Session Type Isomorphisms

Session Type Isomorphisms

Mariangiola Dezani-Ciancaglini
Università di Torino, Italy

Luca Padovani
Università di Torino, Italy

Jovanka Pantovic
Univerzitet u Novom Sadu, Serbia

There has been a considerable amount of work on retrieving functions in function libraries using their type as search key. The availability of rich component specifications, in the form of behavioral types, enables similar queries where one can search a component library using the behavioral type of a component as the search key. Just like for function libraries, however, component libraries will contain components whose type differs from the searched one in the order of messages or in the position of the branching points. Thus, it makes sense to also look for those components whose type is different from, but isomorphic to, the searched one.

In this article we give semantic and axiomatic characterizations of isomorphic session types. The theory of session type isomorphisms turns out to be subtle. In part this is due to the fact that it relies on a non-standard notion of equivalence between processes. In addition, we do not know whether the axiomatization is complete. It is known that the isomorphisms for arrow, product and sum types are not finitely axiomatisable, but it is not clear yet whether this negative results holds also for the family of types we consider in this work.

1 Introduction

We have all experienced, possibly during a travel abroad, using an ATM that behaves differently from the ones we are familiar with. Although the information requested for accomplishing a transaction is essentially always the same – the PIN, the amount of money we want to withdraw, whether or not we want a receipt – we may be prompted to enter such information in an unexpected order, or we may be asked to dismiss sudden popup windows containing informative messages – “charges may apply” – or commercials. Subconsciously, we *adapt* our behavior so that it matches the one of the ATM we are operating, and we can usually complete the transaction provided that the expected and actual behaviors are *sufficiently similar*. An analogous problem arises during software development or execution, when we need a component that exhibits some desired behavior while the components we have at hand exhibit similar, but not exactly equal, behaviors which could nonetheless be adapted to the one we want. In this article, we explore one particular way of realizing such adaptation in the context of binary sessions, where the behavior of components is specified as session types.

There are two key notions to be made precise in the previous paragraph: first of all, we must clarify what it means for two behaviors to be “similar” to the point that one can be adapted into the other; second, as for the “subconscious” nature of adaptation, we translate this into the ability to synthesize the adapter automatically – *i.e.* without human intervention – just by looking at the differences between the required and actual behaviors of the component. Clearly we have to find a trade-off: the coarser the similarity notion is the better, for this means widening the range of components we can use; at the same time, it is reasonable to expect that the more two components differ, the harder it gets to automatically synthesize a sensible adapter between them. The methodology we propose in this work is based on the theory of *type isomorphisms* [10]. Intuitively, two types T and S are isomorphic if there exist two adapters $A : T \to S$ and $B : S \to T$ such that A transforms a component of type (or, that behaves like) T into one of type S, and B does just the opposite. It is required that these transformations must not *lose any information*. This

Alastair F. Donaldson, Vasco Vasconcelos (Eds.): Proceedings
of the 7th Workshop on Programming Language Approaches
to Concurrency and Communication-cEntric Software (PLACES 2014)
EPTCS 155, 2014, pp. 61–71, doi:10.4204/EPTCS.155.9

can be expressed saying that if we compose A and B in any order they annihilate each other, that is we obtain adapters $A \|\!\lceil B : T \to T$ and $B \|\!\lceil A : S \to S$ that are equivalent to the “identity” trasformations on T and S respectively.

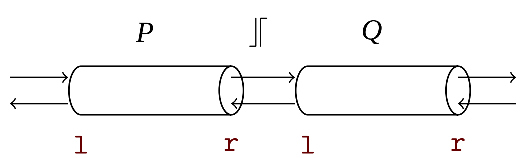
In the following we formalize these concepts: we define syntax and semantics of processes as well as a notion of process equivalence (Section 2). Next, we introduce a type system for processes, the notion of session type isomorphism, and show off samples of the transformations we can capture in this framework (Section 3). We conclude with a quick survery of related works and open problems (Section 4).

2 Processes

We let m, n, $\dots$ range over integer numbers; we let c range over the set $\{\mathtt{l},\mathtt{r}\}$ of *channels* and ℓ range over the set $\{\mathtt{inl},\mathtt{inr}\}$ of *selectors*. We define an involution $\overline{\cdot}$ over channels such that $\overline{\mathtt{l}} = \mathtt{r}$. We assume a set of *basic values* v, $\dots$ and *basic types* t, s, $\dots$ that include the unitary value $\mathtt{()}$ of type $\mathtt{unit}$, the booleans $\mathtt{true}$ and $\mathtt{false}$ of type $\mathtt{bool}$, and the integer numbers of type $\mathtt{int}$. We write $\mathsf{v} \in t$ meaning that v has type t. We use a countable set of *variables* x, y, $\dots$; *expressions* e, $\dots$ are either variables or values or the equality $\mathsf{e}_1 = \mathsf{e}_2$ between two expressions. Additional expression forms can be added without substantial issues. *Processes* are defined by the grammar

$$P \;::=\; \mathbf{0} \;\mid\; \mathsf{c}?(x:t).P \;\mid\; \mathsf{c}!\langle\mathsf{e}\rangle.P \;\mid\; \mathsf{c}\triangleleft\ell.P \;\mid\; \mathsf{c}\triangleright\{P,Q\} \;\mid\; \mathtt{if}\ \mathsf{e}\ \mathtt{then}\ P\ \mathtt{else}\ Q \;\mid\; P \|\!\lceil Q$$

which includes the terminated process $\mathbf{0}$, input $\mathsf{c}?(x:t).P$ and output $\mathsf{c}!\langle\mathsf{e}\rangle.P$ processes, as well as labeled-driven selection $\mathsf{c}\triangleleft\ell.P$ and branching $\mathsf{c}\triangleright\{P,Q\}$, the conditional process $\mathtt{if}\ \mathsf{e}\ \mathtt{then}\ P\ \mathtt{else}\ Q$, and parallel composition $P \|\!\lceil Q$. The peculiarity of the calculus is that communication occurs only between adjacent processes. Such communication model is exemplified by the diagram below which depicts the composition $P \|\!\lceil Q$. Each process sends and receives messages through the channels $\mathtt{l}$ and $\mathtt{r}$. Messages sent by P on $\mathtt{r}$ are received by Q from $\mathtt{l}$, and messages sent by Q on $\mathtt{l}$ are received by P from $\mathtt{r}$. Therefore, unlike more conventional parallel composition operators, $\|\!\lceil$ is associative but not symmetric in general. Intuitively, $P \|\!\lceil Q$ models a binary session where P and Q are the processes accessing the two endpoints of the session. By compositionality, we can also represent more complex scenarios like $P \|\!\lceil A \|\!\lceil Q$ where the interaction of the same two processes P and Q is mediated by an adapter A that filters and/or transforms the messages exchanged between P and Q. In turn, A may be the parallel composition of several simpler adapters.



The operational semantics of processes is formalized as a reduction relation closed by reduction contexts and a structural congruence relation. *Reduction contexts* $\mathscr{C}$ are defined by the grammar

$$\mathscr{C} \;::=\; [\,] \;\mid\; \mathscr{C} \|\!\lceil P \;\mid\; P \|\!\lceil \mathscr{C}$$

and, as usual, we write $\mathscr{C}[P]$ for the process obtained by replacing the hole in $\mathscr{C}$ with P.

Structural congruence is the least congruence defined by the rules

$$\mathbf{0} \|\!\lceil \mathbf{0} \equiv \mathbf{0} \qquad P \|\!\lceil (Q \|\!\lceil R) \equiv (P \|\!\lceil Q) \|\!\lceil R$$

while *reduction* is the least relation $\longrightarrow$ defined by the rules in Table 1. The rules are familiar and therefore unremarkable. We assume a deterministic *evaluation* relation $\mathsf{e} \downarrow \mathsf{v}$ expressing the fact that v

Table 1: Reduction relation.

$$\text{[R-COMM 1]}\ \frac{\mathsf{e}\downarrow\mathsf{v} \qquad \mathsf{v}\in t}{\mathtt{r}!\langle\mathsf{e}\rangle.P \mathbin{]\![} \mathtt{l}?(x:t).Q \longrightarrow P \mathbin{]\![} Q\{\mathsf{v}/x\}} \qquad \text{[R-COMM 2]}\ \frac{\mathsf{e}\downarrow\mathsf{v} \qquad \mathsf{v}\in t}{\mathtt{r}?(x:t).P \mathbin{]\![} \mathtt{l}!\langle\mathsf{e}\rangle.Q \longrightarrow P\{\mathsf{v}/x\} \mathbin{]\![} Q}$$

$$\text{[R-CHOICE 1]}\ \mathtt{r}\triangleleft\ell.P \mathbin{]\![} \mathtt{l}\triangleright\{Q_{\mathtt{inl}},Q_{\mathtt{inr}}\} \longrightarrow P \mathbin{]\![} Q_\ell \qquad \text{[R-CHOICE 2]}\ \mathtt{r}\triangleright\{P_{\mathtt{inl}},P_{\mathtt{inr}}\} \mathbin{]\![} \mathtt{l}\triangleleft\ell.Q \longrightarrow P_\ell \mathbin{]\![} Q$$

$$\text{[R-COND]}\ \frac{\mathsf{e}\downarrow\mathsf{v} \qquad \mathsf{v}\in\mathtt{bool}}{\mathtt{if}\ \mathsf{e}\ \mathtt{then}\ P_{\mathtt{true}}\ \mathtt{else}\ P_{\mathtt{false}} \longrightarrow P_{\mathsf{v}}} \qquad \text{[R-CONTEXT]}\ \frac{P\longrightarrow Q}{\mathscr{C}[P]\longrightarrow\mathscr{C}[Q]} \qquad \text{[R-STRUCT]}\ \frac{P\equiv P' \qquad P'\longrightarrow Q' \qquad Q'\equiv Q}{P\longrightarrow Q}$$

is the value of e. We write $\longrightarrow^*$ for the reflexive, transitive closure of $\longrightarrow$ and $P \not\longrightarrow$ if there is no Q such that $P \longrightarrow Q$. With these notions we can characterize the set of correct processes, namely those that complete every interaction and eventually reduce to $\mathbf{0}$:

Definition 1 (correct process)**.** We say that a process P is *correct* if $P \longrightarrow^* Q \not\longrightarrow$ implies $Q \equiv \mathbf{0}$.

