Formal Specification and Verification

Deductive Verification: An introduction

2.02.2022

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de
Overview

Introduction to deductive verification

Idea: Succinct representation of sets of states and of transitions between states

- **Set of states:** Formula (property of all states in the set)
- **Transitions:** Formulae (relation between the old values of the variables and the new values of the variables)
Programs as transition systems

Verification problem: Program + Description of the “bad” states

Succinct representation:

\[ P = (V, pc, Init, \mathcal{R}) \quad \phi_{err} \]

- \( V \) - finite (ordered) set of program variables
- \( pc \) - program counter variable (\( pc \) included in \( V \))
- \( Init \) - initiation condition given by formula over \( V \)
- \( \mathcal{R} \) - a finite set of transition relations
  - Every transition relation \( \rho \in \mathcal{R} \) is given by a formula over the variables \( V \) and their primed versions \( V' \)
- \( \phi_{err} \) - an error condition given by a formula over \( V \)
A program computation is a sequence of states $s_1 s_2 \ldots$ such that:

- The first element is an initial state, i.e., $s_1 \models \text{Init}$
- Each pair of consecutive states $(s_i, s_{i+1})$ is connected by a program transition, i.e., $(s_i, s_{i+1}) \models \rho R$.
- If the sequence is finite then the last element does not have any successors i.e., if the last element is $s_n$, then there is no state $s$ such that $(s_n, s) \models \rho R$. 

Computation
Correctness: Safety

- A state is reachable if it occurs in some program computation.
- A program is safe if no error state is reachable.
- ... if and only if no error state lies in $\phi_{reach}$,
  \[ \phi_{err} \land \phi_{reach} \models \bot \]
  where $\phi_{reach}$ = set of program states which are reachable from some initial state.
- ... if and only if no initial state lies in $\phi_{reach^{-1}}$,
  \[ Init \land \phi_{reach^{-1}}(\phi_{err}) \models \bot \]
  where $\phi_{reach^{-1}}(\phi_{err})$ = set of program states from which some state in $\phi_{err}$ is reachable.
Let $\phi$ be a formula over $V$

Let $\rho$ be a formula over $V$ and $V'$

Define a post-condition function $post$ by:

$$post(\phi, \rho) = \exists V'' : \phi[V''/V] \land \rho[V''/V][V/V']$$

An application $post(\phi, \rho)$ computes the image of the set $\phi$ under the relation $\rho$.

$post^*(\phi, \rho) = n$-fold application of post to $\phi$ under $\rho$

Characterize $\phi_{reach}$ using iterates of post:

$$\phi_{reach} = Init \lor post(Init, \rho_{\mathcal{R}}) \lor post(post(Init, \rho_{\mathcal{R}}), \rho_{\mathcal{R}}) \lor \ldots$$

$$= \bigvee_{i \geq 0} post^i(Init, \rho_{\mathcal{R}})$$
Problem

Assume there exists $m \in \mathbb{N}$ such that

$$\bigvee_{i=0}^{m} \text{post}^i(\text{Init}, \rho_{\mathcal{R}}) = \bigvee_{i=0}^{m+1} \text{post}^i(\text{Init}, \rho_{\mathcal{R}})$$

i.e. fixpoint reached.

Let $\phi_{\text{reach}} := \bigvee_{i=1}^{m} \text{post}^i(\text{Init}, \rho_{\mathcal{R}})$

**How to check whether error states are reachable?**

$\phi_{\text{reach}}, \phi_{\text{err}}$ are formulae.

No error states are reachable iff $\phi_{\text{reach}} \land \phi_{\text{err}} \models \bot$

Both for forward and for backward reachability:

Reasoning modulo theories
Reasoning modulo theories

**Goal:** Devise efficient methods for reasoning modulo theories

SAT checking (can reduce entailment to checking satisfiability)

**Example:**
Check whether conjunctions of constraints in linear arithmetic is satisfiable: classical methods exist, e.g. simplex.
Check whether a conjunction of equalities and disequalities of ground terms is satisfiable: methods exist (e.g. congruence closure)

**Challenge:** efficient methods for handling arbitrary Boolean combinations of constraints in such theories.

**Possible solution:** Extend the DPLL method to reasoning modulo theories

↔ Decision Procedures for Verification
Reminder: The DPLL algorithm

State: \( M \parallel F, \)

where:

- \( M \) partial assignment (sequence of literals),
  
  some literals are annotated \((L^d: \text{decision literal})\)

- \( F \) clause set.
A succinct formulation

UnitPropagation

\[ M \parallel F, C \lor L \Rightarrow M, L \parallel F, C \lor L \]

if \( M \models \neg C \), and \( L \) undefined in \( M \)

Decide

\[ M \parallel F \Rightarrow M, L^d \parallel F \]

if \( L \) or \( \neg L \) occurs in \( F \), \( L \) undefined in \( M \)

Fail

\[ M \parallel F, C \Rightarrow \text{Fail} \]

if \( M \models \neg C \), \( M \) contains no decision literals

Backjump

\[ M, L^d, N \parallel F \Rightarrow M, L' \parallel F \]

if there is some clause \( C \lor L' \) s.t.:

\[ F \models C \lor L', M \models \neg C, \]
\[ L' \text{ undefined in } M \]
\[ L' \text{ or } \neg L' \text{ occurs in } F. \]
Some problems are more naturally expressed in richer logics than just propositional logic, e.g:

- Software/Hardware verification needs reasoning about equality, arithmetic, data structures, ...

