Asynchronous Readers and Writers

A Half-Synchronous Operator
Overview

Introduction of the \( \downarrow \) together with the \( i \) and \( i' \)

\( ! \) and \( ? \) versus \( i \) and \( i' \)

\( i \) and \( i' \) using \( \odot \)

Design Level

Semantics of \( \downarrow, i, \) and \( i' \)

Example Application

The Future
Introduction, system overview

Asynchronous Readers and Writers
Introduction

- Purpose: Asynchronous Writing and Reading.
Introduction

- Purpose: Asynchronous Writing and Reading.
- In CSP writing over a channel is restricted to two processes interacting synchronously via an action containing the ! and the ?.
Introduction

- Purpose: Asynchronous Writing and Reading.
- In CSP writing over a channel is restricted to two processes interacting synchronously via an action containing the `!` and the `?`.
- Proposal: a *half-synchronous action* which allows a process to write a value $x$ over a channel $c$, together with a half-synchronous parallel alphabetised operator.
Introduction

- Purpose: Asynchronous Writing and Reading.
- In CSP writing over a channel is restricted to two processes interacting synchronously via an action containing the `!` and the `?`.
- Proposal: a *half-synchronous action* which allows a process to write a value $x$ over a channel $c$,
- without the requirement that the reading processes must be in a state where they can read the value $x$ over a channel $c$. 
Introduction

- Purpose: Asynchronous Writing and Reading.
- In CSP writing over a channel is restricted to two processes interacting synchronously via an action containing the `!` and the `?`.
- Proposal: a *half-synchronous action* which allows a process to write a value \( x \) over a channel \( c \),
- without the requirement that the reading processes must be in a state where they can read the value \( x \) over a channel \( c \).
- Together with a half-synchronous parallel alphabetised operator.
Introduction

Advantages of the half-synchronous operator ↓ with half-synchronous actions containing ↓ or ↓:

- it eases the complexity of the design eliminating arguably complex process specifications:
  - it is not necessary to use a buffer process in the model to achieve asynchronous writing and reading,
  - the writes (↓) and reads (↑) are asynchronous, which makes it possible to have an order of writes and reads that, if synchronous (→), would lead to a deadlock,

- by reducing the number of actions involved in this asynchronous writing and reading of the processes, improves the performance of the periodic hard real-time application,

- in a distributed computing system, for example a processor-coprocessor combination, the waiting time of the processor-coprocessor can be reduced.
Advantages of the half-synchronous operator $\Downarrow$ with half-synchronous actions containing $\mathfrak{1}$ or $\mathfrak{2}$:

- it eases the complexity of the design eliminating arguably complex process specifications:

"..."
Introduction

Advantages of the half-synchronous operator $\downarrow$ with half-synchronous actions containing $\downarrow$ or $\uparrow$:  
- it eases the complexity of the design eliminating arguably complex process specifications:
  - it is not necessary to use a buffer process in the model to achieve asynchronous writing and reading,
Introduction

Advantages of the half-synchronous operator \( \triangleright \) with half-synchronous actions containing \( \triangledown \) or \( \triangledown \):

- it eases the complexity of the design eliminating arguably complex process specifications:
  - it is not necessary to use a buffer process in the model to achieve asynchronous writing and reading,
  - the writes (\( \triangledown \)) and reads (\( \triangledown \)) are asynchronous, which makes it possible to have an order of writes and reads that, if synchronous (\( \triangledown, \triangledown \)), would lead to a deadlock,
Advantages of the half-synchronous operator $\downarrow$ with half-synchronous actions containing $\mathbf{i}$ or $\mathbf{?}$:

- it eases the complexity of the design eliminating arguably complex process specifications:
  - it is not necessary to use a buffer process in the model to achieve asynchronous writing and reading,
  - the writes ($\mathbf{i}$) and reads ($\mathbf{?}$) are asynchronous, which makes it possible to have an order of writes and reads that, if synchronous ($!$, $?$), would lead to a deadlock,
- by reducing the number of actions involved in this asynchronous writing and reading of the processes, improves the performance of the periodic hard real-time application,
Introduction

Advantages of the half-synchronous operator ↓ with half-synchronous actions containing ¡ or ¿:

- it eases the complexity of the design eliminating arguably complex process specifications:
  - it is not necessary to use a buffer process in the model to achieve asynchronous writing and reading,
  - the writes (¡) and reads (¿) are asynchronous, which makes it possible to have an order of writes and reads that, if synchronous (!, ?), would lead to a deadlock,
- by reducing the number of actions involved in this asynchronous writing and reading of the processes, improves the performance of the periodic hard real-time application,
- in a distributed computing system, for example a processor-coprocessor combination, the waiting time of the processor-coprocessor can be reduced.
Overview

