Modelling and Validation of Concurrent System

António Ravara

May 7, 2024

Applications

Models in Computer Science

Aim
Formally and rigorously represent the behaviour of systems

Models in Computer Science

Aim
Formally and rigorously represent the behaviour of systems

Some kinds

1. Sequential programming is adequately modeled with *functions*
   From an input calculate in finite time an output
   (if the problem is decidable).

Models in Computer Science

Aim
Formally and rigorously represent the behaviour of systems

Some kinds

1. Sequential programming is adequately modeled with *functions*
   From an input calculate in finite time an output
   (if the problem is decidable).
2. Reactive systems are adequately modeled with *processes*
   Represent (possibly infinite) communications/events patterns

The Calculus of Communicating Systems

Key ingredients

1. A language to specify communication-based concurrent systems.

The Calculus of Communicating Systems

Key ingredients

1. A language to specify communication-based concurrent systems.
2. An execution mechanism
   Structural Operation Semantics.

The Calculus of Communicating Systems

Key ingredients

1. A language to specify communication-based concurrent systems.
2. An execution mechanism
   Structural Operation Semantics.
3. A mathematical model supporting behavioural reasoning
   Labelled Transition Systems.

The Calculus of Communicating Systems

Key ingredients

1. A language to specify communication-based concurrent systems.
2. An execution mechanism
   Structural Operation Semantics.
3. A mathematical model supporting behavioural reasoning
   Labelled Transition Systems.
4. A congruence notion, capturing behavioural equivalence
   Bisimilarity.

For what is CCS used?

Versions of it are used to model and show correct various types of systems:

1. Hardware design
2. Operating Systems
3. Systems Biology
4. (Business and Computer) Protocols

For what is CCS used?

Versions of it are used to model and show correct various types of systems:

1. Hardware design
2. Operating Systems
3. Systems Biology
4. (Business and Computer) Protocols

A famous example

In 1995, Gavin Lowe (Univ of Oxford) found an attack on the Needham-Schroeder public-key authentication protocol

He was actually trying to prove it correct...

For what is CCS used?

Versions of it are used to model and show correct various types of systems:

1. Hardware design
2. Operating Systems
3. Systems Biology
4. (Business and Computer) Protocols

A famous example

In 1995, Gavin Lowe (Univ of Oxford) found an attack on the Needham-Schroeder public-key authentication protocol

He was actually trying to prove it correct...

Let us illustrate how to use CCS by implementing a communication protocol, and by proving it correct.

The Alternating-Bit Protocol

ABP is a simple communication protocol which assumes that the network may loose messages. Its aim is managing the retransmission of lost messages in a communicating system.

The Alternating-Bit Protocol

ABP is a simple communication protocol which assumes that the network may loose messages. Its aim is managing the retransmission of lost messages in a communicating system.

Description

1. Messages have a content and an extra bit.

The Alternating-Bit Protocol

ABP is a simple communication protocol which assumes that the network may loose messages. Its aim is managing the retransmission of lost messages in a communicating system.

Description

1. Messages have a content and an extra bit.
2. Messages are repeatedly sent (with the same bit) until an acknowledgement message with the same bit is received.

The Alternating-Bit Protocol

ABP is a simple communication protocol which assumes that the network may loose messages. Its aim is managing the retransmission of lost messages in a communicating system.

Description

1. Messages have a content and an extra bit.
2. Messages are repeatedly sent (with the same bit) until an acknowledgement message with the same bit is received.
3. Consider a sender S and a receiver R, and assume a communication medium M from S to R (which do not communicate directly).

The Alternating-Bit Protocol

The protocol at work

1. R sends an acknowledgement message to S (with the same bit) as soon as it receives the first message.

The protocol at work

1. R sends an acknowledgement message to S (with the same bit) as soon as it receives the first message.
   The first message received is processed; the subsequent are just acknowledge.

The protocol at work

1. R sends an acknowledgement message to S (with the same bit) as soon as it receives the first message.
   The first message received is processed; the subsequent are just acknowledge.
2. S stops transmitting a message when it receives an acknowledgement with the same bit.

