Calculi of Communicating Processes

Rocco De Nicola

Dipartimento di Sistemi ed Informatica
Università di Firenze

Process Algebras and Concurrent Systems

CCS: Calculus of Communicating Processes

Milner - 1980

The set of actions Act_τ consists of a set of labels Λ, of the set $\overline{\Lambda}$ of complementary labels and of the distinct action τ, the syntax is

$$E ::= nil \mid \mu.E \mid E\backslash L \mid E[f] \mid E_1 + E_2 \mid E_1|E_2 \mid recX.E$$

Moreover we have:

- $\mu \in Act_\tau$;
- $L \subseteq \Lambda$;
- $f : Act_\tau \to Act_\tau$;
- $f(\bar{\alpha}) = \overline{f(\alpha)}$ and $f(\tau) = \tau$.

CCS has been studied with Bisimulation and Testing Semantics

SCCS: Synchronous Calculus of Communicating Processes

Milner - 1983

The set of actions Act is an Abelian group containing a set of labels Λ, and of complementary actions $\overline{\Lambda}$ with over-dashed actions, the neutral element is 1, the syntax is

$$E ::= nil \mid \mu : E \mid E \upharpoonright L \mid E_1 + E_2 \mid E_1 \times E_2 \mid recX.E$$

where

- $\mu \in Act \cup \{1\}$,
- $L \subseteq \Lambda$,
- : denotes action prefixing

There is no relabelling operator, it is expressible via the other operators.

SCCS has been studied with Bisimulation Semantics

TCSP: Theoretical Communicating Sequential Processes

Brookes-Hoare-Roscoe - 1984

The set of actions is a set Λ, and the syntax is

$$E ::= stop \mid skip \mid a \rightarrow E \mid E \backslash L \mid E[f] \mid E_1 ; E_2 \mid E_1 \sqcap E_2 \mid E_1 \Box E_2 \mid E_1 \parallel E_2 \mid E_1 \lll E_2 \mid E_1 \,|[L]|\, E_2 \mid A$$

where

- $a \in \Lambda$, $L \subseteq \Lambda$, $f : \Lambda \rightarrow \Lambda$,
- the operators $\sqcap$ and $\Box$ denote internal and external choice respectively;
- the operator $\rightarrow$ denotes action prefixing
- A is a process constant

CSP has been studied with Failure Semantics - a variant of Testing Sem.

ACP: Algebra of Communicating Processes

Bergstra-Klop - 1984

The set of actions Λ_τ consists of a finite set of labels Λ and of special action τ, the syntax is

$$E ::= \sqrt{} \mid a \mid E\backslash L \mid E/L \mid E[f] \mid E_1 \cdot E_2 \mid E_1 + E_2 \mid E_1 \parallel E_2 \mid E_1 \between E_2 \mid E_1 |_c E_2 \mid A$$

- $a \in \Lambda_\tau$, $L \subseteq \Lambda$, $f : \Lambda \to \Lambda$;
- the operator $\cdot$ denotes sequential composition;
- A is a process constant.
- The original notation for operators $\cdot\backslash L$, $\cdot/L$ e $\cdot[f]$ are $\delta_L(\cdot)$, $\tau_L(\cdot)$ and $\rho_f(\cdot)$) respectively.

ACP has been studied with Bisimulation and Branching Bis. Semantics

LOTOS: Language of Temporal Order Specification

Standard ISO - 1988

The set of actions Λ_i contains a set of labels Λ and the distinct label i, the syntax is

$$E ::= stop \mid exit \mid \mu; E \mid E/L \mid E[f] \mid E_1 \gg E_2 \mid E_1 [> E_2 \mid E_1 + E_2 \mid E_1 \parallel E_2 \mid E_1 \lll E_2 \mid E_1 \|[L]\| E_2 \mid A$$

- $\mu \in \Lambda_i$, $L \subseteq \Lambda$, $f : \Lambda \to \Lambda$;
- the operator ; denotes action prefixing;
- the operator $\gg$ denotes parallel composition;
- A is a process constant.

