A.A. 2006-2007, CDLS in Informatica

Introduction to Formal Methods

06: SAT Based Bounded Model Checking

Roberto Sebastiani – rseba@dit.unitn.it

Some material (text, figures) displayed in these slides is courtesy of M. Benerecetti, A. Cimatti, P. Pandya, M. Pistore, M. Roveri, S.Tonetta. S. Tonetta.

Last update: December 6, 2006

Content

SAT-based Bounded Model Checking

- ▷ Key problems with BDD's:
  - they can explode in space
  - an expert user can make the difference (e.g. reordering, algorithms)
- ▷ A possible alternative:
  - Propositional Satisfiability Checking (SAT)
  - SAT technology is very advanced
- ▷ Advantages:
  - reduced memory requriements
  - limited sensitivity: one good setting, does not require expert users
  - much higher capacity (more variables) than BDD based techniques

SAT-based Bounded Model Checking [cont.]

Key ideas:

- ▷ look for counter-example paths of increasing length k
  - oriented to finding bugs
- ▷ for each k, builds a boolean formula that is satisfiable iff there is a counter-example of length k
  - can be expressed using $k \cdot |\mathbf{s}|$ variables
  - formula construction is not subject to state explosion
- ▷ satisfiability of the boolean formulas is checked using a *SAT procedure*
  - can manage complex formulae on several 100K variables
  - returns satisfying assignment (i.e., a counter-example)

Content

Basic notation & definitions

- Boolean formula
  - $\top, \bot$ are formulas
  - A propositional atom $A_1, A_2, A_3, \ldots$ is a formula;
  - if φ_1 and φ_2 are formulas, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2$ are formulas.
- Literal: a propositional atom A_i (positive literal) or its negation $\neg A_i$ (negative literal)
- N.B.: if $l := \neg A_i$, then $\neg l := A_i$
- $Atoms(\varphi)$: the set $\{A_1, \ldots, A_N\}$ of atoms occurring in φ.
- a boolean formula can be represented as a tree or as a DAG

Basic notation & definitions (cont)

- ▷ Total truth assignment μ for φ:

  $\mu : Atoms(\varphi) \longmapsto \{\top, \bot\}$.

- ▷ Partial Truth assignment μ for φ:

  $\mu : \mathcal{A} \longmapsto \{\top, \bot\}$, $\mathcal{A} \subset Atoms(\varphi)$.

- ▷ Set and formula representation of an assignment:

  - μ can be represented as a set of literals:

    EX: $\{\mu(A_1) := \top, \mu(A_2) := \bot\} \implies \{A_1, \neg A_2\}$

  - μ can be represented as a formula:

    EX: $\{\mu(A_1) := \top, \mu(A_2) := \bot\} \implies A_1 \wedge \neg A_2$

Basic notation & definitions (cont)

- $\mu \models \varphi$ (μ satisfies φ):
  - $\mu \models A_i \Longleftrightarrow \mu(A_i) = \top$
  - $\mu \models \neg\varphi \Longleftrightarrow$ *not* $\mu \models \varphi$
  - $\mu \models \varphi_1 \wedge \varphi_2 \Longleftrightarrow \mu \models \varphi_1$ *and* $\mu \models \varphi_2$
  - ...
- φ is satisfiable iff $\mu \models \varphi$ for some μ
- $\varphi_1 \models \varphi_2$ (φ_1 entails φ_2):

  $\varphi_1 \models \varphi_2$ iff for every μ $\mu \models \varphi_1 \Longrightarrow \mu \models \varphi_2$
- $\models \varphi$ (φ is valid):

  $\models \varphi$ iff for every μ $\mu \models \varphi$
- φ is valid $\Longleftrightarrow$ $\neg\varphi$ is not satisfiable

Equivalence and equi-satisfiability

- φ_1 and φ_2 are equivalent iff, for every μ,
  $\mu \models \varphi_1$ iff $\mu \models \varphi_2$
- φ_1 and φ_2 are equi-satisfiable iff
  exists μ_1 s.t. $\mu_1 \models \varphi_1$ iff exists μ_2 s.t. $\mu_2 \models \varphi_2$
- φ_1, φ_2 equivalent

  $$\Downarrow \quad \not\Uparrow$$

  φ_1, φ_2 equi-satisfiable

- EX: $\varphi_1 \vee \varphi_2$ and $(\varphi_1 \vee \neg A_3) \wedge (A_3 \vee \varphi_2)$, A_3 not in $\varphi_1 \vee \varphi_2$, are equi-satisfiable but not equivalent.

Complexity

- ▷ The problem of deciding the satisfiability of a propositional formula is NP-complete.
- ▷ The most important logical problems (validity, inference, entailment, equivalence, ...) can be straightforwardly reduced to satisfiability, and are thus (co)NP-complete.

$$\Downarrow$$

No existing worst-case-polynomial algorithm.

POLARITY of subformulas

Polarity: the number of nested negations modulo 2.

▷ Positive/negative occurrences

- φ occurs positively in φ;
- if $\neg\varphi_1$ occurs positively [negatively] in φ,
  then φ_1 occurs negatively [positively] in φ
- if $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ occur positively [negatively] in φ,
  then φ_1 and φ_2 occur positively [negatively] in φ;
- if $\varphi_1 \rightarrow \varphi_2$ occurs positively [negatively] in φ,
  then φ_1 occurs negatively [positively] in φ and φ_2 occurs positively [negatively] in φ;
- if $\varphi_1 \leftrightarrow \varphi_2$ occurs in φ,
  then φ_1 and φ_2 occur positively and negatively in φ;

Negative normal form (NNF)

- φ is in Negative normal form iff it is given only by applications of $\wedge, \vee$ to literals.
- every φ can be reduced into NNF:
  1. substituting all $\rightarrow$'s and $\leftrightarrow$'s:

$$\varphi_1 \rightarrow \varphi_2 \quad \Longrightarrow \quad \neg\varphi_1 \vee \varphi_2$$

$$\varphi_1 \leftrightarrow \varphi_2 \quad \Longrightarrow \quad (\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2)$$

  2. pushing down negations recursively:

$$\neg(\varphi_1 \wedge \varphi_2) \quad \Longrightarrow \quad \neg\varphi_1 \vee \neg\varphi_2$$

$$\neg(\varphi_1 \vee \varphi_2) \quad \Longrightarrow \quad \neg\varphi_1 \wedge \neg\varphi_2$$

$$\neg\neg\varphi_1 \quad \Longrightarrow \quad \varphi_1$$

- The reduction is linear if a DAG representation is used.
- Preserves the equivalence of formulas.

Conjunctive Normal Form (CNF)

- φ is in Conjunctive normal form iff it is a conjunction of disjunctions of literals:

$$\bigwedge_{i=1}^{L} \bigvee_{j_i=1}^{K_i} l_{j_i}$$

- the disjunctions of literals $\bigvee_{j_i=1}^{K_i} l_{j_i}$ are called clauses
- Easier to handle: list of lists of literals.

