Data

Process Refinement

November 2006

Contents

- Data
- Channels
- Messages
- Media

Data

The process language of CSP is augmented by a language of datatypes: sets, functions, and sequences.

The names of processes and events may be parameterised using values from sets of any type, including sets of events.

Process parameters

We may use parameters in process names to record state information, representing the stages of a process in terms of the value of a collection of variables.

If a process name includes a state parameter, it is better to enclose it within a local definition environment.

Local definitions

If D is a collection of declarations—of sets, functions, or processes—then the expression

let D within E

has the value of E within the additional context provided by D.

Example: lift

$$
\begin{aligned}
Lift \;=\; & \text{let} \\
& \quad LiftAtFloor(0) = up \rightarrow LiftAtFloor(1) \\
& \quad LiftAtFloor(1) = down \rightarrow LiftAtFloor(0) \\
& \qquad\qquad\qquad\qquad \Box \\
& \qquad\qquad\qquad\qquad up \rightarrow LiftAtFloor(2) \\
& \quad LiftAtFloor(2) = down \rightarrow LiftAtFloor(1) \\
& \text{within} \\
& \quad LiftAtFloor(0)
\end{aligned}
$$

Conditionals

The expression

$$\text{if } B \text{ then } X \text{ else } Y$$

is X if B is true, and Y otherwise.

Example: Lift

$$
\begin{aligned}
\mathit{Lift} \;=\; & \text{let} \\
& \quad \mathit{LiftAtFloor}(n) \;=\; \\
& \qquad \text{if } n = 0 \text{ then} \\
& \qquad\quad \mathit{up} \rightarrow \mathit{LiftAtFloor}(1) \\
& \qquad \text{else if } n \in 1\,..\,8 \text{ then} \\
& \qquad\quad \mathit{down} \rightarrow \mathit{LiftAtFloor}(n-1) \\
& \qquad\quad \Box \\
& \qquad\quad \mathit{up} \rightarrow \mathit{LiftAtFloor}(n+1) \\
& \qquad \text{else} \\
& \qquad\quad \mathit{down} \rightarrow \mathit{LiftAtFloor}(8) \\
& \text{within} \\
& \quad \mathit{LiftAtFloor}(0)
\end{aligned}
$$

Boolean guards

$B \,\&\, P$

denotes the conditional process expression

if B then P else $Stop$

$$B \,\&\, P$$
$$\Box$$
$$C \,\&\, Q$$

is a more convenient way of presenting

$$
\begin{array}{l}
\text{if } B \wedge C \text{ then} \\
\quad P \;\Box\; Q \\
\text{else if } B \wedge \neg\, C \text{ then} \\
\quad P \\
\text{else if } C \wedge \neg\, B \text{ then} \\
\quad Q \\
\text{else} \\
\quad \mathit{Stop}
\end{array}
$$

Example: lift

$$
\begin{aligned}
Lift \;=\; & \textbf{let} \\
& \quad LiftAtFloor(n) \;= \\
& \qquad n \leq 8 \;\&\; up \rightarrow LiftAtFloor(n+1) \\
& \qquad \Box \\
& \qquad n \geq 1 \;\&\; down \rightarrow LiftAtFloor(n-1) \\
& \textbf{within} \\
& \quad LiftAtFloor(0)
\end{aligned}
$$

Process instances

We may also use parameters to identify particular instances of a generic process.

If we employ the local declaration syntax, then the two kinds of process parameter need never meet: instance (or static) parameters keep the same value for the lifetime of the process, and appear as part of the main, external process name.

Example: lift

$$
\begin{aligned}
GenericLift(max) \;=\; & \text{let} \\
& \quad LiftAtFloor(n) \;=\; \\
& \qquad n \leq max \;\&\; up \rightarrow LiftAtFloor(n+1) \\
& \qquad \square \\
& \qquad n \geq 1 \;\&\; down \rightarrow LiftAtFloor(n-1) \\
& \text{within} \\
& \quad LiftAtFloor(0)
\end{aligned}
$$

$$
GenericLift(9) \;=\; Lift
$$

Compound events

We use parameters in event names to indicate an event that involves a particular instance of a generic process, or to represent data being passed between processes in a parallel combination.

In either case, each parameter will appear after a single dot in the process name.

Example: car doors

$$\alpha SimpleDoor(i) = \{open.i, close.i, lock.i, unlock.i\}$$

$$\begin{aligned} SimpleDoor(i) = {} & \mathrm{let} \\ & \quad Open = close.i \rightarrow Closed \\ & \quad Closed = open.i \rightarrow Open \\ & \qquad\qquad \Box \\ & \qquad\qquad lock.i \rightarrow unlock.i \rightarrow Closed \\ & \mathrm{within} \\ & \quad Open \end{aligned}$$

The process $SimpleDoor(1)$ is capable of engaging in events from the set

$$\{open.1, close.1, lock.1, unlock.1\}$$

It can perform these events independently of the other three processes in the following parallel combination

$$CarDoors = \parallel k : 1 \mathrel{..} 4 \bullet SimpleDoor(k)$$

Channels

A set of parameterised events with the same basic name, but with a range of values for the parameters, is called a channel.

