Introduction to Communicating Sequential Process (CSP) (Lecture 7)

Mannheim, September 2007

Contents

- Abstraction
  - Hiding
  - Divergence
- Communication

Hiding

- If P is a process over Σ and if C is a finite subset of Σ then

  $$P \setminus C,$$

  pronounced P hide C, is the process which behaves like P but with the events in C concealed. Such events are not observed by the environment but are deemed to have happened autonomously.
- Instead of $P \setminus \{c\}$ we write $P \setminus c$.

Hiding

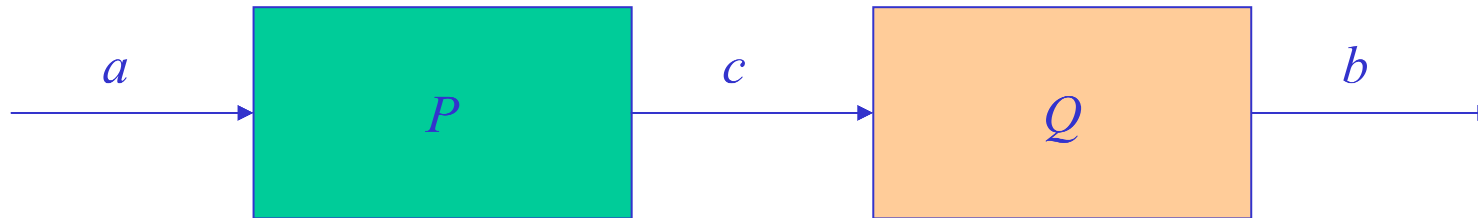
- Recall the noisy vending machine

  $NOISYVM = coin \text{ -> } clink \text{ -> } choc \text{ -> } clunk \text{ -> } NOISYVM$

  Soundproofing the simple vending machine

  $NOISYVM \setminus \{clink, clunk\} = \mu X.\ coin \text{ -> } choc \text{ -> } X$

Hiding

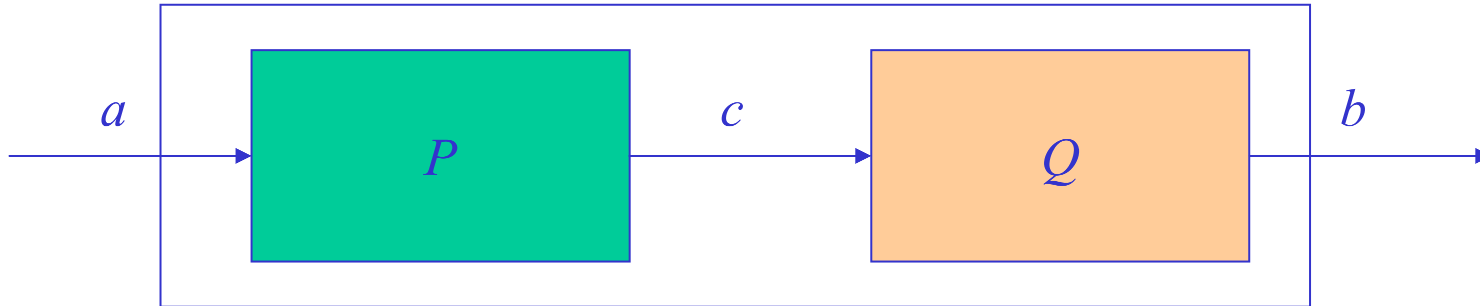


$\mu X.\ (a\text{->}\ c\ \text{->}\ X)\ [|c|]\ (\mu X.\ c\text{->}\ b\ \text{->}\ X)$

$=$

$a\text{->}\ c\ \text{->}\ \mu X.\ (a\text{->}\ b\ \text{->}c\text{->}\ X\ |\ b\text{->}\ a\text{->}c\text{->}X)$

