SMT-based Model Checking

Cesare Tinelli

The University of Iowa

Modeling Computational Systems

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

- $\mathcal{S}$ is a set of *states*, the state space
- $\mathcal{I} \subseteq \mathcal{S}$ is a set of *initial states*
- $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{S}$ is a (right-total) *transition relation*
- $\mathcal{L} : \mathcal{S} \to 2^{\mathcal{P}}$ is a *labeling function* where $\mathcal{P}$ 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 $\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$

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

Safety ~~Model~~ Checking

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

- for a suitable model $\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ 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 $\mathcal{M} = (\mathcal{S}, \mathcal{I}, \mathcal{T}, \mathcal{L})$ be a transition system

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

1. $\mathcal{I} \subseteq \mathcal{R}_\mathcal{I}$ (initial states are reachable)
2. $\mathcal{R}_\mathcal{I} \bowtie \mathcal{T} \subseteq \mathcal{R}_\mathcal{I}$ ($\mathcal{T}$-successors of reachable states are reachable)

Let $\mathcal{E} \subseteq \mathcal{S}$ (a *state property*)

The set $\mathcal{B}_\mathcal{E}$ of *bad states wrt $\mathcal{E}$* is the smallest subset of $\mathcal{S}$ such that

1. $\mathcal{E} \subseteq \mathcal{B}_\mathcal{E}$ (the states of $\mathcal{E}$ are bad)
2. $\mathcal{T} \bowtie \mathcal{B}_\mathcal{E} \subseteq \mathcal{B}_\mathcal{E}$ ($\mathcal{T}$-predecessors of bad states are bad)

Safety and Invariance

$\mathcal{M}$ is *safe* wrt a state property $\mathcal{E}$ if $\mathcal{R}_{\mathcal{I}} \cap \mathcal{E} = \emptyset$

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

A state property $\mathcal{P}$ is *invariant (for $\mathcal{M}$)* iff $\mathcal{R}_{\mathcal{I}} \subseteq \mathcal{P}$

Note:

$\mathcal{M}$ is safe wrt $\mathcal{E}$ iff $\mathcal{S} \setminus \mathcal{E}$ is invariant for $\mathcal{M}$

Checking Safety

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

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

(Forward rechability)

Checking Safety

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

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

(Forward rechability)

or

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

(Backward rechability)

Checking Safety

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

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

(Forward rechability)

or

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

(Backward rechability)

This can be done explicitly only if $\mathcal{S}$ is finite, and relatively small ($< 10M$ states)

Checking Safety

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

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

(Forward rechability)

or

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

(Backward rechability)

Alternatively, we can represent $\mathcal{M}$ symbolically and use

- BDD-based methods, if $\mathcal{S}$ is finite,
- automata-based methods,
- logic-based methods, or
- abstract interpretation methods

Checking Safety

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

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

(Forward rechability)

or

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

(Backward rechability)

Alternatively, we can represent $\mathcal{M}$ symbolically and use

- BDD-based methods, if $\mathcal{S}$ is finite,
- automata-based methods,
- logic-based methods, or
- abstract interpretation methods

Logic-based Symbolic Model Checking

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

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

Logic-based Symbolic Model Checking

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

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

Examples of $\mathbb{L}$:

- Propositional logic
- Quantified Boolean Formulas
- Bernay-Schönfinkel logic
- Quantifier-free real (or linear integer) arithmetic with arrays and uninterpreted functions
- ...

Logical encodings of transitions systems

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

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

Logical encodings of transitions systems

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

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

- states $\sigma \in \mathcal{S}$ identified with $\mathcal{L}(\sigma)$ and encoded as n-tuples of V^n

Logical encodings of transitions systems

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

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

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

$$\sigma \in \mathcal{I} \text{ iff } \models_{\mathbb{L}} I[\sigma]$$

Logical encodings of transitions systems

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

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

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

$$\sigma \in \mathcal{I} \text{ iff } \models_{\mathbb{L}} I[\sigma]$$

- $\mathcal{T}$ encoded as a formula $T[\mathbf{x}, \mathbf{x}']$ such that

$$\models_{\mathbb{L}} T[\sigma, \sigma'] \text{ for all } (\sigma, \sigma') \in \mathcal{T}$$

Logical encodings of transitions systems

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

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

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

$$\sigma \in \mathcal{I} \text{ iff } \models_{\mathbb{L}} I[\sigma]$$

- $\mathcal{T}$ encoded as a formula $T[\mathbf{x}, \mathbf{x}']$ such that

$$\models_{\mathbb{L}} T[\sigma, \sigma'] \text{ for all } (\sigma, \sigma') \in \mathcal{T}$$

