

Amending Contracts for Choreographies*

Laura Bocchi Julien Lange Emilio Tuosto

Department of Computer Science, University of Leicester, UK

bocchi@mcs.le.ac.uk

jlange@mcs.le.ac.uk

emilio@mcs.le.ac.uk

Distributed interactions can be suitably designed in terms of *choreographies*. Such abstractions can be thought of as global descriptions of the coordination of several distributed parties. *Global assertions* define contracts for choreographies by annotating multiparty session types with logical formulae to validate the content of the exchanged messages. The introduction of such constraints is a critical design issue as it may be hard to specify contracts that allow each party to be able to progress without violating the contract. In this paper, we propose three methods that automatically correct inconsistent global assertions. The methods are compared by discussing their applicability and the relationships between the amended global assertions and the original (inconsistent) ones.

1 Introduction

Choreographies are high level models that describe the conversations among distributed parties from a global perspective. *Global types* [6] and *global assertions* [3] provide an effective methodology for the design of distributed choreographies (as e.g., in [5]) by allowing static checking of a number of properties such as deadlock freedom and session fidelity.

Intuitively, global types establish the interaction pattern for the harmonious coordination of distributed parties while global assertions combine global types with logic to feature *design-by-contract* [8]. Basically, global assertions decorate global types with logical formulae (*predicates*) that constrain interactions, declaring senders' obligations and receivers' requirements on exchanged data and on the choice of the branches to follow. This adds fine-grained constraints to the specification of the interaction structure. For instance, the global assertion

$$\begin{aligned} \text{Alice} &\rightarrow \text{Bob} : \{a \mid a > 0\}. \\ \text{Bob} &\rightarrow \text{Carol} : \{b \mid b > a\} \end{aligned} \tag{1.1}$$

describes a protocol with three participants, Alice, Bob, and Carol, who agree on a “contract” constraining the *interaction variables* a and b . The contract stipulates that (i) Alice has to send a positive value to Bob in the first interaction, and that (ii) Bob is obliged to send Carol a value strictly greater than the one fixed for a in the first interaction. Notice that Bob can fulfill his pledge (i.e., the assertion $b > a$ in the second interaction above) only after he has received the value a from Alice.

Once designed, a global assertion \mathcal{G} is *projected* on *endpoint assertions* that are local types – modelling the behaviour of a specific participant – constrained according to the predicates of \mathcal{G} . For instance, the projection for Alice in the example (1.1) above is an endpoint assertion prescribing that Alice has to send a positive value to Bob. Endpoint assertions can be used for static validation of the actual processes implementing one or more roles in a choreography represented by \mathcal{G} , and/or to synthesise monitor processes for run-time checking/enforcement.

*This work has been supported by the project Leverhulme Trust Award Tracing Networks.

The methodology described above can be applied only when global assertions are *well-asserted* [3], namely when global assertions obey two precise design principles: *history-sensitivity* (HS for short) and *temporal satisfiability* (TS for short). Informally, HS demands that a party having an obligation on a predicate has enough information for choosing a set of values that guarantees it. Instead, TS requires that the values sent in each interaction do not make predicates of future interactions unsatisfiable.

The main motivation of our interest in HS and TS is that, in global assertions, they are the technical counterparts of the fundamental coordination issue that could be summarized in the slogan “who does what and when does (s)he do it”. In fact, HS pertains to *when* variables are constrained and *who* constrains them, while TS pertains to *which* values variables take. The contracts specified in global assertions are, on the one hand, “global” as they pertain to the whole choreography while, on the other hand, they are also “local” in (at least) two aspects. The first is that they assign responsibilities to participants (*who*) at definite moments of the computation (*when*). The second aspect is that the values assigned to variables are critical because either one could over-constrain variables fixed in the past or over-restrict the range of those assigned in the future (*which*). These conditions (especially TS) are rather crucial as global assertions that violate them may be infeasible or fallacious. For instance, if the predicate for Bob in the second interaction in (1.1) were $3 > b > a$ then Bob could not fulfill his contract if Alice had fixed the value 2 for a in the first interaction.

Guaranteeing HS and TS is often non-trivial, and this burden is on the software architect; using tools like the ones described in [7], one only highlights the problems but does not help to fix them. HS and TS are global semantic properties that may be hard to achieve. Namely, TS requires to trace back for “under-constrained” interactions (i.e., which allow values causing future predicates to be unsatisfiable) and re-distribute there the unsatisfiable constraints.

Contributions We show a few techniques that help software architects to amend *global assertions* during the design of distributed choreographies. The preliminary notions used in the rest of the paper are given in § 2. In § 3 we give two algorithms which, if applicable, automatically fix HS in global assertions; the first algorithm strengthens a predicate while the second one is based on variable propagation. In § 4 we give an algorithm which, if possible, moves predicates up in the global assertion in order to remove TS violations. § 5 outlines a methodology based on the three algorithms. Conclusions and future work are discussed in § 6.

2 Preliminaries

Let \mathcal{P} (ranged over by p, q, s, r, \dots) and \mathcal{V} (ranged over by u, v, x, y, \dots) be two infinitely countable sets of identifiers. We assume $\mathcal{P} \cap \mathcal{V} = \emptyset$ and call their elements *participants* and *interaction variables*, respectively. Hereafter, $\vec{}$ represents a list of some elements (for instance, \vec{v} is a list of interaction variables); the concatenation of \vec{x} and \vec{y} is denoted by the juxtaposition $\vec{x} \vec{y}$, and, abusing notation, we confound lists with the underlying sets of their elements (e.g., $a \in \vec{x}$ indicates that a occurs in the list \vec{x}). Also, expressions (ranged over by e) include variables in \mathcal{V} , basic data types (e.g., integers, booleans, etc.), and usual arithmetic operations/relations; $var(e)$ is the set of (free) variables in e ; and, we denote logic implication with the symbol \supset .

As in [3], we parametrise our constructions wrt a logical language Ψ , which we assume to be a decidable fragment of a first-order logic with expressions and quantifiers on variables; the set of free interaction variables of $\psi \in \Psi$ is denoted as $var(\psi)$ and we write $\psi(\vec{v})$ to emphasise that $var(\psi) \subseteq \vec{v}$.

The main ingredients of global assertions are *interactions*, abbreviated ι , which have the form:

$$\mathbf{s} \rightarrow \mathbf{r} : \{\vec{v} \mid \Psi\} \quad (2.1)$$

where $\mathbf{s}, \mathbf{r} \in \mathcal{P}$ are the *sender* and the *receiver*, $\vec{v} \subseteq \mathcal{V}$ is a pairwise-distinct list of variables, and $\Psi \in \Psi$. Variables \vec{v} are called *interaction variables* and, in (2.1), we say that they are *introduced* by \mathbf{s} . The interaction (2.1) reads as “ \mathbf{s} has to send to \mathbf{r} some values for \vec{v} that satisfy Ψ ” or as “ \mathbf{r} relies that the values fixed by \mathbf{s} for \vec{v} satisfy Ψ ”. For instance,¹

$$\mathbf{s} \rightarrow \mathbf{r} : \{v \ w \mid \exists u. v = u \times w\}$$

states that \mathbf{s} has the obligation to send \mathbf{r} two values such that the first is a multiple of the second.

Remark 1. In [3], interactions specify a channel over which participants communicate. In (2.1) we omit channels since they are inconsequential to our results ([2] shows that channels can indeed be removed).

Given ι as in (2.1), we define

$$\text{snd}(\iota) \stackrel{\text{def}}{=} \mathbf{s}, \quad \text{rcv}(\iota) \stackrel{\text{def}}{=} \mathbf{r}, \quad \text{var}(\iota) \stackrel{\text{def}}{=} \vec{v}, \quad \text{and} \quad \text{cst}(\iota) \stackrel{\text{def}}{=} \Psi$$

Def. 2 below is essentially borrowed from [3] but for a slightly simplified syntax.

Definition 2 (Global Assertions). Global assertions are defined by the following productions.

$$\begin{array}{ll} \mathcal{G} & ::= \iota. \mathcal{G} & \text{Prefix} \\ & | \mathbf{s} \rightarrow \mathbf{r} : \left(\{\Psi_j\} l_j : \mathcal{G}_j \right)_{j \in J} & \text{Branching} \\ & | \mu \mathbf{t} \langle \vec{e} \rangle \{ \vec{v} \mid \Psi \}. \mathcal{G} & \text{Recursive definition} \\ & | \mathbf{t} \langle \vec{e} \rangle & \text{Recursive call} \\ & | \text{end} & \text{End session} \end{array}$$

where $\Psi, \Psi_j \in \Psi$ and l_j ranges over a set of labels. We let $\mathcal{G}, \mathcal{G}', \mathcal{G}_j$ range over global assertions.

The first production in Def. 2 represents an interaction prefix; interaction variables $\text{var}(\iota)$ are bound in the continuation of the prefix and in $\text{cst}(\iota)$. The second production allows the selector \mathbf{s} to choose one of the labels $\{l_j\}_{j \in J}$ and send it to \mathbf{r} ; the choice of label l_j is guarded by Ψ_j (guaranteed by \mathbf{s}) and is followed by \mathcal{G}_j . The formal parameters $\vec{v} \subseteq \mathcal{V}$ in recursive definitions² are constrained by the invariant Ψ which must be satisfied at each recursive call (this is guaranteed when the global assertion satisfies TS). The initialisation vector \vec{e} (of the same length as \vec{v}) specifies the initial values of the formal parameters. Recursive calls must be prefix-guarded.

The termination of the session is represented by *end* (trailing occurrences are often omitted). We denote with $\text{var}(\mathcal{G})$ the set of interaction variables and recursion parameters in \mathcal{G} .

Remark 3. For simplicity, we assume Barendregt’s convention (i.e., bound variables are all distinct and they differ from any free variable). Moreover, global assertions \mathcal{G} are closed, i.e., each free occurrence of $v \in \text{var}(\mathcal{G})$ is either preceded by an interaction ι such that $v \in \text{var}(\iota)$ or by a recursive definition having v as one of its formal parameters.

