Branching Input Strongest Typing
Judgment

\[ P = a?\{l_j(x_j) = P_j\}_{j \in J} \]

6 Let \( \emptyset; \Gamma^*_1 \cup \{x_j\}_1; \emptyset; \Gamma_{(j)_u} \vdash_R P_j \) be the strongest judgment for \( P_j \). (By induction, \( \exists! \Gamma^*_1 \))

6 Let \( \Delta_1; \Gamma_1; \Delta_u; \Gamma_u \vdash_R P \) be any judgment for process \( P \). (R-INP) \( \Rightarrow \Delta_1 = \{a\}_1, \Gamma_1 = \Gamma^*_1, \Delta_u = \emptyset \) and \( \forall j \in J, \emptyset; \Gamma^*_1 \cup \{x_j\}_1; \emptyset; \Gamma_u \cup \{x_j\}_u \vdash_R P_j \).

6 By induction hypothesis : \( \forall j \in J, \Gamma_{(j)_u} \subseteq \Gamma_u \cup \{x_j\}_u \).

6 Therefore : \( \bigcup_{j \in J} (\Gamma_{(j)_u} \setminus \{x_j\}_u) \subseteq \Gamma_u \).

6 The strongest judgment for \( P \) is then : \( \{a\}_1; \emptyset; \bigcup_{j \in J} (\Gamma_{(j)_u} \setminus \{x_j\}_u) \vdash_R a?\{l_j(x_j) = P_j\}_{j \in J} \)
What Is TyCO, After All?

Maxime Gamboni

EPFL
Asynchronous $\pi$-Calculus

Basic Components

- **Names**: $a, b, c, x \ldots$
- **Processes**: $P, Q, \ldots$

Processes use names as *channels* for sending or receiving data

- Sending $x$ on $a$: $a!x$
- Receiving $x$ on $a$ (and then processing it in $P$): $a?(x).P$
Processes can be combined using *parallel composition*

\[ P | Q \]

Example:

\[ (a!x) | (a?(y).yb) \rightarrow 0|x!b \]

Names can be *restricted* to a part of a process

\[ ((\nu x) P) | Q : x \text{ is visible in } P \text{ but not (directly) in } Q \]

\[ ((\nu x) a!x) | (a?(y).yb) \rightarrow (\nu x) (0|x!b) : P \text{ sends } x \text{ on a name visible to } Q, \text{ which extrudes the scope of } x. \]
Names sent on a channel can be labeled:

- in \( a!v \),

\[
v \ ::= \begin{cases} \text{a name} \\ \text{labeled value} \end{cases}
\]

- Labels are not names, they are just labels.

A case destructor can be used when receiving a labeled value:

- case \( v \) of \( \{ l_j(x_j) = P_j \} \) \( j \in J \)
What is TyCO?

With slight syntactic changes, TyCO is just a sub-calculus of $\pi^V_a$:

- Any output must be with a name having a single label (no label-nesting)
- At input time the case destruction is done immediately.
- Additionally, instead of writing $a?(v).\text{case } v \text{ of } \{l_j(x_j) = P_j\}_{j \in J}$ like we would in $\pi^V_a$, we write $a?\{l_j(x_j) = P_j\}_{j \in J}$, which illustrates the atomicity of input and case-destruction.
Example: Church-Encoding of Natural Numbers

\[ Zero(x) \overset{\text{def}}{=} x?\{q(a)=a!z\langle x\rangle\} \] (let \( x \) be the number zero)

\[ Succ(y, x) \overset{\text{def}}{=} x?\{q(a)=a!s\langle y\rangle\} \] (let \( x \) be the successor of \( y \))

\[ Add(x, y, z) \overset{\text{def}}{=} \]
\[ x!q(\nu a).a?\{z(b)=z!a\langle y\rangle, \]
\[ s(b)=(\nu t) Add(b, y, t)| \]
\[ t?\{a(n)=z!a(\nu r).Succ(n, r)\} \]
Is $\pi^V_a$ More Expressive than TyCO?

- TyCO being a sub-calculus of $\pi^V_a$, an encoding of TyCO into $\pi^V_a$ is straightforward.
- Is it possible however to make an encoding of $\pi^V_a$ into TyCO?
- We need to encode nested variants as single-level variants and break the input / variant-destruction atomicity.
- The encoding needs to respect the process equivalences, i.e. $P \mathcal{R} Q \Leftrightarrow [P] \mathcal{R} [Q]$
To write a description of TyCO-$\pi_a^V$ encoding and prove it is valid

My guide @ EPFL provided me with a $\pi_a^V$-TyCO encoding

The goal of my project is to prove that it is valid and maybe to make necessary changes to it

If I have enough time, then study whether Non-Uniform TyCO can be encoded into $\pi_a^V$