Event algebra for transition systems composition Application to timed automata

Elie Fares, Jean-Paul Bodeveix, M Filali

Event algebra for transition systems composition application to timed automata

Elie Fares[1] · **Jean-Paul Bodeveix** · **Mamoun Filali**[2]

Abstract Formal specification languages have a lot of notions in common. They all introduce entities usually called processes, offer similar operators, and most importantly define their operational semantics based on labelled transition systems (LTS). However, each language defines specific synchronizing and/or memory structures. For instance, in CSP, the synchronization is defined between identical events, while in CCS and in synchronization vectors-based views it is defined respectively between complementary events or between possibly different events. In this paper, we aim at capturing some similarities of specification languages by defining a label-based formal framework for reasoning on LTS, their seman-tics and related properties. Firstly, we define a high-level synchronization mechanism in the form of an abstract label structure and identify some properties. Then, we introduce oper-ators for composing and transforming label structures, study their intrinsic properties and explore how label structure properties propagate. Secondly, we introduce a LTS-based behav-ioral framework. We then lift the label structure composition and transformation operators to the LTS level and establish LTS-related properties derived from those of their underlying labelled structures. Thirdly, we consider extended transition systems, more specifically timed automata, as LTS built on top of specific labelled structures. Their semantics is reconstructed by applying operators of our framework on the syntactic LTS, which allows the direct proof of some semantic properties such as compositionality.

1 Introduction

For the past three decades, specification languages such as CSP [32], CCS [29], LOTOS [25], μ CRL [21], Altarica [6], and BIP [7] have proven valuable in the specification and design of concurrent and distributed systems. The behavioral aspects of these languages share a common base since they all define their operational semantics in terms of labelled transition systems. Yet, the difference lies in the synchronizing structure of the labels of these systems.

✉ Jean-Paul Bodeveix
bodeveix@irit.fr
IRIT - Université Paul Sabatier, Toulouse, France
2 IRIT - CNRS, Université de Toulouse, Toulouse, France

For example in CSP the synchronization is defined between two identical events, while in CCS and in synchronization vectors-based views, it is defined respectively between complementary events or between possibly different events. Through the years, the basic versions of some of these languages have been extended by time, memory, and priority notions. Accordingly, other formalisms have emerged in order to model the semantics of these extensions. For example, we can cite Alur and Dill's timed automata [4] and Henzinger et al's timed transition systems [22] that both capture the time addition or the semantic model of [35] used to model priorities. However, even though the rules of the composition operations of these formalisms are the same in nature (synchronous and asynchronous rules), each formalism is specified with specific sets of rules, maybe because of the specific attributes that come with each formalism. A distinct composition operation is then introduced for each defined formalism.

In this paper, we aim at capturing some similarities of specification languages by providing a semantic framework for system composition. For this purpose, we introduce a high-level synchronization mechanism in the form of a label structure. It is equipped with a composition operator which encapsulates the specific composition laws of each language and would further serve as a parameter of the behavioral framework. Thanks to the separation between the composition laws and the behavioral framework, the latter, which is based on LTS, offers a unique LTS composition which is reused to define syntactic composition of timed automata and a compositional semantics.

The idea of a label structure or similar constructs is not new since it appears in earlier studies [12,24,26–28,34]. In [12,26–28], the label composition operator appears under a functional form (the same as ours, see Label Structure: Sect. 2) but the authors do not go beyond this definition. They simply use it in order to model the composition, whether it is blocking or not, of multiple labels. Their work is not intended to build a whole framework around label structures. In [24,34], it appears under a relational form. In these latter studies, the authors are interested in reaching generic semantic rules for process calculi behavioral operators (prefix, choice, …). This is orthogonal to our goal that consists in reaching a label-based composition framework for specification languages. In Sect. 4, we cover the link of our work with existing work.

Our contribution can be viewed from two perspectives. One way to see this work is as an abstraction of the composition of different behavioral formalisms via a separation of the label composition laws of each language and a reuse of the LTS composition. In fact, depending on the language, the label structure is defined and instantiated differently. Using the common framework, one would then proceed by giving the semantics of other behavioral operators of the specification language in question. Another way to see our label structure and their associated operations is as a generalization of the composition functions given and used in [26,27]. Indeed, we show how such abstract composition functions (or label structures in our terms) may be implemented and instantiated to simulate existing synchronization mechanisms at the LTS level. Moreover, in order to ease the proofs of properties about label structures, we propose composition and transformation operators which preserve the considered properties. Some of these operators are then lifted to the LTS level. This allows the conversion of a LTS over one label structure to a LTS over another label structure while preserving some properties. This mechanism is illustrated on the timed automata (TA) theory where we show how TA semantics can be reconstructed using operators provided by our framework. In addition to behavioural formalisms, semantic properties can also be derived from properties of our LTS and label structure operators. This is illustrated by proving product compositionality on two variants of timed automata, which highlights the originality of the proposed approach. Furthermore, all the results presented here have been formalized and verified using the proof assistant Coq [33].

This work is a revised and extended version of [20]. Namely, we revise some of our definitions and properties, we add some others, and we extend our theorems, propositions and corollaries with their corresponding proofs. Other proofs are also given in the appendix section. This revision is in general a more self-contained work. It offers the reader a more readable work thanks to the simplification that comes with the revised definitions.

The rest of the paper is organized as follows. In the second section, we start by defining label structures along with their properties of interest. Then, we propose operations for composing and transforming label structures. We study their own properties as well as the preservation of label structure properties. In the third section, we introduce labelled transition systems on top of label structures and define our behavioral framework which reuses the label structure notions. The fifth section is dedicated to the representation of timed systems and their semantics by Labelled Transition Systems over suitable label structures. Section five proposes a general framework used to attach a semantics to extended transition systems. We conclude the paper in the sixth section.

2 Label structure

We first introduce basic mathematical notions that will be used throughout this paper. Then, we define label structures, give some examples associated to usual process calculi, define operators on label structures, state properties and study their preservation.

2.1 Basic mathematical notions

Some mathematical notions and notations will be used throughout this paper. We summarize them here:

- Given two sets A and B, their disjoint union will be denoted as $A+B$. We write $a^\bullet$ and ${}^\bullet b$ to denote respectively the application of the left and right injection on an element $a \in A$ and $b \in B$, resulting in an element of $A + B$, also denoted $A^\bullet \cup {}^\bullet B$. As a consequence, for any set A, $A^\bullet = \{a^\bullet \mid a \in A\}$ and ${}^\bullet A = \{{}^\bullet a \mid a \in A\}$. The $\_^\bullet$ and ${}^\bullet\_$ annotations denote respectively the left and right embedding of A and B inside the disjoint union. Furthermore, injections may be omitted when A and B are disjoint.
- Given two sets A and B, we denote by $A \leftrightarrow B \triangleq 2^{A\times B}$ the set of relations between A and B. Given two relations $R_1 \in A_1 \leftrightarrow B_1$ and $R_2 \in A_2 \leftrightarrow B_2$, we introduce the sum and product operators over relations as:
  $R_1 \oplus R_2 = \{(x^\bullet, y^\bullet) \mid (x, y) \in R_1\} \cup \{({}^\bullet x, {}^\bullet y) \mid (x, y) \in R_2\}$
  $R_1 \otimes R_2 = \{((x_1, x_2), (y_1, y_2)) \mid (x_1, y_1) \in R_1 \wedge (x_2, y_2) \in R_2\}$
- Given two sets A and B, we denote as $A \nrightarrow B$ the set of partial functions from A to B. Given a partial function $f \in A \nrightarrow B$, we will consider its domain $\mathbf{dom}(f) \subseteq A$ and its range $\mathbf{ran}(f) \subseteq B$. For a subset $S \subseteq A$, $f[S]$ denotes the set of images by f of elements of $S \cap \mathbf{dom}(f)$.
- Given two structures $S = \langle A, o_A\rangle$ and $T = \langle B, o_B\rangle$, a (partial) morphism from S to T is a (partial) function f from A to B that commutes with the operators of the structure: if o_A and o_B are binary, we should have $\forall x, y \in A \cdot f(x\ o_A\ y) = f(x)\ o_B\ f(y)$ when both expressions are defined.
- A partial order is a binary relation which is reflexive, antisymmetric and transitive. If R_1 and R_2 are partial orders, their sum[1] $(R_1 \oplus R_2)$ and their product $(R_1 \otimes R_2)$ are partial orders.

[1] not their union.

– Given a partially ordered set S, the join of $a \in S$ and $b \in S$ is, when it does exist, the least upper bound of $\{a, b\}$ in S. A join semilattice is a partially ordered set where each pair of elements has a join.

2.2 Label structures

Definition 1 (*Label Structure*) A label structure is a tuple $\langle L, \bowtie \rangle$ where L is a set of labels and $(\bowtie\colon L \times L \nrightarrow L)$ is a partial binary composition operator (called the fusion operator) over L.

The function is partial because some compositions may be blocked since $\bowtie$ describes exclusively synchronous compositions. The asynchronous aspects are covered later (see LTS composition). Our composition then models the following cases:

1. A successful synchronization between l and l' that results in $l \bowtie l' \in L$.
2. A blocking synchronization between l and l': $(l, l') \notin \mathbf{dom}(\bowtie)$.

Let the reader not confuse our label structure with other event structuring propositions, namely with the event structures [15]. Event structures model the occurrence of events during the system *execution* via the introduction of a causal dependency relation and a conflict relation between the events. In our case, we introduce a label structure which models the way the labels (i.e., events) are *statically* composed.

Definition 2 (*Commutativity of a Label Structure*) Given a label structure $LS = \langle L, \bowtie \rangle$, LS is said to be commutative if its composition operator $\bowtie$ is commutative. Formally, LS is commutative if for any $l_1, l_2 \in L$, we have:

$$(l_1, l_2) \in \mathbf{dom}(\bowtie) \Rightarrow (l_2, l_1) \in \mathbf{dom}(\bowtie) \wedge l_1 \bowtie l_2 = l_2 \bowtie l_1$$

Definition 3 (*Associativity of a Label Structure*) Given a label structure $LS = \langle L, \bowtie \rangle$, LS is said to be associative if its composition operator $\bowtie$ is associative. Formally, LS is associative if the following conditions are satisfied for any l_1, l_2 and $l_3 \in L$:

1. $(l_1, l_2) \in \mathbf{dom}(\bowtie) \wedge ((l_1 \bowtie l_2), l_3) \in \mathbf{dom}(\bowtie) \Leftrightarrow (l_2, l_3) \in \mathbf{dom}(\bowtie) \wedge (l_1, (l_2 \bowtie l_3)) \in \mathbf{dom}(\bowtie)$. This means that independently of the composition order, both expressions are defined at the same time.
2. $(l_1, l_2) \in \mathbf{dom}(\bowtie) \wedge ((l_1 \bowtie l_2), l_3) \in \mathbf{dom}(\bowtie) \Rightarrow ((l_1 \bowtie l_2) \bowtie l_3) = (l_1 \bowtie (l_2 \bowtie l_3))$. This means that independently of the composition order, both expressions lead to the same result.

