Introduction to Communicating Sequential Process (CSP) (Lecture 8)

Mannheim, September 2007

Contents

- Sequential Composition
- Semantics

Termination

Forms of unsuccessful termination resulting from design flaws are

- *Stop*, representing deadlock
- *Div* representing livelock.

By comparison process *Skip* represents deliberate successful termination on completion of a task.

A terminating trace of process P is a trace t after which P may terminate

$$P \ \mathit{after} \ t \sqsubseteq \mathit{Skip}.$$

Sequential Composition

If P and Q are processes over then

$$P \; ; \; Q$$

denotes their *sequential composition* which first behaves like P; if P terminates it then behaves like Q; if P doesn't terminate neither does $P;Q$.

The *iteration*, P^*, of P is defined $P^* = P \; ; \; P^*$.

Sequential Composition: Example

A vending machine which serves one customer is

$V1 = coin \rightarrow (choc \rightarrow Skip \mid toffee \rightarrow Skip).$

One which serves two is

$$V1 \; ; \; V1.$$

And one which serves customers forever is

$$V = V1^*.$$

Sequential Composition: Example

Recall the infinite mutual recursion

$R = R_0 = (\mathrm{around} \rightarrow R \mid \mathrm{up} \rightarrow R_1)$
$R_{n+1} = (\mathrm{up} \rightarrow R_{n+2} \mid \mathrm{down} \rightarrow R_n)$.

That process is expressed in finite form using sequential composition

$Z = (around \rightarrow Z \mid up \rightarrow P \,;\, Z)$
$P = (up \rightarrow P \,;\, P \mid down \rightarrow Skip)$.

Sequential Composition: Example

The language consisting of strings having any number of *a*'s, followed by a *b*, followed by the same number of *c*'s as *a*'s is

$$\{<a>^n \;\hat{}\; <b>\hat{}<c>^n \mid n \;\square\; N\}.$$

A process for that language is

$$L = \mu X \bullet (b \rightarrow Skip$$
$$\mid a \rightarrow (X\,;\, c \rightarrow Skip)).$$

Sequential Composition: Example

The language whose strings start as above and are then followed by a *d* and then the same number of *e*'s as *a*'s is

$$\{<a>^n \wedge <b>^\wedge<c>^n \wedge <d>^\wedge<e>^n \mid n \in N\}.$$

A process for that language is

$$M = (L;d \rightarrow Skip)\ [|\{c,d\}|]\ f\ L,$$

where the injective relabelling f is defined

$f\,a = c,\ f\,b = d,\ f\,c = e.$

Sequential Composition: Laws

Sequential composition is associative and distributive in each argument, with unit Skip

- $(P \; ; \; Q) \; ; \; R = P \; ; \; (Q \; ; \; R)$
- $(P \sqcap Q) \; ; \; R = (P \; ; \; R) \sqcap (Q \; ; \; R)$
- $P \; ; \; (Q \sqcap R) = (P \; ; \; Q) \sqcap (P \; ; \; R)$
- $Skip \; ; \; P = P = P \; ; \; Skip$

Stop is a left zero, as is any divergent process

- $Stop \; ; \; P = Stop$
- $Div \; ; \; P = Div$ ...

Sequential Composition: Laws

- Processes do not share their local variables. Thus in $P \,;\, Q$ the final state of P is independent of the initial state of Q.

  For example in the sequential composition

  $$(\ldots \rightarrow out!x \rightarrow Skip) \,;\, (in?x \rightarrow \ldots)$$

  the value of x in the first process has no relationship to the value of x in the second.

Sequential Composition: Laws

For example

$$in?x \longrightarrow out!x \longrightarrow Skip$$

$$\neq$$

$$(in?x \longrightarrow Skip)\ ;\ (out!x \longrightarrow Skip).$$

Indeed the latter process may output any value of the appropriate type on channel out whilst the former can output only the value it has input on in.

Sequential Composition: Laws

- However, provided a variable x is not free in process Q

  $(?x:A \text{ -> } P(x));Q = ?x:A \text{ -> } (P(x);Q)$

Sequential Composition: Traces

The event of successful termination is represented by $\surd$, an event not in any Σ. It occurs only as the last event of a terminating process and is not available like other elements of for synchronisation, nor can it be hidden or renamed.

$$\textit{traces Skip} = \{<>, <\surd>\}.$$

Write

$$\Sigma_{\surd} = \Sigma \; \square \; \{\surd\}$$

$$\Sigma_{\surd}^{*} = \Sigma^{*} \; \square \; \{t \,\hat{}\, <\surd> \mid t \; \square \; \Sigma^{*}\}$$

Sequential Composition: Traces

- The traces of *P; Q* consist of those of *P* or those terminating traces of *P* with √ removed and catenated with a trace of *Q*

$$traces(P\ ;\ Q) = traces\ P \cup \{s \mathbin{^\wedge} t \mid (s \mathbin{^\wedge} \langle \surd \rangle \in traces\ P \text{ and } t \in traces\ Q\}.$$