  $\Longrightarrow$ no reasoning on the recursive structure of the formula

Classic CNF Conversion $CNF(\varphi)$

- Every φ can be reduced into CNF by, e.g.,
  1. converting it into NNF;
  2. applying recursively the DeMorgan's Rule:

$$(\varphi_1 \wedge \varphi_2) \vee \varphi_3 \implies (\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3)$$

- Worst-case exponential.
- $Atoms(CNF(\varphi)) = Atoms(\varphi)$.
- $CNF(\varphi)$ is equivalent to φ.
- Normal: if φ_1 equivalent to φ_2, then $CNF(\varphi_1)$ identical to $CNF(\varphi_2)$ modulo reordering.
- Rarely used in practice.

Labeling CNF conversion $CNF_{label}(\varphi)$

- Every φ can be reduced into CNF by, e.g., applying recursively bottom-up the rules:

$$\varphi \implies \varphi[(l_i \vee l_j)|B] \wedge CNF(B \leftrightarrow (l_i \vee l_j))$$

$$\varphi \implies \varphi[(l_i \wedge l_j)|B] \wedge CNF(B \leftrightarrow (l_i \wedge l_j))$$

$$\varphi \implies \varphi[(l_i \leftrightarrow l_j)|B] \wedge CNF(B \leftrightarrow (l_i \leftrightarrow l_j))$$

  l_i, l_j being literals and B being a “new” variable.
- Worst-case linear.
- $Atoms(CNF_{label}(\varphi)) \supseteq Atoms(\varphi)$.
- $CNF_{label}(\varphi)$ is equi-satisfiable w.r.t. φ.
- Non-normal.
- More used in practice.

DPLL

- Davis-Putnam-Longeman-Loveland procedure (DPLL)
- Tries to build recursively an assignment μ satisfying φ;
- At each recursive step assigns a truth value to (all instances of) one atom.
- Performs deterministic choices first.

DPLL Algorithm

```
function DPLL(φ, μ)
    if φ = ⊤                                            /* base      */
        then return True;
    if φ = ⊥                                            /* backtrack */
        then return False;
    if {a unit clause (l) occurs in φ}                  /* unit      */
        then return DPLL(assign(l, φ), μ ∧ l);
    if {a literal l occurs pure in φ}                   /* pure      */
        then return DPLL(assign(l, φ), μ ∧ l);
    l := choose-literal(φ);                             /* split     */
    return  DPLL(assign(l, φ), μ ∧ l)  or
            DPLL(assign(¬l, φ), μ ∧ ¬l);
```

Variants of DPLL

DPLL is a family of algorithms.

- ▷ different splitting heuristics
- ▷ preprocessing: (subsumption, 2-simplification)
- ▷ backjumping
- ▷ learning
- ▷ random restart
- ▷ horn relaxation
- ▷ ...

DPLL – summary

- Handles CNF formulas
- Probably the most efficient SAT algorithm
- Requires polynomial space!!!

  $\Longrightarrow$ very limited memory requirements
- Choose_literal() critical!
- Advanced optimization techniques
- Many very efficient implementations [e.g., Chaff]

Many applications of SAT

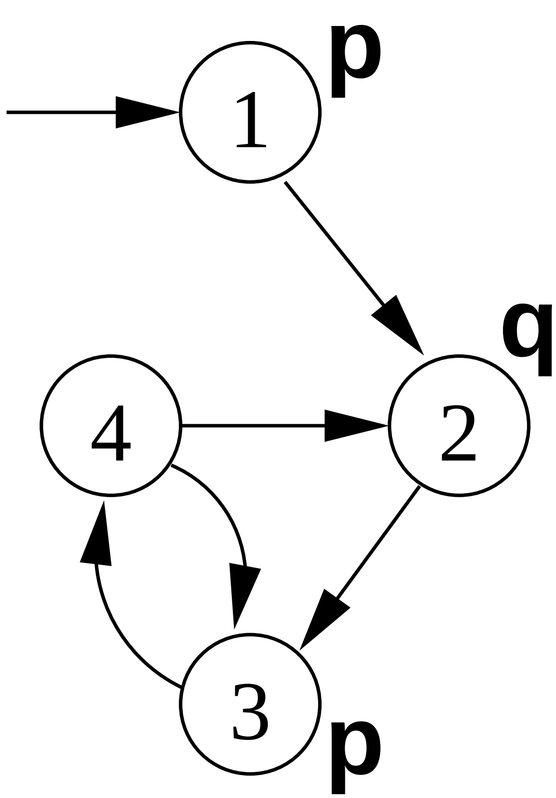
▷ Many successful applications of SAT:
- Boolean circuits
- (Bounded) Planning
- (Bounded) Model Checking
- Cryptography
- Scheduling
- ...

▷ All NP-complete problem can be (polynomially) converted to SAT.

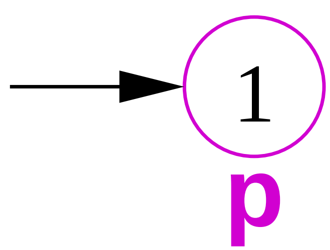
▷ Key issue: find an efficient encoding.

Content

Bounded Model Checking: Example

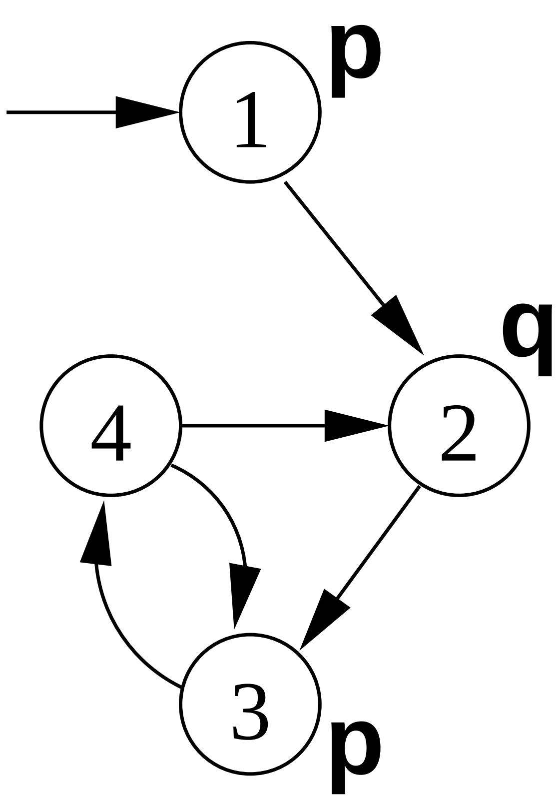


- LTL Formula: **G(p −> Fq)**
- Negated Formula (violation): **F(p & G ! q)**
- $k = 0$:

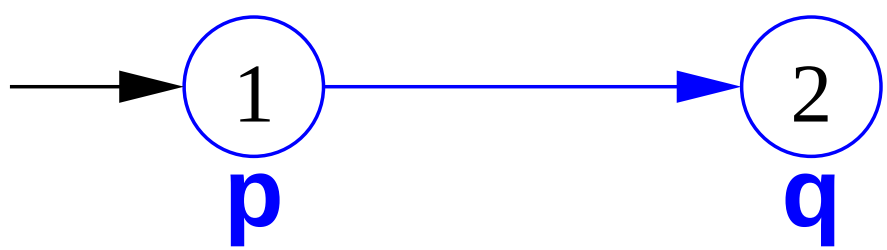


- No counter-example found.

