Automated Reasoning and Formal Verification
Module II: Formal Verification
Ch. 07: **SAT-Based Model Checking**

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it
URL: https://disi.unitn.it/rseba/DIDATTICA/arfv2026/
Teaching assistant: Gabriele Masina – gabriele.masina@unitn.it

M.S. in Computer Science, Mathematics, & Artificial Intelligence Systems
Academic year 2025-2026

last update: Wednesday 18th February, 2026, 16:36

Outline

1. SAT-based Model Checking: Generalities
2. Bounded Model Checking
   - Intuitions
   - General Encoding
   - Relevant Subcases
   - An Example
   - Computing Upper Bounds
   - Discussion
3. Inductive reasoning on invariants (aka “K-Induction”)
   - K-Induction
   - An Example
4. Exercises

Outline

1. SAT-based Model Checking: Generalities
2. Bounded Model Checking
   - Intuitions
   - General Encoding
   - Relevant Subcases
   - An Example
   - Computing Upper Bounds
   - Discussion
3. Inductive reasoning on invariants (aka “K-Induction”)
   - K-Induction
   - An Example
4. Exercises

SAT-based Model Checking

- Key problems with BDD's:
  - they can explode in space
- A possible alternative:
  - Propositional Satisfiability Checking (SAT)
  - SAT technology is very advanced
- Advantages:
  - reduced memory requirements
  - limited sensitivity: one good setting, does not require expert users
  - much higher capacity (more variables) than BDD based techniques
- Various techniques:
  - Bounded Model Checking (BMC) $\Longrightarrow$ this chapter
  - K-induction $\Longrightarrow$ this chapter
  - Counter-example guided abstraction refinement (CEGAR) $\Longrightarrow$ next chapter
  - Interpolant-based $\Longrightarrow$ not presented in this course
  - IC3/PDR $\Longrightarrow$ not presented in this course
  - ...

SAT-based Model Checking

- Key problems with BDD's:
  - they can explode in space
- A possible alternative:
  - Propositional Satisfiability Checking (SAT)
  - SAT technology is very advanced
- Advantages:
  - reduced memory requirements
  - limited sensitivity: one good setting, does not require expert users
  - much higher capacity (more variables) than BDD based techniques
- Various techniques:
  - Bounded Model Checking (BMC) $\Longrightarrow$ this chapter
  - K-induction $\Longrightarrow$ this chapter
  - Counter-example guided abstraction refinement (CEGAR) $\Longrightarrow$ next chapter
  - Interpolant-based $\Longrightarrow$ not presented in this course
  - IC3/PDR $\Longrightarrow$ not presented in this course
  - ...

SAT-based Model Checking

- Key problems with BDD's:
  - they can explode in space
- A possible alternative:
  - Propositional Satisfiability Checking (SAT)
  - SAT technology is very advanced
- Advantages:
  - reduced memory requirements
  - limited sensitivity: one good setting, does not require expert users
  - much higher capacity (more variables) than BDD based techniques
- Various techniques:
  - Bounded Model Checking (BMC) $\Longrightarrow$ this chapter
  - K-induction $\Longrightarrow$ this chapter
  - Counter-example guided abstraction refinement (CEGAR) $\Longrightarrow$ next chapter
  - Interpolant-based $\Longrightarrow$ not presented in this course
  - IC3/PDR $\Longrightarrow$ not presented in this course
  - ...

SAT-based Model Checking

- Key problems with BDD's:
  - they can explode in space
- A possible alternative:
  - Propositional Satisfiability Checking (SAT)
  - SAT technology is very advanced
- Advantages:
  - reduced memory requirements
  - limited sensitivity: one good setting, does not require expert users
  - much higher capacity (more variables) than BDD based techniques
- Various techniques:
  - Bounded Model Checking (BMC) $\Longrightarrow$ this chapter
  - K-induction $\Longrightarrow$ this chapter
  - Counter-example guided abstraction refinement (CEGAR) $\Longrightarrow$ next chapter
  - Interpolant-based $\Longrightarrow$ not presented in this course
  - IC3/PDR $\Longrightarrow$ not presented in this course
  - ...

SAT-based Bounded Model Checking & K-Induction

Key Ideas:

- BMC: look for counter-example paths of increasing length k
  $\Longrightarrow$ oriented to finding bugs
- K-Induction: look for an induction proofs of increasing length k
  $\Longrightarrow$ oriented to prove correctness
- BMC [resp. K-induction]: for each k, build a Boolean formula that is satisfiable [resp. unsatisfiable] iff there is a counter-example [resp. proof] 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 by a SAT solver
  - can manage complex formulae on up to 10^7 Boolean variables (!)
  - returns satisfying assignment (i.e., a counter-example)
  - exploit incrementality

SAT-based Bounded Model Checking & K-Induction

Key Ideas:

- BMC: look for counter-example paths of increasing length k

  $\Longrightarrow$ oriented to finding bugs
- K-Induction: look for an induction proofs of increasing length k

  $\Longrightarrow$ oriented to prove correctness
- BMC [resp. K-induction]: for each k, build a Boolean formula that is satisfiable [resp. unsatisfiable] iff there is a counter-example [resp. proof] 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 by a SAT solver
  - can manage complex formulae on up to 10^7 Boolean variables (!)
  - returns satisfying assignment (i.e., a counter-example)
  - exploit incrementality

SAT-based Bounded Model Checking & K-Induction

Key Ideas:

- BMC: look for counter-example paths of increasing length k

  $\Longrightarrow$ oriented to finding bugs
- K-Induction: look for an induction proofs of increasing length k

  $\Longrightarrow$ oriented to prove correctness
- BMC [resp. K-induction]: for each k, build a Boolean formula that is satisfiable [resp. unsatisfiable] iff there is a counter-example [resp. proof] 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 by a SAT solver
  - can manage complex formulae on up to 10^7 Boolean variables (!)
  - returns satisfying assignment (i.e., a counter-example)
  - exploit incrementality

SAT-based Bounded Model Checking & K-Induction

Key Ideas:

- BMC: look for counter-example paths of increasing length k

  $\Longrightarrow$ oriented to finding bugs
- K-Induction: look for an induction proofs of increasing length k

  $\Longrightarrow$ oriented to prove correctness
- BMC [resp. K-induction]: for each k, build a Boolean formula that is satisfiable [resp. unsatisfiable] iff there is a counter-example [resp. proof] 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 by a SAT solver
  - can manage complex formulae on up to 10^7 Boolean variables (!)
  - returns satisfying assignment (i.e., a counter-example)
  - exploit incrementality

Outline

1 SAT-based Model Checking: Generalities

2 Bounded Model Checking
- Intuitions
- General Encoding
- Relevant Subcases
- An Example
- Computing Upper Bounds
- Discussion

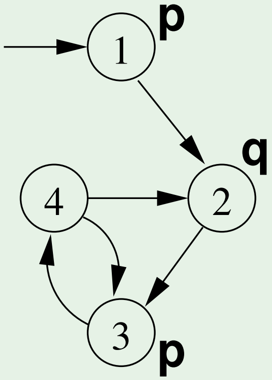
3 Inductive reasoning on invariants (aka “K-Induction”)
- K-Induction
- An Example

4 Exercises

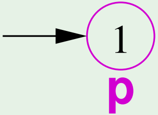
Outline

1. SAT-based Model Checking: Generalities
2. Bounded Model Checking
   - Intuitions
   - General Encoding
   - Relevant Subcases
   - An Example
   - Computing Upper Bounds
   - Discussion
3. Inductive reasoning on invariants (aka “K-Induction”)
   - K-Induction
   - An Example
4. Exercises

Bounded Model Checking: Example

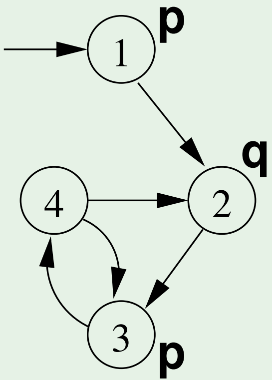


- LTL Formula: $\mathbf{G}(p \rightarrow \mathbf{F}q)$
- Negated Formula (violation): $\mathbf{F}(p \wedge \mathbf{G}\neg q)$
- $k = 0$:

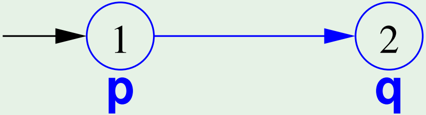


- No counter-example found.

Bounded Model Checking: Example

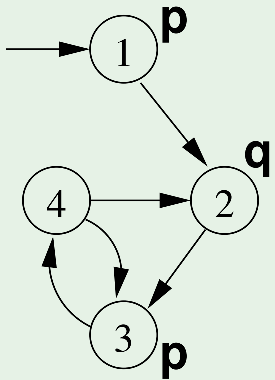


- LTL Formula: $\mathbf{G}(p \rightarrow \mathbf{F}q)$
- Negated Formula (violation): $\mathbf{F}(p \wedge \mathbf{G}\neg q)$
- $k = 1$:

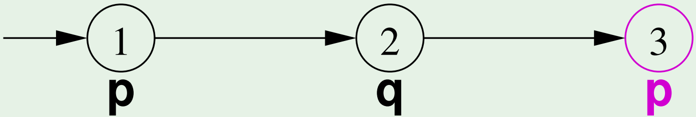


- No counter-example found.

Bounded Model Checking: Example

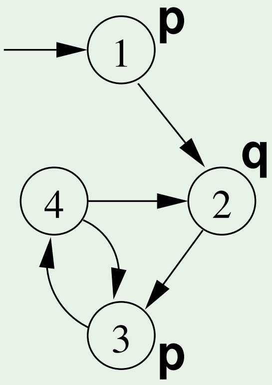


- LTL Formula: $\mathbf{G}(p \rightarrow \mathbf{F}q)$
- Negated Formula (violation): $\mathbf{F}(p \wedge \mathbf{G}\neg q)$
- $k = 2$:

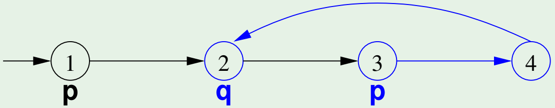


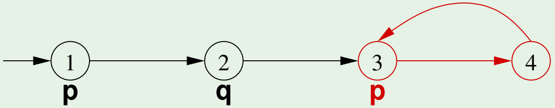
- No counter-example found.

Bounded Model Checking: Example



- LTL Formula: $\mathbf{G}(p \rightarrow \mathbf{F}q)$
- Negated Formula (violation): $\mathbf{F}(p \wedge \mathbf{G}\neg q)$
- $k = 3$:





- The 2nd trace is a counter-example!

Outline

1. SAT-based Model Checking: Generalities
2. Bounded Model Checking
   - Intuitions
   - General Encoding
   - Relevant Subcases
   - An Example
   - Computing Upper Bounds
   - Discussion
3. Inductive reasoning on invariants (aka “K-Induction”)
   - K-Induction
   - An Example
4. Exercises

The problem [Biere et al, 1999]

Ingredients:

Assume states represented by an array s of n Boolean variables

- a system written as a Kripke structure $M := \langle I(s), R(s, s')\rangle$
- a property f written as a LTL formula
- an integer $k \geq 0$ (bound)

Problem

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

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

f is the negation of the property in the LTL model checking problem $M \models \neg f$, and π is a counter-example of length k (bug).

Note: [Biere et al. TACAS 1999] use "$M \models \mathbf{E}f$" as "there exists a path of M verifying f", so that $M \not\models \neg f \iff M \models \mathbf{E}f$

- The check is repeated for increasing values of $k = 0, 1, 2, 3, \ldots$

The problem [Biere et al, 1999]

Ingredients:

Assume states represented by an array s of n Boolean variables

- a system written as a Kripke structure $M := \langle I(s), R(s, s') \rangle$
- a property f written as a LTL formula
- an integer $k \geq 0$ (bound)

Problem

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

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

f is the negation of the property in the LTL model checking problem $M \models \neg f$, and π is a counter-example of length k (bug).

Note: [Biere et al. TACAS 1999] use "$M \models \mathbf{E}f$" as "there exists a path of M verifying f", so that $M \not\models \neg f \iff M \models \mathbf{E}f$

- The check is repeated for increasing values of $k = 0, 1, 2, 3, \ldots$

The problem [Biere et al, 1999]

Ingredients:

Assume states represented by an array s of n Boolean variables

- a system written as a Kripke structure $M := \langle I(s), R(s, s') \rangle$
- a property f written as a LTL formula
- an integer $k \geq 0$ (bound)

Problem

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

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

f is the negation of the property in the LTL model checking problem $M \models \neg f$, and π is a counter-example of length k (bug).

Note: [Biere et al. TACAS 1999] use "$M \models \mathbf{E}f$" as "there exists a path of M verifying f", so that $M \not\models \neg f \iff M \models \mathbf{E}f$

- The check is repeated for increasing values of $k = 0, 1, 2, 3, ...$

The problem [Biere et al, 1999]

Ingredients:

Assume states represented by an array s of n Boolean variables

- a system written as a Kripke structure $M := \langle I(s), R(s, s') \rangle$
- a property f written as a LTL formula
- an integer $k \geq 0$ (bound)

Problem

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

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

f is the negation of the property in the LTL model checking problem $M \models \neg f$, and π is a counter-example of length k (bug).

Note: [Biere et al. TACAS 1999] use "$M \models \mathbf{E} f$" as "there exists a path of M verifying f", so that $M \not\models \neg f \iff M \models \mathbf{E} f$

- The check is repeated for increasing values of $k = 0, 1, 2, 3, ...$

The general encoding

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

$$[[M,f]]_k := [[M]]_k \wedge [[f]]_k$$

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

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

- 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 general encoding

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

$$
\begin{aligned}
[[M, f]]_k &:= [[M]]_k \wedge [[f]]_k \\
[[M]]_k &:= I(s^0) \wedge \bigwedge_{i=0}^{k-1} R(s^i, s^{i+1}), \\
[[f]]_k &:= (\neg \bigvee_{l=0}^{k} R(s^k, s^l) \wedge [[f]]_k^0) \vee \bigvee_{l=0}^{k} (R(s^k, s^l) \wedge {}_l[[f]]_k^0),
\end{aligned}
$$

- 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 general encoding

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

$$
\begin{aligned}
[[M, f]]_k &:= [[M]]_k \wedge [[f]]_k \\
[[M]]_k &:= I(s^0) \wedge \bigwedge_{i=0}^{k-1} R(s^i, s^{i+1}), \\
[[f]]_k &:= (\neg \bigvee_{l=0}^{k} R(s^k, s^l) \wedge [[f]]_k^0) \vee \bigvee_{l=0}^{k} (R(s^k, s^l) \wedge {}_l[[f]]_k^0),
\end{aligned}
$$

- 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 general encoding

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

$$
\begin{aligned}
[[M, f]]_k &:= [[M]]_k \wedge [[f]]_k \\
[[M]]_k &:= I(s^0) \wedge \bigwedge_{i=0}^{k-1} R(s^i, s^{i+1}), \\
[[f]]_k &:= (\neg \bigvee_{l=0}^{k} R(s^k, s^l) \wedge [[f]]_k^0) \vee \bigvee_{l=0}^{k} (R(s^k, s^l) \wedge {}_l[[f]]_k^0),
\end{aligned}
$$

- 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 general encoding [cont.]