A participant \mathbf{p} knows a variable $v \in \text{var}(\mathcal{G})$ if either

¹For simplicity, we assume the typing of variables understood.

²Variables \vec{v} are pairwise distinct and their free occurrences in the body of the recursion are bound by the recursive definition.

- there is ι in \mathcal{G} such that $v \in \text{var}(\iota)$ and $\mathfrak{p} \in \{\text{snd}(\iota), \text{rcv}(\iota)\}$
- or there is a recursive definition $\mu \mathfrak{t} \langle \vec{e}_1 e \vec{e}_2 \rangle \{\vec{v}_1 v \vec{v}_2 \mid \Psi\} \cdot \mathcal{G}'$ in \mathcal{G} such that \mathfrak{p} knows all the variables³ in $\text{var}(e)$ and, for each recursive invocation $\mathfrak{t} \langle \vec{e}'_1 e' \vec{e}'_2 \rangle$ in \mathcal{G}' , \mathfrak{p} knows all variables in $\text{var}(e')$.

We denote with $\text{knows}_{\mathfrak{p}}(\mathcal{G}) \subseteq \text{var}(\mathcal{G})$ the set of variables in \mathcal{G} that \mathfrak{p} knows.

Example 4. Consider the following global assertion

$$\begin{aligned} \mathcal{G}_{\text{ex4}} &= \mu \mathfrak{t} \langle 10 \rangle \{v \mid \Psi\}. \\ &\quad \text{Alice} \rightarrow \text{Bob} : \{v_1 \mid \Psi_1\}. \\ &\quad \text{Bob} \rightarrow \text{Carol} : \{v_2 \mid \Psi_2\}. \\ &\quad \mathfrak{t} \langle v_1 \rangle \end{aligned}$$

repeatedly executing a computation where (i) Alice sends a variable v_1 to Bob and (ii) Bob sends a variable v_2 to Carol. At each step, the invariant Ψ must be satisfied, namely at the first invocation $\Psi[10/v]$ must hold and in all subsequent invocations $\Psi[v_1/v]$ must hold.

In \mathcal{G}_{ex4} , Alice knows v_1 , since she sends it, while $v_1, v_2 \in \text{knows}_{\text{Bob}}(\mathcal{G}_{\text{ex4}})$, since Bob receives v_1 and sends v_2 , respectively. Carol knows v_2 , since she receives it. Also, $v \in \text{knows}_{\text{Alice}}(\mathcal{G}_{\text{ex4}}) \cap \text{knows}_{\text{Bob}}(\mathcal{G}_{\text{ex4}})$, since Alice and Bob know v_1 , the unique variable in the expression of the recursive call (and they trivially know all the variables in the initial expression, i.e. the constant 10). However, Carol does not know v since she does not know v_1 .

It is convenient to treat global assertions as trees whose nodes are drawn from a set \mathcal{N} (ranged over by n, n', \dots) and labelled with information on the syntactic categories of Def. 2. Hereafter, we write $n \in T$ if n is a node of a tree T , \underline{n} to denote the label of n , and T^\bullet for the root of T .

Definition 5 (Assertion Tree). The assertion tree $\mathbb{T}(\mathcal{G})$ of a global assertion \mathcal{G} is defined as follows:

- If $\mathcal{G} = \iota \cdot \mathcal{G}'$ then $\mathbb{T}(\mathcal{G})^\bullet$ has label ι and its unique child is $\mathbb{T}(\mathcal{G}')^\bullet$.
- If $\mathcal{G} = \mathfrak{s} \rightarrow \mathfrak{r} : \left(\{\Psi_j\} l_j : \mathcal{G}_j \right)_{j \in J}$ then $\mathbb{T}(\mathcal{G})^\bullet$ has label $\mathfrak{s} \rightarrow \mathfrak{r}$ and its children are $\{n_j\}_{j \in J} \subseteq \mathcal{N}$ such that, for each $j \in J$, $\underline{n}_j = \{\Psi_j\} l_j$ and $\mathbb{T}(\mathcal{G}_j)^\bullet$ is the unique child of n_j .
- If $\mathcal{G} = \mu \mathfrak{t} \langle \vec{e} \rangle \{\vec{v} \mid \Psi\} \cdot \mathcal{G}'$ then $\mathbb{T}(\mathcal{G})^\bullet$ has label $\mu \mathfrak{t} \langle \vec{e} \rangle \{\vec{v} \mid \Psi\}$ and its unique child is $\mathbb{T}(\mathcal{G}')^\bullet$.
- If $\mathcal{G} = \mathfrak{t} \langle \vec{e} \rangle$ then $\mathbb{T}(\mathcal{G})$ consists of one node with label $\mathfrak{t} \langle \vec{e} \rangle$.
- If $\mathcal{G} = \text{end}$ then $\mathbb{T}(\mathcal{G})$ consists of one node with label end .

We denote the set of assertion trees as \mathcal{T} and let T, T', \dots range over \mathcal{T} .

For convenience, given $T \in \mathcal{T}$, we will use the partial functions

$$\text{var}_T : \mathcal{N} \rightarrow 2^{\mathcal{V}}, \quad \text{cst}_T : \mathcal{N} \rightarrow \Psi, \quad \text{and} \quad \text{snd}_T, \text{rcv}_T : \mathcal{N} \rightarrow \mathcal{P}$$

that are undefined⁴ on $\mathcal{N} \setminus \{n \mid n \in T\}$ and defined as follows otherwise:

$$\begin{aligned} \text{var}_T(n) &= \begin{cases} \text{var}(\iota), & \text{if } \underline{n} = \iota \\ \emptyset, & \text{otherwise} \end{cases} & \text{cst}_T(n) &= \begin{cases} \Psi, & \text{if } \underline{n} = \iota \text{ and } \text{cst}(\iota) = \Psi, \text{ or } \underline{n} = \{\Psi\} l \\ \text{true}, & \text{otherwise} \end{cases} \\ \text{snd}_T(n) &= \begin{cases} \text{snd}(\iota), & \text{if } \underline{n} = \iota \\ \mathfrak{s}, & \text{if } \underline{n} = \mathfrak{s} \rightarrow \mathfrak{r} \end{cases} & \text{rcv}_T(n) &= \begin{cases} \text{rcv}(\iota), & \text{if } \underline{n} = \iota \\ \mathfrak{r}, & \text{if } \underline{n} = \mathfrak{s} \rightarrow \mathfrak{r} \end{cases} \end{aligned}$$

Moreover, we shall use the following functions:

³Assume that the length of \vec{e}_i and \vec{e}'_i is the same of \vec{v}_i for $i \in \{1, 2\}$.

⁴We write $f(x) = \perp$ when the function f is undefined on x .

- $parent_T(n)$ returning ε if $n = T^\bullet$, the parent of n in T if $n \in T$, and \perp otherwise.
- $n \uparrow_T$ returning the path from T^\bullet to n if $n \in T$, and \perp otherwise.

Given $T \in \mathcal{T}$, let $A(T)$ be the global assertion obtained by appending the labels of the nodes in (depth-first) preorder traversal visit of T .

Fact 6. $A(T(\mathcal{G})) = \mathcal{G}$

Fact 6 allows us to extend $knows_p(-)$ to \mathcal{T} by $knows_p(T) \stackrel{\text{def}}{=} knows_p(A(T))$.

Fact 7. *If $T \in \mathcal{T}$ then $T(A(T)) = T$*

Facts 6 and 7 basically induce an isomorphism between global assertions and their parsing trees.

3 Towards a Better Past

In a distributed choreography, parties have to make local choices on the communicated values; such choices impact on the graceful coordination of the distributed parties. It is therefore crucial that the responsible party has “enough information” to commit to an “appropriate” local choice, in each point of the choreography. For global assertions, this distills into *history sensitivity* (HS), a property defined in [3] demanding each sender/selector to know all the variables involved in the predicates (s)he must guarantee. We illustrate HS with Example 8 below.

Example 8. *The global assertion \mathcal{G}_{ex8} violates HS.*

$$\begin{aligned} \mathcal{G}_{ex8} &= \text{Alice} \rightarrow \text{Bob} : \{v_1 \mid v_1 > 0\}. \\ &\quad \text{Bob} \rightarrow \text{Carol} : \{v_2 \mid v_2 > 0\}. \\ &\quad \text{Carol} \rightarrow \text{Alice} : \{v_3 \mid v_3 > v_1\} \end{aligned}$$

In fact, Carol’s obligation $v_3 > v_1$ cannot be fulfilled because $v_1 \notin \text{knows}_{\text{Carol}}(\mathcal{G}_{ex8})$.

Given a global assertion \mathcal{G} , the function $\overline{\text{HS}}(\mathcal{G})$ below returns the nodes of $T(\mathcal{G})$ where HS is violated

$$\overline{\text{HS}}(\mathcal{G}) \stackrel{\text{def}}{=} \{n \in T(\mathcal{G}) \mid \text{var}(cst_T(n)) \not\subseteq \text{knows}_s(n \uparrow_T) \text{ and } s = \text{resp}_T(n)\}$$

where $\text{resp}_T(-) : \mathcal{N} \rightarrow \mathcal{P}$ yields the responsible party of a node and is defined as

$$\text{resp}_T(n) \stackrel{\text{def}}{=} \begin{cases} \text{snd}_T(n), & \text{if } \underline{n} = \mathfrak{t} \\ \text{snd}_T(\text{parent}_T(n)), & \text{if } \underline{n} = \{\Psi\}l \\ \perp, & \text{otherwise} \end{cases}$$

Intuitively, to determine whether a node $n \in T(\mathcal{G})$ violates HS, one checks if the responsible party of n knows all the variables involved in $cst_{T(\mathcal{G})}(n)$.

