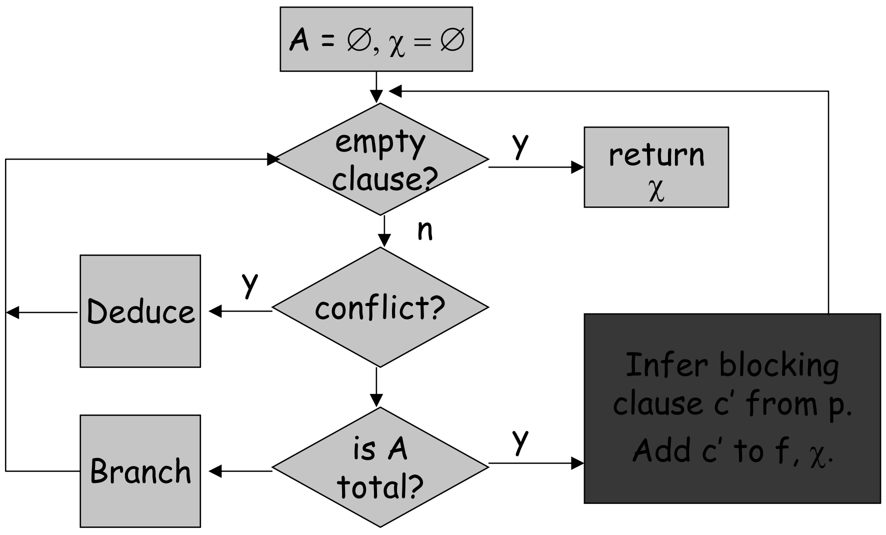
Blocking clause example





We stop when the clause has only inputs

CNF characterization algorithm



Universal Quantifier Elimination

Given

- a circuit p, and
- a subset W of the input variables,

we want to compute a CNF formula equivalent to

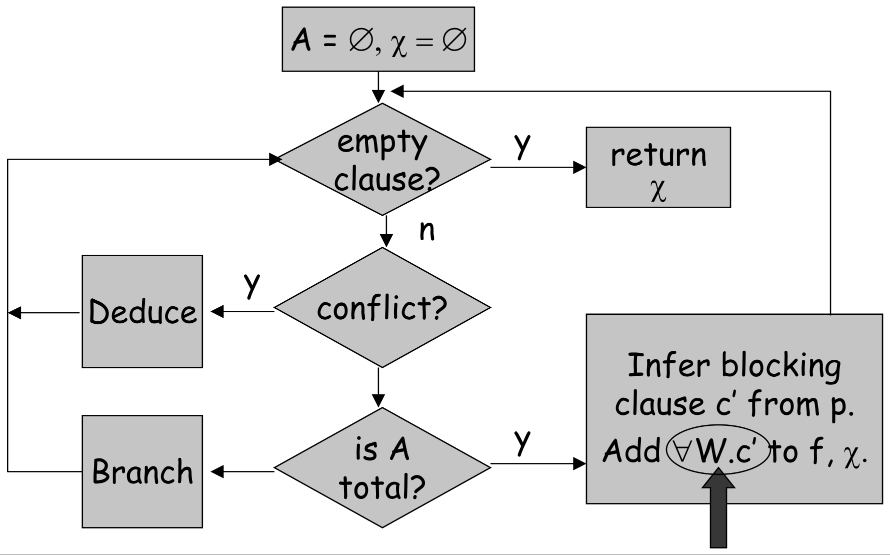
$$\forall W.p$$

Idea: Eliminating in CNF formulas is trivial.

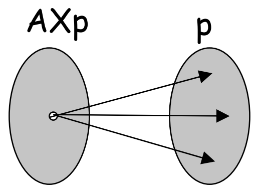
e.g.: $\forall a. (a \vee b) \wedge (\neg a \vee \neg c \vee d) \;=\; (b) \wedge (\neg c \vee d)$

... just push $\forall$ inside $\wedge$ ...

$\forall$ - elimination algorithm



CTL Model Checking with SAT



$$AXp = \forall W.\ p \langle \delta_i / s_i \rangle$$

input variables

transition function

state variable

Recent related work

- Sheng, Hsiao (DATE 2003)
  - Uses ATPG methods
- Chauhan, Clarke, Kroenig
  - Computes forward rather than backward image

SAT-based image

- ◆May provide a good alternative when BDD's fail.
- ◆Does not take advantage of SAT solver's ability to filter out irrelevant facts, since exact image is computed.

Image over-approximation

- ◆BMC and Craig interpolation allow us to compute image over-approximatino relative to property.
  - ● Avoid computing exact image.
  - ● Maintain SAT solver's advantage of filtering out irrelevant facts.

Interpolation

(Craig,57)

- If $A \wedge B$ = false, there exists an *interpolant* A' for (A,B) such that:

  $A \Rightarrow A'$

  $A' \wedge B$ = false

  A' refers only to common variables of A,B
- Example:
  - $A = p \wedge q$, $B = \neg q \wedge r$, $A' = q$
- New result
  - given a resolution refutation of $A \wedge B$,

    A' can be derived in linear time.

