SMT-based Model Checking

Cesare Tinelli

The University of Iowa

STRESS School 2014
Modeling Computational Systems

Software or hardware systems can be often represented as a state transition system $\mathcal{M} = (S, I, T, L)$ where

- $S$ is a set of states, the state space
- $I \subseteq S$ is a set of initial states
- $T \subseteq S \times S$ is a (right-total) transition relation
- $L : S \rightarrow 2^P$ is a labeling function where $P$ is a set of state predicates

Typically, the state predicates denote variable-value pairs $x = v$
Model Checking

Software or hardware systems can be often represented as a state transition system \( M = (S, I, T, L) \)

\( M \) can be seen as a model both

1. in an engineering sense:
   an abstraction of the real system

   and

2. in a mathematical logic sense:
   a Kripke structure in some modal logic
Model Checking

The functional properties of a computational system can be expressed as *temporal* properties

- for a suitable model $\mathcal{M} = (S, I, T, L)$ of the system
- in a suitable temporal logic
Model Checking

The functional properties of a computational system can be expressed as *temporal* properties

- for a suitable model $\mathcal{M} = (S, I, T, L)$ of the system
- in a suitable temporal logic

Two main classes of properties:

- *Safety properties*: nothing bad ever happens
- *Liveness properties*: something good eventually happens
The functional properties of a computational system can be expressed as *temporal* properties

- for a suitable model $\mathcal{M} = (S, I, T, L)$ of the system
- in a suitable temporal logic

Two main classes of properties:

- *Safety properties*: nothing bad ever happens
- *Liveness properties*: something good eventually happens

I will focus on checking safety in this talk
Talk Roadmap

• Checking safety properties

• Logic-based model checking

• Satisfiability Modulo Theories
  • theories
  • solvers

• SMT-based model checking
  • main approaches
  • k-induction
    • basic method
    • enhancements
  • interpolation
Basic Terminology

Let $\mathcal{M} = (S, I, T, L)$ be a transition system

The set $R$ of \textit{reachable states (of $\mathcal{M}$)} is the smallest subset of $S$ such that

1. $I \subseteq R$ (initial states are reachable)
2. $(R \triangleright T) \subseteq R$ ($T$-successors of reachable states are reachable)
Basic Terminology

Let $\mathcal{M} = (S, I, T, L)$ be a transition system

The set $R$ of *reachable states (of $\mathcal{M}$)* is the smallest subset of $S$ such that

1. $I \subseteq R$ (initial states are reachable)
2. $(R \Join T) \subseteq R$ ($T$-successors of reachable states are reachable)

Let $E \subseteq S$ (an *error property*)

The set $B_E$ of *bad states wrt $E$* is the smallest subset of $S$ such that

1. $E \subseteq B_E$ (error states are bad)
2. $(T \Join B_E) \subseteq B_E$ ($T$-predecessors of bad states are bad)
$M$ is *safe* wrt an error property $\mathcal{E}$ if $\mathcal{R} \cap \mathcal{E} = \emptyset$

iff $\mathcal{I} \cap \mathcal{B}_\mathcal{E} = \emptyset$

- **Safe**
  - $\mathcal{R}$
  - $\mathcal{E}$
  - $\mathcal{I}$
  - $\mathcal{B}_\mathcal{E}$

- **Unsafe**
  - $\mathcal{R}$
  - $\mathcal{E}$
  - $\mathcal{I}$
  - $\mathcal{B}_\mathcal{E}$
Invariance

A state property $\mathcal{P} \subseteq S$ is \textit{invariant (for $\mathcal{M}$)} iff $\mathcal{R} \subseteq \mathcal{P}$

\[
\begin{array}{c}
\text{invariant} \\
\begin{tikzpicture}
\draw (0,0) ellipse (1.5 and 1); 
\draw (0,0) ellipse (2 and 1.5);
\draw (0,0) rectangle (2,2);
\node at (1,1) {$\mathcal{P}$};
\node at (0.5,0) {$\mathcal{R}$};
\node at (0,0) {$S$};
\end{tikzpicture}
\end{array}
\hspace{1cm}
\begin{array}{c}
\text{not invariant} \\
\begin{tikzpicture}
\draw (0,0) ellipse (1.5 and 1); 
\draw (0,0) ellipse (2 and 1.5);
\draw (0,0) rectangle (2,2);
\node at (1,1) {$\mathcal{P}$};
\node at (0.5,0) {$\mathcal{R}$};
\node at (0,0) {$S$};
\end{tikzpicture}
\end{array}
\]

\textbf{Note:} $\mathcal{P}$ is invariant for $\mathcal{M}$ iff $\mathcal{M}$ is safe wrt $S \setminus \mathcal{P}$
Checking Safety

In principle, to check that $\mathcal{M}$ is safe wrt $\mathcal{E}$ it suffices to

1. compute $\mathcal{R}$ and
2. check that $\mathcal{R} \cap \mathcal{E} = \emptyset$ (Forward reachability)
Checking Safety

In principle, to check that $M$ is safe wrt $E$ it suffices to

1. compute $R$ and  
2. check that $R \cap E = \emptyset$  

or

1. compute $B_E$ and  
2. check that $I \cap B_E = \emptyset$  

(Forward reachability)  

(Backward reachability)
Checking Safety

In principle, to check that $\mathcal{M}$ is safe wrt $\mathcal{E}$ it suffices to

1. compute $R$ and
2. check that $R \cap \mathcal{E} = \emptyset$  
   
   (Forward reachability)

or

1. compute $B_\mathcal{E}$ and
2. check that $I \cap B_\mathcal{E} = \emptyset$  

   (Backward reachability)

This can be done explicitly only if $S$ is finite, and relatively small ($< 10M$ states)
Checking Safety

In principle, to check that $M$ is safe wrt $E$ it suffices to

1. compute $R$ and
2. check that $R \cap E = \emptyset$  

(Forward reachability)

or

1. compute $B_E$ and
2. check that $I \cap B_E = \emptyset$  

(Backward reachability)