LOTOS has been studied with Bisimulation and Testing Semantics

From CCS to π-calculus - 1

Consider a scenario of somebody willing to buy a pizza.

In CCS, we can model this situation by composing in parallel the client C, and the "pizzaiolo" P.

From CCS to π-calculus - 1

Consider a scenario of somebody willing to buy a pizza.

In CCS, we can model this situation by composing in parallel the client C, and the "pizzaiolo" P.

$$C \triangleq \overline{askPizza}.\overline{pay}.pizza$$

The client C asks for a pizza, pays for it and takes it away.

From CCS to π-calculus - 1

Consider a scenario of somebody willing to buy a pizza.

In CCS, we can model this situation by composing in parallel the client C, and the "pizzaiolo" P.

$$C \triangleq \overline{askPizza}.\overline{pay}.pizza \qquad P \triangleq askPizza.pay.\overline{pizza}$$

The client C asks for a pizza, pays for it and takes it away.
The "pizzaiolo" P receives the request for the pizza, gets the money and delivers the pizza.

From CCS to π-calculus - 2

If we use values, i.e. CCS with value passing, we can add further details to our system.

From CCS to π-calculus - 2

If we use values, i.e. CCS with value passing, we can add further details to our system.

$$C \triangleq \overline{askPizza}\langle margherita\rangle.\overline{pay}\langle 5\ Euro\rangle.pizza$$

The client asks for a Margherita, pays the due amount and eats the pizza.

From CCS to π-calculus - 2

If we use values, i.e. CCS with value passing, we can add further details to our system.

$$C \triangleq \overline{askPizza}\langle margherita\rangle.\overline{pay}\langle 5\ Euro\rangle.pizza$$

$$P \triangleq askPizza(x).pay(y).\textbf{if } y = price(x) \textbf{ then } pizza \textbf{ else } \textbf{if } y > price(x) \textbf{ then } \overline{pizza}.\overline{output}\langle y - price(x)\rangle \textbf{ else } \overline{askMoney}$$

The client asks for a Margherita, pays the due amount and eats the pizza. The "pizzaiolo" receives the request for the pizza, gets the money then checks the received amount and gives back the requested pizza and possibly the change.

From CCS to π-calculus - 3

With π-calculus we can do more: home delivery of pizza!

$$
\begin{aligned}
C &\triangleq \overline{\overline{askPizza}}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \\
P &\triangleq askPizza(y).pay.(\nu\, pizza)\overline{y}\langle pizza\rangle.P
\end{aligned}
$$

The client can communicate the address where he wants the pizza be delivered.

From CCS to π-calculus - 3

With π-calculus we can do more: home delivery of pizza!

$$C \triangleq \overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle$$
$$P \triangleq askPizza(y).pay.(\nu pizza)\overline{y}\langle pizza\rangle.P$$

The client can communicate the address where he wants the pizza be delivered.

$$\overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid askPizza(y).pay.(\nu pizza)\overline{y}\langle pizza\rangle.P$$

From CCS to π-calculus - 3

With π-calculus we can do more: home delivery of pizza!

$$C \triangleq \overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle$$
$$P \triangleq askPizza(y).pay.(\nu pizza)\overline{y}\langle pizza\rangle.P$$

The client can communicate the address where he wants the pizza be delivered.

$$\overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid askPizza(y).pay.(\nu pizza)\overline{y}\langle pizza\rangle.P$$
$$\xrightarrow{\tau} \overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid pay.(\nu pizza)\overline{myHome}\langle pizza\rangle.P$$

From CCS to π-calculus - 3

With π-calculus we can do more: home delivery of pizza!

$$
\begin{aligned}
C &\triangleq \overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \\
P &\triangleq askPizza(y).pay.(\nu pizza)\overline{y}\langle pizza\rangle.P
\end{aligned}
$$

The client can communicate the address where he wants the pizza be delivered.

$$
\begin{aligned}
&\overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid \\
&askPizza(y).pay.(\nu pizza)\overline{y}\langle pizza\rangle.P \\
&\quad \xrightarrow{\tau} \overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid pay.(\nu pizza)\overline{myHome}\langle pizza\rangle.P
\end{aligned}
$$

