Scientific Annals of Computer Science vol. 22 (1), 2012, pp. 61–104
DOI: 10.7561/SACS.2012.1.61

Three Algorithms and a Methodology for Amending Contracts for Choreographies[1]

Laura BOCCHI[2], Julien LANGE[2], Emilio TUOSTO[2]

Abstract

Distributed interactions are crucial design aspects to consider in modern applications. They can be suitably designed in terms of *choreographies*, that are 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. We propose three algorithms to 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. Also, we specify a methodology that exploits our algorithms to help designers to amend their choreographies. To show how the methodology can be applied we consider a simple scenario.

Keywords: multiparty session types, design-by-contract, assertions, choreography, satisfiability

[1]This work has been supported by the project Leverhulme Trust Award Tracing Networks.

[2] Department of Computer Science, University of Leicester, University Road, Leicester LE1 7RH, UK. Email: {lb148, jl250, et52}@le.ac.uk

1 Introduction

The coordination of distributed activities is not an easy task and it has to be tackled at different levels of abstraction. In particular, the distributed coordination of distributed applications has recently received considerable attention. The main approaches in this context are *orchestrations* and *choreographies*; in this paper we focus on the latter.

Choreographies are high-level models that describe the conversations among distributed parties from a global perspective. The techniques used to represent choreographies span from informal approaches (like UML's sequence diagrams) to formal ones (like e.g., Message State Charts) [1, 10, 12]. Among the latter family of models, *global types* [8] and *global assertions* [3] provide an effective approach for the design of choreographies (as e.g., in [6]) by allowing static checking of a number of properties such as progress 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* [11]. For instance, consider a choreography where two distributed participants, say `Alice` and `Bob` have a "request-reply" interaction; a global type for such choreography could be

$$\begin{array}{l}\texttt{Alice} \to \texttt{Bob} : \ \mathbf{a}\langle \mathit{String}\rangle.\\ \texttt{Bob} \to \texttt{Alice} : \ \mathbf{b}\langle \mathit{Bool}\rangle\end{array}$$

where the first interaction is a request from `Alice` to `Bob` with a message of type *String* on a communication channel **a**, and the latter interaction is the reply of `Bob` to `Alice` on **b** with a message of type *Bool*.

Global assertions decorate global types with logical formulae (*predicates*) that constrain interactions, declaring senders' obligations and receivers' requirements on the values of the exchanged data and on the choice of the branches to follow. This adds fine-grained constraints to the specification of the interaction structure. Consider for instance the global assertion below, where the values of the messages are represented by the *interaction variables* x and y (the communication channels and the types of the exchanged data are immaterial, hence omitted)

$$\begin{array}{l}\texttt{Alice} \to \texttt{Bob} : \{x \mid x > 0\}.\\ \texttt{Bob} \to \texttt{Carol} : \{y \mid y > x\}\end{array} \tag{1.1}$$

(1.1) describes a protocol with three participants, `Alice`, `Bob`, and `Carol`, who agree on a "contract" constraining the interaction variables x and y.

The contract stipulates that (*i*) `Alice` has to send `Bob` a positive value x in the first interaction, and that (*ii*) `Bob` is obliged to send `Carol` a value y strictly greater than x, fixed by `Alice` in the first interaction. Notice that `Bob` can fulfill his pledge (i.e., the predicate $y > x$ in the second interaction above) only after he has received the value x 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 on `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 [7].

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 $3 > y > x$ then `Bob` could not fulfill his contract if `Alice` had fixed the value 2 for x in the first interaction. Remarkably, a global assertion not satisfying TS may lead to conversations in which progress is not guaranteed unless one of the participants deliberately violates the contract.

Guaranteeing HS and TS is often non-trivial, and this burden is on the software architect; using tools like the ones described in [9], 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. Our results include: two algorithms to solve HS problems, one algorithm to solve TS problems, and a methodology for applying the algorithms to protocol design. We show that the algorithms satisfy the following properties: *structure preservation*, i.e., they do not modify the underlying type structure (Proposition 2 and Proposition 4), *properties preservation*, i.e., they do not introduce new violations (Proposition 5), and *correctness*, i.e., if applicable they correct all the problems (Theorems 1 and 2).

This article is the full version of the extended abstract published in [4]. We include here extended definitions and explanations, as well as proofs for our key results, and sketches of proofs for trivial ones. In addition, we discuss the practical applications of the methodology in a case study.

Structure of the paper The preliminary notions used in the rest of the paper are given in § 2. In § 3 we give two algorithms to fix HS violations; 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 the occurrence of predicates earlier in the global assertion in order to remove TS violations. § 5 outlines a methodology based on the three algorithms. § 6 illustrates the methodology via an example. Conclusions and future work are discussed in § 7.

2 Preliminaries

Let $\mathcal{P}$ (ranged over by $\mathtt{p}, \mathtt{q}, \mathtt{s}, \mathtt{r}, \ldots$) and $\mathcal{V}$ (ranged over by $u, v, x, y, \ldots$) be two countably infinite sets of identifiers. We assume $\mathcal{P} \cap \mathcal{V} = \varnothing$ 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 $\vec{x}\,\vec{y}$;

abusing notation, we identify a one-element list with its (unique) element and identify lists with the underlying sets of their elements (e.g., $a \in \vec{x}$ indicates that a occurs in the list $\vec{x}$). As in [3], we parameterise our constructions by abstracting away from the logical language Ψ adopted. Here, it suffices to assume that Ψ is a decidable fragment of a first-order logic obtained by adding first-order quantification to a language of boolean expressions. In fact, we allow expressions (ranged over by e) that include constructors and operators/relations of common data types (e.g., strings, integers, booleans, etc.) and include variables drawn from $\mathcal{V}$. (For simplicity, our examples use basic numeric types or strings.) We write $\mathit{var}(e)$ to denote the set of variables occurring in e and use the symbol $\supset$ to denote logic implication. Then a predicate $\psi \in \Psi$ is either a boolean expression e (understood to be a boolean expression in our language of expressions), or a quantified predicate $\forall \vec{v}.e$ or $\exists \vec{v}.e$. Given a predicate ψ in Ψ, $\mathit{var}(\psi)$ is the set of free interaction variables of ψ (we write $\psi(\vec{v})$ to emphasise that $\mathit{var}(\psi) \subseteq \vec{v}$).

The main ingredients of global assertions are *interactions*, abbreviated ι, like

$$\mathtt{s} \to \mathtt{r} : \{\vec{v} \mid \psi\} \tag{2.1}$$

where $\mathtt{s}, \mathtt{r} \in \mathcal{P}$ are the *sender* and the *receiver*, $\vec{v} \subseteq \mathcal{V}$ is a pairwise distinct list of variables, and $\psi \in \Psi$. We say that the variables $\vec{v}$ in (2.1) are *introduced* by $\mathtt{s}$. The interaction (2.1) reads as "$\mathtt{s}$ has to send to $\mathtt{r}$ some values for $\vec{v}$ that satisfy ψ" or as "$\mathtt{r}$ relies that the values fixed by $\mathtt{s}$ for $\vec{v}$ satisfy ψ". For instance,[3]

$$\mathtt{s} \to \mathtt{r} : \{v\ w \mid \exists u.v = u \times w\}$$

states that $\mathtt{s}$ has the obligation to send $\mathtt{r}$ two values such that the first is a multiple of the second. Given ι as in (2.1), we define

$$\mathit{snd}(\iota) \stackrel{\text{def}}{=} \mathtt{s}, \qquad \mathit{rcv}(\iota) \stackrel{\text{def}}{=} \mathtt{r}, \qquad \mathit{var}(\iota) \stackrel{\text{def}}{=} \vec{v}, \qquad \text{and} \qquad \mathit{cst}(\iota) \stackrel{\text{def}}{=} \psi$$

Remark 1. *In [3], interactions specify a channel over which participants communicate. In (2.1) and in the rest of the paper we omit channels since they are inconsequential to our results. In fact, the algorithms we present do not use identities of channels but only those of participants and variables.*

[3] For simplicity, we assume the typing of variables understood.

Global assertions are ranged over by $\mathcal{G}$ and have the following syntax:

$\mathcal{G}$	::=	$\iota.\mathcal{G}$	Prefix
	|	$\mathtt{s} \to \mathtt{r} : \left\{\{\psi_j\} l_j : \mathcal{G}_j\right\}_{j \in J}$	Branching
	|	$\mu\ \mathbf{t}\ \langle \vec{e} \rangle \{ \vec{v} \mid \psi \}.\mathcal{G}$	Recursive definition
	|	$\mathbf{t}\langle \vec{e} \rangle$	Recursive call
	|	end	End session

where $\psi, \psi_j \in \Psi$ and l_j ranges over a set of labels.

The syntax above is essentially borrowed from [3] but for a slightly simplified notation. In [3], the semantics of global assertions is given in terms of *endpoint assertions* (by projecting global assertions to endpoint assertions and exploiting the operational semantics of the latter). For the sake of this paper, only the syntactic aspects of global assertions given below are relevant; therefore, we give an informal account of the semantics of global assertions.

The prefix production $\iota.\mathcal{G}$ defines a global assertion where the interaction ι must precede the interactions in $\mathcal{G}$. The branching production allows the selector $\mathtt{s}$ to choose one of the labels l_j (with j in a finite index set J) and send it to $\mathtt{r}$, then the interactions in $\mathcal{G}_j$ occur. Recursion is dealt with as usual but for the presence of an initialisation vector $\vec{e}$ (of the same length as $\vec{v}$) which specifies the initial values of each formal parameter in $\vec{v}$ and onto which a recursion invariant ψ is specified. Finally, the last production represents a completed global assertion; trailing occurrences of end are often omitted.

In a recursive definition $\mu\ \mathbf{t}\ \langle \vec{e} \rangle \{ \vec{v} \mid \psi \}.\mathcal{G}$, occurrences of $\mathbf{t}$ (i.e., recursive calls) in $\mathcal{G}$ must be prefix-guarded and the length of $\vec{e}$ is the same as $\vec{v}$; also, we assume that variables $\mathbf{t}$ are always in the scope of a recursive definition $\mu\ \mathbf{t}\ \langle \_ \rangle \{ \_ \mid \_ \}$.

We denote with $var(\mathcal{G})$ the set of interaction variables and recursion parameters in $\mathcal{G}$. The interaction variables $var(\iota)$ of global assertion $\iota.\mathcal{G}$ are bound in $\mathcal{G}$ and in $cst(\iota)$; similarly, the formal parameters $\vec{v}$ in a recursive definition $\mu\ \mathbf{t}\ \langle \_ \rangle \{ \vec{v} \mid \psi \}.\mathcal{G}$ are bound in ψ and in the recursion body $\mathcal{G}$. We consider *closed* global assertions (i.e. for any occurrence of $v \in \mathcal{V}$ in $\mathcal{G}$ either the occurrence is in a recursive definition having v as formal parameter or there is an interaction ι in $\mathcal{G}$ such that $v \in var(\iota)$ that precedes that occurrence of v).

Remark 2. *For simplicity, we assume that bound variables are pairwise distinct.*

Definition 1 (Knows)**.** *Under the syntactic restrictions listed above, we say that a participant* p knows *a variable $v \in var(\mathcal{G})$ if one of the following conditions holds:*

- *there is ι in $\mathcal{G}$ such that $v \in var(\iota)$ and $\mathtt{p} \in \{snd(\iota), rcv(\iota)\}$ or*
- *there is a recursive definition $\mu\ \mathbf{t}\ \langle \vec{e_1}\ e\ \vec{e_2}\rangle\{\vec{v_1}\ v\ \vec{v_2}\ \mid\ \psi\}.\mathcal{G}'$ in $\mathcal{G}$ such that* p *knows all the variables in $var(e)$ and, for each recursive invocation $\mathbf{t}\langle \vec{e_1'}\ e'\ \vec{e_2'}\rangle$ in $\mathcal{G}'$,* p *knows all the variables in $var(e')$.*

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

Example 1. *Consider the following global assertion*

$$
\begin{array}{rcl}
\mathcal{G}_{ex1} & = & \mathtt{I} \to \mathtt{Server} : \{x \mid x \geq 3\}. \\
& & \mu\ \mathbf{t}\ \langle 3\rangle\{r \mid \mathsf{true}\}.\mathtt{Server} \to \mathtt{Player} : \\
& & \quad \{r > x\}\ \mathsf{less} : \mathtt{Player} \to \mathtt{Server} : \{y \mid \mathsf{true}\}.\mathbf{t}\langle y\rangle, \\
& & \quad \{r < x\}\ \mathsf{greater} : \mathtt{Player} \to \mathtt{Server} : \{z \mid \mathsf{true}\}.\mathbf{t}\langle z\rangle, \\
& & \quad \{r = x\}\ \mathsf{win} : \mathsf{end}
\end{array}
$$

where I *initialises a value $x \geq 3$ for* Server*. Then, repeatedly,* Server *sends a label chosen in the set $\{\mathsf{less}, \mathsf{greater}, \mathsf{win}\}$ to* Player *depending of r being greater, smaller, or equal to the value of x; and* Player *replies with an integer in the first two cases while the interaction ends if* win *was sent by* Server*. In $\mathcal{G}_{ex1}$, both* I *and* Server *know x while* Player *does not know it; instead the recursion parameter r is known only to* Server *and* Player*.*

It is convenient to treat global assertions as trees whose nodes are drawn from a set $\mathcal{N}$ (ranged over by $n, n', \ldots$) and labelled with information on the syntactic categories of the syntax of global assertions. 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.