- State properties encoded as formulas $P[\mathbf{x}]$

Strongest Inductive Invariant

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

Strongest Inductive Invariant

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

Suppose we can compute R from I and T

Then, checking that $\mathcal{M}$ is safe wrt a property $P[\mathbf{x}]$ reduces to checking that $R[\mathbf{x}] \models_{\mathbb{L}} \neg P[\mathbf{x}]$

Strongest Inductive Invariant

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

Suppose we can compute R from I and T

Then, checking that $\mathcal{M}$ is safe wrt a property $P[\mathbf{x}]$ reduces to checking that $R[\mathbf{x}] \models_{\mathbb{L}} \neg P[\mathbf{x}]$

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

Strongest Inductive Invariant

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

Suppose we can compute R from I and T

Then, checking that $\mathcal{M}$ is safe wrt a property $P[\mathbf{x}]$ reduces to checking that $R[\mathbf{x}] \models_{\mathbb{L}} \neg P[\mathbf{x}]$

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 $a[2*j+1] + x \geq car(l) - f(x)$

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 Σ

> The $\mathcal{T}$-satisfiability problem for a class $\mathcal{C}$ of Σ-formulas:
> decide for $\varphi[\mathbf{x}] \in \mathcal{C}$ whether $\mathcal{T} \cup \{\exists \mathbf{x}.\, \varphi\}$ is satisfiable

Satisfiability Modulo Theories

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

The $\mathcal{T}$-satisfiability problem for a class $\mathcal{C}$ of Σ-formulas:
decide for $\varphi[\mathbf{x}] \in \mathcal{C}$ whether $\mathcal{T} \cup \{\exists \mathbf{x}.\, \varphi\}$ is satisfiable

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

Satisfiability Modulo Theories

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

The $\mathcal{T}$-satisfiability problem for a class $\mathcal{C}$ of Σ-formulas: decide for $\varphi[\mathbf{x}] \in \mathcal{C}$ whether $\mathcal{T} \cup \{\exists \mathbf{x}.\, \varphi\}$ is satisfiable

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
- Inductive data types (enumerations, lists, trees, . . .)
- . . .

Satisfiability Modulo Theories

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

> The $\mathcal{T}$-satisfiability problem for a class $\mathcal{C}$ of Σ-formulas:
> decide for $\varphi[\mathbf{x}] \in \mathcal{C}$ whether $\mathcal{T} \cup \{\exists \mathbf{x}.\, \varphi\}$ is satisfiable

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

SMT Solvers

Differ from traditional theorem provers for having built-in theories, and using specialized methods to reason about them

SMT Solvers

Differ from traditional theorem provers for having built-in theories, and using specialized methods to reason about them

Are typically built to be embeddable in larger systems: they are automatic, on-line, incremental, restartable, . . .

SMT Solvers

Differ from traditional theorem provers for having built-in theories, and using specialized methods to reason about them

Are typically built to be embeddable in larger systems: they are automatic, on-line, incremental, restartable, . . .

Provide additional functionalities besides satisfiability checking

- compute satisfying assignments
- evaluate terms
- identify unsatisfiable cores
- generate interpolants
- construct proof objects
- . . .

SMT Solvers

Differ from traditional theorem provers for having built-in theories, and using specialized methods to reason about them

Are typically built to be embeddable in larger systems: they are automatic, on-line, incremental, restartable, . . .

Provide additional functionalities besides satisfiability checking

Are being extended to reason efficiently, if incompletely, with quantified formulas as well

SMT Solvers

Differ from traditional theorem provers for having built-in theories, and using specialized methods to reason about them

Are typically built to be embeddable in larger systems: they are automatic, on-line, incremental, restartable, ...

Provide additional functionalities besides satisfiability checking

Are being extended to reason efficiently, if incompletely, with quantified formulas as well

Are now the backend of a variety of FM tools :

model checkers, equivalence checkers, extended static checkers, type checkers, program verifiers, symbolic simulators, malware detectors, test case generators, invariant generators, ...

SMT Solvers

Differ from traditional theorem provers for having built-in theories, and using specialized methods to reason about them

Are typically built to be embeddable in larger systems: they are automatic, on-line, incremental, restartable, . . .

Provide additional functionalities besides satisfiability checking

Are being extended to reason efficiently, if incompletely, with quantified formulas as well

Are now the backend of a variety of FM tools

Increasingly conform to a standard I/O language: the SMT-LIB format [BST10]

Modern SMT Solvers

Such as Alt-Ergo, CVC3, MathSat, OpenSMT, VeriT, Yices, Z3, . . . ,

- are based on many-sorted first-order logic
- support a combination of several built-in theories
- allow user-defined function and predicate symbols
- follow a stack-based, assert-and-query execution model
- provide a rich API

Modern SMT Solvers

Such as Alt-Ergo, CVC3, MathSat, OpenSMT, VeriT, Yices, Z3, . . . ,

- provide a rich API

  declare: symbol → type → unit

  define: symbol → λ-term → unit

  assert: formula → unit

  push: unit → unit

  pop: unit → unit

  check_sat: unit → unit

  eval: term → value

  next_model: unit → unit

  . . .