A single channel may be shared by several processes—all parties to the same transaction—with each process able to constrain the values of the various parameters of any event that occurs.

Example: meeting

$$meeting.london.15/03/2006.am$$

$$\begin{aligned} Steve &= \Box\, l : \{london\};\; d : Date;\; t : TimeOfDay \bullet \\ &\qquad meeting.l.d.t \rightarrow \ldots \end{aligned}$$

$$\begin{aligned} Fiona &= \Box\, l : Location;\; d : \{14/03/2006\};\; t : TimeOfDay \bullet \\ &\qquad meeting.l.d.t \rightarrow \ldots \end{aligned}$$

$$\begin{aligned} James &= \Box\, l : Location;\; d : \{14/03/2006, 16/03/2006\};\; t : \{am, lunch\} \\ &\qquad meeting.l.d.t \rightarrow \ldots \end{aligned}$$

$$
\begin{aligned}
&Steve \parallel Fiona \parallel James \\
&\quad = \Box\, l : \{london\};\ d : \{14/03/2006\};\ t : \{am, lunch\} \bullet \\
&\qquad meeting.l.d.t \ldots \\
&\quad = meeting.london.14/03/2006.am \rightarrow \ldots \\
&\qquad \Box \\
&\qquad meeting.london.14/03/2006.lunch \rightarrow \ldots
\end{aligned}
$$

? and !

$$c?(x : T) \to P \;=\; \Box\, x : T \bullet c.x \to P$$

If T describes all possible values of the parameter x, then we write $c?(x : T)$ simply as $c?x$.

If the range is a singleton set, containing expression e, then we write $c?(x : \{e\})$ simply as $c!e$.

Example: meeting

$$Steve = meeting!london?d?t \rightarrow \ldots$$

$$Fiona = meeting?l!14/03/2006?t \rightarrow \ldots$$

$$James = meeting?l?(d : \{14/03/2006, 16/03/2006\})?(t : \{am, lunch\})$$

Messages

If a channel is shared between two processes, it can be used to describe message-passing communication.

In most cases, one process will determine the value of any parameters, and the other will be ready to accept any combination.

A special case of the step law for parallel composition shows how this corresponds to values being passed.

Law

$$P = c!e \rightarrow P'$$

$$Q = c?x \rightarrow Q(x)$$

$$P \parallel Q \;=\; c!e \rightarrow (\, P' \parallel Q(e)\,)$$

Example: vending machines

$$
\begin{array}{l}
SVM(price) \ = \\
\quad \text{let} \\
\quad\quad Ready(holding) \ = \ insertCoin?value \rightarrow \text{if } holding + value < price \text{ th} \\
\quad\quad\quad\quad Ready(holding + value) \\
\quad\quad\quad \text{else} \\
\quad\quad\quad\quad Select(holding + value) \\
\quad\quad Select(holding) \ = \ insertCoin?value \rightarrow Select(holding + value) \\
\quad\quad\quad \square \\
\quad\quad\quad select?x \rightarrow \text{if } holding < 2 * price \text{ then} \\
\quad\quad\quad\quad dispense!x \rightarrow Ready(holding - price \\
\quad\quad\quad \text{else} \\
\quad\quad\quad\quad dispense!x \rightarrow Select(holding - price \\
\quad \text{within} \\
\quad\quad Ready(0)
\end{array}
$$

$$User \;=\; \sqcap\, c : \{20, 50\} \bullet \\ insertCoin!c \rightarrow select!orange \rightarrow dispense?x \rightarrow Stop$$

$$User \parallel SVM(50) \;=$$

$$insertCoin.20 \rightarrow Stop \\ \sqcap \\ insertCoin.50 \rightarrow select.orange \rightarrow dispense.orange \rightarrow Stop$$

Communication

A single channel is an adequate representation of message-passing communication only if the sending of a message is synonymous with its reception.

If it matters that a component may send a message without knowing that it has been received, or that some event may be performed by another component while the message is being passed, then we may need to use two channels, and an intervening process, to model the communication.

Media

A channel is not an adequate representation of a communication medium: there is no sense in which a value is being pushed into some medium and then carried subsequently to the other process.

A channel may be used in place of a perfectly reliable, synchronous communication medium, but other media should be represented as processes.

Convention

In describing communication media, we will often write

- *send* to denote a channel carrying data into a medium, and
- *receive* to denote a channel carrying data out

In Hoare (1985) and Roscoe (1998), the names *left* and *right* are used instead.