A key ingredient of our development is a notion of process equivalence that relates two processes P and Q whenever they can be completed by the same contexts $\mathscr{C}$ to form a correct process. Formally:

Definition 2 (equivalence)**.** We say that two processes P and Q are *equivalent*, notation $P \approx Q$, whenever for every $\mathscr{C}$ we have that $\mathscr{C}[P]$ is correct if and only if $\mathscr{C}[Q]$ is correct.

Note that the relation $\approx$ differs from more conventional equivalences between processes. In particular, $\approx$ is insensitive to the exact time when visible actions are made available on the two interfaces of a process. For example, we have

$$\mathtt{l}?(x:\mathtt{int}).\mathtt{r}!\langle\mathtt{true}\rangle.\mathtt{l}?(y:\mathtt{unit}) \approx \mathtt{l}?(x:\mathtt{int}).\mathtt{l}?(y:\mathtt{unit}).\mathtt{r}!\langle\mathtt{true}\rangle \tag{1}$$

despite the fact that the two processes perform visible actions in different orders. Note that the processes in (1) are not (weakly) bisimilar.

3 Type System and Isomorphisms

Session types $T, S, \ldots$ are defined by the grammar

$$T \quad ::= \quad \mathtt{end} \quad | \quad ?t.T \quad | \quad !t.T \quad | \quad T+S \quad | \quad T\oplus S$$

and are fairly standard, except for branching $T+S$ and selection $T\oplus S$ which are binary instead of n-ary operators, consistently with the process language. As usual, we denote by $\overline{T}$ the *dual* of T, namely the session type obtained by swapping inputs with outputs and selections with branches in T.

We let Γ range over *environments* which are finite maps from variables to types of the form

$$x_1 : t_1, \ldots, x_n : t_n.$$

The typing rules are given in Table 2. Judgments have the form:

- $\Gamma \vdash \mathsf{e} : t$ stating that e is well typed and has type t in the environment Γ and

Table 2: Typing rules for expressions and processes.

$$[\textsc{T-var}]\quad \Gamma, x : t \vdash x : t \qquad [\textsc{T-value}]\ \frac{\mathsf{v} \in t}{\Gamma \vdash \mathsf{v} : t} \qquad [\textsc{T-eq}]\ \frac{\Gamma \vdash \mathsf{e}_1 : t \qquad \Gamma \vdash \mathsf{e}_2 : t}{\Gamma \vdash \mathsf{e}_1 = \mathsf{e}_2 : \mathtt{bool}} \qquad [\textsc{T-input}]\ \frac{\Gamma, x : t \vdash P \blacktriangleright \{\mathsf{c} : T, \bar{\mathsf{c}} : S\}}{\Gamma \vdash \mathsf{c}?(x : t).P \blacktriangleright \{\mathsf{c} : ?t.T, \bar{\mathsf{c}} : S\}}$$

$$[\textsc{T-output}]\ \frac{\Gamma \vdash \mathsf{e} : t \qquad \Gamma \vdash P \blacktriangleright \{\mathsf{c} : T, \bar{\mathsf{c}} : S\}}{\Gamma \vdash \mathsf{c}!\langle \mathsf{e} \rangle.P \blacktriangleright \{\mathsf{c} : !t.T, \bar{\mathsf{c}} : S\}} \qquad [\textsc{T-branch}]\ \frac{\Gamma \vdash P_i \blacktriangleright \{\mathsf{c} : T_i, \bar{\mathsf{c}} : S\}\ ^{(i=1,2)}}{\Gamma \vdash \mathsf{c} \triangleright \{P_1, P_2\} \blacktriangleright \{\mathsf{c} : T_1 + T_2, \bar{\mathsf{c}} : S\}}$$

$$[\textsc{T-select left}]\ \frac{\Gamma \vdash P \blacktriangleright \{\mathsf{c} : T_1, \bar{\mathsf{c}} : S\}}{\Gamma \vdash \mathsf{c} \triangleleft \mathtt{inl}.P \blacktriangleright \{\mathsf{c} : T_1 \oplus T_2, \bar{\mathsf{c}} : S\}} \qquad [\textsc{T-select right}]\ \frac{\Gamma \vdash P \blacktriangleright \{\mathsf{c} : T_2, \bar{\mathsf{c}} : S\}}{\Gamma \vdash \mathsf{c} \triangleleft \mathtt{inr}.P \blacktriangleright \{\mathsf{c} : T_1 \oplus T_2, \bar{\mathsf{c}} : S\}} \qquad [\textsc{T-idle}]\ \Gamma \vdash \mathbf{0} \blacktriangleright \{\mathtt{l} : \mathtt{end}, \mathtt{r} : \mathtt{end}\}$$

$$[\textsc{T-conditional}]\ \frac{\Gamma \vdash \mathsf{e} : \mathtt{bool} \qquad \Gamma \vdash P_i \blacktriangleright \{\mathtt{l} : T, \mathtt{r} : S\}\ ^{(i=1,2)}}{\Gamma \vdash \mathtt{if}\ \mathsf{e}\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2 \blacktriangleright \{\mathtt{l} : T, \mathtt{r} : S\}} \qquad [\textsc{T-parallel}]\ \frac{\Gamma \vdash P \blacktriangleright \{\mathtt{l} : T, \mathtt{r} : T'\} \qquad \Gamma \vdash Q \blacktriangleright \{\mathtt{l} : \overline{T}', \mathtt{r} : S\}}{\Gamma \vdash P \mathbin{]\![} Q \blacktriangleright \{\mathtt{l} : T, \mathtt{r} : S\}}$$

- $\Gamma \vdash P \blacktriangleright \{\mathsf{c} : T, \bar{\mathsf{c}} : S\}$ stating that P is well typed in the environment Γ and uses channel c according to T and $\bar{\mathsf{c}}$ according to S.

Theorem 1. If $\vdash P \blacktriangleright \{\mathtt{l} : \mathtt{end}, \mathtt{r} : \mathtt{end}\}$, then P is correct.

Proof. Looking at the typing rules it is clear that P can only be $\mathbf{0}$, or a conditional or a parallel composition. The first two case are immediate. In the third case let P be $P_1 \mathbin{]\![} \dots \mathbin{]\![} P_i \mathbin{]\![} \dots \mathbin{]\![} P_n$, where $P_1, \dots, P_i, \dots, P_n$ are single-threaded. Then rule [T-PARALLEL] requires

$$\vdash P_1 \blacktriangleright \{\mathtt{l} : \mathtt{end}, \mathtt{r} : T_1\}, \vdash P_i \blacktriangleright \{\mathtt{l} : \overline{T_{i-1}}, \mathtt{r} : T_i\} \text{ for } 2 \le i \le n-1 \text{ and } \vdash P_n \blacktriangleright \{\mathtt{l} : \overline{T_{n-1}}, \mathtt{r} : \mathtt{end}\}$$

for some types $T_1, \dots, T_{n-1}$. The proof is by induction on $T_1, \dots, T_{n-1}$. The first step coincides with the first case. For the induction step we can assume that $P_1, \dots, P_i, \dots, P_n$ are not conditionals, since otherwise at least one of them could be reduced by rule [R-COND]. Notice that $\mathtt{r}$ is the only channel in P_1 and $\mathtt{l}$ is the only channel in P_n. Then there must be at least one index j $(1 \le j \le n-1)$ such that P_j starts with a communication/selection/branching on channel $\mathtt{r}$ and P_{j+1} starts with a communication/selection/branching on channel $\mathtt{l}$. We only consider the case $T_j = T_{\mathtt{inl}} \oplus T_{\mathtt{inr}}$, the proofs for the other cases being similar. Rules [T-SELECT LEFT], [T-SELECT RIGHT] and [T-BRANCH] require $P_j \equiv \mathtt{r} \triangleleft \ell.Q$ and $P_{j+1} \equiv \mathtt{l} \triangleright \{Q_{\mathtt{inl}}, Q_{\mathtt{inr}}\}$. Therefore $P \longrightarrow P_1 \mathbin{]\![} \dots \mathbin{]\![} Q \mathbin{]\![} Q_\ell \mathbin{]\![} \dots \mathbin{]\![} P_n$ by rules [R-CHOICE 1] and [R-CONTEXT]. This concludes the proof, since $\vdash Q \blacktriangleright \{\mathtt{l} : \overline{T_{j-1}}, \mathtt{r} : T_\ell\}, \vdash Q_\ell \blacktriangleright \{\mathtt{l} : \overline{T_\ell}, \mathtt{r} : T_{j+1}\}$. □