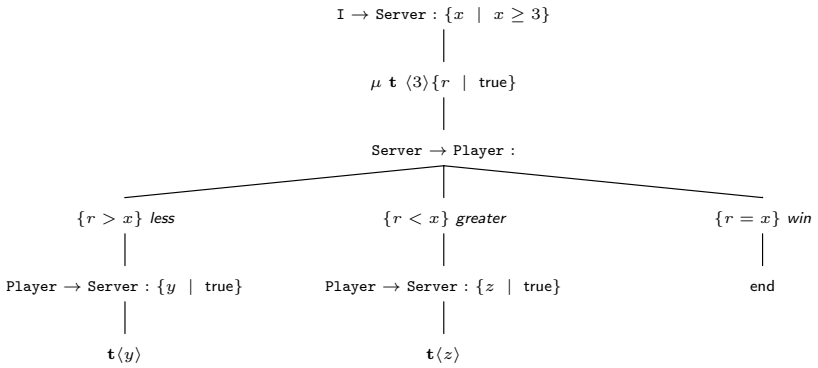
Definition 2 (Assertion Tree)**.** *The* assertion tree $\mathtt{T}(\mathcal{G})$ *of a global assertion $\mathcal{G}$ is defined as follows:*

- *If $\mathcal{G} = \iota.\mathcal{G}'$ then $\mathtt{T}(\mathcal{G})^\bullet$ has label ι and its unique child is $\mathtt{T}(\mathcal{G}')$.*

- *If* $\mathcal{G} = \mathtt{s} \to \mathtt{r} : \left\{\{\psi_j\}l_j : \mathcal{G}_j\right\}_{j\in J}$ *then* $\mathtt{T}(\mathcal{G})^\bullet$ *has label* $\mathtt{s} \to \mathtt{r}$ *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* $\mathtt{T}(\mathcal{G}_j)$ *is the unique child of* n_j.
- *If* $\mathcal{G} = \mu\ \mathtt{t}\ \langle\vec{e}\rangle\{\vec{v} \mid \psi\}.\mathcal{G}'$ *then* $\mathtt{T}(\mathcal{G})^\bullet$ *has label* $\mu\ \mathtt{t}\ \langle\vec{e}\rangle\{\vec{v} \mid \psi\}$ *and its unique child is* $\mathtt{T}(\mathcal{G}')$.
- *If* $\mathcal{G} = \mathbf{t}\langle\vec{e}\rangle$ *then* $\mathtt{T}(\mathcal{G})$ *consists of one node with label* $\mathbf{t}\langle\vec{e}\rangle$.
- *If* $\mathcal{G} = \mathsf{end}$ *then* $\mathtt{T}(\mathcal{G})$ *consists of one node with label* end.

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

Example 2. *The assertion tree for the global assertion of Example 1 can be depicted as:*



where identities of nodes are not shown and only their labels appear.

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

$$var_T : \mathcal{N} \to 2^{\mathcal{V}}, \qquad cst_T : \mathcal{N} \to \Psi, \qquad \text{and} \qquad snd_T, rcv_T : \mathcal{N} \to \mathcal{P}$$

that are undefined[4] on $\mathcal{N} \setminus \{n \mid n \in T\}$ and defined as follows otherwise:

$$var_T(n) = \begin{cases} var(\iota), & \text{if } \underline{n} = \iota \\ \emptyset, & \text{otherwise} \end{cases} \qquad cst_T(n) = \begin{cases} \psi, & \text{if } \underline{n} = \iota \text{ and } cst(\iota) = \psi \\ \psi, & \text{if } \underline{n} = \{\psi\}l \\ \mathsf{true}, & \text{otherwise} \end{cases}$$

$$snd_T(n) = \begin{cases} snd(\iota), & \text{if } \underline{n} = \iota \\ \mathtt{s}, & \text{if } \underline{n} = \mathtt{s} \to \mathtt{r} \end{cases} \qquad rcv_T(n) = \begin{cases} rcv(\iota), & \text{if } \underline{n} = \iota \\ \mathtt{r}, & \text{if } \underline{n} = \mathtt{s} \to \mathtt{r} \end{cases}$$

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

Moreover, we shall use the following functions:

- $\mathit{parent}_T(n)$ returning ϵ if $n = T^\bullet$, the parent of n in T if $n \in T$, and $\bot$ otherwise.
- $n{\uparrow_T}$ returning the path from $T^\bullet$ to n (including n) if $n \in T$, and $\bot$ otherwise.

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

Fact 1. $\mathtt{A}(\mathtt{T}(\mathcal{G})) = \mathcal{G}$

Fact 1 allows us to extend $\mathtt{knows}_\mathtt{p}(\_)$ to $\mathcal{T}$ by $\mathtt{knows}_\mathtt{p}(T) \stackrel{\text{def}}{=} \mathtt{knows}_\mathtt{p}(\mathtt{A}(T))$.

Fact 2. *If* $T \in \mathcal{T}$ *then* $\mathtt{T}(\mathtt{A}(T)) = T$

Facts 1 and 2 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, at each point of the choreography. For global assertions, this distills into *history sensitivity* (HS), a property defined in [3] as follows:

> A predicate guaranteed by a participant p can only contain those interaction variables that p knows.

HS demands the sender/selector of each interaction in a given assertion to know all the variables involved in the predicate associated to that interaction. We illustrate HS with the following example.

Example 3. *The global assertion* $\mathcal{G}_{ex3}$ *below violates HS.*

$$\begin{array}{rcl} \mathcal{G}_{ex3} & = & \mathtt{Alice} \to \mathtt{Bob} : \{v_1 \mid v_1 > 0\}. \\ & & \mathtt{Bob} \to \mathtt{Carol} : \{v_2 \mid v_2 > 0\}. \\ & & \mathtt{Carol} \to \mathtt{Alice} : \{v_3 \mid v_3 > v_1\} \end{array}$$

In fact, Carol*'s obligation* $v_3 > v_1$ *cannot be fulfilled because* $v_1 \notin \mathtt{knows}_{\mathtt{Carol}}(\mathcal{G}_{ex3})$.

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

$$\overline{\mathtt{HS}}(\mathcal{G}) \stackrel{\text{def}}{=} \{n \in \mathtt{T}(\mathcal{G}) \mid var(cst_T(n)) \not\subseteq \mathtt{knows_s}(n\!\uparrow_T) \text{ and } \mathtt{s} = \mathtt{resp}_{\mathtt{T}(\mathcal{G})}(n)\}$$

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

$$\mathtt{resp}_T(n) \stackrel{\text{def}}{=} \begin{cases} snd_T(n), & \text{if } \underline{n} = \iota \\ snd_T(parent_T(n)), & \text{if } \underline{n} = \{\psi\}l \\ \bot, & \text{otherwise} \end{cases}$$

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

Given $T \in \mathcal{T}$, $\mathtt{varHS_T}(\_) : \mathcal{N} \to 2^{\mathcal{V}}$ is defined as

$$\mathtt{varHS_T}(n) \stackrel{\text{def}}{=} var(cst_T(n)) \setminus \mathtt{knows_s}(n\!\uparrow_T) \qquad \text{where} \quad \mathtt{s} = \mathtt{resp}_T(n)$$

Namely, $\mathtt{varHS_T}(n)$ yields the variables of n not known to the responsible party of n. It is a simple observation that if HS is violated at 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 $n \in \overline{\mathtt{HS}}(\mathcal{G})$ then $\mathtt{varHS_T}(n) \neq \varnothing$).

Example 4. *Consider the following global assertion:*

$$\begin{array}{rcl} \mathcal{G}_{ex4} & = & \mu\, \mathbf{t}\ \langle 10\rangle\{v \mid v > 0\}. \\ & & \quad \mathtt{Alice} \to \mathtt{Bob} : \{v_1 \mid v \geq v_1\}. \\ & & \quad \mathtt{Bob} \to \mathtt{Carol} : \{v_2 \mid v_2 > v_1\}. \\ & & \quad \mathtt{Carol} \to \mathtt{Alice} : \{v_3 \mid v_3 > v_1\}. \\ & & \quad \mathtt{Carol} \to \mathtt{Bob} : \{v_4 \mid v_4 > v\}. \\ & & \quad \mathbf{t}\langle v_1\rangle \end{array}$$

$\overline{\mathtt{HS}}(\mathcal{G}_{ex4}) = \{n_3, n_4\}$ *where* n_3 *and* n_4 *are the nodes in* $\mathtt{T}(\mathcal{G}_{ex4})$ *corresponding to the third and fourth interactions of* $\mathcal{G}_{ex4}$*, i.e.* $\underline{n_3} = \mathtt{Carol} \to \mathtt{Alice} : \{v_3 \mid v_3 > v_1\}$ *and* $\underline{n_4} = \mathtt{Carol} \to \mathtt{Bob} : \{v_4 \mid v_4 > v\}$.

In Example 4, `Carol` is responsible for both violations (i.e., $\mathtt{resp}_{\mathtt{T}(\mathcal{G}_{ex4})}(n_3) = \mathtt{resp}_{\mathtt{T}(\mathcal{G}_{ex4})}(n_4) = \mathtt{Carol}$). The violation in n_3 is on $\mathtt{varHS}_{\mathtt{T}(\mathcal{G}_{ex4})}(n_3) = \{v_1\}$ (i.e., `Carol` has to choose v_3 so that $v_3 > v_1$ without knowing v_1) and the violation in n_4 is on $\mathtt{varHS}_{\mathtt{T}(\mathcal{G}_{ex4})}(n_4) = \{v\}$ (i.e., `Carol` has to choose v_4 so that $v_4 > v$ without knowing v). Note that the violation of HS above 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 4 as the running example of § 3.1 and § 3.2.

3.1 Strengthening

Throughout this section we fix a global assertion $\mathcal{G}$ and its assertion tree $T = \mathtt{T}(\mathcal{G})$ and assume HS is violated at $n \in T$ with $cst_T(n) = \psi$ and $\mathtt{resp}_T(n) = \mathtt{s}$.

Violations occur when the responsible party $\mathtt{s}$ of n ignores at least one variable $v \in var(\psi)$. The strengthening algorithm (cf. Definition 4) replaces ψ in $\mathcal{G}$ with a predicate $\psi[v'/v]$ so that

(1) v' is a variable that $\mathtt{s}$ knows,

(2) if $\psi[v'/v]$ and the predicates occurring from $T^\bullet$ to $parent_T(n)$ are satisfied then also ψ is satisfied.

Intuitively, the method above *strengthens* ψ by replacing it with $\psi[v'/v]$ so that: due to (1) the presence of variable v, which is unknown to the sender/selector, is removed, and due to (2) ψ can still be guaranteed. In fact, relying on the information provided by all the predicates occurring before n, if the sender/selector guarantees $\psi[v'/v]$ then (s)he also guarantees ψ. If there is no variable v' that ensures (1) and (2) then we say that *strengthening is not applicable*.

Let $\mathtt{PRED}_T : \mathcal{N} \to \Psi$ yield the conjunction of the predicates on the path from $T^\bullet$ to the parent of a node:

$$\mathtt{PRED}_T(n) \stackrel{\text{def}}{=} \begin{cases} \bot, & \text{if } parent_T(n) = \bot \\ \mathsf{true}, & \text{if } parent_T(n) = \epsilon \\ cst_T(parent_T(n)) \ \wedge \ \mathtt{PRED}_T(parent_T(n)), & \text{otherwise} \end{cases}$$

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

Definition 3 (`strengthen`). *If* $\overline{\mathtt{HS}}(\mathcal{G}) = \varnothing$ *then* $\mathtt{strengthen}(\mathcal{G})$ *returns* $\mathcal{G}$. *If* $n \in \overline{\mathtt{HS}}(\mathcal{G})$, $v \in \mathtt{varHS_T}(n)$ *and there exists* $v' \in \mathtt{knows_s}(n{\uparrow_T})$ *such that*

$$\mathtt{PRED}_T(n) \wedge \psi[v'/v] \supset \psi \quad \textit{with} \quad \psi = cst_T(n) \tag{3.1}$$

then $\mathtt{strengthen}(\mathcal{G})$ *returns* $\mathtt{A}(T')$ *where* T' *is obtained from* T *by replacing* ψ *with* $\psi[v'/v]$ *in* $\underline{n}$.

When Condition 3.1 does not hold for any $v \in \mathtt{knows_s}(n{\uparrow_T})$, $\mathtt{strengthen}(\mathcal{G})$ *returns* $\mathcal{G}\not\downarrow_n$, *indicating that* $\mathcal{G}$ *violates HS at* $n \in \overline{\mathtt{HS}}(\mathcal{G})$.

The algorithm Σ in Definition 4 recursively applies $\mathtt{strengthen}(\_)$ until either the global assertion satisfies HS or Σ is not applicable anymore.

Definition 4 (Σ). *The algorithm* Σ *is defined as follows*

$$\Sigma(\mathcal{G}) \stackrel{def}{=} \begin{cases} \mathtt{strengthen}(\mathcal{G}), & \textit{if} \quad \mathtt{strengthen}(\mathcal{G}) \in \{\mathcal{G}, \mathcal{G}\not\downarrow_n\} \\ \Sigma(\mathtt{strengthen}(\mathcal{G})), & \textit{otherwise} \end{cases}$$

Example 5. *Consider* $\mathcal{G}_{ex4}$ *from Example 4 and recall that* $\overline{\mathtt{HS}}(\mathcal{G}_{ex4}) = \{n_3, n_4\}$. *Strengthening is applicable to* n_3 *since by substituting* v_1 *with* v_2 *in* $v_3 > v_1$ *(with* $v_2 \in \mathtt{knows_{Carol}}(n_3{\uparrow_{\mathtt{T}(\mathcal{G}_{ex4})}})$*) we have that condition (3.1) in Definition 3 holds:*

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

The invocation of $\mathtt{strengthen}(\mathcal{G}_{ex4})$ *returns (by substituting* v_1 *with* v_2*):*

$$\begin{array}{rcl} \mathcal{G}' & = & \mu\, \mathtt{t}\ \langle 10\rangle\{v \mid v > 0\}. \\ & & \quad \mathtt{Alice} \to \mathtt{Bob} : \{v_1 \mid v \geq v_1\}. \\ & & \quad \mathtt{Bob} \to \mathtt{Carol} : \{v_2 \mid v_2 > v_1\}. \\ & & \quad \mathtt{Carol} \to \mathtt{Alice} : \{v_3 \mid v_3 > v_2\}. \\ & & \quad \mathtt{Carol} \to \mathtt{Bob} : \{v_4 \mid v_4 > v\}. \\ & & \quad \mathtt{t}\langle v_1\rangle \end{array}$$

The invocation of $\mathtt{strengthen}(\mathcal{G}')$ *returns* $\mathcal{G}'\not\downarrow_{n_4}$ *since* $\mathcal{G}'$ *has still one violating node* n_4 *for which strengthening is not applicable. In fact,* $\mathtt{knows_{Carol}}(n_4{\uparrow_{\mathtt{T}(\mathcal{G}')}}) = \{v_2, v_3\}$ *and:*

- *by substituting* v *with* v_2, *condition (3.1) in Definition 3 does not hold since* $(v > 0 \wedge v \geq v_1 \wedge v_2 > v_1 \wedge v_3 > v_2) \wedge (v_4 > v_2) \not\supset (v_4 > v)$
- *by substituting* v *with* v_3, *condition (3.1) in Definition 3 does not hold since* $(v > 0 \wedge v \geq v_1 \wedge v_2 > v_1 \wedge v_3 > v_2) \wedge (v_4 > v_3) \not\supset (v_4 > v)$.

