Parameterized verification of fault-tolerant protocols by infinite-state model checking

Francesco Alberti

Formal Verification and Security Group
University of Lugano, Switzerland

March 14, 2011 (AVM 2011)
Outline

1. Parameterized systems and Uniform verification
2. MCMT - Model Checker Modulo Theories
3. Case study - The problem of Reliable Broadcast
4. Future work
Parameterized systems and Uniform verification

- Parameterized system: System given schematically in terms of a parameter $n$: $S_n$
  - E.g. Protocols involving $n$ processes in their execution

The problem of Uniform Verification

Verify if a given property $\phi$ holds in the parameterized system $S_n$ for any value of $n \geq 2$

Problems:
- In theory - This problem is, in general, undecidable [Apt and Kozen, 1986]
  - We will focus on a restricted family of parameterized systems for which it becomes decidable and only on safety properties
- In practice - $S_n$ is intrinsically an infinite-state system
  - How can we handle possibly infinite set of states?
Parameterized systems and Uniform verification

- Parameterized system: System given schematically in terms of a parameter $n$: $S_n$
  
- E.g. Protocols involving $n$ processes in their execution

**The problem of Uniform Verification**

Verify if a given property $\phi$ holds in the parameterized system $S_n$ for any value of $n \geq 2$
Parameterized systems and Uniform verification

- **Parameterized system**: System given schematically in terms of a parameter $n$: $S_n$
  - E.g. Protocols involving $n$ processes in their execution

### The problem of Uniform Verification

Verify if a given property $\phi$ holds in the parameterized system $S_n$ for any value of $n \geq 2$

**Problems:**

- **In theory** - This problem is, in general, undecidable
  [Apt and Kozen, 1986]
  - We will focus on a restricted family of parameterized systems for which it becomes decidable and only on safety properties
Parameterized systems and Uniform verification

- Parameterized system: System given schematically in terms of a parameter $n$: $S_n$
  - E.g. Protocols involving $n$ processes in their execution

The problem of Uniform Verification

Verify if a given property $\phi$ holds in the parameterized system $S_n$ for any value of $n \geq 2$

Problems:

- **In theory** - This problem is, in general, undecidable [Apt and Kozen, 1986]
  - We will focus on a restricted family of parameterized systems for which it becomes decidable and only on safety properties

- **In practice** - $S_n$ is intrinsically an infinite-state system
  - How can we handle possibly infinite set of states?
Let’s consider a very simple protocol involving $n$ processes in its execution.

Every process is an instance of the same state-machine.

A process is in the $R$ location if it is in the critical section.
One possible *configuration* of our parameterized system can have 4 processes, and one of them is in the critical section.
Another configuration: 8 processes none in the critical section
Encoding configurations

Problem

How to define a compact representation of states?
Encoding configurations

Problem
How to define a compact representation of states?

Solution
- Symbolic approach: formulae are used to represent set of states
- Topology and data are described in a declarative way using two theories $T_I$ and $T_E$
- ARRAY state variables
We fix a theory $T_I = (\Sigma_I, C_I)$ for the topology

- $T_I$ has one sort symbol, INDEX
- $C_I$ consists of all (finite) sets, linear orders, forests/trees, graphs, ...
We fix a theory $T_I = (\Sigma_I, C_I)$ for the topology
- $T_I$ has one sort symbol, INDEX
- $C_I$ consists of all (finite) sets, linear orders, forests/trees, graphs, ...

We fix a theory $T_E = (\Sigma_E, C_E)$ for the data
- $T_E$ has one sort symbol, ELEM
- Usually $C_E$ contains just one structure: integers, reals, Booleans, ...
Encoding configurations
Towards Array-based systems [Ghilardi et al., 2008]

- We fix a theory $T_I = (\Sigma_I, C_I)$ for the topology
  - $T_I$ has one sort symbol, INDEX
  - $C_I$ consists of all (finite) sets, linear orders, forests/trees, graphs, ...

- We fix a theory $T_E = (\Sigma_E, C_E)$ for the data
  - $T_E$ has one sort symbol, ELEM
  - Usually $C_E$ contains just one structure: integers, reals, Booleans, ...

