Bounded Model Checking

SAT-Based Model Checking

Wallace Wu

Department of Electrical and Computer Engineering
University of Waterloo

March 3, 2011

Outline

Outline

Model Checking

- Exhaustive model checking algorithms
  - Inefficient; enumerate all states and transitions
  - Check a few million states in reasonable amount of time

Model Checking

- Exhaustive model checking algorithms
  - Inefficient; enumerate all states and transitions
  - Check a few million states in reasonable amount of time
- Symbolic model checking
  - Represent states using Boolean functions
  - Manipulating Boolean formulas: Reduced Ordered Binary Decision Trees (ROBDD or BDD for short)
  - Check $\geq 10^{20}$ states in reasonable amount of time
  - **Bottleneck:** Memory required for storing and manipulating BDDs
  - Full design verifications is generally still beyond the capacity of BDD-based model checkers

About BMC

- Proposed by Biere, Clarke et al. [Biere et al., 1999]

About BMC

- Proposed by Biere, Clarke et al. [Biere et al., 1999]
- BMC relies on exponential procedure (still limited in its capacity)

About BMC

- Proposed by Biere, Clarke et al. [Biere et al., 1999]
- BMC relies on exponential procedure (still limited in its capacity)
- Complimentary to BDD-based model checking
  - BMC can solve many cases that BDD-based techniques cannot and vice versa
  - No correlation between hardness of SAT and BDD problems
  - Does NOT replace other model checking techniques

About BMC

- Proposed by Biere, Clarke et al. [Biere et al., 1999]
- BMC relies on exponential procedure (still limited in its capacity)
- Complimentary to BDD-based model checking
  - BMC can solve many cases that BDD-based techniques cannot and vice versa
  - No correlation between hardness of SAT and BDD problems
  - Does NOT replace other model checking techniques
- **Disadvantage:** Cannot prove absence of errors in most realistic cases

Overview of BMC

Idea: Search for a counterexample in executions (paths) whose length is $\leq k$ for some integer k

Overview of BMC

Idea: Search for a counterexample in executions (paths) whose length is $\leq k$ for some integer k

Method: Efficiently reduce problem to a propositional satisfiability (SAT) problem

- Resolves state explosion problem

Overview of BMC

Idea: Search for a counterexample in executions (paths) whose length is $\leq k$ for some integer k

Method: Efficiently reduce problem to a propositional satisfiability (SAT) problem

- Resolves state explosion problem

Process: If no bug is found, increase k until either:

- A bug is found
- Problem becomes intractable
- Some predetermined upper bound for k is reached

Unique Characteristics of BMC

- User must provide a bound on the number of cycles that should be explored
  - Experiments show that BMC outperforms BDD-based techniques for k up to $\sim 60 - 80$

Unique Characteristics of BMC

- User must provide a bound on the number of cycles that should be explored
  - Experiments show that BMC outperforms BDD-based techniques for k up to $\sim 60 - 80$
- Uses SAT solving techniques to check models
  - Details of SAT solvers not covered

Outline

Verifying a 16 $\times$ 16 bit shift and add multiplier

		SMV$_2$		Prover	
bit	k	t_{exec} (s)	Memory (MB)	t_{exec} (s)	Memory (MB)
0	1	25	79	<1	1
1	2	25	79	<1	1
2	3	26	80	<1	1
3	4	27	82	1	2
4	5	33	92	1	2
5	6	67	102	1	2
6	7	258	172	2	2
7	8	1741	492	7	3
8	9		>1024	29	3
9	10			58	3
10	11			91	3
11	12			125	3
12	13			156	4
13	14			186	4
14	15			226	4
15	16			183	5

Verifying Various Designs

Model	k	Rulebase$_1$	Rulebase$_2$	Grasp	Grasp (tuned)	Chaff
1	18	7	6	282	3	2.2
2	5	70	8	1.1	0.8	<1
3	14	597	375	76	3	<1
4	24	690	261	510	12	3.7
5	12	803	184	24	2	<1
6	22	–	356	–	18	12.2
7	9	–	2671	10	2	<1
8	35	–	–	6317	20	85
9	38	–	–	9035	25	131.6
10	31	–	–	–	312	380.5
11	32	152	60	–	–	34.7
12	31	1419	1126	–	–	194.3
13	4	–	3626	–	–	9.8

All values that are right of the column k are given in seconds

Outline

Outline

Kripke Structure

- The finite automaton can be represented by a Kripke structure, a quadruple $\boldsymbol{M = (S, I, T, L)}$ where
  - $\boldsymbol{S}$ is the set of states
  - $\boldsymbol{I}$ is the set of initial states, $\boldsymbol{I \subseteq S}$
  - $\boldsymbol{T}$ is the translation relation, $\boldsymbol{T \subseteq S \times S}$
  - $\boldsymbol{L}$ is the labeling function, $\boldsymbol{L : S \rightarrow 2^A}$, where A is the set of atomic propositions, and 2^A is the powerset of A
    - $L(s), s \in S$, is made of $A_s \subseteq A$ that hold in s

Sequential Behaviour of Kripke Structures

- Use the notion of paths to define behaviour of a Kripke structure, M
- Each path, π in M is an infinite OR finite sequence of states in an order that respects T

$$\pi = (s_0, s_1, \ldots), \quad T(s_i, s_{i+1})\ \forall 0 \leq i < |\pi| - 1$$

- For $i < |\pi|$
  - $\pi(i)$ denotes the i-th state s_i in the sequence
  - $\pi_i = (s_i, s_{i+1}, \ldots)$ denotes the suffix of π starting with state s_i

Sequential Behaviour of Kripke Structures (cont'd)

- If $I(s_0)$ (i.e., s_0 is an initial state) and $s_0 \in \pi$, π is an initialized path
- If a state is not reachable $\Rightarrow$ no initialized paths that contain it

Assumptions about Kripke Structures

For a Kripke structure, M,

- $I \neq \emptyset$
- $\forall s \in S,\ \exists t \in S$ with $T(s, t)$ (*total* transition relation)

Mutual Exclusion Example

Pseudocode

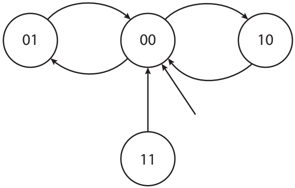
```
PROCESS A
A.pc = 0
while TRUE
        wait for B.pc == 0
        A.pc = 1
        // critical section
        A.pc = 0
```

```
PROCESS B
B.pc = 0
while TRUE
        wait for A.pc == 0
        B.pc = 1
        // critical section
        B.pc = 0
```

Mutual Exclusion Example

Modeling

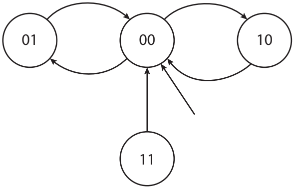
- We can encode the set of states using $A.pc$ and $B.pc$: $A.pc \cdot B.pc$.