The protocol at work

1. R sends an acknowledgement message to S (with the same bit) as soon as it receives the first message.
   The first message received is processed; the subsequent are just acknowledge.
2. S stops transmitting a message when it receives an acknowledgement with the same bit.
   Then it flips the bit and starts transmitting another message.

How can we implement it in CCS? We need to be able of sending values (the bit, in this case).

Value-passing CCS

Let us consider a straightforward extension of CCS where, apart from names, we also use value expressions (integer, boolean, etc).

Value-passing CCS

Let us consider a straightforward extension of CCS where, apart from names, we also use value expressions (integer, boolean, etc).

To support communication of values

1. Outputs may send values: consider that the expression e evaluates to value v (denoted) $e \Downarrow v$:
   $\overline{a}\langle e\rangle.0 \xrightarrow{\overline{a}\langle v\rangle} 0$

Value-passing CCS

Let us consider a straightforward extension of CCS where, apart from names, we also use value expressions (integer, boolean, etc).

To support communication of values

1. Outputs may send values: consider that the expression e evaluates to value v (denoted) $e \Downarrow v$:
   $\overline{a}\langle e\rangle.0 \xrightarrow{\overline{a}\langle v\rangle} 0$
2. The correspondent inputs have a formal parameter:
   $a(x).P \xrightarrow{a(x)} P$

Value-passing CCS

Let us consider a straightforward extension of CCS where, apart from names, we also use value expressions (integer, boolean, etc).

To support communication of values

1. Outputs may send values: consider that the expression e evaluates to value v (denoted) $e \Downarrow v$:
   $\overline{a}\langle e\rangle.0 \xrightarrow{\overline{a}\langle v\rangle} 0$
2. The correspondent inputs have a formal parameter:
   $a(x).P \xrightarrow{a(x)} P$
3. Communication happens as follows
   $\overline{a}\langle e\rangle.0 \mid a(x).P \xrightarrow{\tau} 0 \mid P\{x \leftarrow v\}$

Value-passing CCS

Let us consider a straightforward extension of CCS where, apart from names, we also use value expressions (integer, boolean, etc).

To support communication of values

1. Outputs may send values: consider that the expression e evaluates to value v (denoted) $e \Downarrow v$:
   $\overline{a}\langle e\rangle.0 \xrightarrow{\overline{a}\langle v\rangle} 0$
2. The correspondent inputs have a formal parameter:
   $a(x).P \xrightarrow{a(x)} P$
3. Communication happens as follows
   $\overline{a}\langle e\rangle.0 \mid a(x).P \xrightarrow{\tau} 0 \mid P\{x \leftarrow v\}$

Example: one place buffer (memory cell)

$Cell = in(x).\overline{out}\langle x\rangle.Cell$

Value-passing CCS

It is also useful to include a conditional process, to encode decisions:

$$\textbf{if}\,(e)\,P\,\textbf{else}\,Q$$

executes either P or Q,
depending on the boolean value obtained from e

Value-passing CCS

It is also useful to include a conditional process, to encode decisions:

$$\textbf{if}\,(e)\,P\,\textbf{else}\,Q$$

executes either P or Q,
depending on the boolean value obtained from e

Example: natural addition

(parameters x and y, reply name r)

$$Sum(x, y, r) = \textbf{if}\,(y = 0)\,\overline{r}\langle x\rangle\,\textbf{else}\,Sum(x + 1, y - 1, r)$$

Example: n**-place buffer (parametric definition)**

Starts empty; it is able of storing n values. Let $k \geq 1$.

$$
\begin{array}{rcl}
ECell(n) & = & in(x).Buf(x, n) \\
Buf(x_1, \ldots, x_k, n) & = & \textbf{if}\,(k = n)\, Out\langle x_1, \ldots, x_k, n\rangle\, \textbf{else} \\
 & & \quad \textbf{if}\,(k < n)\, IO\langle x_1, \ldots, x_k, n\rangle\, \textbf{else}\, 0 \\
Out(x_1, \ldots, x_k, n) & = & \overline{out}\langle x_k\rangle Buf\langle x_1, \ldots, x_{k-1}, n\rangle \\
In(x_1, \ldots, x_k, n) & = & in(x).Buf\langle x, x_1, \ldots, x_k, n\rangle \\
IO(x_1, \ldots, x_k, n) & = & In\langle x_1, \ldots, x_k, n\rangle + Out\langle x_1, \ldots, x_k, n\rangle
\end{array}
$$