- A third theory $A^E_I = (\Sigma, C)$ is obtained from $T_I$ and $T_E$
We fix a theory $T_I = (\Sigma_I, C_I)$ for the topology
- $T_I$ has one sort symbol, INDEX
- $C_I$ consists of all (finite) sets, linear orders, forests/trees, graphs, ...

We fix a theory $T_E = (\Sigma_E, C_E)$ for the data
- $T_E$ has one sort symbol, ELEM
- Usually $C_E$ contains just one structure: integers, reals, Booleans, ...

A third theory $A^E_I = (\Sigma, C)$ is obtained from $T_I$ and $T_E$
- $A^E_I$ has three sort symbols: INDEX, ELEM and ARRAY
- $\Sigma$ contains all the symbols in the disjoint union $\Sigma_I \cup \Sigma_E \cup \text{[-]}$
  - $\text{[-]} : \text{ARRAY} \times \text{INDEX} \rightarrow \text{ELEM}$
encoding configurations

Towards Array-based systems [Ghilardi et al., 2008]

- We fix a theory \( T_I = (\Sigma_I, C_I) \) for the topology
  - \( T_I \) has one sort symbol, INDEX
  - \( C_I \) consists of all (finite) sets, linear orders, forests/trees, graphs, ...

- We fix a theory \( T_E = (\Sigma_E, C_E) \) for the data
  - \( T_E \) has one sort symbol, ELEM
  - Usually \( C_E \) contains just one structure: integers, reals, Booleans, ...

- A third theory \( A^E_I = (\Sigma, C) \) is obtained from \( T_I \) and \( T_E \)
  - \( A^E_I \) has three sort symbols: INDEX, ELEM and ARRAY
  - \( \Sigma \) contains all the symbols in the disjoint union \( \Sigma_I \cup \Sigma_E \cup [-] \)
    - \([-] : \text{ARRAY} \times \text{INDEX} \rightarrow \text{ELEM} \)
  - \( \mathcal{M} \in C \) if symbols of sort ARRAY are interpreted as (total) functions from \( \text{INDEX}^{\mathcal{M}} \) to \( \text{ELEM}^{\mathcal{M}} \)
An **array-based system** on $A^E_1$ with ARRAY state variable $a$ is the following pair of formulae:

$$S_n = \langle I(a); \tau(a, a') \rangle$$
An **array-based system** on $A^E_i$ with ARRAY state variable $a$ is the following pair of formulae:

$$S_n = \langle I(a); \tau(a, a') \rangle$$

A state of an array-based system is an assignment to the variable $a$ in a model of $A^E_i$.
An array-based system on $A_i^E$ with ARRAY state variable $a$ is the following pair of formulae:

$$S_n = \langle I(a); \tau(a, a') \rangle$$

A state of an array-based system is an assignment to the variable $a$ in a model of $A_i^E$
The initial set of states will be represented by a $\forall^I$-formula

- E.g. all the processes are in the blue location: $\forall x. (L[x] = B)$
- $\forall^I$-formulae can also be used to express invariants
The initial set of states will be represented by a $\forall I$-formula
- E.g. all the processes are in the blue location: $\forall x. (L[x] = B)$
- $\forall I$-formulae can also be used to express invariants

The unsafe set of states will be represented by an $\exists I$-formula
- E.g. violation of the mutual exclusion:
  $\exists x, y. (x \neq y \land L[x] = R \land L[y] = R)$
The initial set of states will be represented by a $\forall^I$-formula
- E.g. all the processes are in the blue location: $\forall x. (L[x] = B)$
- $\forall^I$-formulae can also be used to express invariants

The unsafe set of states will be represented by an $\exists^I$-formula
- E.g. violation of the mutual exclusion:
  $\exists x, y. (x \neq y \land L[x] = R \land L[y] = R)$

Transition relation is represented by a disjunction of formulae of kind $\exists^I$.
  $\exists i. (\phi(i, a[i]) \land a' = \lambda j. F(i, a[i], j, a[j]) )$
- $F$ is a case-defined function
Encoding configurations