Bounded Model Checking: Example

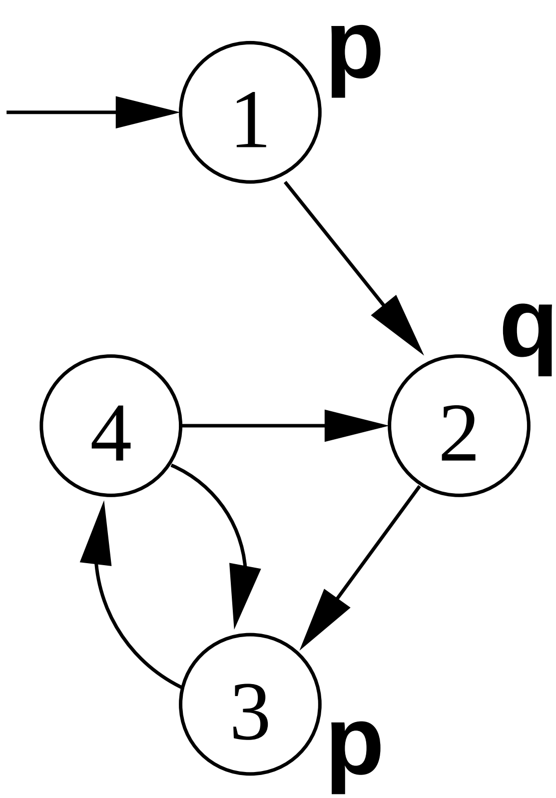


- LTL Formula: **G(p −> Fq)**
- Negated Formula (violation): **F(p & G ! q)**
- $k = 1$:

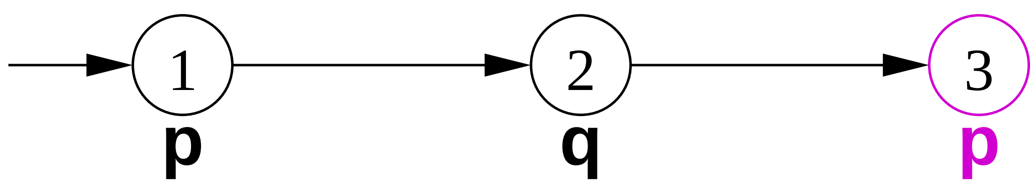


- No counter-example found.

Bounded Model Checking: Example

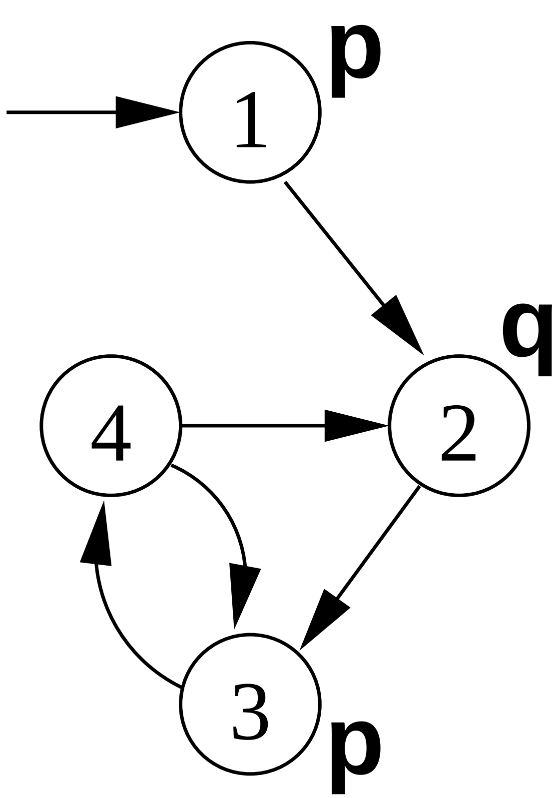


- LTL Formula: **G(p -> Fq)**
- Negated Formula (violation): **F(p & G ! q)**
- $k = 2$:

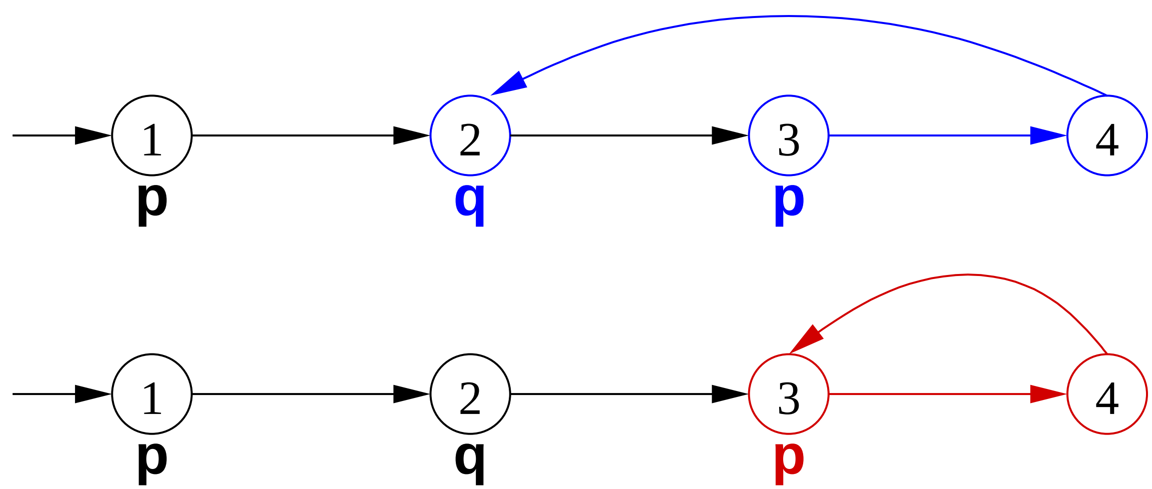


- No counter-example found.

Bounded Model Checking: Example



- LTL Formula: **G(p −> Fq)**
- Negated Formula (violation): **F(p & G ! q)**
- $k = 3$:



- The 2nd trace is a counter-example!

Content

The problem [Biere et al, 1999]

Ingredients:

- A system written as a Kripke structure $M := \langle S, I, T, \mathcal{L} \rangle$
- A property f written as a LTL formula:
- an integer k (bound)

Problem:

- Is there an execution path of M of length k satisfying the temporal property f?:

$$M \models_k \mathbf{E} f$$

- check repeated for increasing values of $k = 1, 2, 3, ...$

The encoding

Equivalent to the satisfiability problem of a boolean formula $[[M, f]]_k$ defined as follows:

$$[[M, f]]_k := [[M]]_k \wedge [[f]]_k \tag{1}$$

$$[[M]]_k := I(s^0) \wedge \bigwedge_{i=0}^{k-1} R(s^i, s^{i+1}), \tag{2}$$

$$[[f]]_k := (\neg \bigvee_{l=0}^{k} R(s^k, s^l) \wedge [[f]]_k^0) \vee \bigvee_{l=0}^{k} (R(s^k, s^l) \wedge {}_l[[f]]_k^0), \tag{3}$$

- the vector s of propositional variables is replicated k+1 times $s^0, s^1, ..., s^k$
- $[[M]]_k$ encodes the fact that the k-path is an execution of M
- $[[f]]_k$ encodes the fact that the k-path satisfies f

The Encoding [cont.]

