Interactive Verification of Concurrent Systems

Philipp Rümmer
ph_r@gmx.net

University of Karlsruhe
Institute for Logic, Complexity and Deduction Systems
D-76128 Karlsruhe, Germany
Overview

- Introduction to CSP:
Overview

- Introduction to CSP:
  - Concept, basic operators
Overview

• Introduction to CSP:
  • Concept, basic operators
• Introduction to JCSP:
Overview

- Introduction to CSP:
  - Concept, basic operators
- Introduction to JCSP:
  - Basic classes
Overview

- Introduction to CSP:
  - Concept, basic operators
- Introduction to JCSP:
  - Basic classes
- Approach for verification of JCSP systems:
• Introduction to CSP:
  • Concept, basic operators
• Introduction to JCSP:
  • Basic classes
• Approach for verification of JCSP systems:
  • Representation of systems
Overview

- Introduction to CSP:
  - Concept, basic operators
- Introduction to JCSP:
  - Basic classes
- Approach for verification of JCSP systems:
  - Representation of systems
  - Concept of a calculus
Communicating Sequential Processes

Communicating Sequential Processes

- Formalism to design/describe interacting systems
Communicating Sequential Processes

- Formalism to design/describe interacting systems
- Today widely used to model protocols or hardware
Communicating Sequential Processes

- Formalism to design/describe interacting systems
- Today widely used to model protocols or hardware
- Analysis (mostly) through model checking (e.g. the FDR model checker)
Introduction to CSP: Processes

- Processes pose central entity of CSP concept
Introduction to CSP: Processes

- Processes pose central entity of CSP concept
- A process is described uniquely by its potential communication with an environment
Introduction to CSP: Processes

- Processes pose central entity of CSP concept
- A process is described uniquely by its potential communication with an environment
- Communication is a sequence of atomic \textit{events} (e.g. Processes $\equiv$ Languages)
States of automata can be regarded as processes:
Automata as Processes

- States of automata can be regarded as processes:

\[ traces[Q] = \{ \epsilon \} \]
States of automata can be regarded as processes:

\[
\begin{align*}
\text{traces } [Q] &= \{ \epsilon \} \\
\text{traces } [P] &= \{ a, bca, \ldots \}
\end{align*}
\]
States of automata can be regarded as processes:

- $\text{traces } [Q] = \{ \epsilon \}$
- $\text{traces } [P] = \{ a, bca, \ldots \}$
- $\text{traces } [R] = \{ ca, cbca, \ldots \}$
Automata as Processes

- States of automata can be regarded as processes:

\[
\begin{align*}
\text{traces}[Q] &= \{\epsilon\} \\
\text{traces}[P] &= \{a, bca, \ldots\} \\
\text{traces}[R] &= \{ca, cbca, \ldots\}
\end{align*}
\]

- Generalised as \textit{labelled transition systems} (LTS), usually infinite
Processes are usually described through CSP terms
CSP Terms

- Processes are usually described through CSP terms
- Definition of automaton using terms:

```
P  a  b  c
\rightarrow Q  \rightarrow R
```

Interactive Verification of Concurrent Systems – p.6/43
CSP Terms

- Processes are usually described through CSP terms
- Definition of automaton using terms:

\[ Q = \text{stop} \]
CSP Terms

- Processes are usually described through CSP terms
- Definition of automaton using terms:

  $\begin{align*}
  Q &= \text{stop} \\
  R &= c \rightarrow P 
  \end{align*}$
CSP Terms

- Processes are usually described through CSP terms
- Definition of automaton using terms:

\[ Q = \text{stop} \]
\[ R = c \rightarrow P \]
\[ P = (a \rightarrow Q) \Box (b \rightarrow R) \]
Basic CSP Operators: Locked Process

- Term notation: stop
Basic CSP Operators: Locked Process

- Term notation: \( \text{stop} \)
- LTS appearance: \( \text{stop} \)
Basic CSP Operators: Prefixing

