Postgraduate Course: Process Algebra

Communicating Sequential Process

Chapter 5 Sequential Processes

Huibiao Zhu

Software Engineering Institute, East China Normal University

Introduction

1. Motivation

(1) Process $STOP$ is defined as one that never engages in any action (not useful, and probably results from a deadlock or other design error)

(2) How to say a process terminating successfully?

Introducing a special event $\surd$ to stand for the termination event

$(x : B \to P(x))$ is invalid if $\surd \in B$

(3) $SKIP_A$ is defined as a process which does nothing but terminate successfully

$$\alpha SKIP_A = A \cup \{\surd\}$$

Example: A vending machine that is intended to serve only one customer with chocolate or toffee and then terminate successfully

$$VMONE = (coin \to (choc \to SKIP \mid toffee \to SKIP))$$

2. Sequential Composition

If P and Q are sequential processes with the same alphabet, their sequential composition

$$P \; ; \; Q$$

is a process which first behaves like P; but when P terminates successfully, $(P;Q)$ continues by behaving as Q. If P never terminates successfully, neither does $(P;Q)$.

Example: A vending machine designed to serve exactly two customers, one after the other

$$VMTWO = VMONE \; ; \; VMONE$$

Note: A process which repeats similar actions as often as required is known as a loop; it can be defined as a special case of recursion

$$\begin{aligned} {}^{*}P &= \mu X \bullet (P;X) \\ &= P;P;P;... \\ \alpha({}^{*}P) &= \alpha P - \{\surd\} \end{aligned}$$

3. Examples

(1) A vending machine designed to serve any number of customers

$$VMCT =^* VMONE$$

This is identical to VMCT (1.1.3 X3).

(2) A sentence of Pidgingol consists of a noun clause followed by a predicate. A predicate is a verb followed by a noun clause. A verb is either bites or scratches. The definition of a noun clause is given more formally below

$$\begin{aligned}
\alpha PIDGINGOL &= \{a, the, cat, dog, bites, scratches\} \\
PIDGINGOL &= NOUNCLAUSE; PREDICATE \\
PREDICATE &= VERB; NOUNCLAUSE \\
VERB &= (bites \rightarrow SKIP \mid scratches \rightarrow SKIP) \\
NOUNCLAUSE &= ARTICLE; NOUN \\
ARTICLE &= (a \rightarrow SKIP \mid the \rightarrow SKIP) \\
NOUN &= (cat \rightarrow SKIP \mid dog \rightarrow SKIP)
\end{aligned}$$

(2) A noun clause which may contain any number of adjectives furry or prize

$$NOUNCLAUSE = ARTICLE \ ; \ \mu X \bullet (furry \rightarrow X \mid prize \rightarrow X \\ \mid cat \rightarrow SKIP \mid dog \rightarrow SKIP)$$

(3) A process which accepts any number of as followed by a b and then the same number of cs, after which it terminates successfully

$$A^n BC^n = \mu X \bullet (b \rightarrow SKIP | a \rightarrow (X; (c \rightarrow SKIP)))$$

(4) A process which first behaves like $A^n BC^n$, but the accepts a d followed by the same number of es

$$A^n BC^n DE^n = ((A^n BC^n); d \rightarrow SKIP)||C^n DE^n$$

where $C^n DE^n = f(A^n BC^n)$ for f which maps a to c, b to d, and c to e.