Alternatively, we can represent $M$ symbolically and use

- BDD-based methods, if $S$ is finite,
- automata-based methods,
- logic-based methods, or
- abstract interpretation methods
Checking Safety

In principle, to check that $M$ is safe wrt $E$ it suffices to

1. compute $R$ and
2. check that $R \cap E = \emptyset$  \hspace{1cm} \text{(Forward reachability)}

or

1. compute $B_E$ and
2. check that $I \cap B_E = \emptyset$  \hspace{1cm} \text{(Backward reachability)}

Alternatively, we can represent $M$ symbolically and use

- BDD-based methods, if $S$ is finite,
- automata-based methods,
- logic-based methods, or
- abstract interpretation methods
Logic-based Symbolic Model Checking

Applicable if we can encode $\mathcal{M} = (S, I, T, L)$ in some (classical) logic $\mathcal{L}$ with decidable entailment $\models_{\mathcal{L}}$

$(\varphi \models_{\mathcal{L}} \psi \text{ iff } \varphi \land \neg \psi \text{ is unsatisfiable in } \mathcal{L})$
Logic-based Symbolic Model Checking

Applicable if we can encode $\mathcal{M} = (S, I, T, L)$ in some (classical) logic $\mathbb{L}$ with decidable entailment $\models_{\mathbb{L}}$

$(\varphi \models_{\mathbb{L}} \psi \iff \varphi \land \neg \psi$ is unsatisfiable in $\mathbb{L})$

Examples of $\mathbb{L}$:

- Propositional logic
- Quantified Boolean Formulas
- Bernay-Schrönfinkel logic
- Quantifier-free real (or linear integer) arithmetic with arrays and uninterpreted functions
- \ldots
Logical encodings of transitions systems

\[ \mathcal{M} = (S, I, T, L) \]  
\[ X : \text{set of variables} \quad V : \text{set of values in } L \]

Not.: if \( x = (x_1, \ldots, x_n) \) and \( s = (v_1, \ldots, v_n) \), \( \phi[s] := \phi[v_1/x_1, \ldots, v_n/x_n] \)
Logical encodings of transitions systems

\[ \mathcal{M} = (S, I, T, L) \quad X: \text{set of variables} \quad V: \text{set of values in } \mathbb{L} \]

Not.: if \( x = (x_1, \ldots, x_n) \) and \( s = (v_1, \ldots, v_n) \), \( \phi[s] := \phi[v_1/x_1, \ldots, v_n/x_n] \)

- states \( s \in S \) encoded as \( n \)-tuples of \( V^n \)
Logical encodings of transitions systems

\[ M = (S, I, \mathcal{T}, \mathcal{L}) \]

\( X \): set of variables  
\( V \): set of values in \( \mathbb{L} \)

Not.: if \( \mathbf{x} = (x_1, \ldots, x_n) \) and \( \mathbf{s} = (v_1, \ldots, v_n) \), \( \phi[\mathbf{s}] := \phi[v_1/x_1, \ldots, v_n/x_n] \)

- states \( s \in S \) encoded as \( n \)-tuples of \( V^n \)
- \( I \) encoded as a formula \( I[\mathbf{x}] \) with free variables \( \mathbf{x} \) such that

\[ s \in \mathcal{I} \text{ iff } \models_{\mathbb{L}} I[s] \]
Logical encodings of transitions systems

$\mathcal{M} = (S, I, T, L)$  $X$: set of variables  $V$: set of values in $\mathbb{L}$

Not.: if $x = (x_1, \ldots, x_n)$ and $s = (v_1, \ldots, v_n)$, $\phi[s] := \phi[v_1/x_1, \ldots, v_n/x_n]$

- states $s \in S$ encoded as $n$-tuples of $V^n$
- $I$ encoded as a formula $I[x]$ with free variables $x$ such that
  
  $s \in I$ iff $\models_l I[s]$

- $T$ encoded as a formula $T[x, x']$ such that
  
  $\models_l T[s, s']$ for all $(s, s') \in T$
Logical encodings of transitions systems

\[ \mathcal{M} = (S, I, T, \mathcal{L}) \]

\( X \): set of variables  
\( V \): set of values in \( \mathbb{L} \)

Not.: \( \text{if } x = (x_1, \ldots, x_n) \text{ and } s = (v_1, \ldots, v_n), \phi[s] := \phi[v_1/x_1, \ldots, v_n/x_n] \)

- states \( s \in S \) encoded as \( n \)-tuples of \( V^n \)
- \( I \) encoded as a formula \( I[x] \) with free variables \( x \) such that
  \[ s \in I \iff \models \mathcal{L} I[s] \]
- \( T \) encoded as a formula \( T[x, x'] \) such that
  \[ \models \mathcal{L} T[s, s'] \text{ for all } (s, s') \in T \]
- State properties encoded as formulas \( P[x] \)
Strongest Inductive Invariant

The strongest inductive invariant (for $M$ in $\mathbb{L}$) is a formula $R[x]$ such that $\models_L R[s]$ iff $s \in R$.
Strongest Inductive Invariant

The *strongest inductive invariant (for $M$ in $\mathbb{L}$)* is a formula $R[x]$ such that $\models_{\mathbb{L}} R[s]$ iff $s \in \mathcal{R}$

Suppose we can compute $R$ from $I$ and $T$. Then
The **strongest inductive invariant (for \( M \) in \( \mathbb{L} \))** is a formula \( R[x] \) such that \( \models_{\mathbb{L}} R[s] \) iff \( s \in R \).

Suppose we can compute \( R \) from \( I \) and \( T \). Then

checking that \( M \) is safe wrt a property \( P[x] \) reduces to checking that \( R[x] \models_{\mathbb{L}} \neg P[x] \).
The strongest inductive invariant (for $M$ in $\mathbb{L}$) is a formula $R[x]$ such that $\models \mathbb{L} R[s]$ iff $s \in \mathcal{R}$.