A Coq transcript of the definition of a label structure, its commutativity and its associativity are given in Appendix A.

Definition 4 (*Unital Label Structure*) Given a label structure $LS = \langle L, \bowtie \rangle$, LS is said to be unital if there exists an element τ (said to be neutral) such that:

1. $(\tau, \tau) \in \mathbf{dom}(\bowtie)$
2. $\forall l \in L \cdot (\tau, l) \in \mathbf{dom}(\bowtie) \Rightarrow \tau \bowtie l = l$
3. $\forall l \in L \cdot (l, \tau) \in \mathbf{dom}(\bowtie) \Rightarrow l \bowtie \tau = l$

Note that due to the partiality of the $\bowtie$ operator, a label structure can have several neutral elements.

Unital label structures will be used to define the extensions of the $\bowtie$ operator to the cartesian product (see ¶2.6.3).

2.3 Label structure examples

Basic CSP Synchronizing Structure. Here we model the case of the completely synchronous composition of CSP. For C a set of communication ports, a synchronizing structure on C is the label structure:

$$Sync_{CSP} = \langle C, (c_1, c_2) \mapsto c_1 \textbf{ if } c_1 = c_2 \rangle$$

The synchronization of two ports of the set C is only defined when these two ports are the same. Otherwise events interleave.

CCS Synchronizing Structure. For C a set of events, $\overline{C} = \{\overline{c} \mid c \in C\}$, $\tau \notin C \cup \overline{C}$, the CCS label structure is represented as follows:

$$Sync_{CCS} = \langle C \cup \overline{C} \cup \{\tau\}, \{(c, \overline{c}) \mapsto \tau \mid c \in C\} \cup \{(\overline{c}, c) \mapsto \tau \mid c \in C\} \rangle$$

Time Label Structure. For Δ a time domain, e.g., non negative real numbers, natural numbers, etc., we introduce the time structure TS on the domain Δ. Its composition operator is only defined between identical time labels δ and returns the label itself.

$$TS = \langle \Delta, (\delta_1, \delta_2) \mapsto \delta_1 \textbf{ if } \delta_1 = \delta_2 \rangle$$

We can note that time label structures on a non-empty domain are unital with any element as neutral. Furthermore, we can remark that this definition is exactly the same as CSP label structure. In fact, they differ with respect to sequential composition, which is not addressed in this paper.

Modal Label Structure. Modal transition systems, as defined by [8] for example, introduce *may* and *must* modalities. These modalities are significant with respect to refinement: a *must* transition must be preserved while a *may* transition may be removed. Here, we consider the product of modalities. It is defined as follows: Thus, the corresponding label structure is

$\|$	must	may
must	must	may
may	may	may

defined by:

$$MM = \left\langle \{\text{may}, \text{must}\}, \begin{pmatrix} \text{must}, \text{must} \mapsto \text{must} \\ m_1, m_2 \mapsto \text{may } \textbf{if } m_1 \neq \text{must} \vee m_2 \neq \text{must} \end{pmatrix} \right\rangle$$

Resource Label Structure. In the same way, we can manage additive resource consumption as it is modelled in process calculi such as PARS [30] or ACSR [16]. For instance, in ACSR, labels are partial finite maps from a set of resources R to a time domain Δ. Composition is the union of the maps and is defined if domains are disjoint:

$$RSC = \langle R \nrightarrow \Delta, (r_1, r_2) \mapsto r_1 \cup r_2 \textbf{ if } \textbf{dom}(r_1) \cap \textbf{dom}(r_2) = \emptyset \rangle$$

Table 1 Label structure properties

Property	Definition	Examples
Idempotence	$\forall l \in L \cdot (l, l) \in \mathbf{dom}(\ltimes) \wedge l \ltimes l = l$	$Sync_{CSP}, TS$
Unique composition	$\forall l_1, l_2 \in L \cdot (l_1, l_2) \in \mathbf{dom}(\ltimes) \Rightarrow \forall l \in L \cdot ((l_1 \ltimes l_2), l) \notin \mathbf{dom}(\ltimes) \wedge (l, (l_1 \ltimes l_2)) \notin \mathbf{dom}(\ltimes)$	$Sync_{CCS}$
Diagonality	$\forall l_1, l_2 \in L \cdot (l_1, l_2) \in \mathbf{dom}(\ltimes) \Rightarrow l_1 = l_2$	$Sync_{CSP}, TS$

2.4 Additional label structure properties

In addition to associativity and commutativity, our label structure may enjoy other properties. Table 1 defines some of the properties frequently used in this paper. The process calculi CSP and CCS and timed transition systems provide examples of label structures satisfying these properties.

Remark. The unique composition is a sufficient condition for composition associativity.

We denote by ***ACI*** the conjunction of the associativity, commutativity and idempotence properties. A label structure satisfying the ***ACI*** property is seen as a join semilattice where $\ltimes$ is interpreted as the join operator and the partial order relation is introduced as $l \leq l' \triangleq (l, l') \in \mathbf{dom}(\ltimes) \wedge l \ltimes l' = l'$.

The $\leq$ relation will be used to relate required actions, e.g., by a client and actual actions, executed, e.g., by a server. The partial order allows the synchronous composition of several client requests with the server. Such a use is illustrated in Sect. 2.7.2.

Proposition 1 (Join semilattice) *Given an **ACI** label structure, the associated $\leq$ relation is a partial order and $\ltimes$ corresponds to a partial supremum for $\leq$.*

This well-known property is easy to establish as shown by the following proof sketch. To ease the reading, the fact that calls to $\ltimes$ are defined is not discussed but directly comes from the definitions of ***ACI*** and $\leq$.

- Reflexivity: $l \leq l \equiv l \ltimes l$: true thanks to Hypothesis ***I***.
- Antisymmetry:

$$\begin{aligned} & l_1 \leq l_2 \wedge l_2 \leq l_1 \\ & \quad \Rightarrow l_1 \ltimes l_2 = l_2 \wedge l_2 \ltimes l_1 = l_1 \\ & \quad \Rightarrow l_1 = l_2 \ \text{ thanks to Hypothesis } \boldsymbol{C}. \end{aligned}$$

- Transitivity:

$$\begin{aligned} & l_1 \leq l_2 \wedge l_2 \leq l_3 \\ & \quad \Rightarrow l_1 \ltimes l_2 = l_2 \wedge l_2 \ltimes l_3 = l_3 \\ & \quad \Rightarrow l_1 \ltimes l_3 = l_1 \ltimes (l_2 \ltimes l_3) = (l_1 \ltimes l_2) \ltimes l_3 = l_2 \ltimes l_3 = l_3 \\ & \quad \text{thanks to Hypothesis } \boldsymbol{A}. \end{aligned}$$

2.5 Operators and properties over sets of labels

Given a label structure $LS = \langle L, \bowtie \rangle$, we define some operations and properties over sets of labels of LS. The notations and names we use recall those usually used in the context of the cartesian product. They are needed to express sufficient conditions for the associativity of the composition of labelled transition systems (Theorem 2) and to define stability (Definition 7).

Definition 5 (Set Projection) Given a set $S \subseteq L$, we define its left and right projections as:[2]

$$S{\downarrow_1} = \{l_1 \in L \mid \exists l_2 \in L \cdot (l_1, l_2) \in \mathbf{dom}(\bowtie) \wedge l_1 \bowtie l_2 \in S\}$$
$$S{\downarrow_2} = \{l_2 \in L \mid \exists l_1 \in L \cdot (l_1, l_2) \in \mathbf{dom}(\bowtie) \wedge l_1 \bowtie l_2 \in S\}$$

Definition 6 (*Set Extension*) Given a set $S \subseteq L$, we define its left and right extensions as:

$$S{\uparrow^1} = \{l_1 \bowtie l_2 \mid l_1 \in S \wedge l_2 \in L \wedge (l_1, l_2) \in \mathbf{dom}(\bowtie)\}$$
$$S{\uparrow^2} = \{l_1 \bowtie l_2 \mid l_1 \in L \wedge l_2 \in S \wedge (l_1, l_2) \in \mathbf{dom}(\bowtie)\}$$

Note that none of the usual results for the cartesian product about projection and extension hold here. In the following, we introduce a property called stability which introduces hypotheses on the previous operators.

Definition 7 (*Stability of a set of labels*) Given a set of labels $S \subseteq L$ and LS a label structure we say that:

- S is p-stable over LS if $S{\downarrow_1} \subseteq S$ and $S{\downarrow_2} \subseteq S$,
- S is e-stable over LS if $S{\uparrow^1} \subseteq S$ and $S{\uparrow^2} \subseteq S$
- S is stable over LS if S is both p-stable and e-stable over LS.

This property will be used to express sufficient conditions for the associativity of LTS parallel composition operator (see Theorem 2).

2.6 Composition of label structures

We define the product and the sum of two label structures and the option operator. The product operation builds new labels as pairs of the composed labels. For example, this is used when composing synchronization and memory access labels. Unlike the product operation, the labels of the sum operation are defined over the union of the composed labels. This is used when composing synchronization and time labels to specify that only one of the events may occur. The option operator introduces a neutral element within the label structure. It will be used to embed a label into a product type, the missing argument being the neutral element of the option type.

2.6.1 Product of label structures

The aim of the *product of label structures* is to define composite labels. In the following, we build composite labels from synchronization labels and memory operations. Using the same mechanism, it would be possible to attach *may/must* modalities to existing labels (the structured labels of [8]), or weights and probabilities, as in [17].

[2] The set projection could also be defined as $S{\downarrow_1} = \mathbf{fst}[\bowtie^{-1}[S]]$ where $\mathbf{fst}(a, b) = a$.

