# A Symbolic Framework for the Conformance Checking of Value-Passing Choreographies[⋆]

Huu Nghia Nguyen[1], Pascal Poizat[1,2], Fatiha Zaïdi[1]

[1] LRI; Univ. Paris-Sud, CNRS, Orsay, France
[2] Univ. Évry Val d'Essonne, Evry, France
{huu-nghia.nguyen, pascal.poizat, fatiha.zaidi}@lri.fr

**Abstract.** Choreographies, thanks to their abstract and global perspective, are well-suited to the specification of distributed systems such as service compositions and collaborative business processes. Choreography conformance checking aims at verifying whether a set of distributed peers or local role specifications match a global specification. This activity is central in both top-down and bottom-up development processes for distributed systems. Such systems usually collaborate through information exchange, thus requiring value-passing choreography languages and models. However, most of the conformance checking techniques abstract value-passing or bound the domains for the exchanged data. As an alternative, we propose a conformance checking framework based on symbolic models and an extension of the symbolic bisimulation equivalence. This enables one to take into account value passing while avoiding state space explosion issues. Our framework is fully tool supported.

**Keywords:** choreography, specification, conformance, symbolic transition systems, symbolic branching bisimulation, tools.

## 1 Introduction

**Context.** *Choreography* is the description with a global perspective of *interactions* between *roles* played by *peers* (services, organizations, humans) in some collaboration. A choreography has two components: a set of roles, and the specification of the legal orderings of message exchanges between these roles. Choreography languages may be classified wrt. their underlying interaction model [1, 2]: *interconnected interface models* (*e.g.*, BPMN 1.0 Collaboration Diagrams) where peer models are defined first, before being connected, and *interaction models* (*e.g.*, BPMN 2.0 Choreography Diagrams) where interaction between peers is the specification atom. The later better suit the needs of choreography specification due to their perspective abstracting away from peer implementation.

[⋆] Version 1 – July 30th, 2012. Part of this works has been published in the proceedings of ICSOC'2012. This work is supported by the PIMI project (ANR-2010-VERS-0014-03) of the French National Agency for Research.

**Issues.** One key issue in choreography-based development is checking the *conformance* of a set of local descriptions wrt. the choreography global specification. This issue naturally arises in a bottom-up development process, where peers implemented as services and advertising behavioural descriptions (conversations) are reused and composed. The question here is *"Will these peers behave altogether as prescribed in the choreography?"*. Conformance is also central in a top-down development process, where local obligations, kinds of "behavioural skeletons", are generated by projection from the choreography. The question here is *"Am I sure that if I implement these local obligations then I will have a correct implementation of the choreography?"*. Conformance is therefore a cornerstone for choreography *realizability checking* that addresses whether projection may or may not be used to get a conform set of peer skeletons. Even simple choreographies such as *"A and B must interact first before C and D may interact"* can be unrealizable as-is and require additional exchanges in the implementation to enforce the ordering of messages (here, an additional message sent from B to C to prevent C to interact with D before A and B have done so). Therefore, the definition of conformance should not be too strict. It should support *choreography refinement*, *e.g.*, with peers and interactions being added in the implementation by the service architect in order to enforce the specification. Finally, entities in a distributed system usually exchange information, *i.e.*, data, while interacting. As a consequence, *data should be supported* in choreography specifications, in the descriptions of the local entities, and in the conformance relation.

**Related Work.** Testing may be used to check for the conformance of an implementation wrt. a choreography specification [3, 4]. However, it is generally desirable to detect faults as soon as possible in system development. Further, verification enables one to check the conformance of systems-to-be, *e.g.*, in a top-down development process where local obligations are not yet implemented.

In Table 1, we compare approaches for conformance checking. Columns 2 and 3 focus on data support. Some approaches just *abstract away from data*. This is known to yield over-approximation issues, *e.g.*, false negatives in the verification process. For the following, let us suppose a very simple choreography $C$: *"A and B exchange an integer lower than 5"*. Data can be supported by working on *closed implementation-level systems* where sent messages contain only ground data, *e.g.*, *"A sends 4 to B"* or *"A sends x+3 to B, knowing that x is 1"*. In such a

**Table 1.** Choreography conformance approaches

| | Data & Value-Passing | | Expressiveness | | Conformance | |
|---|---|---|---|---|---|---|
| | supported | treatment | loops | assignment | global | relation (based on) |
| [5] | no | - | yes | no | yes | Trace equivalence |
| [6] | | | yes | no | yes | Weak bisimulation |
| [7] | | | yes | no | yes | Strong bisimulation |
| [8] | yes | closure | yes | yes | no | Weak bisimulation |
| [9] | | closure | no | yes | yes | Branching bisimulation |
| [10] | | bound data | yes | no | yes | Branching bisimulation |
| this paper | | symbolic | yes | no | yes | Branching bisimulation |

case, the state space explosion of the system model is limited. This is because even if the reception of a message in some entity is denoted with a free variable, *e.g.*, *"B receives from A a y lower than 10"*, upon making it correspond with a sent message, the variable will be bound. Here, $y = 4$, satisfying $B$ since $4 < 10$ and conforming to $C$ since $4 < 5$. However, this is not adequate when working on abstract specifications where there are no such ground sent messages but only free variables and constraints on their values, *e.g.*, *"A sends some x greater than 3 to B"* (note, here, that the *exact* $x$ is not given, since it can be known only at run time). Another solution is to *bound data domains*, *e.g.*, integers are bound to $[0..b]$. The issue is that conformance may not yield outside the bounds. On our example, it works if $b < 5$ but not if $b \geq 5$ since $A$ may then choose to send 6. Defining bounds in order to avoid false positives in the verification process can be difficult. In our framework, data is supported using a *symbolic approach and the use of an SMT solver.* Conformance may be checked for whole data domains. If it does not yield, we can determine conditions for it to do. Here, $x < 5$. This is particularly relevant in a top-down development process or for online verification, respectively to add conditional statements in peer skeletons generated from choreography specifications, or to detect as soon as possible that an interaction will eventually lead to a conformance error.

Columns 4 and 5 are relative to the *expressiveness* of the choreography language. Loops and assignments may yield state space explosion in presence of data if one does not close the system or bound data domains. We chose to support loops since we believe that assignments are a less important feature in a specification language: a specification should express *what* should be done (*"A sends an x to B and then B replies with a y greater than x"*) rather than *how* it should be done (*"A sends an x to B, B adds 1 to x and sends it back to A"*).

The last two columns are relative to the kind of conformance being supported and the behavioural equivalence being used. *Global conformance* is important in conformance checking since one wants not only to know if each peer is conform to its role, *i.e.*, *local conformance*, but also if the peers altogether have a behaviour that is conform to the choreography. Local conformance does not implies global conformance. Weak and branching bisimulations are able to support internal actions and hiding (formally, $\tau$ actions). This is important, *e.g.*, if one has to deal with messages added to make some choreography realizable. Branching bisimulation [11] has been preferred over weak bisimulation in the last years since it is a congruence, hence supports compositional reasoning.

Symbolic bisimulations, defined on Symbolic Transition Graphs (STGs), have been introduced in [12] with both of early and late semantics. In this work, we use late semantics. The STGs have then been extended to STGs with assignments (STGAs) in [13, 14]. These works mostly concentrate on strong and weak bisimulation. Symbolic branching bisimulation has not yet received much attention. As a consequence, there is tool support for symbolic strong bisimulation [15] and open bisimulation[3], but not for symbolic branching bisimulation.