Suppose we can compute $R$ from $I$ and $T$. Then checking that a property $P[x]$ is invariant for $M$ reduces to checking that $R[x] \models \mathbb{L} P[x]$.
Strongest Inductive Invariant

The *strongest inductive invariant (for \( M \) in \( L \))* is a formula \( R[x] \) such that \( \models_L R[s] \iff s \in R \)

Suppose we can compute \( R \) from \( I \) and \( T \). Then

checking that a property \( P[x] \) is invariant for \( M \) reduces to

checking that \( R[x] \models_L P[x] \)

**Problem:** \( R \) may be very expensive or impossible to compute, or not even representable in \( L \)

Strongest Inductive Invariant

The strongest inductive invariant (for \( M \) in \( \mathbb{L} \)) is a formula \( R[x] \) such that \( \models_{\mathbb{L}} R[s] \) iff \( s \in R \)

Suppose we can compute \( R \) from \( I \) and \( T \). Then checking that a property \( P[x] \) is invariant for \( M \) reduces to checking that \( R[x] \models_{\mathbb{L}} P[x] \)

**Problem:** \( R \) may be very expensive or impossible to compute, or not even representable in \( \mathbb{L} \)

Logic-based model checking is about approximating \( R \) as efficiently as possible and as precisely as needed
Main Logic-based Approaches

- Bounded model checking [CBRZ01, AMP06, BHvMW09]
- Interpolation-based model checking [McM03, McM05a]
- Property Directed Reachability [BM07, Bra10, EMB11]
- Temporal induction [SSS00, dMRS03, HT08]
- Backward reachability [ACJT96, GR10]
- . . .

Past accomplishments: mostly based on propositional logic, with SAT solvers as reasoning engines

New frontier: based on logics decided by solvers for Satisfiability Modulo Theories [Seb07, BSST09]
Model Checking Modulo Theories

We invariably reason about transition systems in the context of some theory $\mathcal{T}$ of their data types.

**Examples**

- Pipelined microprocessors: theory of equality, atoms like $f(g(a, b), c) = g(c, a)$
- Timed automata: theory of integers/reals, atoms like $x - y < 2$
- General software: combination of theories, atoms like $a[2 \times j + 1] + x \geq \text{car}(l) - f(x)$

Such reasoning can be reduced to checking the satisfiability of certain formulas in (or *modulo*) the theory $\mathcal{T}$. 
Satisfiability Modulo Theories

Let $\mathcal{T}$ be a first-order theory of signature $\Sigma$

The $\mathcal{T}$-satisfiability problem for a class $C$ of $\Sigma$-formulas: determine for $\varphi[x] \in C$ if $\{\exists x \varphi\}$ holds in a model of $\mathcal{T}$
Satisfiability Modulo Theories

Fact: the $\mathcal{T}$-satisfiability of quantifier-free formulas is decidable for many theories $\mathcal{T}$ of interest in model checking.
Satisfiability Modulo Theories

Fact: the $\mathcal{T}$-satisfiability of quantifier-free formulas is decidable for many theories $\mathcal{T}$ of interest in model checking

- Equality with “Uninterpreted Function Symbols”
- Linear Arithmetic (Real and Integer)
- Arrays (i.e., updatable maps)
- Finite sets and multisets
- Strings
- Inductive data types (enumerations, lists, trees, . . . )
- . . .
Satisfiability Modulo Theories

Fact: the $\mathcal{T}$-satisfiability of quantifier-free formulas is decidable for many theories $\mathcal{T}$ of interest in model checking.

Thanks to advances in SAT and in decision procedures, this can be done very efficiently in practice by current SMT solvers.
Model Checking: SAT or SMT?

SMT encodings in model checking provide several advantages over SAT encodings:

- more powerful language
  - (unquantified) first-order formulas instead of Boolean formulas
- satisfiability still efficiently decidable
- similar high level of automation
- more natural and compact encodings
- greater scalability
- not limited to finite-state systems
Model Checking: SAT or SMT?

SMT encodings in model checking provide several advantages over SAT encodings

SMT-based model checking techniques are blurring the line between traditional model checking and deductive verification
Talk Roadmap

✓ Checking safety properties

✓ Logic-based model checking

✓ Satisfiability Modulo Theories
  ✓ theories
  ✓ solvers

• SMT-based model checking
  • main approaches
  • k-induction
    • basic method
    • enhancements
  • interpolation
SMT-based Model Checking

A few approaches:

• Predicate abstraction + finite model checking
• Bounded model checking
• Backward reachability
• Temporal induction (aka $\kappa$-induction)
• Interpolation-based model checking
SMT-based Model Checking

A few approaches:

- Predicate abstraction + finite model checking
- Bounded model checking
- Backward reachability
- Temporal induction (aka $\kappa$-induction)
- Interpolation-based model checking

Will focus more on temporal induction
Technical Preliminaries

Let’s fix

- $\mathcal{L}$, a logic decided by an SMT solver
  (e.g., quantifier-free linear arithmetic and EUF)
- $\mathcal{M} = (I[x], T[x, x'])$, an encoding in $\mathcal{L}$ of a system $\mathcal{M}$
- $P[x]$, a state property to be proven invariant for $\mathcal{M}$
Example: Parametric Resettable Counter

Model

Vars

\[
\begin{align*}
\text{input pos int } & n_0 \\
\text{input bool } & r \\
\text{int } & c, n
\end{align*}
\]

Initialization

\[
\begin{align*}
c & := 1 \\
n & := n_0
\end{align*}
\]

Transitions

\[
\begin{align*}
n' & := n \\
c' & := \text{if (} r' \text{ or } c = n \text{)} \\
& \quad \text{then } 1 \\
& \quad \text{else } c + 1
\end{align*}
\]