Description

- The medium starts with no message in transit;

The Alternating-Bit Protocol in CCS

Description

- The medium starts with no message in transit;
- The *Sender* may receive (old) acknowledgement messages (of a message already delivered) or accept a new message;

The Alternating-Bit Protocol in CCS

Description

- The medium starts with no message in transit;
- The *Sender* may receive (old) acknowledgement messages (of a message already delivered) or accept a new message;
- once *Sender* accepts a new message it flips the bit (to distinguish its acknowledgements from those of the previous message) and starts repeatedly sending the message; or,

The Alternating-Bit Protocol in CCS

Description

- The medium starts with no message in transit;
- The *Sender* may receive (old) acknowledgement messages (of a message already delivered) or accept a new message;
- once *Sender* accepts a new message it flips the bit (to distinguish its acknowledgements from those of the previous message) and starts repeatedly sending the message; or,
- if the received acknowledgement message contains the right bit, then the deliver of the message is confirmed and *Sender* is ready again to accept a new message.

The Alternating-Bit Protocol in CCS

Let us ignore the content of the messages, assume b is the current bit in use, and define the system as

$$\begin{aligned} \mathit{System}(b) &= (\textbf{new}\, \mathit{ack}, \mathit{rec}, \mathit{reply}, \mathit{send}) \\ &\quad (\mathit{Sender}\langle b\rangle \mid \mathit{Medium} \mid \mathit{Receiver}\langle b\rangle) \end{aligned}$$

$$\mathit{Sender}(b) = \mathit{accept}.\mathit{Sending}\langle b+1\rangle + \mathit{ack}(x).\mathit{Sender}\langle b\rangle$$

with

$$\begin{aligned} \mathit{Sending}(b) &= \overline{\mathit{send}}\langle b\rangle.\mathit{Sending}\langle b\rangle + \\ &\quad \mathit{ack}(x).\textbf{if}\,(x = b)\, \mathit{Sender}\langle b\rangle\, \textbf{else}\, \mathit{Sending}\langle b\rangle \end{aligned}$$

Recall bit addition: $0 + 1 = 1$ and $1 + 1 = 0$

The Alternating-Bit Protocol in CCS

The *Receiver* has the following definition:

$$
\begin{array}{l}
\mathit{Receiver}(b) = \\
\mathit{rec}(x).\mathbf{if}\,(x = b+1)\,\mathit{Received}\langle x, b+1\rangle\,\mathbf{else}\,\mathit{Receiver}\langle b\rangle
\end{array}
$$

where

$$
\begin{array}{rcl}
\mathit{Received}(x, b) & = & \overline{\mathit{reply}}\langle b\rangle.\mathit{Received}\langle x, b\rangle + \\
 & & \mathit{rec}(x). \\
 & & \quad \mathbf{if}\,(x = b)\,\overline{\mathit{deliver}}.\mathit{Receiver}\langle b\rangle\,\mathbf{else}\,\mathit{Received}\langle x, b\rangle
\end{array}
$$

The Alternating-Bit Protocol in CCS

The *Receiver* has the following definition:

$$Receiver(b) =$$
$$rec(x).\mathbf{if}\,(x = b+1)\, Received\langle x, b+1\rangle\, \mathbf{else}\, Receiver\langle b\rangle$$

where

$$\begin{array}{rcl} Received(x,b) & = & \overline{reply}\langle b\rangle.Received\langle x,b\rangle + \\ & & rec(x). \\ & & \quad \mathbf{if}\,(x = b)\,\overline{deliver}.Receiver\langle b\rangle\,\mathbf{else}\,Received\langle x,b\rangle \end{array}$$

Medium connects *Sender* and *Receiver*, forwarding messages back and forth, but in an unreliable way (may loose some messages):