[3] http://sbriais.free.fr/tools/abc/

**Contributions.** Our contributions are the following. Based on process algebras for choreography [16, 5], we propose *a specification and description language* with an *interaction model* addressing both the *global* (choreography) and the *local* (peers description, role requirements) perspective over distributed systems. Our language supports *information exchange and data-related constructs* (conditional and loop constructs). We give a *fully symbolic semantics* to this language using a model transformation into STGs, thus avoiding data abstraction and over-approximation, restriction to manually bound data domains, and limitation to implementation-level closed descriptions. Accordingly, we build on branching bisimulation [11] and on a symbolic extension of weak bisimulation [14] to develop a specific *symbolic version of branching bisimulation* dedicated at *checking the conformance* of a set of local entities wrt. a choreography specification. Our equivalence enables one to check conformance in presence of choreography *refinement*, *i.e.*, where new peers and/or interactions may be added wrt. the specification. Going further than a *true* vs. *false* result for conformance, our approach supports the generation of the *most general constraint over exchanged information* in order to have conformance. Finally, our framework is *fully automated with a tool-chain* that is freely available online (please see Section 4 for the URL).

**Overview.** The remainder of the paper is structured as follows. We present our language, the formal models we use, and our model transformation in Section 2. We then introduce our conformance relation in Section 3. The implementation of our framework and experiments are discussed in Section 4. Finally, we conclude and discuss perspectives of our work.

## 2 A Language for Choreographies, Roles, and Peers

In this section, we present our specification and description language. It can be used to specify distributed systems with a global perspective, *i.e.*, *choreographies*, to define local requirements, *i.e.*, *roles*, and to describe the pieces of a distributed implementation, *i.e.*, *peers*. Due to this multi-purpose objective, it is first presented in terms of an abstract alphabet, $A$. We then explain how $A$ can be realized for the different purposes. In a second step, we give a semantics to our language, in terms of a symbolic model, namely Symbolic Transition Graphs.

**General syntax.** The basis of a behavioural description is the notion of event. To promote generality, let us first suppose a set $A$ (alphabet) of events of interest. The syntax of our specification language, $L(A)$, is given by:

$$L ::= \mathbf{1} \;\mid\; \alpha \;\mid L; L \;\mid\; L + L \;\mid\; L|L \;\mid\; L[\!>L \;\mid\; [\phi] \triangleright L \;\mid\; [\phi] * L$$

In $L(A)$ we can specify activities which are either basic or composite. A basic activity is either termination ($\mathbf{1}$) or a regular activity ($\alpha$, with $\alpha \in A$). We then have structuring operators, that can be used to specify composite activities: sequencing (;), non deterministic choice (+), parallel activities (|), and interruption ($[>$). Furthermore, we support data exchange (see below, *Choreography syntax*). Accordingly, we have data-based conditional constructs, namely guards ($\triangleright$) and while loops ($*$), where $\phi$ is the condition (a boolean expression).

Process algebras have successfully been used to give a formal semantics to business process languages and notations [17]. We take inspiration in all-purpose process algebras such as LOTOS and choreography-oriented ones [5, 16]. Since we are interested in an abstract formal choreography language, *i.e.*, implementation independent and with an interaction-based model [1], the usual $\tau$ actions can be ignored [18]. Moreover, since we support data, we do not require $\tau$s to model conditional constructs. While a specification such as `if x>y then P else Q` is usually encoded into $\tau; P + \tau; Q$, we can encode it into $[x > y] \triangleright P + [\neg(x > y)] \triangleright Q$.

**Choreography (global) specification.** A choreography specification describes, with a global perspective, the legal interactions among roles played by the participants of a distributed system. In a choreography, each role is identified by a unique name. The basis of an interaction-model choreography description is the interaction. Let us denote an interaction $c$ from role $a$ to role $b$ by $c^{[a,b]}.x$, where $x$ is a variable that represents the information exchanged during interaction ($x$ is omitted when there is no such information). We stress out that $x$ can be structured, *e.g.*, to denote a multiple data exchange as done in Web services with XML message types. A *choreography (or global) specification* for a set of roles $R$, a set of interactions $C$, and a set of variables $V$, is an element of $L(A)$ with $A = \{c^{[a,b]}.x \mid c \in C \wedge a \in R \wedge b \in R \wedge a \neq b \wedge x \in V\}$.

*Example 1.* Let us suppose a shipping choreography between two roles: c (client) and s (shipper). The client first requests shipping by providing the weight of goods to be sent. If this is less than 5 kgs then the goods will be sent for free. Otherwise, the shipping has to be paid. This can be described as follows:

$$\mathit{Shipping} ::= \mathbf{Request}^{[c,s]}.x_1; ([x_1 < 5] \triangleright \mathbf{FreeShip}^{[s,c]} + [x_1 \geq 5] \triangleright \mathbf{PayShip}^{[s,c]})$$

**Role and Peer (local) descriptions.** In a top-down development process, local descriptions correspond to role requirements (or role for short) obtained by projection from global specifications. Each role describes what is expected from a peer that would implement it. In a bottom-up development process, local specifications correspond to the behavioural contracts or interfaces that peers advertise in order to foster reuse, composition, and adaptation [17]. In the sequel, we will use the term *local entity*, or *entity* for short, in order to abstract from the process development being used. Interactions specified in choreographies correspond to communication primitives in entities. In an entity $a$, these primitives can be abstracted as sending and reception events, denoted respectively with $c^{[a,b]}!x$ and $c^{[b,a]}?x$, where $b$ is another entity, *i.e.*, $a \neq b$, and $x$ is the information exchanged during interaction ($x$ is omitted when there is no such information). An *entity (or local) description* for an entity $a$, a set of roles $R$ with $a \in R$, a set of interactions $C$, and a set of variables $V$, is an element of $L(A)$ with $A = \{c^{[a,b]}!x,\ c^{[b,a]}?x \mid c \in C \wedge b \in R \wedge a \neq b \wedge x \in V\}$.

*Example 2.* A tentative to implement the shipping choreography (which is correct as we will see in the following) is that the client sends the weight to the shipper and then waits for either free or paid shipping, while it is the shipper that checks the weight in order to decide which shipping is used:

$$\begin{aligned}\textit{Client } c &::= \mathbf{Request}^{[c,s]}!y_1; (\mathbf{FreeShip}^{[s,c]}? + \mathbf{PayShip}^{[s,c]}?)\\ \textit{Shipper } s &::= \mathbf{Request}^{[c,s]}?z_1; ([z_1 < 5] \triangleright \mathbf{FreeShip}^{[s,c]}! + [z_1 \geq 5] \triangleright \mathbf{PayShip}^{[s,c]}!)\end{aligned}$$

**Symbolic Transition Graphs.** A Symbolic Transition Graph [12] is a transition system where a set of variables, possibly empty, is associated to each state and where each transition may be guarded by a boolean expression $\phi$ that determines if the transition can be fired or not. Actions labeling transitions will correspond in our work to the elements of the alphabets we have seen earlier on. We also add a specific event, $\checkmark$, to denote activity termination. The set of free and bound variables of these actions are defined as following: $bv(c^{[-,-]}.x) = bv(c^{[-,-]}?x) = \{x\}$, $fv(c^{[-,-]}!x) = \{x\}$, and otherwise both $fv(\alpha)$ and $bv(\alpha)$ are empty. We also use $fv(e)$ to denote the set of free variables in expression $e$.

**Definition 1 (Symbolic Transition Graph).** *A Symbolic Transition Graph (STG) is a tuple $(S, s_0, T)$ where $S$ is a non empty set of states, each state having an associated set of free variables $fv(s)$, $s_0 \in S$ is the initial state, and $T$ is a set of transitions of the form $s \xrightarrow{[\phi]\ \alpha} s'$ with $s, s' \in S$ and $fv(\phi) \cup fv(\alpha) \subseteq fv(s)$ and $fv(s') \subseteq fv(s) \cup bv(\alpha)$.*

A non-symbolic semantics for STGs would bound free variables using domain enumeration, *e.g.*, bound an integer $x$ to $\{0, 1, 2, \ldots\}$, thus yielding state space explosion. In the symbolic semantics, substitutions are associated to STG states. Let us begin with some notations. $V$ is a set of variables, ranged over by $x, y, z, x_1$, etc. A variable substitution (substitution for short) $\sigma$ is a mapping from $V$ to $V$. $\emptyset$ denotes the empty substitution. $\sigma[x \mapsto y]$ is the substitution $\sigma$ where the mapping from $x$ to $y$ is added, eventually erasing a previous mapping from $x$. $e\sigma$ denotes the application of $\sigma$ to $e$. A *term* [12] is a pair $s_\sigma$ where $s$ is a state and $\sigma$ is a substitution. In the sequel, we denote states by $s$, $s_1$, $s'$, etc. and terms by $t$, $t_1$, $t'$, etc. We may now define the (late) symbolic semantics of an STG as relation over terms (Fig. 1). Substitutions apply to free variables in the guard and in the sent information. In case of a reception (rule RECEIVE) or an interaction (rule INTERACT), a variable, $x$, becomes bound. To denote this symbolically (without enumeration), the substitution in the target state is updated using a fresh variable, $z$, instead of a value. Please note that $\rightarrow$ is used for STG transitions (over states) and $\mapsto$ is used for STG semantics (over terms).