Array-based systems

\[ \exists x, y. \ (x \neq y \land L[x] = R \land L[y] = B) \]

\[ \exists x, y. \ (x \neq y \land L'[x] = R \land L'[y] = R) \]

\[ \tau = \exists x. \ (L[x] = B \land L' = \lambda j. (\text{if } x = j \text{ then } R \text{ else } L[j])) \]
Encoding configurations
Array-based systems

\[ \exists x, y. (x \neq y \land L[x] = R \land L[y] = B) \]

\[ \exists x, y. (x \neq y \land L'[x] = R \land L'[y] = R) \]

\[ \tau = \exists x. (L[x] = B \land L' = \lambda j. (\text{if } x = j \text{ then } R \text{ else } L[j])) \]

ARRAY variable \( L \)
Encoding configurations
Array-based systems

\[ \exists x, y. (x \neq y \land L[x] = R \land L[y] = B) \]

\[ \exists x, y. (x \neq y \land L'[x] = R \land L'[y] = R) \]
Encoding configurations
Array-based systems

\[ \exists x, y. (x \neq y \land L[x] = R \land L[y] = B) \]

\[ \exists x, y. (x \neq y \land L'[x] = R \land L'[y] = R) \]

\[ \begin{array}{cccc}
L & \cdots & R & \cdots & B & \cdots \\
\end{array} \]

\[ \tau \]

\[ \begin{array}{cccc}
L' & \cdots & R & \cdots & R & \cdots \\
\end{array} \]
Encoding configurations
Array-based systems

ARRAY variable $L$

$\exists x, y. (x \neq y \land L[x] = R \land L[y] = B)$

$\exists x, y. (x \neq y \land L'[x] = R \land L'[y] = R)$

$L \cdots R \cdots B \cdots \xrightarrow{\tau} L' \cdots R \cdots R \cdots$

$\tau = \exists x. \left( L[x] = B \land L' = \lambda j. (\text{if } (x = j) \text{ then } R \text{ else } L[j]) \right)$
Model checking

The safety problem [Ghilardi et al., 2008]

Ingredients:

- Theories $T_I, T_E$
- Array-based system $S = \langle l(a); \tau(a, a') \rangle$ on $A^E_I$
- $\exists^I$-formula $U(a)$ describing a set of unsafe states

Safety problem - Backward reachability algorithm

Check the (un)reachability of an unsafe state $U(a)$ by executing a (fully symbolic) backward reachability procedure $Br(a)$:

$$Br(a) := \text{Pre}(\tau, U(a)) = \exists a'. (\tau(a, a') \land U(a'))$$

Can be proved that $Br(a)$ is $\exists^I$-equivalent to an effectively computable $\exists^I$-formula $F$. Alberti (USI)
Model checking

The safety problem [Ghilardi et al., 2008]

Ingredients:

- Theories $T_I, T_E$
- Array-based system $S = \langle I(a); \tau(a, a') \rangle$ on $A^E_I$
- $\exists^I$-formula $U(a)$ describing a set of unsafe states

Safety problem - Backward reachability algorithm

Check the (un)reachability of an unsafe state $U$ by executing a (fully symbolic) backward reachability procedure

$$Br(a) := \text{Pre}(\tau, U) = \exists a'. (\tau(a, a') \land U(a'))$$
Model checking

The safety problem [Ghilardi et al., 2008]

Ingredients:
- Theories $T_I, T_E$
- Array-based system $S = \langle I(a); \tau(a, a') \rangle$ on $A^E_I$
- $\exists^I$-formula $U(a)$ describing a set of unsafe states

Safety problem - Backward reachability algorithm

Check the (un)reachability of an unsafe state $U$ by executing a (fully symbolic) backward reachability procedure

$$Br(a) := \text{Pre}(\tau, U) = \exists a'. (\tau(a, a') \land U(a'))$$

Can be proved that $Br(a)$ is $A^E_I$-equivalent to an effectively computable $\exists^I$-formula
We iteratively compute the preimage of $U$ applying backward $\tau$
We iteratively compute the preimage of $U$ applying backward $\tau$.
We iteratively compute the preimage of $U$ applying backward $\tau$.
We iteratively compute the preimage of $U$ applying backward $\tau$.
We iteratively compute the preimage of $U$ applying backward $\tau$

... until we find an intersection with the set of initial states...
We iteratively compute the preimage of $U$ applying backward $\tau$

... until we find an intersection with the set of initial states...