In order to introduce the product label structure, we need to define the $\bowtie$ operator on these elements. It applies the fusion operator of the two label structures on composable elements. More precisely:

$$\langle L, \bowtie \rangle \otimes \langle L', \bowtie' \rangle = \langle L \times L', \bowtie_p \rangle$$

where $\bowtie_p$ is defined as follows:

$$((l_1, l_1'), (l_2, l_2')) \mapsto (l_1 \bowtie l_2, l_1' \bowtie' l_2') \textbf{ if } (l_1, l_2) \in \textbf{dom}(\bowtie) \ \wedge \ (l_1', l_2') \in \textbf{dom}(\bowtie')$$

The domain of $\bowtie_p$ is thus the direct product of $\textbf{dom}(\bowtie)$ and $\textbf{dom}(\bowtie')$.
A transcript of the modelling of the product operation in Coq is given in Appendix B.

2.6.2 Sum of label structures

The aim of the *sum of label structures* is to introduce labels that are either labels from one label structure or from the other. Given $\langle L, \bowtie \rangle$ and $\langle L', \bowtie' \rangle$, their sum ranges over the disjoint union $L + L'$. The join operator is defined on sum labels if they both originate from the same set of labels. In this case, the corresponding join operator is applied. Formally, the sum is defined as:

$$\langle L, \bowtie \rangle \oplus \langle L', \bowtie' \rangle = \left\langle L + L', \begin{pmatrix} (l_1{}^\bullet, l_2{}^\bullet) \mapsto (l_1 \bowtie l_2)^\bullet \textbf{ if } (l_1, l_2) \in \textbf{dom}(\bowtie) \\ ({}^\bullet l_1', {}^\bullet l_2') \mapsto {}^\bullet(l_1' \bowtie' l_2') \textbf{ if } (l_1', l_2') \in \textbf{dom}(\bowtie') \end{pmatrix} \right\rangle$$

A transcript of the modelling of the sum operation in Coq is given in Appendix C.

2.6.3 Optional label structure

The aim of the *option operator over label structures* is to introduce an optional label τ which will play the role of a neutral element for the product. In fact, a label becomes optional if it provides the τ element which is supposed not belonging to L. Given a label structure $LS = \langle L, \bowtie \rangle$, the label structure LS^τ is defined as:

$$\langle L \cup \{\tau\}, \bowtie_\tau \rangle$$

where $\bowtie_\tau$ is defined by:

$$\begin{array}{lll} (l, l') & \mapsto l \bowtie l' & \textbf{if } (l, l') \in \textbf{dom}(\bowtie) \\ (\tau, \tau) & \mapsto \tau & \\ (\tau, l) & \mapsto l & \textbf{if } l \in L \\ (l, \tau) & \mapsto l & \textbf{if } l \in L \end{array}$$

The domain of $\bowtie_\tau$ is thus defined as:

$$\textbf{dom}(\bowtie_\tau) = \textbf{dom}(\bowtie) \cup (\{\tau\} \times L) \cup (L \times \{\tau\}) \cup \{\tau, \tau\}$$

Remark. The optional label structure is unital with τ as a neutral element.

2.6.4 Preservation of ACI properties

Proposition 2 (Preservation of ***ACI***) *Given LS and LS', if LS and LS' satisfy one of the **ACI** properties then $LS \oplus LS'$, $LS \otimes LS'$ and LS^τ satisfy this same property.*

As an example, we give a sketch of the proof of the ***ACI*** preservation by the $\oplus$ operator. Given the two label structures LS and LS' supposed to be respectively ***A***, ***C*** or ***I***, we prove the corresponding property on their sum.

- *$LS \oplus LS'$ associativity*: given l_1, l_2, l_3 such that $(l_1 \bowtie l_2) \bowtie l_3$ exists, they must all be a left embedding or all be a right embedding. The result directly comes from the ***A*** property of either LS or LS'.
- *$LS \oplus LS'$ commutativity*: given l_1, l_2 such that $l_1 \bowtie l_2$ exists, both must be a left embedding or both a right embedding. Then we get the result from the ***C*** property of either LS or LS'.
- *$LS \oplus LS'$ idempotence*: $l \bowtie l$ -defined as l- is either a left embedding or a right embedding and LS and LS' have the ***I*** property. Then, the product is l.

2.7 Label structure transformations

We introduce morphisms between label structures as partial functions from labels to labels which have some additional properties selected to help proving compositionality results on transformations of transition systems (Paragraph 3.2). These results will eventually be used to establish properties about the semantics of timed automata. A label structure transformation is a homomorphism between label structures. We start by giving the definition of a transformation. Then we show instances of such transformations that will be used later to define the semantics of timed automata (Sect. 5.2).

Definition 8 (*Transformation*) A transformation f between two label structures $LS_1 = \langle L_1, \bowtie_1 \rangle$ and $LS_2 = \langle L_2, \bowtie_2 \rangle$ is defined as a partial morphism f from LS_1 labels to LS_2 labels.

As label composition and label transformation are partial, a transformation is characterized by the three following conditions:

- the composition of two transformable labels is transformable: $\forall l, l' \in \mathbf{dom}(f) \cdot (l, l') \in \mathbf{dom}(\bowtie_1) \Rightarrow l \bowtie_1 l' \in \mathbf{dom}(f)$
- the images of composable labels are composable: $\forall l, l' \in \mathbf{dom}(f) \cdot (l, l') \in \mathbf{dom}(\bowtie_1) \Rightarrow (f(l), f(l')) \in \mathbf{dom}(\bowtie_2)$.
- composition and transformation commute: $\forall l, l' \in \mathbf{dom}(f) \cdot (l, l') \in \mathbf{dom}(\bowtie_1) \Rightarrow f(l \bowtie_1 l') = f(l) \bowtie_2 f(l')$.

We write $f : LS_1 \Rrightarrow LS_2$ to denote such a transformation. We now introduce two other properties which are not satisfied by all the transformations we will consider (Paragraph 2.7.1). These properties are sufficient conditions for some equalities between labelled transition systems. They are for example among the conditions of Theorem 3 and Corollary 5 of page 21 which express that label structure transformations lifted to labelled transition systems commute with parallel composition.

Definition 9 (*Reverse composability*) We say that a transformation $f : LS_1 \Rrightarrow LS_2$ is reverse composable, denoted r_composable(f), if when transformed labels are composable, source labels are also composable:

$$\forall l, l' \in \mathbf{dom}(f) \cdot (f(l), f(l')) \in \mathbf{dom}(\bowtie_2) \Rightarrow (l, l') \in \mathbf{dom}(\bowtie_1)$$

Definition 10 (*Reverse compatibility*) We say that a transformation $f : LS_1 \Rrightarrow LS_2$ is reverse compatible with a set of labels S of LS_1 if when two labels have the same image, they are both in S or both outside S:

$$\forall l_1 l_2 \cdot f(l_1) = f(l_2) \Rightarrow (l_1 \in S \Leftrightarrow l_2 \in S)$$

Note that reverse compatibility with any set is trivially satisfied by injective transformations.

2.7.1 Basic transformations

Given two label structures LS_1 and LS_2, we define label structure transformations which are used to embed a label into a sum of labels, destruct a label sum, and extend a label to a pair of labels. These transformations will be used to define timed automata semantics in Sect. 5.2: user defined labels will be extended by memory access and time information, which requires the embedding of the original label in the larger structure. After composition with a controller, supplementary information is removed using projections.

Extensions (as well as projections) are not defined as done usually: they are partial and only defined when only one of the elements of the product is present. These transformations are given in the following table:

Name	Notation	Signature	Definition	Rev. compo.
		Identity		
id	$\mathbb{1}$	$LS \Rrightarrow LS$	$l \mapsto l$	yes
		Embedding		
In_l	$\uparrow^l$	$LS_1 \Rrightarrow LS_1 \oplus LS_2$	$l \mapsto l^\bullet$	yes
In_r	$\uparrow^r$	$LS_2 \Rrightarrow LS_1 \oplus LS_2$	$l \mapsto {}^\bullet l$	yes
		Retraction		
Out_l	$\downarrow^l$	$LS_1 \oplus LS_2 \Rrightarrow LS_1$	$l^\bullet \mapsto l$	yes
Out_r	$\downarrow^r$	$LS_1 \oplus LS_2 \Rrightarrow LS_2$	${}^\bullet l \mapsto l$	yes
		Extension		
Ext_l	$\uparrow^{l\tau}$	$LS_1 \Rrightarrow LS_1 \otimes LS_2$	$l \mapsto (l, \tau)$	yes
Ext_r	$\uparrow^{\tau r}$	$LS_2 \Rrightarrow LS_1 \otimes LS_2$	$l \mapsto (\tau, l)$	yes
		Projection		
Prj_l	$\downarrow^{l\tau}$	$LS_1 \otimes LS_2 \Rrightarrow LS_1$	$(l, \tau) \mapsto l$	no
Prj_r	$\downarrow^{\tau r}$	$LS_1 \otimes LS_2 \Rrightarrow LS_2$	$(\tau, l) \mapsto l$	no
		Option		
Opt	$\uparrow^\perp$	$LS \Rrightarrow LS^\tau$	$l \mapsto l$	yes
Opt^{-1}	$\downarrow^\perp$	$LS^\tau \Rrightarrow LS$	$l \mapsto l$ **if** $l \in LS$	yes

where τ is supposed to be a neutral element of LS_1 or LS_2, supposed to be unital.

It is not difficult to see that the transformation properties (see Definition 8) are satisfied by the transformations we have defined. Note that all these transformations are injective over their respective domain. This property also holds for the projections because they are only defined if one projection of the couple is τ. All these transformations, except projections, are also reverse composable. In order to convince the reader we pick the left retraction and show that it is a transformation.

Proof Given two label structures LS_1 and LS_2, we consider the operator $o \triangleq \downarrow^l$ of signature $LS_1 \oplus LS_2 \Rrightarrow LS_1$ and $l, l' \in \mathbf{dom}(o)$ such that $l \bowtie_{LS_1 \oplus LS_2} l'$ is defined. By the definition of the sum of label structures, l and l' must both be left embeddings of labels of LS_1 : $l = {l_1}^\bullet$ and $l' = {l'_1}^\bullet$, and $l_1 \bowtie_1 l'_1$ is defined. We must show the three properties of transformations:

- $l \bowtie_{LS_1 \oplus LS_2} l' = (l_1 \bowtie_1 l'_1)^\bullet \in \mathbf{dom}(o)$.
- $l_1 \bowtie_1 l'_1$ is defined.
- $o(l \bowtie_{LS_1 \oplus LS_2} l') = o(l) \bowtie_1 o(l')$: they are both equal to $l_1 \bowtie_1 l'_1$. □

Since the three properties are satisfied, the left retraction is a label structure transformation.