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 k-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 k-induction)
- Interpolation-based model checking

Will focus more on temporal induction

Technical Preliminaries

Let's fix

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

Example: Parametric Resettable Counter

Model

Vars

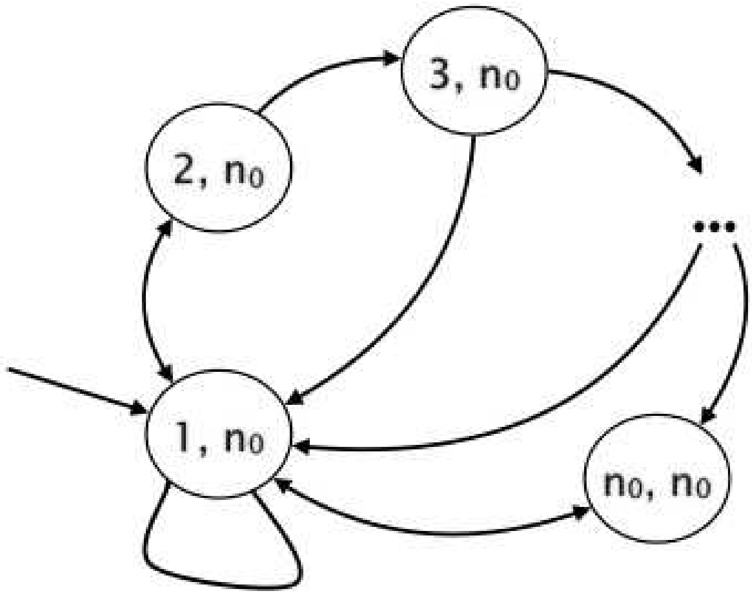
input pos int n_0
input bool r
int c, n

Initialization

c := 1
n := n_0

Transitions

n' := n
c' := if (r' or c = n)
then 1
else c + 1



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' := if (r' or c = n)
then 1
else c + 1

Encoding in $\mathbb{L}$

$$\mathbf{x} := (c, n, r, n_0)$$

$$I[\mathbf{x}] := (c = 1) \wedge (n = n_0)$$

$$\begin{aligned} T[\mathbf{x}, \mathbf{x}'] &:= (n' = n) \\ &\wedge \ (r' \vee (c = n) \rightarrow (c' = 1)) \\ &\wedge \ (\neg r' \wedge (c \neq n) \rightarrow (c' = c + 1)) \end{aligned}$$

$$P[\mathbf{x}] := c < n + 1$$

Inductive Reasoning

Let $S = (I[\mathbf{x}], T[\mathbf{x}, \mathbf{x}'])$

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

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

   and

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

Inductive Reasoning

Let $S = (I[\mathbf{x}], T[\mathbf{x}, \mathbf{x}'])$

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

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

   and

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

An SMT solver can check both entailments above
($\varphi \models_{\mathbb{L}} \psi$ iff $\varphi \wedge \neg\psi$ is unsatisfiable in $\mathbb{L}$)

Inductive Reasoning

Let $S = (I[\mathbf{x}], T[\mathbf{x}, \mathbf{x}'])$

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

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

   and

2. $P[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} P[\mathbf{x}']$ (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 $(c, n, r) = (4, 3, \mathit{false})$ and $(c, n, r)' = (5, 3, \mathit{false})$

Improving Induction's Precision

1. $I[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ 2. $P[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} P[\mathbf{x}']$

A few options:

Improving Induction's Precision

1. $I[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ 2. $P[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} P[\mathbf{x}']$

A few options:

- Strengthen P: find a property Q such that $Q[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ and prove Q inductive

Improving Induction's Precision

1. $I[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ 2. $P[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} P[\mathbf{x}']$

A few options:

- Strengthen P: find a property Q such that $Q[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ and prove Q inductive

  Difficult to automate

Improving Induction's Precision

1. $I[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ 2. $P[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} P[\mathbf{x}']$

A few options:

- Strengthen P: find a property Q such that $Q[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ and prove Q inductive

  Difficult to automate

- Strengthen T: find another invariant $Q[\mathbf{x}]$ and use $Q[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge Q[\mathbf{x}']$ instead of $T[\mathbf{x}, \mathbf{x}']$

Improving Induction's Precision

1. $I[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ 2. $P[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} P[\mathbf{x}']$

A few options:

- Strengthen P: find a property Q such that $Q[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ and prove Q inductive

  Difficult to automate

- Strengthen T: find another invariant $Q[\mathbf{x}]$ and use $Q[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge Q[\mathbf{x}']$ instead of $T[\mathbf{x}, \mathbf{x}']$

