Self-Adaptation and Information Flow in Multiparty Communications

Joint work with

Ilaria Castellani (INRIA, FR)

Jorge A. Pérez (University of Groningen, NL)

ABCD meeting

London, 20th April, 2015

Outline

1. Context
2. An Example
3. Technical Details
4. Concluding Remarks

A Mismatch

- Behavioural types have proved useful to analyse structured communications featuring various concerns:
  - ⋆ protocol conformance [binary/multiparty, synchronous/asynchronous, ...]
  - ⋆ resource-usage and security policies
  - ⋆ error handling and self-adaptation
  - ⋆ time-related requirements
- Existing typed frameworks have been developed in isolation, focused on a single concern only.
- This contrasts with correctness in actual software systems, which is best characterised by the interplay of several different concerns.

This Work: A First Step to Correct the Mismatch

- A typed framework integrating
  - ⋆ security guarantees (access control and secure information flow)
  - ⋆ self-adaptation mechanisms
- Key Idea: Security violations trigger self-adaptation mechanisms.
- Assumptions:
  - ⋆ multiparty, asynchronous communication
  - ⋆ distributed partners, governed by monitors
  - ⋆ reading and writing violations are not necessarily catastrophic
  - ⋆ self-adaptation may be local or global
- A typed operational semantics, as a starting point for further developments and application-driven refinements.

Our Proposal, In a Nutshell

- global types describe the overall communication choreography, together with initial reading permissions.
- monitors govern local communication protocols for each participant and maintain both reading and writing permissions.
- processes implement an associated monitor, relying on the adequacy of process types wrt monitors.

The composition of monitored processes defines a network.

The formal semantics is given by:

1. An LTS on processes – carrying over security levels for values.
2. A (typed) reduction semantics for networks – implementing communication and adaptation mechanisms (local and global).

Outline

1. Context
2. An Example
3. Technical Details
4. Concluding Remarks

An Example: "Rate your trip" (1)

Participants:

- A travel agency
- Individual and corporate clients (groups)
- An individual and a corporate sales representative
- A statistics service

An associated structured protocol, informally:

- After returning from a trip, individual/corporate clients send to individual/corporate representatives feedback information (containing private information)
- Representatives forward received feedback:
  - ⋆ to the agency (with clients in cc)
  - ⋆ to the statistics service (in anonymized form)

An Example: “Rate your trip” (2)

Two possible security violations:

1. Representative sends a feedback with private information on a client to the statistics service (a reading violation for the service)
2. Representative sends a report, testing feedbacks from more than one client, but declassified to the level of one of these clients, to agency (with the same client in cc) (a writing violation for the representative)

How serious are these violations?

One may say that a violation is minor if a single client is concerned, and major if a corporate client is involved.

Outline

1. Context
2. An Example
3. Technical Details
4. Concluding Remarks

Global Types and Monitors

- Global types G (p and q denote participants):

$$
\begin{array}{rcll}
\mathsf{G} & ::= & \mathsf{p} \to \mathsf{q} : \{\lambda_i(S_i).\mathsf{G}_i\}_{i\in I} & \text{labelled communication} \\
& | & \mathbf{t} \;\; | \; \mu\mathbf{t}.\mathsf{G} & \text{recursion} \\
& | & \mathsf{end} & \text{completed global protocol}
\end{array}
$$

- A security global type assigns a reading level to each participant

- Monitors $\mathcal{M}$: projections of global types onto individual participants. [A sort of local types, but not quite.]

$$
\begin{array}{rcll}
\mathcal{M} & ::= & \mathsf{p}?\{\lambda_i(S_i).\mathcal{M}_i\}_{i\in I} & \text{labelled input} \\
& | & \mathsf{q}!\{\lambda_i(S_i).\mathcal{M}_i\}_{i\in I} & \text{labelled output} \\
& | & \mathbf{t} \;\; | \; \mu\mathbf{t}.\mathcal{M} & \text{recursion} \\
& | & \mathsf{end} & \text{completed local protocol}
\end{array}
$$