Given $T \in \mathcal{T}$, $\text{varHS}_T(-) : \mathcal{N} \rightarrow 2^{\mathcal{V}}$ is defined as

$$\text{varHS}_T(n) \stackrel{\text{def}}{=} \text{var}(cst_T(n)) \setminus \text{knows}_s(n \uparrow_T) \text{ where } s = \text{resp}_T(n)$$

Namely, $\text{varHS}_T(n)$ yields the variables of n not known to the responsible party of n . It is a simple observation that if HS is violated in a node n , then there exists a variable in the predicate of n which is not known to the responsible party of n (namely if $n \in \overline{\text{HS}}(\mathcal{G})$ then $\text{varHS}_T(n) \neq \emptyset$).

Example 9. Consider the following global assertion:

$$\begin{aligned}
\mathcal{G}_{\text{ex9}} &= \mu \mathbf{t} \langle 10 \rangle \{v \mid v > 0\}. \\
&\quad \text{Alice} \rightarrow \text{Bob} : \{v_1 \mid v \geq v_1\}. \\
&\quad \text{Bob} \rightarrow \text{Carol} : \{v_2 \mid v_2 > v_1\}. \\
&\quad \text{Carol} \rightarrow \text{Alice} : \{v_3 \mid v_3 > v_1\}. \\
&\quad \text{Carol} \rightarrow \text{Bob} : \{v_4 \mid v_4 > v\}. \\
&\quad \mathbf{t} \langle v_1 \rangle
\end{aligned}$$

$\overline{\text{HS}}(\mathcal{G}_{\text{ex9}}) = \{n_3, n_4\}$ where n_3 and n_4 are the nodes in $\mathbf{T}(\mathcal{G}_{\text{ex9}})$ corresponding to the third and fourth interactions of \mathcal{G}_{ex9} , i.e. $\underline{n}_3 = \text{Carol} \rightarrow \text{Alice} : \{v_3 \mid v_3 > v_1\}$ and $\underline{n}_4 = \text{Carol} \rightarrow \text{Bob} : \{v_4 \mid v_4 > v\}$.

In Example 9, Carol is responsible for both violations (i.e., $\text{resp}_{\mathbf{T}(\mathcal{G}_{\text{ex9}})}(n_3) = \text{resp}_{\mathbf{T}(\mathcal{G}_{\text{ex9}})}(n_4) = \text{Carol}$). $\text{varHS}_{\mathbf{T}(\mathcal{G}_{\text{ex9}})}(n_3) = \{v_1\}$ (i.e., Carol has an obligation on $v_3 > v_1$ without knowing v_1) and the violation in n_4 is on $\text{varHS}_{\mathbf{T}(\mathcal{G}_{\text{ex9}})}(n_4) = \{v\}$ (i.e., Carol has an obligation on $v_4 > v$ without knowing v). Note that the violation on HS does not imply that Carol will actually violate the condition $v_3 > v_1$. In fact, Carol could unknowingly choose either a violating or a non violating value for v_3 .

In § 3.1 and § 3.2, we present two algorithms that fix, when possible, violations of HS in a global assertion. We discuss and compare their applicability, as well as the relationship between the amended global assertion and the original one. We shall use Example 9 as the running example of § 3.1 and § 3.2.

3.1 Strengthening

Fix a global assertion \mathcal{G} and its assertion tree $T = \mathbf{T}(\mathcal{G})$. Assume HS is violated at $n \in T$ and $\text{cst}_T(n) = \Psi$. Violations occur when the responsible party \mathbf{s} of n is ignorant of at least one variable $v \in \text{var}(\Psi)$. The strengthening algorithm (cf. Def. 11) replaces Ψ in \mathcal{G} with an assertion $\Psi[v'/v]$ so that

- (1) v' is a variable that \mathbf{s} knows,
- (2) if $\Psi[v'/v]$ and the predicates occurring from T^\bullet to $\text{parent}_T(n)$ are satisfied then also Ψ is satisfied.

If there is no variable v' that ensures (1) and (2) then we say that *strengthening is not applicable*. Intuitively, the method above *strengthens* Ψ with $\Psi[v'/v]$. Due to (2), Ψ can be still guaranteed relying on the information provided by all the predicates occurring before n . Let $\text{PRED}_T : \mathcal{N} \rightarrow \Psi$ yield the conjunction of the predicates on the path from T^\bullet to the parent of a node:

$$\text{PRED}_T(n) \stackrel{\text{def}}{=} \begin{cases} \perp, & \text{if } \text{parent}_T(n) = \perp \\ \text{true}, & \text{if } \text{parent}_T(n) = \varepsilon \\ \text{cst}_T(\text{parent}_T(n)) \wedge \text{PRED}_T(\text{parent}_T(n)), & \text{otherwise} \end{cases}$$

The function $\text{strengthen}(\mathcal{G})$ uses PRED_T to compute a global assertion \mathcal{G}' by replacing in \mathcal{G} , if possible, the assertion violating HS with a stronger predicate.

Definition 10 (*strengthen*). If $\overline{\text{HS}}(\mathcal{G}) = \emptyset$ then $\text{strengthen}(\mathcal{G})$ returns \mathcal{G} . If $n \in \overline{\text{HS}}(\mathcal{G})$, $v \in \text{varHS}_T(n)$ and there exists $v' \in \text{knows}_{\mathbf{s}}(n \uparrow_T)$ such that

$$\text{PRED}_T(n) \wedge \Psi[v'/v] \supset \Psi \quad \text{with} \quad \Psi = \text{cst}_T(n) \tag{3.1}$$

then $\text{strengthen}(\mathcal{G})$ returns $\mathbf{A}(T')$ where T' is obtained from T by replacing Ψ with $\Psi[v'/v]$ in \underline{n} .

Finally, when the two cases above cannot be applied, $\text{strengthen}(\mathcal{G})$ returns $\mathcal{G}'_{\perp n}$, namely it indicates that \mathcal{G} violates HS at $n \in \overline{\text{HS}}(\mathcal{G})$.

The algorithm Φ_1 in Def. 11 recursively applies $\text{strengthen}(_)$ until either the global assertion satisfies HS or Φ_1 is not applicable anymore.

Definition 11 (Φ_1). *The algorithm Φ_1 is defined as follows*

$$\Phi_1(\mathcal{G}) \stackrel{\text{def}}{=} \begin{cases} \text{strengthen}(\mathcal{G}), & \text{if } \text{strengthen}(\mathcal{G}) \in \{\mathcal{G}, \mathcal{G}'_{\not\leq n}\} \\ \Phi_1(\text{strengthen}(\mathcal{G})), & \text{otherwise} \end{cases}$$

Example 12. Consider \mathcal{G}_{ex9} from Example 9 and recall that $\overline{\text{HS}}(\mathcal{G}_{\text{ex9}}) = \{n_3, n_4\}$. Strengthening is applicable to n_3 where we can substitute v_1 with v_2 in $v_3 > v_1$ to satisfy condition (3.1) in Def. 10:

$$(v > 0 \wedge v \geq v_1 \wedge v_2 > v_1) \wedge (v_3 > v_2) \supset (v_3 > v_1)$$

The invocation of $\text{strengthen}(\mathcal{G}_{\text{ex9}})$ returns

$$\begin{aligned} \mathcal{G}' &= \mu \mathbf{t} \langle 10 \rangle \{v \mid v > 0\}. \\ &\quad \text{Alice} \rightarrow \text{Bob} : \{v_1 \mid v \geq v_1\}. \\ &\quad \text{Bob} \rightarrow \text{Carol} : \{v_2 \mid v_2 > v_1\}. \\ &\quad \text{Carol} \rightarrow \text{Alice} : \{v_3 \mid v_3 > v_2\}. \\ &\quad \text{Carol} \rightarrow \text{Bob} : \{v_4 \mid v_4 > v\}. \\ &\quad \mathbf{t} \langle v_1 \rangle \end{aligned}$$

The invocation of $\text{strengthen}(\mathcal{G}')$ returns $\mathcal{G}'_{\not\leq n_4}$ since \mathcal{G}' has still one violating node n_4 for which strengthening is not applicable e.g., $(v > 0 \wedge v \geq v_1 \wedge v_2 > v_1 \wedge v_3 > v_2) \wedge (v_4 > v_2) \not\supset (v_4 > v)$.

3.2 Variable Propagation

An alternative approach to solve HS problems is based on the modification of global assertions by letting responsible parties of the violating nodes know the variables causing the violation. The idea is that such variables are propagated within a “chain of interactions”.

Definition 13 (\prec_T). *Let $n, n' \in T$, $n \prec_T n'$ iff n appears in $n' \uparrow_T$ and $\text{rcv}_T(n) = \text{snd}_T(n')$. A vector of nodes n_1, \dots, n_t is a chain in T iff $n_i \prec_T n_{i+1}$ for all $i \in \{1, \dots, t-1\}$.*

The relation \prec_T is similar to the IO-dependency defined in [6] but does not consider branching, since a branching does not carry interaction variables.

Fix a global assertion \mathcal{G} ; let $T = \mathbf{T}(\mathcal{G})$, $n \in \overline{\text{HS}}(\mathcal{G})$, $v \in \text{varHS}_T(n)$, and $\mathbf{s} = \text{resp}_T(n)$.

The *propagation* algorithm (cf. Def. 17) is *applicable* only if there exists a \prec_T -chain in $n \uparrow_T$ through which v can be propagated from a node whose sender knows v to n , in which $\mathbf{s} = \text{resp}_T(n)$ can receive it. Given a chain $\vec{n} = n_1 \cdots n_t$ in T , let the *propagation of v in \vec{n}* be the tree $T' \in \mathcal{T}$ obtained by updating the nodes in T as follows:

- $\text{var}_{T'}(n_1) = \text{var}_T(n_1) \cup \{v_1\}$ and $\text{cst}_{T'}(n_1) = \text{cst}_T(n_1) \wedge (v_1 = v)$, with $v_1 \in \mathcal{V}$ fresh.
- for $i = 2 \dots t-1$, $\text{var}_{T'}(n_i) = \text{var}_T(n_i) \cup \{v_i\}$ and $\text{cst}_{T'}(n_i) = \text{cst}_T(n_i) \wedge (v_i = v_{i-1})$, with $v_2, \dots, v_{t-1} \in \mathcal{V}$ fresh.
- $\text{cst}_{T'}(n_t) = \text{cst}_T(n_t)[v_{t-1}/v]$
- all the other nodes of T remain unchanged.