The encoding for a formula f with k steps, $[[f]]_k$ is the disjunction of:

- The constraints needed to express a model without loopback:

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

  - $[[f]]_k^i,\ i \in [0, k]$:
    "f holds in s^i under the assumption that $s^0, ..., s^k$ is a no-loopback path"

- The constraints needed to express a model with some loopback:

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

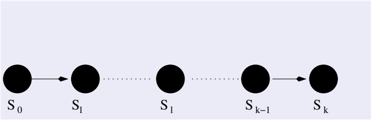
  - ${}_l[[f]]_k^i,\ i \in [0, k]$:
    "f holds in s^i under the assumption that $s^0, ..., s^k$ is a path with a loopback from s^k to s^l"

The general encoding [cont.]

The encoding for a formula f with k steps, $[[f]]_k$ is the disjunction of:

- The constraints needed to express a model without loopback:

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

  - $[[f]]_k^i,\ i \in [0, k]$:
    "f holds in s^i under the assumption that $s^0, \ldots, s^k$ is a no-loopback path"
- The constraints needed to express a model with some loopback:

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

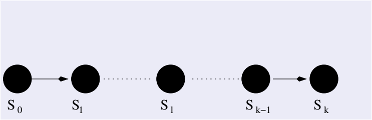
  - ${}_l[[f]]_k^i,\ i \in [0, k]$:
    "f holds in s^i under the assumption that $s^0, \ldots, s^k$ is a path with a loopback from s^k to s^l"

The general encoding [cont.]

The encoding for a formula f with k steps, $[[f]]_k$ is the disjunction of:

- The constraints needed to express a model without loopback:

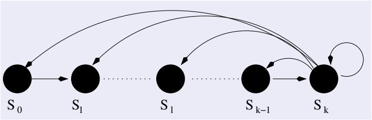
  $$(\neg(\bigvee_{l=0}^{k} R(s^k, s^l)) \wedge [[f]]_k^0)$$

  - $[[f]]_k^i,\ i \in [0, k]$:
    "f holds in s^i under the assumption that $s^0, \ldots, s^k$ is a no-loopback path"

- The constraints needed to express a model with some loopback:

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

  - ${}_l[[f]]_k^i,\ i \in [0, k]$:
    "f holds in s^i under the assumption that $s^0, \ldots, s^k$ is a path with a loopback from s^k to s^l"

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

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

Outline

1 SAT-based Model Checking: Generalities

2 Bounded Model Checking
- Intuitions
- General Encoding
- Relevant Subcases
- An Example
- Computing Upper Bounds
- Discussion

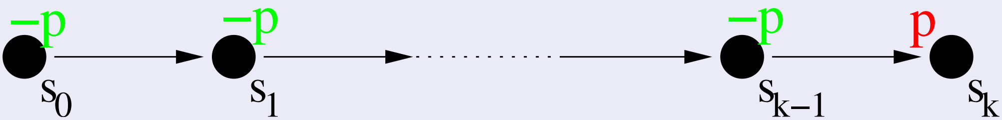
3 Inductive reasoning on invariants (aka “K-Induction”)
- K-Induction
- An Example

4 Exercises

Relevant Subcase: **F**p (reachability)

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

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



Important: incremental encoding

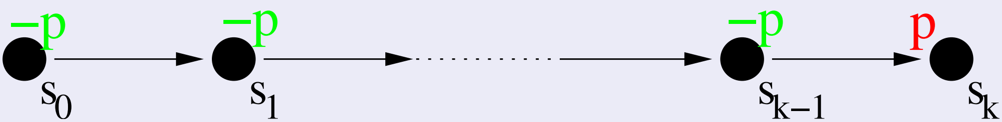
if done for increasing value of k, then it suffices that $[[M, f]]_k$ is:

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

Relevant Subcase: **F**p (reachability)

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

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



Important: incremental encoding

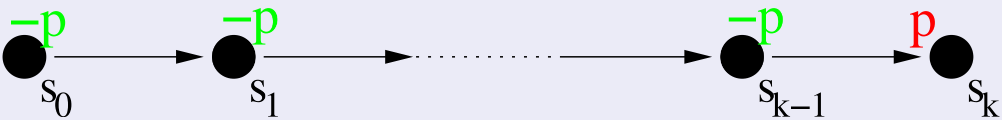
if done for increasing value of k, then it suffices that $[[M, f]]_k$ is:

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

Relevant Subcase: Fp (reachability)

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

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



Important: incremental encoding

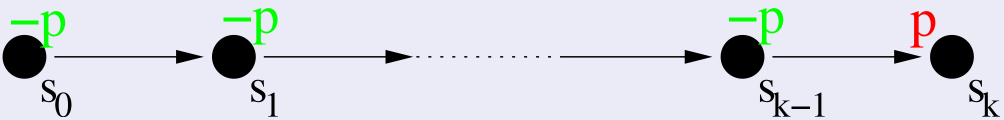
if done for increasing value of k, then it suffices that $[[M, f]]_k$ is:

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

Relevant Subcase: **F**p (reachability)

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

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



Important: incremental encoding

if done for increasing value of k, then it suffices that $[[M, f]]_k$ is:

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

Relevant Subcase: **G**p

- $f := \mathbf{G}p$, s.t. p Boolean: 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:

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

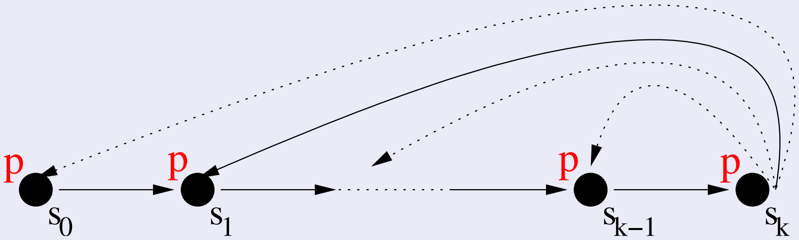
Relevant Subcase: **G**p

- $f := \mathbf{G}p$, s.t. p Boolean: 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:

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

Relevant Subcase: **G***p*

- *f* := **G***p*, s.t. *p* Boolean: 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

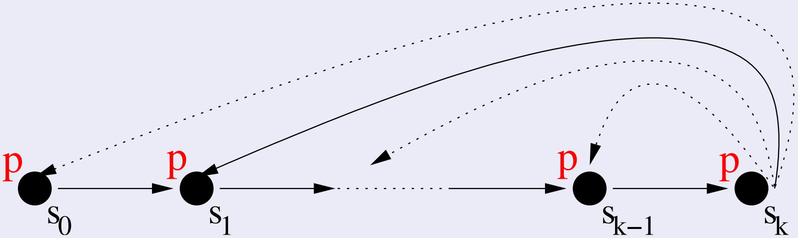


- $[[M, f]]_k$ is:

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

Relevant Subcase: $\mathbf{G}p$

- $f := \mathbf{G}p$, s.t. p Boolean: 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:

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

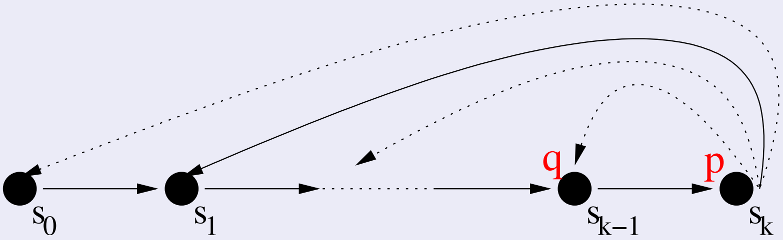
Relevant Subcase: **GF**q (fair states)

- $f := \mathbf{GF}q$, s.t. q Boolean: does q hold infinitely often?
- Again, we need to produce an infinite behaviour, with a finite number of transitions
- $[[M, f]]_k$ is:

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

Relevant Subcase: **GF**q (fair states)

- $f := \mathbf{GF}q$, s.t. q Boolean: does q hold infinitely often?
- Again, we need to produce an infinite behaviour, with a finite number of transitions

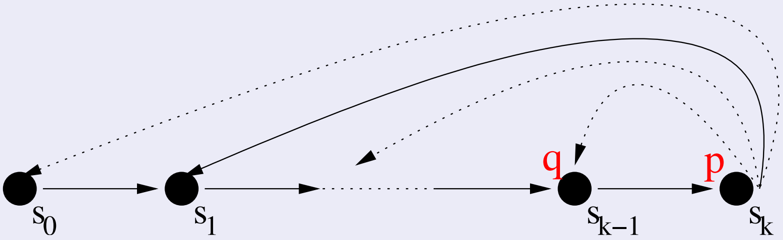


- $[[M, f]]_k$ is:

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

Relevant Subcase: **GF**q (fair states)

- $f :=$ **GF**q, s.t. q Boolean: does q hold infinitely often?
- Again, we need to produce an infinite behaviour, with a finite number of transitions



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

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

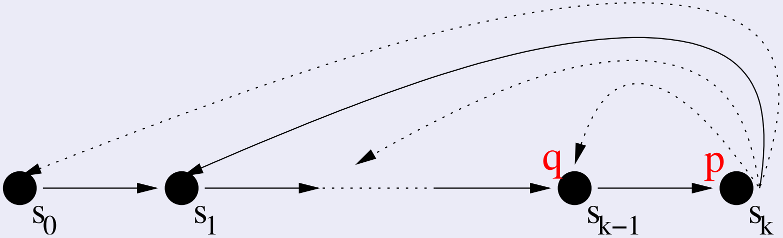
Subcase Combination: $\mathbf{GF}q \wedge \mathbf{F}p$ (fair reachability)

- $f := \mathbf{GF}q \wedge \mathbf{F}p$, s.t. p, q Boolean: 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:

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

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

- $f := \mathbf{GF}q \wedge \mathbf{F}p$, s.t. p, q Boolean: 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

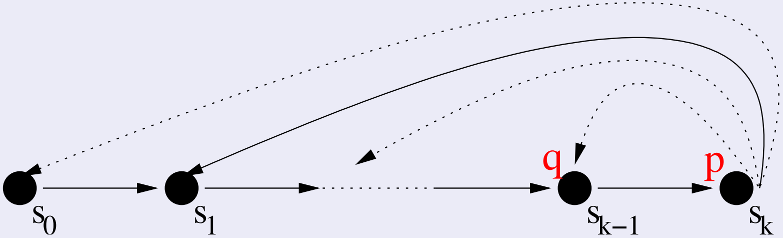


- $[[M, f]]_k$ is:

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

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

- $f := \mathbf{GF}q \wedge \mathbf{F}p$, s.t. p, q Boolean: 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:

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

Outline

1. SAT-based Model Checking: Generalities

2. Bounded Model Checking
   - Intuitions
   - General Encoding
   - Relevant Subcases
   - An Example
   - Computing Upper Bounds
   - Discussion

3. Inductive reasoning on invariants (aka “K-Induction”)
   - K-Induction
   - An Example

4. Exercises

Example: a bugged 3-bit shift register

- System M:
  - $I(x) := \neg x[0] \land \neg x[1] \land x[2]$
  - Correct R: $R(x, x') := (x'[0] \leftrightarrow x[1]) \land (x'[1] \leftrightarrow x[2]) \land (x'[2] \leftrightarrow 0)$
  - Bugged R: $R(x, x') := (x'[0] \leftrightarrow x[1]) \land (x'[1] \leftrightarrow x[2]) \land (x'[2] \leftrightarrow 1)$
- Property: $\mathbf{F}(\neg x[0] \land \neg x[1] \land \neg x[2])$
- BMC Problem: is there an execution π of $\mathcal{M}$ of length k s.t. $\pi \models \mathbf{G}((x[0] \lor x[1] \lor x[2]))$?

Example: a bugged 3-bit shift register

- System M:
  - $I(x) := \neg x[0] \wedge \neg x[1] \wedge x[2]$
  - 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: $\mathbf{F}(\neg x[0] \wedge \neg x[1] \wedge \neg x[2])$
- BMC Problem: is there an execution π of $\mathcal{M}$ of length k s.t. $\pi \models \mathbf{G}((x[0] \vee x[1] \vee x[2]))$?

Example: a bugged 3-bit shift register

- System M:
  - $I(x) := \neg x[0] \wedge \neg x[1] \wedge x[2]$
  - 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: $\mathbf{F}(\neg x[0] \wedge \neg x[1] \wedge \neg x[2])$
- BMC Problem: is there an execution π of $\mathcal{M}$ of length k s.t. $\pi \models \mathbf{G}((x[0] \vee x[1] \vee x[2]))$?

Example: a bugged 3-bit shift register

- System M:
  - $I(x) := \neg x[0] \wedge \neg x[1] \wedge x[2]$
  - 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: $\mathbf{F}(\neg x[0] \wedge \neg x[1] \wedge \neg x[2])$
- BMC Problem: is there an execution π of $\mathcal{M}$ of length k s.t. $\pi \models \mathbf{G}((x[0] \vee x[1] \vee x[2]))$?

Example: a bugged 3-bit shift register

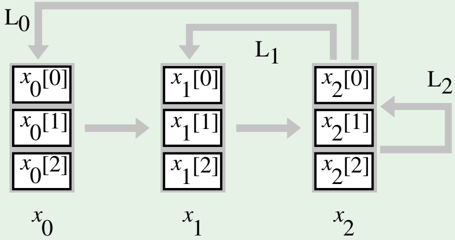
- System M:
  - $I(x) := \neg x[0] \wedge \neg x[1] \wedge x[2]$
  - Correct R: $R(x, x') := (x'[0] \leftrightarrow x[1]) \wedge (x'[1] \leftrightarrow x[2]) \wedge (x'[2] \leftrightarrow 0)$
  - Bugged R: $R(x, x') := (x'[0] \leftrightarrow x[1]) \wedge (x'[1] \leftrightarrow x[2]) \wedge (x'[2] \leftrightarrow 1)$
- Property: $\mathbf{F}(\neg x[0] \wedge \neg x[1] \wedge \neg x[2])$
- BMC Problem: is there an execution π of $\mathcal{M}$ of length k s.t. $\pi \models \mathbf{G}((x[0] \vee x[1] \vee x[2]))$?

Example: a bugged 3-bit shift register

- System M:
  - $I(x) := \neg x[0] \land \neg x[1] \land x[2]$
  - Correct R: $R(x, x') := (x'[0] \leftrightarrow x[1]) \land (x'[1] \leftrightarrow x[2]) \land (x'[2] \leftrightarrow 0)$
  - Bugged R: $R(x, x') := (x'[0] \leftrightarrow x[1]) \land (x'[1] \leftrightarrow x[2]) \land (x'[2] \leftrightarrow 1)$
- Property: $\mathbf{F}(\neg x[0] \land \neg x[1] \land \neg x[2])$
- BMC Problem: is there an execution π of $\mathcal{M}$ of length k s.t. $\pi \models \mathbf{G}((x[0] \lor x[1] \lor x[2]))$?

