Structural Equivalence in Reversible Calculus of Communicating Systems (Abstract)
Clément Aubert, Ioana Cristescu

To cite this version:
Clément Aubert, Ioana Cristescu. Structural Equivalence in Reversible Calculus of Communicating Systems (Abstract). 2020. hal-02118036v2

HAL Id: hal-02118036
https://hal.science/hal-02118036v2
Preprint submitted on 26 Dec 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
The formalization of process algebras usually starts with a minimal core of operators and rules for its transition system, and then relax the system to improve its usability and ease the proofs. In the calculus of communicating systems (CCS), the structural congruence plays this role by making e.g. parallel composition commutative and associative: without it, the system would be cumbersome to use and reason about, and it can be proven that this change is innocuous in a precise technical sense. For the two reversible calculi extending CCS, the situation is less clear: CCS with Communication Keys (CCSK) was first defined without any structural congruence, and then was endowed with a fragment of CCS’s congruence. Reversible CCS (RCCS) made the choice of “backing in” the structural equivalence, that became part of the “minimal core” of the system. In this short oral communication, we would like to re-consider the status and role of the structural congruence in general, to question its role in RCCS in particular, and to ask the more general question of the structural equivalences legitimacy.

1 Introduction

It is often useful to have a look at the fundamentals of your research field. For instance, a classical textbook states that constructing a process algebra often uses a two-stages approach [4, p. V]:

“...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.”

This is generally the approach taken to define the calculus of communicating systems (CCS) [9].

Stage 1: SYS_{CCS}  CCS processes are defined using 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:

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

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

To create SYS_{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
Where fn(τ) denotes the set of free names in τ, and \(\alpha\) is the \(\alpha\)-equivalence given by capture-free substitution.

The SYS\(_{CCS}\) specification, although capturing the intended calculus, is cumbersome because of its syntactical rigidity: everything has to be spelled out rigorously, and basic properties like the commutativity of the product (i.e., writing \(P_1 \parallel P_2\) or \(P_1 \parallel P_2\), \(P_1 \parallel P_2\) should not make a difference) are cumbersome to prove. The solution is to define an equivalence relation on processes, define another specification using it—SPEC\(_{CCS}\)—, and then to prove that they coincide.

**Definition 1 (CCS Structural equivalence)** Structural equivalence on processes is the smallest equivalence relation—sometimes “closed under context”—generated by the following rules:

\[
\begin{align*}
P | Q \equiv Q | P \\
P + Q \equiv Q + P \\
(P \setminus a) | Q \equiv (P | Q) \setminus a \text{ with } a \notin \text{fn}(Q) \\
A \equiv P \Rightarrow A \equiv P
\end{align*}
\]

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

SPEC\(_{CCS}\) is then defined by adding the rule \(P_1 \equiv P_1 \alpha \Rightarrow P_2 \equiv P_2\) con. to LTS\(_{CCS}\), and removing com.1 (or com.2), rec. and rel. from it. Finally, the correctness of SPEC\(_{CCS}\) is proven by showing that it is a “conservative extension of SYS\(_{CCS}\)”, in the following sense:

**Lemma 1** If \(P \alpha \rightarrow P'\) with SYS\(_{CCS}\) and \(P \equiv Q\) then \(Q \alpha \rightarrow Q'\) with SPEC\(_{CCS}\) and \(P' \equiv Q'\).

This lemma guarantees that reasoning modulo the structural congruence does not alter the correctness of the system, and that we don’t have to track small and innocuous modifications in the terms under study—it just cuts the red tape.

### 2 What About Reversible Calculi?

There are two ways of extending CCS to reversible calculus: the “static” approach “marks” each occurrence of an action, and gave birth to CCS with Communication Keys (CCSK) [11]. The “dynamic” approach uses memories attached to the threads of a process, and gave Reversible CCS (RCCS) [5, 6]. Both calculi evolved over the time, notably regarding structural equivalence.
CCSK  Some versions of CCSK “does not exploit any structural congruence”\(^1\) [11], while some other [7] uses a structural congruence that contains a significant subset of the one from CCS, plus a rule to deal with keys [7, p. 105].