For a sequence of nodes \vec{n} , $\text{P}_T(v, \vec{n})$ denotes T' as computed above if \vec{n} is a \prec_T -chain and \perp otherwise.

Example 14. In the global assertion \mathcal{G}_{ex14} below assume Alice knows v from previous interactions (the ellipsis in \mathcal{G}_{ex14}).

$$\begin{aligned} \mathcal{G}_{ex14} = \dots \quad & \text{Alice} \rightarrow \text{Bob} : \{u_1 \mid \Psi_1\}. \\ & \text{Bob} \rightarrow \text{Carol} : \{u_2 \mid \Psi_2\}. \\ & \text{Bob} \rightarrow \text{Dave} : \{u_3 \mid \Psi_3\}. \\ & \text{Dave} \rightarrow \text{Alice} : \{u_4 \mid u_4 > v\} \end{aligned}$$

For the chain $\vec{n} = n_1 n_3 n_4$ in $\mathsf{T}(\mathcal{G}_{ex14})$ (where n_i corresponds to the i -th interaction in \mathcal{G}_{ex14}), $\mathsf{P}_{\mathsf{T}(\mathcal{G}_{ex14})}(v, \vec{n})$ returns T' such that $\mathsf{A}(T')$ is simply \mathcal{G}_{ex14} with Ψ_1 replaced by $\Psi_1 \wedge v = v_1$, Ψ_3 replaced by $\Psi_3 \wedge v_1 = v_2$, and Ψ_4 replaced by $u_4 > v_2$ and the fresh variables v_1 and v_2 is added to the interaction variables of the first and third interactions, respectively.

We define a function `propagate` which takes a global assertion \mathcal{G} and returns: (1) \mathcal{G} itself if HS is satisfied, (2) \mathcal{G}'_n if HS is violated at $n \in \mathsf{T}(\mathcal{G})$ and propagation is not applicable, (3) \mathcal{G}' otherwise, where \mathcal{G}' is obtained by propagating a violating variable v of node n ; in the latter case, observe that v has been surely introduced in a node $n' \in n \uparrow_{\mathsf{T}(\mathcal{G})}$ from which v can be propagated, since we assume \mathcal{G} closed.

Definition 15 (`propagate`). The function `propagate`(\mathcal{G}) returns

- \mathcal{G} , if $\overline{\mathsf{HS}}(\mathcal{G}) = \emptyset$
- $\mathsf{P}_T(v, \vec{n})$, if $T = \mathsf{T}(\mathcal{G})$ and there exists $n \in \overline{\mathsf{HS}}(\mathcal{G})$ with $v \in \mathsf{varHS}_T(n)$ and there exists $\vec{n} = n_0 \vec{n}_1 n$ chain in T such that $\mathsf{snd}_T(n_0)$ knows v
- \mathcal{G}'_n with $n \in \overline{\mathsf{HS}}(\mathcal{G})$ otherwise.

Example 16. Consider again the global assertion \mathcal{G}' obtained after the invocation `strengthen`(\mathcal{G}_{ex9}) in Example 12. In this case $\overline{\mathsf{HS}}(\mathcal{G}') = \{n_4\}$ with $n_4 = \text{Carol} \rightarrow \text{Bob} : \{v_4 \mid v_4 > v\}$. Propagation is applicable to n_4 and `propagate`(\mathcal{G}') returns

$$\begin{aligned} \mathcal{G}'' = \mu \mathbf{t} \langle 10 \rangle \{v \mid v > 0\}. \\ & \text{Alice} \rightarrow \text{Bob} : \{v_1 \mid v \geq v_1\}. \\ & \text{Bob} \rightarrow \text{Carol} : \{v_2 u_1 \mid v_2 > v_1 \wedge u_1 = v\}. \\ & \text{Carol} \rightarrow \text{Alice} : \{v_3 \mid v_3 > v_2\}. \\ & \text{Carol} \rightarrow \text{Bob} : \{v_4 \mid v_4 > u_1\}. \\ & \mathbf{t} \langle v_1 \rangle \end{aligned}$$

by propagating v from the second interaction where the sender Bob knows v to Carol, \mathcal{G}'' satisfies HS. The predicate of the last interaction derives from the substitution $(v_4 > v)[u_1/v]$.

The propagation algorithm is defined below and is based on a repeated application of `propagate`(-).

Definition 17 (Φ_2). Given a global assertion \mathcal{G} , the function Φ_2 is defined as follows:

$$\Phi_2(\mathcal{G}) = \begin{cases} \text{propagate}(\mathcal{G}), & \text{if } \text{propagate}(\mathcal{G}) \in \{\mathcal{G}, \mathcal{G}'_n\} \\ \Phi_2(\text{propagate}(\mathcal{G})), & \text{otherwise} \end{cases}$$

3.3 Properties of Φ_1 and Φ_2

We now discuss the properties of the global assertions amended by each algorithm and we compare them. Hereafter, we say Φ_1 (resp. Φ_2) returns \mathcal{G} if either it returns \mathcal{G} or it returns \mathcal{G}'_n for some n .

The applicability of Φ_1 depends on whether it is possible to find a variable known by the responsible party of the violating node such that condition (3.1) in Def. 10 is satisfied. The applicability of Φ_2 depends on whether there exists a chain through which the problematic variable can be propagated.⁵

⁵ Linearity of the underlying multiparty session types (i.e., a property that ensures the existence of a dependency chain between the interactions) [6] does not guarantee that Φ_2 is always applicable. The reason is that $n_1 \prec n_2$ in the sense of [6] does not imply $n_1 \prec_\tau n_2$ since \prec_τ does not take into account branching but only interactions.

Notably, there are cases in which Φ_1 is applicable and Φ_2 is not, and vice versa. Also, Φ_1 and Φ_2 return, respectively, two different global assertions from the original one; hence it may not always be clear which one should be preferred.

Remark 18. *In distributed applications it is often necessary to guarantee that exchanged information is accessible only to intended participants. It is worth observing that Φ_2 discloses information about the propagated variable to the participants involved in the propagation chain. The architect should therefore evaluate when it is appropriate to use Φ_2 .*

First we show that both Φ_1 and Φ_2 do not change the structure of the given global assertion.

Proposition 19. *Let \mathcal{G} be a global assertion. If $\Phi_1(\mathcal{G})$ or $\Phi_2(\mathcal{G})$ return \mathcal{G}' then $\mathsf{T}(\mathcal{G})$ and $\mathsf{T}(\mathcal{G}')$ are isomorphic, namely they have the same tree structure, but different labels.*

Whereas Φ_1 does not change the underlying type of the global assertion, Φ_2 does. Indeed, in the resulting global assertion, more variables are exchanged in each interaction involved in the propagation. However, the structure of the tree remains the same.

Let $\text{erase}(\mathcal{G})$ be the function that returns the underlying global type [6] corresponding to \mathcal{G} (i.e. a global assertion without predicates).

Proposition 20 (Underlying Type Structure). *Let \mathcal{G} be a global assertion,*

- if $\Phi_1(\mathcal{G})$ returns \mathcal{G}' then $\text{erase}(\mathcal{G}) = \text{erase}(\mathcal{G}')$
- if $\Phi_2(\mathcal{G})$ returns \mathcal{G}' then for all $n \in \mathsf{T}(\mathcal{G})$ and its corresponding node $n' \in \mathsf{T}(\mathcal{G}')$,

$$\text{var}_{\mathsf{T}(\mathcal{G})}(n) \subseteq \text{var}_{\mathsf{T}(\mathcal{G}')} (n')$$

Proof sketch. The proof is by induction on the structure of \mathcal{G} and it trivially follows from the fact that neither Φ_1 nor Φ_2 changes the structure of the assertion tree. In fact, Φ_1 changes only the predicates. On the other hand, Φ_2 changes the predicates and adds fresh variables to interaction nodes, therefore changing the type of the exchanged data. \square

The application of Φ_1 and Φ_2 affects the predicates of the original global assertion. In Φ_1 , strengthening allows less values for the interaction variables of the amended interaction. Conversely, the predicates computed by Φ_2 are equivalent to the original ones (i.e., they allow sender and receiver to chose/expect the same set of values). Nevertheless, such predicates are syntactically different as Φ_2 adds the equality predicates on the propagated variables.

Proposition 21 (Assertion Predicates). *Let \mathcal{G} be a global assertion,*

1. if $\Phi_1(\mathcal{G})$ returns \mathcal{G}' then for all $n \in \mathsf{T}(\mathcal{G})$ whose label is modified by Φ_1 and its corresponding node $n' \in \mathsf{T}(\mathcal{G}')$ (cf. Proposition 20), it holds that $\text{PRED}_{\mathsf{T}(\mathcal{G}')} (n') \wedge \text{cst}_{\mathsf{T}(\mathcal{G}')} (n') \supset \text{cst}_{\mathsf{T}(\mathcal{G})} (n)$
2. if $\Phi_2(\mathcal{G})$ returns \mathcal{G}' then for all $n \in \mathsf{T}(\mathcal{G})$ whose label is modified by Φ_2 and its corresponding node $n' \in \mathsf{T}(\mathcal{G}')$

(a) $\text{cst}_{\mathsf{T}(\mathcal{G}')} (n')$ is the predicate $\text{cst}_{\mathsf{T}(\mathcal{G})} (n) \wedge \Psi$

(b) $\text{PRED}_{\mathsf{T}(\mathcal{G})} (n) \supset \text{cst}_{\mathsf{T}(\mathcal{G})} (n) \wedge \Psi \iff \text{PRED}_{\mathsf{T}(\mathcal{G}')} (n') \supset \text{cst}_{\mathsf{T}(\mathcal{G}')} (n')$

For some $\Psi \in \Psi$ satisfiable.