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 5 ($\prec_T$)**.** *Let $n, n' \in T$, $n \prec_T n'$ iff n appears in $n'\!\uparrow_T$ and $rcv_T(n) = snd_T(n')$. A vector of nodes $n_1, \ldots, n_t$ is a* chain in T *(or a $\prec_T$-chain) iff $n_i \prec_T n_{i+1}$ for all $i \in \{1, \ldots, t-1\}$.*

The relation $\prec_T$ is similar to the IO-dependency defined in [8] but does not consider branching, since a branching does not carry interaction variables.

Given a chain $\vec{n} = n_1 \cdots n_t$ in T, let the *propagation in $\vec{n}$ of $v_0 \in var_T(n_t)$* be the tree $T' \in \mathcal{T}$ obtained by updating the nodes in T as follows:

- for $i = 1, \ldots, t-1$, $var_{T'}(n_i) = var_T(n_i)\ v_i$ and $cst_{T'}(n_i) = cst_T(n_i) \wedge (v_i = v_{i-1})$, with $v_1, \ldots, v_{t-1} \in \mathcal{V}$ fresh and pairwise distinct
- $cst_{T'}(n_t) = cst_T(n_t)[v_{t-1}/v_0]$ (note that $t > 1$)
- all the other nodes of T remain unchanged.

For a sequence of nodes $\vec{n}$, $\mathtt{P}_T(v_0, \vec{n})$ denotes T' as computed above if $\vec{n}$ is a $\prec_T$-chain and $\bot$ otherwise.

Example 6. *In the global assertion $\mathcal{G}_{ex6}$ below assume* `Alice` *knows v from previous interactions (the ellipsis in $\mathcal{G}_{ex6}$).*

$$\begin{array}{rcll} \mathcal{G}_{ex6} & = & \ldots & \mathtt{Alice} \to \mathtt{Bob} : \{u_1 \mid \psi_1\}. \\ & & & \mathtt{Bob} \to \mathtt{Carol} : \{u_2 \mid \psi_2\}. \\ & & & \mathtt{Bob} \to \mathtt{Dave} : \{u_3 \mid \psi_3\}. \\ & & & \mathtt{Dave} \to \mathtt{Alice} : \{u_4 \mid u_4 > v\} \end{array}$$

For the chain $\vec{n} = n_1\ n_3\ n_4$ in $\mathtt{T}(\mathcal{G}_{ex6})$ (where n_i corresponds to the i-th interaction in $\mathcal{G}_{ex6}$), $\mathtt{P}_{\mathtt{T}(\mathcal{G}_{ex6})}(v, \vec{n})$ returns T' such that $\mathtt{A}(T')$ is

$$\begin{array}{rcll} \mathcal{G}'_{ex6} & = & \ldots & \mathtt{Alice} \to \mathtt{Bob} : \{u_1\ v_1 \mid \psi_1 \wedge v = v_1\}. \\ & & & \mathtt{Bob} \to \mathtt{Carol} : \{u_2 \mid \psi_2\}. \\ & & & \mathtt{Bob} \to \mathtt{Dave} : \{u_3\ v_2 \mid \psi_3 \wedge v_1 = v_2\}. \\ & & & \mathtt{Dave} \to \mathtt{Alice} : \{u_4 \mid u_4 > v_2\} \end{array}$$

Hereafter, we fix a global assertion $\mathcal{G}$; let $T = \mathtt{T}(\mathcal{G})$, $n \in \overline{\mathtt{HS}}(\mathcal{G})$, $v \in \mathtt{varHS}_{\mathtt{T}}(n)$, and $\mathtt{s} = \mathtt{resp}_T(n)$. The *propagation* algorithm (cf. Definition 7) 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 and has $\mathtt{s}$ as receiver down to n.

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

Definition 6 (`propagate`)**.** *The function* $\mathtt{propagate}(\mathcal{G})$ *returns*

- $\mathcal{G}$*, if* $\overline{\mathtt{HS}}(\mathcal{G}) = \varnothing$
- $\mathtt{P}_T(v, \vec{n})$*, if* $T = \mathtt{T}(\mathcal{G})$ *and there exists* $n \in \overline{\mathtt{HS}}(\mathcal{G})$*,* $v \in \mathtt{varHS}_{\mathtt{T}}(n)$*, and* $\vec{n} = n_0\, \vec{n_1}\, n$ *chain in* T *such that* $snd_T(n_0)$ *knows* v
- $\mathcal{G}\!\not\downarrow_n$ *with* $n \in \overline{\mathtt{HS}}(\mathcal{G})$ *otherwise.*

Example 7. *Consider again the global assertion* $\mathcal{G}'$ *obtained after the invocation* $\mathtt{strengthen}(\mathcal{G}_{ex4})$ *in Example 5. In this case* $\overline{\mathtt{HS}}(\mathcal{G}') = \{n_4\}$ *with* $\underline{n_4} = \mathtt{Carol} \to \mathtt{Bob} : \{v_4 \mid v_4 > v\}$*. Propagation is applicable to* n_4 *and* $\mathtt{propagate}(\mathcal{G}')$ *returns*

$$
\begin{array}{rcl}
\mathcal{G}'' & = & \mu\, \mathtt{t}\ \langle 10\rangle\{v \mid v > 0\}.\\
& & \mathtt{Alice} \to \mathtt{Bob} : \{v_1 \mid v \geq v_1\}.\\
& & \mathtt{Bob} \to \mathtt{Carol} : \{v_2\ u_1 \mid v_2 > v_1 \wedge u_1 = v\}.\\
& & \mathtt{Carol} \to \mathtt{Alice} : \{v_3 \mid v_3 > v_2\}.\\
& & \mathtt{Carol} \to \mathtt{Bob} : \{v_4 \mid v_4 > u_1\}.\\
& & \mathtt{t}\langle v_1\rangle
\end{array}
$$

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 $\mathtt{propagate}(\_)$.

Definition 7 (Π)**.** *Given a global assertion* $\mathcal{G}$*, the function* Π *is defined as follows:*

$$
\Pi(\mathcal{G}) = \begin{cases} \mathtt{propagate}(\mathcal{G}), & \text{if } \ \mathtt{propagate}(\mathcal{G}) \in \{\mathcal{G}, \mathcal{G}\!\not\downarrow_n\} \\ \Pi(\mathtt{propagate}(\mathcal{G})), & \text{otherwise} \end{cases}
$$

3.3 Properties of Σ and Π

We now discuss the properties of the global assertions amended by each algorithm and we compare them. Hereafter, we say Σ (resp. Π) returns $\mathcal{G}$ if either it returns $\mathcal{G}$ or it returns $\mathcal{G}\!\!\not\downarrow_n$ for some n.

The applicability of Σ 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 Definition 3 is satisfied. The applicability of Π depends on whether there exists a chain through which the problematic variable can be propagated. Observe also that there are cases in which Σ is applicable and Π is not, and vice versa. Moreover, $\Sigma(\mathcal{G}) \neq \Pi(\mathcal{G})$ in general, hence it may not always be clear which one should be preferred.

Remark 3. *Linearity of the underlying multiparty session types [8] guarantees the existence of a dependency chain between the interactions. However, linearity does not guarantee that* Π *is always applicable. The reason is that* $n_1 \prec n_2$ *in the sense of [8] does not imply* $n_1 \prec_T n_2$ *since* $\prec_T$ *does not take into account branching but only interactions.*

Remark 4. *In distributed applications it is often necessary to guarantee that exchanged information is accessible only to intended participants. It is worth observing that* Π *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* Π*.*

One could think of an extension of $\texttt{propagate}(\mathcal{G})$ *which propagates variables only to participants entitled to know them. The existence of a propagation chain* $\vec{n}$ *for a variable* v *may be parameterised by two sets of participants chosen by the architect: a set* A *containing the participants who are allowed to know the value of* v *and a set* N *of participants not allowed to know it. Let* $T = \mathtt{T}(\mathcal{G})$*, and* v_0 *a variable causing an HS problem in* $\mathcal{G}$*, an acceptable chain* $\vec{n}$ *is defined as in § 3.2 and such that*

- *for all* $n \in \vec{n}$*,* $rcv_T(n) \notin N$*, and*
- *there is no other chain* $\vec{n}'$ *with* $\mathtt{P}_T(v_0, \vec{n}') \neq \bot$ *such that* $|P(\vec{n})| < |P(\vec{n}')|$*, where*

$$P(\vec{n}) = \{\mathtt{r} \mid \textit{there exist } n \in \vec{n} \textit{ such that } \mathtt{r} = rcv_T(n) \in A\}$$

Note that even though this additional condition provides a more fine-grained control on the way problems are solved, it also decreases the appli-

cability range of the algorithm since the existence of such a chain is not guaranteed.

We now show that the weakening and propagation algorithms terminate.

Lemma 1. *Let $\mathcal{G}$ be a global assertion;* $\texttt{strengthen}(\mathcal{G})$ *(resp.* $\texttt{propagate}(\mathcal{G})$*) always returns $\mathcal{G}'$ such that $\texttt{T}(\mathcal{G}')$ is isomorphic to $\texttt{T}(\mathcal{G})$, namely it has the same tree structure (but possibly different labels).*

Proof: By Definition 3 $\texttt{strengthen}(\mathcal{G})$ always returns either $\mathcal{G}$ (which includes the case for $\mathcal{G}\!\!\downarrow_n$) or $\mathcal{G}'$. $\texttt{T}(\mathcal{G}')$ is isomorphic to $\texttt{T}(\mathcal{G})$ as the two trees either are the same or only differ in the label of one node. By Definition 6 $\texttt{propagate}(\mathcal{G})$ always returns either $\mathcal{G}$ (which includes the case for $\mathcal{G}\!\!\downarrow_n$) or $\mathcal{G}' = \texttt{P}_{\texttt{T}(\mathcal{G})}(v_0, \vec{n})$ for some chain $\vec{n}$ and variable v_0. By definition of $\texttt{P}_{\texttt{T}(\mathcal{G})}(v_0, \vec{n})$, $\texttt{T}(\mathcal{G}')$ only differs from $\texttt{T}(\mathcal{G})$ in the labels of the nodes in $\vec{n}$, hence $\texttt{P}_{\texttt{T}(\mathcal{G})}(v_0, \vec{n})$ is isomorphic to $\texttt{T}(\mathcal{G})$. □

Next we show that both Σ and Π do not change the structure of the given global assertion.

Proposition 1. *Let $\mathcal{G}$ be a global assertion. If it terminates, $\Sigma(\mathcal{G})$ (resp. $\Pi(\mathcal{G})$) returns $\mathcal{G}'$ such that $\texttt{T}(\mathcal{G}')$ is isomorphic to $\texttt{T}(\mathcal{G})$.*

Proof: We proceed by induction on the number of recursive invocations of Σ. We omit the proof for Π since it is similar. In the base case $\Sigma(\mathcal{G}) = \texttt{strengthen}(\mathcal{G})$ (first case of Definition 4); by Lemma 1) $\texttt{strengthen}(\mathcal{G})$ always returns $\mathcal{G}'$ such that $\texttt{T}(\mathcal{G})$ is isomorphic to $\texttt{T}(\mathcal{G}')$, hence $\Sigma(\mathcal{G})$ returns $\mathcal{G}'$ such that $\texttt{T}(\mathcal{G})$ is isomorphic to $\texttt{T}(\mathcal{G}')$ as required. In the inductive case $\Sigma(\mathcal{G}) = \Sigma(\texttt{strengthen}(\mathcal{G}))$ (second case of Definition 4); by inductive hypothesis if $\Sigma(\texttt{strengthen}(\mathcal{G}))$ returns $\mathcal{G}'$ then $\texttt{T}(\mathcal{G}')$ is isomorphic to $\texttt{T}(\texttt{strengthen}(\mathcal{G}))$, and by Lemma 1 $\texttt{T}(\texttt{strengthen}(\mathcal{G}))$ is isomorphic to $\texttt{T}(\mathcal{G})$; the thesis follows by transitivity of isomorphism. □

We next show that Σ and Π terminate as a corollary of Lemma 2.

Lemma 2. *Let $\mathcal{G}$ be a global assertion, $T = \texttt{T}(\mathcal{G})$, and k be the number of HS violations in T.*[5] *Let k_1, k_2 be the number of HS violations in $T_1 = \texttt{T}(\texttt{strengthen}(\mathcal{G}))$ and $T_2 = \texttt{T}(\texttt{propagate}(\mathcal{G}))$, respectively; then either $T_i = T$ or $k_i = k - 1$ with $i \in \{1, 2\}$.*

[5] More than one violation may occur in one node if the sender does not know several variables.