SMT consists of deciding the satisfiability of a ground 1st-order formula with respect to a background theory $T$. 
SAT Modulo Theories (SMT)

The “very eager” approach to SMT

Method:
– translate problem into equisatisfiable propositional formula;
– use off-the-shelf SAT solver

• Why “eager”?  
  Search uses all theory information from the beginning

• Characteristics:
  + Can use best available SAT solver
    – Sophisticated encodings are needed for each theory
    – Sometimes translation and/or solving too slow

Main Challenge for alternative approaches is to combine:
- DPLL-based techniques for handling the boolean structure
- Efficient theory solvers for conjunctions of $\ell$-literals
SAT Modulo Theories (SMT)

“Lazy” approaches to SMT: Idea

Example: consider $T = \text{UIF}$ and the following set of clauses:

$$
\begin{align*}
f(g(a)) \not\approx f(c) \lor \neg g(a) \approx d, \\
\neg P_1 \\
\end{align*}
\begin{align*}
g(a) \approx c, \\
P_2 \\
\end{align*}
\begin{align*}
c \not\approx d \\
P_3 \\
\end{align*}
$$

1. Send $\{\neg P_1 \lor P_2, \ P_3, \neg P_4\}$ to SAT solver

SAT solver returns model $[\neg P_1, P_3, \neg P_4]$.

Theory solver says $\neg P_1 \land P_3 \land \neg P_4$ is $T$-inconsistent.

2. Send $\{\neg P_1 \lor P_2, \ P_3, \neg P_4, \ P_1 \lor \neg P_3 \lor P_4\}$ to SAT solver

SAT solver returns model $[P_1, P_2, P_3, \neg P_4]$.

Theory solver says $P_1 \land P_2 \land P_3 \land \neg P_4$ is $T$-inconsistent.

3. Send $\{\neg P_1 \lor P_2, \ P_3, \neg P_4, \ P_1 \lor \neg P_3 \lor P_4, \ P_1 \lor \neg P_2 \lor \neg P_3 \lor P_4\}$ to SAT solver

SAT solver says UNSAT.
Problems

It is not guaranteed that the fixpoint is reached in a finite/bounded number of steps.
Problems

It is not guaranteed that the fixpoint is reached in a finite/bounded number of steps.

Need to analyze alternative solutions
Verification

Modeling/Formalization

System Specification

Is the system safe?

Is safety guaranteed on all paths of length < n which start in an initial state?

Is the safety property an invariant of the system?
Can we generate an invariant which implies safety?

Invariant checking/ BMC  Model Checking  Abstraction/ Refinement
Verification

Modeling/Formalization

System Specifications

Complex theories

Automated reasoning
- full theory
- abstraction of theory

Interpolation
- use interpolants for refining abstraction

Invariant checking/ BMC
Model Checking
Abstraction/ Refinement
Abstraction/Refinement

Concrete program

Abstract program

feasible path

location reachable

feasible path

location unreachable

check feasibility

⇒

conjunction of constraints: $\phi(1) \land Tr(1, 2) \land \cdots \land Tr(n-1, n) \land \neg\text{safe}(n)$

- satisfiable: feasible path

- unsatisfiable: refine abstract program s.t. the path is not feasible

[McMillan 2003-2006] use ‘local causes of inconsistency’

⇒ compute interpolants
Invariant checking; Bounded model checking

$S$ specification $\mapsto \Sigma_S$ signature of $S$; $T_S$ theory of $S$; $T_S$ transition system

$\text{Init}(\overline{x}); \rho_R(\overline{x}, \overline{x}')$

Given: Safe($x$) formula (e.g. safety property)

- Invariant checking
  
  (1) $T_S \models \text{Init}(\overline{x}) \rightarrow \text{Safe}(\overline{x})$ (Safe holds in the initial state)
  
  (2) $T_S \models \text{Safe}(\overline{x}) \land \rho_R(\overline{x}, \overline{x}') \rightarrow \text{Safe}(\overline{x}')$ (Safe holds before $\Rightarrow$ holds after update)

- Bounded model checking (BMC):

  Check whether, for a fixed $k$, unsafe states are reachable in at most $k$ steps, i.e. for all $0 \leq j \leq k$:

  $T_S \models \text{Init}(x_0) \land \rho_R(x_0, x_1) \land \cdots \land \rho_R(x_{j-1}, x_j) \land \neg\text{Safe}(x_j) \rightarrow \bot$
Reasoning modulo theories

**Goal:** Devise efficient methods for reasoning modulo theories
Problems

- First order logic is undecidable
- In applications, theories do not occur alone
  ⟷ need to consider combinations of theories

+ Fragments of theories occurring in applications are often decidable
+ Often provers for the component theories can be combined efficiently
Probleme

- First order logic is undecidable
- In applications, theories do not occur alone
  \[\rightarrow\] need to consider combinations of theories

+ Fragments of theories occurring in applications are often decidable
+ Often provers for the component theories can be combined efficiently

**Important goals:**

- Identify decidable theories which are important in applications (Extensions/Combinations) possibly with low complexity
- Development & Implementation of efficient Decision Procedures
Example: ETCS Case Study (AVACS project)