  Difficult to automate (but lots of recent progress)

Improving Induction's Precision

1. $I[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ 2. $P[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} P[\mathbf{x}']$

A few options:

- Strengthen P: find a property Q such that $Q[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ and prove Q inductive

  Difficult to automate

- Strengthen T: find another invariant $Q[\mathbf{x}]$ and use $Q[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge Q[\mathbf{x}']$ instead of $T[\mathbf{x}, \mathbf{x}']$

  Difficult to automate (but lots of recent progress)

- Consider longer T-paths: k-induction

Improving Induction's Precision

1. $I[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ 2. $P[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \models_{\mathbb{L}} P[\mathbf{x}']$

A few options:

- Strengthen P: find a property Q such that $Q[\mathbf{x}] \models_{\mathbb{L}} P[\mathbf{x}]$ and prove Q inductive

  Difficult to automate

- Strengthen T: find another invariant $Q[\mathbf{x}]$ and use $Q[\mathbf{x}] \wedge T[\mathbf{x}, \mathbf{x}'] \wedge Q[\mathbf{x}']$ instead of $T[\mathbf{x}, \mathbf{x}']$

  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: $I_i := I[\mathbf{x}_i],\ P_i := P[\mathbf{x}_i],\ T_i := T[\mathbf{x}_{i-1}, \mathbf{x}_i]$

```
for i = 0 to ∞ do
    if not (I_0 ∧ T_1 ∧ ··· ∧ T_i ⊨_𝕃 P_i) then
        return fail
    if (P_0 ∧ ··· ∧ P_i ∧ T_1 ∧ ··· ∧ T_{i+1} ⊨_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: $I_i := I[\mathbf{x}_i]$, $P_i := P[\mathbf{x}_i]$, $T_i := T[\mathbf{x}_{i-1}, \mathbf{x}_i]$

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

Basic k-Induction with SMT Solvers

Solver maintains current set of *asserted* formulas

Two solver instances: b, i

(0) $\textsf{assert}_\mathrm{b}(I_0)$
(0) $k := 0$
(0) **loop**
(0) $\quad\textsf{assert}_\mathrm{b}(T_k)$ // $T_0 = \textit{true}$ by convention
(0) $\quad$**if not** $\textsf{entailed}_\mathrm{b}(P_k)$ **then return** $\textsf{cex}_\mathrm{b}()$
(0) $\quad\textsf{assert}_\mathrm{i}(P_k)$; $\textsf{assert}_\mathrm{i}(T_{k+1})$
(0) $\quad$**if** $\textsf{entailed}_\mathrm{i}(P_{k+1})$ **then return** success
(0) $\quad k := k + 1$

$\textsf{assert}_s(F)$: adds formula F to asserted formulas
$\textsf{entailed}_s(F)$: checks if F is entailed by asserted formulas
$\textsf{cex}_s()$: returns counterexample after failed entailment

Actual k-Induction with SMT Solvers

Solver maintains current set of *asserted* formulas

Two solver instances: b, i

(0) $\text{assert}_{\mathrm{b}}(I_0)$; $\text{assert}_{\mathrm{i}}(\neg P_1)$
(0) $k := 0$
(0) **loop**
(0) $\quad \text{assert}_{\mathrm{b}}(T_k)$ $\quad // \; T_0 = \mathit{true}$ by convention
(0) $\quad$ **if not** $\text{entailed}_{\mathrm{b}}(P_k)$ **then return** $\text{cex}_{\mathrm{b}}()$
(0) $\quad \text{assert}_{\mathrm{i}}(P_{-k})$; $\text{assert}_{\mathrm{i}}(T_{-k+1})$
(0) $\quad$ **if** $\text{unsat}_{\mathrm{i}}()$ **then return** success
(0) $\quad k := k + 1$

$\text{assert}_s(F)$: adds formula F to asserted formulas
$\text{entailed}_s(F)$: checks if F is entailed by asserted formulas
$\text{cex}_s()$: returns counterexample after failed entailment
$\text{unsat}_s()$: succeeds iff asserted formulas are jointly unsatisfiable

Definition of $\mathsf{entailed}_s$

(0) **proc** $\mathsf{entailed}_s(F)$
(0) $\quad$ $\mathsf{push}()$
(0) $\quad$ $\mathsf{assert}_s(\neg F)$
(0) $\quad$ $r := \mathsf{unsat}()$
(0) $\quad$ $\mathsf{pop}()$
(0) $\quad$ **return** r

$\mathsf{unsat}_s()$: succeeds iff asserted formulas are jointly unsatisfiable

Enhancements to k-Induction

- Abstraction and refinement
- Path compression
- Termination checks
- Property strengthening
- Invariant generation
- Multiple property checking

Path Compression (simplified)

Let $E[\mathbf{x}, \mathbf{y}]$ be a formula s.t. $E[\mathbf{x}, \mathbf{y}] \models_{\mathbb{L}} \forall \mathbf{z} \, (T[\mathbf{x}, \mathbf{z}] \Leftrightarrow T[\mathbf{y}, \mathbf{z}])$

(**Ex:** $E[\mathbf{x}, \mathbf{y}] := \mathbf{x} = \mathbf{y}$)

Path Compression (simplified)

Let $E[\mathbf{x},\mathbf{y}]$ be a formula s.t. $E[\mathbf{x},\mathbf{y}] \models_{\mathbb{L}} \forall \mathbf{z}\,(T[\mathbf{x},\mathbf{z}] \Leftrightarrow T[\mathbf{y},\mathbf{z}])$
(**Ex:** $E[\mathbf{x},\mathbf{y}] := \mathbf{x} = \mathbf{y}$)

Can strengthen the premise of the inductive step as follows