2.7.2 Adding communication

This transformation gives a *send/receive marker* to a label structure. Thus, for each label c, the transformed label structure has the two complementary labels $c!$ (sending mark) and $c?$ (receiving mark). Given the label structure $\langle L, \bowtie \rangle$, we denote by $L_!$ (resp. $L_?$) the set of L labels suffixed by the send (resp. receive) marker.

The intuition behind the proposed definition is that the server performs at least the actions requested by a client and that multiple requests originating from several clients can be combined. This mechanism can be used to perform independent memory accesses concurrently. It is illustrated when giving a semantics for Timed Automata (Sect. 5.2).
The communication label structure $\langle L, \bowtie \rangle_{!?}$ is thus defined as:[3]

$$\langle L, \bowtie \rangle_{!?} = \left\langle L_! \cup L_?, \left(\begin{array}{ll} (l_1!, l_2!) & \mapsto (l_1 \bowtie l_2)! \textbf{ if } (l_1, l_2) \in \textbf{dom}(\bowtie) \\ (l_1!, l_2?) & \mapsto l_2? \textbf{ if } l_1 \leq l_2 \\ (l_1?, l_2!) & \mapsto l_1? \textbf{ if } l_2 \leq l_1 \\ (l?, l?) & \mapsto l? \end{array} \right) \right\rangle$$

Remark 1 Other semantics can be given to the product such as a broadcast semantics. The corresponding fusion operator would propagate the *send* annotation:

$$\begin{array}{l} (l_1!, l_2!) \mapsto (l_1 \bowtie l_2)! \textbf{ if } (l_1, l_2) \in \textbf{dom}(\bowtie) \\ (l_1!, l_2?) \mapsto (l_1 \bowtie l_2)! \textbf{ if } (l_1, l_2) \in \textbf{dom}(\bowtie) \end{array}$$

Proposition 3 (Preservation of the ***ACI*** property) *Given an **ACI** label structure LS, its transformation $LS_{!?}$ is also **ACI**. More precisely, commutativity and idempotence are preserved. Associativity is preserved if LS is commutative.*

Associativity is not straightforward. About 40 cases are generated by the proof assistant, depending on the send/receive nature of labels. Furthermore, commutativity (present in the ***ACI*** hypothesis) must be assumed and properties of the lattice structure associated to the $\leq$ relation are also used.

New transformations can now be defined, which add or remove send or receive annotations to labels. Removing communications is defined as partial to ensure the satisfaction of transformation properties. Furthermore, for the $rRecv$ operator, we suppose that LS is idempotent.

Name	Notation	Signature	Precondition	Definition
		Add Communication		
aSend	$\uparrow^!$	$LS \Rightarrow LS!?$		$l \mapsto l!$
aRecv	$\uparrow^?$	$LS \Rightarrow LS!?$	LS diagonal, idempotent	$l \mapsto l?$
		Remove Communication		
rSend	$\downarrow^!$	$LS!? \Rightarrow LS$		$l! \mapsto l$
rRecv	$\downarrow^?$	$LS!? \Rightarrow LS$	LS idempotent	$l? \mapsto l$

In the following, we use the fact that $aSend$ satisfies injectivity, domain stability and reverse composability.

[3] $l \leq l'$ is defined by $l \leq l' \triangleq (l, l') \in \textbf{dom}(\bowtie) \wedge l \bowtie l' = l'$ in Sect. 2.4.

2.7.3 Composition of transformations

We define here three composition operators over label structure transformations:

Op	Signature	Resulting transformation
$\_ \circ \_$	$(tr_1 : LS_1 \Rrightarrow LS_2)$ $(tr_2 : LS_2 \Rrightarrow LS_3)$ $\rightarrow$ $LS_1 \Rrightarrow LS_3$	$l \mapsto tr_2(tr_1(l))$ **if** $\begin{array}{l} l \in \mathbf{dom}(tr_1) \wedge \\ tr_1(l) \in \mathbf{dom}(tr_2) \end{array}$
$\_ \oplus \_$	$(tr_1 : LS_1 \Rrightarrow LS'_1)$ $(tr_2 : LS_2 \Rrightarrow LS'_2)$ $\rightarrow$ $LS_1 \oplus LS_2 \Rrightarrow LS'_1 \oplus LS'_2$	$\begin{cases} l_1{}^\bullet \mapsto tr_1(l_1)^\bullet \textbf{ if } l_1 \in \mathbf{dom}(tr_1) \\ {}^\bullet l_2 \mapsto {}^\bullet tr_2(l_2) \textbf{ if } l_2 \in \mathbf{dom}(tr_2) \end{cases}$
$\_ \otimes \_$	$(tr_1 : LS_1 \Rrightarrow LS'_1)$ $(tr_2 : LS_2 \Rrightarrow LS'_2)$ $\rightarrow$ $LS_1 \otimes LS_2 \Rrightarrow LS'_1 \otimes LS'_2$	$(l_1, l_2) \mapsto (tr_1(l_1), tr_2(l_2))$ **if** $\begin{array}{l} l_1 \in \mathbf{dom}(tr_1) \wedge \\ l_2 \in \mathbf{dom}(tr_2) \end{array}$

We note here that the transformation properties are satisfied by the resulting functions. Furthermore, these composition operators also preserve the injectivity of their arguments.

Proposition 4 (Property preservation by composition). *Stability (see Definition 7) of the transformation domain, injectivity and reverse composability (see Definition 9) are preserved by the three composition operators.*

As an example, we sketch the proof of the preservation of the stability of the transformation domain by the $\_ \oplus \_$ high level transformation.

Proof Given two transformations $tr_1 : LS_1 \Rrightarrow LS'_1$ and $tr_2 : LS_2 \Rrightarrow LS'_2$ having stable domains, we show that the domain of $tr_1 \oplus tr_2$ is stable. We need to prove left and right p-stability at one hand and e-stability at the other. We only consider the left case:

- (left p-stability): We need to prove that $l \in \mathbf{dom}(tr_1 \oplus tr_2) \downarrow_1$: there exists $l' \in \mathbf{dom}(tr_1 \oplus tr_2)$ such that $l \bowtie_{LS1_\oplus LS_2} l'$ is defined and belongs to $\mathbf{dom}(tr_1 \oplus tr_2)$. We must show that $l \in \mathbf{dom}(tr_1 \oplus tr_2)$. As $l \bowtie_{LS1_\oplus LS_2} l'$ is defined, l and l' are either both a left embedding or both a right embedding. The two cases are symmetric. So we consider the left case: $l = l_1{}^\bullet$ and $l' = l'_1{}^\bullet$. As $l \bowtie_{LS_1 \oplus LS_2} l' = l_1 \bowtie_{LS_1} l'_1{}^\bullet \in \mathbf{dom}(tr_1 \oplus tr_2)$, $l_1 \bowtie_{LS_1} l'_1 \in \mathbf{dom}(tr_1)$. As the domain of tr_1 is p-stable, $l_1 \in \mathbf{dom}(tr_1)$. Thus, $l \in \mathbf{dom}(tr_1 \oplus tr_2)$.
- (left e-stability): We need to prove that $l \in \mathbf{dom}(tr_1 \oplus tr_2) \uparrow_1$: there exists $l' \in \mathbf{dom}(tr_1 \oplus tr_2)$ and l'' such that $l' \bowtie_{LS1_\oplus LS_2} l''$ is defined and $l = l' \bowtie_{LS1_\oplus LS_2} l''$. We must show that $l \in \mathbf{dom}(tr_1 \oplus tr_2)$. As $l' \bowtie_{LS1_\oplus LS_2} l''$ is defined, l' and l'' are either both a left embedding or both a right embedding. The two cases are symmetric. So we consider the left case: $l' = l'_1{}^\bullet$ and $l'' = l''_1{}^\bullet$. As $l'_1 \in \mathbf{dom}(tr_1)$ which is e-stable, we have $l'_1 \bowtie_{LS_1} l''_1 \in \mathbf{dom}(tr_1)$. Thus, $l \in \mathbf{dom}(tr_1 \oplus tr_2)$.

□

3 Behavioral framework

This section introduces Labelled Transition Systems (LTS) associated to a given label structure, which can be seen as the type (in λ-calculus terms) of the LTS family. Then, we define

operations on LTS such as parallel composition and renaming by application of label transformations. These definitions, together with related properties, form the behavioral framework that is built on top of the label structure framework.

3.1 Labelled transition systems (LTS)

In this paragraph, we introduce labelled transition systems and their properties. After recalling the usual definition of a bisimulation relation which is an equivalence relation used to compare two LTS, we introduce the so-called *upto-bisimulation* which allows a dynamic renaming that operates depending on the required label.

Definition 11 (*Labelled Transition System*) Given a label structure $LS = \langle L, \bowtie \rangle$, a labelled transition system $\mathcal{L}$ over LS is defined as a tuple $\langle Q, Q^0 \subseteq Q, T \subseteq Q \times L \times Q \rangle$ where Q, Q^0, T denote respectively the sets of states, initial states, and transitions. We denote by LTS_{LS} the set of LTSs over LS. We write $q \xrightarrow{l} q'$ for an element (q, l, q') of T. Furthermore, we define the alphabet of $\mathcal{L} \in LTS_{LS}$ – denoted as $\alpha\mathcal{L}$ – as the set of labels that are actually used by the transitions of $\mathcal{L}$: $\alpha\mathcal{L} = \{l \in L \mid \exists q, q' \cdot \ q \xrightarrow{l} q' \in T\}$.

Definition 12 *(LTS Diagonality, Idempotence and Determinism)* A LTS is said to be diagonal (resp. idempotent) if the restriction of its label structure to the LTS alphabet is diagonal (resp. idempotent, see Sect. 2.4). A LTS is said to be deterministic if whenever $q \xrightarrow{l} q'$ and $q \xrightarrow{l} q''$ then $q' = q''$. In the rest of the paper, this set of LTS properties will be named ***DID***.