Proof sketch. The proof of item 1 relies on the fact that Φ_1 either does not change \mathcal{G} or replaces a problematic variable by a variables for which (3.1) holds. The proof of item 2 relies on Def.13, i.e. a predicate of the form $v_1 = v$ or $v_i = v_{i-1}$ is added to each predicate of the nodes in the chain. The additional predicates are satisfiable since they constrain only fresh variables (i.e. v_i). \square

The statement 2b in Proposition 21 amounts to say that $cst_{\mathsf{T}(\mathcal{G})}(n) \wedge \psi$ is equivalent to $cst_{\mathsf{T}(\mathcal{G}')} (n')$ when such predicates are taken in their respective contexts.

Finally, we show that Φ_1 and Φ_2 do not add violations (of either HS or TS) to the amended global assertions (Proposition 22) and that if the return value is not of the type \mathcal{G}'_n then the amended global assertion satisfies HS (Theorem 23).

Proposition 22 (Properties Preservation). *Assume $\Phi_i(\mathcal{G})$ returns \mathcal{G}' with $i \in \{1, 2\}$. If $\overline{\text{HS}}(\mathcal{G}) = \emptyset$ then $\overline{\text{HS}}(\mathcal{G}') = \emptyset$ and if $\overline{\text{TS}}(\mathcal{G}) = \emptyset$ then $\overline{\text{TS}}(\mathcal{G}') = \emptyset$.*

Proof sketch. The proof of HS preservation by both algorithms follows by the fact that they both return \mathcal{G} if $\overline{\text{HS}}(\mathcal{G}) = \emptyset$. TS preservation in Φ_1 follows from the fact that predicates may only be changed by a variable substitution. For $T = \mathsf{T}(\mathcal{G})$, such that $\overline{\text{TS}}(\mathcal{G}) = \emptyset$, we have that, for any $n \in T$

$$\text{PRED}_T(n) \supset \exists \text{var}_T(n). \phi$$

by definition of TS. And, by (3.1), we have that

$$\text{PRED}_T(n) \supset \exists \text{var}_T(n). \phi[v/v']$$

i.e. TS is preserved by Φ_1 . TS preservation in Φ_2 follows from the fact that the predicates of a global assertions are only modified by adding equalities between problematic variables and fresh variables (see statement 2b in Propostion 21). \square

Theorem 23 (Correctness). *If there is \mathcal{G}' such that $\Phi_1(\mathcal{G}) = \mathcal{G}'$ or $\Phi_2(\mathcal{G}) = \mathcal{G}'$ then $\overline{\text{HS}}(\mathcal{G}') = \emptyset$.*

Proof sketch. We only consider the cases where the algorithms do return a different tree. The proof for Φ_1 follows simply from the fact that, at each iteration of the algorithm, the variable chosen to replace the problematic one is selected so that the responsible party *knows* it.

The proof for Φ_2 is by induction on the length of the \prec_T -chain at each iteration, and follows from the condition to form such a chain. Let T be an assertion tree, $\vec{n} = n_1 \dots n_t$ be the \prec_T -chain used to solve a HS problem at $n \in T$ on a variable v . By construction, the sender of n_1 knows v , and each variable v_i added at n_i is known to the sender of n_i (by definition of knows). In addition, the receiver of the n_i is the *responsible party* of n , who therefore knows the variable v_i which replaces v in n . \square

4 Back to the Future

In a distributed choreography, the local choices made by some parties may restrict later choices of other parties to the point that no suitable values is available. This would lead to an abnormal termination since the choreography cannot continue. For global assertions, this distills into *temporal satisfiability* (TS) which requires that the values sent in each interaction do not compromise the satisfiability of future interactions. The formal definition of temporal satisfiability is adapted from [3].

Definition 24 (TS [3]). *A global assertion \mathcal{G} satisfies TS (in symbols $\text{TS}(\mathcal{G})$) iff $\text{GSat}(\mathcal{G}, \text{true})$ holds where*

$$\text{GSat}(\mathcal{G}, \psi) \text{ iff } \begin{cases} \text{GSat}(\mathcal{G}', \psi \wedge \text{cst}(\mathfrak{t})), & \text{if } \mathcal{G} = \mathfrak{t}. \mathcal{G}' \text{ and } \psi \supset \exists \text{var}(\mathfrak{t}). \text{cst}(\mathfrak{t}) \\ \bigwedge_{j \in J} \text{GSat}(\mathcal{G}_j, \psi \wedge \Psi_j), & \text{if } \mathcal{G} = \mathfrak{s} \rightarrow \mathfrak{r} : \left(\{\Psi_j\} l_j : \mathcal{G}_j \right)_{j \in J} \text{ and } \psi \supset \bigvee_{j \in J} (\Psi_j) \\ \text{GSat}(\mathcal{G}', \psi \wedge \Psi'), & \text{if } \mathcal{G} = \mu \mathfrak{t} \langle \vec{e} \rangle \{ \vec{v} \mid \Psi' \}. \mathcal{G}' \text{ or } \mathcal{G} = \mathfrak{t}_{\Psi'(\vec{v})} \langle \vec{e} \rangle, \text{ and } \psi \supset \Psi'[\vec{e}/\vec{v}] \\ \mathcal{G} = \text{end}, & \text{otherwise} \end{cases}$$

For an assertion tree $T \in \mathcal{T}$, $\text{TS}(T)$ holds iff $\text{GSat}(\mathbf{A}(T), \text{true})$.

Intuitively, ψ in GSat is equivalent to the conjunction of all the predicates that precede an interaction. In the first case, all the values satisfying ψ allow to instantiate the interaction variables $\text{var}(\iota)$ so to satisfy the constraint $\text{cst}(\iota)$ of ι . For branching, GSat requires that at least one branch can be chosen and that each possible path satisfies GSat . The recursive definition requires that the initial parameters satisfy the invariant ψ' . In recursive calls, we assume an annotation giving the invariant of the corresponding recursive definition (i.e. $\psi'(\vec{v})$).

Often, TS problems appear when one tries to restrict the domain of a variable after its introduction. To illustrate this, we introduce the following running example.

Example 25. Consider $\mathcal{G}_{\text{ex25}}$ below, where p constrains x and y :

$$\begin{aligned} \mathcal{G}_{\text{ex25}} &= \text{p} \rightarrow \text{q} : \{x \mid x < 10\}. \\ &\quad \text{p} \rightarrow \text{q} : \{y \mid y > 8\}. \\ &\quad \text{q} \rightarrow \text{p} : \{z \mid x > z \wedge z > 6 \wedge y \neq z\} \end{aligned}$$

When q introduces z , both x and y are further restricted. Noticeably, in Example 25, if p chooses, e.g. $x = 6$ then q cannot choose a value for z .

Possibly, TS can be regained by rearranging some predicates. In particular, we can “lift” a predicate to a previous interaction node. For instance, in Example 25, one could lift the predicate $\exists z. x > z > 6$ (adapted from the last interaction) to the first interaction’s predicate.

Without loss of generality, we assume that only one variable is introduced at the nodes where TS is violated. Also, we first consider TS violations occurring in interactions and recursive definitions. Amending violations arising in branching and recursive calls is similar but complicates the presentation. Hence, for the sake of simplicity, such violations are considered in § 4.2.

4.1 Lifting algorithm

We formalise the lifting algorithm. First, we give a function telling us whether a *node* violates TS.

Definition 26 (TSnode). Given $T \in \mathcal{T}$, $\text{TSnode}_T(n)$ holds iff $n \in T$, and $\text{TS}(T')$ holds where T' is the assertion tree consisting of the path $n \uparrow_T$ where the children of n (if any) are replaced by nodes with label end . In addition, we assume that TSnode holds for nodes with label $\text{s} \rightarrow \text{r}$.

We can now define a function that returns a set of nodes violating TS such that all the previous nodes in the tree do not violate TS.

Definition 27 ($\overline{\text{TS}}$). The function $\overline{\text{TS}} : \mathcal{T} \rightarrow \mathcal{N}$ is defined as follows:

$$\overline{\text{TS}}(T) \stackrel{\text{def}}{=} \{n \in T \mid \text{TSnode}_T(n) \text{ is false, and } \text{TSnode}_T(n') \text{ is true for all } n' \in \text{parent}_T(n) \uparrow_T\}$$

For instance, in Example 25, we have that $\overline{\text{TS}}(T_{\text{ex25}})$ is the singleton $\{n_{\text{ex25}}\}$ where $T_{\text{ex25}} = \mathbf{T}(\mathcal{G}_{\text{ex25}})$ and n_{ex25} is the node corresponding to the last interaction of $\mathcal{G}_{\text{ex25}}$.

Once an *interaction node* $n \in \overline{\text{TS}}(T)$ is chosen, we rearrange its predicate as two sub-predicates such that the first one constrains only the variable introduced at n , and the second one involves other variables (which have been introduced previously in T).

Definition 28 (rewrite). Let $\text{rewrite} : \Psi \times \mathcal{V} \rightarrow \Psi \times \Psi$ be defined as follows:

$$\text{rewrite}(\psi, v) = (\phi(v), \psi'(\vec{w}))$$

where $\psi(\vec{w}) \iff \phi(v) \wedge \psi'(\vec{w})$.

Note that `rewrite` is a non-deterministic total function as $\phi(v)$ could simply be true. The application of `rewrite` to Example 25 yields $\text{rewrite}(\text{cst}(n_{\text{ex25}}), \text{var}(n_{\text{ex25}})) = (z > 6, x > z \wedge y \neq z)$.

Remark 29. For a tree $T \in \mathcal{T}$ and $n \in \overline{\text{TS}}(T)$ such that $\text{rewrite}(\text{cst}_T(n), v) = (\phi, \psi')$, we may have $\text{PRED}_T(n) \not\vdash \exists v. \phi$. For instance, if the predicate defined on v alone is not satisfiable, e.g., $\phi = v < 7 \wedge v > 7$. In this case the algorithm is not applicable.

We can define a relation among predicates ψ and ϕ in a context ψ' to identify the problematic part of an assertion in an interaction node.

