# Structural Equivalence in Reversible Calculus of Communicating Systems (Abstract)

Clément Aubert, Ioana Cristescu

# Structural Equivalence in Reversible Calculus of Communicating Systems (Abstract)

**Clément Aubert**
School of Computer and Cyber Sciences, Augusta University, USA

**Ioana Cristescu**
Inria Rennes, France

"In a process-algebraic approach to system verification, one typically writes two specifications. One, call it SYS, captures the design of the actual system and the other, call it SPEC, describes the system's desired 'high-level' behavior. One may then establish the correctness of SYS with respect to SPEC by showing that SYS behaves the 'same as' SPEC." ([2, p. V])

The approach described is used for the calculus of communicating systems (CCS) [5], an important actor in process calculus, a branch of Computer Science that formally models concurrent systems. However, this approach was not adopted when the reversible calculus of communicating systems (RCCS) [3, 4] was defined. Our current investigation aims at understanding if there is a fundamental reason not to do so, and this lead us to also question how a *syntactic* equivalence used to go from $\text{SYS}_{\text{CCS}}$ to $\text{SPEC}_{\text{CCS}}$ was defined.

## 1 From $\text{SYS}_{\text{CCS}}$ to $\text{SPEC}_{\text{CCS}}$

CCS processes are defined using Backus–Naur form with simple operators (parallel composition, name prefixing or action, choice or sum, recursion, restriction and renaming), a single empty process denoted 0, and a collection of (co-)names $a, \overline{a}, b, \overline{b}, \ldots$ over which $\lambda$ ranges:

▶ **Definition 1** (CCS process). *The set of CCS processes is inductively defined:*

$$P, Q \coloneqq P \mid Q \ \parallel\ \lambda.P \ \parallel\ \sum_{i \in I} P_i \ \parallel\ A \ \parallel\ P\backslash a \ \parallel\ P[a \leftarrow b] \ \parallel\ 0 \qquad \text{(CCS Processes)}$$

*where $A$ are (recursive) definitions of processes, that is $A \overset{def}{=} P$ and $A$ can occur in $P$, $P[a \leftarrow b]$ is the capture-avoiding substitution, and we write $P[\overrightarrow{a} \leftarrow \overrightarrow{b}]$ for $P[a_1 \leftarrow b_1, \ldots, a_n \leftarrow b_n]$.*

To create $\text{SYS}_{\text{CCS}}$, the processes are endowed with an evaluation mechanism given by the labeled transition system (LTS) of Fig. 1. A process $P$ reduces, or evaluates, to $P'$ with label $\alpha$ (which can be a (co-)name, or the special label $\tau$ for "silent", internal, transitions) if a tree whose root is $P \xrightarrow{\alpha} P'$ can be derived using the rules of Fig. 1.

This specification, although capturing the intended calculus, is cumbersome because of its syntactical rigidity: everything has to be spelled out rigorously, and basics properties like the commutativity of the product (i.e., writing $P_1 \mid P_2$ or $P_1 \mid P_2$ should not make a difference)

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \text{com.}_1 \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ \text{com.}_2 \qquad \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \text{syn.}$$

$$\frac{}{\lambda.P \xrightarrow{\lambda} P}\ \text{act.} \qquad \frac{P_j \xrightarrow{\alpha} P_j' \quad j \in I}{\sum_{i \in I} P_i \xrightarrow{\alpha} P_i'}\ \text{sum.} \qquad \frac{P \xrightarrow{\alpha} P' \quad A \overset{\text{def}}{=} P}{A \xrightarrow{\alpha} P'}\ \text{rec.}$$

$$\frac{P \xrightarrow{\alpha} P' \quad a \notin \alpha}{P\backslash a \xrightarrow{\alpha} P'\backslash a}\ \text{res.} \qquad \frac{P \xrightarrow{\alpha} P'}{P[\overrightarrow{a} \leftarrow \overrightarrow{b}] \xrightarrow{\alpha[\overrightarrow{a} \leftarrow \overrightarrow{b}]} P'[\overrightarrow{a} \leftarrow \overrightarrow{b}]}\ \text{rel.}$$

**Figure 1** Rules of the labeled transition system of CCS ($\text{LTS}_{\text{CCS}}$)

are impossible to prove. The solution is to define an equivalence relation on processes, define another specification using it—$\text{SPEC}_{\text{CCS}}$—, and then to prove that they coincide.