Definition 13 *(Simulation)* Given $\mathcal{L}_a = \langle Q_a, Q_a^0, T_a \rangle$ and $\mathcal{L}_c = \langle Q_c, Q_c^0, T_c \rangle$, a relation $R \subseteq Q_c \times Q_a$ defines a simulation between $\mathcal{L}_c$ and $\mathcal{L}_a$, which is denoted as $\mathcal{L}_c \precsim_R \mathcal{L}_a$ iff:

1. $\forall q_c^0 \in Q_c^0 \quad \exists q_a^0 \in Q_a^0$ such that $(q_c^0, q_a^0) \in R$
2. $\forall q_c, q_c', q_a, l$ if $q_c \xrightarrow{l} q_c'$ and $(q_c, q_a) \in R$, $\exists q_a' \in Q_a$ such that $q_a \xrightarrow{l} q_a'$ and $(q_c', q_a') \in R$.

Two LTSs $\mathcal{L}$ and $\mathcal{L}'$ are said to be *bisimilar* through the relation $R \subseteq Q \times Q'$, denoted as $\mathcal{L} \simeq_R \mathcal{L}'$, if $\mathcal{L} \precsim_R \mathcal{L}'$ and $\mathcal{L}' \precsim_{R^{-1}} \mathcal{L}$. We simply write $\mathcal{L} \simeq \mathcal{L}'$ if $\mathcal{L} \simeq_R \mathcal{L}'$ for some R. Furthermore, we say that $\mathcal{L}$ and $\mathcal{L}'$ are state-bisimilar if transition labels are not required to match: in condition (2.), we do not require abstract and concrete labels to be identical (here to l).

However, some behavioral equivalence results presented in this paper (see Sect. 5.2.5) relate transition systems with non-matching label elements. They are related by an $\leq$ relation between possibly under-specified memory access orders. So, we introduce simulation-upto which declares explicitly the relation between labels. Upto-simulation is frequently used in development environments such as Event-B [1] where it is possible to rename events and to declare an arbitrary relation between event parameters during the refinement process.

Definition 14 (*Simulation-upto*) Given $\mathcal{L}_a = \langle Q_a, Q_a^0, T_a \rangle$ and $\mathcal{L}_c = \langle Q_c, Q_c^0, T_c \rangle$, the relations $R \subseteq Q_c \times Q_a$ and $\overrightarrow{R} \subseteq L \times L$ define a simulation upto between $\mathcal{L}_c$ and $\mathcal{L}_a$, which is denoted as $\mathcal{L}_c \precsim_R^{\overrightarrow{R}} \mathcal{L}_a$ iff:

1. $\forall q_c^0 \in Q_c^0 \quad \exists q_a^0 \in Q_a^0$ such that $(q_c^0, q_a^0) \in R$
2. $\forall q_c, l_c, q_c', q_a$ if $q_c \xrightarrow{l_c} q_c'$ and $(q_c, q_a) \in R$, $\exists l_a \in L, q_a' \in Q_a$ such that $q_a \xrightarrow{l_a} q_a'$ and $(q_c', q_a') \in R$ and $(l_c, l_a) \in \overrightarrow{R}$.

As for bisimulation, we introduce a notation for upto-bisimulation by $\mathcal{L} \simeq_R^{\overrightarrow{R}} \mathcal{L}'$.

3.1.1 LTS composition

Given two LTSs defined over $\langle L, \bowtie \rangle$, a subset $S \subseteq L$ denoting the allowed synchronization results, and two subsets, $A_l \subseteq L$ and $A_r \subseteq L$ denoting respectively the left and right interleaving labels, the label composition function $\bowtie$ is extended to a *LTS composition function* ${}^{A_l}_{S}\langle \bowtie \rangle^{A_r}$ as follows:

$$\langle Q_1, Q_1^0, T_1 \rangle \; {}^{A_l}_{S}\langle \bowtie \rangle^{A_r} \; \langle Q_2, Q_2^0, T_2 \rangle = \langle Q_1 \times Q_2, Q_1^0 \times Q_2^0, T \rangle$$

where the set T is defined by the following rules:

$$\frac{q_1 \xrightarrow{l_1}_{T_1} q_1' \quad q_2 \xrightarrow{l_2}_{T_2} q_2' \quad (l_1, l_2) \in \mathbf{dom}(\bowtie) \wedge (l_1 \bowtie l_2) \in S}{(q_1, q_2) \xrightarrow{l_1 \bowtie l_2}_{T} (q_1', q_2')} \textsc{Sync}$$

$$\frac{q_1 \xrightarrow{l_1}_{T_1} q_1' \quad l_1 \in A_l}{(q_1, q_2) \xrightarrow{l_1}_{T} (q_1', q_2)} \textsc{Interleaving}_L \qquad \frac{q_2 \xrightarrow{l_2}_{T_2} q_2' \quad l_2 \in A_r}{(q_1, q_2) \xrightarrow{l_2}_{T} (q_1, q_2')} \textsc{Interleaving}_R$$

S can be omitted when it is equal to L. $\langle \bowtie \rangle$ is the fully synchronous composition operator. We also introduce the $\langle \bowtie \rangle_S$ notation to designate ${}^{S^c}_{S}\langle \bowtie \rangle^{S^c}$ where transitions with labels not in the synchronisation set are made asynchronously.

We can cite two specializations of the LTS composition function:

- CSP generalized parallel operator. $P_1 \; {}_A\|_B \; P_2 \triangleq P_1 \, {}^{A \setminus B}_{A \cap B}\langle \bowtie \rangle^{B \setminus A} P_2$
- CSP interface parallel operator. $P_1 \|_S P_2 \triangleq P_1 \, {}^{S^c}_{S}\langle \bowtie \rangle^{S^c} P_2$
- CCS parallel operator. $P_1 \mid P_2 \triangleq P_1 \, {}^{L}_{\{\tau\}}\langle \bowtie \rangle^{L} P_2$ where C is the set of channels and $L = C \cup \overline{C} \cup \{\tau\}$.

Remark 2 (Internal product) We could also define an internal LTS product inspired by separation logic [24]: as we have attached a structure to labels, we could attach a structure to states and allow state decomposition. The product rule could then be rewritten as follows, where the $\bowtie_s$ and $\bowtie_l$ operators are defined respectively on states and labels:

$$\frac{q_1 \xrightarrow{l_1}_{T_1} q_1' \;\; q_2 \xrightarrow{l_2}_{T_2} q_2' \;\; (q_1, q_2) \in \mathbf{dom}(\bowtie_s) \;\; (l_1, l_2) \in \mathbf{dom}(\bowtie_l) \;\; (q_1', q_2') \in \mathbf{dom}(\bowtie_s)}{q_1 \bowtie_s q_2 \xrightarrow{l_1 \bowtie_l l_2}_{T} q_1' \bowtie_s q_2'} \textsc{ISync}$$

Theorem 1 (Bisimulation Compatibility) *Given four LTSs $\mathcal{L}_1$, $\mathcal{L}_1'$, $\mathcal{L}_2$, and $\mathcal{L}_2'$ defined over the same label structure LS, if $\mathcal{L}_1$ and $\mathcal{L}_2$ are bisimilar upto R and $\mathcal{L}_1'$ and $\mathcal{L}_2'$ are bisimilar upto R' then their respective parallel compositions are bisimilar upto the relational product of R and R' (defined in Paragraph 2.1):*

$$\mathcal{L}_1 \simeq_R \mathcal{L}_2 \wedge \mathcal{L}_1' \simeq_{R'} \mathcal{L}_2' \Rightarrow \mathcal{L}_1 \, {}^{A_l}_{S}\langle \bowtie \rangle^{A_r} \, \mathcal{L}_1' \simeq_{R \otimes R'} \mathcal{L}_2 \, {}^{A_l}_{S}\langle \bowtie \rangle^{A_r} \, \mathcal{L}_2'$$

This theorem states that bisimulation is a congruence for the LTS product. It allows us to substitute a LTS by a bisimilar one. It is mainly used to establish the compositionality of the

semantics of timed automata (see Sect. E in the Appendix). It is in fact an instance of the general theorem of [37] which sets a generic framework to prove congruence theorems for various LTS composition operators.

Proof Suppose $\mathcal{L}_1 \simeq_R \mathcal{L}_2$ and $\mathcal{L}'_1 \simeq_{R'} \mathcal{L}'_2$.
We need to prove $\mathcal{L}_1 \overset{A_l}{\underset{S}{\langle\bowtie\rangle}}{}^{A_r} \mathcal{L}'_1 \simeq_{R\otimes R'} \mathcal{L}_2 \overset{A_l}{\underset{S}{\langle\bowtie\rangle}}{}^{A_r} \mathcal{L}'_2$ which can be divided into two simulation relations: $\mathcal{L}_1 \overset{A_l}{\underset{S}{\langle\bowtie\rangle}}{}^{A_r} \mathcal{L}'_1 \precsim_{R\otimes R'} \mathcal{L}_2 \overset{A_l}{\underset{S}{\langle\bowtie\rangle}}{}^{A_r} \mathcal{L}'_2$ (1) and $\mathcal{L}_2 \overset{A_l}{\underset{S}{\langle\bowtie\rangle}}{}^{A_r} \mathcal{L}'_2 \precsim_{R'\otimes R} \mathcal{L}_1 \overset{A_l}{\underset{S}{\langle\bowtie\rangle}}{}^{A_r} \mathcal{L}'_1$ (2). In the following we will only show the proof of (1). The proof for the other direction (2) is similar.