Sequential Composition: Traces

- The traces of *P; Q* consist of those of *P* or those terminating traces of *P* with √ removed and catenated with a trace of *Q*

$$traces(P\,;\,Q) = traces\ P \cup \{s \,\hat{}\, t \mid (s \,\hat{}\, \langle \surd \rangle \in traces\ P \text{ and } t \in traces\ Q\}.$$

Sequential Composition: Traces

- The traces of *P; Q* consist of those of *P* or those terminating traces of *P* with √ removed and catenated with a trace of *Q*

$$traces(P\ ;\ Q) = traces\ P \cup \{s \mathbin{\hat{}} t \mid (s \mathbin{\hat{}} \langle \surd \rangle \in traces\ P \text{ and } t \in traces\ Q\}.$$

Sequential Composition: Traces

- The traces of $P; Q$ consist of those of P or those terminating traces of P with √ removed and catenated with a trace of Q

$$traces(P\ ;\ Q) = traces\ P\ \square \{s\ \hat{}\ t\ |(s\ \hat{}\ <\surd>\ \square\ traces\ P\ and\ \ t\ \square\ traces\ Q\}.$$

Assignment

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

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

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

Assignment: Examples

- A process that behaves like Rocket

$$
\begin{aligned}
X1 = \quad & \mu X.\ (around \text{ -> } X \mid up \text{ -> } (n{:=}1; X)) \\
& \qquad \lhd n=0 \rhd \\
& \qquad (up \text{ -> } (n{:=}n+1; X) \mid down \text{ -> } (n{:=}n-1; X))
\end{aligned}
$$

Assignment: Examples

- A process which divides a natural number x by a positive number y, assigning the quotient to q and the remainder to r

$$QUOT = (q := x \div y;\ r := x - q \,.\, y)$$

Assignment: Laws

- $(x := x) = SKIP$
- $(x := e;\ x := f(x)) = (x := f(e))$
- *If* x, y *is a list of distinct variables* $(x := e) = (x, y := e, y)$
- *If* x, y, z *are of the same length as* e, f, g *respectively*

  $$(x, y, z := e, f, g) = (x, z, y := e, g, f)$$
- $x := e\ ;\ (P \lhd b(x) \rhd Q) = (x := e; P) \lhd b(x) \rhd (x := e; Q)$
- $((x := e; P) || Q) = (x := e\ ;\ (P || Q))$ *provided that* P *and* Q *are data independent…*

Assignment: Laws

- $(x{:=}x) = SKIP$
- $(x{:=}e;\ x{:=}f(x)) = (x{:=}f(e))$
- *If* x, y *is a list of distinct variables* $(x{:=}e) = (x,y := e,y)$
- *If* x,y,z *are of the same length as* e,f,g *respectively*

  $$(x,y,z := e,f,g) = (x,z,y := e,g,f)$$
- $x{:=}e\ ;\ (P \lhd b(x) \rhd Q) = (x{:=}e;P) \lhd b(x) \rhd (x{:=}e;Q)$
- $((x{:=}e;P)||Q) = (x{:=}e\ ;\ (P||Q))$ *provided that* P *and* Q *are data independent…*

Semantics

Traces do not distinguish internal and external choice

$$traces(P \sqcap Q) = traces(P[]Q).$$

How do those processes differ?

- Since $a \rightarrow A[]b \rightarrow B$ offers its environment the choice between a and b the environment cannot refuse either; whichever of them is offered by the environment must be performed.
- Since $a \rightarrow A \sqcap b \rightarrow B$ permits its environment no say in which of the two processes occurs, it may refuse either a or b but not both; whichever of them is offered by the environment, deadlock may occur.

Semantics: Refusals

If P is a (nonsequential) process its *refusals*, *refusals P*, are those subsets E of the universe which it may (initially) refuse to perform; if the environment offers a general choice from E, deadlock may occur.

For example over universe $\{a, b\}$,

$refusals(a \rightarrow A [] b \rightarrow B) = \{\{\,\}\}$

$refusals(a \rightarrow A \sqcap b \rightarrow B) = \{\{\,\}, \{b\}, \{a\}\}.$

Refusals thus distinguish internal and external choice.

Semantics: Refusals

Observe

$$refusals(a \longrightarrow A) = \{\{\,\}, \{b\}\}$$

$$refusals(b \longrightarrow B) = \{\{\,\}, \{a\}\}.$$

Thus from that example, and in general,

$$refusals(P[\,]Q) = refusals(P) \cap refusals(Q)$$

$$refusals(P \sqcap Q) = refusals(P) \cup refusals(Q).$$

Semantics: Failures

- If *P* is a (nonsequential) process its *failures*, *failures P*, consists of those pairs *(t ,E)* for which *t* is a trace of *P* and *E* is a refusal of *P* after *t* . Thus after it has engaged in trace *t* the process may refuse *E*.
- For example over universe $\Sigma = \{a, b\}$:

  $$failures\ Stop = \{(<>, \{\ \}), (<>, \{a\}), (<>, \{b\}), (<>, \{a, b\})\}$$
  $$= \{(<>, E) \mid E \subseteq \Sigma\}$$
- The traces of a process can be reclaimed from its failures

  $$traces\ P = \{t : \Sigma^* \mid (t, \{\ \}) \in failures\ P\}.$$

Semantics: Failures

- $failures(a \rightarrow Stop) = \{(<>, \{\ \}), (<>, \{b\}), (<a>, \{\ \}), (<a>, \{a\}), (<a>, \{b\}), (<a>, \{a, b\})\} = \{(<>,E) \mid a \notin E \subseteq \Sigma\} \cup \{(<a>,E) \mid E \subseteq \Sigma\}$