$Medium = StoR \mid RtoS$,where

$$\begin{array}{rcl} StoR & = & send(x).(\overline{rec}\langle x\rangle.StoR + \tau.StoR) \\ RtoS & = & reply(x).(\overline{ack}\langle x\rangle.RtoS + \tau.RtoS) \end{array}$$

Correctness of the Alternating-Bit Protocol in CCS

Our implementation of ABP:

$$\begin{array}{rcl} \mathit{System}(b) & = & (\mathbf{new}\, \mathit{ack}, \mathit{rec}, \mathit{reply}, \mathit{send}) \\ & & (\mathit{Sender}\langle b\rangle \mid \mathit{Medium} \mid \mathit{Receiver}\langle b\rangle) \end{array}$$

- The initial state of the system behaves like a one-place buffer (remembering the last used bit), ready to accept a message and meanwhile discarding old acknowledgements.
- In spite of losses or of duplication of messages, exactly one message should be transmitted.

Ideal system (not considering the content of the message)
$\mathit{Spec} = \mathit{accept}.\overline{\mathit{deliver}}.\mathit{Spec}$

Correctness criterion
System(b) and *Spec* should be equivalent (for any b).

How to prove correct our implementation of the ABP?

Do we want to show that $System(b) \sim Spec$? This obviously does not hold:

1. $System(b)$ transits by *accept* to

   $(\mathbf{new}\, ack, rec, reply, send)(Sending\langle b+1\rangle | Medium | Receiver\langle b\rangle)$

2. $Spec$ transits by *accept* to $\overline{deliver}.Spec$
3. $System(b)$ can now do a (possibly infinite) sequence of τ-steps, and then may do $\overline{deliver}$
4. $Spec$ can only do $\overline{deliver}$

So, an attacker easily wins the bisimulation game...

How to prove correct our implementation of the ABP?

Do we want to show that $\mathit{System}(b) \sim \mathit{Spec}$? This obviously does not hold:

1. $\mathit{System}(b)$ transits by *accept* to

   $(\mathbf{new}\, \mathit{ack}, \mathit{rec}, \mathit{reply}, \mathit{send})(\mathit{Sending}\langle b+1\rangle | \mathit{Medium} | \mathit{Receiver}\langle b\rangle)$
2. Spec transits by *accept* to $\overline{\mathit{deliver}}.\mathit{Spec}$
3. $\mathit{System}(b)$ can now do a (possibly infinite) sequence of τ-steps, and then may do $\overline{\mathit{deliver}}$
4. Spec can only do $\overline{\mathit{deliver}}$

So, an attacker easily wins the bisimulation game...

However, the problem is that the bisimilarity relation discriminates too much: the internal actions of *System* are not relevant as are not observable and should be discarded.

Weak bisimilarity

Again, a behavioural equivalence for concurrency...

Problem

Strong bisimilarity does not abstract away from τ actions.

Again, a behavioural equivalence for concurrency...

Problem

Strong bisimilarity does not abstract away from τ actions.

$$\begin{array}{ccc} a.\tau.0 & \overset{?}{\sim} & a.0 \\ \Big\downarrow{\scriptstyle a} & & \Big\downarrow{\scriptstyle a} \\ \tau.0 & \not\sim & 0 \end{array}$$

Again, a behavioural equivalence for concurrency...

Problem

Strong bisimilarity does not abstract away from τ actions.

$$
\begin{array}{ccc}
a.\tau.0 & \overset{?}{\sim} & a.0 \\
\Big\downarrow a & & \Big\downarrow a \\
\tau.0 & \not\sim & 0
\end{array}
$$

We need to (carefully) disregard silent actions.

Weak bisimulation - a naïve approach

Let $(\mathrm{Proc}, \mathrm{Act}, \{\xrightarrow{a} \mid a \in \mathrm{Act}\})$ be an LTS such that $\tau \in \mathrm{Act}$.

Weak Transition Relation

$$\overset{a}{\Longrightarrow} = \begin{cases} (\xrightarrow{\tau})^* \circ \xrightarrow{a} \circ (\xrightarrow{\tau})^*, & \text{if } a \neq \tau \\ (\xrightarrow{\tau})^*, & \text{otherwise} \end{cases}$$

Weak bisimulation - a naïve approach

Let $(\mathrm{Proc}, \mathrm{Act}, \{\xrightarrow{a} \mid a \in \mathrm{Act}\})$ be an LTS such that $\tau \in \mathrm{Act}$.