Channel naming convention

Example: lossy

$$
\begin{array}{l}
Lossy = \\
\quad send?x \rightarrow \\
\quad\quad (\ receive!x \rightarrow \\
\quad\quad\quad Lossy\ ) \\
\quad\quad \sqcap \\
\quad\quad Lossy
\end{array}
$$

Example: echoing

$$
\begin{aligned}
&Echoing = \\
&\quad send?x \rightarrow \\
&\qquad receive!x \rightarrow \\
&\qquad\quad receive!x \rightarrow \\
&\qquad\qquad Echoing
\end{aligned}
$$

Buffers

A buffer is a process that stores and forwards messages so that

- no messages are lost
- the order of messages is preserved

It should always be ready to accept input (unless it is already full) and ready to provide output (unless it is empty).

Example

A deterministic buffer of capacity 1:

$$
\begin{aligned}
&Copy = \\
&\quad send?x \rightarrow \\
&\quad\quad receive!x \rightarrow \\
&\quad\quad\quad Copy
\end{aligned}
$$

Example

A deterministic buffer of capacity N:

$$
\begin{array}{l}
Copy(N) = \\
\quad \text{let} \\
\qquad BufferState(s) = \\
\qquad\quad s \neq \langle\rangle \;\&\; receive!head(s) \rightarrow BufferState(tail(s)) \\
\qquad\quad \Box \\
\qquad\quad \#s \leq N - 1 \;\&\; send?x \rightarrow BufferState(s \frown \langle x \rangle) \\
\quad \text{within} \\
\qquad BufferState(\langle\rangle)
\end{array}
$$

Example

A nondeterministic buffer of capacity no greater than N.

$$
\begin{array}{l}
Buffer(N) = \\
\quad \text{let} \\
\qquad BufferState(s) = \\
\qquad\quad s \neq \langle\rangle \mathbin{\&} receive!head(s) \rightarrow BufferState(tail(s)) \\
\qquad\quad \Box \\
\qquad\quad s = \langle\rangle \mathbin{\&} send?x \rightarrow BufferState(s \frown \langle x \rangle) \\
\qquad\quad \Box \\
\qquad\quad (\, Stop \sqcap \#s \leq N - 1 \mathbin{\&} send?x \rightarrow BufferState(s \frown \langle x \rangle) \,) \\
\quad \text{within} \\
\qquad BufferState(\langle\rangle)
\end{array}
$$

Question

Which of these statements are true?

- $Buffer(N) \sqsubseteq Buffer(N-1)$
- $Buffer(N) \sqsubseteq Copy(N)$
- $Copy(N) \sqsubseteq Copy(N-1)$

Protocols

A protocol is a set of rules for collaboration.

If each party follows the rules that apply to them, then the collaboration will be successful.

In the case of a communication protocol, the outcome of a successful collaboration is a service that can be used to transfer data.

Describing protocols

In describing a protocol, we may use processes to represent:

- the protocol components: nodes, clients, servers, senders, or receivers;
- the communication media that link them;
- the service that the protocol is intended to provide.

Verifying protocols

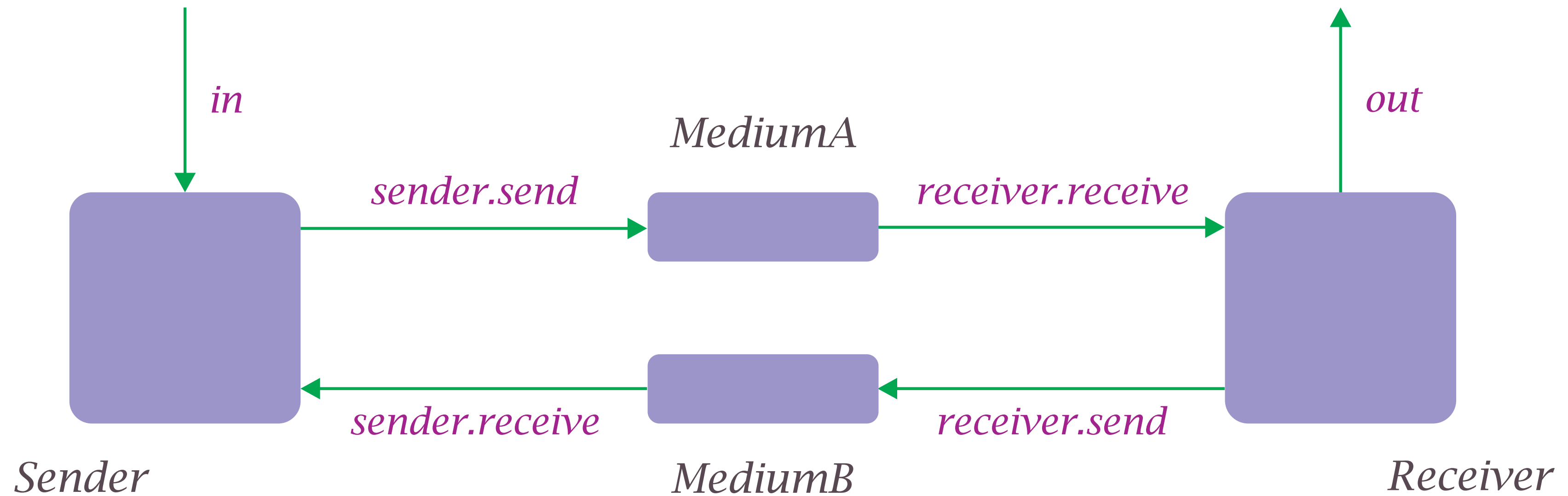
If *Internal* is the set of all events performed by the protocol that do not form part of the service provided, we wish to show that