Mutual Exclusion Example

Modeling

- We can encode the set of states using $A.pc$ and $B.pc$: $A.pc \cdot B.pc$.



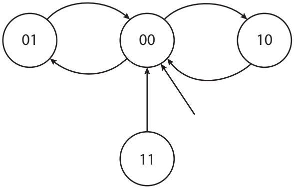
- The transition relation $T \subseteq S^2 = \{0,1\}^4$ is:

$$T = \{0100, 1000, 1100, 0001, 0010\}$$

Mutual Exclusion Example

Modeling

- We can encode the set of states using $A.pc$ and $B.pc$: $A.pc \cdot B.pc$.



- The transition relation $T \subseteq S^2 = \{0,1\}^4$ is:

$$T = \{0100, 1000, 1100, 0001, 0010\}$$

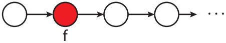
- The sequence $11, 00, 10, \ldots$ is a valid path, but it is not initialized, since $I = \{s_0\}$

Outline

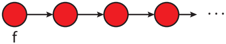
LTL Temporal Operators

Let f and g be temporal formulas. The temporal operators are:

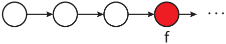
Next time: $\bigcirc f$



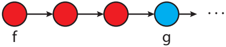
Globally: $\square f$



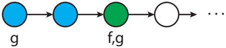
Finally: $\lozenge f$



Until: $f\,\mathbf{U}g$



Release: $f\,\mathbf{R}g$



LTL Semantics

Let π be an infinite path of a Kripke structure M and let f, g, p be temporal formulas. We recursively define LTL semantics as:

$$
\begin{array}{lcl}
\pi \models p & \Leftrightarrow & p \in L(\pi(0)) \\
\pi \models \neg p & \Leftrightarrow & \pi \not\models f \\
\pi \models f \wedge g & \Leftrightarrow & \pi \models f \text{ and } \pi \models g \\
\pi \models \bigcirc f & \Leftrightarrow & \pi_1 \models f \\
\pi \models \Box f & \Leftrightarrow & \pi_i \models f \ \forall i \geq 0 \\
\pi \models \Diamond f & \Leftrightarrow & \pi_i \models f \text{ for some } i \geq 0 \\
\pi \models f \mathbf{U} g & \Leftrightarrow & \pi_i \models g \text{ for some } i \geq 0 \text{ and } \pi_j \models f \ \forall 0 \leq j < i \\
\pi \models f \mathbf{R} g & \Leftrightarrow & \pi_i \models g \text{ if for all } j < i, \pi_j \not\models f
\end{array}
$$

LTL Semantics (cont'd)

- $M \models f \Rightarrow \pi \models f$ $\forall$ initialized paths π of M
- LTL formulas f and g are equivalent (i.e., $f \equiv g$) iff $M \models f \leftrightarrow M \models g$ $\forall M$
- **Duality:** $\neg\Diamond\neg p \equiv \Box p$

Outline

LTL Model Checking

- Standard technique:
  - compute product of Kripke structure M with automaton representing the negation of the property to be checked

$$\mathcal{A}_{\neg\phi}$$

  - emptiness of the product automaton $\Rightarrow$ correctness of the property

$$L(M||\mathcal{A}_{\neg\phi}) = \emptyset \Rightarrow M \models \phi$$

Outline

BMC

- Motivation was to leverage success in SAT solving in model checking

BMC

- Motivation was to leverage success in SAT solving in model checking
- Search for counterexample within a predetermined bound
  - i.e., Consider only prefixes of paths bounded by k in the search
  - In practice, progressively increase k, looking for longer witnesses in longer traces

BMC

- Motivation was to leverage success in SAT solving in model checking
- Search for counterexample within a predetermined bound
  - i.e., Consider only prefixes of paths bounded by k in the search
  - In practice, progressively increase k, looking for longer witnesses in longer traces
- Since LTL formulas are defined over all paths, a counterexample is a trace/path that contradicts the property
  - such a trace is called a witness for the property
  - **Example:** a counterexample to $M \models \Box p$ is the existence of a witness such that $\Diamond \neg p$

Outline

Prefixes

- Bounded semantics approximate unbounded semantics

Prefixes

- Bounded semantics approximate unbounded semantics
- In BMC, finite prefixes of paths are considered

Prefixes

- Bounded semantics approximate unbounded semantics
- In BMC, finite prefixes of paths are considered
- Only the first $k + 1$ states $(s_0, \ldots, s_k)$ of a path are used

Prefixes

- Bounded semantics approximate unbounded semantics
- In BMC, finite prefixes of paths are considered
- Only the first $k+1$ states $(s_0, \ldots, s_k)$ of a path are used
- A finite length prefix represents an infinite path if there is a back loop from the last state of the prefix to any previous state(s)

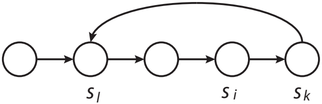


Figure: Prefix with back loop

Prefixes

- Bounded semantics approximate unbounded semantics
- In BMC, finite prefixes of paths are considered
- Only the first $k+1$ states $(s_0, \ldots, s_k)$ of a path are used
- A finite length prefix represents an infinite path if there is a back loop from the last state of the prefix to any previous state(s)
- A prefix without back loop(s) only represents the finite behaviour of the path up to state s_k

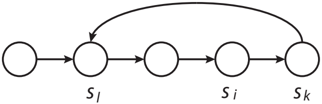


Figure: Prefix with back loop

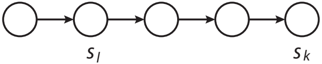


Figure: Prefix without back loop

Example of Prefix Behaviour

- Consider the LTL property: $\Box p$

Example of Prefix Behaviour

- Consider the LTL property: $\Box p$
- With back loop:
  - Property can be satisfied in the prefix

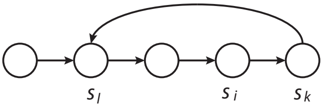
Example of Prefix Behaviour

- Consider the LTL property: $\Box p$
- With back loop:
  - Property can be satisfied in the prefix
- Without back loop:
  - Property CANNOT be satisfied in the prefix
  - If p holds for all states $s_0, \ldots, s_k$, we still cannot conclude that the property holds since p may not hold at s_{k+1}

Prefixes With Back Loops - (k, l)-loop

Definition 1

For $1 \leq k$, we call a path π a (k, l)-loop if $T(\pi(k), \pi(l))$ and $\pi = u \cdot v^{\omega}$ with $u = (\pi(0), \ldots, \pi(l-1))$ and $v = (\pi(l), \ldots, \pi(k))$. We call π a k-loop if there exists $k \geq l \geq 0$ for which π is a (k, l)-loop.