Processes

- Expressions $(\mathsf{e}, \mathsf{e}', \ldots)$ include booleans, naturals, and a set of Nonces (dummy fresh values).
- Every expression e is equipped with a security level, written $lev(\mathsf{e})$.
- Processes (c stands for a user channel y or a session channel $\mathsf{s}[\mathrm{p}]$):

$$
\begin{array}{lclll}
P & ::= & \mathsf{c}?\lambda(x).P \quad \mid \quad \mathsf{c}!\lambda(\mathsf{e}).P & & \text{input and output} \\
 & \mid & \text{if e then } P \text{ else } P \quad \mid \quad P + P & & \text{conditionals and sums} \\
 & \mid & X \quad \mid \quad \mu X.P \quad \mid \quad \mathbf{0} & & \text{recursion and inaction}
\end{array}
$$

Monitored Processes and Networks

A monitored process is written

$$\mathcal{M}^{\mathsf{r},\mathsf{w}}[P]$$

- ⋆ Monitor ($\sim$ local type) $\mathcal{M}$ enables the actions that P may perform.
- ⋆ Permission r denotes an upper bound for reading.
- ⋆ Permission w denotes a lower bound for writing.

Networks $N, N', \ldots$ are collections of monitored processes:

$$\begin{array}{lcll} N & ::= & \mathsf{new}(\mathsf{G}, \mathsf{L}) & \text{initiator of global protocol G} \\ & \parallel & \mathcal{M}^{\mathsf{r},\mathsf{w}}[P] & \text{monitored process} \\ & \parallel & \mathsf{s} : h & \text{queue } h \text{ for session s} \\ & \parallel & N \mid N & \text{parallel composition} \\ & \parallel & (\nu \mathsf{s})N & \text{restriction} \end{array}$$

Process Types, Informally

- Describe communication behaviour. Ranged over by $\mathsf{T}, \mathsf{T}', \ldots$.
- Labeled inputs and outputs together with intersection and union types.
- Defined as pre-types plus conditions to rule out ambiguities.
- Related to monitors via an adequacy relation, noted $\mathsf{T} \propto \mathcal{M}$, which relies on subtyping.

Typed Reduction Semantics for Networks

Denoted $N \longrightarrow N'$. Main ingredients:

- An LTS on monitors $\mathcal{M}$:

$$\mathtt{p}?\{\lambda_i(S_i).\mathcal{M}_i\}_{i\in I} \xrightarrow{\mathtt{p}?\lambda_j} \mathcal{M}_j$$

$$\mathsf{q}!\{\lambda_i(S_i).\mathcal{M}_i\}_{i\in I} \xrightarrow{\mathtt{q}!\lambda_j} \mathcal{M}_j \qquad j \in I$$

- An LTS on processes P. Two sample rules:

$$\mathsf{s}[\mathrm{p}]?\lambda(x).P \quad \xrightarrow{\mathsf{s}[\mathrm{p}]?\lambda(u)} \quad P\{u/x\}$$

$$\text{if e then } P \text{ else } Q \quad \xrightarrow{lev(\mathsf{e})} \quad Q \qquad \mathsf{e} \downarrow \text{false}$$

- A collection $\mathcal{P}$ of pairs (P, T) of typed processes.
- A structural equivalence $\equiv$ which erases processes with end monitor and commutes independent messages in queues.

Reduction for Networks: Key Ideas (1)

- A single reduction step for networks relies on labelled transitions on processes and/or monitors.
- The semantics is instrumented to ensure both access control and information flow. The two guarantees are complementary:
  - ⋆ Access control only: arbitrary outputs after tests
  - ⋆ Information flow only: arbitrary inputs by any party
- Access control is ensured via reading permissions, whereas information flow is ensured with writing permissions.
- The semantics enables reduction even if reading and writing violations arise. In that case, adaptation is triggered.

Reduction for Networks: Key Ideas (2)

Adaptation may react to violations in two ways:

- Locally by "ignoring" unauthorised actions.
  Monitors for the involved participants are dynamically modified.

- Globally by sending out nonces instead of (unauthorised) values.
  Monitors are kept unchanged until a certain point, in which the global protocol for all participants handling nonces is "restarted".