Introduction of the \( \downarrow \) together with the \( \downarrow \) and \( ? \)

\( ! \) and \( ? \) versus \( \downarrow \) and \( ? \)

\( \downarrow \) and \( ? \) using \( \uparrow \)

Design Level

Semantics of \( \downarrow \), \( \downarrow \) and \( ? \)

Example Application

The Future
! and ? versus ⌜ and ⌝

Listing 1: Deadlock due to synchronous writing and reading

\[
A = c!x_1 \rightarrow c!y_1 \rightarrow d?x_2 \rightarrow d?y_2 \rightarrow SKIP \\
B = c?x_1 \rightarrow d!x_2 \rightarrow c?y_1 \rightarrow d!y_2 \rightarrow SKIP \\
AB = A\parallel B
\]
! and ? versus \(\uparrow\) and \(\downarrow\)

Listing 1: Deadlock due to synchronous writing and reading

\[
A = c!x_1 \rightarrow c!y_1 \rightarrow d?q x_2 \rightarrow d?q y_2 \rightarrow \text{SKIP}
\]

\[
B = c?q x_1 \rightarrow d!x_2 \rightarrow c?q y_1 \rightarrow d!y_2 \rightarrow \text{SKIP}
\]

\[
AB = A \parallel B
\]

trace: \(c.x_1\)
and versus \( i \) and \( {?} \)

Listing 1: Deadlock due to synchronous writing and reading

\[
A = c!x_1 \rightarrow c!y_1 \rightarrow d?x_2 \rightarrow d?y_2 \rightarrow SKIP \\
B = c?x_1 \rightarrow d!x_2 \rightarrow c?y_1 \rightarrow d!y_2 \rightarrow SKIP \\
AB = A || B
\]

trace: \( c.x_1 \)

Listing 2: No deadlock due to asynchronous writing and reading

\[
A = c_i x_1 \rightarrow c_i y_1 \rightarrow d_i x_2 \rightarrow d_i y_2 \rightarrow SKIP \\
B = c_i x_1 \rightarrow d_i x_2 \rightarrow c_i y_1 \rightarrow d_i y_2 \rightarrow SKIP \\
AB = A \downarrow B
\]
! and ? versus i and ė

Listing 1: Deadlock due to synchronous writing and reading
\[
A = \text{\texttt{c!x}}_1 \rightarrow \text{\texttt{c!y}}_1 \rightarrow \text{\texttt{d?x}}_2 \rightarrow \text{\texttt{d?y}}_2 \rightarrow \text{\texttt{SKIP}} \\
B = \text{\texttt{c?x}}_1 \rightarrow \text{\texttt{d!x}}_2 \rightarrow \text{\texttt{c?y}}_1 \rightarrow \text{\texttt{d!y}}_2 \rightarrow \text{\texttt{SKIP}} \\
AB = A || B
\]
trace: \text{\texttt{c.x}}_1

Listing 2: No deadlock due to asynchronous writing and reading
\[
A = \text{\texttt{c!x}}_1 \rightarrow \text{\texttt{c!y}}_1 \rightarrow \text{\texttt{d!x}}_2 \rightarrow \text{\texttt{d!y}}_2 \rightarrow \text{\texttt{SKIP}} \\
B = \text{\texttt{c!x}}_1 \rightarrow \text{\texttt{d!x}}_2 \rightarrow \text{\texttt{c!y}}_1 \rightarrow \text{\texttt{d!y}}_2 \rightarrow \text{\texttt{SKIP}} \\
AB = A \downarrow B
\]
many possible traces, for example:
\[
\text{\texttt{c!x}}_1 \rightarrow \text{\texttt{c!y}}_1 \rightarrow \text{\texttt{c!x}}_1 \rightarrow \text{\texttt{d!x}}_2 \rightarrow \text{\texttt{c!y}}_1 \rightarrow \text{\texttt{d!y}}_2 \rightarrow \text{\texttt{d!x}}_2 \rightarrow \\
\text{\texttt{d!y}}_2 \rightarrow \text{\texttt{SKIP}}
\]
Overview

Introduction of the \( \downarrow \) together with the \( \downarrow \) and \( \uparrow \)

\( \downarrow \) and \( \uparrow \) versus \( \downarrow \) and \( \uparrow \)