The transition relation contains infinitely many instances of the schema above, one for each \( n_0 > 0 \)
Example: Parametric Resettable Counter

Model

Vars
input pos int n_0
input bool r
int c, n

Initialization
\[ c := 1 \]
\[ n := n_0 \]

Transitions
\[ n' := n \]
\[ c' := \text{if } (r' \text{ or } c = n) \text{ then } 1 \]
\[ \text{else } c + 1 \]

Encoding in \( \mathbb{L} = \text{LIA} \)

\[ x := (c, n, r, n_0) \]
\[ I[x] := (c = 1) \land (n = n_0) \]
\[ T[x, x'] := (n' = n) \]
\[ \land (r' \lor (c = n) \rightarrow (c' = 1)) \]
\[ \land (\neg r' \land (c \neq n) \rightarrow (c' = c + 1)) \]

Property

\[ P[x] := c \leq n \]
Inductive Reasoning

Let $M = (I[x], T[x, x'])$
Inductive Reasoning

Let $M = (I[x], T[x, x'])$

To prove $P[x]$ invariant for $M$ it suffices to show that it is *inductive* for $M$, i.e.,

1. $I[x] \models P[x]$ (base case)

   and

2. $P[x] \land T[x, x'] \models P[x']$ (inductive step)
Inductive Reasoning

Let \( M = (I[x], T[x, x']) \)

To prove \( P[x] \) invariant for \( M \) it suffices to show that it is *inductive* for \( M \), i.e.,

1. \( I[x] \models_{\mathbb{L}} P[x] \) (base case)

and

2. \( P[x] \land T[x, x'] \models_{\mathbb{L}} P[x'] \) (inductive step)

An SMT solver can check both entailments above

\( \varphi \models_{\mathbb{L}} \psi \) iff \( \varphi \land \neg \psi \) is unsatisfiable in \( \mathbb{L} \)
Inductive Reasoning

Let $M = (I[x], T[x, x'])$

To prove $P[x]$ invariant for $M$ it suffices to show that it is inductive for $M$, i.e.,

1. $I[x] \models P[x]$ (base case)
   and
2. $P[x] \land T[x, x'] \models P[x']$ (inductive step)

Problem: Not all invariants are inductive

Example: In the parametric resettable counter, $P := c \leq n + 1$ is invariant but (2) above is falsifiable, e.g., by $(c, n, r) = (4, 3, false)$ and $(c, n, r)' = (5, 3, false)$
Improving Induction’s Precision

1. $I[x] \models L P[x]

2. $P[x] \land T[x, x'] \models L P[x']$

A few options:

SMT-based Model Checking – p.24/56
Improving Induction’s Precision

1. $I[x] \models_L P[x]$  

2. $P[x] \land T[x, x'] \models_L P[x']$

A few options:

- **Strengthen $P$:** find a property $Q$ such that $Q[x] \models_L P[x]$
  and prove $Q$ inductive
Improving Induction’s Precision

1. \( I[x] \models_L P[x] \)
2. \( P[x] \land T[x, x'] \models_L P[x'] \)

A few options:

- **Strengthen** \( P \): find a property \( Q \) such that \( Q[x] \models_L P[x] \) and prove \( Q \) inductive
  
  Difficult to automate (but lots of progress at prop. level)
Improving Induction’s Precision

1. \[ I[x] \models_L P[x] \]
2. \[ P[x] \land T[x, x'] \models_L P[x'] \]

A few options:

- **Strengthen \( P \):** find a property \( Q \) such that \[ Q[x] \models_L P[x] \]
  and prove \( Q \) inductive
  
  Difficult to automate (but lots of progress at prop. level)

- **Strengthen \( T \):** find another invariant \( Q[x] \) and use
  \[ Q[x] \land T[x, x'] \land Q[x'] \] instead of \( T[x, x'] \)
Improving Induction’s Precision

1. \[ I[x] \models_L P[x] \]
2. \[ P[x] \land T[x, x'] \models_L P[x'] \]

A few options:

- **Strengthen** \( P \): find a property \( Q \) such that \( Q[x] \models_L P[x] \) and prove \( Q \) inductive
  
  Difficult to automate (but lots of progress at prop. level)

- **Strengthen** \( T \): find another invariant \( Q[x] \) and use \( Q[x] \land T[x, x'] \land Q[x'] \) instead of \( T[x, x'] \)
  
  Difficult to automate (but lots of recent progress)
Improving Induction’s Precision

1. $I[x] \models P[x]$

2. $P[x] \land T[x, x'] \models P[x']$

A few options:

- **Strengthen $P$:** find a property $Q$ such that $Q[x] \models P[x]$ and prove $Q$ inductive
  
  Difficult to automate (but lots of progress at prop. level)

- **Strengthen $T$:** find another invariant $Q[x]$ and use $Q[x] \land T[x, x'] \land Q[x']$ instead of $T[x, x']$
  
  Difficult to automate (but lots of recent progress)

- **Consider longer $T$-paths:** $k$-induction
Improving Induction’s Precision

1. \( I[x] \models_{L} P[x] \)

2. \( P[x] \land T[x, x'] \models_{L} P[x'] \)

A few options:

- **Strengthen \( P \):** find a property \( Q \) such that \( Q[x] \models_{L} P[x] \) and prove \( Q \) inductive
  
  Difficult to automate (but lots of progress at prop. level)

- **Strengthen \( T \):** find another invariant \( Q[x] \) and use \( Q[x] \land T[x, x'] \land Q[x'] \) instead of \( T[x, x'] \)
  
  Difficult to automate (but lots of recent progress)

- **Consider longer \( T \)-paths:** \( k \)-induction
  
  Easy to automate (but fairly weak in its basic form)
Basic $k$-Induction (Naive Algorithm)

