Parametrised Compositional Verification with Multiple Process and Data Types

Antti Siirtola and Keijo Heljanko

Department of Information and Computer Science
Aalto University, School of Science
firstname.lastname@aalto.fi

ACSD, 8th July, 2013
Outline

Introduction

Process Model

Parametrised Processes

Results

Conclusions
Introduction
Software Systems

Software systems have many natural parameters:

- the number of concurrent threads,
- the number of replicated objects,
- the size of parametric data types,
- the size of stack = the number of recursions.
Software systems have many natural parameters:

- the number of concurrent threads,
- the number of replicated objects,
- the size of parametric data types,
- the size of stack = the number of recursions.

Software systems are naturally represented as parametrised systems.
Main Question

Given

- $\text{Sys}(i)$ – a system implementation parametrised by $i$,
- $\text{Spec}(i)$ – a system specification parameterised by $i$,
- $I$ – an (infinite) set of parameter values,

determine whether

- $\text{Sys}(i) \preceq \text{Spec}(i)$ for all $i \in I$?
Compositional Verification

Software systems are made of components/subsystems. which may be available only in the interface specification form!
Compositional Verification

Software systems are made of components/subsystems, which may be available only in the interface specification form!

There is a need for compositional verification.

Assume that

\[ Sys = SubSys_1 \otimes SubSys_2, \]
\[ SubSys_i \preceq SubSpec_i \text{ for both } i = 1, 2, \]
\[ SubSpec_1 \otimes SubSpec_2 \preceq Spec. \]

Then it should hold that

\[ Sys = SubSys_1 \otimes SubSys_2 \preceq SubSpec_1 \otimes SubSpec_2 \preceq Spec. \]
Compositional Verification

Software systems are made of components/subsystems, which may be available only in the interface specification form!

\( \preceq \) should be precongruence!

reflexive, transitive, compositional, i.e., if \( P_1 \otimes Q_1 \) and \( P_2 \otimes Q_2 \), then \( P_1 \otimes P_2 \preceq Q_1 \otimes Q_2 \).
Parametrised Compositional Verification (PCV)

Behavioural Fixed Point (BFP) Technique (Valmari & Tienari 1991; Valmari & Kokkarinen 1998)

+ Systems of a linear topology (rings, chains, etc.)
- Other topologies, parametrised data
**Parametrised Compositional Verification (PCV)**

**Behavioural Fixed Point (BFP) Technique** (Valmari & Tienari 1991; Valmari & Kokkarinen 1998)

- Systems of a linear topology (rings, chains, etc.)
- Other topologies, parametrised data

**Data Independence (DI) results** (Lazić 1999)

- Systems with parametrised data types
- Parametrised processes
Parametrised Compositional Verification (PCV)

Behavioural Fixed Point (BFP) Technique (Valmari & Tienari 1991; Valmari & Kokkarinen 1998)

+ Systems of a linear topology (rings, chains, etc.)
  - Other topologies, parametrised data

Data Independence (DI) results (Lazić 1999)

+ Systems with parametrised data types
  - Parametrised processes

Precongruence Reduction (PR) method (Siirtola & Kortelainen 2009)

+ Systems with an arbitrary number of concurrent processes
  - Parametrised data, linear topologies
Combinations of PCV Techniques

BFP + DI (Creese 2001)

+ various topologies, parametrised data and processes
- only partly automatic
Combinations of PCV Techniques

BFP + DI (Creese 2001)

+ various topologies, parametrised data and processes
- only partly automatic

BFP + PR (Siirtola 2010)

+ various topologies, parametric processes
- not fully automatic
Combinations of PCV Techniques

BFP + DI (Creese 2001)

+ various topologies, parametrised data and processes
- only partly automatic

BFP + PR (Siirtola 2010)

+ various topologies, parametric processes
- not fully automatic

PR + DI (this paper)

+ parametrised data and processes
+ fully algorithmic
- linear topologies are not easily supported
Challenge

PR method

+ supports parametrised process types
- does not allow for components with a parametrised state space

DI results

+ supports parametrised data types
- does not allow the system topology to be parametrised
Processes

\[ \Sigma = \{ a, b \} \]
Parallel Composition

Hoare-style alphabet-based synchronisation

\[ \Sigma = \{a, c\} \]

\[ \Sigma = \{b, c\} \]
Parallel Composition

Hoare-style alphabet-based synchronisation

\[ \Sigma = \{a, c\} \]

\[ \Sigma = \{b, c\} \]

\[ \Sigma = \{a, b, c\} \]
$\Sigma = \{a, c\}$
Hiding

\[ \Sigma = \{a, c\} \]

\[ = \]

\[ \Sigma = \{c\} \]
Parametrised Processes
Parameters

Types

- process types – an arbitrarily large set of process identifiers
- data types – an arbitrarily large set of data values
Parameters