In general, the encoding for a formula f with k steps

$$[[f]]_k$$

is the disjunction of

- ▷ the constraints needed to express a model without loopback,

$$(\neg(\bigvee_{l=0}^{k} R(s^k, s^l)) \wedge\ [[f]]_k^0)$$

- ▷ the constraints needed to express a given loopback, for all possible points of loopback

$$\bigvee_{l=0}^{k} (R(s^k, s^l) \wedge\ {}_l[[f]]_k^0)$$

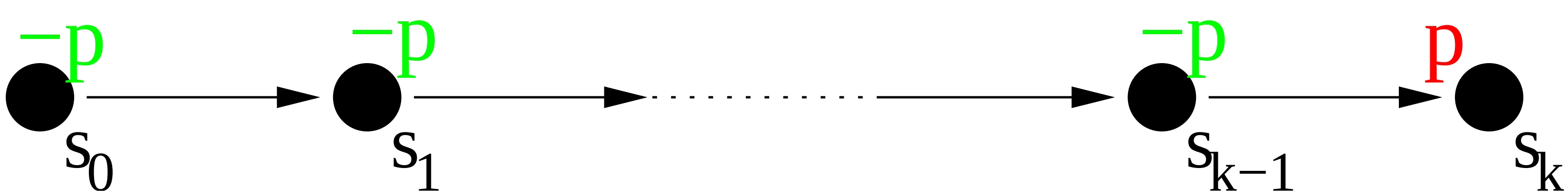
The encoding of $[[f]]_k^i$ and ${}_l[[f]]_k^i$

f	$[[f]]_k^i$	${}_l[[f]]_k^i$
p	p_i	p_i
$\neg p$	$\neg p_i$	$\neg p_i$
$h \wedge g$	$[[h]]_k^i \wedge [[g]]_k^i$	${}_l[[h]]_k^i \wedge {}_l[[g]]_k^i$
$h \vee g$	$[[h]]_k^i \vee [[g]]_k^i$	${}_l[[h]]_k^i \vee {}_l[[g]]_k^i$
$\mathbf{X}g$	$[[g]]_k^{i+1} \quad if\ i < k$ $\perp \quad otherwise.$	${}_l[[g]]_k^{i+1} \quad if\ i < k$ ${}_l[[g]]_k^l \quad otherwise.$
$\mathbf{G}g$	$\perp$	$\bigwedge_{j=min(i,l)}^k {}_l[[g]]_k^j$
$\mathbf{F}g$	$\bigvee_{j=i}^k [[g]]_k^j$	$\bigvee_{j=min(i,l)}^k {}_l[[g]]_k^j$
$h\mathbf{U}g$	$\bigvee_{j=i}^k \left([[g]]_k^j \wedge \bigwedge_{n=i}^{j-1} [[h]]_k^n \right)$	$\bigvee_{j=i}^k \left({}_l[[g]]_k^j \wedge \bigwedge_{n=i}^{j-1} {}_l[[h]]_k^n \right) \vee$ $\bigvee_{j=l}^{i-1} \left({}_l[[g]]_k^j \wedge \bigwedge_{n=i}^{k} {}_l[[h]]_k^n \wedge \bigwedge_{n=l}^{j-1} {}_l[[h]]_k^n \right)$
$h\mathbf{R}g$	$\bigvee_{j=i}^k \left([[h]]_k^j \wedge \bigwedge_{n=i}^{j} [[g]]_k^n \right)$	$\bigwedge_{j=min(i,l)}^k {}_l[[g]]_k^j \vee$ $\bigvee_{j=i}^k \left({}_l[[h]]_k^j \wedge \bigwedge_{n=i}^{j} {}_l[[g]]_k^n \right) \vee$ $\bigvee_{j=l}^{i-1} \left({}_l[[h]]_k^j \wedge \bigwedge_{n=i}^{k} {}_l[[g]]_k^n \wedge \bigwedge_{n=l}^{j} {}_l[[g]]_k^n \right)$

Example: $\mathbf{F}p$ (reachability)

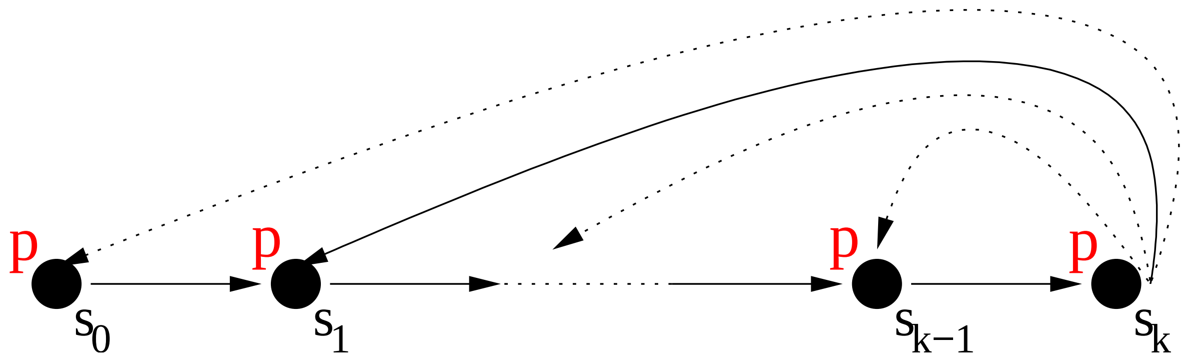
- $f := \mathbf{F}p$: is there a reachable state in which p holds?
- a finite path can show that the property holds
- $[[M, f]]_k$ is:

$$I(s^0) \wedge \bigwedge_{i=0}^{k-1} R(s^i, s^{i+1}) \wedge \bigvee_{j=0}^{k} p^j$$



Example: $\mathbf{G}p$

- $f := \mathbf{G}p$: is there a path where p holds forever?
- We need to produce an infinite behaviour, with a finite number of transitions
- We can do it by imposing that the path loops back

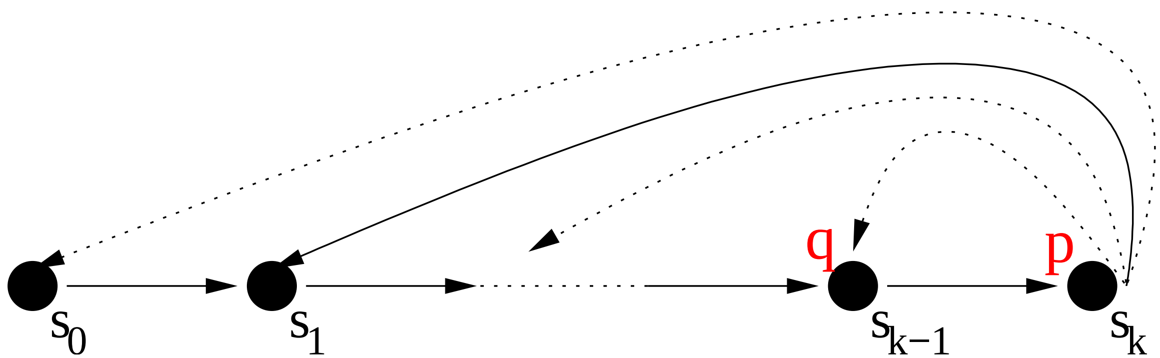


- $[[M, f]]_k$ is:

$$I(s^0) \wedge \bigwedge_{i=0}^{k-1} R(s^i, s^{i+1}) \wedge \bigvee_{l=0}^{k} R(s^k, s^l) \wedge \bigwedge_{j=0}^{k} p^j$$