From CCS to π-calculus - 3

With π-calculus we can do more: home delivery of pizza!

$$
\begin{aligned}
C &\triangleq \overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \\
P &\triangleq askPizza(y).pay.(\nu pizza)\overline{y}\langle pizza\rangle.P
\end{aligned}
$$

The client can communicate the address where he wants the pizza be delivered.

$$
\begin{aligned}
&\overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid \\
&askPizza(y).pay.(\nu pizza)\overline{y}\langle pizza\rangle.P \\
&\quad \xrightarrow{\tau} \overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid pay.(\nu pizza)\overline{myHome}\langle pizza\rangle.P \\
&\quad \xrightarrow{\tau} myHome(x).\overline{eat}\langle x\rangle \mid (\nu pizza)\overline{myHome}\langle pizza\rangle.P
\end{aligned}
$$

From CCS to π-calculus - 3

With π-calculus we can do more: home delivery of pizza!

$$
\begin{aligned}
C &\triangleq \overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \\
P &\triangleq askPizza(y).pay.(\nu pizza)\overline{y}\langle pizza\rangle.P
\end{aligned}
$$

The client can communicate the address where he wants the pizza be delivered.

$$
\begin{aligned}
&\overline{askPizza}\langle myHome\rangle.\overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid \\
&askPizza(y).pay.(\nu pizza)\overline{y}\langle pizza\rangle.P \\
&\quad \xrightarrow{\tau} \overline{pay}.myHome(x).\overline{eat}\langle x\rangle \mid pay.(\nu pizza)\overline{myHome}\langle pizza\rangle.P \\
&\quad \xrightarrow{\tau} myHome(x).\overline{eat}\langle x\rangle \mid (\nu pizza)\overline{myHome}\langle pizza\rangle.P \\
&\quad \xrightarrow{\tau} (\nu pizza)(\overline{eat}\langle pizza\rangle \mid P)
\end{aligned}
$$

Syntax of π-calculus

We assume a countably infinite set of names $\mathcal{N}$ is defined.

(Processes) P	$::=$	S	sum
	$\mid$	$P_1 \mid P_2$	parallel composition
	$\mid$	$(\nu x)P$	name restriction
	$\mid$	$!P$	replication
(Sums) S	$::=$	$\mathbf{0}$	inactive process (nil)
	$\mid$	$\pi.P$	prefix
	$\mid$	$S_1 + S_2$	choice
(Prefixes) π	$::=$	$\overline{x}\langle y \rangle$	sends y on x
	$\mid$	$x(z)$	replaces z with the name received on x
	$\mid$	τ	internal action
	$\mid$	$[x = y]\pi$	matching: tests equality of x and y

Notation, Comments and Remarks

- $(\nu z)P$ is alike CCS restriction $P\backslash z$.
- $!P$ models replication and denotes the parallel composition of an arbitrary number of copies of P.
- $[x = y]\pi.P$ is known as name matching: it is equivalent to **if** $x = y$ **then** $\pi.P$.
- Occurrences of $\mathbf{0}$ will sometimes be omitted, thus, e.g., $\overline{x}\langle y\rangle.\mathbf{0}$ will be written $\overline{x}\langle y\rangle$.

Notation, Comments and Remarks

- $(\nu z)P$ is alike CCS restriction $P\backslash z$.
- $!P$ models replication and denotes the parallel composition of an arbitrary number of copies of P.
- $[x = y]\pi.P$ is known as name matching: it is equivalent to **if** $x = y$ **then** $\pi.P$.
- Occurrences of $\mathbf{0}$ will sometimes be omitted, thus, e.g., $\overline{x}\langle y\rangle.\mathbf{0}$ will be written $\overline{x}\langle y\rangle$.
- $x(z)$ indicates input while $\overline{x}\langle y\rangle$ indicates output.