**Model transformation to STGs.** We use STGs as a formal model to give semantics to our language. This is achieved by a rule-based transformation defined in Figure 2 where $\mathbf{0}$ is the empty description (only used for semantics), $\checkmark$ denotes activity termination, $\alpha$ denotes any event but for $\checkmark$, and $\hat{\alpha}$ denotes any event (including $\checkmark$). The symmetric rules for CHOICE$_1$ and PAR$_1$ can be inferred from them and are omitted here.

$$(\mathit{TERMINATE})\ \frac{s \xrightarrow{[\phi]\ \checkmark} s'}{s_\sigma \vdash\!\xrightarrow{[\phi\sigma]\ \checkmark} s'_\sigma} \qquad (\mathit{SEND})\ \frac{s \xrightarrow{[\phi]\ c^{[-,-]}!x} s'}{s_\sigma \vdash\!\xrightarrow{[\phi\sigma]\ c^{[-,-]}!x\sigma} s'_\sigma} \qquad (\mathit{RECEIVE})\ \frac{s \xrightarrow{[\phi]\ c^{[-,-]}?x} s'}{s_\sigma \vdash\!\xrightarrow{[\phi\sigma]\ c^{[-,-]}?z} s'_{\sigma[x\mapsto z]}} \qquad (\mathit{INTERACT})\ \frac{s \xrightarrow{[\phi]\ c^{[-,-]}.x} s'}{s_\sigma \vdash\!\xrightarrow{[\phi\sigma]\ c^{[-,-]}.z} s'_{\sigma[x\mapsto z]}}$$

**Fig. 1.** Symbolic semantics of an STG

**Product of STGs.** The product of STGs is used to give a semantics to a set of interacting local entities (Fig. 3). We assume that the STGs use disjoint sets of variables which can be achieved using, *e.g.*, indexing by the name of the entity. The first rule (TERM) expresses that the composition terminates successfully when all its components do so. The second rule (PAR$_1$) represents the independent evolution of one of the STGs (the first one). Finally, the third rule (COM$_1$) denotes a synchronization between $a$ sending a data in $x$ to $b$ and $b$ receiving it in variable $y$. Normally, the variable $y$ of $b$ would be fixed by $x$ however we intend to evaluate $y$ with all its possible valuation. In this rule, $t[x \mapsto y]$ with $t = s_\sigma$ denotes the term $s_{\sigma[x\mapsto y]}$. Further, $\langle t_1, t_2\rangle$, with $t_1 = s_{1\sigma_1}$ and $t_2 = s_{2\sigma_2}$, denotes the term $(s_1, s_2)_{\sigma_1\cup\sigma_2}$. The symmetric rules for PAR$_1$ and COM$_1$ can be inferred from them and are omitted here. We give a binary version of the product for simplicity purposes. An n-ary version of it can be obtained working on tuples $\langle t_1, \ldots, t_n\rangle$ of terms instead of pairs $\langle t_1, t_2\rangle$, generalizing TERM to n terminations, and having n symmetric rules PAR$_1$, ..., PAR$_n$. COM does not change being generic on $a$ and $b$. Our algorithm for the product of STGs can be found in section 4.

*Example 3.* The STGs for the choreography in Example 1 and for the client and shipper in Example 2 are shown in Figure 4(a-c). Figure 4(d) presents the

$$(\mathit{SKIP})\ \frac{}{\mathbf{1} \xrightarrow{[true]\ \checkmark} \mathbf{0}} \qquad (\mathit{ACT})\ \frac{}{\alpha \xrightarrow{[true]\ \alpha} \mathbf{1}} \qquad (\mathit{SEQ}_1 - \text{L}_1 \text{ does not end})\ \frac{L_1 \xrightarrow{[\phi]\ \alpha} L_1'}{L_1;L_2 \xrightarrow{[\phi]\ \alpha} L_1';L_2} \qquad (\mathit{SEQ}_2 - \text{L}_1 \text{ ends, begin L}_2)\ \frac{L_1 \xrightarrow{[\phi_1]\ \checkmark} L_1',\ L_2 \xrightarrow{[\phi_2]\ \hat\alpha} L_2'}{L_1;L_2 \xrightarrow{[\phi_1\wedge\phi_2]\ \hat\alpha} L_2'}$$

$$(\mathit{CHOICE}_1 - \text{choose L}_1)\ \frac{L_1 \xrightarrow{[\phi]\ \hat\alpha} L_1'}{L_1+L_2 \xrightarrow{[\phi]\ \hat\alpha} L_1'} \qquad (\mathit{PAR}_1 - \text{one step in L}_1)\ \frac{L_1 \xrightarrow{[\phi]\ \alpha} L_1'}{L_1|L_2 \xrightarrow{[\phi]\ \alpha} L_1'|L_2} \qquad (\mathit{PAR}_3 - \text{synchronous termination})\ \frac{L_1 \xrightarrow{[\phi_1]\ \checkmark} L_1',\ L_2 \xrightarrow{[\phi_2]\ \checkmark} L_2'}{L_1|L_2 \xrightarrow{[\phi_1\wedge\phi_2]\ \checkmark} L_1'|L_2'}$$

$$(\mathit{GUARD})\ \frac{L \xrightarrow{[\phi']\ \hat\alpha} L'}{[\phi]\triangleright L \xrightarrow{[\phi\wedge\phi']\ \hat\alpha} L'} \qquad (\mathit{LOOP}_1 - \text{one more iteration})\ \frac{L \xrightarrow{[\phi']\ \hat\alpha} L'}{[\phi]*L \xrightarrow{[\phi\wedge\phi']\ \hat\alpha} L';[\phi]*L} \qquad (\mathit{LOOP}_2 - \text{end of the loop})\ \frac{}{[\phi]*L \xrightarrow{[\neg\phi]\ \checkmark} \mathbf{0}}$$

$$(\mathit{INTER}_1 - \text{one L}_1 \text{ step})\ \frac{L_1 \xrightarrow{[\phi]\ \alpha} L_1'}{L_1[\!>L_2 \xrightarrow{[\phi]\ \alpha} L_1'[\!>L_2} \qquad (\mathit{INTER}_2 - \text{L}_1 \text{ ends, interruption not possible})\ \frac{L_1 \xrightarrow{[\phi]\ \checkmark} L_1'}{L_1[\!>L_2 \xrightarrow{[\phi]\ \checkmark} L_1'} \qquad (\mathit{INTER}_3 - \text{interruption by L}_2)\ \frac{L_2 \xrightarrow{[\phi]\ \hat\alpha} L_2'}{L_1[\!>L_2 \xrightarrow{[\phi]\ \hat\alpha} L_2'}$$

**Fig. 2.** Transformation from our language to STGs

(TERM)
$$\frac{t_1 \xmapsto{[\phi_1]\ \checkmark} t_1',\ t_2 \xmapsto{[\phi_2]\ \checkmark} t_2'}{\langle t_1, t_2\rangle \xmapsto{[\phi_1 \wedge \phi_2]\ \checkmark} \langle t_1', t_2'\rangle}$$