Example: $\mathbf{GF}q \wedge \mathbf{F}p$ (fair reachability)

- $f := \mathbf{GF}q \wedge \mathbf{F}p$: provided that q holds infinitely often, is there a reachable state in which p holds?
- Again, we need to produce an infinite behaviour, with a finite number of transitions



- $[[M,f]]_k$ is:

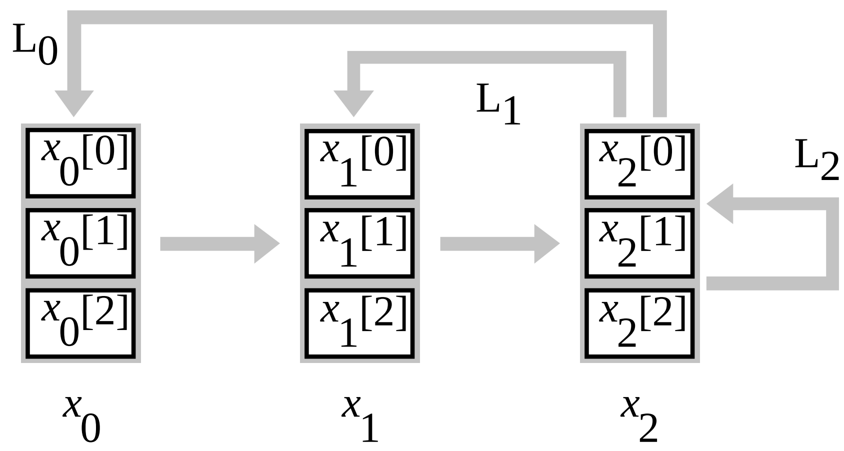
$$I(s^0) \wedge \bigwedge_{i=0}^{k-1} R(s^i, s^{i+1}) \wedge \bigvee_{j=0}^{k} p_j \wedge \bigvee_{l=0}^{k} \left(R(s^k, s^l) \wedge \bigvee_{j=l}^{k} q \right)$$

Example: a bugged 3-bit shift register

- System M:
  - $I(x) := \top$ (arbitrary initial state)
  - Correct R:
    $R(x,x') := (x'[0] \leftrightarrow x[1]) \wedge (x'[1] \leftrightarrow x[2]) \wedge (x'[2] \leftrightarrow 0)$
  - Bugged R:
    $R(x,x') := (x'[0] \leftrightarrow x[1]) \wedge (x'[1] \leftrightarrow x[2]) \wedge (x'[2] \leftrightarrow 1)$
- Property: $\mathbf{AF}(!x[0] \wedge !x[1] \wedge !x[2])$
- BMC Problem: $M \models_k \mathbf{EG}((x[0] \vee x[1] \vee x[2]))$

Example: a bugged 3-bit shift register (cont.)

$k = 2$:



$$[[M]]_2: \quad \left(\begin{array}{l} (x_1[0] \leftrightarrow x_0[1]) \wedge (x_1[1] \leftrightarrow x_0[2]) \wedge (x_1[2] \leftrightarrow 1) \wedge \\ (x_2[0] \leftrightarrow x_1[1]) \wedge (x_2[1] \leftrightarrow x_1[2]) \wedge (x_2[2] \leftrightarrow 1) \end{array} \right) \wedge$$

$$\bigvee_{l=0}^{2} L_l: \quad \left(\begin{array}{l} ((x_0[0] \leftrightarrow x_2[1]) \wedge (x_0[1] \leftrightarrow x_2[2]) \wedge (x_0[2] \leftrightarrow 1)) \vee \\ ((x_1[0] \leftrightarrow x_2[1]) \wedge (x_1[1] \leftrightarrow x_2[2]) \wedge (x_1[2] \leftrightarrow 1)) \vee \\ ((x_2[0] \leftrightarrow x_2[1]) \wedge (x_2[1] \leftrightarrow x_2[2]) \wedge (x_2[2] \leftrightarrow 1)) \end{array} \right) \wedge$$

$$\bigwedge_{i=0}^{2} (x \neq 0): \quad \left(\begin{array}{l} (x_0[0] \vee x_0[1] \vee x_0[2]) \wedge \\ (x_1[0] \vee x_1[1] \vee x_1[2]) \wedge \\ (x_2[0] \vee x_2[1] \vee x_2[2]) \end{array} \right)$$

$\Longrightarrow$ SAT: $x_i[j] := 1 \ \forall i, j$

Bounded Model Checking: summary

▷ incomplete technique:
  - if you find all formulas unsatisfiable, it tells you nothing
  - computing the maximum k (diameter) possible but extremely hard

▷ very efficient for some problems (tipically debugging)

▷ lots of enhancements

▷ current symbolic model checkers embed a SAT based BMC tool

Efficiency Issues in Bounded Model Checking

- ▷ Caching different problems:
  - can we exploit the similarities between problems at k and $k+1$?
- ▷ Simplification of encodings
  - Reduced Boolean Circuits (RBC)
  - Boolean Expression Diagrams (BED)
  - Simplification based on Binary-Clauses Reasoning
- ▷ Extend usage to CTL formulae
- ▷ When can we stop increasing the bound k if we don't find violations?

Content

Basic Bound

Theorem. If $k = |M|$, then $M \models \mathbf{E} f \Longleftrightarrow M \models_k \mathbf{E} f$.

- ▷ $|M|$ is always a bound of k. ($2^{|s|}$ is a bound as well.)
  - $|M|$ huge!
  - not so easy to compute in a symbolic setting.

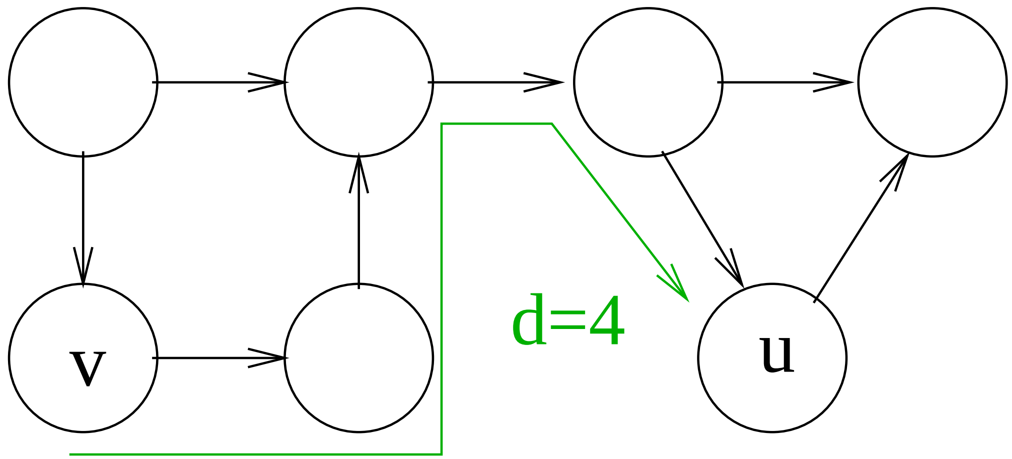
$\Longrightarrow$ need to find better bounds!