\( \downarrow \) and \( \uparrow \) using

Design Level

Semantics of \( \downarrow \), \( \downarrow \) and \( \uparrow \)

Example Application

The Future
and using

\[ G_1 \quad G_2 \]

\[ G_1 \bowtie G_2 \]

Asynchronous Readers and Writers August 23, 2016 9 / 22
Overview

Introduction of the \( \downarrow \) together with the \( \perp \) and \( ? \)

\( \perp \) and \( ? \) versus \( \perp \) and \( ? \)

\( \perp \) and \( ? \) using \( \square \)

Design Level

Semantics of \( \downarrow, \perp \) and \( ? \)

Example Application

The Future
Design Level

Separation of write/read actions in time
Design Level

Separation of write/read actions in time
Needs a Buffer
Design Level

Separation of write/read actions in time
Needs a Buffer

\[
A = \text{write}.x \rightarrow \text{SKIP} \\
B = \text{read}.x \rightarrow \text{SKIP} \\
Buffer = \text{write}.x \rightarrow \text{read}.x \rightarrow \text{SKIP} \\
AB = A || B || Buffer
\]
Design Level

Separation of write/read actions in time
Needs a Buffer

\[ A = \text{write}.x \rightarrow \text{SKIP} \]
\[ B = \text{read}.x \rightarrow \text{SKIP} \]
\[ \text{Buffer} = \text{write}.x \rightarrow \text{read}.x \rightarrow \text{SKIP} \]
\[ AB = A \parallel B \parallel \text{Buffer} \]

\[ A = c_i x \rightarrow \text{SKIP} \]
\[ B = c_o x \rightarrow \text{SKIP} \]
\[ AB = A \downarrow B \]
Overview

Introduction of the ↓ together with the ↓ and ↓

↓ and ↑ versus ↓ and ↓

↓ and ↑ using ↓

Design Level

Semantics of ↓, ↓ and ↓

Example Application

The Future
Semantics of $\downarrow$, $\iota$ and $\pi$

\[
\begin{align*}
P^c_{\iota x : T} & \xrightarrow{c} P' , \quad Q_1^c_{\iota x : T} \xrightarrow{c} Q'_1 , \ldots , \quad Q_n^c_{\iota x : T} \xrightarrow{c} Q'_n \\
P \downarrow Q_1 \downarrow \ldots \downarrow Q_n \xrightarrow{c} P' \downarrow Q_1 \downarrow \ldots \downarrow Q_n & \xrightarrow{c} P' \downarrow Q'_1 \downarrow \ldots \downarrow Q'_n \\
c_{\iota x : T} & \notin (X, Z)
\end{align*}
\]
*Semantics of \(\downarrow\), \(\uparrow\) and \(\ll\)*

\[
P^c_\uparrow x:T \rightarrow P', \quad Q_1^c_\uparrow x:T \rightarrow Q'_1, \ldots, \quad Q_n^c_\uparrow x:T \rightarrow Q'_n
\]

\[
P \downarrow Q_1 \downarrow \cdots \downarrow Q_n \uparrow P' \downarrow Q_1 \downarrow \cdots \downarrow Q_n \uparrow P' \downarrow Q'_1 \downarrow \cdots \downarrow Q'_n
\]

\[c_\downarrow x : T \notin (X, Z)\]

\[
Q_i^c_\uparrow x:T \rightarrow Q'_i, \quad Q_j^y \rightarrow Q'_j
\]

\[Q_i \downarrow Q_j^y \rightarrow Q_i \downarrow Q'_j, \quad y \neq c_\downarrow x : T, \quad c_\downarrow x : T \in (Y_i \cdot Y_j),
\]

\[y \notin (X, Y_k=1, \ldots, j \neq k, Z)\]
Semantics of $\downarrow$, $\iota$ and $\zeta$

\[
P \xrightarrow{c \iota} P' \quad Q_i \xrightarrow{c \iota} Q'_i, \ldots, Q_n \xrightarrow{c \iota} Q'_n
\]

\[
P \downarrow Q_1 \downarrow \cdots \downarrow Q_n \xrightarrow{c \iota} P' \downarrow Q_1 \downarrow \cdots \downarrow Q_n \xrightarrow{c \iota} P' \downarrow Q'_1 \downarrow \cdots \downarrow Q'_n
\]

\[
c \iota \ x : T \notin (X, Z)
\]

\[
Q_i \xrightarrow{c \iota} Q'_i, \quad Q_j \xrightarrow{y} Q'_j
\]