(5) A process which accepts any interleaving of *downs* and *ups*, except that it terminates successfully on the first occasion that the number of *downs* exceeds the number of *ups*

$$POS = (down \rightarrow SKIP \mid up \rightarrow (POS\ ;\ POS))$$

(6) The process C_0 behaves like CT_0 (1.1.4 X2)

$$C_0 = (around \rightarrow C_0 \mid up \rightarrow C_1)$$

$$C_{n+1} = POS\ ;\ C_n$$

(7) A USER process manipulates two count variables named l and m (see 2.6.2 X3)

$$l : CT_0 || m : CT_3 || USER$$

The following subprocess (inside the USER) adds the current value of l to m

$$\begin{aligned} ADD = (&l.around \rightarrow SKIP \\ &\mid l.down \rightarrow (ADD; (m.up \rightarrow l.up \rightarrow SKIP))) \end{aligned}$$

Laws

L1 $SKIP\ ;\ P = P\ ;\ SKIP = P$

L2 $(P; Q); R = P; (Q; R)$

L3 $(x : B \rightarrow P(x)); Q = (x : B \rightarrow (P(x); Q))$

L4 $(a \rightarrow P); Q = a \rightarrow (P; Q)$

L5 $STOP\ ;\ Q = STOP$

L6 $SKIP_A || SKIP_B = SKIP_{A \cup B}$

L7 $((x : B \rightarrow P(x)) || SKIP_A) = (x : (B - A) \rightarrow (P(x) || SKIP_A))$

L8 $STOP_A || SKIP_B = SKIP_B$ if $\surd \notin A \ \wedge\ A \subseteq B$

Note: The laws L1 to L3 may be used to prove the claim made in 5.1 X9 that C_0 behaves like CT_0 (1.1.4 X2).

Solution: We can proceed the proof mathematical induction.

(1) The equation for CT_0 is the same as that for C_0.

$$C_0 = (around \rightarrow C_0 \mid up \rightarrow C_1)$$

(2) For $n > 0$, we need to prove $\quad C_n = (up \rightarrow C_{n+1} \mid down \rightarrow C_{n-1})$

$$
\begin{array}{cll}
 & LHS & \\
= & POS; C_{n-1} & \{\text{def of } C_n\} \\
= & (down \rightarrow SKIP | up \rightarrow POS; POS); C_{n-1} & \{\text{def of } POS\} \\
= & (down \rightarrow (SKIP; C_{n-1}) \mid up \rightarrow (POS; POS); C_{n-1}) & \{L3\} \\
= & (down \rightarrow C_{n-1} | up \rightarrow POS; (POS; C_{n-1})) & \{L1, L2\} \\
= & (down \rightarrow C_{n-1} | up \rightarrow POS; C_n) & \{\text{def } C_n\} \\
= & RHS & \{\text{def } C_n\}
\end{array}
$$

Mathematical Treatment

1. Deterministic Processes

L0 $traces(SKIP) = \{\langle\rangle, \langle\surd\rangle\}$

Natation: $(s\ ;\ t) = s$

$$(s \frown \langle\surd\rangle\ ;\ t) = s \frown t$$

L1 $traces(P\ ;\ Q) = \{s\ ;\ t \mid s \in traces(P) \wedge t \in traces(Q)\}$

An equivalent definition is

L1A $traces(P\ ;\ Q) = \{s \mid s \in traces(P) \wedge \neg\langle\surd\rangle\ \mathbf{in}\ s\}\ \cup$

$$\{s \frown t \mid s \frown \langle\surd\rangle \in traces(P) \wedge t \in traces(Q)\}$$

L2 $P/s = SKIP$ if $s \frown \langle\surd\rangle \in traces(P)$

Law **L2** is not correct for nondeterministic process. It can be weaken to the law below.

L2A $s^\frown\langle\surd\rangle \in traces(P) \Rightarrow (P/s) \sqsubseteq SKIP$

In addition to the laws given earlier in this chapter, sequential composition of nondeterministic processes satisfies the following laws.