Types

- process types – an arbitrarily large set of process identifiers
- data types – an arbitrarily large set of data values

Typed Variables

- process variables – a process identifier
- data variables – a data value
Parametrised Processes

Parametrised action: $c(x_1, \ldots, x_k)$

- $c$ is a channel
- $x_1, \ldots, x_k$ are typed variables
Parametrised Processes

Parametrised action: $c(x_1, \ldots, x_k)$

- $c$ is a channel
- $x_1, \ldots, x_k$ are typed variables

Sequential Parametrised Process (SPP)

- A process with parametrised actions is a CPP
Parametrised Processes

Parametrised action: $c(x_1, \ldots, x_k)$

- $c$ is a channel
- $x_1, \ldots, x_k$ are typed variables

Sequential Parametrised Process (SPP)

- A process with parametrised actions is a CPP

Concurrent Parametrised Process (CPP)

- An SPP is a CPP
- There are also other CPPs
Parametrised Parallel Composition

Syntax: $\widehat{\|igcirc} \; x : \mathcal{P}$

- $x$ is a variable of a process type $P$
- $\mathcal{P}$ is a CPP
Parametrised Parallel Composition

Syntax: $\hat{\parallel}_x : \mathcal{P}$

- $x$ is a variable of a process type $P$
- $\mathcal{P}$ is a CPP

Instance of $\hat{\parallel}_x : \mathcal{P}$ when

$P = \{p_1, p_2, p_3\}$ and $\mathcal{P} =$

```
\begin{align*}
\xrightarrow{a(p_1)} & \xrightarrow{a(p_2)} & \xrightarrow{a(p_3)} \\
\xrightarrow{a(p_2)} & \xrightarrow{a(p_3)} & \\
\xrightarrow{a(p_1)} & \xrightarrow{a(p_2)} & \xrightarrow{a(p_3)} \\
\end{align*}
```
Parametrised Choice

Applied to transitions within SPPs
Parametrised Choice

Applied to transitions within SPPs

Example

Parametrised LTS: \( s_0 \xrightarrow{\Box y: a(y)} s_1(y) \xrightarrow{b(y)} s_2 \)

\( y \) is a variable of a data type \( D \)
Parametrised Choice

Applied to transitions within SPPs

Example
Parametrised LTS: $s_0 \xrightarrow{\square y : a(y)} s_1(y) \xrightarrow{b(y)} s_2$

$y$ is a variable of a data type $D$

Instance when $D = \{d_1, d_2, d_3\}$:

$\xrightarrow{a(d_1)} s_0 \xrightarrow{a(d_2)} s_1(d_1) \xrightarrow{b(d_1)} s_2$

$\xrightarrow{a(d_2)} s_0 \xrightarrow{a(d_3)} s_1(d_2) \xrightarrow{b(d_2)} s_2$

$\xrightarrow{a(d_3)} s_0 \xrightarrow{b(d_1)} s_2$
Other Constructs

Binary parallel composition
Syntax: $\mathcal{P}_1 \parallel \mathcal{P}_2$
Other Constructs

Binary parallel composition
Syntax: $\mathcal{P}_1 \parallel \mathcal{P}_2$

Guards – (in)equality tests between variables

Example 1:

\[
\begin{array}{c}
\xrightarrow{s_0} \quad \square x, y : [\wedge x \equiv y] a(x) \\
\xrightarrow{s_1(y)} \quad b(y) \\
\xrightarrow{s_2} 
\end{array}
\]

Example 2:

\[
\begin{array}{c}
\xrightarrow{\parallel x \parallel y} [\wedge x \equiv y] \mathcal{P} 
\end{array}
\]
Other Constructs

Binary parallel composition
Syntax: $\mathcal{P}_1 \parallel \mathcal{P}_2$

 Guards – (in)equality tests between variables

Example 1: $s_0 \xrightarrow{x, y : [\hat{x} \equiv y]a(x)} s_1(y) \xrightarrow{b(y)} s_2$
Example 2: $\hat{\bigwedge}_x \hat{\bigwedge}_y [\hat{x} \equiv y] \mathcal{P}$

Channel-based hiding
Syntax: $\mathcal{P} \setminus E$, $E$ is a set of channel names
Results
Compositionality

Definition

$P_1$ is a trace refinement of $P_2$ if and only if every instance of $P_1$ is a trace refinement of the corresponding instance of $P_2$. 