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

where $C_k := \bigwedge_{0 \le i < j \le k} \neg E[\mathbf{x}_i, \mathbf{x}_j]$

Path Compression (simplified)

Let $E[\mathbf{x}, \mathbf{y}]$ be a formula s.t. $E[\mathbf{x}, \mathbf{y}] \models_{\mathbb{L}} \forall \mathbf{z}\, (T[\mathbf{x}, \mathbf{z}] \Leftrightarrow T[\mathbf{y}, \mathbf{z}])$
(**Ex:** $E[\mathbf{x}, \mathbf{y}] := \mathbf{x} = \mathbf{y}$)

Can strengthen the premise of the inductive step as follows

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

where $C_k := \bigwedge_{0 \le i < j \le k} \neg E[\mathbf{x}_i, \mathbf{x}_j]$

Rationale: Consider a path that breaks original (2)

$$\pi := \sigma_0, \ldots, \sigma_i, \sigma_{i+1}, \ldots, \sigma_j, \sigma_{j+1}, \ldots, \sigma_{k+1}$$

with $E[\sigma_i, \sigma_j]$ and $i < j$. If π is on an actual execution of $\mathcal{M}$, so is the shorter path $\sigma_0, \ldots, \sigma_i, \sigma_{j+1}, \ldots, \sigma_{k+1}$

Path Compression (simplified)

Let $E[\mathbf{x}, \mathbf{y}]$ be a formula s.t. $E[\mathbf{x}, \mathbf{y}] \models_{\mathbb{L}} \forall \mathbf{z}\, (T[\mathbf{x}, \mathbf{z}] \Leftrightarrow T[\mathbf{y}, \mathbf{z}])$
(**Ex:** $E[\mathbf{x}, \mathbf{y}] := \mathbf{x} = \mathbf{y}$)

Can further strengthen the premise of the inductive step with

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

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

Path Compression (simplified)

Let $E[\mathbf{x}, \mathbf{y}]$ be a formula s.t. $E[\mathbf{x}, \mathbf{y}] \models_{\mathbb{L}} \forall \mathbf{z}\, (T[\mathbf{x}, \mathbf{z}] \Leftrightarrow T[\mathbf{y}, \mathbf{z}])$
(**Ex:** $E[\mathbf{x}, \mathbf{y}] := \mathbf{x} = \mathbf{y}$)

Can further strengthen the premise of the inductive step with

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

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

Rationale: if
$\sigma_0, \ldots, \sigma_i, \ldots, \sigma_{k+1}$ breaks original (2) and $I[\sigma_i]$, then
$\sigma_i, \ldots, \sigma_{k+1}$ breaks the base case in the first place

Path Compression (simplified)

Let $E[\mathbf{x}, \mathbf{y}]$ be a formula s.t. $E[\mathbf{x}, \mathbf{y}] \models_{\mathbb{L}} \forall \mathbf{z}\, (T[\mathbf{x}, \mathbf{z}] \Leftrightarrow T[\mathbf{y}, \mathbf{z}])$
(**Ex:** $E[\mathbf{x}, \mathbf{y}] := \mathbf{x} = \mathbf{y}$)

Can further strengthen the premise of the inductive step with

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

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

Better E's than $\mathbf{x} = \mathbf{y}$ can be generated by an analysis of $\mathcal{M}$

More sophisticated notions of compressions have been proposed [dMRS03]

Termination check

$C_k := \bigwedge_{0 \le i < j \le k} \neg E[\mathbf{x}_i, \mathbf{x}_j]$

(0) **for** $k = 0$ **to** ∞ **do**
(0) **if not** $(I_0 \wedge T_1 \wedge \cdots \wedge T_k \models_{\mathbb{L}} P_k)$ **then**
(0) **return** fail
(0) **if** $(P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \models_{\mathbb{L}} P_{k+1})$ **then**
(0) **return** success
(0) **if** $(I_0 \wedge T_1 \wedge \cdots \wedge T_{k+1} \models_{\mathbb{L}} \neg C_{k+1})$ **then**
(0) **return** success

Termination check

$C_k := \bigwedge_{0 \le i < j \le k} \neg E[\mathbf{x}_i, \mathbf{x}_j]$

for $k = 0$ **to** ∞ **do**
  if not $(I_0 \wedge T_1 \wedge \cdots \wedge T_k \models_{\mathbb{L}} P_k)$ **then**
    return fail
  if $(P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \models_{\mathbb{L}} P_{k+1})$ **then**
    return success
  if $(I_0 \wedge T_1 \wedge \cdots \wedge T_{k+1} \models_{\mathbb{L}} \neg C_{k+1})$ **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

$C_k := \bigwedge_{0 \leq i < j \leq k} \neg E[\mathbf{x}_i, \mathbf{x}_j]$

(0) **for** $k = 0$ **to** ∞ **do**
(0) **if not** $(I_0 \wedge T_1 \wedge \cdots \wedge T_k \models_{\mathbb{L}} P_k)$ **then**
(0) **return** fail
(0) **if** $(P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \models_{\mathbb{L}} P_{k+1})$ **then**
(0) **return** success
(0) **if** $(I_0 \wedge T_1 \wedge \cdots \wedge T_{k+1} \models_{\mathbb{L}} \neg C_{k+1})$ **then**
(0) **return** success

Note: The termination check may slow down the process but increases precision in some cases
It even makes k-induction complete whenever the quotient $\mathcal{S}/E$ is finite (e.g., timed automata)

Property Strengthening