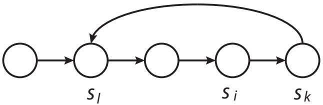


Prefixes With Back Loops - (k, l)-loop

Definition 1

For $1 \leq k$, we call a path π a (k, l)-loop if $T(\pi(k), \pi(l))$ and $\pi = u \cdot v^\omega$ with $u = (\pi(0), \ldots, \pi(l-1))$ and $v = (\pi(l), \ldots, \pi(k))$. We call π a k-loop if there exists $k \geq l \geq 0$ for which π is a (k, l)-loop.

- If a path is a k-loop, then the original LTL semantics are maintained ($\because$ infinite path represented in prefix)



Prefixes With Back Loops - (k, l)-loop

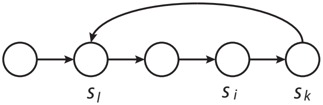
Definition 1

For $1 \leq k$, we call a path π a (k, l)-loop if $T(\pi(k), \pi(l))$ and $\pi = u \cdot v^{\omega}$ with $u = (\pi(0), \ldots, \pi(l-1))$ and $v = (\pi(l), \ldots, \pi(k))$. We call π a k-loop if there exists $k \geq l \geq 0$ for which π is a (k, l)-loop.

- If a path is a k-loop, then the original LTL semantics are maintained ($\because$ infinite path represented in prefix)

Definition 2 (Bounded Semantics for a Loop)

Let $k \geq 0$ and π be a k-loop. Then an LTL formula f is valid along the path π with bound k (denoted by $\pi \models_k f$) iff $\pi \models f$.



Prefixes Without Back Loops

- $\Diamond p$ is valid along π in unbounded semantics if $\exists i \geq 0$ s.t. p is valid along the suffix π_i of π

Prefixes Without Back Loops

- $\Diamond p$ is valid along π in unbounded semantics if $\exists i \geq 0$ s.t. p is valid along the suffix π_i of π
- In bounded semantics, $(k+1)$-th state $\pi(k)$ does not have a successor
  - Cannot define bounded semantics recursively over suffixes of π

Prefixes Without Back Loops

- $\Diamond p$ is valid along π in unbounded semantics if $\exists i \geq 0$ s.t. p is valid along the suffix π_i of π
- In bounded semantics, $(k+1)$-th state $\pi(k)$ does not have a successor
  - Cannot define bounded semantics recursively over suffixes of π
- Introduce notation:
  $$\pi \models_k^i f$$
  where i is the current position in the prefix of π
  - Implies suffix π_i of π satisfies f, i.e.,
  $$\pi \models_k^i \Rightarrow \pi_i \models f$$

Semantics of Prefixes Without Back Loops

Definition 3 (Bounded Semantics without a Loop)

Let $k \geq 0$, and π be a path that is not a k-loop. An LTL formula f is valid along π with bound k (denoted by $\pi \models_k f$) iff $\pi \models_k^0 f$ where

$$
\begin{array}{lcl}
\pi \models_k^i p & \Leftrightarrow & p \in L(\pi(i)) \\
\pi \models_k^i \neg p & \Leftrightarrow & p \notin L(\pi(i)) \\
\pi \models_k^i f \wedge g & \Leftrightarrow & \pi \models_k^i f \text{ and } \pi \models_k^i g \\
\pi \models_k^i f \vee g & \Leftrightarrow & \pi \models_k^i f \text{ or } \pi \models_k^i g \\
\pi \models_k^i \Box f & & \text{is always false} \\
\pi \models_k^i \Diamond f & \Leftrightarrow & \exists j, i \leq j \leq k \bullet \pi \models_k^j f
\end{array}
$$

Semantics of Prefixes Without Back Loops

$$
\begin{array}{lcl}
\pi \models_k^i \bigcirc f & \Leftrightarrow & i < k \text{ and } \pi \models_k^{i+1} f \\
\pi \models_k^i f\mathbf{U}g & \Leftrightarrow & \exists j, i \le j \le k \bullet \pi \models_k^j g \\
 & & \quad \text{and } \forall n, i \le n < j \bullet \pi \models_k^n f \\
\pi \models_k^i f\mathbf{R}g & \Leftrightarrow & \exists j, i \le j \le k \bullet \pi \models_k^j f \\
 & & \quad \text{and } \forall n, i \le n < j \bullet \pi \models_k^n g
\end{array}
$$

Semantics of Prefixes Without Back Loops

$$
\begin{array}{lcl}
\pi \models_k^i \bigcirc f & \Leftrightarrow & i < k \text{ and } \pi \models_k^{i+1} f \\
\pi \models_k^i f\mathbf{U}g & \Leftrightarrow & \exists j, i \leq j \leq k \bullet \pi \models_k^j g \\
 & & \quad \text{and } \forall n, i \leq n < j \bullet \pi \models_k^n f \\
\pi \models_k^i f\mathbf{R}g & \Leftrightarrow & \exists j, i \leq j \leq k \bullet \pi \models_k^j f \\
 & & \quad \text{and } \forall n, i \leq n < j \bullet \pi \models_k^n g
\end{array}
$$

- $\Box f$ is not valid along π in k-bounded semantics since f may not hold for π_{k+1}

Semantics of Prefixes Without Back Loops

$$
\begin{array}{lcl}
\pi \models^i_k \bigcirc f & \Leftrightarrow & i < k \text{ and } \pi \models^{i+1}_k f \\
\pi \models^i_k f\mathbf{U}g & \Leftrightarrow & \exists j, i \leq j \leq k \bullet \pi \models^j_k g \\
 & & \quad \text{and } \forall n, i \leq n < j \bullet \pi \models^n_k f \\
\pi \models^i_k f\mathbf{R}g & \Leftrightarrow & \exists j, i \leq j \leq k \bullet \pi \models^j_k f \\
 & & \quad \text{and } \forall n, i \leq n < j \bullet \pi \models^n_k g
\end{array}
$$