Weak Transition Relation

$$\overset{a}{\Longrightarrow} = \begin{cases} (\xrightarrow{\tau})^* \circ \xrightarrow{a} \circ (\xrightarrow{\tau})^*, & \text{if } a \neq \tau \\ (\xrightarrow{\tau})^*, & \text{otherwise} \end{cases}$$

- $p \overset{\tau}{\Longrightarrow} q$ denotes a transition from p to q by zero or more τ actions.

Weak bisimulation - a naïve approach

Let $(\mathrm{Proc}, \mathrm{Act}, \{\xrightarrow{a} \mid \ a \in \mathrm{Act}\})$ be an LTS such that $\tau \in \mathrm{Act}$.

Weak Transition Relation

$$\overset{a}{\Longrightarrow} = \begin{cases} (\xrightarrow{\tau})^* \circ \xrightarrow{a} \circ (\xrightarrow{\tau})^*, & \text{if } a \neq \tau \\ (\xrightarrow{\tau})^*, & \text{otherwise} \end{cases}$$

- $p \overset{\tau}{\Longrightarrow} q$ denotes a transition from p to q by zero or more τ actions.
- If $a \neq \tau$ then $p \overset{a}{\Longrightarrow} q$ denotes a transition from p to q by:
    1. zero or more τ actions, followed by
    2. a (strong) a transition, followed by
    3. zero or more τ actions

Weak bisimilarity

Weak Simulation

A binary relation $\mathcal{R} \subseteq \mathsf{Proc} \times \mathsf{Proc}$ is a *weak simulation*, if whenever $(p, q) \in \mathcal{R}$ then for each $a \in \mathsf{Act}$:

$$\text{if } p \xrightarrow{a} p' \text{ then } q \overset{a}{\Longrightarrow} q' \text{ for some } q' \text{ such that } (p', q') \in \mathcal{R}$$

Weak bisimilarity

Weak Simulation

A binary relation $\mathcal{R} \subseteq \mathrm{Proc} \times \mathrm{Proc}$ is a *weak simulation*, if whenever $(p, q) \in \mathcal{R}$ then for each $a \in \mathrm{Act}$:

$$\text{if } p \xrightarrow{a} p' \text{ then } q \overset{a}{\Longrightarrow} q' \text{ for some } q' \text{ such that } (p', q') \in \mathcal{R}$$

A weak simulation $\mathcal{R}$ is a *weak bisimulation*,

if $\mathcal{R}^{-1}$ is also a weak simulation.

Weak bisimilarity

Weak Simulation

A binary relation $\mathcal{R} \subseteq \mathrm{Proc} \times \mathrm{Proc}$ is a *weak simulation*, if whenever $(p, q) \in \mathcal{R}$ then for each $a \in \mathrm{Act}$:

$$\text{if } p \xrightarrow{a} p' \text{ then } q \overset{a}{\Longrightarrow} q' \text{ for some } q' \text{ such that } (p', q') \in \mathcal{R}$$

A weak simulation $\mathcal{R}$ is a *weak bisimulation*,

if $\mathcal{R}^{-1}$ is also a weak simulation.

Weak Bisimilarity

Two processes $p, q \in \mathrm{Proc}$ are *weakly bisimilar* ($p_1 \approx p_2$), if there exists a weak bisimulation $\mathcal{R}$ such that $(p, q) \in \mathcal{R}$.

$$\approx \; = \; \cup\{\mathcal{R} \mid \mathcal{R} \text{ is a weak bisimulation}\}$$

Properties of weak bisimilarity

- It includes strong bisimulation.
- It is the largest bisimulation.
- It is an equivalence relation.
- $(\mathrm{Proc}, |, 0)$ and $(\mathrm{Proc}, +, 0)$ are commutative monoïds.
- $a.\tau.P \approx a.P$, $P + \tau.P \approx \tau.P$, and
  $a.(P + \tau.Q) \approx a.(P + \tau.Q) + a.Q$
- It is preserved by prefixing, parallel composition and restriction.