Example: a bugged 3-bit shift register [cont.]

$k = 0$:



I :	$(\neg x_0[0] \wedge \neg x_0[1] \wedge x_0[2]) \wedge$
$\bigvee_{l=0}^{0} L_l$:	$(\ ((x_0[0] \leftrightarrow x_0[1]) \ \wedge \ (x_0[1] \leftrightarrow x_0[2]) \ \wedge \ (x_0[2] \leftrightarrow 1)) \ ) \ \wedge$
$\bigwedge_{i=0}^{0}(x \neq 0)$:	$(\ (x_0[0] \vee x_0[1] \vee x_0[2]) \ )$

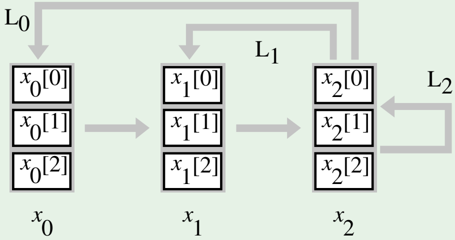
$\Longrightarrow$ UNSAT: unit propagation:

$\neg x_0[0], \neg x_0[1], \quad x_0[2]$

$\Longrightarrow$ loop violated

Example: a bugged 3-bit shift register [cont.]

$k = 0$:



I : $(\neg x_0[0] \wedge \neg x_0[1] \wedge x_0[2]) \wedge$

$\bigvee_{l=0}^{0} L_l$: $(\ ((x_0[0] \leftrightarrow x_0[1]) \wedge (x_0[1] \leftrightarrow x_0[2]) \wedge (x_0[2] \leftrightarrow 1))\ ) \wedge$

$\bigwedge_{i=0}^{0}(x \neq 0)$: $(\ (x_0[0] \vee x_0[1] \vee x_0[2])\ )$

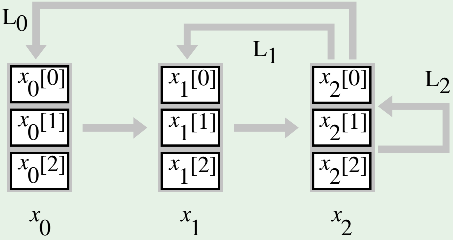
$\Longrightarrow$ UNSAT: unit propagation:

$\neg x_0[0], \neg x_0[1], \quad x_0[2]$

$\Longrightarrow$ loop violated

Example: a bugged 3-bit shift register [cont.]

$k = 0$:



$I:$ $(\neg x_0[0] \wedge \neg x_0[1] \wedge x_0[2]) \wedge$

$\bigvee_{l=0}^{0} L_l :$ $(\ ((x_0[0] \leftrightarrow x_0[1]) \wedge (x_0[1] \leftrightarrow x_0[2]) \wedge (x_0[2] \leftrightarrow 1))\ ) \wedge$

$\bigwedge_{i=0}^{0}(x \neq 0) :$ $(\ (x_0[0] \vee x_0[1] \vee x_0[2])\ )$

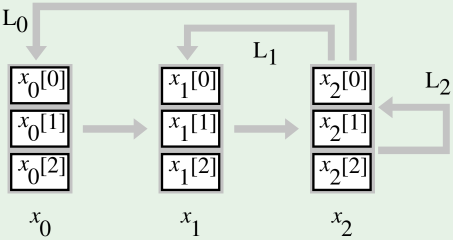
$\Longrightarrow$ UNSAT: unit propagation:

$\neg x_0[0], \neg x_0[1], \quad x_0[2]$

$\Longrightarrow$ loop violated

Example: a bugged 3-bit shift register [cont.]

$k = 1$:



$$
\begin{array}{ll}
I: & (\neg x_0[0] \land \neg x_0[1] \land x_0[2]) \land \\
[[M]]_1: & (\ (x_1[0] \leftrightarrow x_0[1]) \ \land \ (x_1[1] \leftrightarrow x_0[2]) \ \land \ (x_1[2] \leftrightarrow 1) \ ) \ \land \\
\bigvee_{l=0}^{1} L_l: & \left(\begin{array}{l} ((x_0[0] \leftrightarrow x_1[1]) \ \land \ (x_0[1] \leftrightarrow x_1[2]) \ \land \ (x_0[2] \leftrightarrow 1)) \lor \\ ((x_1[0] \leftrightarrow x_1[1]) \ \land \ (x_1[1] \leftrightarrow x_1[2]) \ \land \ (x_1[2] \leftrightarrow 1)) \end{array} \right) \land \\
\bigwedge_{i=0}^{1}(x \neq 0): & \left(\begin{array}{l} (x_0[0] \lor x_0[1] \lor x_0[2]) \land \\ (x_1[0] \lor x_1[1] \lor x_1[2]) \end{array} \right)
\end{array}
$$

$\Longrightarrow$ UNSAT: unit propagation:

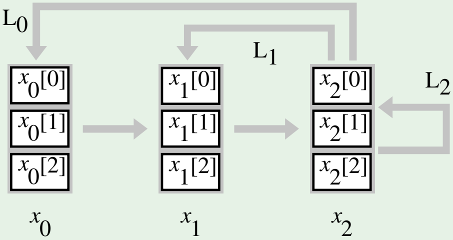
$\neg x_0[0], \neg x_0[1], \quad x_0[2]$

$\neg x_1[0], \quad x_1[1], \quad x_1[2]$

$\Longrightarrow$ both loop disjuncts violated

Example: a bugged 3-bit shift register [cont.]

$k = 1$:



$$
\begin{array}{ll}
I: & (\neg x_0[0] \wedge \neg x_0[1] \wedge x_0[2]) \wedge \\
[[M]]_1: & (\ (x_1[0] \leftrightarrow x_0[1]) \ \wedge\ (x_1[1] \leftrightarrow x_0[2]) \ \wedge\ (x_1[2] \leftrightarrow 1)\ ) \ \wedge \\
\bigvee_{l=0}^{1} L_l: & \left(\begin{array}{l} ((x_0[0] \leftrightarrow x_1[1]) \ \wedge\ (x_0[1] \leftrightarrow x_1[2]) \ \wedge\ (x_0[2] \leftrightarrow 1)) \vee \\ ((x_1[0] \leftrightarrow x_1[1]) \ \wedge\ (x_1[1] \leftrightarrow x_1[2]) \ \wedge\ (x_1[2] \leftrightarrow 1)) \end{array} \right) \wedge \\
\bigwedge_{i=0}^{1}(x \neq 0): & \left(\begin{array}{l} (x_0[0] \vee x_0[1] \vee x_0[2]) \wedge \\ (x_1[0] \vee x_1[1] \vee x_1[2]) \end{array} \right)
\end{array}
$$

$\Longrightarrow$ UNSAT: unit propagation:

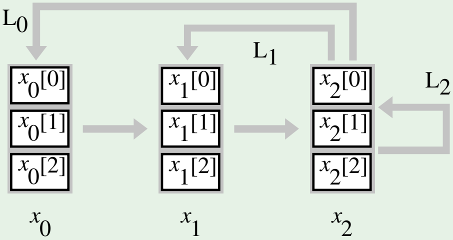
$\neg x_0[0], \neg x_0[1], \quad x_0[2]$

$\neg x_1[0], \quad x_1[1], \quad x_1[2]$

$\Longrightarrow$ both loop disjuncts violated

Example: a bugged 3-bit shift register [cont.]

$k = 1$:



$$
\begin{array}{ll}
I: & (\neg x_0[0] \land \neg x_0[1] \land x_0[2]) \land \\
[[M]]_1: & (\ (x_1[0] \leftrightarrow x_0[1]) \land (x_1[1] \leftrightarrow x_0[2]) \land (x_1[2] \leftrightarrow 1)\ ) \land \\
\bigvee_{l=0}^{1} L_l: & \left(\begin{array}{l} ((x_0[0] \leftrightarrow x_1[1]) \land (x_0[1] \leftrightarrow x_1[2]) \land (x_0[2] \leftrightarrow 1)) \lor \\ ((x_1[0] \leftrightarrow x_1[1]) \land (x_1[1] \leftrightarrow x_1[2]) \land (x_1[2] \leftrightarrow 1)) \end{array} \right) \land \\
\bigwedge_{i=0}^{1} (x \neq 0): & \left(\begin{array}{l} (x_0[0] \lor x_0[1] \lor x_0[2]) \land \\ (x_1[0] \lor x_1[1] \lor x_1[2]) \end{array} \right)
\end{array}
$$

$\Longrightarrow$ UNSAT: unit propagation:

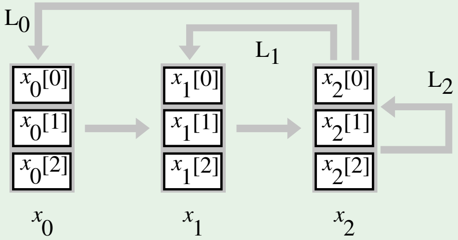
$\neg x_0[0], \neg x_0[1], \quad x_0[2]$

$\neg x_1[0], \quad x_1[1], \quad x_1[2]$

$\Longrightarrow$ both loop disjuncts violated

Example: a bugged 3-bit shift register [cont.]

$k = 2$:



I: $(\neg x_0[0] \land \neg x_0[1] \land x_0[2]) \land$

$[[M]]_2$: $\begin{pmatrix} (x_1[0] \leftrightarrow x_0[1]) \land (x_1[1] \leftrightarrow x_0[2]) \land (x_1[2] \leftrightarrow 1) \land \\ (x_2[0] \leftrightarrow x_1[1]) \land (x_2[1] \leftrightarrow x_1[2]) \land (x_2[2] \leftrightarrow 1) \end{pmatrix} \land$

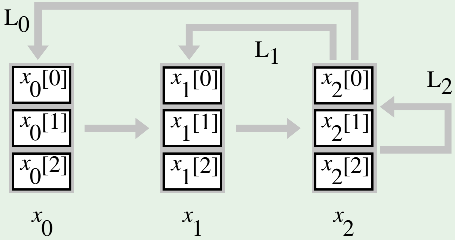
$\bigvee_{l=0}^{2} L_l$: $\begin{pmatrix} ((x_0[0] \leftrightarrow x_2[1]) \land (x_0[1] \leftrightarrow x_2[2]) \land (x_0[2] \leftrightarrow 1)) \lor \\ ((x_1[0] \leftrightarrow x_2[1]) \land (x_1[1] \leftrightarrow x_2[2]) \land (x_1[2] \leftrightarrow 1)) \lor \\ ((x_2[0] \leftrightarrow x_2[1]) \land (x_2[1] \leftrightarrow x_2[2]) \land (x_2[2] \leftrightarrow 1)) \end{pmatrix} \land$

$\bigwedge_{i=0}^{2}(x \neq 0)$: $\begin{pmatrix} (x_0[0] \lor x_0[1] \lor x_0[2]) \land \\ (x_1[0] \lor x_1[1] \lor x_1[2]) \land \\ (x_2[0] \lor x_2[1] \lor x_2[2]) \end{pmatrix}$

$\Longrightarrow$ SAT: $x_0[0] = x_0[1] = x_1[0] = 0;\ x_i[j] := 1\ \forall i, j$

Example: a bugged 3-bit shift register [cont.]

$k = 2$:

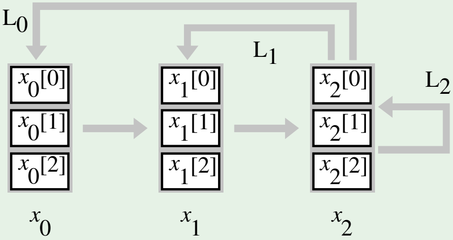


$$
\begin{array}{ll}
I: & (\neg x_0[0] \wedge \neg x_0[1] \wedge x_0[2]) \wedge \\
[[M]]_2: & \left(\begin{array}{l} (x_1[0] \leftrightarrow x_0[1]) \ \wedge \ (x_1[1] \leftrightarrow x_0[2]) \ \wedge \ (x_1[2] \leftrightarrow 1) \ \wedge \\ (x_2[0] \leftrightarrow x_1[1]) \ \wedge \ (x_2[1] \leftrightarrow x_1[2]) \ \wedge \ (x_2[2] \leftrightarrow 1) \end{array} \right) \wedge \\
\bigvee_{l=0}^{2} L_l: & \left(\begin{array}{l} ((x_0[0] \leftrightarrow x_2[1]) \ \wedge \ (x_0[1] \leftrightarrow x_2[2]) \ \wedge \ (x_0[2] \leftrightarrow 1)) \vee \\ ((x_1[0] \leftrightarrow x_2[1]) \ \wedge \ (x_1[1] \leftrightarrow x_2[2]) \ \wedge \ (x_1[2] \leftrightarrow 1)) \vee \\ ((x_2[0] \leftrightarrow x_2[1]) \ \wedge \ (x_2[1] \leftrightarrow x_2[2]) \ \wedge \ (x_2[2] \leftrightarrow 1)) \end{array} \right) \wedge \\
\bigwedge_{i=0}^{2} (x \neq 0): & \left(\begin{array}{l} (x_0[0] \vee x_0[1] \vee x_0[2]) \wedge \\ (x_1[0] \vee x_1[1] \vee x_1[2]) \wedge \\ (x_2[0] \vee x_2[1] \vee x_2[2]) \end{array} \right)
\end{array}
$$

$\Longrightarrow$ SAT: $x_0[0] = x_0[1] = x_1[0] = 0$; $x_i[j] := 1 \ \forall i, j$

Example: a bugged 3-bit shift register [cont.]

$k = 2$:



I : $(\neg x_0[0] \wedge \neg x_0[1] \wedge x_0[2]) \wedge$

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

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

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

$\Longrightarrow$ SAT: $x_0[0] = x_0[1] = x_1[0] = 0;\ x_i[j] := 1\ \forall i, j$

Outline

1. SAT-based Model Checking: Generalities
2. Bounded Model Checking
   - Intuitions
   - General Encoding
   - Relevant Subcases
   - An Example
   - Computing Upper Bounds
   - Discussion
3. Inductive reasoning on invariants (aka “K-Induction”)
   - K-Induction
   - An Example
4. Exercises

Basic bounds for k

Theorem [Biere et al. TACAS 1999]

Let f be a LTL formula.
Then $M \models \mathbf{E}f \iff M \models_k \mathbf{E}f$ for some $k \leq |M| \cdot 2^{|f|}$.

- $|M| \cdot 2^{|f|}$ is always a bound of k.
  - $|M|$ huge!

  $\Longrightarrow$ not so easy to compute in a symbolic setting.

$\Longrightarrow$ need to find better bounds!