Proof: By Proposition 1, T and T_1 are isomorphic. Let $n_1 \in T_1$ be the node corresponding to $n \in T$. By definition of $\overline{\texttt{HS}}$ the violation in a node $n' \in T_1$ is defined only in terms of the nodes in $n'{\uparrow_{T_1}}$. By definition of $\texttt{strengthen}(\_)$ the only node from which T differs from T_1 is n_1. Hence, if a violation is added in T_1 with respect to T it must be in the subtree of T_1 rooted at n_1. However, a violation is not added in n_1 itself since the variable chosen to replace the problematic one is selected so that the responsible party *knows* it. No violation can be added in the subtree rooted at n' since $\texttt{strengthen}(\_)$ does not modify the variables known by the participants (but only the predicates). Thus, either $T_1 = T$ or $k_1 = k - 1$. In the case $T_2 = \texttt{T}(\texttt{propagate}(\mathcal{G}))$, assume $\hat{n} \in \overline{\texttt{HS}}(T)$ with $\underline{\hat{n}} = \texttt{s} \to \texttt{r} : \{\vec{x} \mid \phi\}$, $v \in \texttt{varHS}_\texttt{T}(\hat{n})$, and there exists an $\prec_T$-chain $n_0\ \vec{n}\ \hat{n}$ with $\underline{n_0} = \texttt{s}_\texttt{0} \to \texttt{r}_\texttt{0} : \{\vec{y_0} \mid \psi_0\}$ such that $\texttt{s}_0$ knows v. We proceed by case analysis showing that, in any node $n \in T$, such that $n \neq \hat{n}$, no violation is introduced, and exactly one violation is removed from $\hat{n}$.

- if $n = n_0$ no violation is introduced since $\underline{n_0}$ becomes $\texttt{s}_\texttt{0} \to \texttt{r}_\texttt{0} : \{\vec{y}\ v_0 \mid \psi_0 \wedge v_0 = v\}$ in T_2 where, by Definition 6, $\texttt{s}_\texttt{0}$ knows v.
- if $n_i \in \vec{n}$ by definition of propagation $\underline{n_i} = \texttt{s}_\texttt{i} \to \texttt{r}_\texttt{i} : \{\vec{y_i} \mid \psi_i\}$ becomes $\texttt{s}_\texttt{i} \to \texttt{r}_\texttt{i} : \{\vec{y_i}\ v_i \mid \psi_i \wedge v_{i-1} = v_i\}$ in T_2 where, by Definition 5, $\texttt{r}_{i-1} = \texttt{s}_i$. It follows that $\texttt{s}_i$ has previously received v_{i-1} hence he/she knows it.
- if $n = \hat{n}$, following Definition 5, $cst_{T_2}(\hat{n}) = \phi[v_0/v]$ and the problem on v at $\hat{n}$ has been solved since $\texttt{s}$ knows v_0, and v does not appear in ϕ anymore, hence $k_2 = k - 1$.
- if n does not belong to the $\prec_T$-chain $n_0\ \vec{n}\ \hat{n}$ then n remains unchanged and no violation is introduced. Note that if n is in the subtree of rooted at $\hat{n}$, no violation is introduced since propagation does not decrease the knowledge of any participant.

□

Corollary 1. *Let $\mathcal{G}$ be a global assertion; $\Sigma(\mathcal{G})$ and $\Pi(\mathcal{G})$ terminate.*

Whereas Σ does not change the global type underlying the global assertion, Π 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 $\mathtt{erase}(\mathcal{G})$ be the function that returns the underlying global type [8] corresponding to $\mathcal{G}$ (i.e. a global assertion without predicates).

Definition 8. *Given a global assertion $\mathcal{G}$, erase($\mathcal{G}$) is defined inductively as:*

$$\mathtt{erase}(\mathcal{G}) = \begin{cases} \mathtt{s} \to \mathtt{r} : .\mathtt{erase}(\mathcal{G}') & \text{if } \mathcal{G} = \mathtt{s} \to \mathtt{r} : \{\vec{v} \mid \psi\}.\mathcal{G}' \\ \mathtt{s} \to \mathtt{r} : \Big(l_j : \mathtt{erase}(\mathcal{G}_j)\Big)_{j\in J} & \text{if } \mathcal{G} = \mathtt{s} \to \mathtt{r} : \Big(\{\psi_j\}l_j : \mathcal{G}_j\Big)_{j\in J} \\ \mu\ \mathbf{t}\ .\mathtt{erase}(\mathcal{G}') & \text{if } \mathcal{G} = \mu\ \mathbf{t}\ \langle\vec{e}\rangle\{\vec{v} \mid \psi\}.\mathcal{G}' \\ \mathcal{G} & \text{if } \mathcal{G} = \mathsf{end} \text{ or } \mathcal{G} = \mathbf{t} \end{cases}$$

Lemma 3. *Let $\mathcal{G}$ and $\mathcal{G}'$ be two assertions that differ only in the predicates annotating interactions and branchings. Then erase($\mathcal{G}$) = erase($\mathcal{G}'$).*

Proof: The proof is by structural induction on $\mathcal{G}$.
End session. If $\mathcal{G} = \mathsf{end}$ and $\mathcal{G}' = \mathsf{end}$ the thesis follows from the fact that $\mathtt{erase}(\mathcal{G}) = \mathtt{erase}(\mathcal{G}') = \mathsf{end}$.
Recursive call. If $\mathcal{G} = \mathbf{t}\langle e\rangle$ and $\mathcal{G}' = \mathbf{t}\langle e\rangle$ the thesis follows from the fact that $\mathtt{erase}(\mathcal{G}) = \mathtt{erase}(\mathcal{G}') = \mathbf{t}$.
Interaction (prefix). Let $\mathcal{G} = \mathtt{s} \to \mathtt{r} : \{\vec{v} \mid \psi\}.\mathcal{G}_0$ and $\mathcal{G}' = \mathtt{s} \to \mathtt{r} : \{\vec{v} \mid \psi'\}.\mathcal{G}'_0$ for some ψ' and $\mathcal{G}'_0$. By Definition 8, $\mathtt{erase}(\mathcal{G}) = \mathtt{s} \to \mathtt{r} : .\mathtt{erase}(\mathcal{G}_0)$ and $\mathtt{erase}(\mathcal{G}') = \mathtt{s} \to \mathtt{r} : .\mathtt{erase}(\mathcal{G}'_0)$, hence

$$\mathtt{erase}(\mathcal{G}) = \mathtt{erase}(\mathcal{G}') \tag{3.2}$$

The thesis follows from (3.2) and from the fact that by inductive hypothesis $\mathtt{erase}(\mathcal{G}_0) = \mathtt{erase}(\mathcal{G}'_0)$ holds.
Branching. Let $\mathcal{G} = \mathtt{s} \to \mathtt{r} : \Big(\{\psi_j\}l_j : \mathcal{G}_j\Big)_{j\in J}$ and $\mathcal{G}' = \mathtt{s} \to \mathtt{r} : \Big(\{\psi'_j\}l_j : \mathcal{G}'_j\Big)_{j\in J}$ for some ψ'_j and $\mathcal{G}'_j$. By Definition 8, $\mathtt{erase}(\mathcal{G}) = \mathtt{s} \to \mathtt{r} : \Big(l_j : \mathtt{erase}(\mathcal{G}_j)\Big)_{j\in J}$ and $\mathtt{erase}(\mathcal{G}') = \mathtt{s} \to \mathtt{r} : \Big(l_j : \mathtt{erase}(\mathcal{G}'_j)\Big)_{j\in J}$, hence (3.2) holds also in this case. The thesis follows from (3.2) and by the fact that by inductive hypothesis $\mathtt{erase}(\mathcal{G}_j) = \mathtt{erase}(\mathcal{G}'_j)$ for all $j \in J$.
Recursive definition. Let $\mathcal{G} = \mu\,\mathbf{t}\ \langle\vec{e}\rangle\{\vec{v} \mid \psi\}.\mathcal{G}_0$ and $\mathcal{G}' = \mu\,\mathbf{t}\ \langle\vec{e}\rangle\{\vec{v} \mid \psi\}.\mathcal{G}'_0$ for some $\mathcal{G}'_0$. By Definition 8, $\mathtt{erase}(\mathcal{G}) = \mu\ \mathbf{t}\ .\mathtt{erase}(\mathcal{G}_0)$ and $\mathtt{erase}(\mathcal{G}') = \mu\ \mathbf{t}\ .\mathtt{erase}(\mathcal{G}'_0)$, hence (3.2) holds also in this case. The thesis follows from (3.2) and by the fact that by inductive hypothesis $\mathtt{erase}(\mathcal{G}_0) = \mathtt{erase}(\mathcal{G}'_0)$. □

Lemma 4. *Let $\mathcal{G}$ be a global assertion and $T = \mathtt{T}(\mathcal{G})$. Given a chain $\vec{n} = n_1, \ldots, n_t$ in T, if $\mathtt{P}_T(v_0, \vec{n}) = \mathcal{G}'$ for an interaction variable v_0, then for all $n \in \mathtt{T}(\mathcal{G})$ and its corresponding node $n' \in \mathtt{T}(\mathcal{G}')$,*

$$var_{\mathtt{T}(\mathcal{G})}(n) \subseteq var_{\mathtt{T}(\mathcal{G}')}(n')$$

Proof: Observe that, by Proposition 1, T is isomorphic to T'. Let $\vec{n} = n_1, \ldots, n_t$, $n \in T$, $n' \in T'$, and n' be the corresponding node of n in T'. We proceed by case analysis on the (labels of the) nodes of T, which we divide in three groups following the definition of $\mathtt{P}_T(v_0, \vec{n})$:

- if $n = n_i$ with $i \in \{1, \ldots, t-1\}$ then $var_{T'}(n) = var_T(n')\ v_i$.
- if $n = n_t$ then $var_{T'}(n) = var_T(n')$.
- if $n \notin \vec{n}$ then n' has the same label as n since by definition of $\mathtt{P}_T(v_0, \vec{n})$ all nodes not in $\vec{n}$ are unchanged.

In all cases the above thesis holds. □

Proposition 2 (Underlying Type Structure)**.** *Let $\mathcal{G}$ be a global assertion,*

- *if $\Sigma(\mathcal{G})$ returns $\mathcal{G}'$ then $\mathtt{erase}(\mathcal{G}) = \mathtt{erase}(\mathcal{G}')$*
- *if $\Pi(\mathcal{G})$ returns $\mathcal{G}'$ then for all $n \in \mathtt{T}(\mathcal{G})$ and its corresponding node $n' \in \mathtt{T}(\mathcal{G}')$,*
$$var_{\mathtt{T}(\mathcal{G})}(n) \subseteq var_{\mathtt{T}(\mathcal{G}')}(n')$$

Proof: As to Σ, we first observe that $\mathtt{strengthen}(\mathcal{G})$ either returns $\mathcal{G}$ or a $\mathcal{G}'$ that differs from $\mathcal{G}$ only in the predicate of one (interaction or branching) node. By Lemma 3:

$$\mathtt{erase}(\mathcal{G}) = \mathtt{erase}(\mathtt{strengthen}(\mathcal{G})) \tag{3.3}$$

We proceed by induction on the number of applications of Σ. In the base case, Σ returns $\mathtt{strengthen}(\mathcal{G})$ and the thesis follows immediately from (3.3). In the inductive case we have to prove that $\mathtt{erase}(\Sigma(\mathcal{G}))$ is equal to $\mathtt{erase}(\mathcal{G})$. By definition of Σ, in the inductive case $\Sigma(\mathcal{G}) = \Sigma(\mathtt{strengthen}(\mathcal{G}))$. By inductive hypothesis $\mathtt{erase}(\Sigma(\mathtt{strengthen}(\mathcal{G})) = \mathtt{erase}(\mathtt{strengthen}(\mathcal{G}))$. Finally, by (3.3) $\mathtt{erase}(\mathtt{strengthen}(\mathcal{G})) = \mathtt{erase}(\mathcal{G})$, which yields the thesis.

As to Π, we first observe that if $\mathtt{propagate}(\mathcal{G}) = \mathcal{G}'$ then for all $n \in \mathtt{T}(\mathcal{G})$ and its corresponding node $n' \in \mathtt{T}(\mathcal{G}')$,

$$var_{\mathtt{T}(\mathcal{G})}(n) \subseteq var_{\mathtt{T}(\mathcal{G}')}(n') \tag{3.4}$$

In fact, $\texttt{propagate}(\mathcal{G})$ either returns $\mathcal{G}' = \mathcal{G}$ or $\mathcal{G}' = \mathtt{P}_{\mathtt{T}(\mathcal{G})}(v_0, \vec{n})$. In the former case (3.4) holds trivially, in the latter case it holds by Lemma 4. To prove the thesis for Π we now proceed by induction on the number of applications of Π. In the base case, Π returns $\texttt{propagate}(\mathcal{G})$ and the thesis follows immediately from (3.4). In the inductive case we have to prove that $\texttt{erase}(\Pi(\mathcal{G}))$ is equal to $\texttt{erase}(\mathcal{G})$. By definition of Π, in the inductive case $\Pi(\mathcal{G}) = \Pi(\texttt{propagate}(\mathcal{G}))$. By inductive hypothesis $\texttt{erase}(\Pi(\texttt{propagate}(\mathcal{G})) = \texttt{erase}(\texttt{propagate}(\mathcal{G}))$. Finally, by (3.4) $\texttt{erase}(\texttt{propagate}(\mathcal{G})) = \texttt{erase}(\mathcal{G})$, which yields the thesis. □

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

Proposition 3 (Assertion Predicates)**.** *Let $\mathcal{G}$ be a global assertion,*

1. *if $\Sigma(\mathcal{G})$ returns $\mathcal{G}'$ then for all $n \in \mathtt{T}(\mathcal{G})$ whose label is modified by Σ, and its corresponding node $n' \in \mathtt{T}(\mathcal{G}')$ (cf. Proposition 2), it holds that $\mathtt{PRED}_{\mathtt{T}(\mathcal{G}')}(n') \wedge cst_{\mathtt{T}(\mathcal{G}')}(n') \supset cst_{\mathtt{T}(\mathcal{G})}(n)$*
2. *if $\Pi(\mathcal{G})$ returns $\mathcal{G}'$ then for all $n \in \mathtt{T}(\mathcal{G})$ whose label is modified by Π, and its corresponding node $n' \in \mathtt{T}(\mathcal{G}')$*
   (a) *$cst_{\mathtt{T}(\mathcal{G}')}(n')$ is the predicate $cst_{\mathtt{T}(\mathcal{G})}(n)\sigma \wedge \psi$*
   (b) *$\mathtt{PRED}_{\mathtt{T}(\mathcal{G})}(n) \wedge cst_{\mathtt{T}(\mathcal{G})}(n) \wedge \psi \iff \mathtt{PRED}_{\mathtt{T}(\mathcal{G}')}(n') \wedge cst_{\mathtt{T}(\mathcal{G}')}(n')$*

   For some satisfiable $\psi \in \Psi$ and variable substitution σ.