\[
Q_i \downarrow Q_j \xrightarrow{y} Q_i \downarrow Q'_j, \quad y \neq c \iota \ x : T, \ c \iota \ x : T \in (Y_i \cdot Y_j),
\]

\[
y \notin (X, Y_{k=1, \ldots, n}, j \neq k, Z)
\]

\[
P \xrightarrow{\cdot} P', \quad Q_i \xrightarrow{c \iota} Q'_i
\]

\[
P \xrightarrow{\cdot} P', \quad (\alpha(\cdot) \cdot (Y_1, \ldots, Y_n, Z)) = \emptyset
\]
Semantics of $\downarrow$, $i$ and $\hat{c}$

\[
\begin{align*}
P_{c \hat{c} x : T} & \rightarrow P', \quad Q_1_{c \hat{c} x : T} \rightarrow Q'_1, \ldots, \quad Q_n_{c \hat{c} x : T} \rightarrow Q'_n \\
\frac{P \downarrow Q_1 \downarrow \cdots \downarrow Q_n_{c \hat{c} x : T} \rightarrow P' \downarrow Q_1 \downarrow \cdots \downarrow Q_n_{c \hat{c} x : T} \rightarrow Q'_1 \downarrow \cdots \downarrow Q'_n}{c \hat{c} x : T \notin (X, Z)}
\end{align*}
\]

\[
\begin{align*}
Q_i_{c \hat{c} x : T} & \rightarrow Q'_i, \quad Q_j \rightarrow Q'_j \\
\frac{Q_i \downarrow Q_j \rightarrow Q_i \downarrow Q'_j}{y \neq c \hat{c} x : T, \quad c \hat{c} x : T \in (Y_i \cdot Y_j), \quad y \notin (X, Y_k=1, \ldots, n, j \neq k), Z}
\end{align*}
\]

\[
\begin{align*}
P_{\longrightarrow} & \rightarrow P', \quad Q_i_{c \hat{c} x : T} \rightarrow Q'_i \\
\frac{P_{\longrightarrow} \rightarrow P', \quad Q_i_{c \hat{c} x : T} \rightarrow Q'_i}{\alpha(\longrightarrow) \cdot (Y_1, \ldots, Y_n, Z) = \emptyset}
\end{align*}
\]

\[
\begin{align*}
Q_i & \rightarrow Q'_i, \quad Q_j_{c \hat{c} x : T} \rightarrow Q'_j \\
\frac{\text{SKIP}}{i \neq j}
\end{align*}
\]
Overview

Introduction of the \( \downarrow \) together with the \( i \) and \( ? \)

\( ! \) and \( ? \) versus \( i \) and \( ? \)

\( i \) and \( ? \) using \( \square \)

Design Level

Semantics of \( \downarrow, i \) and \( ? \)

Example Application

The Future
Case study

\[
\text{Application} = c_1 ! x_1 : T \rightarrow c_2 ? y_1 : T \rightarrow \\
\cdots \\
c_1 ! x_8 : T \rightarrow c_2 ? y_8 : T \rightarrow \\
\text{display}_f(y_1, \cdots, y_8) \rightarrow \text{SKIP}
\]
Case study

Application = \( c_1 ! x_1 : T \rightarrow c_2 ? y_1 : T \rightarrow \)
\[ \ldots \]
\( c_1 ! x_8 : T \rightarrow c_2 ? y_8 : T \rightarrow \)
\( display_f(y_1, \ldots, y_8) \rightarrow \text{SKIP} \)

Controller = \( c_1 ? x_1 : T \rightarrow \text{writeCoProc}.x_1 \rightarrow \)
\( \text{readCoProc}.y_1 \rightarrow c_2 ! y_1 : T \rightarrow \)
\[ \ldots \]
\( c_1 ? x_8 : T \rightarrow \text{writeCoProc}.x_8 \rightarrow \)
\( \text{readCoProc}.y_8 \rightarrow c_2 ! y_8 : T \rightarrow \text{SKIP} \)
Case study

Application = $c_1 \parallel x_1 : T \rightarrow c_2 \parallel y_1 : T \rightarrow$

\[ \ldots \]

$c_1 \parallel x_8 : T \rightarrow c_2 \parallel y_8 : T \rightarrow$

display $f(y_1, \cdots, y_8) \rightarrow$ SKIP

Controller = $c_1 \parallel x_1 : T \rightarrow writeCoProc.x_1 \rightarrow$

readCoProc.$y_1 \rightarrow c_2 \parallel y_1 : T \rightarrow$

\[ \ldots \]