(PAR$_1$)
$$\frac{t_1 \xmapsto{[\phi]\ c.x} t_1'}{\langle t_1, t_2\rangle \xmapsto{[\phi]\ c.x} \langle t_1', t_2\rangle}$$

(COM$_1$)
$$\frac{t_1 \xmapsto{[\phi_1]\ c^{[a,b]}!x} t_1',\ t_2 \xmapsto{[\phi_2]\ c^{[a,b]}?y} t_2'}{\langle t_1, t_2\rangle \xmapsto{[\phi_1 \wedge \phi_2]\ c^{[a,b]}.y} \langle t'_{1[x\mapsto y]}, t_2'\rangle}$$

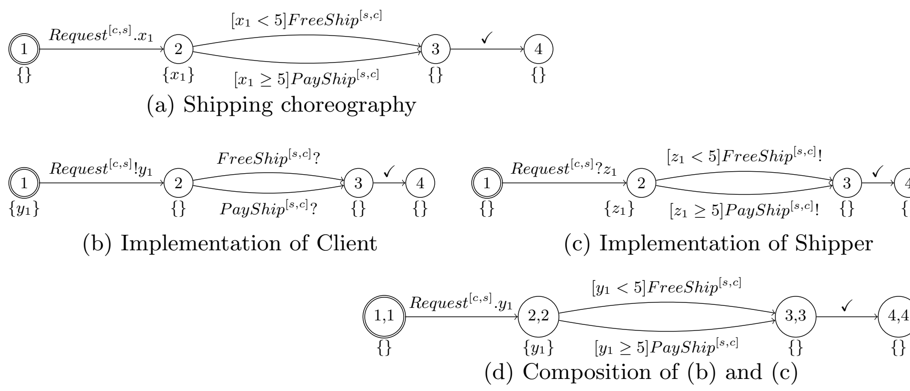
**Fig. 3.** Rules for the product of STGs

product of the STGs in Figure 4(b) and Figure 4(c). The free variables of the states are given below them, *e.g.*, $\{x_1\}$ for state 2 in the choreography STG.

## 3 Choreography Conformance

In this section, we present our conformance relation between a model denoting the semantics of a set of local descriptions for interacting entities, $\mathcal{I}$, and a choreography global specification, $\mathcal{C}$. In the sequel, $\mathcal{I}$ will be named *implementation* even if we have seen before that it may denote either a real implementation or a set of local requirements to be implemented. Since the semantic models are STGs, we define conformance over two STGs, one for $\mathcal{I}$ and one for $\mathcal{C}$. We choose branching bisimulation [11] as a basis since it supports equivalence in presence of $\tau$ actions that result from the hiding of interactions added in implementations wrt. specifications, *i.e.*, refinement. However, branching bisimulation is defined over ground terms (no variables), while STGs may contain free variables.

In [9, 10], this issue is considered by introducing at each state an evaluation function that maps variables to values, thus reducing open terms to ground ones. Conformance is then verified only for some fixed values of the model variables. It is possible to check conformance for different fixed values. However, this may lead to state space explosion when domains of the variables are very big. As



$Request^{[c,s]}.x_1$
$[x_1 < 5]FreeShip^{[s,c]}$
$[x_1 \geq 5]PayShip^{[s,c]}$
$\checkmark$
$\{\}$
$\{x_1\}$
(a) Shipping choreography
$Request^{[c,s]}!y_1$
$FreeShip^{[s,c]}?$
$PayShip^{[s,c]}?$
$\{y_1\}$
(b) Implementation of Client
$Request^{[c,s]}?z_1$
$[z_1 < 5]FreeShip^{[s,c]}!$
$[z_1 \geq 5]PayShip^{[s,c]}!$
$\{z_1\}$
(c) Implementation of Shipper
$Request^{[c,s]}.y_1$
$[y_1 < 5]FreeShip^{[s,c]}$
$[y_1 \geq 5]PayShip^{[s,c]}$
$\{y_1\}$
(d) Composition of (b) and (c)

**Fig. 4.** STGs for Ex. 1 and Ex. 2

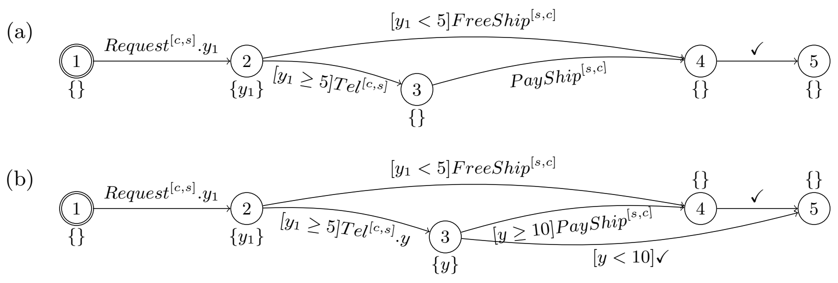

(a)
$Request^{[c,s]}.y_1$
$[y_1 < 5]FreeShip^{[s,c]}$
$[y_1 \geq 5]Tel^{[c,s]}$
$PayShip^{[s,c]}$
✓
{}
$\{y_1\}$
(b)
$[y_1 \geq 5]Tel^{[c,s]}.y$
$\{y\}$
$[y \geq 10]PayShip^{[s,c]}$
$[y < 10]$✓

**Fig. 5.** Two refinements for the shipping implementation in Fig. 4(d)

an alternative, we base our work on (late) symbolic extensions of bisimulations, introduced in [12–14], that directly support open terms.

**Making implementation and specification comparable.** First of all, we remind the reader that we assume the two STGs have disjoint sets of variables which can be achieved using, *e.g.*, indexing. We also assume for simplicity that a local entity has the same identifier than the corresponding role in the choreography. This constraint could be lifted using a mapping function. Additional interactions may have been introduced in the implementation wrt. the specification during refinement, *e.g.*, to make it realizable. In order to compare the STGs, we have first to hide these interactions, which is done using restriction (Def. 2).

**Definition 2 (Restriction).** *Given an STG $\mathcal{S} = (S, s_0, T)$ and a finite set of actions $\mathcal{A}$, the restriction of $\mathcal{S}$ to $\mathcal{A}$, denoted by $\mathcal{S} \downarrow_{\mathcal{A}}$, is the STG, $\mathcal{S}'$, obtained from $\mathcal{S}$ by renaming into $\tau$ all the actions that do not exist in $\mathcal{A}$ and updating the set of free variables of each state in $\mathcal{S}'$ in order to satisfy Definition 1.*

*Example 4.* We give refinement examples in Figure 5. In the first case (a), we have an additional interaction to agree on the phone (there is no choice) that non-free shipping will be used. In the second case (b), the client can specify the maximum (s)he agrees to pay for the shipping (y). This influences the sequel of the

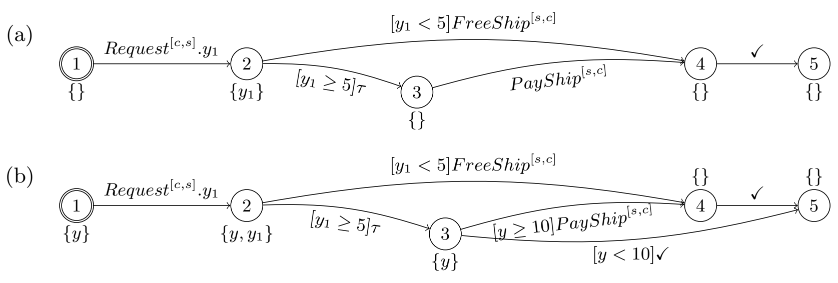


**Fig. 6.** Restrictions of the STGs in Fig. 5

implementation since the non-free shipping costs \$10: if the user requires to pay less, no shipping is done. The restriction of STGs in Figure 5 to the set of actions used in the choreography specification, $\{Request^{[c,s]}, FreeShip^{[s,c]}, PayShip^{[s,c]}\}$, yields the STGs in Figure 6.