- Term notation:

\[ a \rightarrow P \]
Basic CSP Operators: Prefixing

- Term notation:
  \[ a \rightarrow P \]

- LTS appearance:
Basic CSP Operators: Choice

- Term notation:

\[ P \parallel Q \]
Basic CSP Operators: Choice

- Term notation: \( P \parallel Q \)
- LTS appearance:

```
  P
 a   b
 /   \
|    |
P_1  P_2  ...  Q_1  Q_2
```

Interactive Verification of Concurrent Systems – p.10/43
Basic CSP Operators: Choice

- Term notation: $P \diamond Q$
- LTS appearance:
Basic CSP Operators: Choice

• Term notation:

\[ P \parallel Q \]

• LTS appearance:

![LTS Diagram]

Interactive Verification of Concurrent Systems – p.12/43
Basic CSP Operators: Parallelism

- Term notation: \((X: \text{the } interface \text{ set})\)

\[ P \parallel X \parallel Q \]
Basic CSP Operators: Parallelism

- Term notation: \((X: \text{the } \textit{interface} \text{ set})\)

\[ P \parallel [X] Q \]

- LTS appearance (for \(X = \{b\}\)):

\[
\begin{array}{cccc}
P_1 & \overset{a}{\rightarrow} & P_2 & \ldots \\
Q_1 & \overset{b}{\rightarrow} & Q_2 & \overset{c}{\rightarrow}
\end{array}
\]
Basic CSP Operators: Parallelism

• Term notation: \((X: \text{the } \textit{interface} \text{ set})\)

\[ P \parallel \begin{array}{|c|c|}
\hline
X \\
\hline
\end{array} \parallel Q \]

• LTS appearance (for \(X = \{b\}\)):

\[ \begin{array}{c}
P_1 \\
\downarrow^a \\
P_2 \\
\downarrow^b \\
P \\
\downarrow^b \\
Q_1 \\
\downarrow^c \\
Q_2 \\
\end{array} \]

\[ P \parallel \begin{array}{|c|c|}
\hline
X \\
\hline
\end{array} \parallel Q \]
Basic CSP Operators: Parallelism

- Term notation: ($X$: the interface set)

\[ P \parallel [X] Q \]

- LTS appearance (for $X = \{b\}$):

\[ P \parallel [X] Q \]

\[ P \parallel [X] Q \]

\[ Q \parallel [X] Q \]

\[ P \parallel [X] Q \]
Basic CSP Operators: Parallelism

- Term notation: \((X: \text{the interface set})\)
  \[ P \parallel [X] Q \]
- LTS appearance (for \(X = \{b\}\)):

```
  P1                      P2                      Q1                      Q2
  \(P\)                   \(P\)                   \(Q\)                   \(Q\)
  \(a\) \(b\)             \(b\) \(c\)             \(b\) \(c\)             \(b\) \(c\)
  \(P1 [X] Q\)            \(P2 [X] Q1\)          \(P [X] Q2\)
  \(a\) \(b\)             \(a\) \(b\)             \(a\) \(c\)
  \(P [X] Q\)             \(P [X] Q\)            \(P [X] Q2\)
```
Basic CSP Operators: Parallelism

- Parallelism $\iff$ Product of LTSs
Basic CSP Operators: Parallelism

- Parallelism $\rightarrow$ Product of LTSs
- Full synchronisation $P \parallel [A] \parallel Q \rightarrow$ Intersection of languages
Basic CSP Operators: Parallelism

- Parallelism $\rightarrow\rightarrow$ Product of LTSs
- Full synchronisation $P \parallel [AI] Q \rightarrow\rightarrow$ Intersection of languages
- Shorter notation for interleaving:

\[
P \parallel Q \quad := \quad P \parallel [\emptyset] Q
\]
Basic CSP Operators: Messages