Notation, Comments and Remarks

- $(\nu z)P$ is alike CCS restriction $P\backslash z$.
- $!P$ models replication and denotes the parallel composition of an arbitrary number of copies of P.
- $[x = y]\pi.P$ is known as name matching: it is equivalent to **if** $x = y$ **then** $\pi.P$.
- Occurrences of $\mathbf{0}$ will sometimes be omitted, thus, e.g., $\overline{x}\langle y\rangle.\mathbf{0}$ will be written $\overline{x}\langle y\rangle$.
- $x(z)$ indicates input while $\overline{x}\langle y\rangle$ indicates output.
- In $x(z).P$ e $(\nu z)P$, the name z is *bound* in P (i.e., P is the scope of such name). A name that is not bound is called *free*.
- $fn(P)$ e $bn(P)$ are the sets of all free, resp. bound, names of P.
- We take processes up to *alpha-conversion*, denoted by $=_\alpha$, which permits renaming of a bound name with a *fresh* name that is not already used.

An LTS for π-calculus

$$(IN)\ a(x).P \xrightarrow{ab} P[b/x] \qquad (OUT)\ \bar{a}\langle b\rangle.P \xrightarrow{\bar{a}b} P$$

$$(COM)\ \frac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\bar{a}b} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

Symmetric Com Rule

$$(RES)\ \frac{P \xrightarrow{\alpha} P'}{(\nu b)P \xrightarrow{\alpha} (\nu b)P'} \quad b \notin n(\alpha) \qquad (OP)\ \frac{P \xrightarrow{\bar{a}b} P'}{(\nu b)P \xrightarrow{\bar{a}(b)} P'} \quad a \neq b$$

$$(CLO)\ \frac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\bar{a}(b)} Q'}{P \mid Q \xrightarrow{\tau} (\nu b)(P' \mid Q')} \quad b \notin fn(P)$$

Symmetric Close Rule

$$(PAR)\ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad bn(\alpha) \cap fn(Q) = \emptyset$$

Symmetric Par Rule

$$(EQ)\ \frac{P \xrightarrow{\alpha} P'}{[a = a]P \xrightarrow{\alpha} P'} \qquad (REP)\ \frac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

Structural Congruence for π-calculus - I

Why Structural Congruence?

The syntax of π-calculus processes is to some extent *too concrete* (even when taken up-to α-conversion):

1. The order processes are composed in parallel should not matter.
2. The order processes "summed" should not matter.
3. The order names are restricted should not matter.

In fact, we shall make sure that processes differing only for the above aspects are always equivalent.

Structural Congruence for π-calculus - I

Why Structural Congruence?

The syntax of π-calculus processes is to some extent *too concrete* (even when taken up-to α-conversion):

1. The order processes are composed in parallel should not matter.
2. The order processes "summed" should not matter.
3. The order names are restricted should not matter.

In fact, we shall make sure that processes differing only for the above aspects are always equivalent.

By taking processes up to a suitable structural congruence we can:

1. Write processes in a canonical form.
2. Represent all possible interactions with fewer rules.

Structural Congruence for π-calculus - II

$$P \mid \mathbf{0} \equiv P \qquad P_1 \mid P_2 \equiv P_2 \mid P_1 \qquad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$$

$$S + \mathbf{0} \equiv S \qquad S_1 + S_2 \equiv S_2 + S_1 \qquad S_1 + (S_2 + S_3) \equiv (S_1 + S_2) + S_3$$

Structural Congruence for π-calculus - II

$$P \mid \mathbf{0} \equiv P \qquad P_1 \mid P_2 \equiv P_2 \mid P_1 \qquad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$$

$$S + \mathbf{0} \equiv S \qquad S_1 + S_2 \equiv S_2 + S_1 \qquad S_1 + (S_2 + S_3) \equiv (S_1 + S_2) + S_3$$

$$!P \equiv P \mid !P \qquad [a = a]\pi.P \equiv \pi.P$$

$$(\nu a)\mathbf{0} \equiv \mathbf{0} \qquad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \qquad \frac{a \notin fn(P)}{P \mid (\nu a)Q \equiv (\nu a)(P \mid Q)}$$