- $\Box f$ is not valid along π in k-bounded semantics since f may not hold for π_{k+1}
- $\neg\Diamond f \not\equiv \Box\neg f$, i.e., duality between $\Box$ and $\Diamond$ no longer holds

Outline

Model Checking Problem Reduction

- Introduce path quantifiers **E** and **A**
  - **E** denotes that an LTL formula is expected to be correct over some path
  - **A** denotes that an LTL formula is expected to be correct over all paths

Model Checking Problem Reduction

- Introduce path quantifiers **E** and **A**
  - **E** denotes that an LTL formula is expected to be correct over some path
  - **A** denotes that an LTL formula is expected to be correct over all paths
- The existential model checking problem $M \models \mathbf{E}f$ can reduced to a bounded existential model checking problem $M \models_k \mathbf{E}f$

Model Checking Problem Reduction

- Introduce path quantifiers **E** and **A**
  - **E** denotes that an LTL formula is expected to be correct over some path
  - **A** denotes that an LTL formula is expected to be correct over all paths
- The existential model checking problem $M \models \mathbf{E}f$ can reduced to a bounded existential model checking problem $M \models_k \mathbf{E}f$
- $M \models \mathbf{E}f$ means $\exists$ an initialized path in M that satisfies f

Model Checking Problem Reduction (cont'd)

- Basis for this reduction lies in the following lemmas

Model Checking Problem Reduction (cont'd)

- Basis for this reduction lies in the following lemmas

Lemma 1

Let f be an LTL formula and π a path, then $\pi \models_k f \Rightarrow \pi \models f$

Model Checking Problem Reduction (cont'd)

- Basis for this reduction lies in the following lemmas

Lemma 1

Let f be an LTL formula and π a path, then $\pi \models_k f \Rightarrow \pi \models f$

Lemma 2

Let f be an LTL formula and M a Kripke structure. If $M \models \mathbf{E}f$, $\exists k \geq 0$ with $M \models \mathbf{E}f$.

Model Checking Problem Reduction (cont'd)

The following theorem is derived from the lemmas:

Theorem 1

Let f be an LTL formula and M a Kripke structure. Then $M \models \mathbf{E}f$ iff $\exists k \geq 0$ such that $M \models_k \mathbf{E}f$.

Model Checking Problem Reduction (cont'd)

The following theorem is derived from the lemmas:

Theorem 1

Let f be an LTL formula and M a Kripke structure. Then $M \models \mathbf{E}f$ iff $\exists k \geq 0$ such that $M \models_k \mathbf{E}f$.

- Informally, it means that for a sufficiently high bound, bounded and unbounded semantics are equivalent.

Outline

BMC Reduction

- Given a Kripke structure M, an LTL formula f, and a bound k, we can construct a propositional formula

$$[[M, f]]_k$$

BMC Reduction

- Given a Kripke structure M, an LTL formula f, and a bound k, we can construct a propositional formula

$$[[M, f]]_k$$

- Let $s_0, \ldots, s_k$ be a finite sequence of states on path π

BMC Reduction

- Given a Kripke structure M, an LTL formula f, and a bound k, we can construct a propositional formula

$$[[M, f]]_k$$

- Let $s_0, \ldots, s_k$ be a finite sequence of states on path π
- Each state s_i represents a state at time step i and consists of an assignment of truth values to the set of state variables

BMC Reduction

- Given a Kripke structure M, an LTL formula f, and a bound k, we can construct a propositional formula

$$[[M, f]]_k$$

- Let $s_0, \ldots, s_k$ be a finite sequence of states on path π
- Each state s_i represents a state at time step i and consists of an assignment of truth values to the set of state variables
- Encode constraints on $s_0, \ldots, s_k$ so that

$$[[M, f]]_k \text{ is satisfiable } \Leftrightarrow \pi \text{ is a witness for } f$$

Definition of $[[M, f]]_k$

- Three components define $[[M, f]]_k$

Definition of $[[M, f]]_k$

- Three components define $[[M, f]]_k$
  - Propositional formula $[[M]]_k$: constrains $s_0, \ldots, s_k$ to be a valid path starting from an initial state

Definition of $[[M, f]]_k$

- Three components define $[[M, f]]_k$
  - Propositional formula $[[M]]_k$: constrains $s_0, \ldots, s_k$ to be a valid path starting from an initial state
  - Loop condition: a propositional formula that is evaluated to true only if the path π contains a loop

Definition of $[[M, f]]_k$

- Three components define $[[M, f]]_k$
  - Propositional formula $[[M]]_k$: constrains $s_0, \ldots, s_k$ to be a valid path starting from an initial state
  - Loop condition: a propositional formula that is evaluated to true only if the path π contains a loop
  - Propositional formula that constrains π to satisfy f

Propositional Formula $[[M]]_k$

Definition 4 (Unfolding of the Transition Relation)

For a Kripke structure $M, k \geq 0$

$$[[M]]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T\left(s_i, s_{i+1}\right)$$

Loop Condition

- Define propositional formula ${}_lL_k$ to be true iff there is a transition from state s_k to state s_l.
- By definition, ${}_lL_k = T(s_k, s_l)$

Loop Condition

- Define propositional formula ${}_lL_k$ to be true iff there is a transition from state s_k to state s_l.
- By definition, ${}_lL_k = T(s_k, s_l)$

Definition 5 (Loop Condition)

The loop condition L_k is true iff there exists a back loop from state s_k to a previous state or to itself. We define L_k to be:

$$L_k := \bigvee_{l=0}^{k} {}_lL_k$$

Loop Successor

Definition 6 (Successor in a Loop)

Let k, l and i be non-negative integers s.t. $l, i \leq k$. Define the *successor* $\mathrm{succ}(i)$ of i in a (k, l)-loop as:

$$\mathrm{succ}(i) := \begin{cases} i + 1 & i < k \\ l & i = k \end{cases}$$

Definition 7 (Translation of an LTL Formula for a Loop)

Let f be an LTL formula, $k, l, i \geq 0$, with $l, i \leq k$.