- Transmission of values as events
Basic CSP Operators: Messages

- Transmission of values as events
- Sending message $\nu$:

$$x \rightarrow P \quad = \quad !x \rightarrow P$$
Basic CSP Operators: Messages

- Transmission of values as events
- Sending message $\nu$:
  
  $$x \rightarrow P = !x \rightarrow P$$

- Reading of messages: Generalised Choice
  
  $$?x : A \rightarrow P(x) := (v_1 \rightarrow P(v_1)) \square (v_2 \rightarrow P(v_2)) \square \cdots$$

  (where $A = \{v_1, v_2, \ldots\}$)
Example

- A process computing successors:

\[ S = ?n \rightarrow !(n + 1) \rightarrow S \quad (n \in \mathbb{N}) \]
Example

- A process computing successors:

\[ S = \ ?n \rightarrow !(n + 1) \rightarrow S \quad (n \in \mathbb{N}) \]

- LTS view:

```
S

S_0 \ldots \quad n \quad n+1
```

Interactive Verification of Concurrent Systems – p.20/43
Example (2)

- Communication between processes:

\[ A = !41 \rightarrow ?n \rightarrow A(n) \]
Example (2)

- Communication between processes:

\[ A = !41 \rightarrow ?n \rightarrow A(n) \]
\[ B = A \left[ \left[ \mathbb{N} \right] \right] S \]
Example (2)

- Communication between processes:

\[ A = \text{!}41 \rightarrow ?n \rightarrow A(n) \]
\[ B = A \left\langle \mathbb{N} \right\rangle S \]
Example (3)

\[ A \downarrow 1 \quad S \downarrow n \]

\[ S = \ ?n \rightarrow !(n + 1) \rightarrow S \]

\[ A = !1 \rightarrow ?n \rightarrow A(n) \]
Example (3)

\[
\begin{align*}
A & \quad S \\
\downarrow \quad 41 & \quad \downarrow \quad 41 \\
\downarrow \quad ?n \rightarrow A(n) & \quad \downarrow \quad !42 \rightarrow S \\
\downarrow \quad ?n & \quad \downarrow \quad !42 \\
\end{align*}
\]

\[S = ?n \rightarrow !(n + 1) \rightarrow S\]
\[A = !41 \rightarrow ?n \rightarrow A(n)\]
Example (3)

\[ A \xrightarrow{41} S \]
\[ ?n \rightarrow A(n) \quad !42 \rightarrow S \]
\[ 42 \leftarrow \quad 42 \rightarrow \quad A(42) \quad S \]

\[ S = ?n \rightarrow !(n + 1) \rightarrow S \]
\[ A = !41 \rightarrow ?n \rightarrow A(n) \]
CSP for Java (JCSP)

- Implementation of CSP process model in Java by P. D. Austin and P. H. Welch
CSP for Java (JCSP)

- Implementation of CSP process model in Java by P. D. Austin and P. H. Welch
- Very similar to the Occam language
CSP example: $A \parallel [N] \parallel S$
JCSP Example

- CSP example: $A \parallel [N] \parallel S$

- Corresponding JCSP object diagram:
Processes represented by interface
Introduction to JCSP

- Processes represented by interface

- In JCSP processes have identities
CSP operator of parallelism (actually interleaving) captured by

```
<<interface>>
CSProcess
+run()
```

```
Parallel
+run()
+addProcess(p:CSProcess)
```
CSP operator of parallelism (actually interleaving) captured by

- Each process is executed in its own thread
Introduction to JCSP (3)

- Messages are sent through *channels*

```
<<interface>>
Channel
+read(): Object
+write(o:Object)
```

```
One2OneChannel
+read(): Object
+write(o:Object)
...```

No "unbound" events as in CSP
Introduction to JCSP (3)

- Messages are sent through *channels*

  ```java
  <<interface>>
  Channel
  +read(): Object
  +write(o:Object)
  ```

- No “unbound” events as in CSP