Simplified version of ETCS Case Study [Jacobs, VS’06, Faber, Jacobs, VS’07]

Number of trains: \( n \geq 0 \) \( \mathbb{Z} \)

Minimum and maximum speed of trains: \( 0 \leq \text{min} < \text{max} \) \( \mathbb{R} \)

Minimum secure distance: \( l_{\text{alarm}} > 0 \) \( \mathbb{R} \)

Time between updates: \( \Delta t > 0 \) \( \mathbb{R} \)

Train positions before and after update: \( \text{pos}(i), \text{pos}'(i) : \mathbb{Z} \rightarrow \mathbb{R} \)
Example: ETCS Case Study (AVACS project)

Simplified version of ETCS Case Study [Jacobs, VS’06, Faber, Jacobs, VS’07]

Update \((pos, pos')\) :
- \(\forall i \ (i = 0 \rightarrow pos(i) + \Delta t \cdot \text{min} \leq pos'(i) \leq pos(i) + \Delta t \cdot \text{max})\)
- \(\forall i \ (0 < i < n \land pos(i - 1) > 0 \land pos(i - 1) - pos(i) \geq l_{\text{alarm}} \rightarrow pos(i) + \Delta t \cdot \text{min} \leq pos'(i) \leq pos(i) + \Delta t \cdot \text{max})\)

...
Example: ETCS Case Study (AVACS project)

**Safety property:** No collisions

\[
\text{Safe}(\text{pos}) : \forall i, j (i < j \rightarrow \text{pos}(i) > \text{pos}(j))
\]

**Inductive invariant:**

\[
\text{Safe}(\text{pos}) \land \text{Update}(\text{pos}, \text{pos'}) \land \neg \text{Safe}(\text{pos'}) \models T_S \perp
\]

where \( T_S \) is the extension of the (disjoint) combination \( \mathbb{R} \cup \mathbb{Z} \) with two functions, \( \text{pos}, \text{pos'} : \mathbb{Z} \rightarrow \mathbb{R} \)

**Problem:** Satisfiability test for quantified formulae in complex theory
More complex ETCS Case studies

[Faber, Jacobs, VS, 2007]

- Take into account also:
  - Emergency messages
  - Durations

- Specification language: CSP-OZ-DC
  - Reduction to satisfiability in theories for which decision procedures exist

- **Tool chain:** [Faber, Ihlemann, Jacobs, VS]
  CSP-OZ-DC \(\mapsto\) Transition constr. \(\mapsto\) Decision procedures (H-PILoT)
Example 2: Parametric topology

- **Complex track topologies** [Faber, Ihlemann, Jacobs, VS, ongoing work]

**Assumptions:**
- No cycles
- In-degree (out-degree) of associated graph at most 2.
Parametricity and modularity

- **Complex track topologies** [Faber, Ihlemann, Jacobs, VS, ongoing work]

**Assumptions:**
- No cycles
- in-degree (out-degree) of associated graph at most 2.

**Approach:**
- Decompose the system in trajectories (linear rail tracks; may overlap)
- **Task 1:** - Prove safety for trajectories with incoming/outgoing trains
  - Conclude that for control rules in which trains have sufficient freedom (and if trains are assigned unique priorities) safety of all trajectories implies safety of the whole system
- **Task 2:** - General constraints on parameters which guarantee safety
Parametricity and modularity

- **Complex track topologies** [Faber, Ihlemann, Jacobs, VS, ongoing work]

Assumptions:
- No cycles
- in-degree (out-degree) of associated graph at most 2.

Data structures:
- $p_1$: trains
- 2-sorted pointers
- $p_2$: segments
- scalar fields ($f:p_i \to \mathbb{R}$, $g:p_i \to \mathbb{Z}$)
- updates efficient decision procedures (H-PiLoT)
Example: Controller for line track (RBC)

**CSP part:** specifies the processes and their interdependency.

The RBC system passes repeatedly through four phases, modeled by events:

- **updSpd** (speed update)
- **req** (request update)
- **alloc** (allocation update)
- **updPos** (position update)

Between these events, trains may leave or enter the track (at specific segments), modeled by the events **leave** and **enter**.
Example: Controller for line track (RBC)

OZ part. Consists of data classes, axioms, the Init formulae, update rules.

- **1. Data classes** declare function symbols that can change their values during runs of the system

**Data structures:**

- **2-sorted pointers**
  
  train: trains
  
  segm: segments
Example: Controller for line track (RBC)

**OZ part.** Consists of data classes, axioms, the Init formulae, update rules.

- **1. Data classes** declare function symbols that can change their values during runs of the system, and are used in the OZ part of the specification.

- **2. Axioms:** define properties of the data structures and system parameters which do not change
  - $g_{max} : \mathbb{R}$ (the global maximum speed),
  - $dec_{max} : \mathbb{R}$ (the maximum deceleration of trains),
  - $d : \mathbb{R}$ (a safety distance between trains),
  - Properties of the data structures used to model trains/segments
Example: Controller for line track (RBC)

**OZ part.** Consists of data classes, axioms, the Init formulae, update rules.

- **3. Init schema.** describes the initial state of the system.
  - trains - doubly-linked list; placed correctly on the track segments
  - all trains respect their speed limits.

- **4. Update rules** specify updates of the state space executed when the corresponding event from the CSP part is performed.
  