To have an isomorphism between two session types T and S, we need a process A that behaves according to $\overline{T}$ on its left interface and according to S on its right interface. In this way, the process "transforms" T into S. Symmetrically, there must be a process B that performs the inverse transformation. Not all of these transformations are isomorphisms, because we also require that these transformations must not entail any *loss of information*. Given a session type T, the simplest process with this property is the *identity* process id_T defined below:

$$\mathsf{id}_{\mathtt{end}} = \mathbf{0} \qquad \begin{aligned} \mathsf{id}_{!t.T} &= \mathtt{l}?(x : t).\mathtt{r}!\langle x \rangle.\mathsf{id}_T \\ \mathsf{id}_{?t.T} &= \mathtt{r}?(x : t).\mathtt{l}!\langle x \rangle.\mathsf{id}_T \end{aligned} \qquad \begin{aligned} \mathsf{id}_{T \oplus S} &= \mathtt{l} \triangleright \{\mathtt{r} \triangleleft \mathtt{inl}.\mathsf{id}_T, \mathtt{r} \triangleleft \mathtt{inr}.\mathsf{id}_S\} \\ \mathsf{id}_{T + S} &= \mathtt{r} \triangleright \{\mathtt{l} \triangleleft \mathtt{inl}.\mathsf{id}_T, \mathtt{l} \triangleleft \mathtt{inr}.\mathsf{id}_S\} \end{aligned}$$

Notice that $\vdash \mathsf{id}_T \blacktriangleright \{\mathtt{l} : \overline{T}, \mathtt{r} : T\}$. We can now formalize the notion of session type isomorphism:

Definition 3 (isomorphism)**.** We say that the session types T and S are *isomorphic*, notation $T \cong S$, if there exist two processes A and B such that $\vdash A \blacktriangleright \{\mathtt{l} : \overline{T}, \mathtt{r} : S\}$ and $\vdash B \blacktriangleright \{\mathtt{l} : \overline{S}, \mathtt{r} : T\}$ and $A \,\|\!\!\lceil\, B \approx \mathsf{id}_T$ and $B \,\|\!\!\lceil\, A \approx \mathsf{id}_S$.

Example 1. Let $T \stackrel{\text{def}}{=} \mathtt{!int.!bool.end}$ and $S \stackrel{\text{def}}{=} \mathtt{!bool.!int.end}$ and observe that T and S differ in the order in which messages are sent. Then we have $T \cong S$. Indeed, if we take

$$A \stackrel{\text{def}}{=} \mathtt{l?}(x : \mathtt{int}).\mathtt{l?}(y : \mathtt{bool}).\mathtt{r!}\langle y\rangle.\mathtt{r!}\langle x\rangle.\mathbf{0} \quad \text{and} \quad B \stackrel{\text{def}}{=} \mathtt{l?}(x : \mathtt{bool}).\mathtt{l?}(y : \mathtt{int}).\mathtt{r!}\langle y\rangle.\mathtt{r!}\langle x\rangle.\mathbf{0}$$

we derive $\vdash A \blacktriangleright \{\mathtt{l} : \overline{T}, \mathtt{r} : S\}$ and $\vdash B \blacktriangleright \{\mathtt{l} : \overline{S}, \mathtt{r} : T\}$ and moreover $A \,\|\!\!\lceil\, B \approx \mathsf{id}_T$ and $B \,\|\!\!\lceil\, A \approx \mathsf{id}_S$. ■

Example 2. Showing that two session types are *not* isomorphic is more challenging since we must prove that there is no pair of processes A and B that turns one into the other without losing information. We do so reasoning by contradiction. Suppose for example that $\mathtt{!int.end}$ and $\mathtt{end}$ are isomorphic. Then, there must exist $\vdash A \blacktriangleright \{\mathtt{l} : \mathtt{?int.end}, \mathtt{r} : \mathtt{end}\}$ and $\vdash B \blacktriangleright \{\mathtt{l} : \mathtt{end}, \mathtt{r} : \mathtt{!int.end}\}$. The adapter B is suspicious, since it must send a message of type $\mathtt{int}$ on channel $\mathtt{r}$ without ever receiving such a message from channel $\mathtt{l}$. Then, it must be the case that B "makes up" such a message, say it is n (observe that our calculus is deterministic, so B will always output the same integer n). We can now unmask B showing a context that distinguishes $\mathsf{id}_{\mathtt{!int.end}}$ from $A \,\|\!\!\lceil\, B$. Consider

$$\mathscr{C} \stackrel{\text{def}}{=} \mathtt{r!}\langle n+1\rangle.\mathbf{0} \,\|\!\!\lceil\, [\,] \,\|\!\!\lceil\, \mathtt{l?}(x : \mathtt{int}).\mathtt{if}\ x = n+1\ \mathtt{then}\ \mathbf{0}\ \mathtt{else}\ \mathtt{r!}\langle\mathtt{false}\rangle.\mathbf{0}$$

and observe that $\mathscr{C}[\mathsf{id}_{\mathtt{!int.end}}]$ is correct whereas $\mathscr{C}[A \,\|\!\!\lceil\, B]$ is not because

$$\mathscr{C}[A \,\|\!\!\lceil\, B] \longrightarrow^* \mathbf{0} \,\|\!\!\lceil\, \mathtt{if}\ n = n+1\ \mathtt{then}\ \mathbf{0}\ \mathtt{else}\ \mathtt{r!}\langle\mathtt{false}\rangle.\mathbf{0} \longrightarrow \mathbf{0} \,\|\!\!\lceil\, \mathtt{r!}\langle\mathtt{false}\rangle.\mathbf{0} \not\longrightarrow$$

This means that $A \,\|\!\!\lceil\, B \not\approx \mathsf{id}_{\mathtt{!int.end}}$, contradicting the hypothesis that A and B were the witnesses of the isomorphism $\mathtt{!int.end} \cong \mathtt{end}$. ■

Example 3. Another interesting pair of non-isomorphic types is given by $T \stackrel{\text{def}}{=} \mathtt{?int.!bool.end}$ and $S \stackrel{\text{def}}{=} \mathtt{!bool.?int.end}$. A lossless transformation from S to T can be realized by the process

$$B \stackrel{\text{def}}{=} \mathtt{l?}(x : \mathtt{bool}).\mathtt{r?}(y : \mathtt{int}).\mathtt{r!}\langle x\rangle.\mathtt{l!}\langle y\rangle.\mathbf{0}\,,$$

which reads one message from each interface and forwards it to the opposite one. The inverse transformation from T to S is unachieavable without loss of information. Such process necessarily sends at least one message (of type $\mathtt{int}$ or of type $\mathtt{bool}$) on one interface *before* it receives the message of the same type from the opposite interface. Therefore, just like in Example 2, such process must guess the message to send, and in most cases such message does not coincide with the one the process was supposed to forward. ■

Table 3 gathers the session type isomorphisms that we have identified. There is a perfect duality between the odd-indexed axioms (about outputs/selections, on the left) and the even-indexed axioms (about inputs/branchings, on the right), so we briefly discuss the odd-indexed axioms only. Axiom [A1] is a generalization of the isomorphism discussed in Example 1 and is proved by a similar adapter. Axiom [A3] distributes the *same* output on a selection. Basically, this means that the moment of selection is irrelevant with respect to other adjacent output operations. Axiom [A5] shows that sending the unitary value provides no information and therefore is a superfluous operation. Axiom [A7] shows that sending a boolean value is equivalent to making a selection, provided that the continuation does not depend on the

Table 3: Session type isomorphisms.

[A1]	$!t.!s.T \cong !s.!t.T$	[A2]	$?t.?s.T \cong ?s.?t.T$
[A3]	$!t.(T \oplus S) \cong !t.T \oplus !t.S$	[A4]	$?t.(T+S) \cong ?t.T + ?t.S$
[A5]	$!\mathtt{unit}.T \cong T$	[A6]	$?\mathtt{unit}.T \cong T$
[A7]	$!\mathtt{bool}.T \cong T \oplus T$	[A8]	$?\mathtt{bool}.T \cong T+T$
[A9]	$T \oplus S \cong S \oplus T$	[A10]	$T+S \cong S+T$
[A11]	$(T_1 \oplus T_2) \oplus T_3 \cong T_1 \oplus (T_2 \oplus T_3)$	[A12]	$(T_1+T_2)+T_3 \cong T_1+(T_2+T_3)$

Table 4: Adapters for type isomorphism.