The diameter

Diameter: Given M, the **diameter** of M is the minimum integer d s.t. for every path $s_0, \ldots, s_{d+1}$ there exist a path $t_0, \ldots, t_l$ s.t. $l \leq d$, $t_0 = s_0$ and $t_l = s_{d+1}$.

- Intuition: if u is reachable from v, then there is a path from v to u of length d or less.

  $\Longrightarrow$ it is the maximum distance between two states in M.



The diameter: computation

- d is the minimum integer d which makes the following formula true:

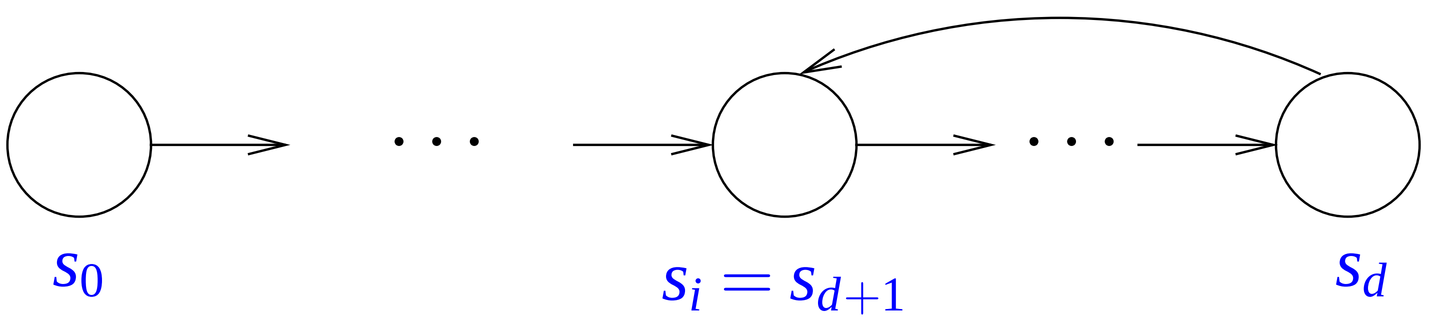
$$\forall s_0,\ldots,s_{d+1}.\exists t_0,\ldots,t_d.\bigwedge_{i=0}^{d} T(s_i,s_{i+1}) \longrightarrow \left(t_0 = s_0 \wedge \bigwedge_{i=0}^{d-1} T(t_i,t_{i+1}) \wedge \bigvee_{i=0}^{d} t_i = s_{d+1}\right)$$

- Quantified boolean formula (QBF): much harder than NP-complete!

The recurrence diameter

Recurrence diameter: Given M, the recurrence diameter of M is the minimum integer d s.t. for every path $s_0, ..., s_{d+1}$ there exist $j \leq d$ s.t. $s_{d+1} = s_j$

- Intuition: the maximum length of a non-loop path

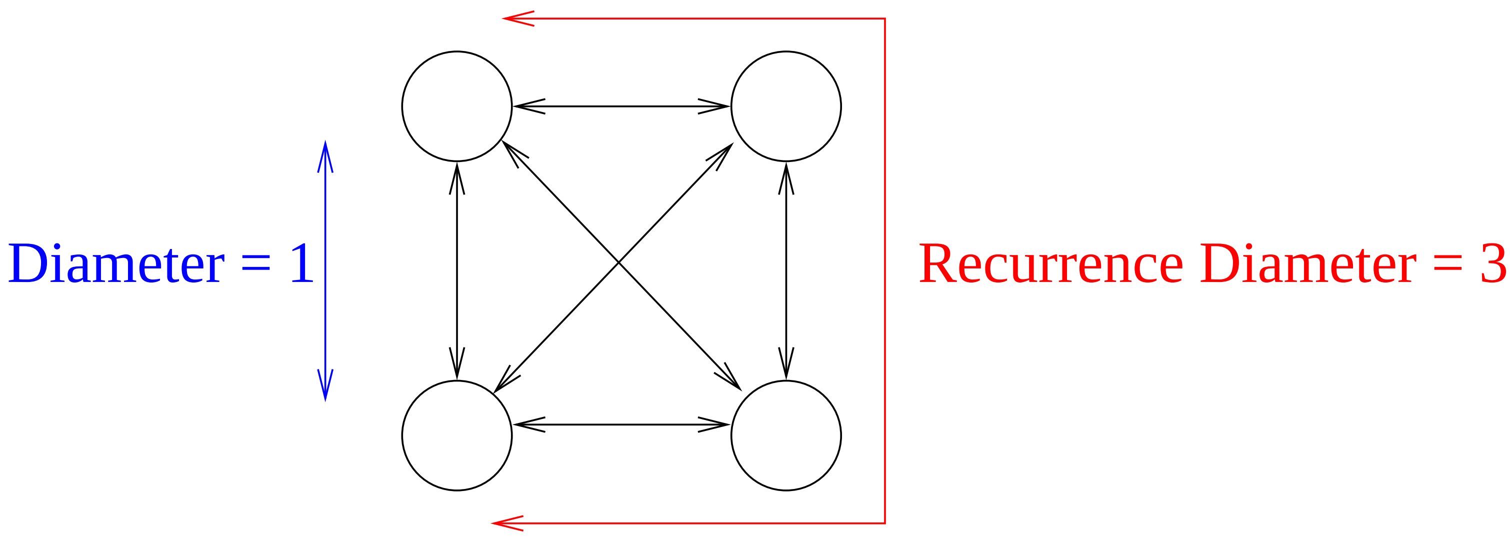


The recurrence diameter: computation

- d is the minimum integer d which makes the following formula true:

$$\forall s_0, \ldots, s_{d+1}. \bigwedge_{i=0}^{d} T(s_i, s_{i+1}) \longrightarrow \bigvee_{i=0}^{d} s_i = s_{d+1}$$

- Validity problem: coNP-complete (solvable by SAT).
- Possibly much longer than the diameter!



Content

Inductive Reasoning on Invariants

1. If all the initial states are good,
2. and if from any good state we only go to good states

$\Rightarrow$ then we can conclude that the system is correct for all reachable states.