Note: [Biere et al. TACAS 1999] use "$M \models \mathbf{E}f$" as "there exists a path of M verifying f", so that $M \not\models \neg f \iff M \models \mathbf{E}f$

Basic bounds for k

Theorem [Biere et al. TACAS 1999]

Let f be a LTL formula.
Then $M \models \mathbf{E}f \iff M \models_k \mathbf{E}f$ for some $k \leq |M| \cdot 2^{|f|}$.

- $|M| \cdot 2^{|f|}$ is always a bound of k.
  - $|M|$ huge!

  $\Longrightarrow$ not so easy to compute in a symbolic setting.

$\Longrightarrow$ need to find better bounds!

Note: [Biere et al. TACAS 1999] use "$M \models \mathbf{E}f$" as "there exists a path of M verifying f", so that $M \not\models \neg f \iff M \models \mathbf{E}f$

Other bounds for k

ACTL & ECTL

- ACTL is a subset of CTL in which "**A**..." (resp. "**E**...") sub-formulas occur only positively (resp. negatively) in each formula. (e.g. $\mathbf{AG}(p \rightarrow \mathbf{AGAF}q)$)
- Many frequently-used LTL properties $\neg f$ have equivalent ACTL representations $\mathbf{A}\neg f'$
  - e.g. $\mathbf{X}q \Longleftrightarrow \mathbf{AX}q$, $\mathbf{G}q \Longleftrightarrow \mathbf{AG}q$, $\mathbf{F}q \Longleftrightarrow \mathbf{AF}q$, $p\mathbf{U}q \Longleftrightarrow \mathbf{A}(p\mathbf{U}q)$, $\mathbf{GF}q \Longleftrightarrow \mathbf{AGAF}q$, $\mathbf{G}(p \rightarrow \mathbf{GF}q) \Longleftrightarrow \mathbf{AG}(p \rightarrow \mathbf{AGAF}q)$
  - ... but not all of them (e.g., $\mathbf{FG} \not\Longleftrightarrow \mathbf{AFAG}p$)
- ECTL is a subset of CTL in which "**E**..." (resp. "**A**...") sub-formulas occur only positively (resp. negatively) in each formula. (e.g. $\mathbf{EF}(p \wedge \mathbf{EFEG}\neg q)$)
- ECTL is the dual subset of ACTL: $\phi \in ECTL \Longleftrightarrow \neg\phi \in ACTL$.

Theorem [Biere et al. TACAS 1999]

Let f be an ECTL formula.
Then $M \models \mathbf{E}f \Longleftrightarrow M \models_k \mathbf{E}f$ for some $k \leq |M|$.

Other bounds for k

ACTL & ECTL

- ACTL is a subset of CTL in which "**A**..." (resp. "**E**...") sub-formulas occur only positively (resp. negatively) in each formula. (e.g. $\mathbf{AG}(p \rightarrow \mathbf{AGAF}q)$)
- Many frequently-used LTL properties $\neg f$ have equivalent ACTL representations $\mathbf{A}\neg f'$
  - e.g. $\mathbf{X}q \Longleftrightarrow \mathbf{AX}q$, $\mathbf{G}q \Longleftrightarrow \mathbf{AG}q$, $\mathbf{F}q \Longleftrightarrow \mathbf{AF}q$, $p\mathbf{U}q \Longleftrightarrow \mathbf{A}(p\mathbf{U}q)$, $\mathbf{GF}q \Longleftrightarrow \mathbf{AGAF}q$, $\mathbf{G}(p \rightarrow \mathbf{GF}q) \Longleftrightarrow \mathbf{AG}(p \rightarrow \mathbf{AGAF}q)$
  - ... but not all of them (e.g., $\mathbf{FG} \not\Longleftrightarrow \mathbf{AFAG}p$)
- ECTL is a subset of CTL in which "**E**..." (resp. "**A**...") sub-formulas occur only positively (resp. negatively) in each formula. (e.g. $\mathbf{EF}(p \wedge \mathbf{EFEG}\neg q)$)
- ECTL is the dual subset of ACTL: $\phi \in ECTL \Longleftrightarrow \neg\phi \in ACTL$.

Theorem [Biere et al. TACAS 1999]

Let f be an ECTL formula.
Then $M \models \mathbf{E}f \Longleftrightarrow M \models_k \mathbf{E}f$ for some $k \leq |M|$.

Other bounds for k (cont)

Theorem [Biere et al. TACAS 1999]

Let p be a Boolean formula and d be the diameter of M.
Then $M \models \mathbf{EF}p \iff M \models_k \mathbf{EF}p$ for some $k \leq d$.

Theorem [Biere et al. TACAS 1999]

Let f be an ECTL formula and d be the recurrence diameter of M.
Then $M \models \mathbf{E}f \iff M \models_k \mathbf{E}f$ for some $k \leq d$.

The diameter

Definition: Diameter

Given M, the diameter of M is the smallest integer d s.t. for every path $s_0, ..., s_{d+1}$ there exist a path $t_0, ..., t_l$ 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

Definition: Diameter

Given M, the diameter of M is the smallest integer d s.t. for every path $s_0, ..., s_{d+1}$ there exist a path $t_0, ..., t_l$ 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.

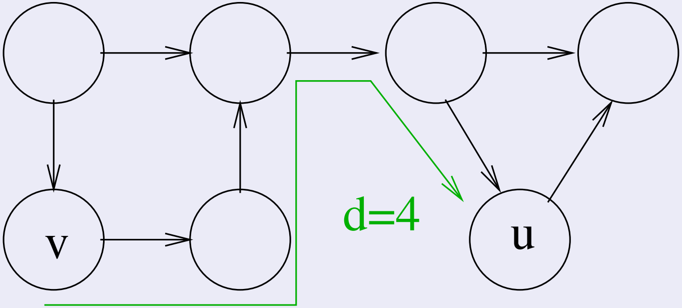
The diameter

Definition: Diameter

Given M, the diameter of M is the smallest integer d s.t. for every path $s_0, ..., s_{d+1}$ there exist a path $t_0, ..., t_l$ 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

Definition: diameter

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

$$\forall s_0, \ldots, s_{d+1} . \exists t_0, \ldots, t_d . \underbrace{\bigwedge_{i=0}^{d} T(s_i, s_{i+1})}_{s_0,\ldots,s_{d+1} \text{ is a path}} \rightarrow \underbrace{\left(t_0 = s_0 \wedge \bigwedge_{i=0}^{d-1} T(t_i, t_{i+1}) \wedge \bigvee_{i=0}^{d} t_i = s_{d+1} \right)}_{t_0,\ldots,t_i \text{ is another path from } s_0 \text{ to } s_{d+1} \text{ for some } i}$$

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

The Diameter: Computation

Definition: diameter

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

$$\forall s_0, \ldots, s_{d+1} . \exists t_0, \ldots, t_d .$$

$$\underbrace{\bigwedge_{i=0}^{d} T(s_i, s_{i+1})}_{s_0,\ldots,s_{d+1} \text{ is a path}} \rightarrow \underbrace{\left(t_0 = s_0 \wedge \bigwedge_{i=0}^{d-1} T(t_i, t_{i+1}) \wedge \bigvee_{i=0}^{d} t_i = s_{d+1} \right)}_{t_0,\ldots,t_i \text{ is another path from } s_0 \text{ to } s_{d+1} \text{ for some } i}$$

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

The recurrence diameter

Definition: recurrence diameter

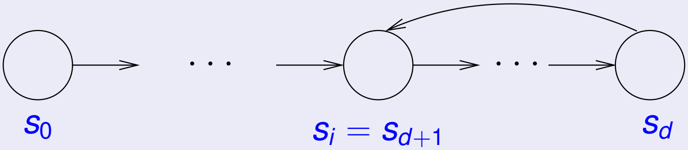
Given M, the recurrence diameter of M is the smallest integer d s.t. for every path $s_0, ..., s_{d+1}$ there exist $j \leq d$ s.t. $s_{d+1} = s_j$.

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

The recurrence diameter

Definition: recurrence diameter

Given M, the recurrence diameter of M is the smallest 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 smallest integer d which makes the following formula true:

$$\forall s_0, \ldots, s_{d+1} . \underbrace{\bigwedge_{i=0}^{d} T(s_i, s_{i+1})}_{s_0,\ldots,s_{d+1} \text{ is a path}} \rightarrow \underbrace{\bigvee_{i=0}^{d} s_i = s_{d+1}}_{s_0,\ldots,s_{d+1} \text{ contains a cicle}}$$

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

The recurrence diameter: computation

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

$$\forall s_0, \ldots, s_{d+1}.\ \underbrace{\bigwedge_{i=0}^{d} T(s_i, s_{i+1})}_{s_0,\ldots,s_{d+1} \text{ is a path}} \rightarrow \underbrace{\bigvee_{i=0}^{d} s_i = s_{d+1}}_{s_0,\ldots,s_{d+1} \text{ contains a cicle}}$$

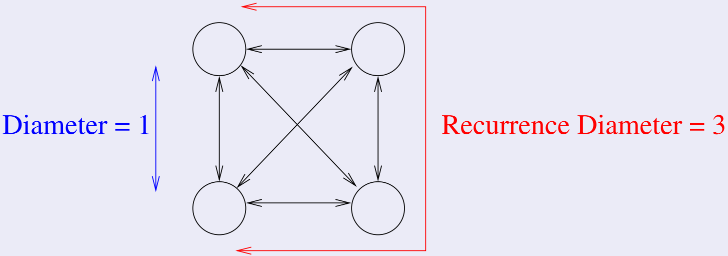
- Validity problem: coNP-complete (solvable by SAT).
- Possibly much longer than the diameter!

The recurrence diameter: computation

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

$$\forall s_0, \ldots, s_{d+1}. \underbrace{\bigwedge_{i=0}^{d} T(s_i, s_{i+1})}_{s_0,\ldots,s_{d+1} \text{ is a path}} \rightarrow \underbrace{\bigvee_{i=0}^{d} s_i = s_{d+1}}_{s_0,\ldots,s_{d+1} \text{ contains a cicle}}$$

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



Outline

1. SAT-based Model Checking: Generalities
2. Bounded Model Checking
   - Intuitions
   - General Encoding
   - Relevant Subcases
   - An Example
   - Computing Upper Bounds
   - Discussion
3. Inductive reasoning on invariants (aka “K-Induction”)
   - K-Induction
   - An Example
4. Exercises

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 (typically debugging)
- Lots of enhancements
- Current symbolic model checkers embed a SAT based BMC tool

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 (typically debugging)
- Lots of enhancements
- Current symbolic model checkers embed a SAT based BMC tool

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 (typically debugging)
- Lots of enhancements
- Current symbolic model checkers embed a SAT based BMC tool

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 (typically debugging)
- Lots of enhancements
- Current symbolic model checkers embed a SAT based BMC tool

Efficiency Issues in Bounded Model Checking

- Incrementality:
  - exploit the similarities between problems at k and $k+1$
- Simplification of encodings
  - Reduced Boolean Circuits (RBC)
  - Boolean Expression Diagrams (BED)
  - And-Inverter Graphs (AIG)
  - Simplification based on Binary-Clauses Reasoning
- Computing bounds not very effective
  $\Longrightarrow$ feasible only on very particular subcases

Efficiency Issues in Bounded Model Checking

- Incrementality:
  - exploit the similarities between problems at k and $k+1$
- Simplification of encodings
  - Reduced Boolean Circuits (RBC)
  - Boolean Expression Diagrams (BED)
  - And-Inverter Graphs (AIG)
  - Simplification based on Binary-Clauses Reasoning
- Computing bounds not very effective
  $\Longrightarrow$ feasible only on very particular subcases

Efficiency Issues in Bounded Model Checking

- Incrementality:
  - exploit the similarities between problems at k and $k + 1$
- Simplification of encodings
  - Reduced Boolean Circuits (RBC)
  - Boolean Expression Diagrams (BED)
  - And-Inverter Graphs (AIG)
  - Simplification based on Binary-Clauses Reasoning
- Computing bounds not very effective
  $\Longrightarrow$ feasible only on very particular subcases

Other Successful SAT-based MC Techniques

- Inductive reasoning on invariants (aka “K-Induction”)
- Counter-example guided abstraction refinement (CEGAR)
  [Clarke et al. CAV 2002]
- Interpolant-based MC
  [Mc Millan, TACAS 2005]
- IC3/PDR
  [Bradley, VMCAI 2011]
- ...

For a survey see e.g.
[Amla et al., CHARME 2005, Prasad et al. STTT 2005].

Outline

1. SAT-based Model Checking: Generalities
2. Bounded Model Checking
   - Intuitions
   - General Encoding
   - Relevant Subcases
   - An Example
   - Computing Upper Bounds
   - Discussion
3. Inductive reasoning on invariants (aka “K-Induction”)
   - K-Induction
   - An Example
4. Exercises

Outline

1. SAT-based Model Checking: Generalities
2. Bounded Model Checking
   - Intuitions
   - General Encoding
   - Relevant Subcases
   - An Example
   - Computing Upper Bounds
   - Discussion
3. Inductive reasoning on invariants (aka “K-Induction”)
   - K-Induction
   - An Example
4. Exercises

Inductive Reasoning on Invariants

Invariant: "**G** *Good*", *Good* being a Boolean formula

(i) If all the initial states are good,

(ii) and if from good states we only go to good states

then the system is correct for all reachable states

Inductive Reasoning on Invariants

Invariant: "**G** *Good*", *Good* being a Boolean formula

(i) If all the initial states are good,

(ii) and if from good states we only go to good states

then the system is correct for all reachable states

Inductive Reasoning on Invariants

Invariant: "**G** *Good*", *Good* being a Boolean formula

(i) If all the initial states are good,

(ii) and if from good states we only go to good states

then the system is correct for all reachable states

Inductive Reasoning on Invariants

Invariant: "$\mathbf{G}\,Good$", *Good* being a Boolean formula

(i) If all the initial states are good,

(ii) and if from good states we only go to good states

then the system is correct for all reachable states

SAT-based Inductive Reasoning on Invariants

(i) If all the initial states are good

- $I(s^0) \rightarrow Good(s^0)$ is valid (i.e. its negation is unsatisfiable)

(ii) if from good states we only go to good states

- $(Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \rightarrow Good(s^k)$ is valid (i.e. its negation is unsatisfiable)

then the system is correct for all reachable states

$\Rightarrow$ Check for the (un)satisfiability of the Boolean formulas:

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

Note

"$(I(s^0) \wedge \neg Good(s^0))$" is step-0 incremental BMC encoding for $\mathbf{F}\neg Good$.

SAT-based Inductive Reasoning on Invariants

(i) If all the initial states are good

- $I(s^0) \rightarrow Good(s^0)$ is valid (i.e. its negation is unsatisfiable)

(ii) if from good states we only go to good states

- $(Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \rightarrow Good(s^k)$ is valid (i.e. its negation is unsatisfiable)

then the system is correct for all reachable states

$\Longrightarrow$ Check for the (un)satisfiability of the Boolean formulas:

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

Note

"$(I(s^0) \wedge \neg Good(s^0))$" is step-0 incremental BMC encoding for $\mathbf{F}\neg Good$.

