Postgraduate Course: Process Algebra

Communicating Sequential Process

Chapter 4 Communication

Huibiao Zhu

Software Engineering Institute, East China Normal University

Introduction

(1) A communication is an event that is described by a pair $c.v$ where c is the name of the channel on which the communication takes place and v is the value of the message which passes.

(2) The set of all messages which P can communicate on channel c is defined

$$\alpha c(P) = \{v | c.v \in \alpha P\}$$

(3) We also define functions which extract channel and message components of a communication

$$channel(c.v) = c, \qquad message(c.v) = v$$

Input and Output

1. Definition of Input and Output

(1) **Input**: Let v be any member of $\alpha c(P)$. A process which first outputs v on the channel c and then behaves like P is defined

$$(c!v \to P) = (c.v \to P)$$

The only event in which this process is initially prepared to engage is the communication event $c.v$.

(2) **Output**: A process which is initially prepared to input any value x communicable on the channel c, and then behave like $P(x)$, is defined

$$(c?x \rightarrow P(x)) = (y : \{y | channel(y) = c\} \rightarrow P(message(y)))$$

Example: Using the new definitions of input and output we can rewrite 1.1.3 X7:

$$COPYBIT = \mu X \bullet (in?x \rightarrow (out!x \rightarrow X))$$

where $\alpha in(COPYBIT) = \alpha out(COPYBIT) = \{0, 1\}$

2. Communications between Two Parallel Processes

(1) **Description**:

Let P and Q be processes, and let c be an output channel of P and an input channel of Q. When P and Q are composed concurrently in the system $(P||Q)$, communication will occur on channel c on each occasion that P outputs a message and Q simultaneously inputs that message.

(2) **Mechanism**:

An outputting process specifies a unique value for the message, whereas the inputting process is prepared to accept any communicable value. Thus the event that will actually occur is the communication $c.v$, where v is the value specified by the outputting process.

(3) Obvious Constraint: $\alpha c(P) = \alpha c(Q)$

abbreviation: αc for $\alpha c(P)$ (or $\alpha c(Q)$)

4. Examples

(1) A process which immediately copies every message it has input from the left by outputting it to the right

$$\alpha left(COPY) = \alpha right(COPY)$$

$$COPY = \mu X \bullet (left?x \to right!x \to X)$$

If $\alpha left = \{0, 1\}$, $COPY$ is almost identical to $COPYBIT$.

(2) A process like $COPY$, except that every number input is doubled before it is output

$$\alpha left = \alpha right = N$$

$$DOUBLE = \mu X \bullet (left?x \to right!(x + x) \to X)$$

(3) The value of a punched card is a sequence of eighty characters, which may be read as a single value along the left channel. A process which reads cards and outputs their characters one at a time

$$\alpha left = \{s | s \in \alpha right^* \wedge \# s = 80\}$$

$$UNPACK = P_{\langle\rangle}$$

where

$$P_{\langle\rangle} = left?s \rightarrow P_s$$

$$P_{\langle x\rangle} = right!x \rightarrow P_{\langle\rangle}$$

$$P_{\langle x\rangle \frown s} = right!x \rightarrow P_s$$

(4) A process which inputs characters one at a time from the left, and assembles them into lines of 125 characters length. Each completed line is output on the right as a single array-valued message

$$\alpha right = \{s | s \in \alpha left^* \wedge \#s = 125\}$$

$$PACK = P_{\langle\rangle}$$

where $P_s = right!s \rightarrow P_{\langle\rangle},$ if $\#s = 125$

$P_s = left?x \rightarrow P_{s \frown \langle x \rangle},$ if $\#s < 125$

(5) A process which copies from left to right, except that each pair of consecutive asterisks（连续的星号） is replaced by a single ↑.

$$\alpha left = \alpha right - \{" \uparrow "\}$$

$$SQUASH = \mu X \bullet left?x \rightarrow$$

```
    if $x \neq " * "$ $then$ $(right!x \rightarrow X)$
        else $left?y \rightarrow$ if $y = " * "$ then $(right!" \uparrow " \rightarrow X)$
                                   else $(right!" * " \rightarrow right!y \rightarrow X))$
```

Note: A process may be prepared initially to communicate on any one of a set of channels, leaving the choice between them to the other processes with which it is connected. For example,

$$(c?x \rightarrow P(x) | d?y \rightarrow Q(y))$$

(6) A process which accepts input on either of the two channels $left1$ or $left2$, and immediately outputs the message to the right

$$\alpha left1 = \alpha left2 = \alpha right$$

$$MERGE = (left1?x \rightarrow right!x \rightarrow MERGE$$
$$| left2?x \rightarrow right!x \rightarrow MERGE)$$

The output of this process is an interleaving（交叉） of the messages input from $left1$ and $left2$.