$$
\begin{array}{lcl}
{}_l[[p]]_k^i & := & p(s_i) \\
{}_l[[\neg p]]_k^i & := & \neg p(s_i) \\
{}_l[[f \vee g]]_k^i & := & {}_l[[f]]_k^i \vee {}_l[[g]]_k^i \\
{}_l[[f \wedge g]]_k^i & := & {}_l[[f]]_k^i \wedge {}_l[[g]]_k^i \\
{}_l[[\Box f]]_k^i & := & {}_l[[f]]_k^i \wedge {}_l[[\Box f]]_k^{\mathrm{succ}(i)} \\
{}_l[[\Diamond f]]_k^i & := & {}_l[[f]]_k^i \vee {}_l[[\Diamond f]]_k^{\mathrm{succ}(i)} \\
{}_l[[f\mathbf{U}g]]_k^i & := & {}_l[[g]]_k^i \vee \left({}_l[[f]]_k^i \wedge {}_l[[f\mathbf{U}g]]_k^{\mathrm{succ}(i)} \right) \\
{}_l[[f\mathbf{R}g]]_k^i & := & {}_l[[g]]_k^i \wedge \left({}_l[[f]]_k^i \vee {}_l[[f\mathbf{R}g]]_k^{\mathrm{succ}(i)} \right) \\
{}_l[[\bigcirc f]]_k^i & := & {}_l[[f]]_k^{\mathrm{succ}(i)}
\end{array}
$$

- ${}_l[[\cdot]]_k^i$ is an intermediate formula
  - l and k defines the start and end of the (k, l)-loop
  - i for the current position in the path

Definition 8 (Translation of an LTL Formula without a Loop)

Inductive Case $\forall i \leq k$

$$
\begin{array}{lcl}
[[p]]_k^i & := & p(s_i) \\
[[\neg p]]_k^i & := & \neg p(s_i) \\
[[f \vee g]]_k^i & := & [[f]]_k^i \vee [[g]]_k^i \\
[[f \wedge g]]_k^i & := & [[f]]_k^i \wedge [[g]]_k^i \\
[[\Box f]]_k^i & := & [[f]]_k^i \wedge [[\Box f]]_k^{i+1} \\
[[\Diamond f]]_k^i & := & [[f]]_k^i \vee [[\Diamond f]]_k^{i+1} \\
[[f\mathbf{U}g]]_k^i & := & [[g]]_k^i \vee \left([[f]]_k^i \wedge [[f\mathbf{U}g]]_k^{i+1} \right) \\
[[f\mathbf{R}g]]_k^i & := & [[g]]_k^i \wedge \left([[f]]_k^i \vee [[f\mathbf{R}g]]_k^{i+1} \right) \\
[[\bigcirc f]]_k^i & := & [[f]]_k^{i+1}
\end{array}
$$

Base Case

$$[[f]]_k^{k+1} := 0$$

General Translation

Definition 9 (General Translation)

Let f be an LTL formula, M a Kripke strucutre and $k \geq 0$.

$$[[M,f]]_k := [[M]]_k \wedge \left(\left(\neg L_k \wedge [[f]]_k^0 \right) \vee \bigvee_{l=0}^{k} \left({}_l L_k \wedge {}_l[[f]]_k^l \right) \right)$$

Satisfiability and Bounded Model Checking

Theorem 2

$[[M, f]]_k$ *is satisfiable iff* $M \models_k \mathbf{E}f$

Outline

Mutual Exclusion Example

Pseudocode

Recall the earlier pseudocode for two processes that wish to gain access to a shared resource:

PROCESS A

```
A.pc = 0
while TRUE
        wait for B.pc == 0
        A.pc = 1
        // critical section
        A.pc = 0
```

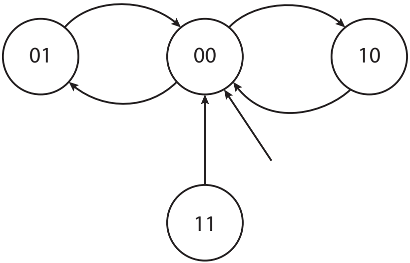
PROCESS B

```
B.pc = 0
while TRUE
        wait for A.pc == 0
        B.pc = 1
        // critical section
        B.pc = 0
```

Mutual Exclusion Example

Modeling

- Each state s of the system M is represented by two-bit variables
  - $s[1]$: high bit (Process A)
  - $s[0]$: low bit (Process B)

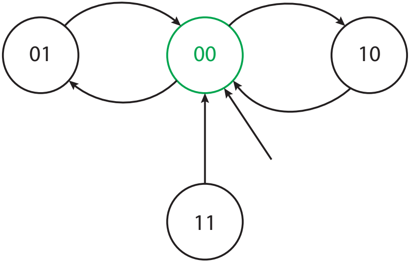


Mutual Exclusion Example

Modeling

- Each state s of the system M is represented by two-bit variables
  - $s[1]$: high bit (Process A)
  - $s[0]$: low bit (Process B)
- Initial state:

$$I(s) := \neg s[1] \wedge \neg s[0]$$

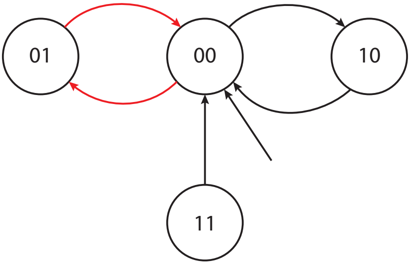


Mutual Exclusion Example

Modeling (cont'd)

- Transition relation:

$$T(s, s') := \left(\neg s[1] \wedge \left(s[0] \leftrightarrow \neg s'[0]\right)\right) \vee$$

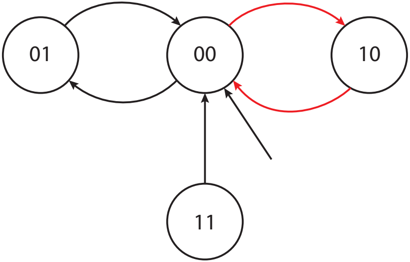


Mutual Exclusion Example

Modeling (cont'd)

- Transition relation:

$$
\begin{aligned}
T(s, s') := \ & (\neg s[1] \wedge (s[0] \leftrightarrow \neg s'[0])) \vee \\
& (\neg s[0] \wedge (s[1] \leftrightarrow \neg s'[1])) \vee
\end{aligned}
$$

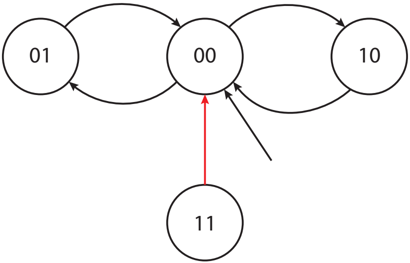


Mutual Exclusion Example

Modeling (cont'd)