Definition 30 (Conflict). The predicate $\psi \in \Psi$ is in conflict on $\vec{v} \subseteq \mathcal{V}$ with ϕ in ψ' iff

$$\psi' \supset \exists \vec{v}. \phi \quad \text{and} \quad \psi' \not\vdash \exists \vec{v}. (\phi \wedge \psi)$$

Using Def. 30 and $\text{PRED}_T(n)$ (cf. § 3), we define

$$\text{split}_T(n, \phi, \psi) \stackrel{\text{def}}{=} \{\psi' \mid \psi \iff \psi' \wedge \psi'' \text{ and } \psi' \text{ is in conflict on } \text{var}(n) \text{ with } \phi \wedge \psi'' \text{ in } \text{PRED}_T(n)\}$$

which returns a set of problematic predicates. Considering again Example 25, the application of `split` yields $\text{split}_{T_{\text{ex25}}}(n_{\text{ex25}}, z > 6, x > z \wedge y \neq z) = \{x > z\}$ since $y \neq z$ allows to choose a suitable value for z .

The next definition formalises the construction of a new assertion tree which possibly regains TS, given a node and an assertion to be “lifted” (i.e. a “problematic” predicate).

Definition 31 (build). The function $\text{build}_T(n, \psi)$ returns

- $\hat{T} \in \mathcal{T}$, if we can construct \hat{T} isomorphic to T except that, each node $n' \in \text{parent}_T(n) \uparrow_T$ such that $\underline{n}' = \mathbf{s} \rightarrow \mathbf{r} : \{\vec{u} \mid \theta\}$ and $\vec{u} \cap \text{var}(\psi) \neq \emptyset$, is replaced by a node \hat{n} with label

$$\mathbf{s} \rightarrow \mathbf{r} : \{\vec{u} \mid \theta \wedge \forall \vec{x}. \exists \vec{y}. \psi\} \quad \text{such that} \quad \theta \wedge \forall \vec{x}. \exists \vec{y}. \psi \text{ is satisfiable}$$

where

- $\vec{x} \subseteq \text{var}(\psi) \setminus \text{knows}_{\mathbf{s}}(T)$ are introduced in a node in $n' \uparrow_T$
- $\vec{y} \subseteq \text{var}(\psi)$ are introduced in a node in the subtree rooted at n'

and there is no $n' \in \text{parent}_T(n) \uparrow_T$ such that $\underline{n}' = \mu \mathbf{t} \langle \vec{e} \rangle \{\vec{v} \mid \psi\}$ and $\vec{v} \cap \text{var}(\psi) \neq \emptyset$.

- \perp otherwise.

Remark 32. In the definition of `build`, we assume that if either \vec{x} or \vec{y} is empty, the corresponding unnecessary quantifier is removed. Recall that global assertions are closed (cf. § 2). Therefore all the variables in $\text{var}(\psi)$ are taken into account in the construction of the new assertion tree.

In Example 25, we would invoke $\text{build}_{T_{\text{ex25}}}(n_{\text{ex25}}, z > 6 \wedge x > z)$ which returns a new assertion tree. The new tree can be transformed into a global assertion isomorphic to $\mathcal{G}_{\text{ex25}}$ with line 1 updated to: $\mathbf{p} \rightarrow \mathbf{q} : \{x \mid x < 10 \wedge \exists z. x > z > 6\}$.

The function $\text{TSres} : \mathcal{T} \times \mathcal{N} \rightarrow \mathcal{T} \cup \perp$ either solves a TS problem n or returns \perp .

Definition 33 (TSres). Given $T \in \mathcal{T}$ and $n \in \overline{\text{TS}}(T)$, we define

$$\text{TSres}_T(n) = \begin{cases} \text{build}_T(n, \phi \wedge \psi'), & \text{if } \underline{n} = \mathbf{v} \text{ and } (\phi, \psi) = \text{rewrite}(\text{cst}_T(n), \text{var}_T(n)) \text{ and there is} \\ & \psi' \in \text{split}_T(n, \phi, \psi) \text{ s.t. } \text{build}_T(n, \phi \wedge \psi') \neq \perp \\ \text{build}_T(n, \psi[\vec{e}/\vec{v}]), & \text{if } \underline{n} = \mu \mathbf{t} \langle \vec{e} \rangle \{\vec{v} \mid \psi\} \\ \perp, & \text{otherwise} \end{cases}$$

The second case of Definition 33 handles TS violations in recursive definitions. The problem is similar to the interaction case, but in this case, the values assigned to the recursion parameters are known (i.e., \vec{v}). It may be possible to lift the recursion invariant, where we replace the recursion parameters by the corresponding initialisation vector. Example 34 illustrates this case.

Example 34. For the global assertion \mathcal{G}_{ex34} given below, $\text{TS}(\mathcal{G}_{ex34})$ does not hold because $\text{true} \not\vdash (x > y > 6)$.

$$\mathcal{G}_{ex34} = \begin{array}{l} p \rightarrow q : \{x \mid \text{true}\}. \\ \mu \mathbf{t} \langle 8 \rangle \{y \mid x > y > 6\}. \mathcal{G}' \end{array}$$

However, using the initialisation parameters, we can lift $x > y > 6$, i.e., the original predicate where we replaced y by 8, to the interaction preceding the recursion. TS now holds in the new global assertion (assuming that $\text{TS}(\mathcal{G}')$ holds as well).

Remark 35. In Example 34, if we had only lifted $x > y > 6$, as in the interaction case, it would not have solved the TS problem. Indeed, the predicate of the first interaction would have become $\exists y. x > y > 6$ which does not exclude values for x which are incompatible with the invariant (e.g., $x = 8$).

The overall lifting procedure is given. It relies on a repeated application of TSres until either the assertion tree validates TS or the function fails to solve the problem. In the latter case, the function returns the most improved version of the tree and the node at which it failed.

Definition 36 (Φ_3). Φ_3 is defined as follows, given a global assertion \mathcal{G} .

$$\Phi_3(\mathcal{G}) = \begin{cases} \mathcal{G}, & \text{if } \text{TS}(\mathcal{G}) \\ \Phi_3(\text{TSres}_{\mathbf{T}(\mathcal{G})}(n)), & \text{if there is } n \in \overline{\text{TS}}(\mathbf{T}(\mathcal{G})) \text{ s.t. } \text{TSres}_{\mathbf{T}(\mathcal{G})}(n) \neq \perp \\ \mathcal{G} \not\downarrow n, & \text{otherwise} \end{cases}$$

4.2 Applying Φ_3 to branching and recursion

Branching. According to Def. 24, TS fails on branching nodes only when *all* the branches are not satisfiable. The underlying idea being that the architect may want to design their choreography in such a way that a branch cannot be taken when some variables have a particular value.

Therefore, the architect should be involved in the resolution of the problem. Two options are possible; either the disjunction of all the predicates found in the branches is lifted, or one of the branches predicate is lifted. Arguably, the latter may also prohibit the other branches to be chosen, as shown in Example 37.

Example 37. As an illustration, we consider the following assertion:

$$\mathcal{G}_{ex37} = \begin{array}{l} p \rightarrow q : \{x \mid \text{true}\}. \\ p \rightarrow q : \begin{array}{l} \{v > 5\} l_1 : \mathcal{G}_1 \\ \{v < 5\} l_2 : \mathcal{G}_2 \end{array} \end{array}$$

Assuming that $\text{TS}(\mathcal{G}_1)$ and $\text{TS}(\mathcal{G}_2)$ hold, we have that $\text{TS}(\mathcal{G}_{ex37})$ does not hold because $\text{true} \not\vdash (v > 5 \vee v < 5)$. It is obvious that if $v = 5$ no branch may be selected.

Let's call \hat{n} the node corresponding to the branching in the second line of \mathcal{G}_{ex37} . Depending on the intention of the architect the problem could be fixed by one of these invocations to build (where, in both cases, superfluous quantifiers are removed).

- $\text{build}_{\mathbf{T}(\mathcal{G}_{ex37})}(\hat{n}, v > 5 \vee v < 5)$ replaces the predicate in the first line by $\text{true} \wedge (v > 5 \vee v < 5)$
- $\text{build}_{\mathbf{T}(\mathcal{G}_{ex37})}(\hat{n}, v < 5)$ replaces the predicate in the first line by $\text{true} \wedge (v < 5)$.

Both solutions solve the TS problem, however the second one prevents the first branch to be ever taken.

Given an assertion tree T and a branching node⁶ $n \in T$ such that TS does not hold. One can invoke

⁶We also assume that TS is not violated in $\text{parent}_T(n) \uparrow_T$ as in Def. 27.

$\text{build}_T(n, \psi)$ where ψ is either the disjunction of all the branching predicates or one of the branches predicate. If the function does not return \perp , then the TS problem is solved. Notice that we do not have to use neither `rewrite` or `split` to solve problems in branching.

Recursion. We have seen that when a TS violation is detected in a recursion definition, lifting may be applied. However, lifting a predicate involving a recursion parameter v would require to strengthen the invariant where v is introduced. This is quite dangerous, therefore the lifting algorithm does not apply in this case. In fact, for recursive definition and calls, Def. 24 requires $\psi \supset \psi'[\vec{e}/\vec{v}]$, where ψ' is the recursion invariant and ψ is the conjunction of the previous predicates. Hence, lifting a predicate involving a recursion parameter may strengthen the invariant, and possibly create a new problem in a corresponding recursive call. Moreover, notice that, in recursive calls, *GSat* (Def. 24) requires that $\psi \wedge \psi' \supset \psi'[\vec{e}/\vec{v}]$; namely, strenghtening ψ' would automatically strenghten $\psi'[\vec{e}/\vec{v}]$ and therefore leave the TS problem unsolved.

On the other hand, TS problems can be solved when they occur in recursive calls. In fact, let a TS problem appear at a node $n \in T$ such that $\underline{n} = \mathbf{t}(\vec{e})$ and let the invariant of the definition of \mathbf{t} being $\psi(\vec{v})$, then if the invocation of $\text{build}_T(n, \psi[\vec{e}/\vec{v}])$ succeeds, the problem is solved.