Structural Congruence for π-calculus - II

$$P \mid \mathbf{0} \equiv P \qquad P_1 \mid P_2 \equiv P_2 \mid P_1 \qquad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$$

$$S + \mathbf{0} \equiv S \qquad S_1 + S_2 \equiv S_2 + S_1 \qquad S_1 + (S_2 + S_3) \equiv (S_1 + S_2) + S_3$$

$$!P \equiv P \mid !P \qquad [a = a]\pi.P \equiv \pi.P$$

$$(\nu a)\mathbf{0} \equiv \mathbf{0} \qquad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \qquad \frac{a \notin fn(P)}{P \mid (\nu a)Q \equiv (\nu a)(P \mid Q)}$$

$$P \equiv P \qquad \frac{P \equiv Q}{Q \equiv P} \qquad \frac{P \equiv Q \quad Q \equiv R}{P \equiv R} \text{ (equivalence)}$$

Structural Congruence for π-calculus - II

$$P \mid \mathbf{0} \equiv P \qquad P_1 \mid P_2 \equiv P_2 \mid P_1 \qquad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$$

$$S + \mathbf{0} \equiv S \qquad S_1 + S_2 \equiv S_2 + S_1 \qquad S_1 + (S_2 + S_3) \equiv (S_1 + S_2) + S_3$$

$$!P \equiv P \mid !P \qquad [a = a]\pi.P \equiv \pi.P$$

$$(\nu a)\mathbf{0} \equiv \mathbf{0} \qquad (\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \qquad \frac{a \notin fn(P)}{P \mid (\nu a)Q \equiv (\nu a)(P \mid Q)}$$

$$P \equiv P \qquad \frac{P \equiv Q}{Q \equiv P} \qquad \frac{P \equiv Q \quad Q \equiv R}{P \equiv R} \text{ (equivalence)}$$

$$\frac{P =_\alpha P'}{P \equiv P'} \qquad \frac{P \equiv P'}{\mathbb{C}[P] \equiv \mathbb{C}[P']} \text{ (congruence)}$$

Canonical Form

For each π-calculus process P there exist:

1. a finite number of names $x_1, ..., x_k$,
2. a finite number of sums $S_1, ..., S_n$, and
3. a finite number of processes $P_1, ..., P_m$ such that

$$P \equiv (\nu x_1)...(\nu x_k)\big(S_1|...|S_n|!P_1|...|!P_m\big)$$

The structural congruence permits rearranging the terms describing π-calculus processes so that any two possibly interacting subterms (composed in parallel) can be put side by side.

All interactions can then be expressed by considering only a very small number of cases.

Reduction Semantics: An alternative sem. for π-calculus

The so-called *reduction semantics* focuses on *internal* moves $P \longmapsto Q$ only and takes significantly advantage of structural congruence.

$$(RTAU) \quad \frac{}{(\tau.P + S) \longmapsto P}$$

$$(RCOM) \quad \frac{}{(a(x).P_1 + S_1)|(\overline{a}\langle b\rangle.P_2 + S_2) \longmapsto P_1[^b/_x]|P_2}$$

$$(RPAR) \quad \frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q}$$

$$(RRES) \quad \frac{P \longmapsto P'}{(\nu a)P \longmapsto (\nu a)P'}$$

$$(RSTRUCT) \quad \frac{P \equiv Q \quad Q \longmapsto Q' \quad Q' \equiv P'}{P \longmapsto P'}$$

Harmony Lemma

The reduction semantics and the LTS semantics can be tightly reconciled.

Notation

Given two relations $\mathcal{R}$ and $\mathcal{S}$ on processes, we write $P\ \mathcal{RS}\ Q$ if there exists a process R such that $P\ \mathcal{R}\ R$ and $R\ \mathcal{S}\ Q$.

Theorem (Harmony Lemma)

For any π-calculus process P we have:

1. $P \equiv \xrightarrow{\alpha} P'$ *implies* $P \xrightarrow{\alpha} \equiv P'$
2. $P \longmapsto P'$ *if and only if* $P \xrightarrow{\tau} \equiv P'$

Electoral Propaganda

We shall see how the use of restricted channels can prevent intrusions.