Reduction for Networks: Initialisation

$$\frac{\mathcal{M}_{\mathrm{p}} = \mathsf{G} \upharpoonright \mathrm{p} \qquad \forall \mathrm{p} \in \mathsf{part}(\mathsf{G}).\ (P_{\mathrm{p}}, \mathsf{T}_{\mathrm{p}}) \in \mathcal{P}\ \&\ \mathsf{T}_{\mathrm{p}} \propto \mathcal{M}_{\mathrm{p}}}{\mathsf{new}(\mathsf{G}, \mathsf{L}) \ \longrightarrow (\nu\, \mathsf{s}) \displaystyle\prod_{\mathrm{p} \in \mathsf{part}(\mathsf{G})} (\mathcal{M}_{\mathrm{p}}^{\mathsf{L}(\mathrm{p}),\perp}[P_{\mathrm{p}}\{\mathsf{s}[\mathrm{p}]/y\}] \mid \mathsf{s} : \emptyset)}$$

- A protocol initiator evolves into a composition of monitored processes and a fresh, empty queue.
- A monitor for each participant is obtained via (usual) projection. Processes obtained from $\mathcal{P}$; types must be adequate for the monitors.
- The initial reading levels are assigned by map L
- The initial writing levels are $\perp$

Reduction for Networks: Updating Writing Permissions

$$\frac{P \xrightarrow{\ell} P'}{\mathcal{M}^{\mathsf{r},\mathsf{w}}[P] \longrightarrow \mathcal{M}^{\mathsf{r},\mathsf{w} \sqcup \ell}[P']}$$

- This rule updates the current writing level, taking into account the level of the expression tested by the process.

  [Above, $\sqcup$ denotes the lub/join. Recall that w is a lower bound.]

Reduction for Networks: Input

$$\frac{\mathcal{M}_{\mathsf{p}} \xrightarrow{\mathsf{q}?\lambda} \widehat{\mathcal{M}}_{\mathsf{p}} \qquad P \xrightarrow{\mathsf{s}[\mathsf{p}]?\lambda(u)} P' \qquad lev(u) \leq \mathsf{r}}{\mathcal{M}_{\mathsf{p}}^{\mathsf{r},\mathsf{w}}[P] \mid \mathsf{s} : (\mathsf{q}, \mathsf{p}, \lambda(u)) \cdot h \longrightarrow \widehat{\mathcal{M}}_{\mathsf{p}}^{\mathsf{r},\mathsf{w}}[P'] \mid \mathsf{s} : h}$$

- Every input action for p must be enabled by its monitor
- The identity of the sender q is provided by the monitor
- The level associated to the received value must be lower than or equal to the current reading level [Recall that r is an upper bound.]

Reduction for Networks: Output

$$\frac{\mathcal{M}_{\mathsf{p}} \xrightarrow{\mathsf{q}!\lambda} \widehat{\mathcal{M}}_{\mathsf{p}} \qquad P \xrightarrow{\mathsf{s}[\mathsf{p}]!\lambda(u)} P' \qquad (u = v \text{ and } \mathsf{w} \leq lev(v)) \text{ or } u \in Nonces}{\mathcal{M}_{\mathsf{p}}^{\mathsf{r},\mathsf{w}}[P] \mid \mathsf{s} : h \longrightarrow \widehat{\mathcal{M}}_{\mathsf{p}}^{\mathsf{r},\mathsf{w}}[P'] \mid \mathsf{s} : h \cdot (\mathsf{p}, \mathsf{q}, \lambda(u))}$$

- The rule defines the enqueue operation of either a nonce or a value. In the latter case, the level of the value must be higher than or equal to the current writing permission of the monitor.

Reduction for Networks: Local Adaptation In

$$\frac{\mathcal{M}_{\mathrm{p}} \xrightarrow{\mathrm{q}?\lambda} \widehat{\mathcal{M}}_{\mathrm{p}} \qquad lev(v) \not\leq \mathsf{r} \qquad \mathsf{T} \propto \widehat{\mathcal{M}}_{\mathrm{p}} \qquad (P', \mathsf{T}) \in \mathcal{P}}{\mathcal{M}_{\mathrm{p}}^{\mathsf{r},\mathsf{w}}[P] \mid \mathsf{s} : (\mathsf{q}, \mathsf{p}, \lambda(v)) \cdot h \longrightarrow \widehat{\mathcal{M}_{\mathrm{p}}^{\mathsf{r},\mathsf{w}}}[P'] \mid \mathsf{s} : h}$$

- In the case of a reading violation, the local adaptation mechanism simply “ignores” the unauthorised input.
- A new implementation in which the input is not considered is installed; the (unreadable) value is removed from the queue.