SAT-based Inductive Reasoning on Invariants

(i) If all the initial states are good

- $I(s^0) \rightarrow Good(s^0)$ is valid (i.e. its negation is unsatisfiable)

(ii) if from good states we only go to good states

- $(Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \rightarrow Good(s^k)$ is valid (i.e. its negation is unsatisfiable)

then the system is correct for all reachable states

$\Rightarrow$ Check for the (un)satisfiability of the Boolean formulas:

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

Note

"$(I(s^0) \wedge \neg Good(s^0))$" is step-0 incremental BMC encoding for $\mathbf{F}\neg Good$.

SAT-based Inductive Reasoning on Invariants

(i) If all the initial states are good

- $I(s^0) \rightarrow Good(s^0)$ is valid (i.e. its negation is unsatisfiable)

(ii) if from good states we only go to good states

- $(Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \rightarrow Good(s^k)$ is valid (i.e. its negation is unsatisfiable)

then the system is correct for all reachable states

$\Rightarrow$ Check for the (un)satisfiability of the Boolean formulas:

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

Note

"$(I(s^0) \wedge \neg Good(s^0))$" is step-0 incremental BMC encoding for $\mathbf{F}\neg Good$.

SAT-based Inductive Reasoning on Invariants

(i) If all the initial states are good

- $I(s^0) \rightarrow Good(s^0)$ is valid (i.e. its negation is unsatisfiable)

(ii) if from good states we only go to good states

- $(Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \rightarrow Good(s^k)$ is valid (i.e. its negation is unsatisfiable)

then the system is correct for all reachable states

⇒ Check for the (un)satisfiability of the Boolean formulas:

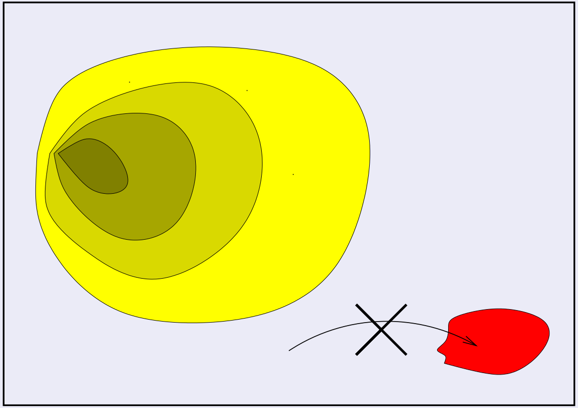
$$(I(s^0) \wedge \neg Good(s^0));$$
$$(Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \wedge \neg Good(s^k)\ )$$

Note

"$(I(s^0) \wedge \neg Good(s^0))$" is step-0 incremental BMC encoding for $\mathbf{F}\neg Good$.

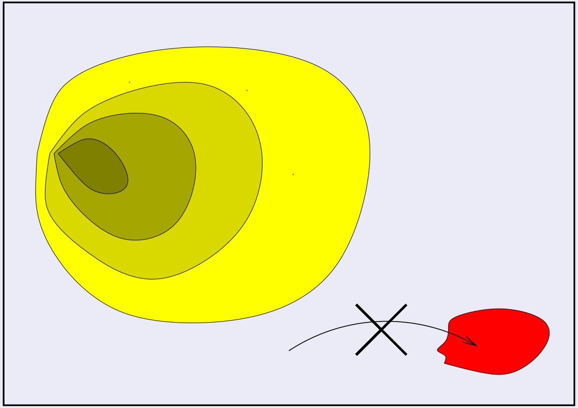
Strengthening of Invariants

- Problem: Induction may fail because of unreachable states:
  - if $(Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \rightarrow Good(s^k)$ is not valid, then this does not mean that the property does not hold
  - both s^{k-1} and s^k might be unreachable



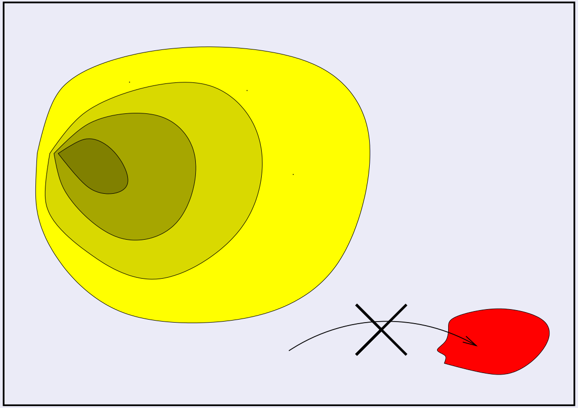
Strengthening of Invariants

- Problem: Induction may fail because of unreachable states:
  - if $(Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \rightarrow Good(s^k)$ is not valid, then this does not mean that the property does not hold
  - both s^{k-1} and s^k might be unreachable



Strengthening of Invariants

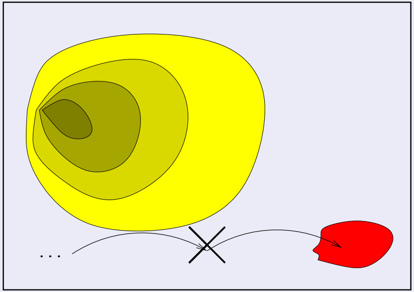
- Problem: Induction may fail because of unreachable states:
  - if $(Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \rightarrow Good(s^k)$ is not valid, then this does not mean that the property does not hold
  - both s^{k-1} and s^k might be unreachable



Strengthening of Invariants [cont.]

Solution (once you know you cannot reach $\neg Good$ in up to 1 step):

- increase the depth of induction
  $(Good(s^{k-2}) \wedge R(s^{k-2}, s^{k-1}) \wedge Good(s^{k-1}) \wedge R(s^{k-1}, s^k) \wedge \neg(s^{k-2} = s^{k-1})) \rightarrow Good(s^k)$

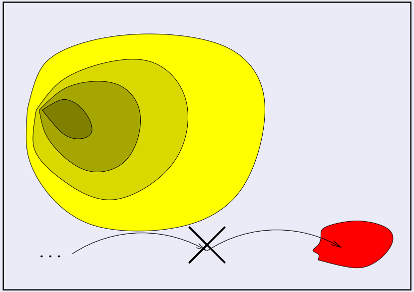


- force loop freedom with $\neg(s^i = s^j)$ for every $i \neq j$ s.t. $i, j \leq k$
- performed after step-1 BMC step returns "unsat":
  $I(s^0) \wedge (R(s^0, s^1) \wedge Good(s^0)) \wedge \neg Good(s^1)$

Strengthening of Invariants [cont.]

Solution (once you know you cannot reach $\neg Good$ in up to 1 step):

- increase the depth of induction
  $(Good(s^{k-2}) \wedge R(s^{k-2}, s^{k-1}) \wedge Good(s^{k-1}) \wedge R(s^{k-1}, s^{k}) \wedge \neg(s^{k-2} = s^{k-1})) \rightarrow Good(s^{k})$

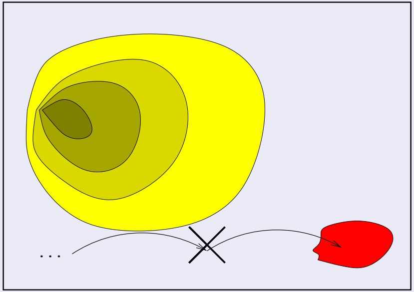


- force loop freedom with $\neg(s^i = s^j)$ for every $i \neq j$ s.t. $i, j \leq k$
- performed after step-1 BMC step returns "unsat":
  $I(s^0) \wedge (R(s^0, s^1) \wedge Good(s^0)) \wedge \neg Good(s^1)$

Strengthening of Invariants [cont.]

Solution (once you know you cannot reach $\neg Good$ in up to 1 step):

- increase the depth of induction
  $(Good(s^{k-2}) \wedge R(s^{k-2}, s^{k-1}) \wedge Good(s^{k-1}) \wedge R(s^{k-1}, s^k) \wedge \neg(s^{k-2} = s^{k-1})) \rightarrow Good(s^k)$



- force loop freedom with $\neg(s^i = s^j)$ for every $i \neq j$ s.t. $i, j \leq k$
- performed after step-1 BMC step returns "unsat":
  $I(s^0) \wedge (R(s^0, s^1) \wedge Good(s^0)) \wedge \neg Good(s^1)$

Strengthening of Invariants [cont.]

$\Longrightarrow$ Check for the [un]satisfiability of the Boolean formulas:

$I(s^0) \wedge \neg Good(s^0);$ $[BMC_0]$

$(Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \wedge \neg Good(s^k);$ $[Kind_0]$

$I(s^0) \wedge (R(s^0, s^1) \wedge Good(s^0)) \wedge \neg Good(s^1);$ $[BMC_1]$

$(Good(s^{k-2}) \wedge R(s^{k-2}, s^{k-1}) \wedge Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \wedge \neg Good(s^k)$
$\wedge \neg (s^{k-2} = s^{k-1});$ $[Kind_1]$

$I(s^0) \wedge (R(s^0, s^1) \wedge Good(s^0) \wedge (R(s^1, s^2) \wedge Good(s^1)) \wedge \neg Good(s^2);$ $[BMC_2]$

...

- Repeat for increasing values of the gap $1, 2, 3, 4, ...$
- Intuition: increasingly tighten the constraint for “spurious” counterexamples: a spurious counterexample must be a chain $s_{k-n}, ..., s_k$ of unreachable and different states s.t. $\neg Good(s_k)$ and $R(s_i, s_{i+1})$, $\forall i$.
- Dual to –and interleaved with– bounded model checking steps
- K-Induction steps can be shifted ($k \stackrel{\text{def}}{=} 0$) to share the subformulas: $\bigwedge_{i=0}^{k-1} (R(s^i, s^{i+1}) \wedge Good(s^i)) \wedge \neg Good(s^{k-2})$

Strengthening of Invariants [cont.]

$\Longrightarrow$ Check for the [un]satisfiability of the Boolean formulas:

$I(s^0) \wedge \neg Good(s^0);$ $[BMC_0]$
$(Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \wedge \neg Good(s^k);$ $[Kind_0]$
$I(s^0) \wedge (R(s^0, s^1) \wedge Good(s^0)) \wedge \neg Good(s^1);$ $[BMC_1]$
$(Good(s^{k-2}) \wedge R(s^{k-2}, s^{k-1}) \wedge Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \wedge \neg Good(s^k)$
$\wedge \neg(s^{k-2} = s^{k-1});$ $[Kind_1]$
$I(s^0) \wedge (R(s^0, s^1) \wedge Good(s^0) \wedge (R(s^1, s^2) \wedge Good(s^1)) \wedge \neg Good(s^2);$ $[BMC_2]$
...

- Repeat for increasing values of the gap $1, 2, 3, 4, \ldots$.
- Intuition: increasingly tighten the constraint for "spurious" counterexamples: a spurious counterexample must be a chain $s_{k-n}, \ldots, s_k$ of unreachable and different states s.t. $\neg Good(s_k)$ and $R(s_i, s_{i+1}), \forall i$.
- Dual to –and interleaved with– bounded model checking steps
- K-Induction steps can be shifted ($k \stackrel{\text{def}}{=} 0$) to share the subformulas: $\bigwedge_{i=0}^{k-1}(R(s^i, s^{i+1}) \wedge Good(s^i)) \wedge \neg Good(s^{k-2})$

Strengthening of Invariants [cont.]

$\Longrightarrow$ Check for the [un]satisfiability of the Boolean formulas:

$I(s^0) \land \neg Good(s^0)$; $[BMC_0]$

$(Good(s^{k-1}) \land R(s^{k-1}, s^k)) \land \neg Good(s^k)$; $[Kind_0]$

$I(s^0) \land (R(s^0, s^1) \land Good(s^0)) \land \neg Good(s^1)$; $[BMC_1]$

$(Good(s^{k-2}) \land R(s^{k-2}, s^{k-1}) \land Good(s^{k-1}) \land R(s^{k-1}, s^k)) \land \neg Good(s^k)$
$\land \neg (s^{k-2} = s^{k-1})$; $[Kind_1]$

$I(s^0) \land (R(s^0, s^1) \land Good(s^0) \land (R(s^1, s^2) \land Good(s^1)) \land \neg Good(s^2)$; $[BMC_2]$

...

- Repeat for increasing values of the gap $1, 2, 3, 4, ...$
- Intuition: increasingly tighten the constraint for "spurious" counterexamples: a spurious counterexample must be a chain $s_{k-n}, ..., s_k$ of unreachable and different states s.t. $\neg Good(s_k)$ and $R(s_i, s_{i+1})$, $\forall i$.
- Dual to –and interleaved with– bounded model checking steps
- K-Induction steps can be shifted ($k \stackrel{\text{def}}{=} 0$) to share the subformulas: $\bigwedge_{i=0}^{k-1} (R(s^i, s^{i+1}) \land Good(s^i)) \land \neg Good(s^{k-2})$

Strengthening of Invariants [cont.]

$\Longrightarrow$ Check for the [un]satisfiability of the Boolean formulas:

$I(s^0) \wedge \neg Good(s^0)$; $[BMC_0]$

$(Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \wedge \neg Good(s^k)$; $[Kind_0]$

$I(s^0) \wedge (R(s^0, s^1) \wedge Good(s^0)) \wedge \neg Good(s^1)$; $[BMC_1]$

$(Good(s^{k-2}) \wedge R(s^{k-2}, s^{k-1}) \wedge Good(s^{k-1}) \wedge R(s^{k-1}, s^k)) \wedge \neg Good(s^k)$
$\wedge \neg(s^{k-2} = s^{k-1})$; $[Kind_1]$

$I(s^0) \wedge (R(s^0, s^1) \wedge Good(s^0) \wedge (R(s^1, s^2) \wedge Good(s^1)) \wedge \neg Good(s^2)$; $[BMC_2]$

...

- Repeat for increasing values of the gap $1, 2, 3, 4, ...$
- Intuition: increasingly tighten the constraint for "spurious" counterexamples: a spurious counterexample must be a chain $s_{k-n}, ..., s_k$ of unreachable and different states s.t. $\neg Good(s_k)$ and $R(s_i, s_{i+1})$, $\forall i$.
- Dual to –and interleaved with– bounded model checking steps
- K-Induction steps can be shifted ($k \stackrel{\text{def}}{=} 0$) to share the subformulas: $\bigwedge_{i=0}^{k-1}(R(s^i, s^{i+1}) \wedge Good(s^i)) \wedge \neg Good(s^{k-2})$

Strengthening of Invariants [cont.]

$\Longrightarrow$ Check for the [un]satisfiability of the Boolean formulas:

$I(s^0) \wedge \neg Good(s^0);\quad [BMC_0]$
$(Good(s^0) \wedge R(s^0, s^1)) \wedge \neg Good(s^1);\quad [Kind_0]$
$I(s^0) \wedge (R(s^0, s^1) \wedge Good(s^0)) \wedge \neg Good(s^1);\quad [BMC_1]$
$(Good(s^0) \wedge R(s^0, s^1) \wedge Good(s^1) \wedge R(s^1, s^2)) \wedge \neg Good(s^2)$
$\wedge \neg(s^0 = s^1);\quad [Kind_1]$
$I(s^0) \wedge (R(s^0, s^1) \wedge Good(s^0) \wedge (R(s^1, s^2) \wedge Good(s^1)) \wedge \neg Good(s^2);\quad [BMC_2]$
...