- Transition relation:

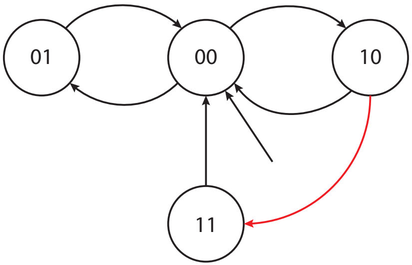
$$
\begin{aligned}
T(s, s') := \; & \big(\neg s[1] \land (s[0] \leftrightarrow \neg s'[0])\big) \lor \\
& \big(\neg s[0] \land (s[1] \leftrightarrow \neg s'[1])\big) \lor \\
& {\color{red}\big(s[0] \land s[1] \land \neg s'[1] \land \neg s'[0]\big)}
\end{aligned}
$$



Mutual Exclusion Example

Adding Faulty Transition

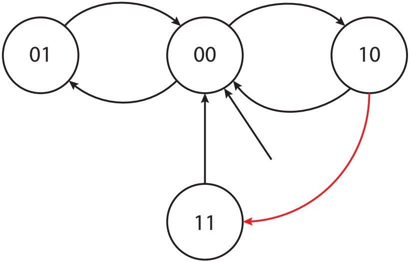
- Suppose we add the fault transition 10 $\rightarrow$ 11 to the model, M:



Mutual Exclusion Example

Adding Faulty Transition

- Suppose we add the fault transition $10 \rightarrow 11$ to the model, M:



- Then we have the new transition relation:

$$T_f(s, s') := T(s, s') \vee (s[1] \wedge \neg s[0] \wedge s'[1] \wedge s'[0])$$

Mutual Exclusion Example

Checking Mutex Property

- Consider the safety property that at most one process can be in the critical section at any time, i.e.:

$$f = \Box \neg p = \Box \neg (s[1] \wedge s[0])$$

Mutual Exclusion Example

Checking Mutex Property

- Consider the safety property that at most one process can be in the critical section at any time, i.e.:

$$f = \Box\neg p = \Box\neg(s[1] \wedge s[0])$$

- Try to find a counterexample of this property, that is, a witness for $\Diamond p$
  - If a witness exists, then $M \not\models f$
  - If a witness cannot be found, then $M \models_k f$ (i.e., property holds up to the bound k)

Mutual Exclusion Example

LTL to Propositional Formula Translation

- Consider the bound $k = 2$.

Mutual Exclusion Example

LTL to Propositional Formula Translation

- Consider the bound $k = 2$.
- Unroll the transition relation:

$$
\begin{aligned}
[[M]]_2 := {} & I(s_0) \wedge \bigwedge_{l=0}^{k-1} T(s_l, s_{l+1}) \\
= {} & I(s_0) \wedge T_f(s_0, s_1) \wedge T_f(s_1, s_2)
\end{aligned}
$$

Mutual Exclusion Example

LTL to Propositional Formula Translation

- Consider the bound $k = 2$.
- Unroll the transition relation:

$$
\begin{aligned}
[[M]]_2 := {} & I(s_0) \wedge \bigwedge_{l=0}^{k-1} T(s_l, s_{l+1}) \\
= {} & I(s_0) \wedge T_f(s_0, s_1) \wedge T_f(s_1, s_2)
\end{aligned}
$$

- The loop condition is:

$$
L_2 := \bigvee_{l=0}^{2} T_f(s_2, s_l)
$$

Mutual Exclusion Example

LTL to Propositional Formula Translation (cont'd)

- Translation for paths without loops is:

$$
\begin{aligned}
[[\Diamond p]]_2^0 &:= p(s_0) \vee [[\Diamond p]]_2^1 \\
[[\Diamond p]]_2^1 &:= p(s_1) \vee [[\Diamond p]]_2^2 \\
[[\Diamond p]]_2^2 &:= p(s_2) \vee [[\Diamond p]]_2^3 \\
[[\Diamond p]]_2^3 &:= 0
\end{aligned}
$$

Mutual Exclusion Example

LTL to Propositional Formula Translation (cont'd)

- Translation for paths without loops is:

$$
\begin{aligned}
[[\Diamond p]]_2^0 &:= p(s_0) \vee [[\Diamond p]]_2^1 \\
[[\Diamond p]]_2^1 &:= p(s_1) \vee [[\Diamond p]]_2^2 \\
[[\Diamond p]]_2^2 &:= p(s_2) \vee [[\Diamond p]]_2^3 \\
[[\Diamond p]]_2^3 &:= 0
\end{aligned}
$$

- Substitute all intermediate terms to obtain:

$$[[\Diamond p]]_2^0 := p(s_0) \vee p(s_1) \vee p(s_2)$$

Mutual Exclusion Example

LTL to Propositional Formula Translation (cont'd)

- Translation for paths with loops is:

$$
\begin{array}{rcl}
{}_0[[\Diamond p]]_2^0 & := & p(s_0) \vee {}_0[[\Diamond p]]_2^1 \\
{}_0[[\Diamond p]]_2^1 & := & p(s_1) \vee {}_0[[\Diamond p]]_2^2 \\
{}_0[[\Diamond p]]_2^2 & := & p(s_2) \vee {}_0[[\Diamond p]]_2^0 \\
{}_1[[\Diamond p]]_2^1 & := & p(s_1) \vee {}_1[[\Diamond p]]_2^2 \\
{}_1[[\Diamond p]]_2^2 & := & p(s_2) \vee {}_1[[\Diamond p]]_2^1 \\
{}_2[[\Diamond p]]_2^2 & := & p(s_2) \vee {}_2[[\Diamond p]]_2^2
\end{array}
$$

Mutual Exclusion Example

Check for a Witness

- Putting everything together:

$$[[F, \Diamond p]]_2 := [[M]]_2 \wedge \left(\left(\neg L_2 \wedge [[\Diamond p]]_2^0 \right) \vee \bigvee_{l=0}^{2} \left({}_l L_2 \wedge {}_l[[\Diamond p]]_2^l \right) \right)$$

Mutual Exclusion Example

Check for a Witness

- Putting everything together:

$$[[F, \Diamond p]]_2 := [[M]]_2 \wedge \left(\left(\neg L_2 \wedge [[\Diamond p]]_2^0 \right) \vee \bigvee_{l=0}^{2} \left({}_l L_2 \wedge {}_l[[\Diamond p]]_2^l \right) \right)$$

- Since a finite path to a bad state is sufficient for falsifying a property, omit the loop condition.