Suppose in the k-induction loop the SMT solver finds a counterexample $\sigma_0, \ldots, \sigma_{k+1}$ for

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

Property Strengthening

Suppose in the k-induction loop the SMT solver finds a counterexample $\sigma_0, \ldots, \sigma_{k+1}$ for

$$2. \quad P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \not\models_{\mathbb{L}} P_{k+1}$$

Then this property is satisfied by σ_0:

$$F[\mathbf{x}_0] := \exists \mathbf{x}_1, \ldots, \mathbf{x}_{k+1}(P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \wedge \neg P_{k+1})$$

Property Strengthening

Suppose in the k-induction loop the SMT solver finds a counterexample $\sigma_0, \ldots, \sigma_{k+1}$ for

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

Then this property is satisfied by σ_0:

$$F[\mathbf{x}_0] := \exists \mathbf{x}_1, \ldots, \mathbf{x}_{k+1}(P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \wedge \neg P_{k+1})$$

(Naive) Algorithm:

1. find a $E[\mathbf{x}]$ in $\mathbb{L}$ satisfied by σ_0 and s.t. $E[\mathbf{x}] \models_{\mathbb{L}} F[\mathbf{x}]$
2. restart the process with $P[\mathbf{x}] \wedge \neg E[\mathbf{x}]$ in place of $P[\mathbf{x}]$

Correctness of Property Strengthening

$$F[\mathbf{x}_0] := \exists \mathbf{x}_1, \ldots, \mathbf{x}_{k+1}\, (P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \wedge \neg P_{k+1})$$

When F is satisfied by some σ_0, we

1. find a $E[\mathbf{x}]$ in $\mathbb{L}$ satisfied by σ_0 and s.t. $E[\mathbf{x}] \models_{\mathbb{L}} F[\mathbf{x}]$
2. replace $P[\mathbf{x}]$ with $Q[\mathbf{x}] := P[\mathbf{x}] \wedge \neg E[\mathbf{x}]$
3. "restart" the k-induction process

- If all states satisfying E 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

$$F[\mathbf{x}_0] := \exists \mathbf{x}_1, \ldots, \mathbf{x}_{k+1} \, (P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \wedge \neg P_{k+1})$$

When F is satisfied by some σ_0, we

1. find a $E[\mathbf{x}]$ in $\mathbb{L}$ satisfied by σ_0 and s.t. $E[\mathbf{x}] \models_{\mathbb{L}} F[\mathbf{x}]$
2. replace $P[\mathbf{x}]$ with $Q[\mathbf{x}] := P[\mathbf{x}] \wedge \neg E[\mathbf{x}]$
3. "restart" the k-induction process

- Normally, computing a E 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.

(Undirected) Invariant Generation

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

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

$$P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \models_{\mathbb{L}} P_{k+1}$$

(Undirected) Invariant Generation

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

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

$$P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \models_{\mathbb{L}} P_{k+1}$$

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 $J[\mathbf{x}]$, add $J_0 \wedge \cdots \wedge J_{k+1}$ to induction hypothesis in induction step

$$P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \models_{\mathbb{L}} P_{k+1}$$

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 $\mathcal{M}$ independently from P, either before or in parallel with k-induction

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

$$P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \models_{\mathbb{L}} P_{k+1}$$

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 $\mathcal{M}$ independently from P, either before or in parallel with k-induction

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

$$P_0 \wedge \cdots \wedge P_k \wedge T_1 \wedge \cdots \wedge T_{k+1} \models_{\mathbb{L}} P_{k+1}$$

Shortcomings:

- Computed invariants may not prune the *right* unreachable states

- Adding too many invariants may swamp the SMT solver

Approximating R with Interpolation

Recall: If $R[\mathbf{x}]$ is the strongest inductive invariant for $\mathcal{M}$ in $\mathbb{L}$,

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

$\mathcal{M}$ is safe wrt some $E[\mathbf{x}]$ iff $R[\mathbf{x}] \wedge E[\mathbf{x}] \models_{\mathbb{L}} \bot$ ($\bot$ = 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 $\widehat{R}[\mathbf{x}]$ such that

- $R[\mathbf{x}] \models_{\mathbb{L}} \widehat{R}[\mathbf{x}]$ ($\widehat{R}$ over-approximates R)
- $\widehat{R}[\mathbf{x}] \wedge B[\mathbf{x}] \models_{\mathbb{L}} \bot$ ($\widehat{R}$ is *disjoint* with E)

Approximating R with Interpolation

Recall: If $R[\mathbf{x}]$ is the strongest inductive invariant for $\mathcal{M}$ in $\mathbb{L}$,