```java
One2OneChannel
+read(): Object
+write(o:Object)
```
import jcsp.lang.*;

public class S implements CSProcess {
    private final Channel c;
    public S(Channel c) {
        this.c = c;
    }

    public void run() {
        while (true) {
            final Integer i = (Integer)c.read();
            c.write(new Integer(i.intValue() + 1));
        }
    }
}
Building Systems from Components

- Interface of a component is a tuple of channels + a protocol
Building Systems from Components

- Interface of a component is a tuple of channels + a protocol
- Systems are assembled from simpler components
Building Systems from Components

- Interface of a component is a tuple of channels + a protocol
- Systems are assembled from simpler components
- Contrary to normal instances of classes, components are active
Building Systems from Components

Diagram:

1 → Δ → 0 → Δ → +

Interactive Verification of Concurrent Systems – p.32/43
A Proof System for JCSP

- Basic concept:
A Proof System for JCSP

- Basic concept:
  - Represent JCSP systems as CSP terms
A Proof System for JCSP

• Basic concept:
  • Represent JCSP systems as CSP terms
  • Symbolically execute CSP terms → LTS
A Proof System for JCSP

- Basic concept:
  - Represent JCSP systems as CSP terms
  - Symbolically execute CSP terms → LTS
  - Specify/Prove LTS properties using a temporal/modal logic
A Proof System for JCSP

- Basic concept:
  - Represent JCSP systems as CSP terms
  - Symbolically execute CSP terms $\rightarrow$ LTS
  - Specify/Prove LTS properties using a temporal/modal logic

<table>
<thead>
<tr>
<th>JavaCard(DL)</th>
<th>CSP model of JCSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSP calculus</td>
<td></td>
</tr>
<tr>
<td>Modal logic/calculus</td>
<td></td>
</tr>
</tbody>
</table>
A sequential Java program $\alpha$ is regarded as a CSP process $T(\alpha)$.
A sequential Java program $\alpha$ is regarded as a CSP process $T(\alpha)$

Communication of $T(\alpha)$ is caused (only) by JCSP primitives
A sequential Java program $\alpha$ is regarded as a CSP process $T(\alpha)$

Communication of $T(\alpha)$ is caused (only) by JCSP primitives

- $\iff$ no shared memory
A sequential Java program $\alpha$ is regarded as a CSP process $T(\alpha)$

Communication of $T(\alpha)$ is caused (only) by JCSP primitives
  - $\implies$ no shared memory

JCSP primitives are modelled using CSP operators
Modelling JCSP Parallelism

- Class Parallel is represented by interleaving:
  
  ```java
  Parallel par = new Parallel();
  par.addProcess(s1);
  par.addProcess(s2); ...
  par.run();
  ```
Modelling JCSP Parallelism

- Class Parallel is represented by interleaving:
  
  ```java
  Parallel par = new Parallel();
  par.addProcess(s1);
  par.addProcess(s2); ...
  par.run();
  ```

- Corresponding process term:

  $$T(\ldots \text{par.run();} \ldots) =$$
  
  $$T(\ldots \text{s1.run();} \ldots) \parallel T(\ldots \text{s2.run();} \ldots) \parallel \cdots$$
Modelling JCSP Channels

- Each JCSP channel object is represented by a routing process.
Modelling JCSP Channels

- Each JCSP channel object is represented by a routing process
- Channel constructors add routers:

\[
T(\ldots \text{new One2OneChannel(); } \ldots) = \\
\text{O2ORouter } [\text{O2OEvents}] \parallel T(\ldots) 
\]
Modelling JCSP Channels

- Each JCSP channel object is represented by a routing process
- Channel constructors add routers:

\[ T(\ldots \text{new One2OneChannel()}; \ldots) = \]
\[ \text{O2ORouter} \parallel [\text{O2OEvents}] \parallel T(\ldots) \]

- Execution of Channel.read(), Channel.write() raises events
Example

System after execution of

```java
Channel c = new One2OneChannel();
Parallel par = new Parallel();
par.addProcess(new P1(c));
par.addProcess(new P2(c));
par.run();
```
Example

System after execution of