Properties of weak bisimilarity

- It includes strong bisimulation.
- It is the largest bisimulation.
- It is an equivalence relation.
- $(\mathrm{Proc}, |, 0)$ and $(\mathrm{Proc}, +, 0)$ are commutative monoïds.
- $a.\tau.P \approx a.P$, $P + \tau.P \approx \tau.P$, and
  $a.(P + \tau.Q) \approx a.(P + \tau.Q) + a.Q$
- It is preserved by prefixing, parallel composition and restriction.

What about choice?

Consider the processes $\tau.a.0$ and $a.0$

- One easily shows that $\tau.a.0 \approx a.0$

Properties of weak bisimilarity

- It includes strong bisimulation.
- It is the largest bisimulation.
- It is an equivalence relation.
- $(\mathrm{Proc}, |, 0)$ and $(\mathrm{Proc}, +, 0)$ are commutative monoïds.
- $a.\tau.P \approx a.P$, $P + \tau.P \approx \tau.P$, and
  $a.(P + \tau.Q) \approx a.(P + \tau.Q) + a.Q$
- It is preserved by prefixing, parallel composition and restriction.

What about choice?

Consider the processes $\tau.a.0$ and $a.0$

- One easily shows that $\tau.a.0 \approx a.0$
- However, one also shows easily that

$$\tau.a.0 + b.0 \not\approx a.0 + b.0$$

Weak bisimilarity is not a congruence relation

Choice does not preserve weak bisimilarity.

Source of the Problem

A τ transition can be matched by no transition.

Weak bisimilarity is not a congruence relation

Choice does not preserve weak bisimilarity.

Source of the Problem

A τ transition can be matched by no transition.

A Solution

A τ-transition must be matched by at least a τ-transition.

Then, $\tau.a.0$ is not equated with $a.0$

Weak bisimilarity is not a congruence relation

Choice does not preserve weak bisimilarity.

Source of the Problem

A τ transition can be matched by no transition.

A Solution

A τ-transition must be matched by at least a τ-transition.

Then, $\tau.a.0$ is not equated with $a.0$

Observational Equivalence

Processes p and q are *observationally equivalent* ($p = q$), whenever:

Weak bisimilarity is not a congruence relation

Choice does not preserve weak bisimilarity.

Source of the Problem

A τ transition can be matched by no transition.

A Solution

A τ-transition must be matched by at least a τ-transition.

Then, $\tau.a.0$ is not equated with $a.0$

Observational Equivalence

Processes p and q are *observationally equivalent* ($p = q$), whenever:

1. $p \approx q$

Weak bisimilarity is not a congruence relation

Choice does not preserve weak bisimilarity.

Source of the Problem

A τ transition can be matched by no transition.

A Solution

A τ-transition must be matched by at least a τ-transition.

Then, $\tau.a.0$ is not equated with $a.0$

Observational Equivalence

Processes p and q are *observationally equivalent* $(p = q)$, whenever:

1. $p \approx q$
2. if $p \xrightarrow{\tau} p'$ then $q \xrightarrow{\tau} q'' \xRightarrow{\tau} q'$ and $p' \approx q'$
3. if $q \xrightarrow{\tau} q'$ then $p \xrightarrow{\tau} p'' \xRightarrow{\tau} p'$ and $p' \approx q'$

Weak bisimilarity is not a congruence relation

Choice does not preserve weak bisimilarity.

Source of the Problem

A τ transition can be matched by no transition.

A Solution

A τ-transition must be matched by at least a τ-transition.

Then, $\tau.a.0$ is not equated with $a.0$

Observational Equivalence

Processes p and q are *observationally equivalent* ($p = q$), whenever:

1. $p \approx q$
2. if $p \xrightarrow{\tau} p'$ then $q \xrightarrow{\tau} q'' \stackrel{\tau}{\Longrightarrow} q'$ and $p' \approx q'$
3. if $q \xrightarrow{\tau} q'$ then $p \xrightarrow{\tau} p'' \stackrel{\tau}{\Longrightarrow} p'$ and $p' \approx q'$

Observational equivalence is a congruence relation.