**Conformance relation.** Thanks to the previous steps (product of the local entity STGs and STG restriction), in the sequel we may consider STGs with interaction, termination, and hidden actions only, *i.e.*, $\hat{\alpha} \in \{c^{[a,b]}.x, \checkmark, \tau\}$. Further, thanks to the use for choreography conformance (Def. 5, below), we know that the first of the two STG we compare may have $\tau$ actions (resulting from restriction) while the second STG (the choreography specification) may not.

A key element is to be able to "absorb" series of $\tau$s. The idea originates from weak forms of bisimulations such as weak bisimulation [19] and branching bisimulation [11] where we have a transition $s \overset{\tau}{\Rightarrow} s'$ if we have a path of zero or more $\xrightarrow{\tau}$ transitions between $s$ and $s'$. However, to work with STGs, these $s \overset{\tau}{\Rightarrow} s'$ transitions have to be extended to support the guards that may appear on $\xrightarrow{\tau}$ transitions. This is achieved using rules EMPTY and TAU$^*$ in Figure 7. Rule SEM in this figure defines the semantics of $\overset{\tau}{\Rightarrow}$ transitions. This semantics is denoted with relations $\overset{\tau}{\Longmapsto}$ over terms in the same way than we had $\rightarrow$ transitions in STG and $\mapsto$ in their semantics. These rules are a simplification of the ones in [14] that support $s \overset{\hat{\alpha}}{\Rightarrow} s'$ transitions where $\hat{\alpha}$ is not always $\tau$ (this difference being a consequence of the definition of weak bisimulation wrt. branching bisimulation).

In presence of variables and guards, the semantics of transition firing in STGs can be supported by domain enumeration as we have seen before. To avoid this (and the risk of state space explosion), we give a symbolic semantics to firing too, associating to transitions the condition under which it can be fired (Def. 3).

**Definition 3.** *A transition* $t \overset{[\phi]\ \hat{\alpha}}{\longmapsto} t'$ *(resp.* $t \overset{[\phi]\ \tau}{\Longmapsto} t'$*) is fireable under condition* $\rho$*,* $fv(\rho) \subseteq fv(\phi)$*, iff* $\rho \Rightarrow \phi$*. In such a case, we write* $\rho \models t \overset{[\phi]\ \hat{\alpha}}{\longmapsto} t'$ *(resp.* $\rho \models t \overset{[\phi]\ \tau}{\Longmapsto} t'$*). By extension, we write* $\rho \models t \overset{[\phi]\ \tau}{\Longmapsto} t' \overset{[\phi']\ \hat{\alpha}}{\longmapsto} t''$ *if we have* $\rho \models t \overset{[\phi]\ \tau}{\Longmapsto} t'$ *and* $\rho \models t' \overset{[\phi']\ \hat{\alpha}}{\longmapsto} t''$*.*

Our conformance relation (Def. 4) is inspired by the extension of weak bisimulation into branching bisimulation [11] and the extension of weak bisimulation into symbolic weak bisimulation [14]. We take into account termination ($\checkmark$) and the fact that there may be $\tau$s only in the first of the two compared STGs.

$$\text{(EMPTY)}\ \frac{}{s \xRightarrow{[true]\ \tau} s} \qquad \text{(TAU}^*\text{)}\ \frac{s \xRightarrow{[\phi_1]\ \tau} s_1, s_1 \xrightarrow{[\phi_2]\ \tau} s'}{s \xRightarrow{[\phi_1 \wedge \phi_2]\ \tau} s'} \qquad \text{(SEMANTICS)}\ \frac{s \xRightarrow{[\phi]\ \tau} s'}{s_\sigma \overset{[\phi\sigma]\ \tau}{\Longmapsto} s'_\sigma}$$

**Fig. 7.** Retrieval of $\overset{\tau}{\Rightarrow}$ transitions and their semantics

**Definition 4 (Symbolic Branching Bisimulation for Conformance).** *A binary relation over terms, $\mathcal{R}^\rho$, parametrized by a boolean formula $\rho$, is a symbolic branching bisimulation for conformance (SBBC) iff $(t_1, t_2) \in \mathcal{R}^\rho$ implies:*

1. $\forall\ (\rho \models t_1 \xmapsto{[\phi_1]\ \tau} t_1')\quad (t_1', t_2) \in \mathcal{R}^\rho$
2. $\forall\ (\rho \models t_1 \xmapsto{[\phi_1]\ \checkmark} t_1')\quad \exists\ (\rho \models t_2 \xmapsto{[\phi_2]\ \checkmark} t_2')$
3. $\forall\ (\rho \models t_1 \xmapsto{[\phi_1]\ c.x_1} t_1')\quad \exists\ (\rho \models t_2 \xmapsto{[\phi_2]\ c.x_2} t_2')\quad ({t'_1}_{[x_1 \mapsto z]}, {t'_2}_{[x_2 \mapsto z]}) \in \mathcal{R}^{\rho'}$ *with* $\rho' = \rho[x_1 \mapsto z, x_2 \mapsto z]$ *and* $z$ *is a fresh variable*
4. $\forall\ (\rho \models t_2 \xmapsto{[\phi_2]\ \checkmark} t_2')\quad \exists\ (\rho \models t_1 \overset{[\phi_1]\ \tau}{\Longmapsto} t_1' \xmapsto{[\phi_1']\ \checkmark} t_1'')\quad (t_1', t_2) \in \mathcal{R}^\rho$
5. $\forall\ (\rho \models t_2 \xmapsto{[\phi_2]\ c.x_2} t_2')\ ,\ \exists\ (\rho \models t_1 \overset{[\phi_1]\ \tau}{\Longmapsto} t_1' \xmapsto{[\phi_1']\ c.x_1} t_1'')\quad (t_1', t_2) \in \mathcal{R}^\rho \wedge ({t''_1}_{[x_1 \mapsto z]}, {t'_2}_{[x_2 \mapsto z]}) \in \mathcal{R}^{\rho'}$ *with* $\rho' = \rho[x_1 \mapsto z, x_2 \mapsto z]$*, and* $z$ *is a fresh variable*

Definition 4 gives the conditions under which two terms, $t_1$ and $t_2$, are $R^\rho$ equivalent. Case (5) states that for an interaction fireable under condition $\rho$ from $t_2$ to $t_2'$, there must be an equivalent interaction fireable under condition $\rho$ from $t_1$ to $t_1''$, possibly after zero or more $\tau$ transitions between $t_1$ and some $t_1'$. Equivalence of interactions is up to renaming of the used variables ($x_1$ and $x_2$) into a fresh variable $z$. Moreover, following branching bisimulation, we must have $t_1'$ and $t_2$ (respectively $t_1''$ and $t_2'$) equivalent. In the later case, we have to take into account the renaming of $x_1$ and $x_2$ by $z$ in terms ($t_1''$ and $t_2'$) and in the equivalence relation condition ($\rho$). Case (4) is simpler. To a termination in $t_2$ corresponds a termination in $t_1$ reachable after zero or more $\tau$s. Since no data is bound by termination, we just have to take into account recurrence over $R^\rho$. Cases (2) and (3) are symmetric versions of cases (4) and (5) respectively, simpler since there are no $\tau$s in the specification/$t_2$. Finally, case (1) states that nothing is to correspond in $t_2$ to a $\tau$ transition in $t_1$, but for the recurrence over $R^\rho$.

Given two STGs $\mathcal{I} = (S_1, s_{01}, T_1)$ and $\mathcal{C} = (S_2, s_{02}, T_2)$, we write $\mathcal{I} \trianglerighteq^\rho \mathcal{C}$ if there exists a SBBC relation $\mathcal{R}^\rho$ between the terms formed by the two initial states with empty substitutions, *i.e.*, $(s_{01\emptyset}, s_{02\emptyset}) \in \mathcal{R}^\rho$. We can now give the formal definition of choreography conformance.