$A_1 = \mathtt{l}?(x:t).\mathtt{l}?(y:s).\mathtt{r}!\langle y\rangle.\mathtt{r}!\langle x\rangle.\mathsf{id}_T$	$B_1 = \mathtt{l}?(x:s).\mathtt{l}?(y:t).\mathtt{r}!\langle y\rangle.\mathtt{r}!\langle x\rangle.\mathsf{id}_T$
$A_2 = \mathtt{r}?(x:t).\mathtt{r}?(y:s).\mathtt{l}!\langle y\rangle.\mathtt{l}!\langle x\rangle.\mathsf{id}_T$	$B_2 = \mathtt{r}?(x:s).\mathtt{r}?(y:t).\mathtt{l}!\langle y\rangle.\mathtt{l}!\langle x\rangle.\mathsf{id}_T$
$A_3 = \mathtt{l}?(x:t).\mathtt{l} \triangleright \{\mathtt{r} \triangleleft \mathtt{inl}.\mathtt{r}!\langle x\rangle.\mathsf{id}_T, \mathtt{r} \triangleleft \mathtt{inr}.\mathtt{r}!\langle x\rangle.\mathsf{id}_S\}$	
$B_3 = \mathtt{l} \triangleright \{\mathtt{l}?(x:t).\mathtt{r}!\langle x\rangle.\mathtt{r} \triangleleft \mathtt{inl}.\mathsf{id}_T, \mathtt{l}?(x:t).\mathtt{r}!\langle x\rangle.\mathtt{r} \triangleleft \mathtt{inr}.\mathsf{id}_S\}$	
$A_4 = \mathtt{r} \triangleright \{\mathtt{r}?(x:t).\mathtt{l}!\langle x\rangle.\mathtt{l} \triangleleft \mathtt{inl}.\mathsf{id}_T, \mathtt{r}?(x:t).\mathtt{l}!\langle x\rangle.\mathtt{l} \triangleleft \mathtt{inr}.\mathsf{id}_S\}$	
$B_4 = \mathtt{r}?(x:t).\mathtt{r} \triangleright \{\mathtt{l} \triangleleft \mathtt{inl}.\mathtt{l}!\langle x\rangle.\mathsf{id}_T, \mathtt{l} \triangleleft \mathtt{inr}.\mathtt{l}!\langle x\rangle.\mathsf{id}_S\}$	
$A_5 = \mathtt{l}?(x:\mathtt{unit}).\mathsf{id}_T$	$B_5 = \mathtt{r}!\langle \mathtt{()}\rangle.\mathsf{id}_T$
$A_6 = \mathtt{l}!\langle \mathtt{()}\rangle.\mathsf{id}_T$	$B_6 = \mathtt{r}?(x:\mathtt{unit}).\mathsf{id}_T$
$A_7 = \mathtt{l}?(x:\mathtt{bool}).\mathtt{if}\ x\ \mathtt{then}\ \mathtt{r} \triangleleft \mathtt{inl}.\mathsf{id}_T\ \mathtt{else}\ \mathtt{r} \triangleleft \mathtt{inr}.\mathsf{id}_T$	$B_7 = \mathtt{l} \triangleright \{\mathtt{r}!\langle \mathtt{true}\rangle.\mathsf{id}_T, \mathtt{r}!\langle \mathtt{false}\rangle.\mathsf{id}_T\}$
$A_8 = \mathtt{r} \triangleright \{\mathtt{l}!\langle \mathtt{true}\rangle.\mathsf{id}_T, \mathtt{l}!\langle \mathtt{false}\rangle.\mathsf{id}_T\}$	$B_8 = \mathtt{r}?(x:\mathtt{bool}).\mathtt{if}\ x\ \mathtt{then}\ \mathtt{l} \triangleleft \mathtt{inl}.\mathsf{id}_T\ \mathtt{else}\ \mathtt{l} \triangleleft \mathtt{inr}.\mathsf{id}_T$
$A_9 = \mathtt{l} \triangleright \{\mathtt{r} \triangleleft \mathtt{inr}.\mathsf{id}_T, \mathtt{r} \triangleleft \mathtt{inl}.\mathsf{id}_S\}$	$B_9 = \mathtt{l} \triangleright \{\mathtt{r} \triangleleft \mathtt{inr}.\mathsf{id}_S, \mathtt{r} \triangleleft \mathtt{inl}.\mathsf{id}_T\}$
$A_{10} = \mathtt{r} \triangleright \{\mathtt{l} \triangleleft \mathtt{inr}.\mathsf{id}_S, \mathtt{l} \triangleleft \mathtt{inl}.\mathsf{id}_T\}$	$B_{10} = \mathtt{r} \triangleright \{\mathtt{l} \triangleleft \mathtt{inr}.\mathsf{id}_T, \mathtt{l} \triangleleft \mathtt{inl}.\mathsf{id}_S\}$
$A_{11} = \mathtt{l} \triangleright \{\mathtt{l} \triangleright \{\mathtt{r} \triangleleft \mathtt{inl}.\mathsf{id}_{T_1}, \mathtt{r} \triangleleft \mathtt{inr}.\mathtt{r} \triangleleft \mathtt{inl}.\mathsf{id}_{T_2}\}, \mathtt{r} \triangleleft \mathtt{inr}.\mathtt{r} \triangleleft \mathtt{inr}.\mathsf{id}_{T_3}\}$	
$B_{11} = \mathtt{l} \triangleright \{\mathtt{r} \triangleleft \mathtt{inl}.\mathtt{r} \triangleleft \mathtt{inl}.\mathsf{id}_{T_1}, \mathtt{l} \triangleright \{\mathtt{r} \triangleleft \mathtt{inl}.\mathtt{r} \triangleleft \mathtt{inr}.\mathsf{id}_{T_2}, \mathtt{r} \triangleleft \mathtt{inr}.\mathsf{id}_{T_3}\}\}$	
$A_{12} = \mathtt{r} \triangleright \{\mathtt{l} \triangleleft \mathtt{inl}.\mathtt{l} \triangleleft \mathtt{inl}.\mathsf{id}_{T_1}, \mathtt{r} \triangleright \{\mathtt{l} \triangleleft \mathtt{inl}.\mathtt{l} \triangleleft \mathtt{inr}.\mathsf{id}_{T_2}, \mathtt{l} \triangleleft \mathtt{inr}.\mathsf{id}_{T_3}\}\}$	
$B_{12} = \mathtt{r} \triangleright \{\mathtt{r} \triangleright \{\mathtt{l} \triangleleft \mathtt{inl}.\mathsf{id}_{T_1}, \mathtt{l} \triangleleft \mathtt{inr}.\mathtt{l} \triangleleft \mathtt{inl}.\mathsf{id}_{T_2}\}, \mathtt{l} \triangleleft \mathtt{inr}.\mathtt{l} \triangleleft \mathtt{inr}.\mathsf{id}_{T_3}\}$	

particular boolean value that is sent. In general, any data type with finitely many values can be encoded as possibly nested choices. Axiom [A9], corresponding to the commutativity of $\oplus$ wrt $\cong$, shows that the actual label used for making a selection is irrelevant, only the continuation matters. Axiom [A11], corresponding to the associativity for $\oplus$ wrt $\cong$, generalizes the irrelevance of labels seen in [A9] to nested selections. Since $\cong$ is a congruence, the axioms in Table 3 can also be closed by transitivity and arbitrary session type contexts.

Table 4 gives all the adapters of the axioms in Table 3. Then the soundness of the axioms in Table 3 amounts to prove:

$$\vdash A_i \blacktriangleright \{\mathtt{l} : \overline{T_i}, \mathtt{r} : S_i\} \qquad \vdash B_i \blacktriangleright \{\mathtt{l} : \overline{S_i}, \mathtt{r} : T_i\} \tag{2}$$

$$A_i \rfloor\!\lceil B_i \approx \mathsf{id}_{T_i} \qquad B_i \rfloor\!\lceil A_i \approx \mathsf{id}_{S_i} \tag{3}$$

where T_i is the l.h.s. and S_i is the r.h.s. of the axiom [Ai] for $1 \le i \le 12$.

Point 2 can be easily shown by cases on the definitions of A_i and B_i taking into account that

$$\vdash \mathsf{id}_T \blacktriangleright \{\mathtt{l} : \overline{T}, \mathtt{r} : T\}$$

Table 5: Symbolic reduction relation.