```java
Channel c = new One2OneChannel();
Parallel par = new Parallel();
par.addProcess(new P1(c));
par.addProcess(new P2(c));
par.run();
```

is described by

```
O2ORouter || O2OEvents \( \langle T(p1.run()); || T(p2.run()); \rangle \)
```
Message Transmission through $c$

O2ORouter $[\text{O2OEvents}]$

$(T(\ldots c.write(o); \ldots) \parallel T(\ldots a=c.read(); \ldots))$
Message Transmission through $c$

O2ORouter $[\text{O2OEvents}]$

$$ (T(\ldots c.\text{write}(o); \ldots) \parallel T(\ldots a=c.\text{read}(); \ldots)) $$

$\rightsquigarrow$ $\ldots [\ldots] (\lnot o \rightarrow T(\ldots\ldots) \parallel \text{?a} \rightarrow T(\ldots\ldots))$
Message Transmission through \( c \)

\[
\begin{align*}
\text{O2ORouter} & \parallel \text{O2OEvents} \\
& (T(\ldots \text{c.write}(o); \ldots) \parallel T(\ldots \text{a}=\text{c.read}(); \ldots)) \\
\leadsto & \cdots [\cdots] (\!o \rightarrow T(\ldots\ldots) \parallel ?a \rightarrow T(\ldots\ldots))
\end{align*}
\]

Further execution is performed by CSP calculus:

\[
\begin{align*}
\leadsto & \cdots \\
\leadsto & \!o \rightarrow (\text{O2ORouter} [\cdots] (T(\ldots\ldots) \parallel T(\ldots\ldots)))
\end{align*}
\]
Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:
  - Calculus
  - LTL, CTL, etc.
  - Based on operational semantics of CSP
  - Rewriting system based on algebraic laws
  - Rewriting system based on partial order extension of CSP
Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:
  - $\mu$-Calculus
Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:
  - $\mu$-Calculus
  - LTL, CTL, etc.
Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:
  - $\mu$-Calculus
  - LTL, CTL, etc.
- Possible CSP calculi:
Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:
  - \( \mu \)-Calculus
  - LTL, CTL, etc.

- Possible CSP calculi:
  - Based on operational semantics of CSP
Symbolic Analysis of JCSP Systems

• Possible logics for expressing properties of processes:
  • μ-Calculus
  • LTL, CTL, etc.

• Possible CSP calculi:
  • Based on operational semantics of CSP
  • Rewriting system based on algebraic laws
Symbolic Analysis of JCSP Systems

- Possible logics for expressing properties of processes:
  - μ-Calculus
  - LTL, CTL, etc.

- Possible CSP calculi:
  - Based on operational semantics of CSP
  - Rewriting system based on algebraic laws
  - Rewriting system based on partial order extension of CSP
Summary: Modelling JCSP

- Representation of simple channels finished
Summary: Modelling JCSP

- Representation of simple channels finished
- Incomplete: More complex communication (e.g. buffered channels, barriers, sending of complex data structures)
Summary: CSP calculi

- Most encouraging results with partial order approach

Further investigation needed for:
- Interface to modal logic
- Interaction with user
- Treatment of proving techniques (postponed): Induction, compositional proving
Summary: CSP calculi

- Most encouraging results with partial order approach
- Further investigation needed for:
  - Interface to modal logic
  - Interaction with user
  - Treatment of proving techniques (postponed): Induction, compositional proving
Summary: CSP calculi

- Most encouraging results with partial order approach
- Further investigation needed for:
  - Interface to modal logic
Summary: CSP calculi

- Most encouraging results with partial order approach
- Further investigation needed for:
  - Interface to modal logic
  - Interaction with user
Summary: CSP calculi

- Most encouraging results with partial order approach
- Further investigation needed for:
  - Interface to modal logic
  - Interaction with user
  - Treatment of proving techniques (postponed): Induction, compositional proving