In order to give a more complex example of the application of Φ_3 , with TS problems in recursive calls, we consider the following example.

Example 38. Consider the global assertion below

$$\begin{aligned} \mathcal{G}_{\text{ex38}} &= \text{Generator} \rightarrow \text{Server} : \{n \mid n > 0\}. \\ &\quad \text{Player} \rightarrow \text{Server} : \{x \mid \text{true}\}. \\ &\quad \mu \mathbf{t} \langle x \rangle \{r \mid r > 0\}. \\ &\quad \quad \text{Server} \rightarrow \text{Player} : \begin{array}{ll} \{r > n\} \text{ less} : & \text{Player} \rightarrow \text{Server} : \{y \mid \text{true}\}.\mathbf{t}\langle y \rangle \\ \{r < n\} \text{ greater} : & \text{Player} \rightarrow \text{Server} : \{z \mid \text{true}\}.\mathbf{t}\langle z \rangle \\ \{r = n\} \text{ win} : & \text{end} \end{array} \end{aligned}$$

modelling a small game where a `Player` has to guess an integer n , following the hints given by a `Server`. The number is fixed by a `Generator`. Each time `Player` sends `Server` a number, `Server` says whether n is less or greater than that number.

Let T_{ex38} be the tree generated from $\mathbf{T}(\mathcal{G}_{\text{ex38}})$. There is a TS problem at the node corresponding to the recursive definition, indeed if $x \leq 0$, the invariant is not respected. After the first loop of $\Phi_3(T_{\text{ex38}})$, the predicate $x > 0$ is added in the second interaction. Then, the algorithm loops two more times to solve the problems appearing before the recursive calls. It adds $y > 0$ and $z > 0$ in the interaction of the *less* and *greater* branch, respectively. The global assertion now validates temporal satisfiability.

4.3 Properties of Φ_3

Similarly to the algorithms of § 3, Φ_3 does not modify the structure of the tree and preserves the properties of the initial assertion.

Proposition 39 (Underlying Type Structure - Φ_3). *Let \mathcal{G} be a global assertion. If $\Phi_3(\mathcal{G})$ returns \mathcal{G}' then $\text{erase}(\mathcal{G}) = \text{erase}(\mathcal{G}')$.*⁷

Proof sketch. The proof is by induction on the structure of \mathcal{G} , similarly to the one of Propostion 20. \square

Also, Φ_3 does not introduce new HS or TS problems.

⁷See Section 3.3 for the definition of *erase*.

Proposition 40 (Properties Preservation - Φ_3). *Assume $\Phi_3(\mathcal{G}) = \mathcal{G}'$. If $\overline{\text{HS}}(\mathcal{G}) = \emptyset$ then $\overline{\text{HS}}(\mathcal{G}') = \emptyset$, and if $\overline{\text{TS}}(\mathcal{G}) = \emptyset$ then $\overline{\text{TS}}(\mathcal{G}') = \emptyset$.*

Proof sketch. The preservation of HS follows from the fact that all the variables which are not known to a participant are quantified (either universally or existentially) in the modified predicates. The proof of TS preservation follows trivially from the first case of Def.36. \square

In addition, we have that Φ_3 preserves the domain of possible values for each variable from the initial assertion.

Proposition 41 (Assertion predicates). *If $\Phi_3(\mathcal{G}) = \mathcal{G}'$ then for all $n \in \text{T}(\mathcal{G})$ such that n is a leaf, and its corresponding node $n' \in \text{T}(\mathcal{G}')$ (cf. Proposition 39)*

$$\text{PRED}_T(n) \iff \text{PRED}_{T'}(n')$$

Proof sketch. The proof follows from the observation that predicates are only duplicated in the tree, i.e. the lifting algorithm does not add any new constraints in the conjunction of the predicates found on the path from the root to a leaf. \square

Finally, Proposition 42 establishes an intermediate result for the correctness of Φ_3 . It says that a successful invocation of TSres on a node removes the problem at that node.

Proposition 42 (Correctness - TSres). *Let T be an assertion tree, and $N = \overline{\text{TS}}(T)$. For each $n \in N$ such that $\text{TSres}_T(n) \neq \perp$, then $n \notin \overline{\text{TS}}(\text{TSres}_T(n))$.*

Proof sketch. We sketch the key part of the proof, i.e. the proof of the correctness of build for interaction nodes.

Let T be an assertion tree with a node n such that $n \in \overline{\text{TS}}(T)$, and $\underline{n} = \mathbf{s} \rightarrow \mathbf{r} : \{v \mid \phi \wedge \beta \wedge \gamma\}$ such that β is in conflict on $\text{var}(n)$ with $\phi \wedge \gamma$ in $\text{PRED}_T(n)$. Then $\phi \wedge \gamma$ is the predicate to be lifted. Assume $\hat{T} = \text{build}_T(n, \phi \wedge \beta)$.

By Def.31, we have that, for suitable $\vec{x}_1, \vec{y}_1 \dots \vec{x}_k, \vec{y}_k$,

$$\text{PRED}_{\hat{T}}(n) = \text{PRED}_T(n) \wedge \forall \vec{x}_1. \exists \vec{y}_1. (\phi \wedge \beta) \sigma_1 \wedge \dots \wedge \forall \vec{x}_k. \exists \vec{y}_k. (\phi \wedge \beta) \sigma_k \quad (4.1)$$

$$\iff \forall \vec{x}_1 \dots \vec{x}_k. \text{PRED}_T(n) \wedge \exists \vec{y}_1. (\phi \wedge \beta) \sigma_1 \wedge \dots \wedge \exists \vec{y}_k. (\phi \wedge \beta) \sigma_k \quad (4.2)$$

Where we assume k substitutions σ_i such that the variables bound by $\forall \vec{x}_i. \exists \vec{y}_i$ in $\phi \wedge \beta$ are pairwise distinct. We have that a quantified version of $\phi \wedge \beta$ is added k times in the assertion tree, above n .

Note that there must be a i such that $\exists \vec{y}_i. (\phi \wedge \beta) \sigma_i \iff \exists v. (\phi \wedge \beta)$. Indeed, the variables which are quantified *existentially* are the ones that (i) appear in $\phi \wedge \beta$, and (ii) are fixed *below* in tree. Therefore, the predicate which is added in the last node before n must quantify existentially v , only. If there were another variable to be quantified existentially then it would not be the last node to be updated.

By Def.31, we also know that every $\exists \vec{y}_i. (\phi \wedge \beta) \sigma_i$ is satisfiable.

By the definition of conflict (Def.30), we have that $\text{PRED}_T(n) \supset \exists v. (\phi \wedge \gamma)$ and $\text{PRED}_T(n) \not\supset \exists v. (\phi \wedge \beta)$ (hence, $\text{PRED}_T(n)$ is satisfiable). Therefore, by weakening, we have that

$$\text{PRED}_{\hat{T}}(n) \supset \exists v. (\phi \wedge \gamma) \quad (4.3)$$

By (4.1), we have that

$$\text{PRED}_{\hat{T}}(n) \supset \exists v. (\phi \wedge \beta) \quad (4.4)$$

since $\exists v.(\phi \wedge \beta)$ (modulo renaming) is one of the conjuncts of $\text{PRED}_{\hat{T}}(n)$.

TS must hold for n , which implies that $n \notin \overline{\text{TS}}(\hat{T})$ and $\text{TSnode}_{\hat{T}}(n)$ holds, i.e.

$$\text{PRED}_{\hat{T}}(n) \supset \exists v.(\phi \wedge \beta \wedge \gamma)$$

Otherwise, that would imply that

$$\text{PRED}_{\hat{T}}(n) \wedge \forall v.(\neg\phi \vee \neg\beta \vee \neg\gamma)$$

which is in contradiction with (4.3) (ϕ and γ) and (4.4) (β). \square

Finally, we can say that, if a repeated application of lifting succeeds, the global assertion which is returned satisfies temporal satisfiability.

Theorem 43 (Correctness - Φ_3). *If $\Phi_3(\mathcal{G}) = \mathcal{G}'$ then $\overline{\text{TS}}(\mathcal{G}') = \emptyset$.*

Proof sketch. The proof is by induction on the number of problematic nodes and the minimum depth of these nodes in the tree. It relies on Proposition 42, i.e. the fact that $\text{TSres}_T(n)$ either solves the problem at n or fails.

Let $T = \mathbb{T}(\mathcal{G})$ and N be the set of nodes in T which violates TS. We write $|n|$ for the depth of n in T (with $|T^\bullet| = 0$).

1. If $N = \emptyset$, then T is TS.
2. If $N \neq \emptyset$, let $n \in \overline{\text{TS}}(T) \subseteq N$, after an invocation to $\text{TSres}_T(n)$, we have
 - (a) If $|n| > 1$ then either
 - i. $N := N \setminus \{n\}$, i.e. the node is simply removed from the set of problematic nodes,
 - ii. $N := N \cup N' \setminus \{n\}$ with $\forall n'_i \in N'. |n'_i| < |n|$, i.e. the problem at n is solved but other problematic nodes, *above* n in T , are added, or,
 - iii. the algorithm fails on n
 - (b) If $|n| \leq 1$ then either $N := N \setminus \{n\}$, or the algorithm fails. In fact, once the algorithm reaches a problem located at a child of the root, then it either fails or solves the problem. Indeed, there cannot be a TS problem at the root node unless the predicate is unsatisfiable (see Def.24), in which case, the algorithm fails.

Note that selecting $n \in \overline{\text{TS}}(T)$ implies that the depth of n is smaller or equal to the depth of the nodes in N .

It can be shown by induction that the algorithm terminates either with $\overline{\text{TS}}(T) = \emptyset$, or a failure.