Notation: $I_i := I[x_i], \ P_i := P[x_i], \ T_i := T[x_{i-1}, x_i]$

for $i = 0$ to $\infty$ do
  if not $(I_0 \land T_1 \land \cdots \land T_i \models_{\mathbb{L}} P_i)$ then
    return fail
  if $(P_0 \land \cdots \land P_i \land T_1 \land \cdots \land T_{i+1} \models_{\mathbb{L}} P_{i+1})$ then
    return success

$P$ is $k$-inductive for some $k \geq 0$, if the first entailment holds for all $i = 0, \ldots, k$ and the second entailment holds for $i = k$

Example: In the parametric resettable counter, $P := c \leq n + 1$ is 1-inductive, but not 0-inductive
Basic $k$-Induction (Naive Algorithm)

Notation: $I_i := I[x_i]$, $P_i := P[x_i]$, $T_i := T[x_{i-1}, x_i]$

for $i = 0$ to $\infty$ do
  if not $(I_0 \land T_1 \land \cdots \land T_i |\equiv L P_i)$ then
    return fail
  if $(P_0 \land \cdots \land P_i \land T_1 \land \cdots \land T_{i+1} |\equiv L P_{i+1})$ then
    return success

$P$ is $k$-inductive for some $k \geq 0$, if the first entailment holds for all $i = 0, \ldots, k$ and the second entailment holds for $i = k$

Note:

- inductive = 0-inductive
- $k$-inductive $\Rightarrow (k + 1)$-inductive $\Rightarrow$ invariant
- some invariants are not $k$-inductive for any $k$
Enhancements to $k$-Induction

- Abstraction and refinement
- Path compression
- Termination checks
- Property strengthening
- Invariant generation
- Multiple property checking
Path Compression (simplified)

Let $F[x, y]$ be a formula s.t. $F[x, y] \models L \forall z (T[y, z] \Rightarrow T[x, z])$
(Ex: $F[x, y] := x = y$)
Path Compression (simplified)

Let $F[x, y]$ be a formula s.t. $F[x, y] \models \forall z (T[y, z] \Rightarrow T[x, z])$

(Ex: $F[x, y] := x = y$)

Can strengthen the premise of the inductive step as follows

2. $P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \land C_k \models P_{k+1}$

where $C_k := \bigwedge_{0 \leq i < j \leq k} \neg F[x_i, x_j]$
Path Compression (simplified)

Let $F[x, y]$ be a formula s.t. $F[x, y] \models \forall z (T[y, z] \Rightarrow T[x, z])$

(Ex: $F[x, y] := x = y$)

Can strengthen the premise of the inductive step as follows

2. $P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \land C_k \models P_{k+1}$

where $C_k := \bigwedge_{0 \leq i < j \leq k} \neg F[x_i, x_j]$

**Rationale:** Consider a path that breaks original (2)

$\pi := s_0, \ldots, s_i, s_{i+1}, \ldots, s_j, s_{j+1}, \ldots, s_{k+1}$

with $F[s_i, s_j]$ and $i < j$. If $\pi$ is on an actual execution of $M$, so is the shorter path $s_0, \ldots, s_i, s_{j+1}, \ldots, s_{k+1}$
Path Compression (simplified)

Let \( F[x, y] \) be a formula s.t. \( F[x, y] \models \forall z (T[y, z] \Rightarrow T[x, z]) \)
(Ex: \( F[x, y] := x = y \))

Can further strengthen the premise of the inductive step with

2. \( P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \land C_k \land N_k \models P_{k+1} \)

where \( N_k := \land_{1 \leq i \leq k+1} \neg I[x_i] \)
Path Compression (simplified)

Let $F[x,y]$ be a formula s.t. $F[x,y] \models_{\text{L}} \forall z \ (T[y,z] \Rightarrow T[x,z])$

(Ex: $F[x,y] := x = y$)

Can further strengthen the premise of the inductive step with

2. $P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \land C_k \land N_k \models_{\text{L}} P_{k+1}$

where $N_k := \bigwedge_{1 \leq i \leq k+1} \neg I[x_i]$

Rationale: if

$s_0, \ldots, s_i, \ldots, s_{k+1}$ breaks original (2) and $I[s_i]$, then

$s_i, \ldots, s_{k+1}$ breaks the base case in the first place
Path Compression (simplified)

Let $F[x, y]$ be a formula s.t. $F[x, y] \models \forall z (T[y, z] \Rightarrow T[x, z])$

(Ex: $F[x, y] := x = y$)

Can further strengthen the premise of the inductive step with

2. $P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \land C_k \land N_k \models P_{k+1}$

where $N_k := \bigwedge_{1 \leq i \leq k+1} \neg I[x_i]$

Better $F$’s than $x = y$ can be generated by an analysis of $M$

More sophisticated notions of compressions, based on forward and backward simulation, have been proposed [dMRS03]
Termination check

\[ C_k := \bigwedge_{0 \leq i < j \leq k} \neg F[x_i, x_j] \]

for \( k = 0 \) to \( \infty \) do
    if not \((I_0 \land T_1 \land \cdots \land T_k) \models L P_k \) then
        return fail
    if \((P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1}) \models L P_{k+1} \) then
        return success
    if \((I_0 \land T_1 \land \cdots \land T_{k+1}) \models L \neg C_{k+1} \) then
        return success

SMT-based Model Checking – p.30/56
Termination check

\[ C_k := \bigwedge_{0 \leq i < j \leq k} \neg F[x_i, x_j] \]

for \( k = 0 \) to \( \infty \) do
  if not \((I_0 \land T_1 \land \cdots \land T_k \models P_k)\) then
    return fail
  if \((P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \models P_{k+1})\) then
    return success
  if \((I_0 \land T_1 \land \cdots \land T_{k+1} \models \neg C_{k+1})\) then
    return success