Mutual Exclusion Example

Check for a Witness

- Putting everything together:

$$[[F, \Diamond p]]_2 := [[M]]_2 \wedge \left(\left(\neg L_2 \wedge [[\Diamond p]]_2^0 \right) \vee \bigvee_{l=0}^{2} \left({}_l L_2 \wedge {}_l[[\Diamond p]]_2^l \right) \right)$$

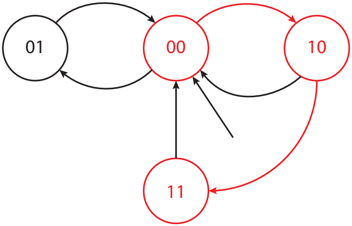
- Since a finite path to a bad state is sufficient for falsifying a property, omit the loop condition.
- This results in the formula:

$$\begin{aligned} [[M, \Diamond p]]_2 : &= [[M]]_2 \wedge [[\Diamond p]]_2^0 \\ &= I(s_0) \wedge T_f(s_0, s_1) \wedge T_f(s_1, s_2) \\ &\quad \wedge (p(s_0) \vee p(s_1) \vee p(s_2)) \end{aligned}$$

Mutual Exclusion Example

Check for a Witness (cont'd)

- $(s_0, s_1, s_2) = (00, 10, 11)$ satisfies $[[M, \Diamond p]]_2$
  - an initialized path that violates the safety property



Outline

Completeness

- Suppose we have a model checking problem $M \models \mathbf{E}f$, where f is the negated version of the property to be checked

Completeness

- Suppose we have a model checking problem $M \models \mathbf{E}f$, where f is the negated version of the property to be checked
- Increment bound k until a finite-length witness is found
  - In this case, we are done and $M \models \mathbf{E}f$ (i.e., model does not satisfy the property)

Completeness

- Suppose we have a model checking problem $M \models \mathbf{E}f$, where f is the negated version of the property to be checked
- Increment bound k until a finite-length witness is found
  - In this case, we are done and $M \models \mathbf{E}f$ (i.e., model does not satisfy the property)
- If $M \not\models \mathbf{E}f$, how do we know when to terminate the BMC model checker?

Outline

Reachability Diameter

- For every finite state system M, a property p, and a translation scheme, there is a number $\mathcal{CT}$ where the absence of errors up to cycle $\mathcal{CT}$ proves that $M \models p$

Reachability Diameter

- For every finite state system M, a property p, and a translation scheme, there is a number $\mathcal{CT}$ where the absence of errors up to cycle $\mathcal{CT}$ proves that $M \models p$
- Completeness threshold is the minimal bound on k for $\Box p$ required to reach all states and called the reachability diameter

Reachability Diameter

- For every finite state system M, a property p, and a translation scheme, there is a number $\mathcal{CT}$ where the absence of errors up to cycle $\mathcal{CT}$ proves that $M \models p$
- Completeness threshold is the minimal bound on k for $\Box p$ required to reach all states and called the reachability diameter

Definition 10

The reachability diameter $rd(M)$ is the minimal number of steps required for reaching all reachable states, i.e.:

$$rd(M) := \min \Bigg\{ i \,|\, \forall s_0, \ldots, s_n \bullet \exists s'_0, \ldots, s'_t, t \leq i \bullet$$

$$I(s_0) \wedge \bigwedge_{j=0}^{n-1} T(s_j, s_{j+1}) \rightarrow \left(I\left(s'_0\right) \wedge \bigwedge_{j=0}^{t-1} T\left(s'_j, s'_{j+1}\right) \wedge s'_t = s_n \right) \Bigg\}$$

Reachability Diameter

- For every finite state system M, a property p, and a translation scheme, there is a number $\mathcal{CT}$ where the absence of errors up to cycle $\mathcal{CT}$ proves that $M \models p$
- Completeness threshold is the minimal bound on k for $\Box p$ required to reach all states and called the reachability diameter

Definition 10

The reachability diameter $rd(M)$ is the minimal number of steps required for reaching all reachable states, i.e.:

$$rd(M) := \min \left\{ i \,|\, \forall s_0, \ldots, s_n \bullet \exists s'_0, \ldots, s'_t, t \leq i \bullet \right.$$

$$\left. I(s_0) \wedge \bigwedge_{j=0}^{n-1} T(s_j, s_{j+1}) \rightarrow \left(I\left(s'_0\right) \wedge \bigwedge_{j=0}^{t-1} T\left(s'_j, s'_{j+1}\right) \wedge s'_t = s_n \right) \right\}$$

Determining $rd(M)$

- Worst case for $n = 2^{|V|}$, where V is the set of variables defining the states of M

Determining $rd(M)$

- Worst case for $n = 2^{|V|}$, where V is the set of variables defining the states of M
- Determine best n
  - Let $n = i + 1$. Check whether every state that can be reached in $i + 1$ can be reached sooner

Determining $rd(M)$

- Worst case for $n = 2^{|V|}$, where V is the set of variables defining the states of M
- Determine best n
  - Let $n = i + 1$. Check whether every state that can be reached in $i + 1$ can be reached sooner

$$rd(M) := \min \left\{ i \,|\, \forall s_0, \ldots, s_{i+1} \bullet \exists s'_0, \ldots, s'_i, \bullet \right.$$

$$\left. I(s_0) \wedge \bigwedge_{j=0}^{i} T(s_j, s_{j+1}) \rightarrow \left(I\left(s'_0\right) \wedge \bigwedge_{j=0}^{i-1} T\left(s'_j, s'_{j+1}\right) \wedge \bigvee_{j=0}^{i} s'_j = s_{i+1} \right) \right\}$$

Determining $rd(M)$

- Worst case for $n = 2^{|V|}$, where V is the set of variables defining the states of M
- Determine best n
  - Let $n = i + 1$. Check whether every state that can be reached in $i + 1$ can be reached sooner

$$rd(M) := \min \Bigg\{ i \,|\, \forall s_0, \ldots, s_{i+1} \bullet \exists s'_0, \ldots, s'_i, \bullet$$

$$I(s_0) \wedge \bigwedge_{j=0}^{i} T(s_j, s_{j+1}) \rightarrow \left(I\left(s'_0\right) \wedge \bigwedge_{j=0}^{i-1} T\left(s'_j, s'_{j+1}\right) \wedge \bigvee_{j=0}^{i} s'_j = s_{i+1} \right) \Bigg\}$$

- Alternation of quantifiers in the two previous expressions are hard to solve in practice