To our knowledge, the innocuity of this latter congruence has not been proven, but it was shown that a model based on reversible bundle structure was “stable” [7, Proposition 4.9] under it.

RCCS  RCCS’ congruence evolved as well, from containing additive structural congruence but not the \(\alpha\)-equivalence [5, Section 2.3] to being pretty much only the \(\alpha\)-equivalence [2]. But in any case, RCCS does not benefit from the clear distinction we discussed at the beginning of this paper. To understand why, let us explain how this system was designed.

In RCCS, reversibility is implemented thanks to memories that are complete—so that every forward step can be backtracked—, minimal—so that only the relevant information is saved—, but also distributed—to avoid centralization. Technically, memories are attached to the threads of a process.

\[
T := m |> P \\
R,S := T \parallel R \parallel S \parallel R \setminus a
\]

(Reversible Thread)  
(RCCS Processes)

for \(m\) a memory stack (whose definition [2] is not included here), and \(P\) a CCS process.

A forward and backward LTS is then defined, but it is extremely limited: any transition leading to \(m |> (P_1 \parallel P_2)\) 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

\[
m |> (P_1 \parallel P_2) \equiv (\gamma . m |> P_1) \parallel (\gamma . m |> P_2)
\]

(Distribution of memory)

that distributes the computation (and the memory, prepended with a special “fork” symbol \(\gamma\)) 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 (which generally contains more than just Distribution of memory) is “the right one”, and have to assume that it is reasonable. The reduction semantics of CCS can also be considered to be \(\text{SPEC}_{\text{CCS}}\), so we loose the clear distinction between the two (reduction semantics and LTS) in RCCS. Furthermore, to prove that RCCS and CCSK are equi-expressive [8], the authors had to remove from the rest of the LTS the possibility to use Distribution of memory between two transitions [8, Lemma 4], making their proof actually harder. All in all, the questions we are asking are,

1. Can we on one hand enrich this congruence (with e.g. some form of commutativity of parallel composition), and on the other extract from it the Distribution of memory rule?

2. What criteria should we use to determine that our newly defined \(\text{SYS}_{\text{RCCS}}\) and \(\text{SPEC}_{\text{RCCS}}\) are correct?

Our first question aims at restoring the conceptual purity of our system and of the structural congruence, but also to provide tools to improve this latter (which is very limited, as of now). Regarding our second question, and similarly to what has been done with CCSK, a model of the memories based on configuration structures has been show to be stable under structural congruence [3, Lemma 43]. However, the congruence considered was very small, and the result applied only to memories, and not to processes as a whole.

\(^1\)As noted in a paper [8] that furthermore note that \(\alpha\)-equivalence can and should be added in any case [8, Remark 2].
3 Conclusion

One of the true puzzle raised by this small introspection is the second question asked previously: while "making up" SYS\textsubscript{RCCS} and SPEC\textsubscript{RCCS} is "easy", gaining evidence that they are the correct systems, and that they are equi-expressive, is harder. The equivalence between CCSK and RCCS, as well as the numerous denotational models for those calculi and their simulation results, are good safe-guards that those models are relevant, and capture reversibility in a satisfactory way. But how can we endow them with "the right" structural congruence?

This investigation actually raises another question, less expected: looking back, the way CCS assesses that its equivalence relation is "the right one" is somehow circular, since our Lemma 1 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 and CCSK. Mechanized methods are likely to be of little help in this domain\textsuperscript{2} as e.g. formalization of CCS in Coq [1] or of the π-calculus in Agda [10] either discard syntactical congruences, or pose them as axioms.

References


\textsuperscript{2}An impressive list of the “mechanized formalizations of the π-calculus (and friends)” can be found at http://why-lambda.blogspot.com/2015/04/mechanized-formalizations-of-pi.html.