Proof: The proof of 1 relies on the fact that Σ either does not change $\mathcal{G}$ or replaces a problematic variable by a variable for which (3.1) holds. We show the result by showing that it holds for each invocation of $\texttt{strengthen}(\_)$ by Σ. Indeed, for each invocation we have that, by Definition 3, if n is modified by Σ, then we have that $n \in \overline{\mathtt{HS}}(\mathtt{T}(\mathcal{G}))$. In addition, there must be $v \in \mathtt{varHS}_{\mathtt{T}(\mathcal{G})}(n)$ such that there exists $v' \in \mathtt{knows}_{\mathtt{s}}(n{\uparrow_{\mathtt{T}(\mathcal{G})}})$ and (3.1) is satisfied. This gives us

$$\underbrace{\mathtt{PRED}_{\mathtt{T}(\mathcal{G})}(n)}_{\mathtt{PRED}_{\mathtt{T}(\mathcal{G}')}(n')} \wedge \underbrace{\psi[v'/v]}_{cst_{\mathtt{T}(\mathcal{G}')}(n')} \supset \underbrace{\psi}_{cst_{\mathtt{T}(\mathcal{G})}(n)}$$

Since only the predicate of node n is updated by substituting v by v', by Definition 3.

The proof of 2 relies on Definition 5, i.e. a predicate of the form $v_1 = v_0$ or $v_i = v_{i-1}$ is added to each predicate of the nodes in the chain, and problematic variables are replaced by fresh ones. The additional predicates are satisfiable since they constrain only fresh variables (i.e. v_i). We have these results by showing that each invocation of `propagate`(_) by Π validates the result.

Case 2a If n is modified by `propagate`(_), then $n \in \vec{n}$, by Definition 6. Assume v_0 is the variable to be propagated.

- If n is not the last node of $\vec{n}$, by definition of $\mathtt{P}_{\mathtt{T}(\mathcal{G})}(v_0, \vec{n})$, we have that $cst_{\mathtt{T}(\mathcal{G}')}(n') = cst_{\mathtt{T}(\mathcal{G})}(n) \wedge (v_i = v_{i-1})$, which gives us the expected result if ψ is $v_i = v_{i-1}$ and σ is the empty substitution.

- If n is the last node of $\vec{n}$, by definition of $\mathtt{P}_{\mathtt{T}(\mathcal{G})}(v_0, \vec{n})$, we have that $cst_{\mathtt{T}(\mathcal{G})}(n') = cst_{\mathtt{T}(\mathcal{G})}(n)[v_{t-1}/v_0]$, which gives the expected result with $\sigma = [v_{t-1}/v_0]$ and $\psi = \mathsf{true}$.

Case 2b If n is modified by `propagate`(_), then $n \in \vec{n}$, by Definition 6. Assume v_0 is the variable to be propagated and the length of $\vec{n}$ is k.

- If n is the first node of $\vec{n}$, then $cst_{\mathtt{T}(\mathcal{G}')}(n') = cst_{\mathtt{T}(\mathcal{G})}(n) \wedge (v_1 = v_0)$, and $\mathtt{PRED}_{\mathtt{T}(\mathcal{G}')}(n') = \mathtt{PRED}_{\mathtt{T}(\mathcal{G})}(n)$ since $n'{\uparrow}_{\mathtt{T}(\mathcal{G}')}$ is unchanged. We have the expected result if ψ is $v_1 = v_0$. Note that by definition of $\mathtt{P}_{\mathtt{T}(\mathcal{G})}(v_0, \vec{n})$, v_1 is a fresh variable therefore $v_1 = v_0$ is satisfiable.

- If n is the i^{th} node in $\vec{n}$ $(1 < i < k)$ then $cst_{\mathtt{T}(\mathcal{G}')}(n') = cst_{\mathtt{T}(\mathcal{G})}(n) \wedge (v_i = v_{i-1})$, and

$$\mathtt{PRED}_{\mathtt{T}(\mathcal{G}')}(n') = \mathtt{PRED}_{\mathtt{T}(\mathcal{G})}(n) \wedge \bigwedge_{1<j<i} v_j = v_{j-1}$$

  where each $v_j = v_{j-1}$ is satisfiable since each variable is freshly introduced. We have the expected result with ψ as $v_i = v_{i-1}$.

- If n is the last node in $\vec{n}$, then $cst_{\mathtt{T}(\mathcal{G}')}(n') = cst_{\mathtt{T}(\mathcal{G})}(n)[v_k/v_0]$, and

$$\mathtt{PRED}_{\mathtt{T}(\mathcal{G}')}(n') = \mathtt{PRED}_{\mathtt{T}(\mathcal{G})}(n) \wedge \bigwedge_{1<j<k} v_j = v_{j-1}$$

  with each $v_j = v_{j-1}$ satisfiable, as before. Let ψ be true, and we have the required result.

□

Proposition 3.2 (b) amounts to saying that $cst_{\mathtt{T}(\mathcal{G})}(n) \wedge \psi$ is equivalent to $cst_{\mathtt{T}(\mathcal{G}')}(n')$ when such predicates are taken in their respective contexts.

Remarkably, Σ and Π do not add violations (of either HS or TS) to the amended global assertions. We postpone the discussion of this property to § 4.3 (Proposition 5) after the formal introduction of TS.

Finally, we prove that if the value returned by Σ or Π is not of the type $\mathcal{G}'_{\lightning n}$ then the amended global assertion satisfies HS.

Theorem 1 (Correctness)**.** *If there is* $\mathcal{G}'$ *such that* $\Sigma(\mathcal{G}) = \mathcal{G}'$ *or* $\Pi(\mathcal{G}) = \mathcal{G}'$ *then* $\overline{\mathtt{HS}}(\mathcal{G}') = \varnothing$.

Proof: **Case** Σ**.** By Definitions 3 and 4, Σ terminates successfully when $\overline{\mathtt{HS}}(\mathcal{G}) = \varnothing$. We show that at each iteration of Σ, the number of HS violations decreases. Assume that there is k violations in $\mathcal{G}$, by Definition 4, we have either

- $\Sigma(\mathcal{G}) = \mathtt{strengthen}(\mathcal{G}) = \mathcal{G}$ in which case, by Definition 3, $\overline{\mathtt{HS}}(\mathcal{G}) = \varnothing$, i.e. $k = 0$, and the function terminates, or
- $\Sigma(\mathcal{G}) = \Sigma(\mathtt{strengthen}(\mathcal{G})) = \mathcal{G}'$ with $\mathcal{G} \neq \mathcal{G}'$, and by Lemma 2 the number of HS violation in $\mathtt{strengthen}(\mathcal{G})$ is strictly less than k.

Case Π**.** The case for Π is similar to the previous case, using Definition 6 (resp. 7) instead of Definition 3 (resp. 4).

Finally, note that both Σ and Π always terminate since (i) the number of violations decreases at each iteration and (ii) we only consider finite assertion trees, therefore the number of variables in a tree is also finite. □

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 value 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 9 (TS [3]). *A global assertion* $\mathcal{G}$ satisfies TS *(in symbols* $\mathtt{TS}(\mathcal{G})$*) iff* $GSat(\mathcal{G}, \mathsf{true})$ *holds where*

$$GSat(\mathcal{G},\psi) \textit{iff} \begin{cases} GSat(\mathcal{G}', \psi \wedge cst(\iota)), & \textit{if } \mathcal{G} = \iota.\mathcal{G}' \textit{ and } \psi \supset \exists var(\iota).cst(\iota) \\ \bigwedge_{j\in J} GSat(\mathcal{G}_j, \psi \wedge \psi_j), & \textit{if } \mathcal{G} = \mathtt{s} \to \mathtt{r} : \left(\{\psi_j\} l_j : \mathcal{G}_j\right)_{j\in J} \textit{ and } \psi \supset \bigvee_{j\in J} (\psi_j) \\ GSat(\mathcal{G}', \psi \wedge \psi'), & \textit{if } \mathcal{G} = \mu\ \mathbf{t}\ \langle\vec{e}\rangle\{\vec{v}\ \mid\ \psi'\}.\mathcal{G}' \textit{ and } \psi \supset \psi'[\vec{e}/\vec{v}] \\ GSat(\mathcal{G}', \psi \wedge \psi'), & \textit{if } \mathcal{G} = \mathbf{t}_{\psi'(\vec{v})}\langle\vec{e}\rangle \textit{ and } \psi \supset \psi'[\vec{e}/\vec{v}] \\ \mathcal{G} = \mathsf{end}, & \textit{otherwise} \end{cases}$$

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

Intuitively, the predicate ψ in Definition 9 is the conjunction of all the predicates that precede an interaction in $\mathcal{G}$. In the first case, all the values satisfying ψ allow to instantiate the interaction variables $var(\iota)$ so to satisfy the constraint $cst(\iota)$ of ι. For branching, *GSat* requires that at least one branch can be chosen and that each possible path satisfies *GSat*. For recursive definition, we require that the initial parameters satisfy the invariant ψ'. We assume that recursive calls are annotated with the invariant of the corresponding recursive definition, i.e. in Definition 9, $\psi'(\vec{v})$ is the predicate corresponding to the invariant of the definition of $\mathbf{t}$. 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 8. *Consider* $\mathcal{G}_{ex8}$ *below, where* $\mathtt{p}$ *constrains* x *and* y*:*

$$\begin{array}{rcl} \mathcal{G}_{ex8} & = & \mathtt{p} \to \mathtt{q} : \{x \mid x < 10\}. \\ & & \mathtt{p} \to \mathtt{q} : \{y \mid y > 8\}. \\ & & \mathtt{q} \to \mathtt{p} : \{z \mid x > z \wedge\ z > 6\ \wedge\ y \neq z\} \end{array}$$

When $\mathtt{q}$ *introduces* z*, both* x *and* y *are further restricted.* $\mathcal{G}_{ex8}$ *violates TS because it does not hold that*

$$\forall x\, y\,.\,(x < 10) \wedge (y > 8) \supset \exists z\,.\,(x > z \wedge\ z > 6\ \wedge\ y \neq z)$$

Noticeably, if $\mathtt{p}$ *chooses, e.g.* $x = 6$ *then* $\mathtt{q}$ *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 8, one could lift the predicate $\exists z.x > z > 6$ (adapted from the last interaction) to the first interaction's predicate.

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; for the sake of clarity, 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* n violates TS.

Definition 10 (TSnode)**.** *Given $T \in \mathcal{T}$, $\mathtt{TSnode}_T(n)$ holds iff $n \in T$, and $\mathtt{TS}(T')$ 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 of the form* $\mathtt{s} \to \mathtt{r} :$ *(since there is no predicate in these nodes, no TS problem can arise).*

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 11 ($\overline{\mathtt{TS}}$)**.** *The function $\overline{\mathtt{TS}} : \mathcal{T} \to \mathcal{N}$ is defined as follows:*

$$\overline{\mathtt{TS}}(T) \stackrel{def}{=} \left\{ n \in T \;\middle|\; \begin{array}{c} \mathtt{TSnode}_T(n) \text{ is false, and } \mathtt{TSnode}_T(n') \\ \text{is true for all } \; n' \in parent_T(n){\uparrow_T} \end{array} \right\}$$

For instance, in Example 8, we have that $\overline{\mathtt{TS}}(T_{\text{ex8}})$ is the singleton $\{n_{\text{ex8}}\}$ where $T_{\text{ex8}} = \mathtt{T}(\mathcal{G}_{\text{ex8}})$ and n_{ex8} is the node corresponding to the last interaction of $\mathcal{G}_{\text{ex8}}$.

Once an *interaction* node $n \in \overline{\mathtt{TS}}(T)$ is chosen, the next step is to identify which part of its predicate is the source of the problem. Thus, we define a relation among predicates ψ and ϕ in a context ψ' to identify the problematic part of a predicate in an interaction node.

Definition 12 (Conflict)**.** *The predicate $\psi \in \Psi$* is in conflict *on $\vec{v} \subseteq \mathcal{V}$ with ϕ* in *ψ' iff*

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

The notion of conflict is based on the definition of TS for interaction nodes (Definition 9). On the one hand, there is the part of predicate which does not spoil TS (ϕ), and, on the other hand, the part which in conjunction with ϕ invalidates TS (ψ).