(7) A process that is always prepared to input a value on the left, or to output to the right a value which it has most recently input

$$\alpha left = \alpha right$$

$$VAR = left?x \rightarrow VAR_x$$

where, $VAR_x = (left?y \rightarrow VAR_y \mid right!x \rightarrow VAR_x)$

(8) A process which inputs from *up* and *left*, outputs to *down* a function of what it has input, before repeating

$$NODE(v)$$

$$= \mu X \bullet (up?sum \rightarrow left?prod \rightarrow down!(sum + v \times prod) \rightarrow X)$$

(9) A process which is at all times ready to input a message on the left, and to output on its right the first message which it has input but not yet output

$$BUFFER = P_{\langle\rangle}$$

where, $P_{\langle\rangle} = left?x \rightarrow P_{\langle x\rangle}$

$$\begin{aligned} P_{\langle x\rangle \frown s} = (&left?y \rightarrow P_{\langle x\rangle \frown s \frown \langle y\rangle} \\ &\mid right!x \rightarrow P_s) \end{aligned}$$

(10) A process which behaves like a stack of messages. When empty, it responds to the signal empty. At all times it is ready to input a new message from the left and put it on top of the stack; and whenever nonempty, it is prepared to output and remove the top element of the stack

$$STACK = P_{\langle\rangle}$$

where, $P_{\langle\rangle} = (empty \rightarrow P_{\langle\rangle} \mid left?x \rightarrow P_{\langle x\rangle})$

$$P_{\langle x\rangle \frown s} = (right!x \rightarrow P_s \mid left?y \rightarrow P_{\langle y\rangle \frown \langle x\rangle \frown s})$$

5. Specifications

(1) Some Notations:

If c is a channel name, we define (see Section 1.9.6)

$$tr \downarrow c = message^*(tr \upharpoonright \alpha c)$$

It is convenient just to omit the $tr \downarrow$, and write $right \leq left$ instead of $tr \downarrow right \leq tr \downarrow left$.

Another useful definition places a lower bound on the length of a prefix

$$s \leq^n t = (s \leq t \wedge \#t \leq \#s + N)$$

Properties:

(a) $s \leq^0 t \equiv (s = t)$

(b) $s \leq^n t \wedge t \leq^m u) \Rightarrow s \leq^{n+m} u$

(c) $s \leq t \equiv \exists n \bullet s \leq^n t$

(2) Examples

X1 $COPY$ **sat** $right \leq^1 left$

X2 $DOUBLE$ **sat** $right \leq^1 double^*(left)$

X3 $UNPACK$ **sat** $right \leq {}^\frown/left$

where, ${}^\frown/\langle s_0, s_1, ..., s_{n-1}\rangle = s_0{}^\frown s_1{}^\frown ... s_{n-1}$

The specification here states that the output on the right is obtained by flattening the sequence of sequences input on the left.

X4 $PACK$ **sat** $(({}^\frown/right \leq^{125} left) \wedge (\#^*right \in \{125\}^*))$

Communication

1. Laws

Thus output may be regarded as a specialized case of the prefix operator, and input a special case of choice; and this leads to the law

L1 $(c!v \rightarrow P) || (c?x \rightarrow Q(x)) = c!v \rightarrow (P || Q(v))$

If desired, such internal communications can be concealed by applying the concealment operator described in Section 3.5 outside the parallel composition of the two processes which communicate on the same channel, as shown by the law

L2 $((c!v \rightarrow P) || (c?x \rightarrow Q(x))) \setminus C = (P || Q(v)) \setminus C$

where $C = \{ c.v \mid v \in \alpha c \}$

2. Examples

We use the examples below to illustrate the communication sequence between two parallel branches