  Example: Speed update
**Modular Verification**

<table>
<thead>
<tr>
<th>COD</th>
<th>$\mapsto \Sigma_S$ signature of $S$; $\mathcal{T}_S$ theory of $S$; $T_S$ transition constraint system</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>specification Init($\overline{x}$); Update($\overline{x}, \overline{x}'$)</td>
</tr>
</tbody>
</table>

**Given:** Safe($x$) formula (e.g. safety property)

- **Invariant checking**
  
  \[
  (1) \models_{\mathcal{T}_S} \text{Init}(\overline{x}) \rightarrow \text{Safe}(\overline{x}) \quad \text{(Safe holds in the initial state)}
  \]
  
  \[
  (2) \models_{\mathcal{T}_S} \text{Safe}(\overline{x}) \land \text{Update}(\overline{x}, \overline{x}') \rightarrow \text{Safe}(\overline{x}') \quad \text{(Safe holds before $\Rightarrow$ holds after update)}
  \]

- **Bounded model checking (BMC):**

  Check whether, for a fixed $k$, unsafe states are reachable in at most $k$ steps, i.e. for all $0 \leq j \leq k$:

  \[
  \text{Init}(x_0) \land \text{Update}_1(x_0, x_1) \land \cdots \land \text{Update}_n(x_{j-1}, x_j) \land \neg\text{Safe}(x_j) \models_{\mathcal{T}_S} \bot
  \]
Trains on a linear track

Example 1: Speed Update

\[ \text{pos}(t) < \text{length}(\text{segm}(t)) - d \rightarrow 0 \leq \text{spd}'(t) \leq \text{lmax}(\text{segm}(t)) \]

\[ \text{pos}(t) \geq \text{length}(\text{segm}(t)) - d \land \text{alloc}(\text{next}_s(\text{segm}(t))) = \text{tid}(t) \rightarrow 0 \leq \text{spd}'(t) \leq \min(\text{lmax}(\text{segm}(t)), \text{lmax}(\text{next}_s(\text{segm}(t)))) \]

\[ \text{pos}(t) \geq \text{length}(\text{segm}(t)) - d \land \text{alloc}(\text{next}_s(\text{segm}(t))) \neq \text{tid}(t) \rightarrow \text{spd}'(t) = \max(\text{spd}(t) - \text{decmax}, 0) \]
Trains on a linear track

Example 1: Speed Update
\[\begin{align*}
pos(t) < \text{length}(\text{segm}(t)) - d &\implies 0 \leq \text{spd}'(t) \leq \text{lmax}(\text{segm}(t)) \\
pos(t) \geq \text{length}(\text{segm}(t)) - d &\land \text{alloc}(\text{next}_s(\text{segm}(t))) = \text{tid}(t) \\
&\implies 0 \leq \text{spd}'(t) \leq \min(\text{lmax}(\text{segm}(t)), \text{lmax}(\text{next}_s(\text{segm}(t)))) \\
pos(t) \geq \text{length}(\text{segm}(t)) - d &\land \text{alloc}(\text{next}_s(\text{segm}(t))) \neq \text{tid}(t) \\
&\implies \text{spd}'(t) = \max(\text{spd}(t) - \text{decmax}, 0)
\end{align*}\]

Proof task:
\[\text{Safe}(\text{pos, next, prev, spd}) \land \text{SpeedUpdate}(\text{pos, next, prev, spd, spd}') \implies \text{Safe}(\text{pos}', \text{next, prev, spd}')\]
Incoming and outgoing trains

**Example 2:** Enter Update (also updates for segm’, spd’, pos’, train’)

**Assume:** \( s_1 \neq \text{null}_s, t_1 \neq \text{null}_t, \text{train}(s) \neq t_1, \text{alloc}(s_1) = \text{idt}(t_1) \)

\( t \neq t_1, \text{ids}(\text{segm}(t)) < \text{ids}(s_1), \text{next}_t(t) = \text{null}_t, \text{alloc}(s_1) = \text{tid}(t_1) \rightarrow \text{next}'(t) = t_1 \land \text{next}'(t_1) = \text{null}_t \)

\( t \neq t_1, \text{ids}(\text{segm}(t)) < \text{ids}(s_1), \text{alloc}(s_1) = \text{tid}(t_1), \text{next}_t(t) \neq \text{null}_t, \text{ids}(\text{segm}(	ext{next}_t(t))) \leq \text{ids}(s_1) \)

\( \rightarrow \text{next}'(t) = \text{next}_t(t) \)

\( \ldots \)

\( t \neq t_1, \text{ids}(\text{segm}(t)) \geq \text{ids}(s_1) \rightarrow \text{next}'(t) = \text{next}_t(t) \)
Incoming and outgoing trains

Example 2: Enter Update (also updates for segm', spd', pos', train')

Assume: $s_1 \neq \text{null}_s$, $t_1 \neq \text{null}_t$, $\text{train}(s) \neq t_1$, $\text{alloc}(s_1) = \text{idt}(t_1)$

$t \neq t_1$, $\text{ids}(	ext{segm}(t)) < \text{ids}(s_1)$, $\text{next}_t(t) = \text{null}_t$, $\text{alloc}(s_1) = \text{tid}(t_1)$ → $\text{next}'(t) = t_1$ ∧ $\text{next}'(t_1) = \text{null}_t$