**Definition 5 (Choreography Conformance).** *Let $C$ be a choreography specification and $I$ be an implementation consisting of $n$ local entities $P_1, \ldots, P_n$, let $\mathcal{C}$ be the STG for $C$, $\mathcal{I}$ be the STG generated by the product of the STGs for $P_1, \ldots, P_n$, and $A$ be the alphabet of $\mathcal{C}$, $I$ conforms to $C$ iff $\mathcal{I} \downarrow_{\mathcal{A}}\ \trianglerighteq^{true} \mathcal{C}$.*

**Conformance computation.** Our algorithm for the computation of the SBBC relation between two STGs is a modification and simplification of the one proposed in [14] that computes symbolic weak bisimulation. Simplification was made possible due to the use of SBBC for choreography conformance: there may be $\tau$s in $\mathcal{I}$ but not in $\mathcal{C}$. The algorithm outputs a set of boolean formulas $\rho_{s_1,s_2}$ relative to pairs of states $(s_1, s_2)$, $s_1$ being in $\mathcal{I}$ and $s_2$ in $\mathcal{C}$. $\rho_{s_1,s_2}$ denotes the conditions under which $s_1$ and $s_2$ are SBBC related (Def. 4). In the algorithm, these boolean formulas are encoded as a Predicate Equation Systems (PESs) [13], *i.e.*, a set of predicate equations. A *predicate equation* (PE) is a function which contains a boolean expression, *e.g.*, $R(x) ::= (x \geq 0)$.

*Example 5.* Applying the algorithm on the STGs in Figure 6(b) (*i.e.*, restriction of Fig. 5(b)) and in Figure 4(a) (specification), we retrieve the following PES:

$$\begin{array}{ll}
R_{1,1}() & ::= \forall Z_0\ R_{2,2}(Z_0, Z_0) \\
R_{2,2}(y_1, x_1) & ::= (((x_1 \geq 5 \Rightarrow y_1 \geq 5 \wedge R_{3,2}(y_1, x_1)) \wedge (y_1 \geq 5 \Rightarrow x_1 \geq 5 \wedge R_{3,2}(y_1, x_1))) \\
 & \quad \wedge((x_1 < 5 \Rightarrow y_1 < 5 \wedge R_{4,3}) \wedge (y_1 < 5 \Rightarrow x_1 < 5 \wedge R_{4,3}))) \\
 & \quad \wedge(\neg(y < 10)) \\
R_{3,2}(y_1, x_1) & ::= ((x_1 \geq 5 \Rightarrow y \geq 10 \wedge R_{4,3}) \wedge (y \geq 10 \Rightarrow x_1 \geq 5 \wedge R_{4,3})) \\
 & \quad \wedge((\neg(y < 10)) \wedge (\neg(x_1 < 5))) \\
R_{4,3}() & ::= \mathit{true}
\end{array}$$

It can be simplified into $\{R_{1,1} ::= y \geq 10, R_{2,2} ::= y \geq 10, R_{3,2} ::= y \geq 10 \wedge Z_0 \geq 5, R_{4,3} ::= \mathit{true}, \}$ but this demonstrates the need for an automatic PES satisfiability checking procedure.

## 4 Implementation and Experiments

In this section, we give details on our tool-chain for choreography conformance checking. We also present some of the experiments we have made to evaluate it.

The architecture of our tool-chain is given in Figure 8. We take as input a choreography global specification $C$, with $m$ roles. We also take an implementation description $I$, given as $n \geq m$ entity local descriptions. These may correspond either to peer descriptions or to role requirements. The case when $n > m$ denotes, *e.g.*, an implementation where some peers have been added to make a choreography realizable. All inputs are first transformed into STGs. The product of STGs and the restriction to actions in $C$ are used to retrieve a unique STG for $I$, thus yielding two STGs to compare: one for $C$ ($\mathcal{C}$) and one for $I$ ($\mathcal{I}$). We then check if $\mathcal{I}$ conforms to $\mathcal{C}$, which generates the largest boolean formula $\rho$ such that the initial states of $\mathcal{I}$ and $\mathcal{C}$ are SBBC related. Finally, this formula is analysed using the Z3 SMT solver in order to reach a conformance verdict. This can be "always true" or "always false", "always" meaning whatever the data values

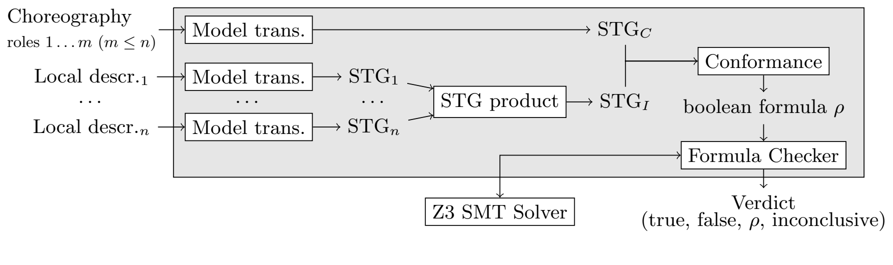


**Fig. 8.** Architecture of our tool-chain

exchanged between peers are. However, sometimes we can have conformance only for a subset of these values. Going further than pure true/false conformance, our tool-chain thus allows to compute the largest constraint on data values, $\rho$, that would yield conformance. Complex constraints may cause the solver to return a timeout. In such a case, we emit inconclusiveness as a verdict.

Our tool-chain may be downloaded from our Web pages[4]. The Z3 SMT solver[5] has to be installed separately for licence reasons. We plan to interface our tool-chain with the SMT-LIB API in order to let users choose other SMT solvers.

**STGs product.** The Algorithm 1 represents the implementation of the production rules in Figure 3. In this algorithm, we use $\sqcup$ to denote disjoint union, *i.e.*, $S_1 \sqcup S_2$ is defined only if $S_1 \cap S_2 = \emptyset$. We use also two other functions *renameVariable* and *updateFreeVariables*. The first one, *renameVariable($\mathcal{S}, s, v, v'$)*, renames the variable $v$ into $v'$ when it is used in guard of transitions after state $s$ of STG $\mathcal{S}$. The second one is used for updating the set of free variables of each state in order to satisfy the STG definition (Def 1).

**Algorithm 1:** Product of $n$ STGs

```
Data: $n$ STGs $\mathcal{S}_i = (S_i, s_{0i}, T_i)$
Result: a STG $\mathcal{S} = (S, s_0, T)$, the product of $\mathcal{S}_1, \ldots, \mathcal{S}_n$
$s_0 := \{s_{01}, s_{02}, \ldots, s_{0n}\}$ ;                          /* initial state */
$S := \{s_0\}$ ;                                                        /* set of nodes */
$T := \emptyset$ ;                                                /* set of transitions */
product ($\{s_{01}, s_{02}, \ldots, s_{0n}\}$) ;
 /* update set of free variables of each state to satisfy STG definition */
updateFreeVariables ($S, s_0, T$);
return ($S, s_0, T$) ;
product ($\{s_1, s_2, \ldots, s_n\}$) = begin
    $s := \{s_1, s_2, \ldots, s_n\}$;
    foreach $s_1 \xrightarrow{[\phi_{i_1 1}]\ \checkmark} s_{i_1 1},\ s_2 \xrightarrow{[\phi_{i_2 2}]\ \checkmark} s_{i_2 2},\ \ldots,\ s_n \xrightarrow{[\phi_{i_n n}]\ \checkmark} s_{i_n n}$, do
        $s' := \{s_{i_1 1}, \ldots, s_{i_n n}\}$;   $S := S \sqcup \{s'\}$;   $T := T \cup s \xrightarrow{[\phi_{i_1 1} \wedge \ldots \wedge \phi_{i_n n}]\ \checkmark} s'$;
    foreach $s_l \xrightarrow{[\phi_{il}]\ c^{[a,b]}!x_{il}} s_{il},\ s_k \xrightarrow{[\phi_{jk}]\ c^{[a,b]}?x_{jk}} s_{jk}$ with $l \neq k \wedge l, k \in \{1, \ldots, n\}$ do
        $s'$ is constructed from $s$ by replacing $x_l$ by $x_{il}$ and $x_k$ by $x_{jk}$ ;
        $S := S \sqcup \{s'\}$;   $T := T \cup s \xrightarrow{[\phi_{il} \wedge \phi_{jk}]\ c^{[a,b]}.x_{jk}} s'$;
        renameVariable ($\mathcal{S}_l, s_l, x_{il}, x_{jk}$) ;
        product ($s'$) ;
```

**Conformance computation.** Following [13], a PES can be written using substitutions in order to simplify its notation. For example, the PES $\{R_0() ::= \forall z\ R_1(z, z); R_1(x, y) ::= (x > y)\}$ is rewritten $\{R_0 ::= \forall z\ R_1([x/z, y/z]); R_1 ::= (x > y)\}$.