In Example 8, we have

$$\forall x\, y\,.\, x < 10 \wedge y > 8 \supset \exists z\,.\, y \neq z \;\text{ and }\; \forall x\, y\,.\, x < 10 \wedge y > 8 \not\supset \exists z\,.\, x > z \wedge z > 6 \wedge y \neq z$$

Using Definition 12 and $\texttt{PRED}_T(n)$ (cf. § 3), we define

$$\texttt{split}_T\,(n, \psi) \stackrel{\text{def}}{=} \left\{ \psi' \;\middle|\; \psi \iff \psi' \wedge \phi \text{ and } \psi' \textit{ is in conflict on } \textit{var}(n) \text{ with } \phi \textit{ in } \texttt{PRED}_T(n) \right\}$$

which returns a set of problematic predicates. Considering again Example 8, the application of `split` yields $\texttt{split}_{T_{\text{ex8}}}\,(n_{\text{ex8}}, z > 6 \wedge x > z \wedge y \neq z) = \{z > 6 \wedge x > z\}$ since $y \neq z$ allows to choose a suitable value for z.

Remark 5. *For a tree $T \in \mathcal{T}$ and $n \in \overline{\texttt{TS}}(T)$ such that $\psi' \in \texttt{split}_T\,(n, \psi)$, we may have $\texttt{PRED}_T(n) \not\supset \exists \vec{v}.\psi'$. For instance, if the predicate ψ' is not satisfiable, e.g., $\psi' = v < 7 \wedge v > 7$. In this case the algorithm is not applicable.*

Remark 6. *Note that at this level, it is not necessary to require ψ' to be minimal in the definition of* `split` *(in terms of, e.g. the size of the formula or the number of variables in ψ'). Indeed, as stated later in Definition 14, only the predicate which can be lifted successfully are used by the algorithm. However, an implementation of the algorithm could minimise the predicate in order to maximise the efficiency of the lifting algorithm.*

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 13 (`build`)**.** *The function $\texttt{build}_T\,(n, \psi)$ returns*

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

  $$\mathtt{s} \to \mathtt{r} : \{\vec{u} \mid \theta \wedge \forall \vec{x}.\exists \vec{y}.\psi\} \quad \textit{such that} \quad \theta \wedge \forall \vec{x}.\exists \vec{y}.\psi \textit{ is satisfiable}$$

  where

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

and there is no $n'' \in parent_T(n)\uparrow_T$ *such that* $\underline{n''} = \mu\ \mathtt{t}\ \langle\vec{e}\rangle\{\vec{v} \mid \psi'\}$ *and* $\vec{v} \cap var(\psi) \neq \varnothing$.

- $\bot$ *otherwise.*

Essentially, Definition 13 duplicates a quantified version of the predicate ψ in the nodes which introduce variables in $var(\psi)$. For each updated node n', the quantification of the variables in $var(\psi)$ operates in the following way. The variables which are introduced before n' in the tree and which are not known to `s` are quantified *universally* (since `s` has no control over them). The variables that are introduced later in the tree are quantified *existentially*, so that `s` may choose values for the variables in $\vec{u}$ which do not compromise the satisfiability of predicates down in the tree.

Remark 7. *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* $var(\psi)$ *are either quantified in the predicate of* $\hat{n}$*, or have been introduced before* n'.

In Example 8, we would invoke $\mathtt{build}_{T_{\text{ex8}}}(n_{\text{ex8}}, z > 6 \wedge x > z)$ which returns a new assertion tree. The new tree can be transformed into a global assertion isomorphic to $\mathcal{G}_{\text{ex8}}$ with line 1 updated to: $\mathtt{p} \to \mathtt{q} : \{x \mid x < 10 \wedge \exists z.x > z > 6\}$.

The function $\mathtt{TSres} : \mathcal{T} \times \mathcal{N} \to \mathcal{T} \cup \bot$ brings the above definitions together in order to either fix a TS problem at n, or return $\bot$.

Definition 14 (`TSres`)**.** *Given* $T \in \mathcal{T}$ *and* $n \in \overline{\mathtt{TS}}(T)$*, we define*

$$\mathtt{TSres}_T(n) = \begin{cases} \mathtt{build}_T(n, \psi), & \text{if } \underline{n} = \iota \text{ and } \exists \psi \in \mathtt{split}_T(n, cst_T(n)) \\ & \text{s.t. } \mathtt{build}_T(n, \psi) \neq \bot \\ \mathtt{build}_T(n, \psi[\vec{e}/\vec{v}]), & \text{if } \underline{n} = \mu\ \mathtt{t}\ \langle\vec{e}\rangle\{\vec{v} \mid \psi\} \\ \bot, & \text{otherwise} \end{cases}$$

The first case of Definition 14 handles TS problems in interaction nodes. If there is a predicate ψ in conflict such that it can be "lifted" by `build` successfully, then the function returns the result of `build`. The second case 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., $\vec{e}$). It may be possible to lift the recursion invariant, where we replace the recursion parameters by the corresponding initialisation vector. Example 9 illustrates this case.

Example 9. *For the global assertion $\mathcal{G}_{ex9}$ given below, $\mathtt{TS}(\mathcal{G}_{ex9})$ does not hold because $\forall x\, y.\mathsf{true} \not\supset (x > y > 6)$.*

$$\begin{array}{rcl} \mathcal{G}_{ex9} & = & \mathtt{p} \to \mathtt{q} : \{x \mid \mathsf{true}\}. \\ & & \mu\ \mathtt{t}\ \langle 8\rangle\{y \mid x > y > 6\}.\mathcal{G}' \end{array}$$

However, using the initialisation parameters, we can lift $x > 8 > 6$, 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 $\mathtt{TS}(\mathcal{G}')$ holds as well).

Remark 8. *In Example 9, 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., $x = 8$).*

Even though lifting may be applied when a TS violation is detected in a recursion definition, 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 (cf. the last line of the first part of Definition 13). In fact, for recursive definition and calls, Definition 9 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* (Definition 9) 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.

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 15 (Λ)**.** Λ *is defined as follows, given a global assertion $\mathcal{G}$.*

$$\Lambda(\mathcal{G}) = \begin{cases} \mathcal{G}, & \text{if } \mathtt{TS}(\mathcal{G}) \\ \Lambda(\mathtt{TSres}_{\mathtt{T}(\mathcal{G})}(n)), & \text{if there is } n \in \overline{\mathtt{TS}}(\mathtt{T}(\mathcal{G})) \text{ s.t. } \mathtt{TSres}_{\mathtt{T}(\mathcal{G})}(n) \neq \bot \\ \mathcal{G}\not\downarrow_n, & \text{otherwise} \end{cases}$$

4.2 Applying Λ to Branching and Recursion

Branching. According to Definition 9, TS fails on branching nodes only when there are values for which none of the branches' predicates are satisfiable, as in Example 10 below.

The underlying idea of branching is to enable the architect to design a choreography where a branch cannot be taken when some variables have a particular value. The architect should be involved in the resolution of the problem, because 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 other branches to be chosen, as shown in Example 10.

Example 10. *As an illustration, we consider the following assertion:*

$$\begin{array}{rcl} \mathcal{G}_{ex10} & = & \mathtt{q} \to \mathtt{p} : \{v \mid \mathsf{true}\}. \\ & & \mathtt{p} \to \mathtt{q} : \quad \{v > 5\}\, l_1 : \mathcal{G}_1 \\ & & \qquad\quad \{v < 5\}\, l_2 : \mathcal{G}_2 \end{array}$$

Assuming that $\mathtt{TS}(\mathcal{G}_1)$ *and* $\mathtt{TS}(\mathcal{G}_2)$ *hold, we have that* $\mathtt{TS}(\mathcal{G}_{ex10})$ *does not hold because* $\mathsf{true} \not\supset (v > 5 \vee v < 5)$*. 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 $\mathcal{G}_{\text{ex10}}$. 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).

- $\mathtt{build}_{\mathtt{T}(\mathcal{G}_{\text{ex10}})}(\hat{n}, v > 5 \vee v < 5)$ replaces the predicate in the first line by $\mathsf{true} \wedge (v > 5 \vee v < 5)$
- $\mathtt{build}_{\mathtt{T}(\mathcal{G}_{\text{ex10}})}(\hat{n}, v < 5)$ replaces the predicate in the first line by $\mathsf{true} \wedge (v < 5)$.

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 node[6] $n \in T$ such that TS does not hold. One can invoke $\mathtt{build}_T(n, \psi)$ where ψ is either the disjunction of all the branching predicates or one of the branches predicate. If the function does not return $\bot$, then the TS problem is solved.

[6] We also assume that TS is not violated in $\mathit{parent}_T(n)\!\uparrow_T$ as in Definition 11.

Recursion. The lifting algorithm can easily be extended to solve TS problems which occur in a recursive call, if we assume an annotation giving the invariant of its corresponding recursive definition (as in Defintion 9). In fact, let a TS problem be located at a node $n \in T$ such that $\underline{n} = \mathbf{t}\langle\vec{e}\rangle$ and let the invariant of the definition of $\mathbf{t}$ be $\psi(\vec{v})$, then if the invocation of $\texttt{build}_T\,(n, \psi[\vec{e}/\vec{v}])$ succeeds, the problem is solved.

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

Example 11. *Consider the global assertion below*

$$
\begin{array}{lll}
\mathcal{G}_{ex11} = & \texttt{Generator} \to \texttt{Server} : \{n \mid n > 0\}. & \\
& \texttt{Player} \to \texttt{Server} : \{x \mid \mathsf{true}\}. & \\
& \mu\ \mathbf{t}\ \langle x\rangle\{r \mid r > 0\}. & \\
& \texttt{Server} \to \texttt{Player} : \quad \{r > n\}\ \mathit{less} : & \texttt{Player} \to \texttt{Server} : \{y \mid \mathsf{true}\}.\mathbf{t}\langle y\rangle \\
& \qquad\qquad\qquad\quad \{r < n\}\ \mathit{greater} : & \texttt{Player} \to \texttt{Server} : \{z \mid \mathsf{true}\}.\mathbf{t}\langle z\rangle \\
& \qquad\qquad\qquad\quad \{r = n\}\ \mathit{win} : & \mathsf{end}
\end{array}
$$

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_{ex11} be the tree generated from $\texttt{T}(\mathcal{G}_{\text{ex11}})$. There is a TS problem at the node corresponding to the recursive definition (let's call it n_3), indeed if $x \leq 0$, the invariant is not respected. After the first loop of $\Lambda(T_{\text{ex11}})$, the predicate $x > 0$ is added in the second interaction, i.e. $\texttt{TSres}_{T_{\text{ex11}}}(n_3)$ is invoked and returns a new tree, say T'_{ex11}, where the second interaction is updated to

$$\texttt{Player} \to \texttt{Server} : \{x \mid x > 0\}$$

Then, the algorithm loops two more times to solve the problems appearing before the recursive calls. Assuming n_4 (resp. n_5) is the node corresponding to the recursive call in the *less* (resp. *greater*) branch. $\texttt{TSres}_{T'_{\text{ex11}}}(n_4)$ is invoked, adding $y > 0$ in the interaction of the *less* branch, let's call this new tree T''_{ex11}. The updated interaction is now

$$\texttt{Player} \to \texttt{Server} : \{y \mid y > 0\}$$

Then, the algorithm invokes $\texttt{TSres}_{T''_{\text{ex11}}}(n_5)$, which adds $z > 0$ in the interaction of the *greater* branch, updating the interaction to

$$\texttt{Player} \to \texttt{Server} : \{z \mid z > 0\}$$

The global assertion now satisfies temporal satisfiability.

4.3 Properties of Λ

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

Proposition 4 (Underlying Type Structure - Λ)**.** *Let $\mathcal{G}$ be a global assertion. If $\Lambda(\mathcal{G})$ returns $\mathcal{G}'$ then $erase(\mathcal{G}) = erase(\mathcal{G}')$ (cf. Definition 8).*

Proof: The proof is by induction on the structure of $\mathcal{G}$, similarly to the one of Proposition 2. □

Proposition 5 below guarantees that Λ does not introduce new HS or TS problems. Likewise, Proposition 5 gives a formal account of the informal remark in § 3.3, showing that also Σ and Π do not add violations (of either HS or TS) to the amended global assertions.

Proposition 5 (Properties Preservation)**.** *Assume $F(\mathcal{G})$ returns $\mathcal{G}'$ with $F \in \{\Sigma, \Pi, \Lambda\}$. If $\overline{\mathtt{HS}}(\mathcal{G}) = \varnothing$ then $\overline{\mathtt{HS}}(\mathcal{G}') = \varnothing$ and if $\overline{\mathtt{TS}}(\mathcal{G}) = \varnothing$ then $\overline{\mathtt{TS}}(\mathcal{G}') = \varnothing$.*

Proof: We first consider HS preservation and then TS preservation for $F(\mathcal{G})$ with $F \in \{\Sigma, \Pi, \Lambda\}$.
HS preservation. The proof of HS preservation by Σ and Π follows by the fact that Σ and Π return $\mathcal{G}$ if $\overline{\mathtt{HS}}(\mathcal{G}) = \varnothing$. For Λ, 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. We show that all the variables not known to the sender of an updated node are quantified. Let T be an assertion tree and ψ be the predicate lifted at a node $n \in T$ such that $snd_T(n) = \mathtt{s}$. The predicate is quantified as in Definition 14 so to obtain $\forall\vec{x}.\exists\vec{y}.\psi$ such that

- $\vec{x} \subseteq var(\psi) \setminus \mathtt{knows_s}(T)$ are introduced in a node in $n{\uparrow_T}$
- $\vec{y} \subseteq var(\psi)$ are introduced in a node in the subtree rooted at n

Let $z \in var(\psi)$, (i) if $z \notin \vec{x}$, by definition, either z is known to $\mathtt{s}$ (therefore z should not be quantified) or z is introduced after n (hence it would have been quantified in $\vec{y}$). (ii) If $z \notin \vec{y}$, by definition, either z is introduced at n (therefore known to $\mathtt{s}$) or z is introduced before n. In that case, if it is known to $\mathtt{s}$ then it should not be quantified, and if it is not known to n, then $z \in \vec{x}$.