**Rationale:** If the last test succeeds, every execution of length \( k + 1 \) is compressible to a shorter one. Hence, the whole reachable state space has been covered without finding counterexamples for \( P \)
Termination check

\[ C_k := \bigwedge_{0 \leq i < j \leq k} \neg F[x_i, x_j] \]

for \( k = 0 \) to \( \infty \) do
  if not \( (I_0 \land T_1 \land \cdots \land T_k \models_L P_k) \) then
    return fail
  if \( (P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \models_L P_{k+1}) \) then
    return success
  if \( (I_0 \land T_1 \land \cdots \land T_{k+1} \models_L \neg C_{k+1}) \) then
    return success

Note: The termination check may slow down the process but increases precision in some cases.
It even makes \( k \)-induction terminating, and so complete, whenever \( F \) is an equivalence and the quotient \( S/F \) is finite (e.g., timed automata)
(Undirected) Invariant Generation

1. Generate invariants for $\mathcal{M}$ independently from $P$, either before or in parallel with $k$-induction

2. For each invariant $J[x]$, add $J_0 \land \cdots \land J_{k+1}$ to induction hypothesis in induction step

$$P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \models P_{k+1}$$
(Undirected) Invariant Generation

1. Generate invariants for $\mathcal{M}$ independently from $P$, either before or in parallel with $k$-induction

2. For each invariant $J[x]$, add $J_0 \land \cdots \land J_{k+1}$ to induction hypothesis in induction step

$$P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \models \mathbb{L} P_{k+1}$$

**Correctness:** states not satisfying $J$ are definitely unreachable and so can be pruned
(Undirected) Invariant Generation

1. Generate invariants for $\mathcal{M}$ independently from $P$, either before or in parallel with $k$-induction

2. For each invariant $J[x]$, add $J_0 \land \cdots \land J_{k+1}$ to induction hypothesis in induction step

$$P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \models L \quad P_{k+1}$$

**Correctness:** states not satisfying $J$ are definitely unreachable and so can be pruned

**Viability:** can use any property-independent method for invariant generation (template-based [KGT11], abstract interpretation-based, . . . )
(Undirected) Invariant Generation

1. Generate invariants for $M$ independently from $P$, either before or in parallel with $k$-induction

2. For each invariant $J[x]$, add $J_0 \land \cdots \land J_{k+1}$ to induction hypothesis in induction step

$$P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \models P_{k+1}$$

Effectiveness: when $P$ is invariant, can substantially improve

- speed, by making $P$ $k$-inductive for a smaller $k$, and
- precision, by turning $P$ from $k$-inductive for no $k$ to $k$-inductive for some $k$
(Undirected) Invariant Generation

1. Generate invariants for $M$ independently from $P$, either before or in parallel with $k$-induction

2. For each invariant $J[x]$, add $J_0 \land \cdots \land J_{k+1}$ to induction hypothesis in induction step

\[
P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \models \text{L} \ P_{k+1}
\]

Shortcomings:

- Computed invariants may not prune the right unreachable states
- Adding too many invariants may swamp the SMT solver
Property Strengthening

Suppose in the \( k \)-induction loop the SMT solver finds a counterexample \( s_0, \ldots, s_{k+1} \) for

\[
2. \quad P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \models_L P_{k+1}
\]
Property Strengthening

Suppose in the $k$-induction loop the SMT solver finds a counterexample $s_0, \ldots, s_{k+1}$ for

$$2. \quad P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \models L P_{k+1}$$

Then this property is satisfied by $s_0$:

$$F[x_0] := \exists x_1, \ldots, x_{k+1} (P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \land \neg P_{k+1})$$
Property Strengthening

Suppose in the $k$-induction loop the SMT solver finds a counterexample $s_0, \ldots, s_{k+1}$ for

$$2. \quad P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \models_{\mathbb{L}} P_{k+1}$$

Then this property is satisfied by $s_0$:

$$F[x_0] := \exists x_1, \ldots, x_{k+1}(P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \land \neg P_{k+1})$$

(\text{Naive}) \text{ Algorithm:}

1. find a $G[x]$ in $\mathbb{L}$ satisfied by $s_0$ and s.t. $G[x] \models_{\mathbb{L}} F[x]$
2. restart the process with $P[x] \land \neg G[x]$ in place of $P[x]$
Correctness of Property Strengthening

\[ F[x_0] := \exists x_1, \ldots, x_{k+1} \left( P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \land \neg P_{k+1} \right) \]

When \( F \) is satisfied by some \( s_0 \), we

1. find a \( G[x] \) in \( \mathbb{L} \) satisfied by \( s_0 \) and s.t. \( G[x] \models_{\mathbb{L}} F[x] \)
2. replace \( P[x] \) with \( Q[x] := P[x] \land \neg G[x] \)
3. “restart” the \( k \)-induction process

- If all states satisfying \( G \) are unreachable, we can remove them from consideration in the inductive step
- Otherwise, \( P \) is not invariant and the base case is guaranteed to fail with \( Q \)
Viability of Property Strengthening

\[ F[x_0] := \exists x_1, \ldots, x_{k+1} (P_0 \land \cdots \land P_k \land T_1 \land \cdots \land T_{k+1} \land \neg P_{k+1}) \]

When \( F \) is satisfied by some \( s_0 \), we

1. find a \( G[x] \) in \( \mathbb{L} \) satisfied by \( s_0 \) and s.t. \( G[x] \models \mathbb{L} F[x] \)
2. replace \( P[x] \) with \( Q[x] := P[x] \land \neg G[x] \)
3. “restart” the \( k \)-induction process

- Normally, computing a \( G \) equivalent to \( F \) requires QE, which may be impossible or very expensive

- Under-approximating \( F \) might be cheaper but less effective in pruning unreachable states.
Multiple Property Checking

Often one wants to prove several properties $P^1, \ldots, P^n$
Multiple Property Checking

Often one wants to prove several properties $P_1, \ldots, P^n$

Proving them separately is time consuming and ineffective
Multiple Property Checking

Often one wants to prove several properties $P^1, \ldots, P^n$

Proving them separately is time consuming and ineffective

Proving them together as $P := P^1 \land \cdots \land P^n$ is inadequate if
Multiple Property Checking