- $failures(b \rightarrow Stop) = \{(<>, \{\ \}), (<>, \{a\}), (<b>, \{\ \}), (<b>, \{a\}),(<b>, \{b\}), (<b>, \{a, b\})\}= \{(<>,E) \mid b \notin E \subseteq \Sigma\} \cup \{(<b>,E) \mid E \subseteq \Sigma\}$

Semantics: Failures

$failures(a \rightarrow Stop[]b \rightarrow Stop)$

$=$

$\{(<>, \{\ \}), (<a>, \{\ \}), (<a>, \{a\}), (<a>, \{b\}), (<a>, \{a, b\}),$
$(<b>, \{\ \}), (<b>, \{a\}), (<b>, \{b\}), (<b>, \{a, b\})\}$

$=$

$\{(<>, \{\ \})\}$

$\cup$

$\{(<a>,E) \mid E \subseteq \Sigma \}$

$\cup$

$\{(<b>,E) \mid E \subseteq \Sigma \}$

Semantics: Failures

$$failures(a \rightarrow Stop \sqcap b \rightarrow Stop)$$

$$=$$

$$\{(<>, \{\ \}), (<>,\{a\}), (<>, \{b\}), (<a>, \{\ \}), (<a>, \{a\}), (<a>, \{b\}), (<a>, \{a, b\}),(<b>, \{\ \}), (<b>, \{a\}), (<b>, \{b\}), (<b>, \{a, b\})\}$$

$$=$$

$$\{(<>, \{\ \}), (<>,\{a\}), (<>, \{b\})\}$$

$$\square$$

$$\{(<a>,E) \mid E \ \square \Sigma \}$$

$$\square$$

$$\{(<b>,E) \mid E \ \square \Sigma \}$$

With failures we can distinguish internal from external choice.

Semantics: Failures

Failures refinement ordering

$$F \sqsubseteq_F G \equiv F \;\square\; G.$$

Informally, every trace of G is a trace of F and if G deadlocks then F deadlocks; thus both the trace behaviour and the deadlock behaviour of G conform to that of F.

Note: restricted to traces, $\sqsubseteq_F$ yields refinement $\sqsubseteq_T$ in the traces model:

$$F \sqsubseteq_F G \textit{ implies } F \sqsubseteq_T G.$$

Semantics: Failures

The failures model is finer than the traces model (it distinguishes $\sqcap$ from []) but is still not fully abstract for CSP (it doesn't distinguish *Div* from *Stop*).

Semantics: Divergences

Failures do not distinguish deadlock and divergence

$failures\ Stop = failures\ Div = \{(\langle\rangle, E) \mid E \subseteq \Sigma\}$.

How do those two processes differ?

- Stop performs no events, deadlocks immediately and does not diverge
- Div performs no events but diverges immediately.

Semantics: Divergences

For process P the divergences of P are the traces after which it diverges

$$divergences\ P = \{t : traces\ P \mid P\ after\ t = Div\}.$$

For example over universe $\{a, b\}$,

$divergences\ Stop = \{\ \}$

$divergences\ Div = ?$

Recall that Div is minimal since $Div \sqcap P = Div$. Similarly from any point in the evolution of a process, divergent behaviour is indistinguishable from arbitrary behaviour.

Semantics: Divergences

Thus after diverging a process behaves like the least element: any trace is a divergence and any subset a refusal.

Hence, because <> □ *divergences Div*,

divergences Div = Σ^*

failures Div = $\Sigma^* \times$ |*P* Σ.

Divergences thus distinguish *Stop* and *Div*.

The healthiness conditions for divergences are

- *if t* □ *divergences P and u* □ Σ^* *then t ^ u* □ *divergences P*
- *if t* □ *divergences P then for all E* □ Σ, *(t ,E)* □ *failures P.*

Semantics: Failures & Divergences

- For finite universe the failures & divergences model of processes over Σ consists of the set N of pairs $(F,D) : F \times \Sigma^*$
- satisfying
  - $t \square D \square u \square \Sigma^* \square t \,\hat{}\, u \square D$
  - $t \square D \square (t, E) \square F$.

Semantics: Failures & Divergences

The space is partially ordered by the failures & divergences refinement ordering

$$(F,D) \sqsubseteq_N (G,E) \equiv F \;\square\; G \;\square\; D \;\square\; E.$$

Thus both the failures behaviour and the divergences behaviour of *(G,E)* conform to that of *(F,D)*.

Semantics: Failures & Divergences

Home exercise: Study the failures and divergence semantics of the constructs of CSP.