- Repeat for increasing values of the gap $1, 2, 3, 4, ...$
- Intuition: increasingly tighten the constraint for "spurious" counterexamples: a spurious counterexample must be a chain $s_{k-n}, ..., s_k$ of unreachable and different states s.t. $\neg Good(s_k)$ and $R(s_i, s_{i+1}), \forall i$.
- Dual to –and interleaved with– bounded model checking steps
- K-Induction steps can be shifted ($k \stackrel{\text{def}}{=} 0$) to share the subformulas:
  $\bigwedge_{i=0}^{k-1}(R(s^i, s^{i+1}) \wedge Good(s^i)) \wedge \neg Good(s^{k-2})$

K-Induction Algorithm [Sheeran et al. 2000]

Algorithm

Given:

$$
\begin{aligned}
Base_n &:= I(\mathbf{s}_0) \wedge \bigwedge_{i=0}^{n-1} (R(\mathbf{s}_i, \mathbf{s}_{i+1}) \wedge \varphi(\mathbf{s}_i)) \wedge \neg\varphi(\mathbf{s}_n) \\
Step_n &:= \bigwedge_{i=0}^{n} (R(\mathbf{s}_i, \mathbf{s}_{i+1}) \wedge \varphi(\mathbf{s}_i)) \wedge \neg\varphi(\mathbf{s}_{n+1}) \\
Unique_n &:= \bigwedge_{0 \le i \le j \le n} \neg(\mathbf{s}_i = \mathbf{s}_{j+1})
\end{aligned}
$$

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

$\Longrightarrow$ Reuses previous search if DPLL is incremental!!

K-Induction Algorithm [Sheeran et al. 2000]

Algorithm

Given:

$$\begin{array}{lcl} Base_n & := & I(\mathbf{s}_0) \wedge \bigwedge_{i=0}^{n-1} (R(\mathbf{s}_i, \mathbf{s}_{i+1}) \wedge \varphi(\mathbf{s}_i)) \wedge \neg\varphi(\mathbf{s}_n) \\ Step_n & := & \bigwedge_{i=0}^{n} (R(\mathbf{s}_i, \mathbf{s}_{i+1}) \wedge \varphi(\mathbf{s}_i)) \wedge \neg\varphi(\mathbf{s}_{n+1}) \\ Unique_n & := & \bigwedge_{0 \le i \le j \le n} \neg(\mathbf{s}_i = \mathbf{s}_{j+1}) \end{array}$$

1. **function** CHECK_PROPERTY (I, R, φ)
2. **for** $n := 0, 1, 2, 3, ...$ **do**
3. **if** (DPLL($Base_n$) == SAT)
4. **then return** PROPERTY_VIOLATED;
5. **else if** (DPLL($Step_n \wedge Unique_n$) == UNSAT)
6. **then return** PROPERTY_VERIFIED;
7. **end for**;

$\Longrightarrow$ Reuses previous search if DPLL is incremental!!

K-Induction Algorithm [Sheeran et al. 2000]

Algorithm

Given:

$$
\begin{array}{lll}
Base_n & := & I(\mathbf{s}_0) \wedge \bigwedge_{i=0}^{n-1} (R(\mathbf{s}_i, \mathbf{s}_{i+1}) \wedge \varphi(\mathbf{s}_i)) \wedge \neg\varphi(\mathbf{s}_n) \\
Step_n & := & \bigwedge_{i=0}^{n} (R(\mathbf{s}_i, \mathbf{s}_{i+1}) \wedge \varphi(\mathbf{s}_i)) \wedge \neg\varphi(\mathbf{s}_{n+1}) \\
Unique_n & := & \bigwedge_{0 \le i \le j \le n} \neg(\mathbf{s}_i = \mathbf{s}_{j+1})
\end{array}
$$

1. **function** CHECK_PROPERTY (I, R, φ)
2. **for** $n := 0, 1, 2, 3, ...$ **do**
3. **if** (DPLL($Base_n$) == SAT)
4. **then return** PROPERTY_VIOLATED;
5. **else if** (DPLL($Step_n \wedge Unique_n$) == UNSAT)
6. **then return** PROPERTY_VERIFIED;
7. **end for**;

$\Longrightarrow$ Reuses previous search if DPLL is incremental!!

Outline

1. SAT-based Model Checking: Generalities
2. Bounded Model Checking
   - Intuitions
   - General Encoding
   - Relevant Subcases
   - An Example
   - Computing Upper Bounds
   - Discussion
3. Inductive reasoning on invariants (aka “K-Induction”)
   - K-Induction
   - An Example
4. Exercises

Example: a correct 3-bit shift register

- System M:
  - $I(x) := (\neg x[0] \wedge \neg x[1] \wedge \neg x[2])$
  - $R(x, x') := ((x'[0] \leftrightarrow x[1]) \wedge (x'[1] \leftrightarrow x[2]) \wedge (x'[2] \leftrightarrow 0))$
- Property: $\mathbf{G}\neg x[0]$

Example: a correct 3-bit shift register

- System M:
  - $I(x) := (\neg x[0] \wedge \neg x[1] \wedge \neg x[2])$
  - $R(x, x') := ((x'[0] \leftrightarrow x[1]) \wedge (x'[1] \leftrightarrow x[2]) \wedge (x'[2] \leftrightarrow 0))$
- Property: $\mathbf{G}\neg x[0]$

Example: a correct 3-bit shift register

- System M:
  - $I(x) := (\neg x[0] \wedge \neg x[1] \wedge \neg x[2])$
  - $R(x, x') := ((x'[0] \leftrightarrow x[1]) \wedge (x'[1] \leftrightarrow x[2]) \wedge (x'[2] \leftrightarrow 0))$
- Property: $\mathbf{G}\neg x[0]$

Example: a correct 3-bit shift register

- System M:
  - $I(x) := (\neg x[0] \wedge \neg x[1] \wedge \neg x[2])$
  - $R(x, x') := ((x'[0] \leftrightarrow x[1]) \wedge (x'[1] \leftrightarrow x[2]) \wedge (x'[2] \leftrightarrow 0))$
- Property: $\mathbf{G}\neg x[0]$

Example: a correct 3-bit shift register [cont.]

- Init (BMC Step 0): $((\neg x^0[0] \land \neg x^0[1] \land \neg x^0[2]) \land x^0[0]) \Longrightarrow$ unsat
- K-Induction Step 1:

$$\left(\begin{array}{l}(\neg x^0[0] \land ((x^1[0] \leftrightarrow x^0[1]) \land (x^1[1] \leftrightarrow x^0[2]) \land (x^1[2] \leftrightarrow 0))) \\ \quad \land\, x^1[0]\end{array}\right)$$

$\Longrightarrow$ (partly by unit-propagation)

sat: $\left\{\begin{array}{ccc}\neg x^0[0], & x^0[1], & x^0[2], \\ x^1[0], & x^1[1], & \neg x^1[2]\end{array}\right\}$

$\Longrightarrow$ not proved

Remark

Both $\{\neg x^0[0], \; x^0[1], \; x^0[2])\}$ and $\{\; x^1[0], \; x^1[1], \neg x^1[2]\}$ are non-reachable.

Example: a correct 3-bit shift register [cont.]

- Init (BMC Step 0): $((\neg x^0[0] \land \neg x^0[1] \land \neg x^0[2]) \land x^0[0]) \Longrightarrow$ unsat
- K-Induction Step 1:

$$\left(\begin{array}{l} (\neg x^0[0] \land ((x^1[0] \leftrightarrow x^0[1]) \land (x^1[1] \leftrightarrow x^0[2]) \land (x^1[2] \leftrightarrow 0))) \\ \quad \land\, x^1[0] \end{array} \right)$$

$\Longrightarrow$ (partly by unit-propagation)

sat: $\left\{ \begin{array}{ccc} \neg x^0[0], & x^0[1], & x^0[2], \\ x^1[0], & x^1[1], & \neg x^1[2] \end{array} \right\}$

$\Longrightarrow$ not proved

Remark

Both $\{\neg x^0[0], \ x^0[1], \ x^0[2])\}$ and $\{ \ x^1[0], \ x^1[1], \neg x^1[2]\}$ are non-reachable.

Example: a correct 3-bit shift register [cont.]

- Init (BMC Step 0): $((\neg x^0[0] \wedge \neg x^0[1] \wedge \neg x^0[2]) \wedge x^0[0]) \Longrightarrow$ unsat
- K-Induction Step 1:

$$\left(\begin{array}{l} (\neg x^0[0] \wedge ((x^1[0] \leftrightarrow x^0[1]) \wedge (x^1[1] \leftrightarrow x^0[2]) \wedge (x^1[2] \leftrightarrow 0))) \\ \quad \wedge\, x^1[0] \end{array} \right)$$

$\Longrightarrow$ (partly by unit-propagation)

sat: $\left\{ \begin{array}{ccc} \neg x^0[0], & x^0[1], & x^0[2], \\ x^1[0], & x^1[1], & \neg x^1[2] \end{array} \right\}$

$\Longrightarrow$ not proved

Remark

Both $\{\neg x^0[0],\ x^0[1],\ x^0[2]\}\}$ and $\{\ x^1[0],\ x^1[1], \neg x^1[2]\}$ are non-reachable.

Example: a correct 3-bit shift register [cont.]

- Init (BMC Step 0): $((\neg x^0[0] \land \neg x^0[1] \land \neg x^0[2]) \land x^0[0]) \Longrightarrow$ unsat
- K-Induction Step 1:

$$\left(\begin{array}{l} (\neg x^0[0] \land ((x^1[0] \leftrightarrow x^0[1]) \land (x^1[1] \leftrightarrow x^0[2]) \land (x^1[2] \leftrightarrow 0))) \\ \quad \land\, x^1[0] \end{array} \right)$$

$\Longrightarrow$ (partly by unit-propagation)

sat: $\left\{ \begin{array}{ccc} \neg x^0[0], & x^0[1], & x^0[2], \\ x^1[0], & x^1[1], & \neg x^1[2] \end{array} \right\}$

$\Longrightarrow$ not proved

Remark

Both $\{\neg x^0[0], \; x^0[1], \; x^0[2])\}$ and $\{ \; x^1[0], \; x^1[1], \neg x^1[2]\}$ are non-reachable.

Example: a correct 3-bit shift register [cont.]

- Init (BMC Step 0): $((\neg x^0[0] \land \neg x^0[1] \land \neg x^0[2]) \land x^0[0]) \Longrightarrow$ unsat
- K-Induction Step 1:

$$\left(\begin{array}{l} (\neg x^0[0] \land ((x^1[0] \leftrightarrow x^0[1]) \land (x^1[1] \leftrightarrow x^0[2]) \land (x^1[2] \leftrightarrow 0))) \\ \quad \land\, x^1[0] \end{array} \right)$$

$\Longrightarrow$ (partly by unit-propagation)

sat: $\left\{ \begin{array}{ccc} \neg x^0[0], & x^0[1], & x^0[2], \\ x^1[0], & x^1[1], & \neg x^1[2] \end{array} \right\}$

$\Longrightarrow$ not proved

Remark

Both $\{\neg x^0[0],\ x^0[1],\ x^0[2])\}$ and $\{\ x^1[0],\ x^1[1], \neg x^1[2]\}$ are non-reachable.

Example: a correct 3-bit shift register [cont.]

- BMC Step 1: (...)$\Longrightarrow$ unsat
- K-Induction Step 2:

$$\left(\begin{array}{l} (\neg x^0[0] \wedge ((x^1[0] \leftrightarrow x^0[1]) \wedge (x^1[1] \leftrightarrow x^0[2]) \wedge (x^1[2] \leftrightarrow 0))\wedge \\ \quad \neg x^1[0] \wedge ((x^2[0] \leftrightarrow x^1[1]) \wedge (x^2[1] \leftrightarrow x^1[2]) \wedge (x^2[2] \leftrightarrow 0)) \\ \;) \wedge x^2[0] \end{array}\right)$$
$$\wedge\neg((x^1[0] \leftrightarrow x^0[0]) \wedge (x^1[1] \leftrightarrow x^0[1]) \wedge (x^1[2] \leftrightarrow x^0[2]))$$

$$\Longrightarrow \text{ sat: } \left\{\begin{array}{rrr} \neg x^0[0], & \neg x^0[1], & x^0[2] \\ \neg x^1[0], & x^1[1], & \neg x^1[2] \\ x^2[0], & \neg x^2[1], & \neg x^2[2] \end{array}\right\} \Longrightarrow \text{ not proved}$$

Remark

$\{\neg x^0[0], \neg x^0[1], \; x^0[2]\}$, $\{\neg x^1[0], \; x^1[1], \neg x^1[2]\}$, and $\{\; x^2[0], \neg x^2[1], \neg x^2[2]\}$ are non-reachable.

Example: a correct 3-bit shift register [cont.]

- BMC Step 1: (...)$\Longrightarrow$ unsat
- K-Induction Step 2:

$$\left(\begin{array}{l} (\neg x^0[0] \land ((x^1[0] \leftrightarrow x^0[1]) \land (x^1[1] \leftrightarrow x^0[2]) \land (x^1[2] \leftrightarrow 0)) \land \\ \quad \neg x^1[0] \land ((x^2[0] \leftrightarrow x^1[1]) \land (x^2[1] \leftrightarrow x^1[2]) \land (x^2[2] \leftrightarrow 0)) \\ \ ) \land x^2[0] \end{array}\right)$$
$$\land \neg((x^1[0] \leftrightarrow x^0[0]) \land (x^1[1] \leftrightarrow x^0[1]) \land (x^1[2] \leftrightarrow x^0[2]))$$

$$\Longrightarrow \text{ sat: } \left\{\begin{array}{ccc} \neg x^0[0], & \neg x^0[1], & x^0[2] \\ \neg x^1[0], & x^1[1], & \neg x^1[2] \\ x^2[0], & \neg x^2[1], & \neg x^2[2] \end{array}\right\} \Longrightarrow \text{ not proved}$$

Remark

$\{\neg x^0[0], \neg x^0[1], \ x^0[2]\}$, $\{\neg x^1[0], \ x^1[1], \neg x^1[2]\}$, and $\{\ x^2[0], \neg x^2[1], \neg x^2[2]\}$ are non-reachable.

Example: a correct 3-bit shift register [cont.]

