Translating synchronous to asynchronous communication and vice versa.

Jan Friso Groote
About which notion of (a)synchrony are we talking?

Synchronous communication: common in modelling

Asynchronous communication: common in implementation
Asynchronous → synchronous

\[ \text{Queue}(q, \text{List}(D)) = \sum_{d \in D} \text{read}(d).\text{Queue}(d \gg q) \]
\[ + (\neg \text{empty}(q)) \rightarrow \text{send}(\text{rhead}(q)).\text{Queue}(\text{rtail}(q)) \]
Asynchrony ≠ asynchrony

Lossy queue

Priority queue

Bounded queue

Complex queue topologies
1986: Philips Computer Integrated Manufacturing architecture (Biemans, Sjoerdsma)

CIM architecture fully described in LOTOS
How to implement this synchronous language?
Synchronous → asynchronous

Buckley & Silberschatz 1983
Ramesh 1987
Occam & transputer
What is the problem? Conflicting interactions.

Which action will occur? a!3, b!7 or c!23
1. Order all processes
2. Propose interaction to higher parties
3. Commit to proposals from lower parties
4. (Dis)agree to commits as soon as possible

Buckley and Silberschatz, 1983
Ramesh, 1987
1. Order all processes
2. Propose interaction to higher parties
3. Commit to proposals from lower parties
4. (Dis)agree to commits as soon as possible

1 2 3
propose a!3
commit a!3
agree a!3
propose b!7
commit b!7
disagree b!7

a!3

a!3

b?y

b!7

c?z

c!23

1. Order all processes
2. Propose interaction to higher parties
3. Commit to proposals from lower parties
4. (Dis)agree to commits as soon as possible
Commit dependencies are non cyclic.

Waiting commits to be (dis)agreed.

Complexity: three messages per proposed interaction.
Optimisations are often possible: 2 messages.

a!3 does not occur in a choice.

Theorem: Implementation by 2 messages preserves testing equivalence (LOTOS, 1988)

This is the communication scheme in OCCAM (Transputer)
Optimisations are often possible: 1 message.

\[
\begin{align*}
   & a!3 \\
   & a?x
\end{align*}
\]

1!3 does not occur in a choice, and has no subsequent activity.

Typical example:

\[
p \parallel a!3.\delta
\]
Asynchronous communication
≈
Synchronous communication
Challenge: Implement synchronous models efficiently in an asynchronous, distributed way

```
obtain_resource1.Use_resource1
+
obtain_resource2.Use_resource2
+
receive_cancel_message.Do_something_else
+
……. (further alternatives)
```