Often one wants to prove several properties $P^1, \ldots, P^n$

Proving them separately is time consuming and ineffective

Proving them together as $P := P^1 \land \cdots \land P^n$ is inadequate if

- some are invariants and some are not:
  then the whole $P$ is not invariant


Multiple Property Checking

Often one wants to prove several properties $P^1, \ldots, P^n$

Proving them separately is time consuming and ineffective

Proving them together as $P := P^1 \land \cdots \land P^n$ is inadequate if

- some are invariants and some are not: then the whole $P$ is not invariant
- they are $k$-inductive for different $k$’s: then $P$ is $k$-inductive only for the largest $k$
Multiple Property Checking

Often one wants to prove several properties $P^1, \ldots, P^n$

Proving them separately is time consuming and ineffective

Proving them together as $P := P^1 \land \cdots \land P^n$ is inadequate if

- some are invariants and some are not: then the whole $P$ is not invariant
- they are $k$-inductive for different $k$’s: then $P$ is $k$-inductive only for the largest $k$

Solution: Incremental multi-property $k$-induction
Incremental Multi-Property $k$-Induction

Main idea:
Incremental Multi-Property $k$-Induction

Main idea:

- Use $P^1 \land \cdots \land P^n$ but be aware of its components

- When basic case fails,
  1. identify falsified properties
  2. remove them from the problem
  3. repeat the step
Incremental Multi-Property $k$-Induction

Main idea:

- Use $P^1 \land \cdots \land P^n$ but be aware of its components

- When basic case fails,
  1. identify falsified properties
  2. remove them from the problem
  3. repeat the step

- When inductive step fails,
  1. set falsified properties aside for next iteration (with increased $k$)
  2. repeat step and (1) until success or no more properties
  3. add proven properties as invariants for next iteration
Incremental Multi-Property $k$-Induction

Pros:

• Much better from an HCI point of view
• Proving multiple invariants in conjunction is easier than proving them separately
• Adding proven properties as invariants often obviates the need for externally provided invariants
Incremental Multi-Property $k$-Induction

Pros:

• Much better from an HCI point of view
• Proving multiple invariants in conjunction is easier than proving them separately
• adding proven properties as invariants often obviates the need for externally provided invariants

Cons:

• More complex implementation
• Having several unrelated properties can diminish the effectiveness of simplifications based on the *cone of influence*
Talk Roadmap

√ Checking safety properties

√ Logic-based model checking

√ Satisfiability Modulo Theories
  √ theories
  √ solvers

• SMT-based model checking
  √ main approaches
  √ k-induction
    √ basic method
    √ enhancements

• interpolation
Approximating $R$ with Interpolation

Recall: If $R[x]$ is the strongest inductive invariant for $M$ in $\mathbb{L}$, $M$ is safe wrt some $E[x]$ iff $R[x] \land E[x] \models_{\mathbb{L}} \bot$ ($\bot = \text{false}$)

Problem: Such invariant may be very expensive or impossible to compute, or not even representable in $\mathbb{L}$
Approximating $R$ with Interpolation

Recall: If $R[x]$ is the strongest inductive invariant for $\mathcal{M}$ in $\mathbb{L}$, $\mathcal{M}$ is safe wrt some $E[x]$ iff $R[x] \land E[x] \models_{\mathbb{L}} \bot$ ($\bot = \text{false}$)

Problem: Such invariant may be very expensive or impossible to compute, or not even representable in $\mathbb{L}$

Observation: It suffices to compute an $\hat{R}[x]$ such that

- $R[x] \models_{\mathbb{L}} \hat{R}[x]$ ($\hat{R}$ over-approximates $R$)
- $\hat{R}[x] \land E[x] \models_{\mathbb{L}} \bot$ ($\hat{R}$ is disjoint with $E$)
Approximating $R$ with Interpolation

Recall: If $R[x]$ is the strongest inductive invariant for $\mathcal{M}$ in $\mathbb{L}$, $\mathcal{M}$ is safe wrt some $E[x]$ iff $R[x] \land E[x] \models_{\mathbb{L}} \bot$ ($\bot = \text{false}$)

Problem: Such invariant may be very expensive or impossible to compute, or not even representable in $\mathbb{L}$

Observation: It suffices to compute an $\hat{R}[x]$ such that

- $R[x] \models_{\mathbb{L}} \hat{R}[x]$  \hspace{1cm} (\hat{R} \text{ over-approximates } R)
- $\hat{R}[x] \land E[x] \models_{\mathbb{L}} \bot$ \hspace{1cm} (\hat{R} \text{ is disjoint with } E)

A solution: Use theory interpolants to compute $\hat{R}[x]$
Logical Interpolation (simplified)

A logic $\mathbb{L}$ has interpolation if

for all $A[y, x]$ and $B[x, z]$ in $\mathbb{L}$ with $A[y, x] \land B[x, z] \models_{\mathbb{L}} \bot$

there is a $P[x]$ in $\mathbb{L}$ such that

$$A[y, x] \models_{\mathbb{L}} P[x] \quad \text{and} \quad P[x] \land B[x, z] \models_{\mathbb{L}} \bot$$

$P$ is an interpolant of $A$ and $B$
Logical Interpolation (simplified)

A logic $\mathbb{L}$ has interpolation if

for all $A[y, x]$ and $B[x, z]$ in $\mathbb{L}$ with $A[y, x] \land B[x, z] \models_{\mathbb{L}} \bot$

there is a $P[x]$ in $\mathbb{L}$ such that

$$A[y, x] \models_{\mathbb{L}} P[x] \quad \text{and} \quad P[x] \land B[x, z] \models_{\mathbb{L}} \bot$$

$P$ is an interpolant of $A$ and $B$

Intuitively, $P$

- is an abstraction of $A$ from the viewpoint of $B$
- summarizes and explains in terms of the shared variables $x$ why $A$ is inconsistent with $B$
Logical Interpolation (simplified)