[SR-UP 1] $\mathtt{l}?(x:t).P \mathbin{\rfloor\!\lceil} Q \rightsquigarrow \mathtt{l}?(x:t).(P \mathbin{\rfloor\!\lceil} Q)$	[SR-UP 2] $P \mathbin{\rfloor\!\lceil} \mathtt{r}?(x:t).Q \rightsquigarrow \mathtt{r}?(x:t).(P \mathbin{\rfloor\!\lceil} Q)$
[SR-UP 3] $\mathtt{l}!\langle x\rangle.P \mathbin{\rfloor\!\lceil} Q \rightsquigarrow \mathtt{l}!\langle x\rangle.(P \mathbin{\rfloor\!\lceil} Q)$	[SR-UP 4] $P \mathbin{\rfloor\!\lceil} \mathtt{r}!\langle x\rangle.Q \rightsquigarrow \mathtt{r}!\langle x\rangle.(P \mathbin{\rfloor\!\lceil} Q)$
[SR-UP 5] $\mathtt{l} \triangleright \{P_{\mathtt{inl}}, P_{\mathtt{inr}}\} \mathbin{\rfloor\!\lceil} Q \rightsquigarrow \mathtt{l} \triangleright \{P_{\mathtt{inl}} \mathbin{\rfloor\!\lceil} Q, P_{\mathtt{inr}} \mathbin{\rfloor\!\lceil} Q\}$	[SR-UP 7] $\mathtt{l} \triangleleft \ell.P \mathbin{\rfloor\!\lceil} Q \rightsquigarrow \mathtt{l} \triangleleft \ell.(P \mathbin{\rfloor\!\lceil} Q)$
[SR-UP 6] $P \mathbin{\rfloor\!\lceil} \mathtt{r} \triangleright \{Q_{\mathtt{inl}}, Q_{\mathtt{inr}}\} \rightsquigarrow \mathtt{r} \triangleright \{P \mathbin{\rfloor\!\lceil} Q_{\mathtt{inl}}, P \mathbin{\rfloor\!\lceil} Q_{\mathtt{inr}}\}$	[SR-UP 8] $P \mathbin{\rfloor\!\lceil} \mathtt{r} \triangleleft \ell.Q \rightsquigarrow \mathtt{r} \triangleleft \ell.(P \mathbin{\rfloor\!\lceil} Q)$
[SR-UP 9] $(\mathtt{if}\ x\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2) \mathbin{\rfloor\!\lceil} Q \rightsquigarrow \mathtt{if}\ x\ \mathtt{then}\ (P_1 \mathbin{\rfloor\!\lceil} Q)\ \mathtt{else}\ (P_2 \mathbin{\rfloor\!\lceil} Q)$	
[SR-UP 10] $P \mathbin{\rfloor\!\lceil} (\mathtt{if}\ x\ \mathtt{then}\ Q_1\ \mathtt{else}\ Q_2) \rightsquigarrow \mathtt{if}\ x\ \mathtt{then}\ (P \mathbin{\rfloor\!\lceil} Q_1)\ \mathtt{else}\ (P \mathbin{\rfloor\!\lceil} Q_2)$	
[SR-SWAP 1] $\mathsf{c}?(x:t).\bar{\mathsf{c}}?(y:s).P \rightsquigarrow \bar{\mathsf{c}}?(y:s).\mathsf{c}?(x:t).P$	[SR-SWAP 2] $\mathsf{c}!\langle x\rangle.\bar{\mathsf{c}}!\langle y\rangle.P \rightsquigarrow \bar{\mathsf{c}}!\langle y\rangle.\mathsf{c}!\langle x\rangle.P$
[SR-SWAP 3] $\mathsf{c}?(x:t).\bar{\mathsf{c}}!\langle y\rangle.P \leftrightsquigarrow \bar{\mathsf{c}}!\langle y\rangle.\mathsf{c}?(x:t).P \quad x \neq y$	
[SR-SWAP 4] $\mathsf{c}?(x:t).\bar{\mathsf{c}} \triangleleft \ell.P \leftrightsquigarrow \bar{\mathsf{c}} \triangleleft \ell.\mathsf{c}?(x:t).P$	
[SR-SWAP 5] $\mathsf{c}!\langle x\rangle.\bar{\mathsf{c}} \triangleleft \ell.P \leftrightsquigarrow \bar{\mathsf{c}} \triangleleft \ell.\mathsf{c}!\langle x\rangle.P$	
[SR-SWAP 6] $\mathsf{c}?(x:t).\bar{\mathsf{c}} \triangleright \{P, Q\} \leftrightsquigarrow \bar{\mathsf{c}} \triangleright \{\mathsf{c}?(x:t).P, \mathsf{c}?(x:t).Q\}$	
[SR-SWAP 7] $\mathsf{c}!\langle x\rangle.\bar{\mathsf{c}} \triangleright \{P, Q\} \leftrightsquigarrow \bar{\mathsf{c}} \triangleright \{\mathsf{c}!\langle x\rangle.P, \mathsf{c}!\langle x\rangle.Q\}$	
[SR-SWAP 8] $\mathsf{c} \triangleright \{\bar{\mathsf{c}} \triangleleft \ell.P, \bar{\mathsf{c}} \triangleleft \ell.Q\} \leftrightsquigarrow \bar{\mathsf{c}} \triangleleft \ell.\mathsf{c} \triangleright \{P, Q\}$	
[SR-SWAP 9] $\mathsf{c} \triangleleft \ell.\bar{\mathsf{c}} \triangleleft \ell'.P \leftrightsquigarrow \bar{\mathsf{c}} \triangleleft \ell'.\mathsf{c} \triangleleft \ell.P$	
[SR-SWAP 10] $\mathsf{c} \triangleright \{\bar{\mathsf{c}} \triangleright \{P_1, Q_1\}, \bar{\mathsf{c}} \triangleright \{P_2, Q_2\}\} \leftrightsquigarrow \bar{\mathsf{c}} \triangleright \{\mathsf{c} \triangleright \{P_1, P_2\}, \mathsf{c} \triangleright \{Q_1, Q_2\}\}$	
[SR-COND] $\mathtt{if}\ x\ \mathtt{then}\ \mathsf{c}!\langle \mathtt{true}\rangle.P\ \mathtt{else}\ \mathsf{c}!\langle \mathtt{false}\rangle.P \rightsquigarrow \mathsf{c}!\langle x\rangle.P$	
[SR-COMM 1] $\mathtt{r}!\langle y\rangle.P \mathbin{\rfloor\!\lceil} \mathtt{l}?(x:t).Q \rightsquigarrow P \mathbin{\rfloor\!\lceil} Q\{y/x\}$	[SR-COMM 2] $\mathtt{r}?(x:t).P \mathbin{\rfloor\!\lceil} \mathtt{l}!\langle y\rangle.Q \rightsquigarrow P\{y/x\} \mathbin{\rfloor\!\lceil} Q$
[SR-CHOICE 1] $\mathtt{r} \triangleleft \ell.P \mathbin{\rfloor\!\lceil} \mathtt{l} \triangleright \{Q_{\mathtt{inl}}, Q_{\mathtt{inr}}\} \rightsquigarrow P \mathbin{\rfloor\!\lceil} Q_\ell$	[SR-CHOICE 2] $\mathtt{r} \triangleright \{P_{\mathtt{inl}}, P_{\mathtt{inr}}\} \mathbin{\rfloor\!\lceil} \mathtt{l} \triangleleft \ell.Q \rightsquigarrow P_\ell \mathbin{\rfloor\!\lceil} Q$
[SR-ID] $\mathsf{id}_T \mathbin{\rfloor\!\lceil} \mathsf{id}_T \rightsquigarrow \mathsf{id}_T$	[SR-CONTEXTS] $\dfrac{P \rightsquigarrow Q}{\mathscr{E}[P] \rightsquigarrow \mathscr{E}[Q]}$

for all types T.

For Point 3 we define a *symbolic reduction relation* which preserves equivalence of closed and typed processes (Theorem 2). This is enough since we will show that all the parallel compositions of the adapters symbolically reduce to the corresponding identities (Theorem 3). The rules of this relation are given in Table 5, where $\leftrightsquigarrow$ stands for reduction in both directions and *symbolic reduction contexts* $\mathscr{E}$ are defined by:

$$\begin{array}{rcl} \mathscr{E} & ::= & [\,] \quad \mid \quad \mathsf{c}?(x:t).\mathscr{E} \quad \mid \quad \mathsf{c}!\langle \mathsf{e}\rangle.\mathscr{E} \quad \mid \quad \mathsf{c} \triangleleft \ell.\mathscr{E} \quad \mid \quad \mathsf{c} \triangleright \{\mathscr{E}, Q\} \quad \mid \quad \mathsf{c} \triangleright \{P, \mathscr{E}\} \\ & \mid & \mathtt{if}\ \mathsf{e}\ \mathtt{then}\ P\ \mathtt{else}\ \mathscr{E} \quad \mid \quad \mathtt{if}\ \mathsf{e}\ \mathtt{then}\ \mathscr{E}\ \mathtt{else}\ Q \end{array}$$

We call this a symbolic reduction relation because it also reduces processes with free variables. We notice that this reduction applied to two parallel processes:

1. moves up the communications/selections/branchings on the left channel of the left process and the communications/selections/branchings on the right channel of the right process and the conditionals,
2. executes the communications/selections/branchings between the right channel of the left process and the left channel of the right process when possible,
3. eliminates superfluous identities,

4. swaps communications/selections/branchings on different channels when this is not forbidden by bound variables.

The more interesting rule is [SR-COND], that transforms a conditional in an output.

Theorem 2. If P is a closed and typed process and $P \rightsquigarrow^* Q$, then $P \approx Q$.

Proof. The proof is by induction on the reduction of Table 5 and by cases on the last applied rule. Notice that the proof for the swap rules is immediate, since these rules can be always reversed. We consider some interesting cases, in which we assume $R_1 \parallel \mathscr{E} \parallel R_2 \longrightarrow^* R_1' \parallel [\,] \parallel R_2'$ (by extending reduction to contexts in the obvious way) and that $\{\vec{\mathsf{v}}/\vec{y}\}$ are the substitutions made on the hole in this reduction.