First, we note that the initial case is trivial. With respect to the inductive case, take a transition $(q_1, q'_1) \xrightarrow{l} (r_1, r'_1)$ of $\mathcal{L}_1 \overset{A_l}{\underset{S}{\langle\bowtie\rangle}}{}^{A_r} \mathcal{L}'_1$ and states (q_2, q'_2) of $\mathcal{L}_2 \overset{A_l}{\underset{S}{\langle\bowtie\rangle}}{}^{A_r} \mathcal{L}'_2$ such that $(R \otimes R')((q_1, q'_1), (q_2, q'_2))$. Given the LTS product rules, we have three cases:

- left case: $l \in A_l$ and $(q_1 \xrightarrow{l} r_1)$ is a transition of $\mathcal{L}_1$ and $q'_1 = r'_1$. As $\mathcal{L}_1 \precsim_R \mathcal{L}_2$ and $R(q_1, q_2)$, there exists r_2 such that $R(r_1, r_2)$ and $(q_2 \xrightarrow{l} r_2)$ in $\mathcal{L}_2$. Thus, with $r'_2 = r_2$, we have that $(q_2, q'_2) \xrightarrow{l} (r_2, r'_2)$ is a transition of $\mathcal{L}_2 \overset{A_l}{\underset{S}{\langle\bowtie\rangle}}{}^{A_r} \mathcal{L}'_2$ with $(R \otimes R')((r_1, r'_1), (r_2, r'_2))$.
- right case: it is the symmetric case.
- synchronous case: $l \in S$ can be written $l = l_1 \bowtie l_2$ with $(q_1 \xrightarrow{l_1} r_1)$ a transition of $\mathcal{L}_1$ and $(q'_1 \xrightarrow{l_2} r'_1)$ a transition of $\mathcal{L}'_1$. As $\mathcal{L}_1 \precsim_{R_1} \mathcal{L}_2$ and $R(q_1, q_2)$, there exists r_2 such that $R(r_1, r_2)$ and $(q_2 \xrightarrow{l_1} r_2)$ in $\mathcal{L}_2$. As $\mathcal{L}'_1 \precsim_{R'} \mathcal{L}'_2$ and $R'(q'_1, q'_2)$, there exists r'_2 such that $R'(r'_1, r'_2)$ and $(q'_2 \xrightarrow{l_2} r'_2)$ in $\mathcal{L}'_2$. We can rebuild the synchronous product of the two transitions to get $(q_2, q'_2) \xrightarrow{l_1 \otimes l_2} (r_2, r'_2)$ which is a transition of $\mathcal{L}_2 \overset{A_l}{\underset{S}{\langle\bowtie\rangle}}{}^{A_r} \mathcal{L}'_2$ with $(R \otimes R')((r_1, r'_1), (r_2, r'_2))$. □

The next result concerns the synchronous composition of LTS where transitions of the product are the transitions that can be synchronously performed by the two LTS. The synchronization set can be restricted to labels belonging to the alphabet of both LTS if the stability condition holds.

Proposition 5 (Synchronous Composition) *Given two LTSs $\mathcal{L}_1$ and $\mathcal{L}_2$ defined over the same label structure LS, we have $\mathcal{L}_1\langle\bowtie\rangle\mathcal{L}_2 \simeq \mathcal{L}_1 \underset{S}{\langle\bowtie\rangle} \mathcal{L}_2$ if S is stable over LS, $\alpha\mathcal{L}_1 \subseteq S$, and $\alpha\mathcal{L}_2 \subseteq S$.*

Proof We show that the two LTS are bisimilar for the identity relation, which makes the initial case trivial. We now consider the inductive case:

- Suppose we have $s \longrightarrow_l s'$ in $\mathcal{L}_1\langle\bowtie\rangle\mathcal{L}_2$. The composition is purely synchronous. Thus, there exists $s_1, s_2, s'_1, s'_2, l_1$ and l_2 such that $l = l_1 \bowtie l_2$, $s = (s_1, s_2)$, $s' = (s'_1, s'_2)$, $s_1 \longrightarrow_{l_1} s_2$ in $\mathcal{L}_1$ and $s_1 \longrightarrow_{l_2} s_2$ in $\mathcal{L}_2$. As $\alpha\mathcal{L}_i \subseteq S$, we have $l_i \in S$. As S is stable over LS, we have $l_1 \bowtie l_2 \in S$. So, we get a synchronous transition for $\mathcal{L}_1 \underset{S}{\langle\bowtie\rangle} \mathcal{L}_2$.
- Suppose we have $s \longrightarrow_l s'$ in $\mathcal{L}_1 \underset{S}{\langle\bowtie\rangle} \mathcal{L}_2$. Three cases should be considered. However, asynchronous transitions are impossible as they should be labelled by elements of $L \setminus S$ which do not exists in $\mathcal{L}_1$ and $\mathcal{L}_2$. Remains the synchronous case. The corresponding transition is a transition for $\mathcal{L}_1\langle\bowtie\rangle\mathcal{L}_2$. □

Commutativity and associativity of the composition operator can now be stated. They are defined modulo the bisimulation relation.

Proposition 6 (Commutativity of ${}^{A_l}\langle\bowtie\rangle^{A_r}_{S}$) *Given two LTSs $\mathcal{L}_1$ and $\mathcal{L}_2$ defined over the same label structure LS, we have $\mathcal{L}_1 \,{}^{A_l}\langle\bowtie\rangle^{A_r}_{S}\, \mathcal{L}_2 \simeq \mathcal{L}_2 \,{}^{A_r}\langle\bowtie\rangle^{A_l}_{S}\, \mathcal{L}_1$ if LS is commutative.*

Proof First, we introduce the refinement relation: it maps (q_1, q_2) to (q_2, q_1), which makes trivial the initial case. The two directions for proving bisimilarity being symmetric, we only consider one of them. A transition of $\mathcal{L}_1 \,{}^{A_l}\langle\bowtie\rangle^{A_r}_{S}\, \mathcal{L}_2$ is either asynchronous or synchronous. In the asynchronous case, we have a transition of $\mathcal{L}_1$ with a label in A_l, or a transition of $\mathcal{L}_2$ with a label in A_r. They are present in $\mathcal{L}_2 \,{}^{A_r}\langle\bowtie\rangle^{A_l}_{S}\, \mathcal{L}_1$. In the synchronous case, we have a transition with a label $l_1 \bowtie l_2 \in S$ with $l_i \in \mathcal{L}_i$. As $l_1 \bowtie l_2 = l_2 \bowtie l_1$, we get a transition of $\mathcal{L}_2 \,{}^{A_r}\langle\bowtie\rangle^{A_l}_{S}\, \mathcal{L}_1$. □

In Theorem 2, we state a general associativity result. We are aware that the theorem has numerous conditions and is not easy to use in practice. That is why we give weakened variants of the theorem with much simpler and reduced conditions.

Theorem 2 (Associativity of ${}^{A_l}\langle\bowtie\rangle^{A_r}_{S}$) *Given a label structure $LS = \langle L, \bowtie\rangle$, label sets A_{l_1},A_{r_1}, A_{l_2}, A_{r_2}, S_1, $S_2 \subseteq L$, and LTSs $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$ over LS. If LS is associative and A'_{l_1}, S'_1, A'_{r_1}, A'_{l_2}, S'_2, A'_{r_2} are such that the following conditions are satisfied:*

$$
\begin{array}{ccc}
S_1 \cap (S_2 \cap \mathbf{ran}(\bowtie))\uparrow^2 \subseteq S'_2 & S_1 \cap A_{r_2}\uparrow^2 \subseteq S'_2 & S'_2 \cap (S'_1 \cap \mathbf{ran}(\bowtie))\uparrow^1 \subseteq S_1 \\
S'_2 \cap A'_{l_1}\uparrow^1 \subseteq S_1 & (S_1 \cap (S_2 \cap \mathbf{ran}(\bowtie))\uparrow^2)\downarrow_1 \subseteq S'_1 & S_1 \cap A_{l_2}\uparrow^2 \subseteq S'_1 \cap A'_{l_2} \\
(S_1 \cap A_{r_2}\uparrow^2)\downarrow_1 \subseteq A'_{l_1} & S_2 \cap \mathbf{ran}(\bowtie) \cap A_{r_1} \subseteq S'_2 & (S_2 \cap A_{r_1})\downarrow_1 \subseteq A'_{r_1} \\
(S'_2 \cap (S'_1 \cap \mathbf{ran}(\bowtie))\uparrow^1)\downarrow_2 \subseteq S_2 & (S'_2 \cap A'_{l_1}\uparrow^1)\downarrow_2 \subseteq A_{r_2} & S'_2 \cap A'_{r_1}\uparrow^1 \subseteq S_2 \cap A_{r_1} \\
S'_1 \cap \mathbf{ran}(\bowtie) \cap A'_{l_2} \subseteq S_1 & (S'_1 \cap A'_{l_2})\downarrow_2 \subseteq A_{l_2} & A'_{r_2} = A_{r_1} \cap A_{r_2} \\
A_{l_1} = A'_{l_2} \cap A'_{l_1} & A_{r_1} \cap A_{l_2} = A'_{r_1} \cap A'_{l_2} &
\end{array}
$$

then we have:

$$\mathcal{L}_1 \,{}^{A_{l_1}}\langle\bowtie\rangle^{A_{r_1}}_{S_1}\, (\mathcal{L}_2 \,{}^{A_{l_2}}\langle\bowtie\rangle^{A_{r_2}}_{S_2}\, \mathcal{L}_3) \;\simeq\; (\mathcal{L}_1 \,{}^{A'_{l_1}}\langle\bowtie\rangle^{A'_{r_1}}_{S'_1}\, \mathcal{L}_2) \,{}^{A'_{l_2}}\langle\bowtie\rangle^{A'_{r_2}}_{S'_2}\, \mathcal{L}_3$$

The Coq script stating this property can be found in Appendix D. It is not difficult by itself but rather long: about 120 proof steps (mainly hypothesis uses) are needed.