Hiding

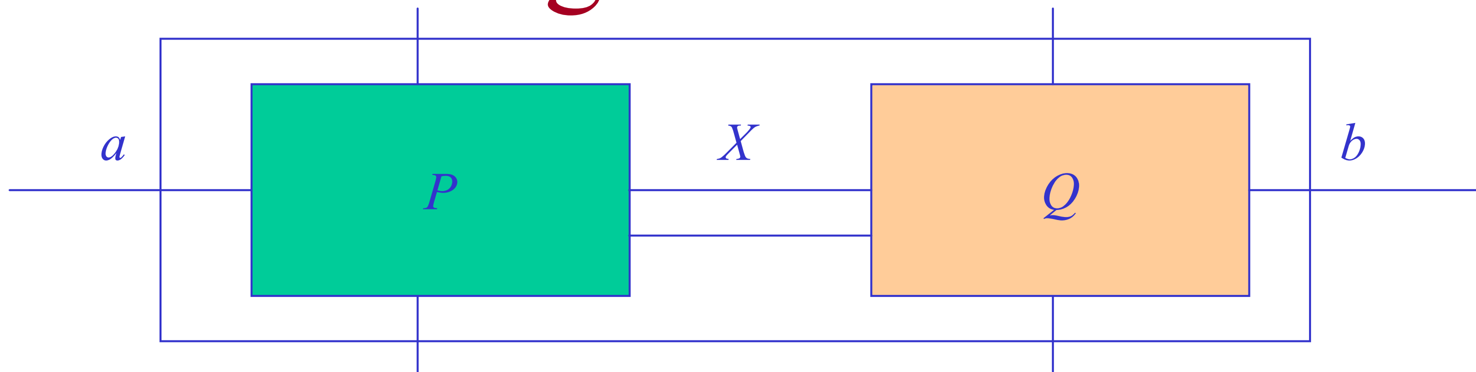


$$(\mu X.\ (a\text{->}\ c\ \text{->}\ X)\ [|c|]\ (\mu X.\ c\text{->}\ b\ \text{->}\ X))\ \backslash\ c$$

$$=$$

$$a\ \text{->}\ \mu X.\ (a\text{->}\ b\ \text{->}X\ |\ b\text{->}\ a\text{->}X)$$

Use of Hiding



- Systems are designed as a composition of several components, which communicate with each other and with the environment
- The interface can be concealed if no further component need to synchronise with events in X.

Hiding: Laws

- Basic laws for the unary process operator of hiding are
  - $P \setminus \{\ \} = P$
  - $(P \setminus B) \setminus C = P \setminus (B \quad C)$
  - $(P \sqcap Q) \setminus C = (P \setminus C) \sqcap (Q \setminus C)$
  - $Stop \setminus C = Stop.$

Hiding: Laws

$(a \rightarrow P) \setminus C = P \setminus C \lhd a \quad C \rhd a \rightarrow (P \setminus C)$

$((a \rightarrow P)[](b \rightarrow Q)) \setminus \{a, b\} = P \setminus \{a, b\} \sqcap Q \setminus \{a, b\}.$

$((a \rightarrow P)[](b \rightarrow Q)) \setminus a = ?$

It is a bit more complicated. We need to consider two cases together:

1. hidden events have priority $RHS = P \setminus a$
2. external events have priority $RHS = b \text{ -> } (Q \setminus a)$

Hiding: Laws

- For comparison, hiding b on the left-hand side we obtain

  $(((a \rightarrow P)[](b \rightarrow Q)) \setminus a) \setminus b$

= <hiding law>

  $((a \rightarrow P)[](b \rightarrow Q)) \setminus \{a, b\}$

= <simple step law for hiding>

  $(P \setminus \{a, b\}) \sqcap (Q \setminus \{a, b\})$.

Hiding: Laws

- But hiding b on the right-hand side we find,

  1. hidden events have priority:

     $$RHS \setminus b = (P \setminus a) \setminus b = P \setminus \{a, b\}$$

  2. external events have priority:

     $$RHS \setminus b = (b \rightarrow (Q \setminus a)) \setminus b = Q \setminus \{a, b\}.$$

- Thus in neither case does LHS = RHS. The correct RHS must somehow offer both hidden and external events.

Aside: timeout

- CSP interprets the RHS as follows.
  - Initially the external event is offered: process $b \rightarrow Q \setminus a$ is offered
  - then after some time, if that has not been chosen, the internal event occurs: process $P \setminus a$ is offered.
- Such behaviour is described by the timeout operator $X \triangleright Y$ which eventually offers Y but before it does, offers X.