$c_1 \parallel x_8 : T \rightarrow writeCoProc.x_8 \rightarrow$

readCoProc.$y_8 \rightarrow c_2 \parallel y_8 : T \rightarrow$ SKIP

System$_1 = Application_A \parallel C Controller$
Example application
Example application

\[ \text{Application} = c_1 \downarrow x_1 : T \rightarrow \cdots \rightarrow c_1 \downarrow x_8 : T \rightarrow c_2 \uparrow y_1 : T \rightarrow \cdots \rightarrow c_2 \uparrow y_8 : T \rightarrow \text{display}_f(y_1, \cdots, y_8) \rightarrow \text{SKIP} \]
Example application

\[
\text{Application} = c_1 \mathcal{i} x_1 : T \rightarrow \cdots \rightarrow c_1 \mathcal{i} x_8 : T \rightarrow c_2 \mathcal{i} y_1 : T \rightarrow \cdots \rightarrow c_2 \mathcal{i} y_8 : T \rightarrow \text{display}_f(y_1, \cdots, y_8) \rightarrow \text{SKIP}
\]

\[
\text{Controller} = c_1 \mathcal{c} x_1 : T \rightarrow \text{writeCoProc}.x_1 \rightarrow \text{readCoProc}.y_1 \rightarrow c_2 \mathcal{i} y_1 : T \rightarrow \cdots
\]

\[
c_1 \mathcal{c} x_8 : T \rightarrow \text{writeCoProc}.x_8 \rightarrow \text{readCoProc}.y_8 \rightarrow c_2 \mathcal{i} y_8 : T \rightarrow \text{SKIP}
\]
Example application

\[
\text{Application} = c_1 \triangleright x_1 : T \rightarrow \cdots \rightarrow c_1 \triangleright x_8 : T \rightarrow \\
c_2 \triangleright y_1 : T \rightarrow \cdots \rightarrow c_2 \triangleright y_8 : T \rightarrow \\
\text{display}_f(y_1, \cdots, y_8) \rightarrow \text{SKIP}
\]

\[
\text{Controller} = c_1 \triangleright x_1 : T \rightarrow \text{writeCoProc}.x_1 \rightarrow \\
\text{readCoProc}.y_1 \rightarrow c_2 \triangleright y_1 : T \rightarrow \\
\cdots
\]

\[
c_1 \triangleright x_8 : T \rightarrow \text{writeCoProc}.x_8 \rightarrow \\
\text{readCoProc}.y_8 \rightarrow c_2 \triangleright y_8 : T \rightarrow \text{SKIP}
\]

\[
\text{System}_2 = \text{Application}_{A \Downarrow C} \downarrow \text{Controller}
\]
Example application
Overview

Introduction of the ↓ together with the ↓ and ↓
↓ and ↓ versus ↓ and ↓
↓ and ↓ using ↓

Design Level

Semantics of ↓, ↓ and ↓

Example Application

The Future
The Future

Future work:

- Index the half-synchronous action such that it is set-wise asynchronous and intra-set-wise synchronous.
- Elaborate the graph-theoretical characteristics of $(VRSP)$ together with the half-synchronous operator, i.e. it is a commutative monoid of consistent graphs.
- Implementation in a tool-chain.
- Perform a case-study on a periodic hard real-time system.
Future work:

- Index the half-synchronous action such that it is set-wise asynchronous and intra-set-wise synchronous,
The Future

Future work:

- Index the half-synchronous action such that it is set-wise asynchronous and intra-set-wise synchronous,
- elaborate the graph-theoretical characteristics of $\mathbb{C}$ (VRSP together with the half-synchronous operator),
The Future

Future work:

- Index the half-synchronous action such that it is set-wise asynchronous and intra-set-wise synchronous,
- elaborate the graph-theoretical characteristics of \( \mathcal{R} \) (VRSP together with the half-synchronous operator), i.e. it is a commutative monoid of consistent graphs,
The Future

Future work:

- Index the half-synchronous action such that it is set-wise asynchronous and intra-set-wise synchronous,

- elaborate the graph-theoretical characteristics of $\mathcal{H}$ (VRSP together with the half-synchronous operator), i.e. it is a commutative monoid of consistent graphs,

- implementation in a tool-chain
The Future

Future work:

- Index the half-synchronous action such that it is set-wise asynchronous and intra-set-wise synchronous,
- Elaborate the graph-theoretical characteristics of $\mathcal{VRSP}$ together with the half-synchronous operator), i.e. it is a commutative monoid of consistent graphs,
- Implementation in a tool-chain and
- Perform a case-study on a periodic hard real-time system.
Thanks!