Regarding step 2(a)ii, note that the algorithm cannot loop on a problematic node indefinitely. Indeed, the number of (sub)predicates available for lifting is finite and, by Def.30, the algorithm moves only the predicates from which the problem originates, e.g. an equivalent constraint cannot be lifted twice. \square

5 A methodology for amending choreographies

The algorithms Φ_1 , Φ_2 , and Φ_3 in § 3 and § 4 can be used to support a methodology for amending contracts in choreographies. The methodology mainly consists of the following steps: (i) the architect design a choreography $\hat{\mathcal{G}}$, (ii) the architect is notified if there are any HS or TS problems in $\hat{\mathcal{G}}$, (iii) using Φ_1 and Φ_2 solutions may be offered for HS problems, while Φ_3 can be used to offer solutions and/or hints on how to solve TS problems; (iv) the architect picks one of the solutions offered in (iii). Steps

(ii) to (iv) are repeated until all the problems have been solved. We sketch our methodology using the following global assertion:

$$\begin{aligned} \widehat{\mathcal{G}} &= \mu \mathbf{t} \langle 10 \rangle \{v \mid v > 0\}. \\ &\quad \text{Alice} \rightarrow \text{Bob} : \{v_1 \mid v \geq v_1\}. \\ &\quad \text{Bob} \rightarrow \text{Carol} : \{v_2 \mid v_2 > v_1\}. \\ &\quad \text{Carol} \rightarrow \text{Alice} : \{v_3 \mid v_3 > v_1\}. \\ &\quad \text{Carol} \rightarrow \text{Bob} : \{v_4 \mid v_4 > v\}. \\ &\quad \text{Alice} \rightarrow \text{Bob} : \{\text{true}\} \text{ cont} : \mathbf{t} \langle v_1 \rangle, \\ &\quad \quad \quad \{\text{true}\} \text{ finish} : \text{Alice} \rightarrow \text{Bob} : \{v_5 \mid v_1 < v_5 < v_3 - 2\} \end{aligned}$$

which extends the global assertion in Example 9.

First, $\widehat{\mathcal{G}}$ is inspected by history sensitivity and temporal satisfiability checkers, such as the ones implemented in [7]. If there are any HS problems, the Φ_1 and Φ_2 algorithms are used, while Φ_3 is used for TS problems. This allows the architect to detect all the problems and consider the ones for which (at least) one of the algorithms is applicable.

We assume here that the architect focuses on HS problems first. In $\widehat{\mathcal{G}}$ there are two HS problems, both of them can be solved automatically, and the methodology will return that

1. At line 4, v_1 is not known by Carol; the problem is solvable by either
 - replacing $v_3 > v_1$ by $v_3 > v_2$ (algorithm Φ_1) at line 4, or
 - by revealing v_1 to Carol (algorithm Φ_2); in this case, line 3 becomes

$$\text{Bob} \rightarrow \text{Carol} : \{v_2 \ u_1 \mid v_2 > v_1 \wedge u_1 = v_1\}$$

and the assertion at line 4 becomes $v_3 > u_1$.

2. At line 5, v is not known by Carol; the problem is solvable by revealing the value of v to Carol (algorithm Φ_2) in which case line 3 becomes

$$\text{Bob} \rightarrow \text{Carol} : \{v_2 \ u_2 \mid v_2 > v_1 \wedge u_2 = v\}$$

and the assertion at line 5 becomes $v_4 > u_2$.

In the *propagation case* (i.e., Φ_2), the methodology gives the architect information on which participants the value of a variable may be disclosed to. Indeed, as discussed in Remark 18, it may not be appropriate to use the suggested solution. Therefore, the actual adoption of the proposed solutions should be left to the architect. In addition, the order in which problems are tackled is also left to the architect (e.g., the same variable may be involved in several problems and solving one of them may automatically fix the others). Assuming that Φ_1 is used to solve the first problem and Φ_2 to solve the second, the first five lines of the new global assertion are those in Example 16 and HS is fixed.

Now HS is satisfied in $\widehat{\mathcal{G}}$, but TS problems are still there. In case a TS problem cannot be solved automatically, additional information can be returned: (a) at which node the problem occurred, (b) which variables or recursion parameters are posing problems (i.e. using `split` and `build`), and (c) where liftings are not possible (i.e. when `build` fails to add a satisfiable predicate to a node). For $\widehat{\mathcal{G}}$ there are two TS problems which are dealt with sequentially. The methodology would report that

1. At line 6, v_1 does not satisfy the invariant $v > 0$. This can be solved by lifting $v_1 > 0$ (i.e. the invariant where v is replaced by the actual parameter v_1) to the interaction at line 2, which would yield the new assertion $v \geq v_1 \wedge v_1 > 0$.

2. At line 7, there might be no value for v_5 such that $v_1 < v_5 < v_3 - 2$. The assertion is *in conflict* (cf. Def. 30) with the previous predicates; this problem cannot be solved since lifting would add the following predicates in line 2 and 4, respectively.
 - $\exists v_3, v_5. v_1 < v_5 < v_3 - 2$ which is indeed satisfiable, but remarkably does not constraint v_1 more than the initial predicate.
 - $\forall v_1. \exists v_5. v_1 < v_5 < v_3 - 2$ which is not satisfiable, therefore the algorithm fails.

The failure of Φ_3 is due to the fact that v_5 is constrained by v_1 and v_3 which are fixed by two different participants. They would have to somehow interact in order to guarantee that there exists a value for v_5 , this cannot be done automatically. Notice that in this case the methodology tells the architect that v_5 , fixed by Alice, is constrained by v_1 and v_3 which are fixed by Alice and Carol, respectively. Our methodology can also suggest that the node introducing v_3 , or (the part of) the assertion over v_3 may be the source of the problem since v_3 is the only variable not known by Alice.

Remark 44. *The application of an algorithm could compromise the application of another one due to some “interference” effect that may arise. For instance, applying strengthening (Φ_1) could spoil the application of lifting (Φ_3) and vice versa (cf. § 6 for an intuitive explanation).*

6 Conclusions

In this paper, we investigated the problem of designing consistent assertions. We focused on two consistency criteria from [3]: history sensitivity and temporal satisfiability. We proposed and compared three algorithms (Φ_1 , Φ_2 , and Φ_3) to amend global assertions. Since each algorithm is applicable only in certain circumstances, we proposed a methodology that supports the architect when violations are not automatically amendable.

On the theoretical side, the algorithms Φ_1 , Φ_2 , and Φ_3 address the general problem of guaranteeing the satisfiability of predicates when: (1) the parts of the system have a different perspective/knowledge of the available information (in the case of history sensitivity), and (2) the constraints are introduced progressively (in the case of temporal satisfiability). The proposed solutions can be adapted and used, for instance, to amend processes (rather than types), orchestrations (rather than choreographies, when we want to check for local constraints), e.g., expressed in formalisms as CC-Pi [4], a language for distributed processes with constraints. Interestingly, temporal satisfiability is similar to the feasibility property in [1] requiring that any initial segment of a computation must be possibly extended to a full computation to prevent “a scheduler from ‘painting itself into a corner’ with no possible continuation”. A promising future development is to investigate more general accounts of satisfiability which is applicable to different application scenarios.

In scope of future work, we will study the “interference” issues of the three algorithms (see Remark 44) so to refine our methodology and use them more effectively. We conjecture, for instance, that conflicts between Φ_1 and Φ_3 appear *only* when the variable introduced where an HS problem is solved by Φ_1 is also involved in a TS problem. More precisely, let v be introduced at a node n having an HS problem. If Φ_1 is used to solved such problem the constraint at n will be strengthened. Now, if a node n' –further down than n in the tree– has a TS problem with a conflict involving v , the predicate at n will be updated (i.e. strengthened) by Φ_3 . Therefore, the predicate at n would be strengthened by each algorithm in an independent way. This may render the predicate at n unsatisfiable.

We will also study the applicability of our methodology in more realistic cases in order to assess the quality of the solutions offered by our algorithms.

We plan to implement our algorithms and support for the methodology by integrating it in the tool introduced in [7].

References

- [1] Krzysztof R. Apt, Nissim Francez & Shmuel Katz (1988): *Appraising fairness in languages for distributed programming*. *Distributed Computing* 2, pp. 226–241.
- [2] Lorenzo Bettini, Mario Coppo, Loris D’Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini & Nobuko Yoshida (2008): *Global Progress in Dynamically Interleaved Multiparty Sessions*. In: Franck van Breugel & Marsha Chechik, editors: *CONCUR, Lecture Notes in Computer Science* 5201, Springer, pp. 418–433. Available at http://dx.doi.org/10.1007/978-3-540-85361-9_33.
- [3] Laura Bocchi, Kohei Honda, Emilio Tuosto & Nobuko Yoshida (2010): *A Theory of Design-by-Contract for Distributed Multiparty Interactions*. In: Paul Gastin & François Laroussinie, editors: *CONCUR, Lecture Notes in Computer Science* 6269, Springer, pp. 162–176. Available at http://dx.doi.org/10.1007/978-3-642-15375-4_12.
- [4] Maria Grazia Buscemi & Ugo Montanari (2007): *CC-Pi: a constraint-based language for specifying service level agreements*. In: *Proceedings of the 16th European conference on Programming, ESOP’07*, Springer-Verlag, Berlin, Heidelberg, pp. 18–32. Available at <http://portal.acm.org/citation.cfm?id=1762174.1762179>.
- [5] Marco Carbone, Kohei Honda & Nobuko Yoshida (2007): *Structured Communication-Centred Programming for Web Services*. In: *19th International Conference on Concurrency Theory (Concur’08)*, Springer, pp. 2–17. Available at <http://www.eecs.qmul.ac.uk/~carbonem/cdlpaper/esop2007.pdf>.
- [6] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty asynchronous session types*. In: *POPL*, pp. 273–284. Available at <http://doi.acm.org/10.1145/1328438.1328472>.
- [7] Julien Lange & Emilio Tuosto (2010): *A Modular Toolkit for Theories of Distributed Interactions*. In: *PLACES*. To appear.
- [8] Bertrand Meyer (1997): *Object-Oriented Software Construction (Chapter 31)*. Prentice Hall.