L1 $CHAOS\ ;\ P = CHAOS$

L2A $(P \sqcap Q)\ ;\ R = (P\ ;\ R) \sqcap (Q\ ;\ R)$

L2B $R\ ;\ (P \sqcap Q) = (R\ ;\ P) \sqcap (R\ ;\ Q)$

D1 $refusals(P\ ;\ Q) = \{X \mid (X \cup \{\surd\}) \in refusals(P)\} \cup$
$$\{X \mid \langle\surd\rangle \in traces(P) \wedge X \in refusals(Q)\}$$

D2 $divergences(P\ ;\ Q)$
$$= \{s \mid s \in divergences(P) \wedge \neg\langle\surd\rangle \textbf{ in } s\} \ \cup$$
$$\{s^\frown t \mid s^\frown\langle\surd\rangle \in traces(P) \wedge \neg\langle\surd\rangle \textbf{ in } s \wedge t \in divergences(Q)\}$$

D3 $failures(P\ ;\ Q)$
$$= \ \{(s, X) \mid (s, X \cup \{\surd\}) \in failures(P)\} \ \cup$$
$$\{(s^\frown t, X) \mid s^\frown\langle\surd\rangle \in traces(P) \wedge (t, X) \in failures(Q)\} \ \cup$$
$$\{(s, X) \mid s \in divergences(P;Q)\}$$

Interrupts

1. Definition

$(P \triangle Q)$: It does not depend on successful termination of P. Instead, the progress of P is just interrupted on occurrence of the first event of Q; and P is never resumed.

$$\alpha(P \triangle Q) = \alpha P \cup \alpha Q$$

$$traces(P \triangle Q) = \{s^\frown t \mid s \in traces(P) \wedge t \in traces(Q)\}$$

To avoid problems, we specify that $\surd$ must not be in αP.

2. Laws

The law below states that it is the environment which determines when Q shall start, by selecting an event which is initially offered by Q but not by P

L1 $(x : B \to P(x)) \triangle Q = Q \square (x : B \to (P(x) \triangle Q))$

If $(P \triangle Q)$ can be interrupted by R, this is the same as P interruptible by $(Q \triangle R)$

L2 $(P \triangle Q) \triangle R = P \triangle (Q \triangle R)$

L3 $P \triangle STOP = P = STOP \triangle P$

L4A $P \triangle (Q \sqcap R) = (P \triangle Q) \sqcap (P \triangle R)$

L4B $(Q \sqcap R) \triangle P = (Q \triangle P) \sqcap (R \triangle P)$

L5 $CHAOS \triangle P = CHAOS = P \triangle CHAOS$

3. Catastrophe

Let $\curvearrowright$ be a symbol standing for a catastrophic interrupt event, which it is reasonable to suppose would not be caused by P; more formally

$$\curvearrowright \notin \alpha P$$

$$P \hat{\curvearrowright} Q = P \triangle (\curvearrowright \rightarrow Q)$$

The first law is just an obvious formulation of the informal description of the operator

L1 $\quad (P \hat{\curvearrowright} Q)/(s ^\frown \langle \curvearrowright \rangle) = Q \qquad\qquad$ for $s \in traces(P)$

The second law gives a more explicit description of the first and subsequent steps of the process. It shows how $\hat{\curvearrowright}$ distributes back through $\rightarrow$

L2 $\quad (x : B \rightarrow P(x)) \hat{\curvearrowright} Q = (x : B \rightarrow (P(x) \hat{\curvearrowright} Q) \mid \curvearrowright \rightarrow Q)$

4. Restart

One possible response to catastrophe is to restart the original process again. Let P be a process such that $\curvearrowright \notin \alpha P$. We specify $\hat{P}$ as a process which behaves as P until $\curvearrowright$ occurs, and after each $\curvearrowright$ behaves like P from the start again.

$$\begin{aligned} \alpha \hat{P} &= \alpha P \cup \{\curvearrowright\} \\ \hat{P} &= \mu X \bullet (P \hat{\curvearrowright} X) \\ &= P \hat{\curvearrowright} (P \hat{\curvearrowright} (P \hat{\curvearrowright} \ldots)) \end{aligned}$$

The informal definition of $\hat{P}$ is expressed by the law

L1 $\quad \hat{P} / s \frown \langle \curvearrowright \rangle = \hat{P} \qquad$ for $s \in traces(P)$

Assignment

1. Definition

If x is a program variable and e is an expression and P a process

$$(x := e; P)$$

is a process which behaves like P, except that the initial value of x is defined to be the initial value of the expression e. Initial values of all other variables are unchanged.