Reduction for Networks: Local Adaptation Out

$$\frac{\mathcal{M}_{\mathsf{p}} \xrightarrow{\mathsf{q}!\lambda} \widehat{\mathcal{M}}_{\mathsf{p}} \qquad P \xrightarrow{\mathsf{s}[\mathsf{p}]!\lambda(v)} P' \qquad \mathsf{w} \not\preceq lev(v) \quad \widehat{\mathcal{M}}_{\mathsf{q}} = \mathcal{M}_{\mathsf{q}} \backslash ?(\mathsf{p}, \lambda) \qquad \mathsf{T} \propto \widehat{\mathcal{M}}_{\mathsf{q}} \qquad (Q', \mathsf{T}) \in \mathcal{P}}{\mathcal{M}_{\mathsf{p}}^{\mathsf{r},\mathsf{w}}[P] \mid \mathcal{M}_{\mathsf{q}}^{\mathsf{r}',\mathsf{w}'}[Q] \longrightarrow \widehat{\mathcal{M}}_{\mathsf{p}}^{\mathsf{r}\sqcap\mathsf{r}',\mathsf{w}}[P'] \mid \widehat{\mathcal{M}}_{\mathsf{q}}^{\mathsf{r}',\mathsf{w}'}[Q'\{\mathsf{s}[\mathsf{q}]/y\}]}$$

- In the case of a writing violation, the local adaptation mechanism simply “strips off” the unauthorised output.
- The receiver’s monitor is modified accordingly; a new implementation without the unauthorised exchange is installed.
- The reading permission of the offending writer is modified. This is to prevent repeated leaks from the sender to the receiver.

Reduction for Networks: Global Adaptation In

$$\frac{\mathcal{M}_{\mathsf{p}} \xrightarrow{\mathsf{q}?\lambda} \widehat{\mathcal{M}}_{\mathsf{p}} \qquad lev(v) \not\leq \mathsf{r} \qquad nonce_i = \mathsf{next}(Nonces) \qquad P \xrightarrow{\mathsf{s}[\mathsf{p}]?\lambda(nonce_i)} P'}{\mathcal{M}_{\mathsf{p}}^{\mathsf{r},\mathsf{w}}[P] \mid \mathsf{s} : (\mathsf{q}, \mathsf{p}, \lambda(v)) \cdot h \longrightarrow \widehat{\mathcal{M}}_{\mathsf{p}}^{\mathsf{r},\mathsf{w}}[P'] \mid \mathsf{s} : h}$$

In the case of a reading violation, the global adaptation mechanism

- inputs a freshly generated nonce
- removes the unreadable value from the queue

Reduction for Networks: Global Adaptation Out

$$\frac{\mathcal{M}_{\mathsf{p}} \xrightarrow{\mathsf{q}!\lambda} \widehat{\mathcal{M}}_{\mathsf{p}} \qquad P \xrightarrow{\mathsf{s}[\mathsf{p}]!\lambda(v)} P' \qquad \mathsf{w} \not\leq \mathit{lev}(v) \qquad \mathit{nonce}_i = \mathsf{next}(\mathit{Nonces}) \qquad R = \mathcal{M}_{\mathsf{q}}^{\mathsf{r}',\mathsf{w}'}[Q]}{\mathcal{M}_{\mathsf{p}}^{\mathsf{r},\mathsf{w}}[P] \mid R \mid \mathsf{s}:h \longrightarrow \widehat{\mathcal{M}}_{\mathsf{p}}^{\mathsf{r}\sqcap\mathsf{r}',\mathsf{w}}[P'] \mid R \mid \mathsf{s}:h\cdot(\mathsf{p},\mathsf{q},\lambda(\mathit{nonce}_i))}$$

In the case of a writing violation, the global adaptation mechanism:

- adds a freshly generated nonce to the queue
- modifies the reading permission of the offending writer (as before)