$t \neq t_1$, $\text{ids}(	ext{segm}(t)) < \text{ids}(s_1)$, $\text{alloc}(s_1) = \text{tid}(t_1)$, $\text{next}_t(t) \neq \text{null}_t$, $\text{ids}(	ext{segm}(\text{next}_t(t))) \leq \text{ids}(s_1)$ → $\text{next}'(t) = \text{next}_t(t)$

...$t \neq t_1$, $\text{ids}(	ext{segm}(t)) \geq \text{ids}(s_1)$ → $\text{next}'(t) = \text{next}_t(t)$
**Safety property**

**Safety property we want to prove:**
no two different trains ever occupy the same track segment:

\[(\text{Safe}) \quad \forall t_1, t_2 \quad \text{segm}(t_1) = \text{segm}(t_2) \implies t_1 = t_2\]

In order to prove that (Safe) is an invariant of the system, we need to find a suitable invariant \((\text{Inv}_i)\) for every control location \(i\) of the TCS, and prove:

1. \((\text{Inv}_i) \models (\text{Safe})\) for all locations \(i\) and
2. the invariants are preserved under all transitions of the system, 
   \((\text{Inv}_i) \land (\text{Update}) \models (\text{Inv}_j')\)
   whenever \((\text{Update})\) is a transition from location \(i\) to \(j\).
Safety property

Safety property we want to prove:
no two different trains ever occupy the same track segment:

(Safe) \( \forall t_1, t_2 \) \( \text{segm}(t_1) = \text{segm}(t_2) \rightarrow t_1 = t_2 \)

In order to prove that (Safe) is an invariant of the system, we need to find a suitable invariant (Inv\(_i\)) for every control location i of the TCS, and prove:

1. (Inv\(_i\)) \models (Safe) for all locations i and
2. the invariants are preserved under all transitions of the system,

\((\text{Inv}_i) \land (\text{Update}) \models (\text{Inv}_j')\)

whenever (Update) is a transition from location i to j.

Here: Inv\(_i\) generated by hand (use poss. of generating counterexamples with H-PILoT)
Verification problems

(1) \((\text{Inv}_i) \models (\text{Safe})\) for all locations \(i\) and

(2) the invariants are preserved under all transitions of the system,
\[(\text{Inv}_i) \land (\text{Update}) \models (\text{Inv}'_j)\]

whenever (\text{Update}) is a transition from location \(i\) to \(j\).

Ground satisfiability problems for pointer data structures

**Problem:** Axioms, Invariants: are universally quantified

**Our solution:** Hierarchical reasoning in local theory extensions
Examples of theories we need to handle

- **Invariants**

  \[(\text{Inv}_1) \forall t : \text{Train.} \ pc \neq \text{InitState} \land \text{alloc}(\text{next}_s(\text{segm}(t))) \neq \text{tid}(t)\]
  \[\rightarrow \text{length}(\text{segm}(t)) - \text{bd}(\text{spd}(t)) > \text{pos}(t) + \text{spd}(t) \cdot \Delta t\]

  \[(\text{Inv}_2) \forall t : \text{Train.} \ pc \neq \text{InitState} \land \text{pos}(t) \geq \text{length}(\text{segm}(t)) - \text{d}\]
  \[\rightarrow \text{spd}(t) \leq \text{lmax}(\text{next}_s(\text{segm}(t)))\]
Examples of theories we need to handle

- **Invariants**

  \[(\text{Inv}_1) \forall t : \text{Train. } pc \neq \text{InitState} \land \text{alloc}(\text{next}_s(\text{segm}(t))) \neq \text{tid}(t) \rightarrow \text{length}(\text{segm}(t)) - \text{bd}(\text{spd}(t)) > \text{pos}(t) + \text{spd}(t) \cdot \Delta t\]

  \[(\text{Inv}_2) \forall t : \text{Train. } pc \neq \text{InitState} \land \text{pos}(t) \geq \text{length}(\text{segm}(t)) - d \rightarrow \text{spd}(t) \leq \text{lmax}(\text{next}_s(\text{segm}(t)))\]

- **Update rules**

  \[\forall t : \phi_1(t) \rightarrow s_1 \leq \text{spd}'(t) \leq t_1\]

  \[\ldots\]

  \[\forall t : \phi_n(t) \rightarrow s_n \leq \text{spd}'(t) \leq t_n\]
Example 2

Hybrid systems $\mapsto$ Hybrid automata
Example 2

Chemical plant

Two substances are mixed; they react. The resulting product is filtered out; then the procedure is repeated.

Check:

• No overflow
• Substances always in the right proportion
• If substances in wrong proportion, tank can be drained in $\leq 200$ s.