Recurrence Diameter for Reachability

- Approximate the reachability diameter instead

Recurrence Diameter for Reachability

- Approximate the reachability diameter instead

Definition 11 (Recurrence Diameter for Reachability)

The *recurrence diameter for reachability* with respect to a model M, denoted by $rdr(M)$, is the longest loop-free path in M starting from an initial state:

$$rdr(M) := \max \left\{ i \mid \exists s_0 \ldots s_i \bullet \right.$$

$$\left. I(s_0) \wedge \bigwedge_{j=0}^{i-1} T(s_j, s_{j+1}) \wedge \bigwedge_{j=0}^{i-1} \bigwedge_{k=j+1}^{i} s_j \neq s_k \right\}$$

Recurrence Diameter for Reachability

- Approximate the reachability diameter instead

Definition 11 (Recurrence Diameter for Reachability)

The *recurrence diameter for reachability* with respect to a model M, denoted by $rdr(M)$, is the longest loop-free path in M starting from an initial state:

$$rdr(M) := \max \left\{ i \,|\, \exists s_0 \ldots s_i \bullet \\ I(s_0) \wedge \bigwedge_{j=0}^{i-1} T(s_j, s_{j+1}) \wedge \bigwedge_{j=0}^{i-1} \bigwedge_{k=j+1}^{i} s_j \neq s_k \right\}$$

- $rdr(M)$ is an over-approximation of $rd(M)$ because every shortest path is a loop-free path

Outline

Liveness

- If a proof for liveness exists, the proof can be established by examining all finite sequences of length k starting from initial states

Liveness

- If a proof for liveness exists, the proof can be established by examining all finite sequences of length k starting from initial states

Definition 12 (Translation for Liveness Properties)

$$[[M, \mathbf{A}\Diamond p]]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \rightarrow \bigvee_{i=0}^{k} p(s_i)$$

Liveness (cont'd)

Theorem 3

$$M \models \mathbf{A}\Diamond p \Leftrightarrow \exists k \bullet [[M, \mathbf{A}\Diamond p]]_k \textit{ is valid.}$$

Liveness (cont'd)

Theorem 3

$$M \models \mathbf{A}\Diamond p \Leftrightarrow \exists k \bullet [[M, \mathbf{A}\Diamond p]]_k \textit{ is valid.}$$

- Need to search for a k that makes the negation of $[[M, \mathbf{A}\Diamond p]]_k$ unsatisfiable

Liveness (cont'd)

Theorem 3

$$M \models \mathbf{A}\Diamond p \Leftrightarrow \exists k \bullet [[M, \mathbf{A}\Diamond p]]_k \textit{ is valid.}$$

- Need to search for a k that makes the negation of $[[M, \mathbf{A}\Diamond p]]_k$ unsatisfiable
- Bound k needed for a proof represent length of longest sequence from an initial state without hitting a state where p holds

Liveness (cont'd)

Theorem 3

$$M \models \mathbf{A}\Diamond p \Leftrightarrow \exists k \bullet [[M, \mathbf{A}\Diamond p]]_k \text{ is valid.}$$

- Need to search for a k that makes the negation of $[[M, \mathbf{A}\Diamond p]]_k$ unsatisfiable
- Bound k needed for a proof represent length of longest sequence from an initial state without hitting a state where p holds
- In BMC, we have semi-decision procedures for

$$M \models \mathbf{E}\Box\neg p \Leftrightarrow M \not\models \mathbf{A}\Diamond p$$

  - $\because$ either $\mathbf{A}\Diamond p$ or $\mathbf{E}\Box\neg p$ must hold, one of the semi-decision procedures must terminate

Outline

Induction

- Inductive techniques can be used to make BMC complete for safety properties

Induction

- Inductive techniques can be used to make BMC complete for safety properties
- Proving that $M \models \mathbf{A}\Box p$ by induction usually involves:
  - Manually finding a strengthening inductive invariant - expression that is inductive and implies the property

Induction

- Inductive techniques can be used to make BMC complete for safety properties
- Proving that $M \models \mathbf{A}\Box p$ by induction usually involves:
  - Manually finding a strengthening inductive invariant - expression that is inductive and implies the property
- Inductive proof:
  - Base case
  - Induction step
  - Strengthening step

Prove Inductive Invariant Holds for First n Steps

- Show that inductive invariant ϕ holds in first n steps by checking whether the following is unsatisfiable:

$$\exists s_0, \ldots, s_n \bullet I(s_0) \wedge \bigwedge_{i=0}^{n-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg\phi(s_i)$$

Prove Inductive Invariant Holds for First n Steps

- Show that inductive invariant ϕ holds in first n steps by checking whether the following is unsatisfiable:

$$\exists s_0, \ldots, s_n \bullet I(s_0) \wedge \bigwedge_{i=0}^{n-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg\phi(s_i)$$

- Base step is equivalent to searching for a counterexample to $\Box p$

Inductive Step

- Prove induction step by showing that the following is unsatisfiable:

$$\exists s_0, \ldots, s_{n+1} \bullet \bigwedge_{i=0}^{n} (\phi(s_i) \land T(s_i, s_{i+1})) \land \neg\phi(s_{n+1})$$

Refining the Inductive Step

- Paths in M restricted to contain distinct states
  - Preserves completeness of BMC for safety properties
  - A bad state is reachable (if it exists) is reachable via a simple path

Refining the Inductive Step

- Paths in M restricted to contain distinct states
  - Preserves completeness of BMC for safety properties
  - A bad state is reachable (if it exists) is reachable via a simple path
- Sufficient to show that the following is unsatisfiable:

$$\exists s_0, \ldots, s_{n+1} \bullet \bigwedge_{j=0}^{n} \bigwedge_{k=j+1}^{n+1} (s_j \neq s_k) \wedge \bigwedge_{i=0}^{n} (\phi(s_i) \wedge T(s_i, s_{i+1})) \wedge \neg\phi(s_{n+1})$$

Strengthening Inductive Invariant Implies Property

- Establish that for an arbitrary i:

$$\forall s_i \bullet \phi(s_i) \rightarrow p(s_i)$$

References

A. Biere, A. Cimatti, E. Clarke, and Y. Zhu
Symbolic Model Checking without BDDs
In *Proc. of the Workshop on Tools and Algorithms for the Consturction and Analysis of Systems (TACAS'99)*, LNCS. Springer-Verlag, 1999.

A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu
Bounded Model Checking
Advances in Computers, Vol. 58, 2003.