▶ **Definition 2** (CCS Structural equivalence). *Structural equivalence on processes is the smallest equivalence relation generated by the following rules:*

$$\begin{array}{ccc}
P \mid Q \equiv Q \mid P & (P \mid Q) \mid V \equiv P \mid (Q \mid V) & P \mid 0 \equiv P \\
P + Q \equiv Q + P & (P + Q) + V \equiv P + (Q + V) & P + 0 \equiv P \\
(P \backslash a) \mid Q \equiv (P \mid Q)\backslash a \text{ with } a \notin \text{fn}(Q) & (P\backslash a)\backslash b \equiv (P \backslash b)\backslash a & \\
A \overset{def}{=} P \Rightarrow A \equiv P & P =_\alpha Q \Rightarrow P \equiv Q &
\end{array}$$

*Where* $\text{fn}(Q)$ *is the set of free names in* $Q$*, and* $=_\alpha$ *is the* $\alpha$*-equivalence given by capture-free substitution.*

$\text{SPEC}_{\text{CCS}}$ is then defined by adding the rule $\dfrac{P_1' \equiv P_1 \qquad P_1 \xrightarrow{\alpha} P_2 \qquad P_2 \equiv P_2'}{P_1' \xrightarrow{\alpha} P_2'}$ con. to $\text{LTS}_{\text{CCS}}$, and removing com1. (or com2.), rec. and rel. from it.

▶ **Lemma.** *If* $P \xrightarrow{\alpha} P'$ *with* $\mathit{SYS}_{\mathit{CCS}}$ *and* $P \equiv Q$ *then* $Q \xrightarrow{\alpha} Q'$ *with* $\mathit{SPEC}_{\mathit{CCS}}$ *and* $P' \equiv Q'$*.*

## 2 Disentangling $\text{SYS}_{\text{RCCS}}$ from $\text{SPEC}_{\text{RCCS}}$

Reversible systems have the possibility of backtracking to return to some previous state. Implementing reversibility in a programming language often requires a mechanism to record the history of the execution. Ideally, this history should be *complete*, so that every forward step can be backtracked, and *minimal*, so that only the relevant information is saved. Concurrent programming languages have additional requirements: the history should be *distributed*, to avoid centralization, and should prevent steps that required a sychronization with other parts of the program to backtrack without undoing this synchronization.

To fulfill those requirements, RCCS uses *memories* attached to the threads of a process.

$$\begin{aligned}
T &:= m \rhd P && \text{(Reversible Thread)} \\
R, S &:= T \;\parallel\; R \mid S \;\parallel\; R\backslash a && \text{(RCCS Processes)}
\end{aligned}$$

for $m$ a memory stack (whose definition is not included here, but that can be find in, e.g., [1]), and $P$ a CCS process.

A forward *and* backward LTS is then be defined, but it is extremely limited: any transition leading to $m \rhd (P_1 \mid P_2)$ is blocked. Indeed, to fulfill our requirement to distribute as much as possible the memories, the system needs to block this execution, and to be given an equivalence rule $m \rhd (P_1 \mid P_2) \equiv (\Upsilon.m \rhd P_1) \mid (\Upsilon.m \rhd P_2)$ that distributes the computation (and the memory, prepended with a special "fork" symbol $\Upsilon$) over two units of computation.

There are two perspectives on this: **1.** We can consider that the equivalence relation now becomes part of the definition of $\text{SYS}_{\text{RCCS}}$, and that there are no $\text{SPEC}_{\text{RCCS}}$, **2.** We can consider that $\text{SPEC}_{\text{RCCS}}$ is strictly more expressive than $\text{SYS}_{\text{RCCS}}$.

Both approaches are somehow problematic since we are losing a way of syntactically assessing that the equivalence relation is "the right one", and have to assume that it is reasonable. But, looking back, the way CCS asses that its equivalence relation is "the right one" is somehow circular, since our Lemma uses the equivalence "on the outside" *and* "on the inside". One of our current investigation is to precisely pinpoint why is this equivalence in CCS acceptable, and to see if the criteria we identify can be imported to RCCS.

## References

1 Clément Aubert and Ioana Cristescu. Contextual equivalences in configuration structures and reversibility. *J. Log. Algebr. Methods Program.*, 86(1):77–106, 2017. doi:10.1016/j.jlamp.2016.08.004.

2 J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. Elsevier Science, Amsterdam, 2001. doi:10.1016/B978-044482830-9/50017-5.

3 Vincent Danos and Jean Krivine. Reversible communicating systems. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR*, volume 3170 of *LNCS*, pages 292–307. Springer, 2004. doi:10.1007/978-3-540-28644-8_19.

4 Vincent Danos and Jean Krivine. Transactions in RCCS. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3653 of *LNCS*, pages 398–412. Springer, 2005. doi:10.1007/11539452_31.

5 Robin Milner. *A Calculus of Communicating Systems*. LNCS. Springer-Verlag, 1980. doi:10.1007/3-540-10235-3.