Parametric description:

• Determine values for parameters such that this is the case
Example 2

Mode 1: Fill  Temperature is low, 1 and 2 do not react. Substances 1 and 2 (possibly mixed with a small quantity of 3) are filled in the tank in equal quantities up to a margin of error.

\[ \text{Inv}_1 \quad x_1 + x_2 + x_3 \leq L_f \land \bigwedge_{i=1}^{3} x_i \geq 0 \land -\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \land 0 \leq x_3 \leq \text{min} \]

\[ \text{flow}_1 \quad \bullet x_1 \geq \text{dmin} \land \bullet x_2 \geq \text{dmin} \land \bullet x_3 = 0 \land -\delta_a \leq \bullet x_1 - \bullet x_2 \leq \delta_a \]

Jumps: (1,4)

If proportion not kept: system jumps into mode 4 (Dump)

\[ e_1 \quad \text{guard}_{e_1} (x_1, x_2, x_3) = x_1 - x_2 \geq \epsilon_a \]

(from 1 to 4) \[ \text{jump}_{e_1} (x_1, x_2, x_3, x'_1, x'_2, x'_3) = \bigwedge_{i=1}^{3} x'_i = 0 \]

\[ e_2 \quad \text{guard}_{e_1} (x_1, x_2, x_3) = x_1 - x_2 \leq -\epsilon_a \]

(from 1 to 4) \[ \text{jump}_{e_1} (x_1, x_2, x_3, x'_1, x'_2, x'_3) = \bigwedge_{i=1}^{3} x'_i = 0 \]
Example

Mode 1: Fill  Temperature is low, 1 and 2 do not react. Substances 1 and 2 (possibly mixed with a small quantity of 3) are filled in the tank in equal quantities up to a margin of error.

\[
\begin{align*}
\text{Inv}_1 & \quad x_1 + x_2 + x_3 \leq L_f \land \bigwedge_{i=1}^{3} x_i \geq 0 \land \\
& \quad -\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \land 0 \leq x_3 \leq \text{min} \\
\text{flow}_1 & \quad \bullet x_1 \geq \text{dmin} \land \bullet x_2 \geq \text{dmin} \land \bullet x_3 = 0 \land -\delta_a \leq \bullet x_1 - \bullet x_2 \leq \delta_a
\end{align*}
\]

Jumps: (1,2)

If the total quantity of substances exceeds level \( L_f \) (tank filled) the system jumps into mode 2 (React).

\[
\begin{align*}
e &= (1, 2) \quad \text{guard}_{(1,2)}(x_1, x_2, x_3) = x_1 + x_2 + x_3 \geq L_f \\
& \quad \text{jump}_{(1,2)}(x_1, x_2, x_3, x_1', x_2', x_3') = \bigwedge_{i=1}^{3} x_i' = x_i
\end{align*}
\]
Example

Mode 2: React  Temperature is high. Substances 1 and 2 react. The reaction consumes equal quantities of substances 1 and 2 and produces substance 3.

\[
\text{Inv}_2: \quad L_f \leq x_1 + x_2 + x_3 \leq L_{\text{overflow}} \land \land_{i=1}^3 x_i \geq 0 \land \\
-\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \land 0 \leq x_3 \leq \text{max}
\]

\[
\text{flow}_2: \quad \bullet x_1 \leq -d_{\text{max}} \land \bullet x_2 \leq -d_{\text{max}} \land \bullet x_3 \geq d_{\text{min}} \\
\land \bullet x_1 = \bullet x_2 \land \bullet x_3 + \bullet x_1 + \bullet x_2 = 0
\]

Jumps:

If the proportion between substances 1 and 2 is not kept the system jumps into mode 4 (Dump);

If the total quantity of substances 1 and 2 is below some minimal level min the system jumps into mode 3 (Filter).
Example

Mode 3: Filter  Temperature is low. Substance 3 is filtered out.

\[ \text{Inv}_3 \quad x_1 + x_2 + x_3 \leq L_{\text{overflow}} \quad \land \quad \land_{i=1}^{3} x_i \geq 0 \quad \land \\
-\epsilon_a \leq x_1 - x_2 \leq \epsilon_a \quad \land \quad x_3 \geq \min \\
\]

\[ \text{flow}_3 \quad \bullet \; x_1 = 0 \land \bullet \; x_2 = 0 \quad \land \quad \bullet \; x_3 \leq -d_{\text{max}} \]

Jumps:

If proportion not kept: system jumps into mode 4 (Dump);

Otherwise, if the concentration of substance 3 is below some minimal level $\min$ the system jumps into mode 1 (Fill).
Mode 4: Dump The content of the tank is emptied. For simplicity we assume that this happens instantaneously:

\[ \text{Inv}_4 : \bigwedge_{i=1}^{3} x_i = 0 \text{ and } \text{flow}_4 : \bigwedge_{i=1}^{3} x_i = 0. \]
Invariant checking: Check whether $\Psi$ is an invariant in a HA $S$, i.e.:

1. $\text{Init}_q \models \psi$ for all $q \in Q$;
2. $\psi$ is invariant under jumps and flows:
   - **(Flow)** For every flow in mode $q$, the continuous variables satisfy $\psi$ during and at the end of the flow.
   - **(Jump)** For every jump according to a control switch $e$, if $\psi$ holds before the jump, it holds after the jump.

Examples:

- Is “$x_1 + x_2 + x_3 \leq L_{\text{overflow}}$” an invariant? (no overflow)
- Is “$-\epsilon_a \leq x_1 - x_2 \leq \epsilon_a$” an invariant?
  (substances always mixed in the right proportion)
Simple verification problems

**Bounded model checking:** Is formula Safe preserved under runs of length $\leq k$?, i.e.:

1. $Init_q \models Safe$ for every $q \in Q$;

2. The continuous variables satisfy Safe during and at the end of all runs of length $j$ for all $1 \leq j \leq k$.

**Example:**

- Is “$x_1 + x_2 + x_3 \leq L_{overflow}$” true after all runs of length $\leq k$ starting from a state with e.g. $x_1 = x_2 = x_3 = 0$?