$\mathcal{M}$ is safe wrt some $E[\mathbf{x}]$ iff $R[\mathbf{x}] \wedge E[\mathbf{x}] \models_{\mathbb{L}} \bot$ ($\bot$ = 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 $\widehat{R}[\mathbf{x}]$ such that

- $R[\mathbf{x}] \models_{\mathbb{L}} \widehat{R}[\mathbf{x}]$ ($\widehat{R}$ over-approximates R)
- $\widehat{R}[\mathbf{x}] \wedge B[\mathbf{x}] \models_{\mathbb{L}} \bot$ ($\widehat{R}$ is *disjoint* with E)

A solution: Use theory interpolants to compute $\widehat{R}[\mathbf{x}]$

Logical Interpolation (simplified)

A logic $\mathbb{L}$ *has the interpolation property* if

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

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

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

P is *an interpolant* of A and B

Logical Interpolation (simplified)

A logic $\mathbb{L}$ *has the interpolation property* if

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

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

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

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 $\mathbf{x}$ why A is inconsistent with B

Logical Interpolation (simplified)

A logic $\mathbb{L}$ *has the interpolation property* if

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

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

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

P is *an interpolant* of A and B

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

Logical Interpolation (simplified)

A logic $\mathbb{L}$ *has the interpolation property* if

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

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

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

P is *an interpolant* of A and B

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

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 div_n [JCG09]
- real arithmetic [McM05b]
- arrays with diff [BGR11]
- combinations of any of the above [YM05, GKT09]
- ...

Interpolation-based Model Checking

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

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

- $R^0[\mathbf{x}] := I[\mathbf{x}]$
- $R^{i+1}[\mathbf{x}] := R^i[\mathbf{x}] \vee \exists \mathbf{y}(R^i[\mathbf{y}] \wedge T[\mathbf{y}, \mathbf{x}])$

Interpolation-based Model Checking

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

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

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

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

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

Constructing $(\widehat{R}^i)_{i\geq 0}$

Fix some $k > 0$, $\quad \widehat{R}^0 := I[\mathbf{x}]$

Base Case.

$A := \widehat{R}^0[\mathbf{x}_0] \wedge T[\mathbf{x}_0, \mathbf{x}_1]$

$B := T[\mathbf{x}_1, \mathbf{x}_2] \wedge \cdots \wedge T[\mathbf{x}_{k-1}, \mathbf{x}_k] \wedge (E[\mathbf{x}_1] \vee \cdots \vee E[\mathbf{x}_k])$

if $A \wedge B$ is satisfiable in $\mathbb{L}$ **then**

fail (M is not safe wrt E)

else

compute an interpolant $P[\mathbf{x}_1]$ of A and B

$\widehat{R}^1 := \widehat{R}^0[\mathbf{x}] \vee P[\mathbf{x}]$

Constructing $(\widehat{R}^i)_{i \geq 0}$

Step Case.

for $i = 1$ **to** ∞

$A := \widehat{R}^i[\mathbf{x}_0] \wedge T[\mathbf{x}_0, \mathbf{x}_1]$

$B := T[\mathbf{x}_1, \mathbf{x}_2] \wedge \cdots \wedge T[\mathbf{x}_{k-1}, \mathbf{x}_k] \wedge (E[\mathbf{x}_1] \vee \cdots \vee E[\mathbf{x}_k])$

if $A \wedge B$ is satisfiable in $\mathbb{L}$ **then**

restart the whole process with a larger k

else

compute an interpolant $P[\mathbf{x}_1]$ of A and B

$\widehat{R}^{i+1} := \widehat{R}^i[\mathbf{x}] \vee P[\mathbf{x}]$

if $\widehat{R}^{i+1} \models_{\mathbb{L}} \widehat{R}^i[\mathbf{x}]$ **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
- 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 reoccurence 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

[AMP06] A. Armando, J. Mantovani, and L. Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. In *Proceedings of the 13th International SPIN Workshop on Model Checking of Software (SPIN'06)*, volume 3925 of *LNCS*, pages 146–162. Springer, 2006

[ACJT96] P. A. Abdulla, K. Cerans, B. Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pages 313–321. IEEE Computer Society, 1996