[SR-UP 1] If $R_1 \parallel \mathscr{E}[\mathtt{l}?(x:t).P \parallel Q] \parallel R_2$ is correct, then each reduction from $R_1 \parallel \mathscr{E}[\mathtt{l}?(x:t).P \parallel Q] \parallel R_2$ to $\mathbf{0}$ must be of the shape

$$\begin{array}{l} R_1 \parallel \mathscr{E}[\mathtt{l}?(x:t).P \parallel Q] \parallel R_2 \longrightarrow^* R_1' \parallel (\mathtt{l}?(x:t).P \parallel Q)\{\vec{\mathsf{v}}/\vec{y}\} \parallel R_2' \longrightarrow^* \\ \mathtt{r}!\langle \mathsf{e}\rangle.R \parallel \mathtt{l}?(x:t).P\{\vec{\mathsf{v}}/\vec{y}\} \parallel Q' \longrightarrow^* R \parallel P\{\vec{\mathsf{v}}/\vec{y}\}\{\mathsf{v}/x\} \parallel Q' \longrightarrow^* \mathbf{0} \end{array}$$

where $R_1' \longrightarrow^* \mathtt{r}!\langle \mathsf{e}\rangle.R$ with $\mathsf{e} \downarrow \mathsf{v}$, $\mathsf{v} \in t$, and $Q\{\vec{\mathsf{v}}/\vec{y}\} \parallel R_2' \longrightarrow^* Q'$. We get

$$\begin{array}{l} R_1 \parallel \mathscr{E}[\mathtt{l}?(x:t).(P \parallel Q)] \parallel R_2 \longrightarrow^* R_1' \parallel \mathtt{l}?(x:t).(P \parallel Q)\{\vec{\mathsf{v}}/\vec{y}\} \parallel R_2' \longrightarrow^* \\ \mathtt{r}!\langle \mathsf{e}\rangle.R \parallel \mathtt{l}?(x:t).(P \parallel Q)\{\vec{\mathsf{v}}/\vec{y}\} \parallel R_2' \longrightarrow R \parallel P\{\vec{\mathsf{v}}/\vec{y}\}\{\mathsf{v}/x\} \parallel Q\{\vec{\mathsf{v}}/\vec{y}\} \parallel R_2' \longrightarrow^* \\ R \parallel P\{\vec{\mathsf{v}}/\vec{y}\}\{\mathsf{v}/x\} \parallel Q' \longrightarrow^* \mathbf{0} \end{array}$$

Vice versa if $R_1 \parallel \mathscr{E}[\mathtt{l}?(x:t).(P \parallel Q)] \parallel R_2$ is correct, then each reduction from $R_1 \parallel \mathscr{E}[\mathtt{l}?(x:t).(P \parallel Q)] \parallel R_2$ to $\mathbf{0}$ must be of the shape shown above, and the proof concludes similarly.

[SR-UP 7] If $R_1 \parallel \mathscr{E}[\mathtt{l} \triangleleft \mathtt{inl}.P \parallel Q] \parallel R_2$ is correct, then each reduction from $R_1 \parallel \mathscr{E}[\mathtt{l} \triangleleft \mathtt{inl}.P \parallel Q] \parallel R_2$ to $\mathbf{0}$ must be of the shape

$$\begin{array}{l} R_1 \parallel \mathscr{E}[\mathtt{l} \triangleleft \mathtt{inl}.P \parallel Q] \parallel R_2 \longrightarrow^* R_1' \parallel (\mathtt{l} \triangleleft \mathtt{inl}.P \parallel Q)\{\vec{\mathsf{v}}/\vec{y}\} \parallel R_2' \longrightarrow^* \\ \mathtt{r} \triangleright \{P_{\mathtt{inl}}, P_{\mathtt{inr}}\} \parallel \mathtt{l} \triangleleft \mathtt{inl}.P\{\vec{\mathsf{v}}/\vec{y}\} \parallel Q' \longrightarrow^* P_{\mathtt{inl}} \parallel P\{\vec{\mathsf{v}}/\vec{y}\} \parallel Q' \longrightarrow^* \mathbf{0} \end{array}$$

where $R_1' \longrightarrow^* \mathtt{r} \triangleright \{P_{\mathtt{inl}}, P_{\mathtt{inr}}\}$ and $Q\{\vec{\mathsf{v}}/\vec{y}\} \parallel R_2' \longrightarrow^* Q'$. We get

$$\begin{array}{l} R_1 \parallel \mathscr{E}[\mathtt{l} \triangleleft \mathtt{inl}.(P \parallel Q)] \parallel R_2 \longrightarrow^* R_1' \parallel \mathtt{l} \triangleleft \mathtt{inl}.(P \parallel Q)\{\vec{\mathsf{v}}/\vec{y}\} \parallel R_2' \longrightarrow^* \\ \mathtt{r} \triangleright \{P_{\mathtt{inl}}, P_{\mathtt{inr}}\} \parallel \mathtt{l} \triangleleft \mathtt{inl}.(P \parallel Q)\{\vec{\mathsf{v}}/\vec{y}\} \parallel R_2' \longrightarrow P_{\mathtt{inl}} \parallel P\{\vec{\mathsf{v}}/\vec{y}\} \parallel Q\{\vec{\mathsf{v}}/\vec{y}\} \parallel R_2' \longrightarrow^* \\ P_{\mathtt{inl}} \parallel P\{\vec{\mathsf{v}}/\vec{y}\} \parallel Q' \longrightarrow^* \mathbf{0} \end{array}$$

Vice versa if $R_1 \parallel \mathscr{E}[\mathtt{l} \triangleleft \mathtt{inl}.(P \parallel Q)] \parallel R_2$ is correct, then each reduction from $R_1 \parallel \mathscr{E}[\mathtt{l} \triangleleft \mathtt{inl}.(P \parallel Q)] \parallel R_2$ to $\mathbf{0}$ must be of the shape shown above, and the proof concludes similarly.

[SR-COND] If $R_1 \parallel \mathscr{E}[\mathtt{if}\ x\ \mathtt{then}\ \mathtt{r}!\langle \mathtt{true}\rangle.P\ \mathtt{else}\ \mathtt{r}!\langle \mathtt{false}\rangle.P] \parallel R_2$ is correct, then each reduction from $R_1 \parallel \mathscr{E}[\mathtt{if}\ x\ \mathtt{then}\ \mathtt{r}!\langle \mathtt{true}\rangle.P\ \mathtt{else}\ \mathtt{r}!\langle \mathtt{false}\rangle.P] \parallel R_2$ to $\mathbf{0}$ must be of the shape

$$\begin{array}{l} R_1 \parallel \mathscr{E}[\mathtt{if}\ x\ \mathtt{then}\ \mathtt{r}!\langle \mathtt{true}\rangle.P\ \mathtt{else}\ \mathtt{r}!\langle \mathtt{false}\rangle.P] \parallel R_2 \longrightarrow^* \\ R_1' \parallel \mathtt{if}\ \mathsf{v}\ \mathtt{then}\ \mathtt{r}!\langle \mathtt{true}\rangle.P\{\vec{\mathsf{v}}/\vec{y}\}\{\mathsf{v}/x\}\ \mathtt{else}\ \mathtt{r}!\langle \mathtt{false}\rangle.P\{\vec{\mathsf{v}}/\vec{y}\}\{\mathsf{v}/x\} \parallel R_2' \longrightarrow^* \\ R_1' \parallel \mathtt{r}!\langle \mathsf{v}\rangle.P\{\vec{\mathsf{v}}/\vec{y}\}\{\mathsf{v}/x\} \parallel R_2' \longrightarrow^* R_1' \parallel \mathtt{r}!\langle \mathsf{v}\rangle.P\{\vec{\mathsf{v}}/\vec{y}\}\{\mathsf{v}/x\} \parallel \mathtt{l}?(z:\mathtt{bool}).R \longrightarrow^* \\ R_1' \parallel P\{\vec{\mathsf{v}}/\vec{y}\}\{\mathsf{v}/x\} \parallel R\{\mathsf{v}/z\} \longrightarrow^* \mathbf{0} \end{array}$$

where $\mathsf{v} \in \mathtt{bool}$ since we start from a typed process and $R_2' \longrightarrow^* \mathtt{l}?(z:\mathtt{bool}).R$. We get

$$\begin{array}{l} R_1 \parallel \mathscr{E}[\mathtt{r}!\langle x\rangle.P] \parallel R_2 \longrightarrow^* R_1' \parallel \mathtt{r}!\langle \mathsf{v}\rangle.P\{\vec{\mathsf{v}}/\vec{y}\}\{\mathsf{v}/x\} \parallel R_2' \longrightarrow^* \\ R_1' \parallel \mathtt{r}!\langle \mathsf{v}\rangle.P\{\vec{\mathsf{v}}/\vec{y}\}\{\mathsf{v}/x\} \parallel \mathtt{l}?(z:\mathtt{bool}).R \longrightarrow R_1' \parallel P\{\vec{\mathsf{v}}/\vec{y}\}\{\mathsf{v}/x\} \parallel R\{\mathsf{v}/z\} \longrightarrow^* \mathbf{0}. \end{array}$$

Vice versa, if $R_1 \parallel \mathscr{E}[\mathtt{r}!\langle x\rangle.P] \parallel R_2$ is correct, then each reduction from $R_1 \parallel \mathscr{E}[\mathtt{r}!\langle x\rangle.P] \parallel R_2$ to $\mathbf{0}$ must be of the shape shown above with $\mathsf{v} \in \mathtt{bool}$, and the proof is similar. $\square$

Theorem 3. $A_i \mathbin{\|\!\lceil} B_i \rightsquigarrow^* \mathsf{id}_{T_i}$ and $B_i \mathbin{\|\!\lceil} A_i \rightsquigarrow^* \mathsf{id}_{S_i}$ for $1 \le i \le 12$.