- Is “$-\epsilon_a \leq x_1 - x_2 \leq \epsilon_a$” true after all runs of length $\leq k$ starting from a state with $x_1 = x_2 = x_3 = 0$?
Simple verification problems

Reductions of verification problems to linear arithmetic

(1) Mode invariants, initial states and guards of mode switches are described as conjunctions of linear inequalities.

Example: $\text{Inv}_q = \bigwedge_{j=1}^{mq} (\sum_{i=1}^{n} a_{ij}^q x_i \leq a_j^q)$ can be expressed by:

$$\text{Inv}_q(x_1(t), \ldots, x_n(t)) = \bigwedge_{j=1}^{mq} (\sum_{i=1}^{n} a_{ij}^q x_i(t) \leq a_j^q)$$
Reductions of verification problems to linear arithmetic

(2) The flow conditions are expressed by non-strict linear inequalities:

\[ \text{flow}_q = \bigwedge_{j=1}^{n_q} \left( \sum_{i=1}^n c_{ij}^q x_i \leq c_j^q \right), \text{ i.e. } \text{flow}_q(t) = \bigwedge_{j=1}^{n_q} \left( \sum_{i=1}^n c_{ij}^q \dot{x}_i(t) \leq c_j^q \right). \]
Simple verification problems

Reductions of verification problems to linear arithmetic

(2) The flow conditions are expressed by non-strict linear inequalities:

\[
\text{flow}_q = \bigwedge_{j=1}^{n_q} \left( \sum_{i=1}^{n} c_{ij}^q \dot{x}_i \leq c_j^q \right), \text{ i.e. } \text{flow}_q(t) = \bigwedge_{j=1}^{n_q} \left( \sum_{i=1}^{n} c_{ij}^q \dot{x}_i(t) \leq c_j^q \right).
\]

Approach: Express the flow conditions in \([t_0, t_1]\) without referring to derivatives.

Flow\(_q\)(t\(_0\), t\(_1\)) : \(\forall t (t_0 \leq t \leq t_1 \rightarrow \text{Inv}_q(\overline{x}(t))) \land \forall t, t' (t_0 \leq t \leq t' \leq t_1 \rightarrow \text{flow}_q(t, t'))\).

where: \(\text{flow}_q(t, t') = \bigwedge_{j=1}^{n_q} \left( \sum_{i=1}^{n} c_{ij}^q (x_i(t') - x_i(t)) \leq c_j^q (t' - t) \right)\).
Simple verification problems

Reductions of verification problems to linear arithmetic

(2) The flow conditions are expressed by non-strict linear inequalities:
\[
\text{flow}_q = \bigwedge_{j=1}^{n^q} \left( \sum_{i=1}^{n} c_{ij}^q \dot{x}_i \leq c_j^q \right), \text{ i.e. } \text{flow}_q(t) = \bigwedge_{j=1}^{n^q} \left( \sum_{i=1}^{n} c_{ij}^q \dot{x}_i(t) \leq c_j^q \right).
\]

**Approach:** Express the flow conditions in \([t_0, t_1]\) without referring to derivatives.

**Flow** \(q((t_0, t_1)) : \forall t \in [t_0, t_1] \rightarrow \text{Inv}_q(\underline{x}(t))) \land \forall t, t' \in [t_0, t'] \rightarrow \text{flow}_q(t, t').

where:
\[
\text{flow}_q(t, t') = \bigwedge_{j=1}^{n^q} \left( \sum_{i=1}^{n} c_{ij}^q (x_i(t') - x_i(t)) \leq c_j^q (t' - t) \right).
\]

**Remark:** \(\text{flow}_q(t_0, t_1)\) contains universal quantifiers.

**Locality results:** Sufficient to use the instances at \(t_0\) and \(t_1\)

**Flow** \(^{\text{Inst}}_q((t_0, t_1)) : \text{Inv}_q(\underline{x}(t_0))) \land \text{Inv}_q(\underline{x}(t_1))) \land \text{flow}_q(t_0, t_1)).
Example

**Invariant:**

\[ \phi_{\text{safe}}(x_1, x_2, x_3) : x_1 + x_2 + x_3 \leq L_{\text{overflow}} \wedge -\epsilon \leq x_1 - x_2 \leq \epsilon. \]

**Illustration:** \( F_{\text{flow}}(2) \) (invariance under the flow in reaction mode):

- \( \Psi(0) \)
  \[ (x_1(0) + x_2(0) + x_3(0) \leq L_{\text{overflow}} \wedge -\epsilon \leq x_1(0) - x_2(0) \leq \epsilon) \wedge \]

- \( \neg \Psi(t) \)
  \[ -(x_1(t) + x_2(t) + x_3(t) \leq L_{\text{overflow}} \wedge -\epsilon \leq x_1(t) - x_2(t) \leq \epsilon) \wedge \]

**Inv}_2(0) \)

\[ L_f \leq x_1(0) + x_2(0) + x_3(0) \leq L_{\text{overflow}} \wedge x_3(0) \leq \max \wedge \]

**Inv}_2(t) \)

\[ L_f \leq x_1(t) + x_2(t) + x_3(t) \leq L_{\text{overflow}} \wedge x_3(t) \leq \max \wedge \]

**flow}_2 \)

\[ x_1(t) - x_1(0) \leq -d_{\text{max}} \cdot t \wedge x_2(t) - x_2(0) \leq -d_{\text{max}} \cdot t \wedge \]

\[ x_3(t) - x_3(0) \geq d_{\text{min}} \cdot t \wedge (x_1(t) - x_1(0)) - (x_2(t) - x_2(0)) = 0 \wedge \]

\[ (x_1(t) - x_1(0)) + (x_2(t) - x_2(0)) + (x_3(t) - x_3(0)) = 0 \]

For fixed values for \( L_f, L_{\text{overflow}} \) — satisfiability check: \( \text{PTIME} \).