TS preservation. TS preservation in Σ follows from the fact that predicates may only be changed by a variable substitution. For $T = \mathtt{T}(\mathcal{G})$, such that $\overline{\mathtt{TS}}(\mathcal{G}) = \varnothing$, we have that, for any $n \in T$

$$\mathtt{PRED}_T(n) \supset \exists var_T(n).\phi$$

by definition of TS (Definition 9), with ϕ being the predicate at node n. And, by (3.1), we have that

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

thus, TS is preserved by Σ. TS preservation in Π 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 3). For $T = \mathtt{T}(\mathcal{G})$, such that $\overline{\mathtt{TS}}(\mathcal{G}) = \varnothing$, we have that, for any $n \in T$

$$\mathtt{PRED}_T(n) \supset \exists var_T(n).\phi \tag{4.1}$$

by definition of TS, with ϕ being the predicate at node n. And, by construction of $\prec_T$-chain, after each modification by Π, we obtain

$$\mathtt{PRED}_T(n) \wedge v = v_0 \wedge \ldots \wedge v_t = v_{t-1} \supset \exists var_T(n).\phi[v_t/v]$$

with $v_0 \ldots v_t$ fresh. This is equivalent to (4.1), i.e. TS is preserved by Π. The proof of TS preservation for Λ follows trivially from the first case of Definition 15. □

Proposition 6 establishes an intermediate result for the correctness of Λ. It says that a successful invocation of `TSres` (cf. Definition 14) on a node removes the problem at that node.

Proposition 6 (Correctness - `TSres`)**.** *Let T be an assertion tree. For each $n \in \overline{\mathtt{TS}}(T)$ such that $\mathtt{TSres}_T(n) \neq \bot$, then $n \notin \overline{\mathtt{TS}}(\mathtt{TSres}_T(n))$.*

Proof: We start by giving the proof of the correctness for *interaction nodes*. Let T be an assertion tree with a node n such that $n \in \overline{\mathtt{TS}}(T)$, and

$$\underline{n} = \mathtt{s} \rightarrow \mathtt{r} : \{\vec{v} \mid \psi\}$$

with $\psi \iff \beta \wedge \gamma$ such that β *is in conflict* on $var(n)$ with γ in $\mathtt{PRED}_T(n)$. Then β is the predicate to be lifted. Assume $\hat{T} = \mathtt{build}_T(n, \beta)$.

By Definition 13, we have that, for suitable $\vec{x}_1, \vec{y}_1 \ldots \vec{x}_k, \vec{y}_k$,

$$\mathtt{PRED}_{\hat{T}}(n) = \mathtt{PRED}_T(n) \wedge \forall \vec{x}_1.\exists \vec{y}_1.\beta \wedge \ldots \wedge \forall \vec{x}_k.\exists \vec{y}_k.\beta \tag{4.2}$$

We have that a quantified version of β is added k times in the assertion tree, above n. We show that

$$\bigwedge_{1\leq i\leq k} \forall \vec{x}_i.\exists \vec{y}_i.\beta \supset \exists \vec{v}.\beta \tag{4.3}$$

Assume that each $\forall \vec{x}_i.\exists \vec{y}_i.\beta$ corresponds to the predicate added to the i^{th} node (n_i) modified by `TSres` (from the root to n). Then $\vec{y}_k = \vec{v} \cap var(\beta)$ since by Definition 13 $\vec{y}_k$ is the set of variables introduced *after* n_k, and we assumed that the global assertion is closed (i.e. all the variables $var(\beta)$ have been introduced before they are used, in n). Since every $\forall \vec{x}_i.\exists \vec{y}_i.\beta$ is satisfiable by Definition 13, we have that the following holds

$$\exists \vec{z}.\forall \vec{x}_k.\exists \vec{y}_k.\beta \quad \text{with } \vec{z} = var(\forall \vec{x}_k.\exists \vec{y}_k.\beta)$$

this gives us (4.3) (note that $\vec{z}$, $\vec{x}_k$ and $\vec{y}_k$ are pairwise disjoints by definition). Since $\forall \vec{x}_k.\exists \vec{y}_k.\beta$ is one of the conjunct of $\mathtt{PRED}_{\hat{T}}(n)$ we also have

$$\mathtt{PRED}_{\hat{T}}(n) \supset \exists \vec{v}.\beta \tag{4.4}$$

By the definition of conflict (Definition 12), we have that $\mathtt{PRED}_T(n) \supset \exists \vec{v}.\gamma$ and $\mathtt{PRED}_T(n) \not\supset \exists \vec{v}.\beta \wedge \gamma$ (hence, $\mathtt{PRED}_T(n)$ is satisfiable). Therefore, by weakening, we have that

$$\mathtt{PRED}_{\hat{T}}(n) \supset \exists \vec{v}.\gamma \tag{4.5}$$

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

$$\mathtt{PRED}_{\hat{T}}(n) \supset \exists \vec{v}.\psi \quad (\text{with } \psi \iff \beta \wedge \gamma)$$

Otherwise, that would imply that

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

which is in contradiction with (4.4) (β) and (4.5) (γ).

Let's now show the result for *recursive nodes*, which is somewhat similar to the previous case. Assume we have

$$\underline{n} = \mu\ \mathbf{t}\ \langle \vec{e} \rangle \{ \vec{v}\ \mid\ \beta \}$$

with $n \in \overline{\mathtt{TS}}(T)$, thus we have that (by Definition 9)

$$\mathtt{PRED}_T(n) \not\supset \beta[\vec{e}/\vec{v}] \tag{4.6}$$

Assuming $\hat{T} = \texttt{build}_T(n, \beta[\vec{e}/\vec{v}])$ (i.e. `build` succeeds), we have that a quantified version of $\beta[\vec{e}/\vec{v}]$ is added k times in the assertion tree, above n. Following a similar argument as before, this gives us

$$\texttt{PRED}_{\hat{T}}(n) = \texttt{PRED}_T(n) \wedge \beta[\vec{e}/\vec{v}] \tag{4.7}$$

By Definition 13, we also know that $\beta[\vec{e}/\vec{v}]$ is satisfiable, and by Definition 11 and (4.6), $\texttt{PRED}_T(n)$ is satisfiable.

Since

$$\texttt{PRED}_{\hat{T}}(n) \supset \beta[\vec{e}/\vec{v}] \iff \texttt{PRED}_T(n) \wedge \beta[\vec{e}/\vec{v}] \supset \beta[\vec{e}/\vec{v}]$$

we have that $n \notin \overline{\texttt{TS}}(\hat{T})$. □

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

Theorem 2 (Correctness - Λ)**.** *If* $\Lambda(\mathcal{G}) = \mathcal{G}'$ *then* $\overline{\texttt{TS}}(\mathcal{G}') = \varnothing$.

Proof: 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 6, i.e. the fact that $\texttt{TSres}_T(n)$ either solves the problem at n or fails.

Let $T = \texttt{T}(\mathcal{G})$ and N be the set of nodes in T which violates TS. We write $|n|$ for the depth of n in T (with $|T^\bullet| = 0$), and we denote by N' the number of problematic node after an invocation to `TSres`.

1. If $N = \varnothing$, then T is TS.
2. If $N \neq \varnothing$, let $n \in \overline{\texttt{TS}}(T) \subseteq N$, after an invocation to $\texttt{TSres}_T(n)$, 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. $N' := N \cup N_* \setminus \{n\}$, where N_* is the set of problematic nodes *added* by `TSres`. We have that $\forall n'_i \in N_*. |n'_i| < |n|$, 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 Definition 9), in which case, the algorithm fails.

Note that selecting $n \in \overline{\mathtt{TS}}(T)$ 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 $\overline{\mathtt{TS}}(T) = \varnothing$, 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 Λ invokes `TSres` only when a problematic node is found. □

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

Proposition 7 (Assertion predicates)**.** *If $\Lambda(\mathcal{G}) = \mathcal{G}'$ then for all $n \in \mathtt{T}(\mathcal{G})$ such that n is a leaf, and its corresponding node $n' \in \mathtt{T}(\mathcal{G}')$*

$$\mathtt{PRED}_{\mathtt{T}(\mathcal{G})}(n) \wedge cst_{\mathtt{T}(\mathcal{G})}(n) \iff \mathtt{PRED}_{\mathtt{T}(\mathcal{G}')}(n') \wedge cst_{\mathtt{T}(\mathcal{G}')}(n')$$

Proof: 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. In addition, the algorithm modifies only predicates which appear above a problematic node in the tree (i.e. a predicate in a leaf will never be modified, by Definition 13).

We show the result for *interaction nodes*, the case for recursive nodes is similar. After each successful iteration of Λ on an assertion tree $T = \mathtt{T}(\mathcal{G})$, where

$$\mathtt{PRED}_T(n) = \psi_1 \wedge \ldots \wedge \psi_i \wedge \ldots \wedge \psi_k$$

and ψ_i is the predicate where the TS problem is located, and β is the lifted predicate (cf. proof of Proposition 6), such that we have, by Definition 12

$$\psi_i \iff \beta \wedge \gamma \tag{4.8}$$

We can then rewrite $\mathtt{PRED}_T(n)$ as

$$\mathtt{PRED}_T(n) = \psi_1 \wedge \ldots \wedge \gamma \wedge \beta \wedge \ldots \wedge \psi_k$$

`build` returns a new tree T' such that the conjunction of predicates in the new tree is of the form

$$\mathtt{PRED}_{T'}(n') = \mathtt{PRED}_T(n) \wedge \bigwedge_{j \in J} \forall \vec{x_j} \exists \vec{y_j} . \beta \tag{4.9}$$

By Definition 13, we have $\vec{x}_j\ \vec{y}_j \subseteq var(\beta)$, for all $j \in J$. Therefore, we have

$$\beta \supset \forall \vec{x}_j \exists \vec{y}_j.\beta \quad \text{for all } j \in J \tag{4.10}$$

This means that the additional predicates do not constrain further the conjunction of predicates, and we have

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

□

5 A Methodology for Amending Choreographies

The algorithms Σ, Π, and Λ in § 3 and § 4 can be used to support a methodology for amending contracts in choreographies. The methodology consists of the following steps:

(*i*) the architect designs a choreography $\widehat{\mathcal{G}}$

(*ii*) the architect is notified if there are any HS or TS problems in $\widehat{\mathcal{G}}$

(*iii*) using Σ and Π solutions may be offered for HS problems, while Λ can be used to offer solutions and/or hints on how to solve TS problems

(*iv*) the architect selects one of the solutions offered in (*iii*)

(*v*) steps (*ii*) to (*v*) are repeated until all problems are addressed.

We show our methodology using the following global assertion:

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

which extends the global assertion in Example 4 and is supposed to be designed by the architect (step (*i*) of the methodology).

Firstly, $\widehat{\mathcal{G}}$ is inspected by history sensitivity and temporal satisfiability checkers, such as the ones described in [9]. If any HS problems are reported

(step (*ii*) of the methodology), algorithms Σ and Π are used, while Λ is used for TS problems. This allows the architect to detect all the problems and to consider the ones for which (at least) one of the algorithms is applicable. In general, the architect can decide which problem to tackle first (step (*iii*) of our methodology). For $\widehat{\mathcal{G}}$, we focus on HS problems first. There are two HS problems in $\widehat{\mathcal{G}}$, both of which 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 Σ) at line 4, or
   - by revealing v_1 to `Carol` (algorithm Π); in this case, line 3 becomes

     $$\texttt{Bob} \to \texttt{Carol} : \{v_2\ u_1 \mid v_2 > v_1 \wedge u_1 = v_1\}$$

     and the predicate 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 Π) in which case line 3 becomes

   $$\texttt{Bob} \to \texttt{Carol} : \{v_2\ u_2 \mid v_2 > v_1 \wedge u_2 = v\}$$

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

In the *propagation case* (i.e., Π), the methodology gives the architect information on which participants the value of a variable may be disclosed to. Indeed, as discussed in Remark 4, 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 tacked 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 Σ is used to solve the first problem and Π to solve the second, the first five lines of the new global assertion are those in Example 7 and HS is fixed. Now HS is satisfied in $\widehat{\mathcal{G}}$, but TS problems are still present.

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 satify 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 predicate $v \geq v_1 \wedge v_1 > 0$.

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. Definition 12) 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. Indeed, the updated predicate (i.e. $v \geq v_1 \wedge \exists v_3, v_5 . v_1 < v_5 < v_3 - 2$) does not constrain v_1 more than the original predicate, $v \geq v_1$.
   - $\forall v_1 . \exists v_5 . v_1 < v_5 < v_3 - 2$ which is not satisfiable, therefore the algorithm fails.

The failure of Λ 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 using the proposed algorithms. 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 approach 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`.

5.1 Amendment Strategies

The methodology above does not specify any particular order for tackling HS and TS problems. In fact, it is for the architect to assess the importance of each problem; furthermore, (s)he should also proof-read the proposed solutions (e.g. in the case of propagation). One of our future work plans is to help the architect making choice regarding the order in which problems should be tackled by designing *amendment strategies*, which maximise the chances of having *all* problems solved using the proposed algorithms.

In fact, the application of an algorithm could be spoiled by the application of another one. For instance, the application of the strengthening algorithm (Σ) might compromise the applicability of the lifting algorithm (Λ), and vice versa. This happens when a variable v introduced by Σ in a node (say n) to solve an HS problem is also involved in a TS problem, in

a descendant node (say n') of n; indeed, both Σ and Λ will independently strengthen the predicate of n. This may compromise the application of the algorithm invoked last, as illustrated below.

Let T be an assertion tree, where there are $n, n' \in T$ such that $\texttt{varHS}_\texttt{T}(n) \neq \varnothing$, n' is a descendant of n,

$$\underline{n} = \texttt{s} \to \texttt{r} : \{\vec{v}_1\ v\ \vec{v}_2\ \mid\ \psi\} \qquad \text{and} \qquad \underline{n'} = \texttt{s}' \to \texttt{r}' : \{\vec{u}\ \mid\ \gamma \wedge \beta\}$$

where $v \in \mathit{var}(\beta)$, and β is in conflict on $\vec{u}$ with γ in $\texttt{PRED}_T(n')$.