Assume we want to campaign for Romano and have set up the following scenario :

Naive Campaigning

$$
\begin{aligned}
\mathit{Speaker} &\triangleq \overline{air}\langle \mathit{vote\ for\ Romano} \rangle \\
\mathit{Microphone} &\triangleq air(x).\overline{wire}\langle x \rangle \\
\mathit{Loudspeaker} &\triangleq wire(y).\overline{highvolume}\langle y \rangle \\
\mathit{Ad} &\triangleq \mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker}
\end{aligned}
$$

Electoral Propaganda

We shall see how the use of restricted channels can prevent intrusions.

Assume we want to campaign for Romano and have set up the following scenario :

Naive Campaigning

$$
\begin{aligned}
\mathit{Speaker} &\triangleq \overline{\mathit{air}}\langle \mathit{vote\ for\ Romano} \rangle \\
\mathit{Microphone} &\triangleq \mathit{air}(x).\overline{\mathit{wire}}\langle x \rangle \\
\mathit{Loudspeaker} &\triangleq \mathit{wire}(y).\overline{\mathit{highvolume}}\langle y \rangle \\
\mathit{Ad} &\triangleq \mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker}
\end{aligned}
$$

This system will evolve as follows:

$$
\begin{aligned}
\mathit{Ad} &\longmapsto \overline{\mathit{wire}}\langle \mathit{vote\ for\ Romano} \rangle \mid \mathit{Loudspeaker} \\
&\longmapsto \overline{\mathit{highvolume}}\langle \mathit{vote\ for\ Romano} \rangle
\end{aligned}
$$

Electoral Propaganda and Intrusions

$$
\begin{aligned}
\mathit{Speaker} &\triangleq \overline{\mathit{air}}\langle \mathit{vote\ for\ Romano} \rangle \\
\mathit{Microphone} &\triangleq \mathit{air}(x).\overline{\mathit{wire}}\langle x \rangle \\
\mathit{Loudspeaker} &\triangleq \mathit{wire}(y).\overline{\mathit{highvolume}}\langle y \rangle \\
\mathit{Ad} &\triangleq \mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker}
\end{aligned}
$$

Electoral Propaganda and Intrusions

$$
\begin{aligned}
\mathit{Speaker} &\triangleq \overline{\mathit{air}}\langle \mathit{vote\ for\ Romano} \rangle \\
\mathit{Microphone} &\triangleq \mathit{air}(x).\overline{\mathit{wire}}\langle x \rangle \\
\mathit{Loudspeaker} &\triangleq \mathit{wire}(y).\overline{\mathit{highvolume}}\langle y \rangle \\
\mathit{Ad} &\triangleq \mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker}
\end{aligned}
$$

Let $\mathit{Rival} \triangleq \mathit{wire}(z).\overline{\mathit{wire}}\langle \mathit{vote\ for\ Silvio} \rangle$

$$
\begin{aligned}
\mathit{Ad} \mid \mathit{Rival} &\longmapsto \overline{\mathit{wire}}\langle \mathit{vote\ for\ Romano} \rangle \mid \mathit{Loudspeaker} \mid \mathit{Rival} \\
&\longmapsto \overline{\mathit{wire}}\langle \mathit{vote\ for\ Silvio} \rangle \mid \mathit{Loudspeaker} \\
&\longmapsto \overline{\mathit{highvolume}}\langle \mathit{vote\ for\ Silvio} \rangle
\end{aligned}
$$

Electoral Propaganda and Intrusions

$$
\begin{aligned}
\mathit{Speaker} &\triangleq \overline{\mathit{air}}\langle \mathit{vote\ for\ Romano} \rangle \\
\mathit{Microphone} &\triangleq \mathit{air}(x).\overline{\mathit{wire}}\langle x \rangle \\
\mathit{Loudspeaker} &\triangleq \mathit{wire}(y).\overline{\mathit{highvolume}}\langle y \rangle \\
\mathit{Ad} &\triangleq \mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker}
\end{aligned}
$$