SAT-based Inductive Reasoning on Invariants

1. If all the initial states are good
   - $I(s^0) \longrightarrow Good(s^0)$ is valid (its negation is unsatisfiable)
2. if from any good state we only go to good states
   - $(Good(s^k) \wedge R(s^k, s^{k+1})) \longrightarrow Good(s^{k+1})$ is valid (its negation is unsatisfiable)

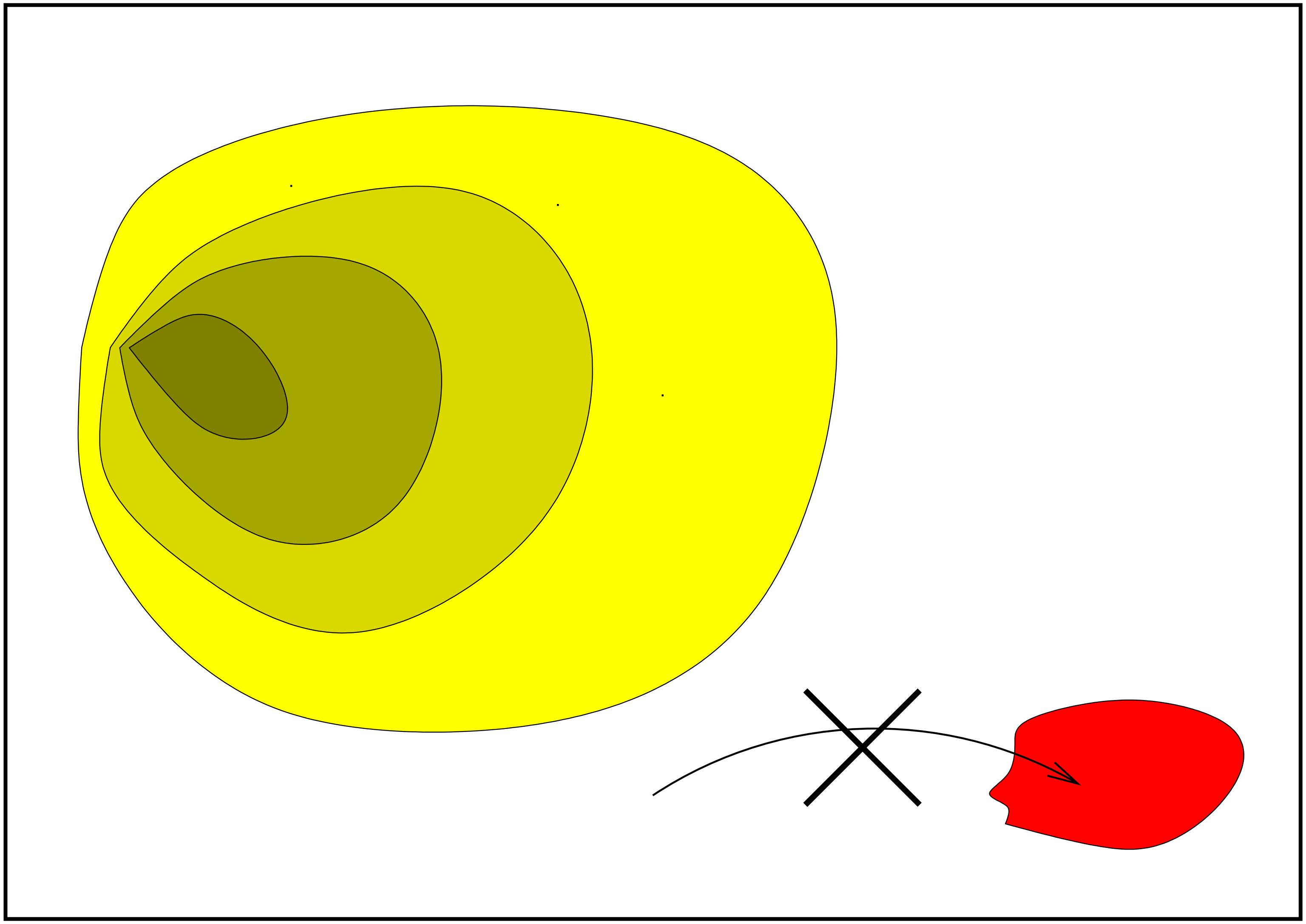
then we can conclude that the system is correct for all reachable states.

$\Longrightarrow$ Check for the unsatisfiability of the boolean formulas:

$$\neg(\ (I(s^0) \longrightarrow Good(s^0))\ );$$

$$\neg(\ (Good(s^k) \wedge R(s^k, s^{k+1})) \longrightarrow Good(s^{k+1})\ )\ )$$

SAT-based Inductive Reasoning on Invariants [cont.]



Strengthening of Invariants

▷ Problem: Induction may fail because of unreachable states:

- if $(Good(s^k) \wedge R(s^k, s^{k+1})) \rightarrow Good(s^{k+1})$ is not valid, this does not mean that the property does not hold
- both s^k and s^{k+1} might be unreachable

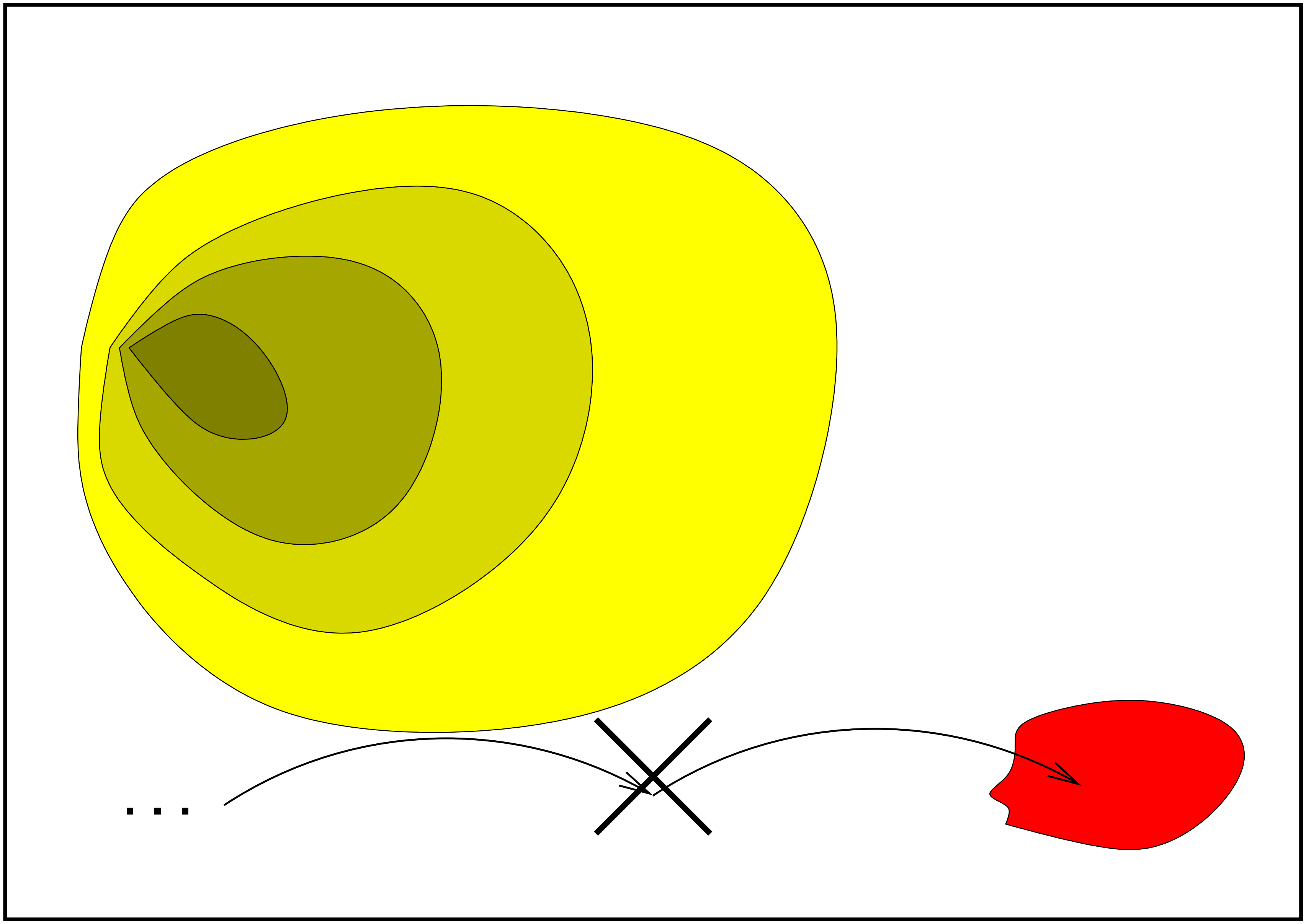
Strengthening of Invariants [cont.]