... or a (global) fix-point.
Reduce intersection and fix-point test to SMT problems:

- Intersection test: is $I \land U_n \ A^E_I$-satisfiable?
Reduce intersection and fix-point test to SMT problems:

- Intersection test: is $I \land U_n \ A_I^E$-satisfiable?
- Fix-point test: is $U_{n+1} \rightarrow U_n \ A_I^E$-valid?
- ...or dually: is $U_{n+1} \land \neg U_n \ A_I^E$-unsatisfiable?
Architecture (v1) [Ghilardi and Ranise, 2010]

- **client** - Computes the preimages and generates the instances of *safety* and *fix-point* checks (handling of quantifiers)
- **server** - SMT-solver: decides the (un)satisfiability of the formulae $\phi$ encoding safety and fix-point checks

\[ I(a), U(a), \tau(a, a'), T_I, T_E \]
1. \( SMT(T_I) \) and \( SMT(T_E) \) problems for quantifier-free formulae are decidable

2. \( \Sigma_I \) and \( \Sigma_E \) contain only constants and predicates

3. The class of models of \( T_I \) is closed under substructures

4. The preorder \( \preceq \) on \( A^E_I \)-configurations is a well-quasi order \(^1\)

\(^1\) A reflexive, transitive binary relation that neither contains infinite strictly decreasing sequences nor infinite sequences of pairwise incomparable elements.
1. $SMT(T_I)$ and $SMT(T_E)$ problems for quantifier-free formulae are decidable

2. $\Sigma_I$ and $\Sigma_E$ contain only constants and predicates

3. The class of models of $T_I$ is closed under substructures

4. The preorder $\preceq$ on $A_I^E$-configurations is a well-quasi order \(^1\)

$\Rightarrow$ backward reachability always terminates.
Parameterized verification

Why MCMT?

- Fully symbolic (formulae represent set of states)
Parameterized verification

Why MCMT?

- Fully symbolic (formulae represent set of states)
- Declarative specification of topology and data with first order theories
Parameterized verification

Why MCMT?

- Fully symbolic (formulae represent set of states)
- Declarative specification of topology and data with first order theories
- Counterexample (if any)
Parameterized verification

Why \textsc{mcmt}?

- Fully symbolic (formulae represent set of states)
- Declarative specification of topology and data with first order theories
- Counterexample (if any)
- High degree of automation
  - As much as possible automatic verification
  - Avoid the introduction of bugs from user interaction by accepting “candidate invariants”
An inductive invariant is a $\forall^I$-formula $\phi(a)$ s.t.

- $A^E_i \models l(a) \rightarrow \phi(a)$
- $A^E_i \models \phi(a) \land \tau(a, a') \rightarrow \phi(a')$
- $\phi(a)$ is $A^E_i$-inconsistent with the formula $U(a)$ describing unreachable states
An inductive invariant is a $\forall^l$-formula $\phi(a)$ s.t.

- $A_i^E \models I(a) \rightarrow \phi(a)$
- $A_i^E \models \phi(a) \land \tau(a, a') \rightarrow \phi(a')$
- $\phi(a)$ is $A_i^E$-inconsistent with the formula $U(a)$ describing unreachable states

- Declarative approach: if $S_n$ is safe w.r.t. $\phi$, then $\neg \phi$ is a safe invariant for the system.
- Make a “plan of work”!
- We can tell to MCMT:
  1. Try to check these invariants: $\phi_1, \phi_2, \phi_3, ...$
  2. Use **only** those you have found to be *real* safe invariants in the main verification process.
Case study: The problem of Reliable Broadcast
[Hadzilacos and Toueg, 1993]

- A process \( p \) of a distributed system wants to send a message \( m \) to all other processes
- Broadcast primitives not available
  \( \Rightarrow \) \( p \) must send \( m \) to each process \textit{separately}
Case study: The problem of Reliable Broadcast

[Hadzilacos and Toueg, 1993]

- A process $p$ of a distributed system wants to send a message $m$ to all other processes.
- Broadcast primitives not available.