Aside: timeout

If the environment is slow it is offered only Y but if it is quick it can choose X; in trying prematurely to choose Y the environment is offered *Stop*. The definition is thus

$$X \rhd Y = (X \sqcap Stop) [] Y\,.$$

- Timout is associative and distributive in each argument. In the law of hiding we are dealing, $X = (b \to Q \setminus a)$ and $Y = (P \setminus a)$.
- Thus eventually the internal event occurs but before it does the external event is offered

$$RHS = (b \to Q \setminus a) \rhd (P \setminus a).$$

Hiding: Laws

The general hiding law is thus

$$(?a : A \longrightarrow P(a)) \setminus C = ?a : A \longrightarrow (P(a) \setminus C)$$

$$\lhd A \cap C = \{\,\} \rhd$$

$$(?a : (A \setminus C) \longrightarrow (P(a) \setminus C) \rhd \sqcap \{P(a) \mid a \quad A \cap C\}).$$

Alternative definition of timeout

- Here is a characterisation of $P \triangleright Q$ which may be easier to understand in general contexts:

$P \triangleright Q$

$= \langle definition \rangle$

$(P \sqcap Stop)[]Q$

$= \langle distributivity \rangle$

$(P[]Q) \sqcap (Stop[]Q)$

Alternative definition of timeout

= <Stop a unit for []>

$(P[]Q) \sqcap Q$

= <distributivity again>

$(P \sqcap Q)[](Q \sqcap Q)$

= <⊓ idemptotent>

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

Non law

Hiding does not distribute through parallel composition: in general

$$(P\ [|X|]\ Q) \setminus C \neq (P \setminus C)\ [|X|]\ (Q \setminus C).$$

- Consider, for example,
- $\Sigma = \{a, b, c\}$
- $X = \{c\}$
- $C = \{c\}$
- $P = a \rightarrow c \rightarrow P$
- $Q = c \rightarrow b \rightarrow Q.$

Non law

- Then,

  LHS

$= \langle$a previous example$\rangle$

$\quad a \rightarrow \mu X \bullet (a \rightarrow b \rightarrow X \mid b \rightarrow a \rightarrow X)$

$\neq \langle$different traces$\rangle$

$\quad (\mu X \bullet a \rightarrow X)\ [|\{c\}|]\ (\mu X \bullet b \rightarrow X)$

$= \langle$definition$\rangle$

  RHS.

- However, if $X \cap C = \{\ \}$, then

$$(P\ [|X|] Q) \setminus C = (P \setminus C)\ [|X|] (Q \setminus C).$$

Non law

- Hiding does not distribute through external choice: in general

$$(P[]Q) \setminus C \neq (P \setminus C)[](Q \setminus C).$$

- Let

  $P = c \longrightarrow Stop$

  $Q = d \longrightarrow Stop$

  $C = \{c\}.$

  Then

  LHS

  = <definitions>

  $(c \longrightarrow Stop[]d \longrightarrow Stop) \setminus c$

  = <hiding law>

Non law

$(d \rightarrow \mathrm{Stop} \sqcap \mathrm{Stop})[]\mathrm{Stop}$

= <[] laws>

$(d \rightarrow \mathrm{Stop}) \sqcap \mathrm{Stop}$

≠ <distinct behaviour>

$d \rightarrow Stop$

= <[] law again>

$Stop[]d \rightarrow Stop$

= <hiding laws>

$(c \rightarrow Stop) \setminus c \; [] \; (d \rightarrow Stop) \setminus c$

= <definitions>

RHS.

Hiding: Law

- Hiding finitely-many events is continuous

  $$(\mu X \bullet F\ X) \setminus C = \mu X \bullet (F\ X) \setminus C.$$

- The set C is assumed to be finite in the definition of hiding in order to ensure that law in CSP.

Hiding: Traces

- The traces of hiding are obtained by removing all hidden events from the traces of the original process:

  $traces(P \setminus C) = \{t \upharpoonright \Sigma \setminus C \mid t \quad traces\ P\}.$

Divergence

- Process performs na infinite sequence of internal events.
- Hiding can introduce divergence
  - $P = (\mu X.a \text{ -> } X)\backslash\{a\}$
  - $Q = a \text{ -> } Q\backslash\{a\}$
  - $P = P \ \ (P = (\mu X.X))$
  - $P = div$

Communication

- Channels
  - Events of the form *c?x* or *c!e* are called *input* and *output* events respectively.
  - *c* is the channel, *e* the expression being output along the channel and *x* is the variable assigned the value input along the channel. The type of a channel is the type of the variables or expressions along it.