- if Σ is used to solve the problem at n, ψ might be strengthened (by variable substitution)
- if Λ is used to solve the problem at n', β will be lifted to the node n since $v \in \mathit{var}(\beta)$.

Call ψ_1 the new predicate produced by Σ and ψ_2 the one produced by Λ. The application of Σ would generate

$$\underline{n} = \texttt{s} \to \texttt{r} : \{\vec{v}_1\ v\ \vec{v}_2\ \mid\ \psi_1\}$$

which might prevent the application of Λ because e.g., $\psi_1 \wedge \forall\vec{x}.\exists\vec{y}.\beta$ is not satisfiable, for suitable $\vec{x}$ and $\vec{y}$. Likewise, the application of Λ first would generate

$$\underline{n} = \texttt{s} \to \texttt{r} : \{\vec{v}_1\ v\ \vec{v}_2\ \mid\ \psi_2\}$$

for which Σ might not be applicable because e.g., no substitution with a variable known to $\texttt{s}$ yields a satisfiable predicate for $\underline{n}$.

We conjecture that this is the *only* source of issue arising from the absence of prescribed order for addressing HS and TS problems in the methodology. Intuitively, the only way one algorithm could spoil the applicability of another is by modifying the satisfiability of a predicate of (at least) one common node. Propagation (Π) preserves the semantics of all the nodes it updates (by Propostion 3); instead, Σ and Λ may strengthen predicates. Note that Σ modifies only the node in which there is an HS problem, while Λ updates only the nodes *above* the TS-problematic one. Therefore, the only possible issue occurs when there is a node with an HS problem "above" another, with a TS problem. Since Λ modifies only the nodes that introduce variables which appear in a *problematic* predicate, we conjecture that this happens only in the case explained above. Note that even though an occurrence of two inter-dependent TS and HS problems as above may compromise the applicability of an algorithm, thus preventing the amendment of existing problems, it will not introduce new violations, as stated in Proposition 5.

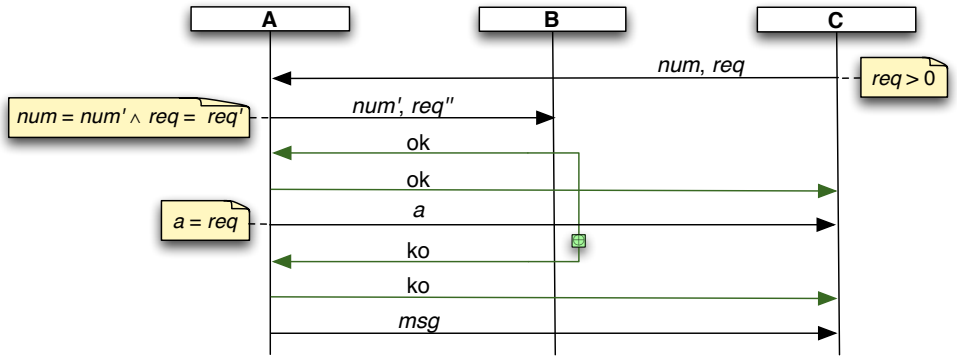


Figure 1: ATM protocol

6 Applying the Methodology

To illustrate our methodology, we consider the design of a couple of services offered by an ATM to the customers of the bank where it is located. The first service offers cash withdrawal. The second service allows customers to request a small line of credit, provided that they are considered trusted by the bank. We propose two global assertions for each of the two functionalities and discuss problems which may be encountered during their design.

6.1 Cash Withdrawal

Consider the following global assertion, also illustrated in Figure 1, where C is the customer, A is the ATM, and B is the bank.

$$\begin{aligned}
&\mathtt{C} \to \mathtt{A} : \{num\ req \mid req > 0\}.\\
&\mathtt{A} \to \mathtt{B} : \{num'\ req' \mid num = num' \wedge req = req'\}.\\
&\mathtt{B} \to \mathtt{A} : \{\mathsf{true}\}\ \mathsf{ok} : \mathtt{A} \to \mathtt{C} : \{\mathsf{true}\}\ \mathsf{ok} : \mathtt{A} \to \mathtt{C} : \{a \mid a = req\},\\
&\qquad \{\mathsf{true}\}\ \mathsf{ko} : \mathtt{A} \to \mathtt{C} : \{\mathsf{true}\}\ \mathsf{ko} : \mathtt{A} \to \mathtt{C} : \{msg \mid \mathsf{true}\}
\end{aligned}$$

This global assertion models a simple cash withdrawal service under the assumption that the credentials of the customer have already been verified. The customer sends the ATM an account number num and the amount of money to withdraw req. The ATM forwards the request to the bank. If the withdrawal is accepted, the bank selects branch ok; in this case the ATM gives the corresponding amount to the customer. Otherwise, the bank selects

branch ko and ATM sends an error message to the customer.

This global assertion is well-asserted, but soon the architect realises that it contains a major flaw: the ATM is expected to give money to the customer even when there is not enough cash available in the machine. The architect corrects the problem by adding a predicate $a \leq \texttt{CASH}$ in the third line, where CASH is the money available at the beginning of the session:

$$\begin{aligned}
&\texttt{C} \to \texttt{A} : \{num\ req \mid req > 0\}.\\
&\texttt{A} \to \texttt{B} : \{num'\ req' \mid num = num' \wedge req = req'\}.\\
&\texttt{B} \to \texttt{A} : \{\mathsf{true}\}\ \mathsf{ok} : \texttt{A} \to \texttt{C} : \{\mathsf{true}\}\ \mathsf{ok} : \texttt{A} \to \texttt{C} : \{a \mid a = req \wedge\ a \leq \texttt{CASH}\},\\
&\qquad \{\mathsf{true}\}\ \mathsf{ko} : \texttt{A} \to \texttt{C} : \{\mathsf{true}\}\ \mathsf{ko} : \texttt{A} \to \texttt{C} : \{msg \mid \mathsf{true}\}
\end{aligned}$$

Although this solves the flaw, a temporal satisfiability issue is introduced in the third line. In fact, A cannot guarantee its obligation if the amount requested req in the first interaction is greater than the cash available.

Fortunately, Λ is applicable and it can amend the global assertion automatically by returning the choreography below

$$\begin{aligned}
&\texttt{C} \to \texttt{A} : \{num\ req \mid req > 0 \wedge \exists a\,.\, a = req \wedge\ a \leq \texttt{CASH}\}.\\
&\texttt{A} \to \texttt{B} : \{num'\ req' \mid num = num' \wedge req = req'\}.\\
&\texttt{B} \to \texttt{A} : \{\mathsf{true}\}\ \mathsf{ok} : \texttt{A} \to \texttt{C} : \{\mathsf{true}\}\ \mathsf{ok} : \texttt{A} \to \texttt{C} : \{a \mid a = req \wedge\ a \leq \texttt{CASH}\},\\
&\qquad \{\mathsf{true}\}\ \mathsf{ko} : \texttt{A} \to \texttt{C} : \{\mathsf{true}\}\ \mathsf{ko} : \texttt{A} \to \texttt{C} : \{msg \mid \mathsf{true}\}
\end{aligned}$$

which is well-asserted.

6.2 Credit Request

We now want to model a service through which a customer can request a small line of credit. The intuition of the protocol is illustrated in Figure 2. The customer C sends his/her account number num and the requested credit a to the ATM. The ATM forwards the request to the bank and, depending whether or not C is elegible for the credit according to the bank's records (i.e., $\texttt{elegible}(num, a)$), the bank selects either branch ok or ko. Finally, the ATM sends a message to the customer notifying his/her of the decision.

Remark 9. *For simplicity, we use* branch mergeability *[13], a slight extension of multiparty session types. Otherwise, it would be necessary to add an extra branch in the inner branching between* A *and* C *to have the same*

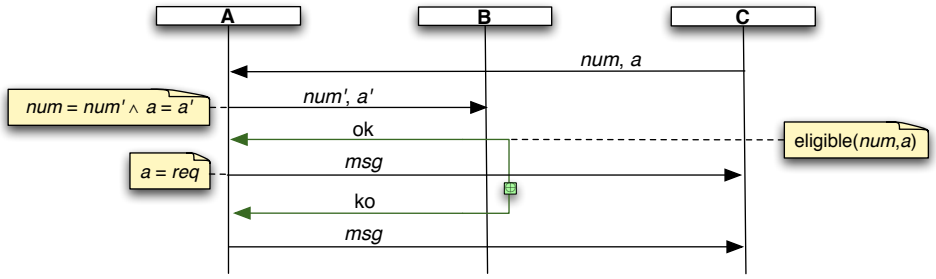


Figure 2: Credit request protocol

behaviour of C *in both branches of the outer branching. Note that this does not affect the applicability of our methodology.*

A naive global assertion modelling this service is as follows:

$$\begin{array}{l}
\mathtt{C} \to \mathtt{A} : \{num\ a \mid \mathsf{true}\}. \\
\mathtt{A} \to \mathtt{B} : \{num'\ a' \mid num = num' \wedge a = a'\}. \\
\mathtt{B} \to \mathtt{A} : \{\mathtt{elegible}(num, a)\}\ \mathsf{ok} : \mathtt{A} \to \mathtt{C} : \{msg \mid \mathsf{true}\}, \\
\qquad\qquad \{\mathsf{true}\} \qquad\qquad\quad \mathsf{ko} : \mathtt{A} \to \mathtt{C} : \{msg \mid \mathsf{true}\}
\end{array}$$

The attentive reader will notice that there is an HS problem at line 4 of this global assertion. Indeed, B does not know num' nor a' and therefore could not guarantee that the customer is in fact elegible. Both Σ and Π algorithms are applicable here. The second algorithm would return the following global assertion:

$$\begin{array}{l}
\mathtt{C} \to \mathtt{A} : \{num\ a \mid \mathsf{true}\}. \\
\mathtt{A} \to \mathtt{B} : \{num'\ a'\ v_1\ v_2 \mid num = num' \wedge a = a' \wedge v_1 = num \wedge v_2 = a\}. \\
\mathtt{B} \to \mathtt{A} : \{\mathtt{elegible}(v_1, v_2)\}\ \mathsf{ok} : \mathtt{A} \to \mathtt{C} : \{msg \mid \mathsf{true}\}, \\
\qquad\qquad \{\mathsf{true}\} \qquad\qquad\quad \mathsf{ko} : \mathtt{A} \to \mathtt{C} : \{msg \mid \mathsf{true}\}
\end{array}$$

Although this solves the problem, we notice that this solution is not ideal. Indeed v_1 and v_2 are somewhat redundant with num' and a', respectively.

Strengthening gives us a better solution in this case:

$$\begin{array}{l}
\mathtt{C} \to \mathtt{A} : \{num\ a \mid \mathsf{true}\}. \\
\mathtt{A} \to \mathtt{B} : \{num'\ a' \mid num = num' \wedge a = a'\}. \\
\mathtt{B} \to \mathtt{A} : \{\mathtt{elegible}(num', a')\}\ \mathsf{ok} : \mathtt{A} \to \mathtt{C} : \{msg \mid \mathsf{true}\}, \\
\qquad\qquad \{\mathsf{true}\} \qquad\qquad\qquad\quad \mathsf{ko} : \mathtt{A} \to \mathtt{C} : \{msg \mid \mathsf{true}\}
\end{array}$$

Which is what one would expect. Note that the algorithm is applicable because

$$num = num' \wedge a = a' \wedge \mathtt{elegible}(num', a') \supset \mathtt{elegible}(num, a)$$

holds and B knows num' and a'.

7 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 (Σ, Π, and Λ) 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 Σ, Π, and Λ 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) expressed in formalisms such as CC-Pi [5], a language for distributed processes with constraints. Interestingly, temporal satisfiability is similar to the feasibility property in [2] 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". An interesting future development is to investigate more general accounts of satisfiability which is applicable to different scenarios. In scope of future work, we will design *amendment strategies* to so to refine our methodology and maximise the applicability of the proposed algorithms (see Section 5.1).

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 [9].

References

[1] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *Software Concepts and Tools*, pages 304–313, 2003.

[2] K. R. Apt, N. Francez, and S. Katz. Appraising fairness in languages for distributed programming. *Distributed Computing*, 2:226–241, 1988.

[3] L. Bocchi, K. Honda, E. Tuosto, and N. Yoshida. A theory of design-by-contract for distributed multiparty interactions. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 2010.

[4] L. Bocchi, J. Lange, and E. Tuosto. Amending contracts for choreographies. In A. Silva, S. Bliudze, R. Bruni, and M. Carbone, editors, *ICE*, volume 59 of *EPTCS*, pages 111–129, 2011.

[5] Maria Grazia Buscemi and Ugo Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2007.

[6] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2007.

[7] Tzu-Chun Chen, Laura Bocchi, Pierre-Malo Deniélou, Kohei Honda, and Nobuko Yoshida. Asynchronous distributed monitoring for multiparty session enforcement. In Roberto Bruni and Vladimiro Sassone, editors, *TGC'11*, LNCS. Springer, 2011. To appear.

[8] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *POPL*, pages 273–284. ACM, 2008.

[9] Julien Lange and Emilio Tuosto. A modular toolkit for distributed interactions. In Kohei Honda and Alan Mycroft, editors, *PLACES*, volume 69 of *EPTCS*, pages 92–110, 2010.

[10] Sjouke Mauw. The formalization of message sequence charts. *Computer Networks and ISDN Systems*, 28(12):1643–1657, 1996.

[11] B. Meyer. *Object-Oriented Software Construction (Chapter 31)*. Prentice Hall, 1997.

[12] Sebastián Uchitel, Jeff Kramer, and Jeff Magee. Detecting implied scenarios in message sequence chart specifications. In *ESEC / SIGSOFT FSE*, pages 74–82, 2001.

[13] Nobuko Yoshida, Pierre-Malo Deniélou, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. In C.-H. Luke Ong, editor, *FOSSACS*, volume 6014 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2010.