Assignment by itself can be given a meaning by the definition

$$(x := e) = (x := e\ ;\ SKIP)$$

Note:

Single assignment generalizes easily to multiple assignment. Let x stand for a list of distinct variables

$$x = x_0, x_1, ...x_{n-1}$$

Let e stand for a list of expressions

$$e = e_0, e_1, ...e_{n-1}$$

Provided that the lengths of the two lists are the same

$$x := e$$

assigns the initial value of e_i to x_i , for all i.

Let b be an expression that evaluates to a Boolean truth value (either true or false). If P and Q are processes $P \lhd b \rhd Q$ (P **if** b **else** Q) is a process which behaves like P if the initial value of b is true, or like Q if the initial value of b is false.

The traditional loop **while** b **do**Q will be written $b * Q$

This may be defined by recursion

D1 $\quad b * Q = \mu X \bullet ((Q; X) \lhd b \rhd SKIP)$

Example: A process that behaves like CT_n (1.1.4 X2)

$$
\begin{aligned}
X1 = \mu X \bullet (&around \rightarrow X \mid up \rightarrow (n := 1; X)) \\
&\lhd n = 0 \rhd \\
&(up \rightarrow (n := n + 1; X) \mid down \rightarrow (n := n - 1; X))
\end{aligned}
$$

The current value of the count is recorded in the variable n.

2. Laws

L1 $(x := x) = SKIP$

L2 $(x := e; x := f(x)) = (x := f(e))$

L3 If x, y is a list of distinct variables

$$(x := e) = (x, y := e, y)$$

L4 If x, y, z are of the same length as e, f , g respectively

$$(x, y, z := e, f, g) = (x, z, y := e, g, f)$$

L5-L6 $\lhd b \rhd$ is idempotent, associative, and distributes through $\lhd c \rhd$

L7 $P \lhd true \rhd Q = P$

L8 $P \lhd false \rhd Q = Q$

L9 $P \lhd \neg b \rhd Q = Q \lhd b \rhd P$

L10 $P \lhd b \rhd (Q \lhd b \rhd R) = P \lhd b \rhd R$

L11 $P \lhd (a \lhd b \rhd c) \rhd Q = (P \lhd a \rhd Q) \lhd b \rhd (P \lhd c \rhd Q)$

L12 $x := e \; ; \; (P \lhd b(x) \rhd Q) = (x := e \; ; \; P) \lhd b(e) \rhd (x := e; Q)$

L13 $(P \lhd b \rhd Q); R = (P; R) \lhd b \rhd (Q; R)$

How to deal effectively with assignment in concurrent processes?

Restriction: No variable assigned in one concurrent process can ever be used in another.

$var(P)$— the set of variables that may be assigned within P

$acc(P)$— the set of variables that may be accessed in expressions within P

All variables which may be changed may also be accessed

$var(P) \subseteq acc(P) \subseteq \alpha P$

P and Q are to be joined by $||$:

$$var(P) \cap acc(Q) = var(Q) \cap acc(P) = \{\}$$

L14 $((x := e; P)||Q) = (x := e; (P||Q))$

provided that $x \subseteq var(P) - acc(Q)$ and $acc(e) \cap var(Q) = \{\}$

An immediate consequence of this is

$$(x := e; P)||(y := f; Q) = (x, y := e, f; (P||Q))$$

provided that $x \in var(P) - acc(Q) - acc(f)$

and $y \in var(Q) - acc(P) - acc(e)$

Concurrent combination distributes through the conditional

L15 $P||(Q \lhd b \rhd R) = (P||Q) \lhd b \rhd (P||R)$

provided that $acc(b) \cap var(P) = \{\}$.