Proof. The proof is by cases on i. For example

$$
\begin{array}{lll}
A_1 \mathbin{\|\!\lceil} B_1 & \rightsquigarrow^* & \mathtt{l}?(x:t).\mathtt{l}?(y:s).(\mathtt{r}!\langle y\rangle.\mathtt{r}!\langle x\rangle.\mathsf{id}_T \mathbin{\|\!\lceil} B_1) \\
& \rightsquigarrow^* & \mathtt{l}?(x:t).\mathtt{l}?(y:s).(\mathsf{id}_T \mathbin{\|\!\lceil} \mathtt{r}!\langle x\rangle.\mathtt{r}!\langle y\rangle.\mathsf{id}_T) \\
& \rightsquigarrow^* & \mathtt{l}?(x:t).\mathtt{l}?(y:s).\mathtt{r}!\langle x\rangle.\mathtt{r}!\langle y\rangle.(\mathsf{id}_T \mathbin{\|\!\lceil} \mathsf{id}_T) \rightsquigarrow^* \mathsf{id}_{!t.!s.T} \\
A_2 \mathbin{\|\!\lceil} B_2 & \rightsquigarrow^* & \mathtt{r}?(x:t).\mathtt{r}?(y:s).\big(A_2 \mathbin{\|\!\lceil} \mathtt{l}!\langle y\rangle.\mathtt{l}!\langle x\rangle.\mathsf{id}_T\big) \\
& \rightsquigarrow^* & \mathtt{r}?(x:t).\mathtt{r}?(y:s).\big(\mathtt{l}!\langle x\rangle.\mathtt{l}!\langle y\rangle.\mathsf{id}_T \mathbin{\|\!\lceil} \mathsf{id}_T\big) \\
& \rightsquigarrow^* & \mathtt{r}?(x:t).\mathtt{r}?(y:s).\mathtt{l}!\langle x\rangle.\mathtt{l}!\langle y\rangle.(\mathsf{id}_T \mathbin{\|\!\lceil} \mathsf{id}_T) \rightsquigarrow^* \mathsf{id}_{?t.?s.T} \\
A_3 \mathbin{\|\!\lceil} B_3 & \rightsquigarrow^* & \mathtt{l}?(x:t).\mathtt{l}\rhd\{\mathtt{r}\lhd\mathtt{inl}.\mathtt{r}!\langle x\rangle.\mathsf{id}_T \mathbin{\|\!\lceil} B_3, \mathtt{r}\lhd\mathtt{inr}.\mathtt{r}!\langle x\rangle.\mathsf{id}_S \mathbin{\|\!\lceil} B_3\} \\
& \rightsquigarrow^* & \mathtt{l}?(x:t).\mathtt{l}\rhd\{\mathtt{r}!\langle x\rangle.\mathsf{id}_T \mathbin{\|\!\lceil} \mathtt{l}?(x:t).\mathtt{r}!\langle x\rangle.\mathtt{r}\lhd\mathtt{inl}.\mathsf{id}_T, \\
& & \qquad\qquad \mathtt{r}!\langle x\rangle.\mathsf{id}_S \mathbin{\|\!\lceil} \mathtt{l}?(x:t).\mathtt{r}!\langle x\rangle.\mathtt{r}\lhd\mathtt{inr}.\mathsf{id}_S\} \\
& \rightsquigarrow^* & \mathtt{l}?(x:t).\mathtt{r}!\langle x\rangle.\mathtt{l}\rhd\{\mathtt{r}\lhd\mathtt{inl}.\mathsf{id}_T, \mathtt{r}\lhd\mathtt{inr}.\mathsf{id}_S\} = \mathsf{id}_{!t.(T\oplus S)} \\
A_4 \mathbin{\|\!\lceil} B_4 & \rightsquigarrow^* & \mathtt{r}?(x:t).(A_4 \mathbin{\|\!\lceil} \mathtt{r}\rhd\{\mathtt{l}\lhd\mathtt{inl}.\mathtt{l}!\langle x\rangle.\mathsf{id}_T, \mathtt{l}\lhd\mathtt{inr}.\mathtt{l}!\langle x\rangle.\mathsf{id}_S\}) \\
& \rightsquigarrow^* & \mathtt{r}?(x:t).\mathtt{r}\rhd\{\mathtt{r}?(x:t).\mathtt{l}!\langle x\rangle.\mathtt{l}\lhd\mathtt{inl}.\mathsf{id}_T \mathbin{\|\!\lceil} \mathtt{l}!\langle x\rangle.\mathsf{id}_T, \\
& & \qquad\qquad \mathtt{r}?(x:t).\mathtt{l}!\langle x\rangle.\mathtt{l}\lhd\mathtt{inr}.\mathsf{id}_S \mathbin{\|\!\lceil} \mathtt{l}!\langle x\rangle.\mathsf{id}_S\} \\
& \rightsquigarrow^* & \mathtt{r}?(x:t).\mathtt{r}\rhd\{\mathtt{l}!\langle x\rangle.\mathtt{l}\lhd\mathtt{inl}.\mathsf{id}_T \mathbin{\|\!\lceil} \mathsf{id}_T, \mathtt{l}!\langle x\rangle.\mathtt{l}\lhd\mathtt{inr}.\mathsf{id}_S \mathbin{\|\!\lceil} \mathsf{id}_S\} \\
& \rightsquigarrow^* & \mathtt{r}?(x:t).\mathtt{l}!\langle x\rangle.\mathtt{r}\rhd\{\mathtt{l}\lhd\mathtt{inl}.\mathsf{id}_T \mathbin{\|\!\lceil} \mathsf{id}_T, \mathtt{l}\lhd\mathtt{inr}.\mathsf{id}_S \mathbin{\|\!\lceil} \mathsf{id}_S\} \\
& \rightsquigarrow^* & \mathtt{r}?(x:t).\mathtt{l}!\langle x\rangle.\mathtt{r}\rhd\{\mathtt{l}\lhd\mathtt{inl}.(\mathsf{id}_T \mathbin{\|\!\lceil} \mathsf{id}_T), \mathtt{l}\lhd\mathtt{inr}.(\mathsf{id}_S \mathbin{\|\!\lceil} \mathsf{id}_S)\} \\
& \rightsquigarrow^* & \mathsf{id}_{?t.(T+S)} \\
A_5 \mathbin{\|\!\lceil} B_5 & \rightsquigarrow^* & \mathtt{l}?(x:\mathtt{unit}).\mathtt{r}!\langle()\rangle.(\mathsf{id}_T \mathbin{\|\!\lceil} \mathsf{id}_T) \rightsquigarrow^* \mathsf{id}_{!\mathtt{unit}.T} \\
A_6 \mathbin{\|\!\lceil} B_6 & \rightsquigarrow^* & \mathtt{r}?(x:\mathtt{unit}).\mathtt{l}!\langle()\rangle.(\mathsf{id}_T \mathbin{\|\!\lceil} \mathsf{id}_T) \rightsquigarrow^* \mathsf{id}_{?\mathtt{unit}.T} \\
A_7 \mathbin{\|\!\lceil} B_7 & \rightsquigarrow^* & \mathtt{l}?(x:\mathtt{bool}).\mathtt{if}\ x\ \mathtt{then}\ (\mathtt{r}\lhd\mathtt{inl}.\mathsf{id}_T \mathbin{\|\!\lceil} B_7)\ \mathtt{else}\ (\mathtt{r}\lhd\mathtt{inr}.\mathsf{id}_T \mathbin{\|\!\lceil} B_7) \\
& \rightsquigarrow^* & \mathtt{l}?(x:\mathtt{bool}).\mathtt{if}\ x\ \mathtt{then}\ (\mathsf{id}_T \mathbin{\|\!\lceil} \mathtt{r}!\langle\mathtt{true}\rangle.\mathsf{id}_T)\ \mathtt{else}\ (\mathsf{id}_T \mathbin{\|\!\lceil} \mathtt{r}!\langle\mathtt{false}\rangle.\mathsf{id}_T) \\
& \rightsquigarrow^* & \mathtt{l}?(x:\mathtt{bool}).\mathtt{if}\ x\ \mathtt{then}\ \mathtt{r}!\langle\mathtt{true}\rangle.\mathsf{id}_T\ \mathtt{else}\ \mathtt{r}!\langle\mathtt{false}\rangle.\mathsf{id}_T \\
& \rightsquigarrow & \mathtt{l}?(x:\mathtt{bool}).\mathtt{r}!\langle x\rangle.\mathsf{id}_T = \mathsf{id}_{!\mathtt{bool}.T} \\
A_8 \mathbin{\|\!\lceil} B_8 & \rightsquigarrow^* & \mathtt{r}?(x:\mathtt{bool}).\big(A_8 \mathbin{\|\!\lceil} \mathtt{if}\ x\ \mathtt{then}\ \mathtt{l}\lhd\mathtt{inl}.\mathsf{id}_T\ \mathtt{else}\ \mathtt{l}\lhd\mathtt{inr}.\mathsf{id}_T\big) \\
& \rightsquigarrow^* & \mathtt{r}?(x:\mathtt{bool}).\big(\mathtt{if}\ x\ \mathtt{then}\ (A_8 \mathbin{\|\!\lceil} \mathtt{l}\lhd\mathtt{inl}.\mathsf{id}_T)\ \mathtt{else}\ (A_8 \mathbin{\|\!\lceil} \mathtt{l}\lhd\mathtt{inr}.\mathsf{id}_T)\big) \\
& \rightsquigarrow^* & \mathtt{r}?(x:\mathtt{bool}).\big(\mathtt{if}\ x\ \mathtt{then}\ \mathtt{l}!\langle\mathtt{true}\rangle.\mathsf{id}_T\ \mathtt{else}\ \mathtt{l}!\langle\mathtt{false}\rangle.\mathsf{id}_T\big) \\
& \rightsquigarrow^* & \mathtt{r}?(x:\mathtt{bool}).\mathtt{l}!\langle x\rangle.\mathsf{id}_T = \mathsf{id}_{?\mathtt{bool}.T} \\
A_9 \mathbin{\|\!\lceil} B_9 & \rightsquigarrow^* & \mathtt{l}\rhd\{\mathtt{r}\lhd\mathtt{inr}.\mathsf{id}_T \mathbin{\|\!\lceil} B_9, \mathtt{r}\lhd\mathtt{inl}.\mathsf{id}_S \mathbin{\|\!\lceil} B_9\} \\
& \rightsquigarrow^* & \mathtt{l}\rhd\{\mathsf{id}_T \mathbin{\|\!\lceil} \mathtt{r}\lhd\mathtt{inl}.\mathsf{id}_T, \mathsf{id}_S \mathbin{\|\!\lceil} \mathtt{r}\lhd\mathtt{inr}.\mathsf{id}_S\} \\
& \rightsquigarrow^* & \mathtt{l}\rhd\{\mathtt{r}\lhd\mathtt{inl}.(\mathsf{id}_T \mathbin{\|\!\lceil} \mathsf{id}_T), \mathtt{r}\lhd\mathtt{inr}.(\mathsf{id}_S \mathbin{\|\!\lceil} \mathsf{id}_S)\} \rightsquigarrow^* \mathsf{id}_{T\oplus S} \\
A_{10} \mathbin{\|\!\lceil} B_{10} & \rightsquigarrow^* & \mathtt{r}\rhd\{A_{10} \mathbin{\|\!\lceil} \mathtt{l}\lhd\mathtt{inr}.\mathsf{id}_T, A_{10} \mathbin{\|\!\lceil} \mathtt{l}\lhd\mathtt{inl}.\mathsf{id}_S\} \\
& \rightsquigarrow^* & \mathtt{r}\rhd\{\mathtt{l}\lhd\mathtt{inl}.(\mathsf{id}_T \mathbin{\|\!\lceil} \mathsf{id}_T), \mathtt{l}\lhd\mathtt{inr}.(\mathsf{id}_S \mathbin{\|\!\lceil} \mathsf{id}_S)\} \rightsquigarrow^* \mathsf{id}_{T+S}
\end{array}
$$