The main function is function `close`, called initially on the initial states of the two STGs we compare. This function compares two states, $s_1$ and $s_2$ up to some

[4] http://www.lri.fr/~nhnghia/tools/
[5] http://research.microsoft.com/en-us/um/redmond/projects/z3/

substitution $\sigma$, and returns the most general formula such that $s_1$ and $s_2$ are SBBC bisimilar. This is formula $false$ in the worst case ($s_1$ and $s_2$ are not branching bisimilar). `close` relies on the `match` function that encodes the different cases in Definition 4. $Trans(s_1, s_2) = \{\alpha | s_1 \xRightarrow{[\phi_1]\ \tau} s'_1 \xrightarrow{[\phi'_1]\ \alpha} s''_1\} \cap \{\alpha | s_2 \xrightarrow{[\phi_2]\ \alpha} s'_2\}$ denotes the common observable actions that can be performed both from $s_1$ and from $s_2$. Since function `close` visits a pair $(s_1, s_2) \in S_1 \times S_2$ at most once, it always terminates after a finite steps, as for the algorithm whose complexity is $\mathcal{O}(|S_1| \times |S_2|)$.

**Algorithm 2:** Conformance PES computation

```
Data: two STGs $\mathcal{I} = (S_1, s_{0,1}, T_1)$ and $\mathcal{C} = (S_2, s_{0,2}, T_2)$
Result: a PES, i.e., a set $P$ of PEs (one for each couple of states in a subset of $S_1 \times S_2$)
W := $\emptyset$ ;                                          /* visited couple nodes */
P := $\emptyset$ ;                          /* predicates for couples of visited nodes */
close ($s_{0,1}$, $s_{0,2}$, $\emptyset$) ;
return $P$ ;

close ($s_1$, $s_2$, $\sigma$) = begin
    if $(s_1, s_2) \notin W$ then
        $W := W \cup \{(s_1, s_2)\}$;   $R_{s_1,s_2}$ := match($s_1$, $s_2$, $\sigma$);   $P := P \cup \{R_{s_1,s_2}\}$ ;
    return $R_{s_1,s_2}(\sigma)$ ;                 /* a predicate $\rho_{s_1,s_2}$ with parameter $\sigma$ */

match ($s_1$, $s_2$, $\sigma$) = begin
    foreach $\gamma \in Trans(s_1, s_2)$ do             /* Trans($s_1$,$s_2$): set of actions $\neq \tau$ of */
        $\rho_\gamma$ := matchEv ($\gamma$, $s_1$, $s_2$, $\sigma$) ;                /* next transitions from $s_1$, $s_2$ */
    /* The others events ($\notin Trans(s_1, s_2)$) must not occur                             */
    foreach $s_1 \xRightarrow{[\phi'_{j,1}]\ \tau} s'_{j,1} \xrightarrow{[\phi_{j,1}]\ \alpha_{j,1}} s_{j,1}$ such that $\alpha_j \notin Trans(s_1, s_2)$ do
        $\rho_{j,1} := \neg\phi_{j,1}$ ;                                 /* exist $\tau$ in implementation */
    foreach $s_2 \xrightarrow{[\phi_{i,2}]\ \alpha_{i,2}} s_{i,2}$ such that $\alpha_i \notin Trans(s_1, s_2)$ do
        $\rho_{i,2} := \neg\phi_{i,2}$ ;                                    /* no $\tau$ in specification */
    return $\bigwedge_\gamma \rho_\gamma \wedge \bigwedge_j \rho_{j,1} \wedge \bigwedge_i \rho_{i,2}$ ;

matchEv ($\checkmark$, $s_1$, $s_2$, $\sigma$) = begin
    foreach $s_1 \xRightarrow{[\phi'_{j,1}]\ \tau} s'_{j,1} \xrightarrow{[\phi_{j,1}]\ \checkmark} s_{j,1}$, $s_2 \xrightarrow{[\phi''_{i,2}]\ \checkmark} s_{i,2}$ do
        /* if no $\tau$ between $s_1$ and $s'_{j,1}$, then $s_{j,1}$ and $s_{i,2}$ always conform           */
        if $s_1 \equiv s'_{j,1}$ then $\rho_{ij}$ := true; else $\rho_{ij}$ := close($s'_{j,1}, s_2, \sigma$) ;
        $\phi_{j,1} := \phi'_{j,1} \wedge \phi''_{j,1}$ ;
    return $\bigwedge_i(\phi_{i,2} \Rightarrow \bigvee_j(\phi_{j,1} \wedge \rho_{ij})) \wedge \bigwedge_j(\phi_{j,1} \Rightarrow \bigvee_i(\phi_{i,2} \wedge \rho_{ij}))$ ;

matchEv ($c$, $s_1$, $s_2$, $\sigma$) = begin
    $z$ := newVar () ;                                      /* create a new fresh variable */
    foreach $s_1 \xRightarrow{[\phi'_{j,1}]\ \tau} s'_{j,1} \xrightarrow{[\phi''_{j,1}]\ c.x_{j,1}} s_{j,1}$, $s_2 \xrightarrow{[\phi_{i,2}]\ c.x_{i,2}} s_{i,2}$ do
        if $s_1 \not\equiv s'_{j,1}$ then $\phi_{j1} := \phi'_{j,1}$;   $\rho_{ij}$ := close($s'_{j,1}, s_2, \sigma$);
        else $\phi_{j,1} := \phi''_{j,1}$;   $\rho_{ij} := \forall z$ close($s_{j,1}, s_{i,2}, \sigma[x_{j,1} \mapsto z, x_{i,2} \mapsto z]$) ;
    return $\bigwedge_i(\phi_{i,2} \Rightarrow \bigvee_j(\phi_{j,1} \wedge \rho_{ij})) \wedge \bigwedge_j(\phi_{j,1} \Rightarrow \bigvee_i(\phi_{i,2} \wedge \rho_{ij}))$ ;
```

**PES satisfiability and conformance verdict.** The formula resulting from conformance checking is under the form of a PES (see Section 3). It has to be analysed in order to reach a conformance verdict. This step is performed with Z3, a state-of-the art theorem prover from Microsoft Research that can be used to check for the satisfiability of a set of formulas, *i.e.*, find if there is an interpretation that makes all asserted formulas true.