A logic $\mathbb{L}$ has interpolation if

for all $A[y,x]$ and $B[x,z]$ in $\mathbb{L}$ with $A[y,x] \land B[x,z] \models_{\mathbb{L}} \bot$

there is a $P[x]$ in $\mathbb{L}$ such that

$A[y,x] \models_{\mathbb{L}} P[x]$ and $P[x] \land B[x,z] \models_{\mathbb{L}} \bot$

$P$ is an interpolant of $A$ and $B$

Note: If $\mathbb{L}$ has quantifier elimination, the strongest interpolant (wrt $\models_{\mathbb{L}}$) is equivalent to $\exists y. A[y,x]$
Logical Interpolation (simplified)

A logic $\mathcal{L}$ has interpolation if

for all $A[y, x]$ and $B[x, z]$ in $\mathcal{L}$ with $A[y, x] \land B[x, z] \models_{\mathcal{L}} \bot$

there is a $P[x]$ in $\mathcal{L}$ such that

$$A[y, x] \models_{\mathcal{L}} P[x] \quad \text{and} \quad P[x] \land B[x, z] \models_{\mathcal{L}} \bot$$

$P$ is an interpolant of $A$ and $B$

Note: If $\mathcal{L}$ has quantifier elimination, the strongest interpolant (wrt $\models_{\mathcal{L}}$) is equivalent to $\exists y. A[y, x]$

Interpolation is an over-approximation of quantifier elimination
Logics with Interpolation

The quantifier-free fragment of several theories used in SMT has the interpolation properties and computable interpolants:

- EUF [McM05b, FGG+09]
- linear integer arithmetic with $\text{div}_n$ [JCG09]
- real arithmetic [McM05b]
- arrays with $\text{diff}$ [BGR11]
- combinations of any of the above [YM05, GKT09]
- ...
Interpolation-based Model Checking

Let \((I[x], T[x, x'])\) be an encoding in \(\mathbb{L}\) of a system \(\mathcal{M}\)

Consider the \textit{bounded reachability} formulas \((R^i[x])_i\) where

- \(R^0[x] := I[x]\)
- \(R^{i+1}[x] := R^i[x] \lor \exists y(R^i[y] \land T[y, x])\)
Interpolation-based Model Checking

Let \((I[x], T[x, x'])\) be an encoding in \(\mathbb{L}\) of a system \(\mathcal{M}\)

Consider the \textit{bounded reachability} formulas \((R^i[x])_i\) where

- \(R^0[x] := I[x]\)
- \(R^{i+1}[x] := R^i[x] \lor \exists y (R^i[y] \land T[y, x])\)

We prove safety wrt an error property \(E\) by using interpolation [McM05a] to compute a sequence \((\hat{R}^i)_{i \geq 0}\) such that

- each \(\hat{R}^i\) overapproximates \(R^i\) and is disjoint with \(E\)
- the sequence is increasing wrt \(\models_{\mathbb{L}}\)
- the sequence has a fixpoint \(\hat{R}\) (modulo equivalence in \(\mathbb{L}\))
Constructing \((\hat{R}^i)_{i \geq 0}\)

Fix some \(k > 0\), \(\hat{R}^0 := I[x]\)

**Base Case.**

\[
A := \hat{R}^0[x_0] \land T[x_0, x_1]
\]
\[
B := T[x_1, x_2] \land \cdots \land T[x_{k-1}, x_k] \land (E[x_1] \lor \cdots \lor E[x_k])
\]

if \(A \land B\) is satisfiable in \(L\) then

\[
\text{fail} \quad (M \text{ is not safe wrt } E)
\]

else

compute an interpolant \(P[x_1]\) of \(A\) and \(B\)

\[
\hat{R}^1 := \hat{R}^0[x] \lor P[x]
\]
Constructing \((\hat{R}^i)_{i \geq 0}\)

Step Case.

for \(i = 1\) to \(\infty\)

\[
A := \hat{R}^i[x_0] \land T[x_0, x_1]
\]

\[
B := T[x_1, x_2] \land \cdots \land T[x_{k-1}, x_k] \land (E[x_1] \lor \cdots \lor E[x_k])
\]

if \(A \land B\) is satisfiable in \(\mathbb{L}\) then

restart the whole process with a larger \(k\)

else

compute an interpolant \(P[x_1]\) of \(A\) and \(B\)

\[
\hat{R}^{i+1} := \hat{R}^i[x] \lor P[x]
\]

if \(\hat{R}^{i+1} \models_{\mathbb{L}} \hat{R}^i[x]\) then succeed (fixpoint found)
Notes on the Interpolation Method

• It needs an interpolating SMT solver

• It is not incremental: a counter-example in the step case requires a real restart

• Like $k$-induction, it can be made terminating when $\mathcal{M}$ has finite bisimulation quotient

• In the terminating cases, it converges more quickly than basic $k$-induction
  ($k$ is bounded by $\mathcal{M}$’s radius, not just the reoccurrence radius as in $k$-induction)
Conclusions

• SMT-based Model Checking is the new frontier in safety checking thanks to powerful and versatile SMT solvers

• Several SAT-based methods can be lifted to the SMT case

• SMT encodings of transitions systems are basically 1-to-1

• Reasoning is at the same level of abstraction as in the original system

• Scalability and scope are higher than approaches based on propositional logic

• Several approaches and enhancements are being tried, capitalizing on different features of SMT solvers

• Lots of anecdotal evidence of successful applications
Future Directions

• Quantifiers are often needed to encode
  • parametrized model checking problems
    (coming, e.g., from multi-process systems)
  • problems with arrays

• New SMT techniques are needed to generate/work with quantified transition relations, interpolants, invariants, ...

• Synergistic combinations with traditional abstract interpretation tools seem possible

• We are starting to see some promising work in these directions, but much is left to do
References


References


References


References


References