□

Point 3 is a straightforward consequence of Theorems 2 and 3.

4 Concluding remarks

Type isomorphisms have been mainly studied for various λ-calculi [10]. Pérez et al. [14] interpret intuitionistic linear logic propositions as session types for concurrent processes, which communicate only channels. So both their types and their processes differ from ours. In this scenario they explain how type isomorphisms resulting from linear logic equivalences are realized by coercions between interface types of session-based concurrent systems.

The notion of isomorphism for session types investigated in this paper can be used for automatically adapting behaviors, when their differences do not entail any loss of information. Adaptation in general [3] is much more permissive than in our approach, where we require adapters to be invertible. Moreover we only adapt processes as in [2, 11], while other works like [1, 8, 7] deal with adaptation of whole choreographies. Our approach shares many similarities with [5, 13] where *contracts* (as opposed to session types) describe the behavior of clients and Web services and filters/orchestrators mediate their interaction. The theory of orchestrators in [13] allows not only permutations of subsequent inputs and subsequent outputs, but also permutations between inputs and outputs if these have no causal dependencies. The induced morphism is therefore coarser than our isomorphism, but it may entail some loss of information.

There are some open problems left for future research. The obvious ones are whether and how our theory extends to recursive and higher-order session types. Also, we do not know yet whether the set of axioms in Table 3 is *complete*. The point is that in the case of arrow, product and sum types or of arrow, intersection, union types, it is known that the set of isomorphisms is not finitely axiomatizable [12, 9, 6]. Despite the fact that session types incorporate constructs that closely resemble product and sum types, it may be the case that the particular structure of the type language allows for a finite axiomatization. A natural question is to what extend our results are a consequence of the presence of just two channels in the process language, or whether they would carry over to calculi with arbitrary channel names. A more interesting research direction is to consider this notion of session type isomorphism in relation to the work on session types and linear logic [4, 15].

References

[1] Gabrielle Anderson & Julian Rathke (2012): *Dynamic Software Update for Message Passing Programs*. In: *APLAS'12*, LNCS 7705, Springer, pp. 207–222, doi:10.1007/978-3-642-35182-2_15.

[2] Mario Bravetti, Cinzia Di Giusto, Jorge A. Pérez & Gianluigi Zavattaro (2012): *Adaptable Processes*. *Logical Methods in Comper Science* 8(4), doi:10.2168/LMCS-8(4:13)2012.

[3] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch-Lafuente & Andrea Vandin (2012): *A Conceptual Framework for Adaptation*. In: *FASE'12*, *LNCS* 7212, Springer, pp. 240–254, doi:10.1007/978-3-642-28872-2.

[4] Luís Caires & Frank Pfenning (2010): *Session Types as Intuitionistic Linear Propositions*. In: *CONCUR'10*, *LNCS* 6269, Springer, pp. 222–236, doi:10.1007/978-3-642-15375-4_16.

[5] Giuseppe Castagna, Nils Gesbert & Luca Padovani (2009): *A Theory of Contracts for Web Services*. *ACM Transactions on Programming Languages and Systems* 31(5), doi:10.1145/1538917.1538920.

[6] Mario Coppo, Mariangiola Dezani-Ciancaglini, Ines Margaria & Maddalena Zacchi (2014): *Isomorphism of Intersection and Union Types*. *Mathematical Structures in Computer Science* http://www.di.unito.it/~dezani/papers/cdmz.pdf, to appear.

[7] Mario Coppo, Mariangiola Dezani-Ciancaglini & Betti Venneri (2014): *Self-Adaptive Monitors for Multiparty Sessions*. In: *PDP'14*, IEEE, pp. 688–696, doi:10.1109/PDP.2014.18.

[8] Mila dalla Preda, Ivan Lanese, Jacopo Mauro, Maurizio Gabbrielli & Saverio Giallorenzo (2013): *Safe Run-time Adaptation of Distributed Systems*. http://www.cs.unibo.it/~lanese/publications/fulltext/safeadapt.pdf.gz.

[9] Mariangiola Dezani-Ciancaglini, Roberto Di Cosmo, Elio Giovannetti & Makoto Tatsuta (2010): *On Isomorphisms of Intersection Types*. *ACM Transactions on Computational Logic* 11(4), pp. 1–22, doi:10.1145/1805950.1805955.

[10] Roberto Di Cosmo (1995): *Isomorphisms of Types: From Lambda-Calculus to Information Retrieval and Language Design*. Birkhauser Boston, doi:10.1017/S0960129505004871.

[11] Cinzia Di Giusto & Jorge A. Pérez (2013): *Disciplined Structured Communications with Consistent Runtime Adaptation*. In: *SAC'13*, ACM Press, pp. 1913–1918, doi:10.1145/2480362.2480716.

[12] Marcelo Fiore, Roberto Di Cosmo & Vincent Balat (2006): *Remarks on Isomorphisms in Typed Lambda Calculi with Empty and Sum Types*. *Annals of Pure and Applied Logic* 141(1–2), pp. 35–50, doi:10.1016/j.apal.2005.09.001.

[13] Luca Padovani (2010): *Contract-Based Discovery of Web Services Modulo Simple Orchestrators*. *Theoretical Computer Science* 411, pp. 3328–3347, doi:10.1016/j.tcs.2010.05.002.

[14] Jorge A. Pérez, Luís Caires, Frank Pfenning & Bernardo Toninho (2012): *Linear Logical Relations for Session-Based Concurrency*. In: *ESOP'12*, *LNCS* 7211, Springer, pp. 539–558, doi:10.1007/978-3-642-28869-2_27.

[15] Philip Wadler (2012): *Propositions as Sessions*. In: *ICFP'12*, ACM, pp. 273–286, doi:10.1145/2364527.2364568.