Communication

- Example:

$$Copy = in?x \rightarrow out\,!x \rightarrow Copy$$

Communication

- That process can be thought of as a short way of writing the process *Co*

$$\Sigma = \{in.0, in.1, out.0, out.1\}$$

$$Co = (in.0 \to out.0 \to Co$$
$$\mid in.1 \to out.1 \to Co)$$
$$= a : \{in.0, in.1\} \to (out.0 \to Co \lhd a = in.0 \rhd$$
$$out.1 \to Co).$$

A simple change in the type of the channels enables it to copy values of any (even infinite) given type.

Communication

Thus

$$c!e \rightarrow P$$

is the process which first outputs expression e on channel c and then behaves like P.

Similarly,

$$c?x \rightarrow P(x)$$

is the process which first inputs any value x on channel c and then behaves like $P(x)$. It must accept any value of the correct type.

Communication

- Conventions
  - Each channel is used in only one direction.
  - Only two processes use a channel, one for output the other for input.
  - If a channel is used by two processes then it is assigned the same type by each.
  - Channels, their names, directions and types, are depicted.
  - If t is a trace of process P having channel c then the sequence of values in $t \restriction c$ is denoted simply c. For example a trace t of *Copy* satisfies $values(t \restriction out) \leq_1 values(t \restriction in)$, which simplifies to $out \leq_1 in$.

Communication

- Example
  - A variable either accepts an input on the write channel or outputs on read the value most recently input.

$$Var = write?x \rightarrow V(x)$$

$$V(x) = (write?y \rightarrow V(y) \mid read!x \rightarrow V(x))$$

Communication

- Example
  - Process *Squash* is like *Copy* acting on the type of symbols, except that each pair of consecutive heart symbols ♥ is replaced by a single spade symbol ♠.

$$
\begin{aligned}
Squash = \mu X \bullet (in?x \rightarrow (&out\,!x \rightarrow X \\
&\lhd\, x \neq \heartsuit \,\rhd \\
&in?y \rightarrow (out\,!\spadesuit \rightarrow X \\
&\qquad\qquad \lhd\, y = \heartsuit \,\rhd \\
&\qquad\qquad out\,!\heartsuit \rightarrow out\,!y \rightarrow X))).
\end{aligned}
$$

Communication

- Example
  - An unbounded queue, *Queue*, either accepts an input or outputs the first message input but not already output.

$$Queue = Q(\langle\rangle) = left?x \rightarrow Q(\langle x\rangle)$$

$$Q(x:xs) = (left?y \rightarrow Q(x:xs\ \hat{}\ \langle y\rangle) \mid right!x \rightarrow Q(xs))$$

Communication

Input and output in parallel result in communication of the expression from the output process to the input process

$(c!e \rightarrow P)\ [|c|]\ (c?x \rightarrow Q(x)) = c!e \rightarrow (P[|c|]\ Q(e)).$

The result *c!e* is an output allowing the resulting process to be placed in parallel with another processes, which inputs on *c*.

(However in practice systems composed of processes having only pairwise communications are most common.)

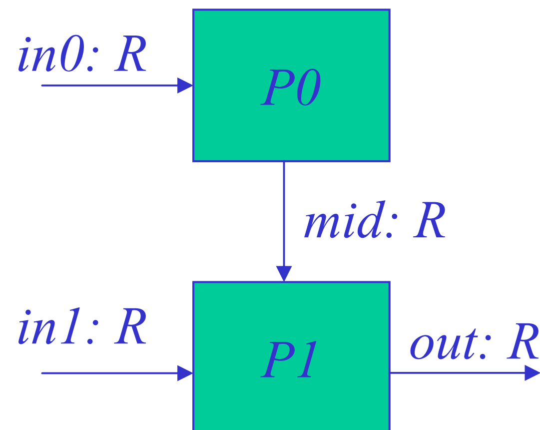
Communication

It is desired to input numbers x, y on different channels and to output a fixed linear combination $a\times x + b\times y$. One design is

$$Slow = (in0?x \rightarrow in1?y \rightarrow out\,!(a\times x + b\times y) \rightarrow Slow$$
$$\mid in1?y \rightarrow in0?x \rightarrow out\,!(a\times x + b\times y) \rightarrow Slow).$$

Communication

But multiplication is time consuming whilst addition is quick. A design Quick performing the multiplications concurrently is