Compositionality

Definition
\( \mathcal{P}_1 \) is a *trace refinement* of \( \mathcal{P}_2 \) if and only if every instance of \( \mathcal{P}_1 \) is a trace refinement of the corresponding instance of \( \mathcal{P}_2 \)

Theorem
*The trace refinement is a precongruence on the set of CPPs.*
Theorem

Let $P$ and $Q$ be CPPs such that $Q$ is deterministic and does not involve hiding.

In order to decide whether $P$ is a trace refinement of $Q$, it is sufficient to solve finitely many trace refinement tasks between the finite-state instances of CPPs up to the cut-offs determined by the structure of $P$ and $Q$. 
Trace Refinement on CPPs

Theorem
Let $\mathcal{P}$ and $\mathcal{Q}$ be CPPs such that $\mathcal{Q}$ is deterministic and does not involve hiding.

In order to decide whether $\mathcal{P}$ is a trace refinement of $\mathcal{Q}$, it is sufficient to solve finitely many trace refinement tasks between the finite-state instances of CPPs up to the cut-offs determined by the structure of $\mathcal{P}$ and $\mathcal{Q}$.

There is a similar theorem for determinism checking, too.
Cut-Offs for Process Types

Apply PR technique

▶ Can be generalised to CPPs since data types do not affect the structure of the system.
▶ The cut-off size for a process type $T$ is the maximum number of nested parametrised parallel compositions $\widehat{\parallel}_x$ where the type of $x$ is $T$. 
Cut-Offs for Process Types

Apply PR technique

- Can be generalised to CPPs since data types do not affect the structure of the system.
- The cut-off size for a process type $T$ is the maximum number of nested parametrised parametrised parallel compositions $\hat{\parallel}_x$ where the type of $x$ is $T$.

Example

\[
P := \hat{\parallel}_u \hat{\parallel}_v \xrightarrow{s_0} \lozenge x, y : [\hat{x} \hat{=} y] a(x) \xrightarrow{s_1(y)} b(y) \xrightarrow{s_2},
\]
where $u, v$ are of the type $T$ and $x, y$ are of the type $D$. The cut-off size for $T$ is 2.
Cut-Offs for Data Types

Apply DI technique

- Can be applied after bounding the process types. (Our proof technique is simpler than the original one.)
- The cut-off size for a data type $D$ is the sum of $n_S \cdot cd_D(S)$, where
  - $S$ ranges over SPPs,
  - $n_S$ is the maximum number of instances of $S$,
  - $cd_D(S)$ is the maximum number of the occurrences of variables of the type $D$ within a single transition of $S$. 
Cut-Offs for Data Types

Apply DI technique

► Can be applied after bounding the process types. (Our proof technique is simpler than the original one.)

► The cut-off size for a data type $D$ is the sum of $n_S \cdot cd_D(S)$, where
  
  ▶ $S$ ranges over SPPs,
  ▶ $n_S$ is the maximum number of instances of $S$,
  ▶ $cd_D(S)$ is the maximum number of the occurrences of variables of the type $D$ within a single transition of $S$.

Example

$\mathcal{P} := \hat{\hat{\prod}}_{u,v} s_0 \xrightarrow{\Box x, y : [\hat{\wedge}x \overset{=}{} y]a(x)} s_1(y) \xrightarrow{b(y)} s_2$, where $u, v$ are of the type $T$ and $x, y$ are of the type $D$. The cut-off size for $D$ is $2 \cdot 2 \cdot 2 = 8$. 
We have implemented the approach in our Bounds tool.

<table>
<thead>
<tr>
<th>System</th>
<th>Process types</th>
<th>Data types</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>number cut-offs</td>
<td>number cut-offs</td>
</tr>
<tr>
<td>Host config protocol</td>
<td>1 2</td>
<td>1 16</td>
</tr>
<tr>
<td>SRS mutex</td>
<td>2 2,1</td>
<td>1 12</td>
</tr>
<tr>
<td>SRS consistency</td>
<td>1 1</td>
<td>1 8</td>
</tr>
<tr>
<td>Cache mutex</td>
<td>2 2,1</td>
<td>1 17</td>
</tr>
<tr>
<td>Cache consistency</td>
<td>1 1</td>
<td>1 14</td>
</tr>
</tbody>
</table>

- In each case, the computation of cut-offs took a fraction of a second.
- The instances up to the cut-offs were checked using FDR2 within a second.
Conclusions
Conclusions

We have combined two parametrised verification methods into a single powerful technique, which

- is sound and complete,
- allows for compositional reasoning,
- lends support to multiple parameters of two different kinds,
- is implemented and fully automatic.
Most direct extensions are undecidable:

- specifications with hiding
- non-deterministic specifications
- mixed types
- more complex guards
- failure-based semantics

We hope to find practically sensible assumptions under which some of the extensions can be realised.