- BMC Step 1: (...)$\Longrightarrow$ unsat
- K-Induction Step 2:

$$\left(\begin{array}{l} (\neg x^0[0] \wedge ((x^1[0] \leftrightarrow x^0[1]) \wedge (x^1[1] \leftrightarrow x^0[2]) \wedge (x^1[2] \leftrightarrow 0))\wedge \\ \quad \neg x^1[0] \wedge ((x^2[0] \leftrightarrow x^1[1]) \wedge (x^2[1] \leftrightarrow x^1[2]) \wedge (x^2[2] \leftrightarrow 0)) \\ \ ) \wedge x^2[0] \end{array}\right)$$
$$\wedge \neg((x^1[0] \leftrightarrow x^0[0]) \wedge (x^1[1] \leftrightarrow x^0[1]) \wedge (x^1[2] \leftrightarrow x^0[2]))$$

$$\Longrightarrow \text{ sat: } \left\{\begin{array}{ccc} \neg x^0[0], & \neg x^0[1], & x^0[2] \\ \neg x^1[0], & x^1[1], & \neg x^1[2] \\ x^2[0], & \neg x^2[1], & \neg x^2[2] \end{array}\right\} \Longrightarrow \text{ not proved}$$

Remark

$\{\neg x^0[0], \neg x^0[1], \ x^0[2]\}$, $\{\neg x^1[0], \ x^1[1], \neg x^1[2]\}$, and $\{\ x^2[0], \neg x^2[1], \neg x^2[2]\}$ are non-reachable.

Example: a correct 3-bit shift register [cont.]

- BMC Step 2: (...) $\Longrightarrow$ unsat
- K-Induction Step 3:

$$
\left(
\begin{array}{l}
(\neg x^0[0] \wedge ((x^1[0] \leftrightarrow x^0[1]) \wedge (x^1[1] \leftrightarrow x^0[2]) \wedge (x^1[2] \leftrightarrow 0))\wedge \\
\quad \neg x^1[0] \wedge ((x^2[0] \leftrightarrow x^1[1]) \wedge (x^2[1] \leftrightarrow x^1[2]) \wedge (x^2[2] \leftrightarrow 0))\wedge \\
\quad \neg x^2[0] \wedge ((x^3[0] \leftrightarrow x^2[1]) \wedge (x^3[1] \leftrightarrow x^2[2]) \wedge (x^3[2] \leftrightarrow 0)) \\
\ ) \wedge x^3[0]
\end{array}
\right)
$$

$$
\begin{array}{l}
\wedge\neg((x^1[0] \leftrightarrow x^0[0]) \wedge (x^1[1] \leftrightarrow x^0[1]) \wedge (x^1[2] \leftrightarrow x^0[2])) \\
\wedge\neg((x^2[0] \leftrightarrow x^0[0]) \wedge (x^2[1] \leftrightarrow x^0[1]) \wedge (x^2[2] \leftrightarrow x^0[2])) \\
\wedge\neg((x^2[0] \leftrightarrow x^1[0]) \wedge (x^2[1] \leftrightarrow x^1[1]) \wedge (x^2[2] \leftrightarrow x^1[2]))
\end{array}
$$

$\Longrightarrow$ (unit-propagation) $\{x^3[0], x^2[1], x^1[2]\}$

$\Longrightarrow$ unsat

$\Longrightarrow$ proved!

Example: a correct 3-bit shift register [cont.]

- BMC Step 2: (...) $\Longrightarrow$ unsat
- K-Induction Step 3:

$$\left(\begin{array}{l} (\neg x^0[0] \wedge ((x^1[0] \leftrightarrow x^0[1]) \wedge (x^1[1] \leftrightarrow x^0[2]) \wedge (x^1[2] \leftrightarrow 0)) \wedge \\ \quad \neg x^1[0] \wedge ((x^2[0] \leftrightarrow x^1[1]) \wedge (x^2[1] \leftrightarrow x^1[2]) \wedge (x^2[2] \leftrightarrow 0)) \wedge \\ \quad \neg x^2[0] \wedge ((x^3[0] \leftrightarrow x^2[1]) \wedge (x^3[1] \leftrightarrow x^2[2]) \wedge (x^3[2] \leftrightarrow 0)) \\ ) \wedge x^3[0] \end{array}\right)$$
$$\wedge \neg((x^1[0] \leftrightarrow x^0[0]) \wedge (x^1[1] \leftrightarrow x^0[1]) \wedge (x^1[2] \leftrightarrow x^0[2]))$$
$$\wedge \neg((x^2[0] \leftrightarrow x^0[0]) \wedge (x^2[1] \leftrightarrow x^0[1]) \wedge (x^2[2] \leftrightarrow x^0[2]))$$
$$\wedge \neg((x^2[0] \leftrightarrow x^1[0]) \wedge (x^2[1] \leftrightarrow x^1[1]) \wedge (x^2[2] \leftrightarrow x^1[2]))$$

$\Longrightarrow$ (unit-propagation) $\{x^3[0], x^2[1], x^1[2]\}$

$\Longrightarrow$ unsat

$\Longrightarrow$ proved!

Example: a correct 3-bit shift register [cont.]

- BMC Step 2: (...) $\Longrightarrow$ unsat
- K-Induction Step 3:

$$\left(\begin{array}{l} (\neg x^0[0] \wedge ((x^1[0] \leftrightarrow x^0[1]) \wedge (x^1[1] \leftrightarrow x^0[2]) \wedge (x^1[2] \leftrightarrow 0)) \wedge \\ \quad \neg x^1[0] \wedge ((x^2[0] \leftrightarrow x^1[1]) \wedge (x^2[1] \leftrightarrow x^1[2]) \wedge (x^2[2] \leftrightarrow 0)) \wedge \\ \quad \neg x^2[0] \wedge ((x^3[0] \leftrightarrow x^2[1]) \wedge (x^3[1] \leftrightarrow x^2[2]) \wedge (x^3[2] \leftrightarrow 0)) \\ ) \wedge x^3[0] \end{array}\right)$$
$$\wedge \neg((x^1[0] \leftrightarrow x^0[0]) \wedge (x^1[1] \leftrightarrow x^0[1]) \wedge (x^1[2] \leftrightarrow x^0[2]))$$
$$\wedge \neg((x^2[0] \leftrightarrow x^0[0]) \wedge (x^2[1] \leftrightarrow x^0[1]) \wedge (x^2[2] \leftrightarrow x^0[2]))$$
$$\wedge \neg((x^2[0] \leftrightarrow x^1[0]) \wedge (x^2[1] \leftrightarrow x^1[1]) \wedge (x^2[2] \leftrightarrow x^1[2]))$$

$\Longrightarrow$ (unit-propagation) $\{x^3[0], x^2[1], x^1[2]\}$

$\Longrightarrow$ unsat

$\Longrightarrow$ proved!

Example: a correct 3-bit shift register [cont.]

- BMC Step 2: (...) $\Longrightarrow$ unsat
- K-Induction Step 3:

$$
\left(\begin{array}{l}
(\neg x^0[0] \wedge ((x^1[0] \leftrightarrow x^0[1]) \wedge (x^1[1] \leftrightarrow x^0[2]) \wedge (x^1[2] \leftrightarrow 0)) \wedge \\
\quad \neg x^1[0] \wedge ((x^2[0] \leftrightarrow x^1[1]) \wedge (x^2[1] \leftrightarrow x^1[2]) \wedge (x^2[2] \leftrightarrow 0)) \wedge \\
\quad \neg x^2[0] \wedge ((x^3[0] \leftrightarrow x^2[1]) \wedge (x^3[1] \leftrightarrow x^2[2]) \wedge (x^3[2] \leftrightarrow 0)) \\
) \wedge x^3[0]
\end{array}\right)
$$

$$
\begin{array}{l}
\wedge \neg((x^1[0] \leftrightarrow x^0[0]) \wedge (x^1[1] \leftrightarrow x^0[1]) \wedge (x^1[2] \leftrightarrow x^0[2])) \\
\wedge \neg((x^2[0] \leftrightarrow x^0[0]) \wedge (x^2[1] \leftrightarrow x^0[1]) \wedge (x^2[2] \leftrightarrow x^0[2])) \\
\wedge \neg((x^2[0] \leftrightarrow x^1[0]) \wedge (x^2[1] \leftrightarrow x^1[1]) \wedge (x^2[2] \leftrightarrow x^1[2]))
\end{array}
$$

$\Longrightarrow$ (unit-propagation) $\{x^3[0], x^2[1], x^1[2]\}$

$\Longrightarrow$ unsat

$\Longrightarrow$ proved!

Outline

1 SAT-based Model Checking: Generalities

2 Bounded Model Checking
- Intuitions
- General Encoding
- Relevant Subcases
- An Example
- Computing Upper Bounds
- Discussion

3 Inductive reasoning on invariants (aka “K-Induction”)
- K-Induction
- An Example

4 Exercises

Ex: Bounded Model Checking

Given the symbolic representation of a FSM M, expressed in terms of the two Boolean formulas: $I(x, y) \stackrel{\text{def}}{=} \neg x \wedge y$, $T(x, y, x', y') \stackrel{\text{def}}{=} (x' \leftrightarrow (x \leftrightarrow \neg y)) \wedge (y' \leftrightarrow \neg y)$, and the LTL property: $\varphi \stackrel{\text{def}}{=} \neg \mathbf{F}(x \wedge y)$,

Ex: Bounded Model Checking

Given the symbolic representation of a FSM M, expressed in terms of the two Boolean formulas: $I(x, y) \stackrel{\text{def}}{=} \neg x \wedge y$, $T(x, y, x', y') \stackrel{\text{def}}{=} (x' \leftrightarrow (x \leftrightarrow \neg y)) \wedge (y' \leftrightarrow \neg y)$, and the LTL property: $\varphi \stackrel{\text{def}}{=} \neg \mathbf{F}(x \wedge y)$,

1. Write a Boolean formula whose solutions (if any) represent executions of M of length 2 which violate φ.

Ex: Bounded Model Checking

Given the symbolic representation of a FSM M, expressed in terms of the two Boolean formulas: $I(x,y) \stackrel{\text{def}}{=} \neg x \wedge y$, $T(x,y,x',y') \stackrel{\text{def}}{=} (x' \leftrightarrow (x \leftrightarrow \neg y)) \wedge (y' \leftrightarrow \neg y)$, and the LTL property: $\varphi \stackrel{\text{def}}{=} \neg \mathbf{F}(x \wedge y)$,

1. Write a Boolean formula whose solutions (if any) represent executions of M of length 2 which violate φ.

   [Solution: The question corresponds to the Bounded Model Checking problem $M \models_2 \mathbf{E}\,\mathbf{F} f$, s.t. $f(x,y) \stackrel{\text{def}}{=} (x \wedge y)$. Thus we have:

$$
\begin{array}{lcl}
\neg x_0 \wedge y_0 & \wedge & // \; I(x_0,y_0) \wedge \\
(x_1 \leftrightarrow (x_0 \leftrightarrow \neg y_0)) \wedge (y_1 \leftrightarrow \neg y_0) & \wedge & // \; T(x_0,y_0,x_1,y_1) \wedge \\
(x_2 \leftrightarrow (x_1 \leftrightarrow \neg y_1)) \wedge (y_2 \leftrightarrow \neg y_1) & \wedge & // \; T(x_1,y_1,x_2,y_2) \wedge \\
((x_0 \wedge y_0) & \vee & // \; (f(x_0,y_0) \vee \\
(x_1 \wedge y_1) & \vee & // \; f(x_1,y_1) \vee \\
(x_2 \wedge y_2)) & & // \; f(x_2,y_2))
\end{array}
$$

   ]

Ex: Bounded Model Checking

Given the symbolic representation of a FSM M, expressed in terms of the two Boolean formulas: $I(x, y) \stackrel{\text{def}}{=} \neg x \wedge y$, $T(x, y, x', y') \stackrel{\text{def}}{=} (x' \leftrightarrow (x \leftrightarrow \neg y)) \wedge (y' \leftrightarrow \neg y)$, and the LTL property: $\varphi \stackrel{\text{def}}{=} \neg \mathbf{F}(x \wedge y)$,

1. Write a Boolean formula whose solutions (if any) represent executions of M of length 2 which violate φ.

   [Solution: The question corresponds to the Bounded Model Checking problem $M \models_2 \mathbf{E}\,\mathbf{F} f$, s.t. $f(x, y) \stackrel{\text{def}}{=} (x \wedge y)$. Thus we have:

$$
\begin{array}{llll}
\neg x_0 \wedge y_0 & \wedge & // \; I(x_0, y_0) \wedge \\
(x_1 \leftrightarrow (x_0 \leftrightarrow \neg y_0)) \wedge (y_1 \leftrightarrow \neg y_0) & \wedge & // \; T(x_0, y_0, x_1, y_1) \wedge \\
(x_2 \leftrightarrow (x_1 \leftrightarrow \neg y_1)) \wedge (y_2 \leftrightarrow \neg y_1) & \wedge & // \; T(x_1, y_1, x_2, y_2) \wedge \\
((x_0 \wedge y_0) & \vee & // \; (f(x_0, y_0) \vee \\
(x_1 \wedge y_1) & \vee & // \; f(x_1, y_1) \vee \\
(x_2 \wedge y_2)) & & // \; f(x_2, y_2))
\end{array}
$$

   ]

2. Is there a solution? If yes, find the corresponding execution; if no, show why.

Ex: Bounded Model Checking

Given the symbolic representation of a FSM M, expressed in terms of the two Boolean formulas: $I(x,y) \stackrel{\text{def}}{=} \neg x \wedge y$, $T(x,y,x',y') \stackrel{\text{def}}{=} (x' \leftrightarrow (x \leftrightarrow \neg y)) \wedge (y' \leftrightarrow \neg y)$, and the LTL property: $\varphi \stackrel{\text{def}}{=} \neg \mathbf{F}(x \wedge y)$,

1. Write a Boolean formula whose solutions (if any) represent executions of M of length 2 which violate φ.

   [Solution: The question corresponds to the Bounded Model Checking problem $M \models_2 \mathbf{E}\,\mathbf{F} f$, s.t. $f(x,y) \stackrel{\text{def}}{=} (x \wedge y)$. Thus we have:

$$
\begin{array}{lcl}
\neg x_0 \wedge y_0 & \wedge & // \ I(x_0,y_0) \wedge \\
(x_1 \leftrightarrow (x_0 \leftrightarrow \neg y_0)) \wedge (y_1 \leftrightarrow \neg y_0) & \wedge & // \ T(x_0,y_0,x_1,y_1) \wedge \\
(x_2 \leftrightarrow (x_1 \leftrightarrow \neg y_1)) \wedge (y_2 \leftrightarrow \neg y_1) & \wedge & // \ T(x_1,y_1,x_2,y_2) \wedge \\
((x_0 \wedge y_0) & \vee & // \ (f(x_0,y_0)\vee \\
(x_1 \wedge y_1) & \vee & // \ f(x_1,y_1)\vee \\
(x_2 \wedge y_2)) & & // \ f(x_2,y_2))
\end{array}
$$

   ]

2. Is there a solution? If yes, find the corresponding execution; if no, show why.

   [Solution: Yes: $\{\neg x_0, y_0, x_1, \neg y_1, x_2, y_2\}$, corresponding to the execution: $(0,1) \to (1,0) \to (1,1)$]

Ex: Bounded Model Checking

Ex: Bounded Model Checking

3. What are the diameter and the recurrence diameter of this system?

Ex: Bounded Model Checking

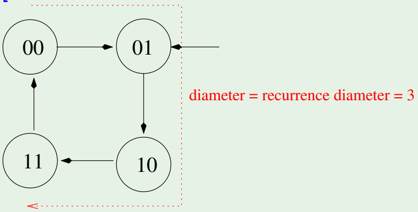
3. What are the diameter and the recurrence diameter of this system?
   [Solution:

   ]

Ex: Bounded Model Checking

3. What are the diameter and the recurrence diameter of this system?

[Solution:

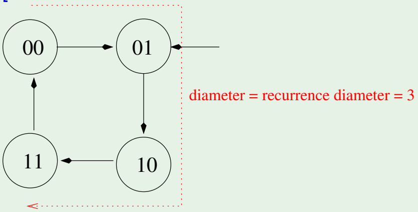


]

Ex: Bounded Model Checking

3. What are the diameter and the recurrence diameter of this system?

   [Solution:



   ]

4. From the solutions to question #1 and #2 we can conclude that:

   (a) $M \models \varphi$

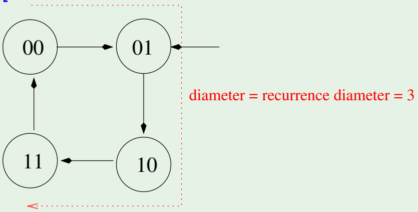
   (b) $M \not\models \varphi$

   (c) we can conclude nothing.