$$Service \sqsubseteq (Protocol \parallel Media) \setminus Internal$$

Example: flow control

$$Protocol = Sender \parallel Receiver$$

$$Media = MediumA \parallel MediumB$$



Protocol

service

$$\alpha Sender = \bigcup \{ \ m : Message; \ a : Ack \bullet \\ \{ in.m, sender.send.m, sender.receive.a \} \ \}$$

$$\alpha Receiver = \bigcup \{ \ m : Message; \ a : Ack \bullet \\ \{ out.m, receiver.receive.m, receiver.send.a \} \ \}$$

```
Sender =
  let
    Empty =
      in?m →
        Send(m)

    Send(m) =
      sender.send!m →
        sender.receive?a →
          Empty
  within
    Empty
```

$$
\begin{array}{l}
Receiver = \\
\quad receiver.receive?m \rightarrow \\
\quad\quad out!m \rightarrow \\
\quad\quad\quad receiver.send!ack \rightarrow \\
\quad\quad\quad\quad Receiver
\end{array}
$$

For the purposes of this protocol, it is enough that *Ack* should consist only of the single acknowledgement value *ack*.

question

$$\begin{aligned}&MediumA = \\&\quad Copy[send \leftarrow sender.send, receive \leftarrow receiver.receive]\end{aligned}$$

that is,

$$\begin{aligned}&MediumA = \\&\quad sender.send?x \rightarrow \\&\qquad receiver.receive!x \rightarrow \\&\qquad\quad MediumA\end{aligned}$$

$$\mathit{MediumB} = \\ \quad \mathit{Copy}[\mathit{send} \leftarrow \mathit{receiver}.\mathit{send}, \mathit{receive} \leftarrow \mathit{sender}.\mathit{receive}]$$

that is,

$$\mathit{MediumB} = \\ \quad \mathit{receiver}.\mathit{send}?x \rightarrow \\ \quad\quad \mathit{sender}.\mathit{receive}!x \rightarrow \\ \quad\quad\quad \mathit{MediumB}$$

$$\begin{aligned} &Service = \\ &\quad Copy[send \leftarrow in, receive \leftarrow out] \end{aligned}$$

$$
\begin{aligned}
&Internal = \\
&\quad \bigcup \{\, m : Message;\ a : Ack \bullet \\
&\qquad\quad \{\, sender.send.m,\ receiver.receive.m, \\
&\qquad\qquad receiver.send.a,\ sender.receive.a \,\} \,\}
\end{aligned}
$$

Service

protocol

Question

For the flow control protocol, is it the case that

$$Service \sqsubseteq (Protocol \parallel Media) \setminus Internal \qquad ?$$

Question

What if *Receiver* were defined like this?

$$
\begin{aligned}
& Receiver = \\
& \quad receiver.receive?m \rightarrow \\
& \quad\quad receiver.send!ack \rightarrow \\
& \quad\quad\quad out!m \rightarrow \\
& \quad\quad\quad\quad Receiver
\end{aligned}
$$

example

Question

What if *MediumA* were defined like this?

$$MediumA =$$
$$Lossy[send \leftarrow sender.send, receive \leftarrow receiver.receive]$$

Summary

- Data
- Channels
- Messages
- Media

Index

2 Contents

3 Data

4 Process parameters

5 Local definitions

6 Example: lift

7 Conditionals

8 Example: Lift

9 Boolean guards

12 Example: lift

13 Process instances

14 Example: lift

15 Compound events
16 Example: car doors
18 Channels
19 Example: meeting
21 query and pling
22 Example: meeting
23 Messages
24 Law
25 Example: vending machines
28 Communication
29 Media
30 Convention
32 Example: lossy

33 Example: echoing
34 Buffers
35 Example
36 Example
37 Example
38 Question
39 Protocols
40 Describing protocols
41 Verifying protocols
42 Example: flow control
52 Question
53 Question
54 Question

55 Summary

56 Index