Reduction for Networks: Global Restart

$$\frac{\mathcal{A}(\{P_{\mathrm{p}} \mid \mathrm{p} \in \Pi\}, \mathit{nonce}_i) = \Pi' \quad F(\{P_{\mathrm{p}} \mid \mathrm{p} \in \Pi'\}) = (\mathsf{G}, \mathsf{L})}{\begin{array}{c}(\nu\, \mathsf{s})\ (\prod_{\mathrm{p}\in\Pi} \mathcal{M}_{\mathrm{p}}[P_{\mathrm{p}}] \mid \mathsf{s} : h) \\ \longrightarrow \\ (\nu\, \mathsf{s})\ (\prod_{\mathrm{p}\in\Pi-\Pi'} \mathcal{M}_{\mathrm{p}}[P_{\mathrm{p}}] \mid \mathsf{s} : h \setminus \Pi') \mid \mathsf{new}(\mathsf{G}, \mathsf{L})\end{array}}$$

- The rule identifies the subset of participants “affected” with nonces. Based on that subset, a new global type is obtained.
- The choreography is split: unaffected participants are kept unchanged and the new global type is started.

Main Results

By inspecting the reduction rules for networks, we may show that reading and writing operations always respect security permissions:

Theorem

Let N be a network.

1. *If* $N = \mathcal{M}_{\mathsf{p}}^{\mathsf{r},\mathsf{w}}[P] \mid \mathsf{s} : (\mathsf{q}, \mathsf{p}, \lambda(v)) \cdot h \longrightarrow \widehat{\mathcal{M}}_{\mathsf{p}}^{\mathsf{r},\mathsf{w}}[P'] \mid \mathsf{s} : h$*, then either* $lev(v) \leq \mathsf{r}$ *or* P' *results from an adaptation step.*
2. *If* $N = \mathcal{M}_{\mathsf{p}}^{\mathsf{r},\mathsf{w}}[P] \mid \mathsf{s} : h \longrightarrow \widehat{\mathcal{M}}_{\mathsf{p}}^{\mathsf{r},\mathsf{w}}[P'] \mid \mathsf{s} : h \cdot (\mathsf{p}, \mathsf{q}, \lambda(v))$*, then* $\mathsf{w} \leq lev(v)$.

Moreover, by extending typability to queues and monitored processes, we may show subject reduction and progress properties.

Outline

1 Context

2 An Example

3 Technical Details

4 Concluding Remarks

Some Related Work

- (Capecchi et al. - 2010): A monitored semantics for multiparty session processes which ensures secure information flow.
- (Di Giusto and Pérez - 2013): Runtime adaptation for binary sessions, based on adaptable processes.
- (Dalla Preda et al. - 2014): Rule-based adaptation for choreographic languages.
- (Bravetti et al. - 2014): Runtime adaptation for choreographies, based on adaptable processes, global and local types.
- (Coppo et al. - 2014): Self-adaptation for finite multiparty sessions based on global state and monitored processes.

Concluding Remarks

- A typed framework for multiparty communications in which access control, information flow, and adaptation are treated harmoniously.
- A simple setting with a nice degree of independence between processes, types, and the intended properties.
- Typed processes specify communication, monitors describe security; adequacy links the two.
- The current semantics for adaptation combines global and local mechanisms in a non-deterministic regime. Specific case studies and applications needed to define ways of refining this choice.

Process Types, Formally (1)

- The set of pre-types is inductively defined by:

$$
\begin{array}{rcll}
T & ::= & ?\lambda(S).T \quad \| \quad !\lambda(S).T & \text{input and output} \\
& \| & T \wedge T & \text{intersection types} \quad \text{[for sums]} \\
& \| & T \vee T & \text{union types} \quad \text{[for conditionals]} \\
& \| & \mathbf{t} \mid \mu\mathbf{t}.T \mid \mathsf{end} & \text{recursive types and inaction}
\end{array}
$$

- A process type is a pre-type satisfying the following constraints [modulo idempotence, commutativity and associativity of $\wedge$ and $\vee$]:
  - ⋆ all occurrences of the shape $T_1 \wedge T_2$ are such that $lin(T_1) \cap lin(T_2) = lout(T_1) \cap lout(T_2) = \emptyset$.
  - ⋆ all occurrences of the shape $T_1 \vee T_2$ are such that $lin(T_1) = lin(T_2) = lout(T_1) \cap lout(T_2) = \emptyset$.

We use T to range over types and $\mathcal{T}$ to denote the set of types.