[Bie09] A. Biere. Bounded model checking. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185, chapter 14, pages 455–481. IOS Press, February 2009

[BM07] A. Bradley and Z. Manna. Checking safety by inductive generalization of counterexamples to induction. In *Proceedings of the 7th International Conference on Formal Methods in Computer-Aided Design*, pages 173–180, 2007

[Bra10] A. Bradley. Sat-based model checking without unrolling. In *In Proc. Verification, Model-Checking, and Abstract-Interpretation (VMCAI)*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer-Verlag, 2010

References

[**BSST09**] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185, chapter 26, pages 825–885. IOS Press, February 2009

[**BST10**] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010

[**BGR11**] Roberto Bruttomesso, Silvio Ghilardi, and Silvio Ranise. Rewriting-based quantifier-free interpolation for a theory of arrays. In Manfred Schmidt-Schauß, editor, *Proc. of the 22nd Int. Conf. on Rewriting Techniques and Applications*, volume 10 of *LIPIcs*, pages 171–186. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011

[**CBRZ01**] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001

[**dMRS03**] L. de Moura, H. Rueß, and M. Sorea. Bounded model checking and induction: From refutation to verification. In *Proceedings of the 15th International Conference on Computer-Aided Verification (CAV 2003)*, volume 2725 of *Lecture Notes in Computer Science*. Springer, 2003

References

[**EMB11**] Niklas Een, Alan Mishchenko, and Robert Brayton. Efficient implementation of property directed reachability. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, pages 125–134, 2011

[**FGG+09**] Alexander Fuchs, Amit Goel, Jim Grundy, Sava Krstić, and Cesare Tinelli. Ground interpolation for the theory of equality. In S. Kowalewski and A. Philippou, editors, *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (York, UK)*, volume 5505 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2009

[**GR10**] S. Ghilardi and S. Ranise. Backward reachability of array-based systems by smt solving: Termination and invariant synthesis. *Logical Methods in Computer Science*, 6(4), 2010

[**GKT09**] Amit Goel, Sava Krstić, and Cesare Tinelli. Ground interpolation for combined theories. In R. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction (Montreal, Canada)*, volume 5663 of *Lecture Notes in Artificial Intelligence*, pages 183–198. Springer, 2009

References

[**HT08**] G. Hagen and C. Tinelli. Scaling up the formal verification of Lustre programs with SMT-based techniques. In *Proceedings of the 8th International Conference on Formal Methods in Computer-Aided Design (FMCAV'08), Portland, Oregon*, pages 109–117. IEEE, 2008

[**JCG09**] Himanshu Jain, Edmund M. Clarke, and Orna Grumberg. Efficient craig interpolation for linear diophantine (dis)equations and linear modular equations. *Formal Methods in System Design*, 35:6–39, August 2009

[**KGT11**] Temesghen Kahsai, Yeting Ge, and Cesare Tinelli. Instantiation-based invariant discovery. In M. Bobaru, K. Havelundand G. Holzmann, and R. Joshi, editors, *Proceedings of the 3rd NASA Formal Methods Symposium (Pasadena, CA, USA)*, volume 6617 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2011

[**McM05b**] Kenneth L. McMillan. An interpolating theorem prover. *Theoretical Computer Science*, 345(1):101–121, 2005

[**McM05a**] K. McMillan. Applications of Craig interpolants in model checking. In *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (Edinburgh, UK)*, volume 3440 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005

References

[McM03] K. McMillan. Interpolation and SAT-based model checking. In *Proceedings of the 15th International Conference on Computer Aided Verification, (Boston, Massachusetts)*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003

[Seb07] R. Sebastiani. Lazy satisability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3(3-4):141–224, 2007

[SSS00] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In *Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, pages 108–125, London, UK, 2000. Springer-Verlag

[YM05] Greta Yorsh and Madanlal Musuvathi. A combination method for generating interpolants. In Robert Nieuwenhuis, editor, *Proceedings of the 20th International Conference on Automated Deduction*, volume 3632 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2005