Let $\mathit{Rival} \triangleq \mathit{wire}(z).\overline{\mathit{wire}}\langle \mathit{vote\ for\ Silvio} \rangle$

$$
\begin{aligned}
\mathit{Ad} \mid \mathit{Rival} &\longmapsto \overline{\mathit{wire}}\langle \mathit{vote\ for\ Romano} \rangle \mid \mathit{Loudspeaker} \mid \mathit{Rival} \\
&\longmapsto \overline{\mathit{wire}}\langle \mathit{vote\ for\ Silvio} \rangle \mid \mathit{Loudspeaker} \\
&\longmapsto \overline{\mathit{highvolume}}\langle \mathit{vote\ for\ Silvio} \rangle
\end{aligned}
$$

Rival could use *wire* because it is a public channel.

A secure propaganda would be:

$$\mathit{SecureAd} \triangleq (\nu\, \mathit{air}, \mathit{wire})(\mathit{Speaker} \mid \mathit{Microphone} \mid \mathit{Loudspeaker})$$

Establishing Secure Communication Channels

Consider two processes Alice e Bob, that want to establish a secret channel using a Trusted Server with which they have a trustworthy (secret) communication link. We have that

- c_{AS} is the communication channel between Alice and the server

Establishing Secure Communication Channels

Consider two processes Alice e Bob, that want to establish a secret channel using a Trusted Server with which they have a trustworthy (secret) communication link. We have that

- c_{AS} is the communication channel between Alice and the server
- c_{BS} is the communication channel between Bob and the server

Establishing Secure Communication Channels

Consider two processes Alice e Bob, that want to establish a secret channel using a Trusted Server with which they have a trustworthy (secret) communication link. We have that

- c_{AS} is the communication channel between Alice and the server
- c_{BS} is the communication channel between Bob and the server
- c_{AB} is the new (secure) channel that Alice and Bob want to establish to communicate.

Establishing Secure Communication Channels

Consider two processes Alice e Bob, that want to establish a secret channel using a Trusted Server with which they have a trustworthy (secret) communication link. We have that

- c_{AS} is the communication channel between Alice and the server
- c_{BS} is the communication channel between Bob and the server
- c_{AB} is the new (secure) channel that Alice and Bob want to establish to communicate.

We can code Alice, Bob and the Server as follows:

$$
\begin{array}{rcl}
A & \triangleq & (\nu c_{AB})\overline{c_{AS}}\langle c_{AB}\rangle.\overline{c_{AB}}\langle mess\rangle \\
S & \triangleq & !c_{AS}(x).\overline{c_{BS}}\langle x\rangle \mid !c_{BS}(y).\overline{c_{AS}}\langle y\rangle \\
B & \triangleq & c_{BS}(z).z(w). < use\ z >
\end{array}
$$

Establishing Secure Communication Channels

Consider two processes Alice e Bob, that want to establish a secret channel using a Trusted Server with which they have a trustworthy (secret) communication link. We have that

- c_{AS} is the communication channel between Alice and the server
- c_{BS} is the communication channel between Bob and the server
- c_{AB} is the new (secure) channel that Alice and Bob want to establish to communicate.

We can code Alice, Bob and the Server as follows:

$$\begin{array}{rcl} A & \triangleq & (\nu c_{AB})\overline{c_{AS}}\langle c_{AB}\rangle.\overline{c_{AB}}\langle mess\rangle \\ S & \triangleq & !c_{AS}(x).\overline{c_{BS}}\langle x\rangle \mid !c_{BS}(y).\overline{c_{AS}}\langle y\rangle \\ B & \triangleq & c_{BS}(z).z(w).\ < use\ z > \end{array}$$

$$(\nu c_{AS}, c_{BS})(A|S|B) \longmapsto\longmapsto\longmapsto (\nu c_{AS}, c_{BS}, c_{AB})(S \mid\ < use\ mess >)$$