Proof First, we introduce the refinement relation: it maps tuples $(s_1, (s_2, s_3))$ to $((s_1, s_2), s_3)$, which makes trivial the initial case. We now give the proof for one of the numerous cases of the step refinement. Consider a transition $(s_1, (s_2, s_3)) \longrightarrow_l (s'_1, (s'_2, s'_3))$ of $\mathcal{L}_1 \,{}^{A_{l_1}}\langle\bowtie\rangle^{A_{r_1}}_{S_1}\, (\mathcal{L}_2 \,{}^{A_{l_2}}\langle\bowtie\rangle^{A_{r_2}}_{S_2}\, \mathcal{L}_3)$. Among the different ways of obtaining that transition, take the one where we have applied the synchronous rule for ${}^{A_{l_1}}\langle\bowtie\rangle^{A_{r_1}}_{S_1}$ and the right asynchronous rule for ${}^{A_{l_2}}\langle\bowtie\rangle^{A_{r_2}}_{S_2}$, which means: $l = l_1 \bowtie l_3$, $l \in S_1$, $l_3 \in A_{r_2}$ and $\mathcal{L}_1$ and $\mathcal{L}_3$ have made a move. Now we must build a transition of $(\mathcal{L}_1 \,{}^{A'_{l_1}}\langle\bowtie\rangle^{A'_{r_1}}_{S'_1}\, \mathcal{L}_2) \,{}^{A'_{l_2}}\langle\bowtie\rangle^{A'_{r_2}}_{S'_2}\, \mathcal{L}_3$ where $\mathcal{L}_1$ and $\mathcal{L}_3$ move, which means we must apply the left asynchronous rule for ${}^{A'_{l_1}}\langle\bowtie\rangle^{A'_{r_1}}_{S'_1}$ and the synchronous rule

for ${}^{A'_{l_2}}\langle \underset{S'_2}{\bowtie} \rangle^{A'_{r_2}}$: we must have $l_1 \in A'_{l_1}$ and $l \in S'_2$. This is true because $(S_1 \cap A_{r_2}\uparrow^2)\downarrow_1 \subseteq A'_{l_1}$ and $S_1 \cap A_{r_2}\uparrow^2 \subseteq S'_2$. More precisely, l_1 is a right argument (belonging to A_{r_2}) of a product resulting in an element of S_1. Thus, it belongs to $(S_1 \cap A_{r_2}\uparrow^2)\downarrow_1$. In the same way l is an element of S_1 and results from the product of some element with an element of A_{r_2}. □

Remark We could have taken stronger and more readable hypotheses such that $A_{r_2} \subseteq A'_{l_1}$ and $S_1 \subseteq S'_2$ but we have chosen to enlighten the brute proof obligations extracted from the proof.

About associativity proofs. It has to be noted that proving associativity for LTS-based parallel composition is known to be a difficult task. In [11] flawed proofs of invalid results are reported and an associativity proof for a variant of timed automata is detailed and appears to be very tedious: 13 toplevel cases are considered and are technically hard. In the same way, in [18], 25 subcases are considered.

Furthermore, a more advanced study of sufficient conditions for ensuring the associativity of LTS composition operators can be found in [18]. For example, the associativity of the choice and parallel operators of CCS are consequences of their general theorem. However, the framework also has limitations. For example, it does not apply for the CSP parallel operator. Actually, our proofs are ad hoc. Our main concern was to exhibit lemmas associated to the hard or the repeated parts of the proof that can be reused for similar theorems. With respect to such proofs, [18] deals with the shape of the proof tree resulting from the format of the rules. It should however be possible to integrate it into our proposal.

Proposition 7 (Idempotence of $\langle \bowtie \rangle$) *Given a label structure $LS = \langle L, \bowtie \rangle$ and a LTS $\mathcal{L}$ over LS, if $\mathcal{L}$ is **DID** (Diagonal, Idempotent and Deterministic) then $\mathcal{L}\langle \bowtie \rangle \mathcal{L} \simeq \mathcal{L}$.*

Proof Given a LTS $\mathcal{L}$ supposed to be diagonal (H_0), idempotent (H_1) and deterministic (H_2), we prove that $(\mathcal{L}\langle \bowtie \rangle \mathcal{L}) \simeq_R \mathcal{L}$ where $R = \{((q,q),q) \mid q \in Q\}$. For $i \in \{1,2\}$, let $(q,q) \xrightarrow{l} (q'_1, q'_2)$ be a transition of $\mathcal{L}\langle \bowtie \rangle \mathcal{L}$: $(q,q) \xrightarrow{l} (q'_1, q'_2)$ implies $q \xrightarrow{l_i} q'_i, l = l_1 \bowtie l_2$. By H_0 we know that $(l_1, l_2 \in \mathbf{dom}(\bowtie) \Rightarrow l_1 = l_2)$, by H_1, $l_1 \bowtie l_2 = l_1$ and by H_2, $q'_1 = q'_2$. Let $q \xrightarrow{l} q'$ be a transition of $\mathcal{L}_{LS}$, we compose $q \xrightarrow{l} q'$ with itself: $q \xrightarrow{l} q', q \xrightarrow{l} q', l \bowtie l = l$ implies $(q,q) \xrightarrow{l} (q',q')$ since by H_1 we know that $l \bowtie l = l$. □

Remark If we drop the *deterministic* hypothesis, the result is false. Consider a diagonal and idempotent label structure of support $\{l_1, l_2, l_3\}$. Figure 1 describes a transition system T and the synchronous product with itself, which is not bisimilar.

3.1.2 Weak variants of the associativity property

The following corollaries are direct instances of theorem 2.

Corollary 1 (CCS Associativity) *The CCS parallel composition operator is associative.*

CCS parallel composition (defined on the associative commutative label structure $\langle L = C \cup \overline{C} \cup \{\tau\}, c \bowtie \overline{c} \mapsto \tau \rangle$ is obtained as ${}^{L}\langle \underset{\{\tau\}}{\bowtie} \rangle^{L}$. Thus A-sets are the full set and S-sets are the singleton $\{\tau\}$, which makes the conditions of Theorem 2 satisfied.

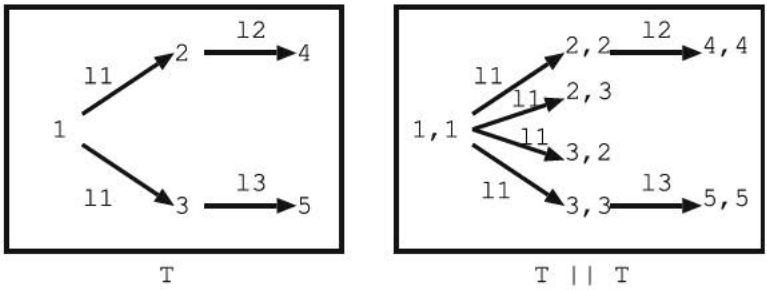

Fig. 1 Counterexample for idempotence

Corollary 2 (CSP Associativity) *The CSP parallel composition operator satisfies the following generalized associativity law stated in* [32]*:*

$$P_1 \ {}_A\|_{B\cup C} \ (P_2 \ {}_B\|_C \ P_3) \simeq (P_1 \ {}_A\|_B \ P_2) \ {}_{A\cup B}\|_C \ P_3$$

Proof CSP parallel composition ${}_A\|_B$ (defined on the associative commutative label structure $\langle C, c \bowtie c \mapsto c\rangle$) is obtained as ${}^{A\setminus B}\langle \underset{A\cap B}{\bowtie} \rangle^{B\setminus A}$. Thus we instantiate our general theorem by taking:

$$\begin{array}{ccc} A_{l_1} = A\setminus(B\cup C) & S_1 = A\cap(B\cup C) & A_{r_1} = (B\cup C)\setminus A \\ A'_{l_1} = A\setminus B & S'_1 = A\cap B & A'_{r_1} = B\setminus A \\ A_{l_2} = B\setminus C & S_2 = B\cap C & A_{r_2} = C\setminus B \\ A'_{l_2} = (A\cup B)\setminus C & S'_2 = (A\cup B)\cap C & A'_{r_2} = C\setminus(A\cup B) \end{array}$$

All the conditions should now be checked. We first note that the $\uparrow$ and $\downarrow$ operators, introduced in Sect. 2.5, are the identity function thanks to the considered label structure. We take the full set for $\mathbf{ran}(\bowtie)$ which only occurs positively[4] and on the left side of the set inclusion operator. It is then easy to prove each condition. The first four ones are considered here:

- $S_1 \cap (S_2 \cap \mathbf{ran}(\bowtie))\uparrow^2 \subseteq S_1 \cap S_2 \uparrow^2 \subseteq S_1 \cap S_2 = A\cap(B\cup C)\cap B\cap C = A\cap B\cap C \subseteq B\cap C = S'_2$
- $S'_2 \cap A_{l_1}\uparrow^1 = (A\cup B)\cap C\cap(A\setminus(B\cup C)) = \emptyset \subseteq S_1$
- $(S_1 \cap A_{r_2}\uparrow^2)\downarrow_1 \subseteq A\cap(B\cup C)\cap C\setminus B = A\cap C\setminus B \subseteq A\setminus B = A'_{l_1}$
- $(S'_2 \cap (S'_1 \cap \mathbf{ran}(\bowtie))\uparrow^1)\downarrow_2 \subseteq (A\cup B)\cap C\cap A\cap B = A\cap B\cap C \subseteq B\cap C = S_2$

□

Corollary 3 (Weakened associativity laws) *Given an associative label structure $LS = \langle L, \bowtie\rangle$, the label set $S \subseteq L$, the LTSs $\mathcal{L}_1, \mathcal{L}_2$, and $\mathcal{L}_3$ over LS such that $\alpha\mathcal{L}_1 \subseteq S_1$ and $\alpha\mathcal{L}_2 \subseteq S_2$, then we have the following results where the product of $\mathcal{L}_2$ and $\mathcal{L}_3$ is either partially or totally synchronous.*

1. $\mathcal{L}_1 \underset{S}{\langle\bowtie\rangle}(\mathcal{L}_2 \underset{S}{\langle\bowtie\rangle}\mathcal{L}_3) \simeq (\mathcal{L}_1 \underset{S}{\langle\bowtie\rangle}\mathcal{L}_2)\underset{S}{\langle\bowtie\rangle}\mathcal{L}_3$ *if S, S^c are stable over LS.*
2. $\mathcal{L}_1 \underset{S}{\langle\bowtie\rangle}(\mathcal{L}_2\langle\bowtie\rangle\mathcal{L}_3) \simeq (\mathcal{L}_1 \underset{S}{\langle\bowtie\rangle}\mathcal{L}_2)\langle\bowtie\rangle\mathcal{L}_3$ *if S is stable over LS and $\alpha\mathcal{L}_1 \subseteq S$.*

[4] not as a sub-expression of the right-hand side of the set-difference operator.

The first statement of corollary 3 is an associativity schema corresponding to the weak associativity theorem of the CSP generalized parallel operator [32], where $\langle \underset{S}{\bowtie} \rangle$ corresponds to the CSP parallel operator $\underset{S}{\|}$. The assumptions added in our context are satisfied by the label structure associated to CSP.