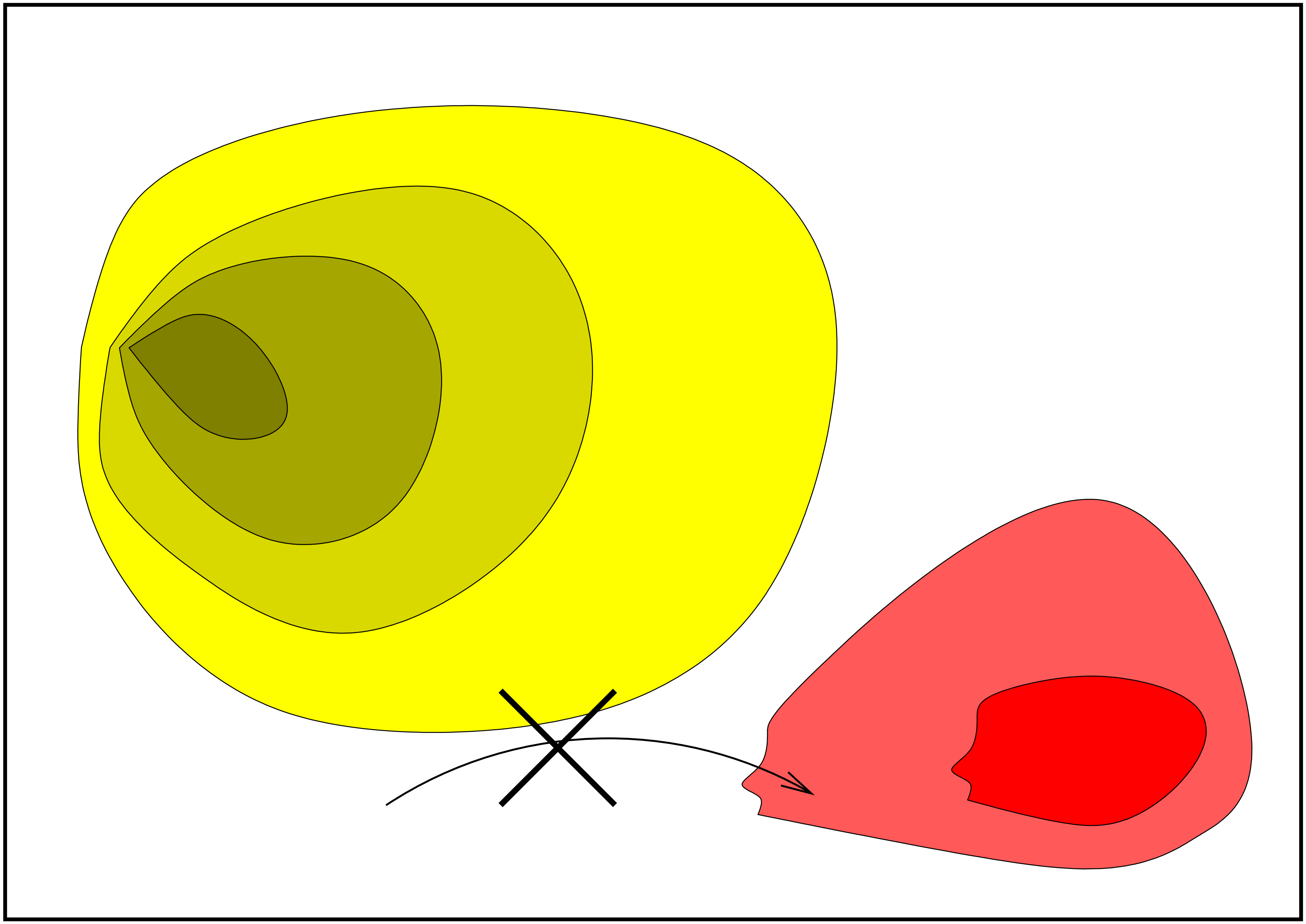
- Solution: increase the depth of induction
  $(Good(s^k) \wedge R(s^k, s^{k+1}) \wedge Good(s^{k+1}) \wedge R(s^{k+1}, s^{k+2})) \longrightarrow Good(s^{k+2})$
  force loop freedom with $\neg(s^i = s^j)$

$\Longrightarrow$ Check for the unsatisfiability of the boolean formulas:

$$\neg(\ (I(s^0) \longrightarrow Good(s^0))\ );$$
$$\neg(\ (Good(s^k) \wedge R(s^k, s^{k+1})) \longrightarrow Good(s^{k+1})\ )\ )$$
$$\neg(\ (Good(s^k) \wedge R(s^k, s^{k+1}) \wedge Good(s^{k+1}) \wedge R(s^{k+1}, s^{k+2})) \longrightarrow Good(s^{k+2})$$

- repeat for increasing values of the gap $1, 2, 3, 4, ...$
- dual to bounded model checking





- ▷ Guess (or, better, infer) ϕ such that $Good \wedge \phi$ is an invariant
- ▷ All the above checks are implementable with SAT technologies

Mixed BMC & Inductive reasoning [Sheeran et al. 2000]

```
function CHECK_PROPERTY (I, R, φ)
    for n := 0, 1, 2, 3, .... do
        if (DPLL(Base_n) == SAT)
            then return PROPERTY_VIOLATED;
        else if (DPLL(Step_n ∧ Unique_n) == UNSAT)
            then return PROPERTY_VERIFIED;
    end for;
```

$$Base_n \quad := \quad I(\mathbf{s}_0) \wedge \bigwedge_{i=0}^{n-1} (R(\mathbf{s}_i, \mathbf{s}_{i+1}) \wedge \varphi(\mathbf{s}_i)) \wedge \neg\varphi(\mathbf{s}_n)$$

$$Step_n \quad := \quad \bigwedge_{i=0}^{n} (R(\mathbf{s}_i, \mathbf{s}_{i+1}) \wedge \varphi(\mathbf{s}_i)) \wedge \neg\varphi(\mathbf{s}_{n+1})$$

$$Unique_n \quad := \quad \bigwedge_{0 \le i \le j \le n} \neg(\mathbf{s}_i = \mathbf{s}_{j+1})$$