Ex: Bounded Model Checking

3. What are the diameter and the recurrence diameter of this system?

   [Solution:



   ]

4. From the solutions to question #1 and #2 we can conclude that:

   (a) $M \models \varphi$

   (b) $M \not\models \varphi$

   (c) we can conclude nothing.

   [Solution: b)]

Ex: Bounded Model Checking

Given the following symbolic representation of a finite state machine M, expressed in terms of the following two formulas:

- $I(x, y) \stackrel{\text{def}}{=} (\neg x \wedge \neg y)$
- $T(x, y, x', y') \stackrel{\text{def}}{=} (x' \leftrightarrow \neg y')$,

and the following LTL property:

- $\varphi \stackrel{\text{def}}{=} \neg \mathbf{F}(x \wedge y)$,

Ex: Bounded Model Checking

Given the following symbolic representation of a finite state machine M, expressed in terms of the following two formulas:

- $I(x, y) \stackrel{\text{def}}{=} (\neg x \wedge \neg y)$
- $T(x, y, x', y') \stackrel{\text{def}}{=} (x' \leftrightarrow \neg y')$,

and the following LTL property:

- $\varphi \stackrel{\text{def}}{=} \neg \mathbf{F}(x \wedge y)$,

1. write a Boolean formula whose solutions (if any) represent executions of M of length 2 which violate φ.

Ex: Bounded Model Checking

Given the following symbolic representation of a finite state machine M, expressed in terms of the following two formulas:

- $I(x, y) \stackrel{\text{def}}{=} (\neg x \wedge \neg y)$
- $T(x, y, x', y') \stackrel{\text{def}}{=} (x' \leftrightarrow \neg y')$,

and the following LTL property:

- $\varphi \stackrel{\text{def}}{=} \neg \mathbf{F}(x \wedge y)$,

1. write a Boolean formula whose solutions (if any) represent executions of M of length 2 which violate φ.
[Solution: The question corresponds to the Bounded Model Checking problem $M \models_2 \mathbf{E}\,\mathbf{F} f$, s.t. $f(x, y) \stackrel{\text{def}}{=} (x \wedge y)$.
Thus we have:

$$\begin{array}{lcl} (\neg x_0 \wedge \neg y_0) & \wedge & //\ I(x_0, y_0) \wedge \\ (x_1 \leftrightarrow \neg y_1) & \wedge & //\ T(x_0, y_0, x_1, y_1) \wedge \\ (x_2 \leftrightarrow \neg y_2) & \wedge & //\ T(x_1, y_1, x_2, y_2) \wedge \\ ((x_0 \wedge y_0) & \vee & //\ (f(x_0, y_0) \vee \\ (x_1 \wedge y_1) & \vee & //\ \ f(x_1, y_1) \vee \\ (x_2 \wedge y_2)) & & //\ \ f(x_2, y_2)) \end{array}$$

]

Ex: Bounded Model Checking

Given the following symbolic representation of a finite state machine M, expressed in terms of the following two formulas:

- $I(x, y) \stackrel{\text{def}}{=} (\neg x \wedge \neg y)$
- $T(x, y, x', y') \stackrel{\text{def}}{=} (x' \leftrightarrow \neg y')$,

and the following LTL property:

- $\varphi \stackrel{\text{def}}{=} \neg \mathbf{F}(x \wedge y)$,

1. write a Boolean formula whose solutions (if any) represent executions of M of length 2 which violate φ.
   [Solution: The question corresponds to the Bounded Model Checking problem $M \models_2 \mathbf{E}\,\mathbf{F} f$, s.t. $f(x, y) \stackrel{\text{def}}{=} (x \wedge y)$.
   Thus we have:

$$
\begin{array}{lcl}
(\neg x_0 \wedge \neg y_0) & \wedge & // \; I(x_0, y_0) \wedge \\
(x_1 \leftrightarrow \neg y_1) & \wedge & // \; T(x_0, y_0, x_1, y_1) \wedge \\
(x_2 \leftrightarrow \neg y_2) & \wedge & // \; T(x_1, y_1, x_2, y_2) \wedge \\
((x_0 \wedge y_0) & \vee & // \; (f(x_0, y_0) \vee \\
(x_1 \wedge y_1) & \vee & // \; f(x_1, y_1) \vee \\
(x_2 \wedge y_2)) & & // \; f(x_2, y_2))
\end{array}
$$

   ]

2. is there a solution? If yes, find the corresponding execution.

Ex: Bounded Model Checking

Given the following symbolic representation of a finite state machine M, expressed in terms of the following two formulas:

- $I(x, y) \stackrel{\text{def}}{=} (\neg x \wedge \neg y)$
- $T(x, y, x', y') \stackrel{\text{def}}{=} (x' \leftrightarrow \neg y')$,

and the following LTL property:

- $\varphi \stackrel{\text{def}}{=} \neg \mathbf{F}(x \wedge y)$,

1. write a Boolean formula whose solutions (if any) represent executions of M of length 2 which violate φ.
   [Solution: The question corresponds to the Bounded Model Checking problem $M \models_2 \mathbf{E}\,\mathbf{F} f$, s.t. $f(x, y) \stackrel{\text{def}}{=} (x \wedge y)$. Thus we have:

$$
\begin{array}{lcl}
(\neg x_0 \wedge \neg y_0) & \wedge & // \; I(x_0, y_0) \wedge \\
(x_1 \leftrightarrow \neg y_1) & \wedge & // \; T(x_0, y_0, x_1, y_1) \wedge \\
(x_2 \leftrightarrow \neg y_2) & \wedge & // \; T(x_1, y_1, x_2, y_2) \wedge \\
((x_0 \wedge y_0) & \vee & // \; (f(x_0, y_0) \vee \\
(x_1 \wedge y_1) & \vee & // \; f(x_1, y_1) \vee \\
(x_2 \wedge y_2)) & & // \; f(x_2, y_2))
\end{array}
$$

   ]

2. is there a solution? If yes, find the corresponding execution.
   [Solution: No: it is easy to see that the formula above is inconsistent]

Ex: Bounded Model Checking [cont.]

1. ...
2. ...

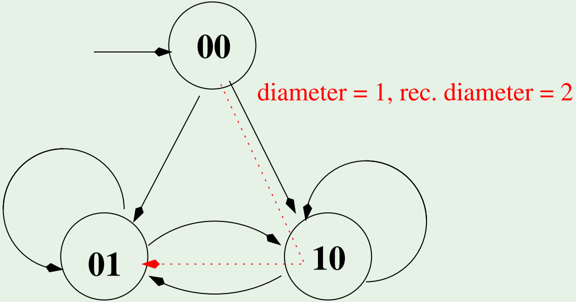
Ex: Bounded Model Checking [cont.]

1. ...
2. ...
3. what are the diameter and the recurrence diameter of this system?

Ex: Bounded Model Checking [cont.]

1. ...
2. ...
3. what are the diameter and the recurrence diameter of this system?

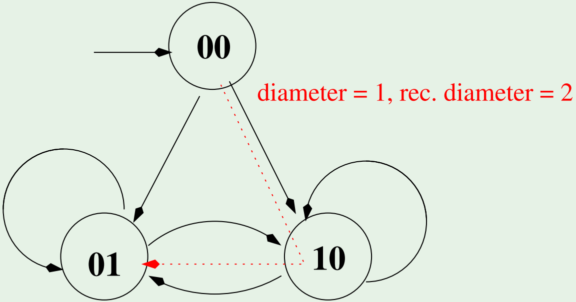
[Solution:



]

Ex: Bounded Model Checking [cont.]

1. ...
2. ...
3. what are the diameter and the recurrence diameter of this system?
   [Solution:

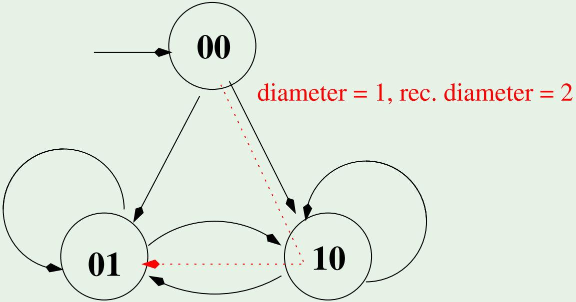


   ]

4. Can we conclude anything about the model-checking problem $M \models \varphi$? Explain why.

Ex: Bounded Model Checking [cont.]

1. ...
2. ...
3. what are the diameter and the recurrence diameter of this system?
   [Solution:



   ]

4. Can we conclude anything about the model-checking problem $M \models \varphi$? Explain why.
   [Solution: yes, we can conclude that $M \models \varphi$, since $M \not\models_2 \mathbf{E}\,\mathbf{F}\neg\varphi$ and rec. diameter=2.]

Ex: K-Induction

Given the following LTL Model Checking problem $M \models \varphi$ expressed in NuSMV input language:

```
MODULE main
VAR x : boolean; y : boolean; z : boolean;
INIT (!x & !y & z)
TRANS ((next(x) <-> (y)) & (next(y) <->  z) & (next(z) <->  x) )
LTLSPEC   G (x | y | z) ;
```

Ex: K-Induction

Given the following LTL Model Checking problem $M \models \varphi$ expressed in NuSMV input language:

```
MODULE main
VAR x : boolean; y : boolean; z : boolean;
INIT (!x & !y & z)
TRANS ((next(x) <-> (y)) & (next(y) <->  z) & (next(z) <->  x) )
LTLSPEC   G (x | y | z) ;
```

1. Write the Boolean formulas describing the k-induction encoding of the problem, with k = 1.

Ex: K-Induction

Given the following LTL Model Checking problem $M \models \varphi$ expressed in NuSMV input language:

```
MODULE main
VAR x : boolean; y : boolean; z : boolean;
INIT (!x & !y & z)
TRANS ((next(x) <-> (y)) & (next(y) <-> z) & (next(z) <-> x) )
LTLSPEC   G (x | y | z) ;
```

1. Write the Boolean formulas describing the k-induction encoding of the problem, with k = 1.
   [Solution: The LTL property is in the form "$\mathbf{G}\,Good(x, y, z)$", hence, applying k-induction:

$$
\begin{array}{llcl}
\varphi_{Base} \stackrel{\text{def}}{=} & (\neg x_0 \wedge \neg y_0 \wedge z_0) & \wedge & // \; I(x_0, y_0, z_0) \wedge \\
& \neg(x_0 \vee y_0 \vee z_0) & & // \; \neg Good(x_0, y_0, z_0) \\
\varphi_{Ind1} \stackrel{\text{def}}{=} & (x_i \vee y_i \vee z_i) & \wedge & // \; Good(x_i, y_i, z_i) \wedge \\
& ((x_{i+1} \leftrightarrow y_i) \wedge (y_{i+1} \leftrightarrow z_i) \wedge (z_{i+1} \leftrightarrow x_i)) & \wedge & // \; T(x_i, y_i, z_i, x_{i+1}, y_{i+1}, z_{i+1}) \wedge \\
& \neg(x_{i+1} \vee y_{i+1} \vee z_{i+1}) & & // \; \neg Good(x_{i+1}, y_{i+1}, z_{i+1})
\end{array}
$$

   ]

Ex: K-Induction [cont.]

1. ...

Ex: K-Induction [cont.]

1. ...
2. Say if they are satisfiable or not. If yes, show a model. If not, explain why.

Ex: K-Induction [cont.]

1. ...
2. Say if they are satisfiable or not. If yes, show a model. If not, explain why. [Solution:
   - φ_{Base} is not satisfiable. In fact, the second row forces the assignments $\neg x_0, \neg y_0, \neg z_0$, which makes the first row false.
   - φ_{Ind1} is not satisfiable. In fact, the third row forces the assignments $\neg x_{i+1}, \neg y_{i+1}, \neg z_{i+1}$, from which the second row forces the assignments $\neg x_i, \neg y_i, \neg z_i$, which makes the first row false.

   ]

Ex: K-Induction [cont.]

1. ...
2. Say if they are satisfiable or not. If yes, show a model. If not, explain why. [Solution:
   - φ_{Base} is not satisfiable. In fact, the second row forces the assignments $\neg x_0, \neg y_0, \neg z_0$, which makes the first row false.
   - φ_{Ind1} is not satisfiable. In fact, the third row forces the assignments $\neg x_{i+1}, \neg y_{i+1}, \neg z_{i+1}$, from which the second row forces the assignments $\neg x_i, \neg y_i, \neg z_i$, which makes the first row false.

   ]
3. From the previous answers we can conclude:

   (*a*) that $M \models \varphi$;

   (*b*) that $M \not\models \varphi$;

   (*c*) we can conclude nothing.

Ex: K-Induction [cont.]

1. ...
2. Say if they are satisfiable or not. If yes, show a model. If not, explain why. [Solution:
   - φ_{Base} is not satisfiable. In fact, the second row forces the assignments $\neg x_0, \neg y_0, \neg z_0$, which makes the first row false.
   - φ_{Ind1} is not satisfiable. In fact, the third row forces the assignments $\neg x_{i+1}, \neg y_{i+1}, \neg z_{i+1}$, from which the second row forces the assignments $\neg x_i, \neg y_i, \neg z_i$, which makes the first row false.

   ]
3. From the previous answers we can conclude:

   (*a*) that $M \models \varphi$;

   (*b*) that $M \not\models \varphi$;

   (*c*) we can conclude nothing.

   [Solution: a) $M \models \varphi$. In fact, we have proved it in one induction step.
   ]