$\Rightarrow$ $p$ must send $m$ to each process separately.

Sender/receiver’s failures may cause inconsistencies!
Case study: The problem of Reliable Broadcast
[Hadzilacos and Toueg, 1993]
- A process \( p \) of a distributed system wants to send a message \( m \) to all other processes
- Broadcast primitives not available
  \( \Rightarrow \) \( p \) must send \( m \) to each process separately

Sender/receiver’s failures may cause inconsistencies!

Failures of processes are described by means of failure models
[Tanenbaum and Steen, 2006]
- E.g. Stopping-failure, Omission, Timing, Response, ...
Parameterized verification of Fault-tolerant protocols

The problem of Reliable Broadcast - A solution [Hadzilacos and Toueg, 1993]
Parameterized verification of Fault-tolerant protocols

The problem of Reliable Broadcast - A solution [Hadzilacos and Toueg, 1993]
Safety property: *agreement*

If a correct process delivers $m$, all correct processes deliver $m$. 
The description of protocols in [Chandra and Toueg, 1990] requires several array variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Domain</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>round</td>
<td>[1; 7]</td>
<td>The location of each process</td>
</tr>
<tr>
<td>message</td>
<td>Boolean</td>
<td>The local message of the process</td>
</tr>
<tr>
<td>delivered</td>
<td>Boolean</td>
<td>The process has delivered the message</td>
</tr>
<tr>
<td>coord</td>
<td>Boolean</td>
<td>The process is the coordinator of the network</td>
</tr>
<tr>
<td>done</td>
<td>Boolean</td>
<td>The process has done the round operations</td>
</tr>
<tr>
<td>id</td>
<td>$\mathbb{Z}$</td>
<td>The id of the sender of the message</td>
</tr>
<tr>
<td>faulty</td>
<td>Boolean</td>
<td>The process is faulty</td>
</tr>
</tbody>
</table>

...
Encoding systems
An example from our case study - Unsafe configuration

Safety property: agreement
If a correct process delivers a message $m$, then all correct processes deliver the same message $m$.

$$U := \exists x, y. \left( x \neq y \land \right.$$
$$\left. \begin{array}{l}
delivered[x] = \top \land faulty[x] = \bot \land \\
delivered[y] = \top \land faulty[y] = \bot \land \\
message[x] \neq message[y]
\end{array} \right)$$
Our case study: Reliable Broadcast

Results [Alberti et al., 2010]

First formal parametrized verification of protocols from [Chandra and Toueg, 1990] (to the best of our knowledge)

<table>
<thead>
<tr>
<th></th>
<th>Crash, pr. 1</th>
<th>S-O, pr.1</th>
<th>S-O, pr.1 (e)</th>
<th>S-O, pr.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>SAFE</td>
<td>UNSAFE</td>
<td>UNSAFE</td>
<td>SAFE</td>
</tr>
<tr>
<td># State variables</td>
<td>8</td>
<td>95</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td># Transitions</td>
<td>13</td>
<td>16</td>
<td>22</td>
<td>28</td>
</tr>
<tr>
<td># Nodes</td>
<td>113 (-21)</td>
<td>464 (-26)</td>
<td>9.679 (-770)</td>
<td>11.158 (-1.290)</td>
</tr>
<tr>
<td># SMT calls</td>
<td>2.792</td>
<td>20.009</td>
<td>1.338.058</td>
<td>2.558.986</td>
</tr>
<tr>
<td>Length unsafe trace</td>
<td>×</td>
<td>11 tr.</td>
<td>33 tr.</td>
<td>×</td>
</tr>
<tr>
<td># Invariants</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>19 (+7)</td>
</tr>
<tr>
<td>Max # processes</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Time</td>
<td>1 s</td>
<td>18 s</td>
<td>28 mins</td>
<td>78 mins</td>
</tr>
</tbody>
</table>

Intel i7 @ 2.66 GHz, 4 GB RAM, Mac OSX 10.6
### First formal parametrized verification of protocols from [Chandra and Toueg, 1990] (to the best of our knowledge)