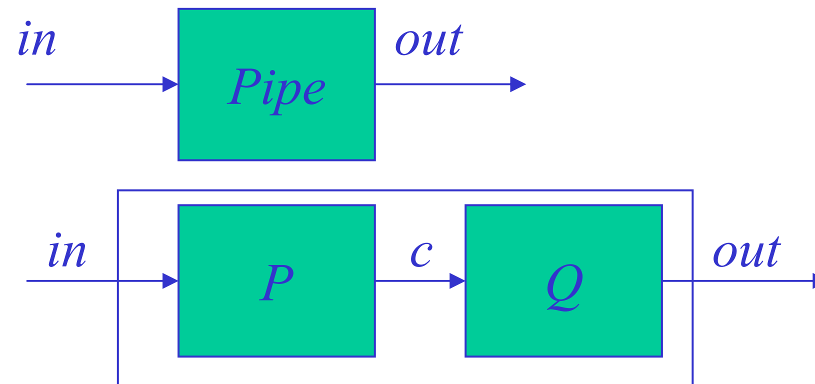
$P0 = in0?x \rightarrow mid!(a \times x) \rightarrow P0$

$P1 = in1?x \rightarrow mid?y \rightarrow out\,!(y + b \times x) \rightarrow P1$

$Quick = (P0\ [|\ mid\ |]\ P1) \setminus mid$

Piping

A pipe is a process having (just) an input channel, *in*, and an output channel, *out*. If *P* and *Q* are pipes then their *piping*, or chaining, is the pipe *P>>Q* defined by placing *Q* after *P* and hiding the intermediate channel



$P>>Q = (P[c/out]\ [|c|]\ Q[c/in]) \setminus c$, where *c* is a fresh channel-variable name.

Piping: Examples

- Process *Copy* is a one-place buffer. An *n*-place buffer is obtained by piping together *n* copies of Copy

$$>>_{0 \le i < n} \mathit{Copy}.$$

Piping: Laws

- Piping is associative, and distributive in each argument

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

$$(P \sqcap Q)>>R = (P>>R) \sqcap (Q>>R)$$

$$P>>(Q \sqcap R) = (P>>Q) \sqcap (P>>R).$$

Piping: Laws

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

$$(out!e \rightarrow P) >> (out!f \rightarrow Q) = out!f \rightarrow ((out!e \rightarrow P) >> Q)$$

$$(out!e \rightarrow P) >> (in?x \rightarrow Q(x)) = P >> Q(e)$$

$$(in?x \rightarrow P) >> (out!f \rightarrow Q) = in?x \rightarrow (P >> (out!f \rightarrow Q))$$

$$[]\, out!f \rightarrow ((in?x \rightarrow P) >> Q)$$

Piping: Laws

If

$P = (in?x : A \rightarrow P(x\,))[](out\,!b : B \rightarrow P'(b))$

$Q = (in?y : C \rightarrow Q(y))[](out\,!d : D \rightarrow Q'(d))$

where x is not free in Q nor y in P, then

$P{>>}Q = (in?x : A \rightarrow (P(x\,){>>}Q))[](out\,!d : D \rightarrow (P{>>}Q'(d))$

$\lhd B \cap C = \{\ \} \rhd$

$(in?x : A \rightarrow (P(x\,){>>}Q))[](out\,!d : D \rightarrow (P{>>}Q'(d))$

$\rhd \sqcap\{P'(z\,){>>}Q(z\,) \mid z \quad B \cap C\}.$

Piping: Laws

Home Exercise: Prove that

$$Copy >> Copy = in?x \rightarrow \mu X \bullet out\,!x \rightarrow in?y \rightarrow X$$
$$[]$$
$$in?y \rightarrow out\,!x \rightarrow X.$$

Piping: Deadlock and Divergence

If P and Q are deadlock free then so too is $P{>>}Q$ since there can be no cycle of dependencies.

But when is $P{>>}Q$ divergence free? Not always:

$(\mu P \bullet out!1 \rightarrow P){>>}(\mu Q \bullet in?x \rightarrow Q) = Div.$

Indeed

$(out!1 \rightarrow P){>>}(in?x \rightarrow Q)$

= third law

$P{>>}Q(1)$

which is an unguarded recursion.

Piping: Deadlock and Divergence

– P>>Q is divergence free if either:
  - P cannot engage in an unbroken sequence of outputs
  - Q cannot engage in an unbroken sequence of inputs.

Piping: Traces

Home exercise: *traces(P>>Q) ?*