X1 Let $P = (left?x \rightarrow mid!(x \times x) \rightarrow P)$

$$Q = (mid?y \rightarrow right!(173 \times y) \rightarrow Q)$$

Clearly

$$P \ \mathbf{sat} \ (mid \leq^1 square^*(left))$$

$$Q \ \mathbf{sat} \ (right \leq^1 173 \times mid)$$

where $(173 \times mid)$ multiples each message of *mid* by 173. It follows that

$$(P||Q) \ \mathbf{sat} \ (right \leq^1 173 \times mid) \wedge (mid \leq^1 square^*(left))$$

The specification here implies

$$right \leq 173 \times square^*(left)$$

which was presumably the original intention.

Pipes

1. Definition of Pipes

(a) Processes: with only two channels, namely an input channel $left$ and an output channel $right$

(b) Connection: The processes P and Q may be joined together so that the right channel of P is connected to the left channel of Q, and the sequence of messages output by P and input by Q on this internal channel is concealed from their common environment. The result of the connection is denoted

$$P >> Q$$

(3) constraints:

$$\alpha(P >> Q) = \alpha left(P) \cup \alpha right(Q)$$

$$\alpha right(P) = \alpha left(Q)$$

2. Examples:

X1 A pipe which outputs each input value multiplied by four (4.2 X2)

$$QUADRUPLE = DOUBLE >> DOUBLE$$

X2 A process which inputs cards of eighty characters and outputs their text, tightly packed into lines of 125 characters each (4.2 X3, X4)

$$UNPACK >> PACK$$

X3 Same as X2, except that each pair of consecutive asterisks is replaced by "↑" (4.2 X5)

$$UNPACK >> SQUASH >> PACK$$

X4 Same as X2, except that the reading of cards may continue when the printer is held up, and the printing can continue when the card reader is held up (4.2 X9)

$$UNPACK >> BUFFER >> PACK$$

X5 In order to avoid undesirable expansion of buffers, it is usual to limit the number of messages buffered. Even the single buffer provided by the COPY process (4.2 X1) may be adequate. Here is a version of X4 which reads one card ahead on input and buffers one line on output

$$COPY >> UNPACK >> PACK >> COPY$$

X6 A double buffer, which accepts up to two messages before requiring output of the first

$$COPY >> COPY$$

Its behaviour is similar to that of $CHAIN2$ (2.6 X4) and VMS2 (1.1.3 X6).

3. Laws

L1 $P >> (Q >> R) = (P >> Q) >> R$

L2 $(right!v \rightarrow P) >> (left?y \rightarrow Q(y)) = P >> Q(v)$

L3 $(right!v \rightarrow P) >> (right!w \rightarrow Q) = right!w \rightarrow ((right!v \rightarrow P) >> Q)$

L4 $(left?x \rightarrow P(x)) >> (left?y \rightarrow Q(y)) =$
$left?x \rightarrow (P(x) >> (left?y \rightarrow Q(y)))$

Note:

There are several other similar laws. Please see page 134-135.

4. Livelock

(a) Bad cases (Divergent behavious): Let

$$P = (right!1 \rightarrow P)$$

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

$(P >> Q)$ is obviously a useless process; it is even worse than $STOP$, in that like an endless loop it may consume unbounded computing resources without achieving anything.

Another example: Let

$$P = (right!1 \rightarrow P \mid left?x \rightarrow P1(x))$$

$$Q = (left?x \rightarrow Q \mid right!1 \rightarrow Q1)$$

In this example, divergence derives from the mere possibility of infinite internal communication.

(b) avoiding livelock:

A simple method to prove $(P >> Q)$ is free of livelock is to show that P is *left-guarded* in the sense that it can never output an infinite series of messages to the right without interspersing inputs from the left.

$$P \text{ is left-guarded} \equiv \exists f \bullet P \ \mathbf{sat} \ (\#right \leq f(left))$$

Properties:

L1 If every recursion used in the definition of P is guarded by an input from the left, then P is left-guarded.

L2 If P is left-guarded then $(P >> Q)$ is free of livelock.

L3 If Q is right-guarded then $(P >> Q)$ is free of livelock.