(Pudlak,Krajicek,97)

Interpolation-based MC

- Interpolation gives us
  - SAT-based algorithm for over-approximate image computation, using interpolation
  - SAT-only symbolic model checking

Reachability

- Is there a path from I to F satisfying transition constraint C?
- Reachability fixed point:

$$R_0 = I$$
$$R_{i+1} = R_i \vee Img(R_i,C)$$
$$R = \cup R_i$$

- Image operator:

$$Img(P,C) = \lambda V'.\ \exists V.\ (P \wedge C)$$

- F is reachable iff $R \wedge F \neq$ false

Overapproximation

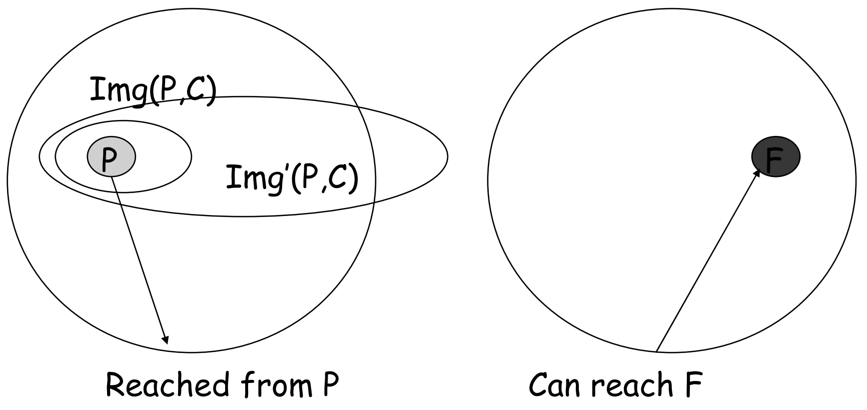
- An overapproximate image op. is Img' s.t.

  for all P, Img(P,C) implies Img'(P,C)
- Overapprimate reachability:

$$R'_0 = I$$
$$R'_{i+1} = R'_i \vee Img'(R'_i,C)$$
$$R' = \cup R'_i$$

- Img' is adequate (w.r.t.) F, when
  - if P cannot reach F, Img′(P,C) cannot reach F
- If Img' is adequate, then
  - F is reachable iff $R' \wedge F \neq$ false

Adequate image



But how do you get an adequate Img'?

k-adequate image operator

◆Img' is k-adequate (w.r.t.) F, when
- if P cannot reach F,
  Img'(P,C) cannot reach F within k steps

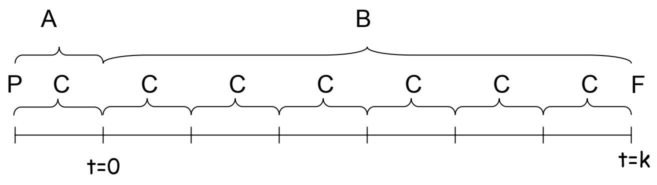
◆Note, if k > diameter, then k-adequate is equivalent to adequate.

Interpolation-based image

◆Idea -- use unfolding to enforce k-adequacy

$A = P_{-1} \wedge C_{-1}$

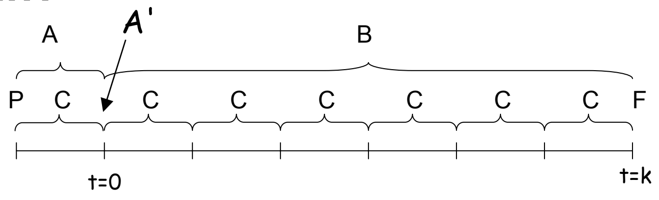
$B = C_0 \wedge C_1 \wedge \ldots \wedge C_{k-1} \wedge F_k$



Let Img'(P)$_0$= A',
  where A' is an interpolant for (A,B)...

Img' is k-adequate!

Huh?



◆$A \Rightarrow A'$

- Img(P,C) $\Rightarrow$ Img'(P,C)

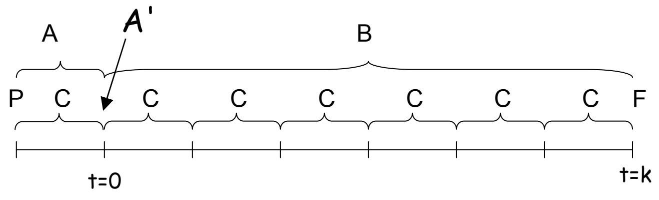
◆$A' \wedge B$ = false

- Img'(P,C) cannot reach F in k steps

◆Hence Img' is k-adequate overapprox.

But note, Img' is partial -- not defined if $A \wedge B$ is sat.

Intuition



- A' tells is everything the SAT solver deduced about the image of P in proving it can't reach F in k steps.
- Hence, A' is in some sense an abstraction of the image relative to the property.

Reachability algorithm

```
let k = 0
repeat
  if I can reach F within k steps, answer reachable
  R = I
  while Img'(R,C) $\wedge$ F = false
     R' = Img'(R,C) $\vee$ R
     if R' = R answer unreachable
     R = R'
  end while
  increase k
```

Termination

- ◆Since k increases at every iteration, eventually k > d, the diameter, in which case Img' is adequate, and hence we terminate.

Notes:

- don't need to know when k > d in order to terminate
- often termination occurs with k << d
- depth bound for earlier method (Sheeran et al '00) is "longest simple path", which can be exponentially longer than diameter

Interpolation-based MC

- ◆Fully SAT-based.
- ◆Inherits SAT solvers ability to concentrate on facts relevant to a property.
- ◆Like CBA, PBA, most effective when
  - Very large set of facts is available
  - Only a small subset are relevant to property
- ◆For true properties, appears to converge for smaller k values.

Conclusion

- SAT solvers are very effective at ignoring irrelevant facts
  - Can think of decision heuristic as a form of CBA
- SAT solvers can produce refutations
- We can exploit in a number of ways:
  - BMC
  - Abstraction for UMC (either CBA or PBA)
  - Abstract image computations using interpolation

This makes it possible to model check *localizable* properties large systems.

Conclusion cont.

- Approaches that compute exact images sacrifice this quality of SAT solvers.
  - still useful as alternative to BDD's
- For non-localizable properties, SAT-based BMC and UMC do not perform well.
- The capacity of SAT-based UMC is comparable to BMC.
  - no need to settle for bounded results!