<table>
<thead>
<tr>
<th></th>
<th>Crash, pr. 1</th>
<th>S-O, pr.1</th>
<th>S-O, pr.1 (e)</th>
<th>S-O, pr.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>SAFE</td>
<td>UNSAFE</td>
<td>UNSAFE</td>
<td>SAFE</td>
</tr>
<tr>
<td># State variables</td>
<td>8</td>
<td>95</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td># Transitions</td>
<td>13</td>
<td>16</td>
<td>22</td>
<td>28</td>
</tr>
<tr>
<td># Nodes</td>
<td>113 (-21)</td>
<td>464 (-26)</td>
<td>9.679 (-770)</td>
<td>11.158 (-1.290)</td>
</tr>
<tr>
<td># SMT calls</td>
<td>2.792</td>
<td>20.009</td>
<td>1.338.058</td>
<td>2.558.986</td>
</tr>
<tr>
<td>Length unsafe trace</td>
<td>×</td>
<td>11 tr.</td>
<td>33 tr.</td>
<td>×</td>
</tr>
<tr>
<td># Invariants</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>19 (+7)</td>
</tr>
<tr>
<td>Max # processes</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Time</td>
<td>1 s</td>
<td>8 s</td>
<td>28 mins</td>
<td>78 mins</td>
</tr>
</tbody>
</table>

Shorter than [Chandra and Toueg, 1990]
Our case study: Reliable Broadcast

Results [Alberti et al., 2010]

First formal parametrized verification of protocols from [Chandra and Toueg, 1990] (to the best of our knowledge)

<table>
<thead>
<tr>
<th></th>
<th>Crash, pr. 1</th>
<th>S-O, pr.1</th>
<th>S-O, pr.1 (e)</th>
<th>S-O, pr.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>SAFE</td>
<td>UNSAFE</td>
<td>UNSAFE</td>
<td>SAFE</td>
</tr>
<tr>
<td># State variables</td>
<td>8</td>
<td>95</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td># Transitions</td>
<td>13</td>
<td>16</td>
<td>22</td>
<td>28</td>
</tr>
<tr>
<td># Nodes</td>
<td>113 (-21)</td>
<td>464 (-26)</td>
<td>9.679 (-770)</td>
<td>11.158 (-1.290)</td>
</tr>
<tr>
<td># SMT calls</td>
<td>2.792</td>
<td>20.009</td>
<td>1.338.058</td>
<td>2.558.986</td>
</tr>
<tr>
<td>Length unsafe trace</td>
<td>×</td>
<td>11 tr.</td>
<td>33 tr.</td>
<td>×</td>
</tr>
<tr>
<td># Invariants</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>19 (+7)</td>
</tr>
<tr>
<td>Max # processes</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Time</td>
<td>1 s</td>
<td>18 s</td>
<td>28 mins</td>
<td>78 mins</td>
</tr>
</tbody>
</table>

Intel i7 © 2.66 GHz, 4 GB RAM, Mac OSX 10.6
MCMT v2

- New data structures to better handling formulae
- Full integration with OpenSMT
- More flexible input language
Conclusion and Future work

MCMT v2

- New data structures to better handling formulae
- Full integration with OpenSMT
- More flexible input language

Formal verification:

- Apply MCMT to imperative programs verification
  - Assumptions required for termination are not satisfied
Conclusion and Future work

jww Roberto Bruttomesso, Silvio Ghilardi, Silvio Ranise, Natasha Sharygina

- **MCMT v2**
  - New data structures to better handling formulae
  - Full integration with OpenSMT
  - More flexible input language

- **Formal verification:**
  - Apply MCMT to imperative programs verification
    - Assumptions required for termination are not satisfied
  - Invariant-search procedure
Conclusion and Future work

MCMT v2
- New data structures to better handling formulae
- Full integration with OpenSMT
- More flexible input language

Formal verification:
- Apply MCMT to imperative programs verification
  - Assumptions required for termination are not satisfied
- Invariant-search procedure
- Abstraction/refinement techniques
Thank you!
Questions?

Francesco Alberti
francesco.alberti@usi.ch