**Listing 1.1.** Translation into the Z3 language of the PES in Example 5

```
; encoding of the PES - it can be tried online at: http://rise4fun.com/z3
; note: newlines should be added manually if copy/paste from PDF is used
(set-option :print-warning false)
(declare-fun y () Int)
(define-fun R4_3 () Bool true)
(define-fun R3_2 ((y_1 Int)(x_1 Int)) Bool (and (and (implies (>= x_1 5) (and
   (>= y 10) R4_3)) (implies (>= y 10) (and (>= x_1 5) R4_3))) (and (not (< y
   10)) (not (< x_1 5)))))
(define-fun R2_2 ((y_1 Int)(x_1 Int)) Bool (and (and (and (implies (>= x_1 5)
   (and (>= y_1 5) (R3_2 y_1 x_1))) (implies (>= y_1 5) (and (>= x_1 5) (R3_2
   y_1 x_1)))) (and (implies (< x_1 5) (and (< y_1 5) R4_3)) (implies (< y_1
  5) (and (< x_1 5) R4_3)))) (not (< y 10))))
(define-fun R1_1 () Bool (forall ((Z_0 Int)) (R2_2  Z_0 Z_0)))
; uncomment for step 1, comment for step 2
(assert (= R1_1 false))
; comment for step 1, uncomment for step 2
; (assert (= R1_1 true))
(check-sat)
```

In order to use Z3, we translate the PES into the Z3 input language as demonstrated in Listing 1.1 for the PES in Example 5. Each predicate equation in the PES is translated as a boolean function (using *define-fun*) and each free variable is translated as an integer function (using *declare-fun*). In our example, variables are integers, but we stress out that complex types such as XML ones can be supported too, following [20].

We check `R1_1` in order to conclude on conformance. For this, the `check-sat` Z3 command is run following Table 2. If `R1_1` asserted *false* (as in Listing 1.1) yields an *unsat* response then there is no interpretation such that $R_{1,1}$ is false, hence we can conclude directly that conformance is *true*. Otherwise, we have to retry with `R1_1` asserted to *true* to reach a verdict.

**Table 2.** Decision table for conformance based on PES satisfiability checking

| `check-sat` response | | conformance decision |
|---|---|---|
| `R1_1` asserted *false* | `R1_1` asserted *true* | |
| *unsat* | not needed | *true* |
| otherwise | *unsat* | *false* |
| | *sat* | $\rho_{1,1}$ $(R_{1,1})$ |
| | *timeout* | *inconclusive* |

**Experiments.** We have experimented our tool-chain, including on examples from the literature (Tab. 3). For the implementations and the specifications, we respectively give the numbers of peers, roles, interactions, and transitions and states in the corresponding STGs. We also give the conformance verdicts in the paper the example is taken from and with our approach. Finally, we give the

**Table 3.** Experimental results

| Id | Name [Reference] | #Peers/Roles | Implementation | | Specification | | Verdict | Duration |
|---|---|---|---|---|---|---|---|---|
| | | | #Int. | #Trans./States | #Int. | #Trans./States | Orig./Ours | (seconds) |
| 01 | Shipping [n/a] | 2/2 | 3 | 4/4 | 3 | 4/4 | -/YES | 0.069 |
| | | 2/2 | 4 | 5/5 | 3 | 4/4 | -/YES | 0.084 |
| | **Example 5** $\rightarrow$ | 2/2 | 4 | 6/5 | 3 | 4/4 | -/$\rho$ | 0.102 |
| 04 | Market [9] | 4/4 | 8 | 9/10 | 8 | 10/10 | YES/NO | 0.118 |
| | | 8/4 | 16 | 27/26 | 8 | 10/10 | YES/NO | 0.201 |
| 06 | RFQ [10] | 3/3 | 6 | 8/7 | 6 | 8/8 | NO/NO | 0.078 |
| 07 | Booking [7] | 4/4 | 8 | 12/11 | 8 | 12/11 | YES/YES | 0.096 |

execution time (Mac Book Air with OS 10.7, 4 GB RAM, core i5 1.7 GHz) for the process described in Figure 8 (but for the time to parse the input files).

Rows 1 to 3 correspond to the specification STG in Figure 4(a) and, respectively, to the implementations STGs in Figures 4(d) (row 1), 5(a) (row 2), and 5(b) (row 3). Rows 4 and 5 correspond to the example and mutation in [9]. The difference in the verdict comes from the fact the we distinguish between an STG ending with ✓ (successful termination) or not, hence an implementation deadlocking after achieving all interactions of a specification will not conform to it: the specification may do ✓ while the implementation may not. Row 6 corresponds to a negative example in [10] and row 7 to a positive one in [7].

## 5 Conclusion

In this paper, we have proposed a formal framework for checking the conformance of a set of role requirements or peer descriptions with reference to a choreography specification. Symbolic models and equivalences enable us to check conformance in presence of data without suffering from state space explosion and without bounding data domains. Going further than strict conformance, we are able to give the most general constraint over data exchanged between peers in order to achieve conformance. Our approach is fully automated with tools we have developed and the use of the Z3 SMT solver.

We advocate that once a choreography projection function supporting data is defined, then our framework could be used not only for conformance checking but also for realizability checking. This is our first perspective. A second perspective is to extend our framework in order to support assignment and asynchronous communication. BPMN is the standard notation for business processes and supports choreography since version 2.0. Our last perspective is to integrate the extensions of our tools as a verification plugin for the BPMN 2.0 Eclipse editor. A BPMN 2.0 to STG model transformation is ongoing, based on our BPMN to LTS (no data) one [21].

## References

1. Decker, G., Kopp, O., Barros, A.: An Introduction to Service Choreographies. Information Technology **50**(2) (2008) 122–127
2. Lohmann, N., Wolf, K.: Decidability Results for Choreography Realization. In: Proc. of ICSOC'11. (2011)
3. Kaschner, K.: Conformance Testing for Asynchronously Communicating Services. In: Proc. of ICSOC'11. (2011)
4. Nguyen, H.N., Poizat, P., Zaïdi, F.: Passive Conformance Testing of Service Choreographies. In: Proc. of SAC'12. (2012)
5. Qiu, Z., Zhao, X., Cai, C., Yang, H.: Towards the Theoretical Foundation of Choreography. In: Proc. of WWW '07. (2007)
6. Basu, S., Bultan, T.: Choreography Conformance via Synchronizability. In: Proc. of WWW'11. (2011)
7. Salaün, G., Bultan, T., Roohi, N.: Realizability of Choreographies using Process Algebra Encodings. In: Proc. of IFM'09. (2009)
8. Li, J., Zhu, H., Pu, G.: Conformance Validation between Choreography and Orchestration. In: Proc. of TASE'07. (2007)
9. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and Orchestration Conformance for System Design. In: Proc. of COORDINATION'06. (2006)
10. Kazhamiakin, R.: Choreography Conformance Analysis : Asynchronous Communications and Information Alignment. In: Proc. of WS-FM'06. (2006)
11. Van Glabbeek, R., Weijland, W.: Branching Time and Abstraction in Bisimulation Semantics. Journal of the ACM **43**(3) (1996)
12. Hennessy, M., Lin, H.: Symbolic Bisimulations. Theoretical Computer Science **138**(2) (1995) 353–389
13. Lin, H.: Symbolic Transition Graph with Assignment. In: Proc. of CONCUR'96. (1996)
14. Li, Z., Chen, H.: Computing Strong/Weak Bisimulation Equivalences and Observation Congruence for Value-Passing Processes. In: Proc. of TACAS'99. (1999)
15. Basu, S., Mukund, M., Ramakrishnan, C., Ramakrishnan, I., Verma, R.: Local and Symbolic Bisimulation Using Tabled Constraint Logic Programming. In: Proc. of 17th ICLP. (2001)
16. Bravetti, M., Zavattaro, G.: Towards a Unifying Theory for Choreography Conformance and Contract Compliance. In: Proc. of SC'07. (2007)
17. Poizat, P.: Formal Model-Based Approaches for the Development of Composite Systems. Habilitation thesis, Université Paris Sud (November 2011) http://www.lri.fr/~poizat/documents/hdr_poizat.pdf.
18. Kopp, O.: Do We Need Internal Behavior in Choreography Models? In: Proc. of 1st ZEUS. (2009)
19. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
20. Bentakouk, L., Poizat, P., Zaïdi, F.: Checking the Behavioral Conformance of Web Services with Symbolic Testing and an SMT Solver. In: Proc. of TAP'11. (2011)
21. Poizat, P., Salaün, G.: Checking the Realizability of BPMN 2.0 Choreographies. In: Proc of SAC'12. (2012)