**Parametric version:** check satisfiability if \( L_f < L_{\text{overflow}} \wedge \epsilon_a < \epsilon \)

or generate constraints on the parameters which guarantee (un)satisfiability
Further extensions (Systems of LHA)

[Damm, Horbach, VS: FroCoS’15] Modularity results and small model property results for (decoupled) families of linear hybrid automata

Examples:

Sensors + Communication Channels

Safety properties: \( \forall i_1, \ldots, i_k \; \phi_{\text{safe}}(i_1, \ldots, i_l) \)

Collision free: \( \forall i, j (\text{lane}(i) = \text{lane}(j) \land \text{pos}(i) \geq \text{pos}(j) \land i \neq j \rightarrow \text{pos}(i) - \text{pos}(j) > d) \)
Model: Families of similar interacting system

Model families \( \{ S(i) \mid i \in I \} \) consisting of an unbounded number of similar interacting systems.

- Model the interaction
- Model the systems \( S(i) \)
- Model the topology updates
Model: Families of similar interacting systems

Model families \( \{ S(i) \mid i \in I \} \) consisting of an unbounded number of similar interacting systems.

- **Model the interaction**
  \[ \mapsto \text{structures } (I, \{ p : I \rightarrow I \}_{p \in P}) \]
  \[ P = P_S \cup P_N \]

The functions in \( P \) model the way the systems perceive their neighbors

\( P_S \): sensors:

- \( \text{sideback}(7) = 3 \)
- \( \text{back}(7) = 3 \)
- \( \text{front}(7) = \text{nil} \)
- \( \text{sidefront}(7) = 10 \)
Model: Families of similar interacting systems

Model families \( \{ S(i) \mid i \in I \} \) consisting of an unbounded number of similar interacting systems.

- Model the interaction \( \mapsto \) structures \( (I, \{ p : I \to I \}_{p \in P}) \)
- Model the systems \( S(i) \) \( \mapsto \) hybrid automata
Model: Spatial families of LHA

Model families \( \{S(i) \mid i \in I\} \) consisting of an unbounded number of similar interacting systems.

- Model the interaction \( \leftrightarrow \) structures \( (I, \{p : I \rightarrow I\}_{p \in P}) \)
- Model the systems \( S(i) \) \( \leftrightarrow \) hybrid automata
- **Model the topology updates** \( \leftrightarrow \) Topology automaton

**Example:**

Update(front, front')

\[
\forall i (i \neq \text{nil} \land \text{Prop}(i) \land \neg \exists j(\text{ASL}(j, i)) \rightarrow \text{front}'(i) = \text{nil})
\]

\[
\forall i (i \neq \text{nil} \land \text{Prop}(i) \land \exists j(\text{ASL}(j, i)) \rightarrow \text{Closest}_f(\text{front}'(i), i))
\]

\[
\forall i (i \neq \text{nil} \land \neg \text{Prop}(i) \rightarrow \text{front}'(i) = \text{front}(i))
\]

ASL\((j, i)\): \( j \neq \text{nil} \land \text{lane}(j) = \text{lane}(i) \land \text{pos}(j) > \text{pos}(i) \) \( j \) is ahead of \( i \) on the same lane

Closest\(_f\((j, i)\): \( \text{ASL}(j, i) \land \forall k(\text{ASL}(k, i) \rightarrow \text{pos}(k) \geq \text{pos}(j)) \) \( j \) is ahead of \( i \); no car between them.
Verification

Is safety property an inductive invariant?
Verification

Is safety property an inductive invariant?

Local extensions: use \texttt{H-PILoT}

• Unsatisfiable $\mapsto$ Safety invariant
• Satisfiable $\mapsto$ Model
Verification

Is safety property an inductive invariant?

**Local extensions:** use **H-PILoT**
- Unsatisfiable $\mapsto$ Safety invariant
- Satisfiable $\mapsto$ Model $\mapsto$ Simulation [J. Wild, BSc Thesis 2018]
Other approaches

**First-Order Dynamic Logic**
Dynamic logic in which the atomic programs contain variables
The KeY System (Bernhard Beckert et al.)

**Hybrid Dynamic Logic**
Dynamic logic in which the atomic programs contain differential equations
The KeYmaera Verification Tool (Andre Platzer)
(Differential dynamic logic)
Summary

• Basic notions in formal specification and verification

Related topics

- Seminar “Decision Procedures and Applications”: Summer Semester

More details on Specification, Model Checking, Verification:

Every summer (usually end of August):
   Summer school “Verification Technology, Systems & Applications”

BSc/MSc Theses in the area