Process Types, Formally (2)

- Subtyping on process types is denoted $\leq$.
  $\mathsf{T}_1 \leq \mathsf{T}_2$: a process with type T_1 has all behaviours denoted by T_2 (but possibly more).
- It is the minimal reflexive and transitive relation on $\mathcal{T}$ such that, e.g., $\mathsf{T}_1 \wedge \mathsf{T}_2 \;\leq\; \mathsf{T}_i$ and $\mathsf{T}_i \leq \mathsf{T}_1 \vee \mathsf{T}_2$ (with $i = 1, 2$).
- Process types and monitors are formally connected via adequacy.
- Let the mapping $|\cdot|$ from monitors to types be defined as

$$\begin{array}{rcl}
|\mathrm{p}?\{\lambda_i(S_i).\mathcal{M}_i\}_{i\in I}| & = & \bigwedge_{i\in I} ?\lambda_i(S_i).|\mathcal{M}_i| \\
|\mathrm{q}!\{\lambda_i(S_i).\mathcal{M}_i\}_{i\in I}| & = & \bigvee_{i\in I} !\lambda_i(S_i).|\mathcal{M}_i| \\
|\mathbf{t}| = \mathbf{t} \qquad |\mu\mathbf{t}.\mathcal{M}| & = & \mu\mathbf{t}.|\mathcal{M}| \qquad |\mathsf{end}| = \mathsf{end}
\end{array}$$

Type T is adequate for a monitor $\mathcal{M}$, written $\mathsf{T} \propto \mathcal{M}$, if $\mathsf{T} \leq |\mathcal{M}|$.

Process Types

Let us write $lin(T)$ and $lout(T)$ to denote the set of initial input and output labels in a pre-type T.

A process type is a pre-type satisfying the following constraints modulo idempotence, commutativity and associativity of unions and intersections:

- all occurrences of the shape $T_1 \wedge T_2$ are such that $lin(T_1) \cap lin(T_2) = lout(T_1) \cap lout(T_2) = \emptyset$.
- all occurrences of the shape $T_1 \vee T_2$ are such that $lin(T_1) = lin(T_2) = lout(T_1) \cap lout(T_2) = \emptyset$.

We use T to range over types and $\mathcal{T}$ to denote the set of types.

Typing Rules for Processes

$$\Gamma \vdash \mathbf{0} \rhd \mathsf{c} : \mathsf{end} \;\; \text{END} \qquad \Gamma, X : \mathsf{T} \vdash X \rhd \mathsf{c} : \mathsf{T} \;\; \text{RV}$$

$$\frac{\Gamma, X : \mathsf{T} \vdash P \rhd \mathsf{c} : \mathsf{T}}{\Gamma \vdash \mu X.P \rhd \mathsf{c} : \mathsf{T}}\;\text{REC} \qquad \frac{\Gamma, x : S \vdash P \rhd \mathsf{c} : \mathsf{T}}{\Gamma \vdash \mathsf{c}?\lambda(x).P \rhd \mathsf{c} :?\lambda(S).\mathsf{T}}\;\text{RCV}$$

$$\frac{\Gamma \vdash P \rhd \mathsf{c} : \mathsf{T} \qquad \Gamma \vdash \mathsf{e} : S}{\Gamma \vdash \mathsf{c}!\lambda(\mathsf{e}).P \rhd \mathsf{c} :!\lambda(S).\mathsf{T}}\;\text{SEND}$$

$$\frac{\Gamma \vdash \mathsf{e} : \mathsf{bool} \quad \Gamma \vdash P_1 \rhd \mathsf{c} : \mathsf{T}_1 \quad \Gamma \vdash P_2 \rhd \mathsf{c} : \mathsf{T}_2 \quad \mathsf{T}_1 \vee \mathsf{T}_2 \in \mathcal{T}}{\Gamma \vdash \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \rhd \mathsf{c} : \mathsf{T}_1 \vee \mathsf{T}_2}\;\text{IF}$$

$$\frac{\Gamma \vdash P_1 \rhd \mathsf{c} : \mathsf{T}_1 \quad \Gamma \vdash P_2 \rhd \mathsf{c} : \mathsf{T}_2 \quad \mathsf{T}_1 \wedge \mathsf{T}_2 \in \mathcal{T}}{\Gamma \vdash P_1 + P_2 \rhd \mathsf{c} : \mathsf{T}_1 \wedge \mathsf{T}_2}\;\text{CHOICE}$$

Subtyping on Process Types

We define $\leq$ as the minimal reflexive, transitive relation on $\mathcal{T}$ s.t.:

$$\mathbf{t} \leq \mathbf{t} \qquad \mathsf{T} \leq \mathsf{end} \qquad \mathsf{T}_1 \wedge \mathsf{T}_2 \ \leq \ \mathsf{T}_i \qquad \mathsf{T}_i \leq \mathsf{T}_1 \vee \mathsf{T}_2 \qquad (i = 1, 2)$$

$$\mathsf{T}_1 \leq \mathsf{T}_2 \text{ implies } !\lambda(S).\mathsf{T}_1 \leq !\lambda(S).\mathsf{T}_2 \text{ and } ?\lambda(S).\mathsf{T}_1 \leq ?\lambda(S).\mathsf{T}_2$$

$$\mathsf{T} \leq \mathsf{T}_1 \text{ and } \mathsf{T} \leq \mathsf{T}_2 \text{ imply } \mathsf{T} \ \leq \ \mathsf{T}_1 \wedge \mathsf{T}_2$$

$$\mathsf{T}_1 \leq \mathsf{T} \text{ and } \mathsf{T}_2 \leq \mathsf{T} \text{ imply } \mathsf{T}_1 \vee \mathsf{T}_2 \leq \mathsf{T}$$

$$(\mathsf{T}_1 \vee \mathsf{T}_2) \wedge \mathsf{T}_3 \leq \mathsf{T} \ \text{ iff } \ \mathsf{T}_1 \wedge \mathsf{T}_3 \leq \mathsf{T} \text{ and } \mathsf{T}_2 \wedge \mathsf{T}_3 \leq \mathsf{T}$$

$$\mathsf{T} \leq (\mathsf{T}_1 \wedge \mathsf{T}_2) \vee \mathsf{T}_3 \ \text{ iff } \ \mathsf{T} \leq \mathsf{T}_1 \vee \mathsf{T}_3 \text{ and } \mathsf{T} \leq \mathsf{T}_2 \vee \mathsf{T}_3$$

$$\mu\mathbf{t}.\mathsf{T} \leq \mu\mathbf{t}.\mathsf{T}' \ \text{ iff } \ \mathsf{T} \leq \mathsf{T}'$$

LTS on Processes

$$\mathsf{s}[\mathsf{p}]?\lambda(x).P \xrightarrow{\mathsf{s}[\mathsf{p}]?\lambda(u)} P\{u/x\}$$

$$\mathsf{s}[\mathsf{p}]!\lambda(\mathsf{e}).P \xrightarrow{\mathsf{s}[\mathsf{p}]!\lambda(u)} P \quad \mathsf{e}\downarrow u$$

$$\text{if} \quad \mathsf{e} \quad \text{then} \quad P \quad \text{else } Q \xrightarrow{lev(\mathsf{e})} P \qquad \mathsf{e}\downarrow \text{true}$$

$$\text{if} \quad \mathsf{e} \quad \text{then} \quad P \quad \text{else } Q \xrightarrow{lev(\mathsf{e})} Q \qquad \mathsf{e}\downarrow \text{false}$$

$$P \xrightarrow{\alpha} P' \;\Rightarrow\; P + Q \xrightarrow{\alpha} P'$$

$$P \xrightarrow{\ell} P' \;\Rightarrow\; P + Q \xrightarrow{\ell} P' + Q$$

Structural Congruence on Networks

A congruence $\equiv$ on networks such that

- parallel is commutative and associative operator, with $\mathbf{0}$ as neutral element.
- monitored processes with end monitor are erased:

$$\mathsf{end}[P] \mid N \ \equiv \ N$$

- messages from unrelated participants are commuted:

$$h \cdot (\mathsf{p}, \mathsf{q}, \lambda(u)) \cdot (\mathsf{p}', \mathsf{q}', \lambda'(u')) \cdot h' \equiv h \cdot (\mathsf{p}', \mathsf{q}', \lambda'(u')) \cdot (\mathsf{p}, \mathsf{q}, \lambda(u)) \cdot h'$$

  if $\mathsf{p} \neq \mathsf{p}'$ or $\mathsf{q} \neq \mathsf{q}'$.
- The equivalence on message queues induces an equivalence on named queues in the expected way:

$$h \equiv h' \ \text{ implies } \ \mathsf{s} : h \equiv \mathsf{s} : h'$$