The second statement of corollary 3 is obtained through the lemma stating the following property:

$$\alpha\mathcal{L}_1 \subseteq S \Rightarrow \mathcal{L}_1 \langle \underset{S}{\bowtie} \rangle \mathcal{L}_2 \simeq \mathcal{L}_1 {}^{\emptyset}\langle \underset{S}{\bowtie} \rangle^{S^c} \mathcal{L}_2$$

which means that if labels of the LTS $\mathcal{L}_1$ are in S, it never runs asynchronously.

3.2 LTS transformations

The label structure of a LTS may be changed in a composition such as making local a global event (CSP hide) or changing its name (CSP renaming operator). Here, we consider LTS labels transformations to be a lifting of label structure transformations to LTS. This amounts to rename LTS labels by using the label structure transformation.

Definition 15 (*LTS Transformation*) Given two label structures LS_1 and LS_2, and a label structure transformation $f\colon LS_1 \Rrightarrow LS_2$, we define the LTS transformation $[f]\colon \mathcal{L}_{LS_1} \to \mathcal{L}_{LS_2}$ as:

$$[f]\langle Q, Q_0, \to_1\rangle \triangleq \langle Q, Q_0, \to_2 = \{(q,\ f(l), q') \mid l \in \mathbf{dom}(f) \wedge (q, l, q') \in \to_1\}\rangle$$

Proposition 8 (Compatibility with bisimulation) *Given $LS_1 = \langle L_1, \bowtie_1\rangle$, $LS_2 = \langle L_2, \bowtie_2\rangle$, two LTSs $\mathcal{L}_1$ and $\mathcal{L}_2$ both over LS_1, and a transformation $f\colon LS_1 \Rrightarrow LS_2$, $\mathcal{L}_1 \simeq \mathcal{L}_2 \Rightarrow [f](\mathcal{L}_1) \simeq [f](\mathcal{L}_2)$.*

This result is straightforward but useful to allow reasoning by equivalence rewriting.

The following theorem states sufficient conditions for a LTS transformation to commute with parallel composition. Two sets of conditions are given: one which only depends on the transformation and the other which depends on labels used by the transition systems.

Theorem 3 (Transformation Compositionality) *Given $LS_1 = \langle L_1, \bowtie_1\rangle$, $LS_2 = \langle L_2, \bowtie_2\rangle$, two LTSs $\mathcal{L}_1$ and $\mathcal{L}_2$ both over LS_1, A_l, A_r, S subsets of L_1, and a transformation $f\colon LS_1 \Rrightarrow LS_2$ such that:*

1. *f is reverse compatible with the sets A_l, S and A_r,*
2. *either $\mathbf{dom}(f)$ is stable over LS_1, or $\alpha\mathcal{L}_1 \cup \alpha\mathcal{L}_2 \subseteq \mathbf{dom}(f)$.*

then we have:

$$[f](\mathcal{L}_1 {}^{A_l}\langle \underset{S}{\bowtie} \rangle^{A_r} \mathcal{L}_2) \simeq [f](\mathcal{L}_1)\, {}^{f(A_l)}\langle \underset{f(S)}{\bowtie} \rangle^{f(A_r)} [f](\mathcal{L}_2)$$

Proof We only sketch the proof of the synchronous case. Given LS_1, LS_2, $\mathcal{L}_{LS_1}$, and $\mathcal{L}'_{LS_1}$, we prove that the two sides are bisimilar through the identity relation. The proof is based on showing that each transition of the first system can be found in the second system and vice-versa. It is depicted in the following implications which can be read from bottom to top and vice-versa, and from left to right and vice-versa. The main points of this proof are first the use of the stability hypothesis so that we conclude that when $l \in \mathbf{dom}(f)$ then

$l_1, l_2 \in \mathbf{dom}(f)$ and conversely, second the use of the injectivity of f in order to connect the two branches of the proof.

$$\left(\dfrac{\dfrac{q_1 \xrightarrow{l_1} q_1', q_2 \xrightarrow{l_2} q_2', l = l_1 \bowtie l_2}{(q_1,q_2) \xrightarrow{l} (q_1',q_2'), l \in \mathbf{dom}(f)}}{(q_1,q_2) \xrightarrow{f(l)} (q_1',q_2')} \quad \dfrac{\dfrac{q_1 \xrightarrow{l_1} q_1', l_1 \in \mathbf{dom}(f)}{q_1 \xrightarrow{f(l_1)} q_1'} \quad \dfrac{q_2 \xrightarrow{l_2} q_2', l_2 \in \mathbf{dom}(f)}{q_2 \xrightarrow{f(l_2)} q_2'} \quad l = l_1 \bowtie l_2}{(q_1,q_2) \xrightarrow{f(l)=f(l_1)\bowtie f(l_2)} (q_1',q_2')} \right)$$

□

Other variants of the transformation compositionality can be deduced. They are given in the following:

Corollary 4 (Transformation Compositionality: first variant) *Given* $LS_1 = \langle L_1, \bowtie_1 \rangle$, $LS_2 = \langle L_2, \bowtie_2 \rangle$, *two LTSs* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ *both over* LS_1, *and a transformation* $tr : LS_1 \Rrightarrow LS_2$, *we have:*

$$\text{r\_composable}(tr) \wedge \text{stable}(\mathbf{dom}(tr)) \Rightarrow [tr](\mathcal{L}_1\langle\bowtie\rangle\mathcal{L}_2) \simeq [tr](\mathcal{L}_1)\langle\bowtie\rangle[tr](\mathcal{L}_2)$$

Corollary 5 (Transformation Compositionality: second variant) *Given* $LS_1 = \langle L_1, \bowtie_1 \rangle$, $LS_2 = \langle L_2, \bowtie_2 \rangle$, *two LTSs* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ *both over* LS_1, *and a transformation* $tr : LS_1 \Rrightarrow LS_2$, *we have:*

$$\begin{array}{l} \mathcal{L}_1 \models \mathbf{dom}(tr) \wedge \mathcal{L}_2 \models \mathbf{dom}(tr) \wedge \text{r\_composable}(tr) \\ \quad \Rightarrow [tr](\mathcal{L}_1\langle\bowtie\rangle\mathcal{L}_2) \simeq [tr](\mathcal{L}_1)\langle\bowtie\rangle[tr](\mathcal{L}_2) \end{array}$$

where $\mathcal{L} \models P$ *means that all the labels of* $\mathcal{L}$ *transitions satisfy the predicate* P.

Proposition 9 (Preservation of ***DID*** properties) *Given two label structures* LS_1 *and* LS_2, *an injective, reverse composability preserving transformation* $f : LS_1 \Rrightarrow LS_2$, *and a LTS* $\mathcal{L}$ *over* LS_1, *each of the* ***DID*** *properties is preserved by the transformation* f. *Formally, for a* ***DID*** *property* P, *we have:* $P(\mathcal{L}) \Rightarrow P([f]\mathcal{L})$.

The following proof sketch makes explicit the use of the different hypothesis for each of the three property preservation results.

Proof Take an injective, reverse composability preserving, transformation $tr : LS_1 \Rrightarrow LS_2$, a LTS $\mathcal{L}$ over LS_1 and two labels $f(l_1)$ and $f(l_2)$ occurring in transitions of $[tr](\mathcal{L})$. Suppose $\mathcal{L}$ has one of the ***DID*** properties.

- (*Preservation of diagonality*) Suppose $(tr(l_1), tr(l_2)) \in \mathbf{dom}(\bowtie_2)$. As f preserves reverse composability, we have $(l_1, l_2) \in \mathbf{dom}(\bowtie_1)$. As LS_1 is diagonal, we have $l_1 = l_2$ and thus $tr(l_1) = tr(l_2)$.
- (*Preservation of idempotence*) As $\mathcal{L}$ is idempotent, $(l_1, l_1) \in \mathbf{dom}(\bowtie_1)$ and $l_1 \bowtie_1 l_1 = l_1$. As tr is a transformation, $(tr(l_1), tr(l_1)) \in \mathbf{dom}(\bowtie_2)$ and $tr(l_1) \bowtie_2 tr(l_1) = tr(l_1 \bowtie_2 l_1) = tr(l_1)$, which means $[tr](\mathcal{L})$ is idempotent.
- (*Preservation of determinism*) Given a state s and two states s' and s'' reachable from s through l' in $[tr](\mathcal{L})$. As tr is injective, there exists a unique l such that $l' = tr(l)$. As $\mathcal{L}$ is deterministic and $s \xrightarrow{l} \{s', s''\}$, we have $s' = s''$. So $[tr](\mathcal{L})$ is deterministic.

□

4 Related work

In the following, we give an overview of work related to label structures: abstract behavior types, structured labels and synchronization algebra. The first one is mainly related to concurrency since it is mostly linked to behavioral aspects. The last two are closer to our work.

4.1 Abstract behavior types

Abstract behavior types (ABT) [5] aim at specifying the interactions of components in order to study their composition. ABT describe the dynamics of a component using a relation between timed data streams flowing through its ports. Composing two ABT consists in computing the join of the two relations. Thus ABT is orthogonal to our proposal as we mainly describe in a structured way a punctual interaction, not the stream of data exchanges. However, a step in this direction will be done in Sect. 6 where we attach a behavior to a label structure.

4.2 Structured labels

Structured Labels, introduced in [8], is a set of labels equipped with a partial order. In the context of modal transition systems, the partial order over labels is used to constrain the refinement between transitions systems: the labels of matching transitions are not supposed to be equal but comparable (as required by the (may or must) modality semantics). As in our case, a product of structured labels is defined, but its support set is the cartesian product of the support sets of two structured labels: the product as a fixed definition. In our proposal, the definition of the product is provided by the user, while a partial order may be derived. The proposals are complementary and could be merged in a future work.

4.3 Synchronization algebra

In [38], the authors introduce a so called *synchronization algebra* as a binary associative commutative operation over labels. It is used to parameterize the product of transition systems as in our case. Furthermore, operations on labels, such as restriction, renaming, and product are lifted to transitions systems. Their composition is rooted in a categorical setting which we do not consider here. Actually, we have adopted the same principles while making some changes to implicit properties of label structures and to their use in LTS composition. We list some of these changes:

- Label structure composition is partial in order to encode a blocking synchronization. [38] uses the symbol **0** as a special element marking undefinedness while we use a predicate to specify its definition domain in the Coq implementation.
- Asynchrony is managed at the *synchronization algebra* level in [38] by the implicit insertion of asynchronous transitions whose label contains the special symbol $\star$. It means that the CSP generalized parallel operator, parameterized by left and right alphabets, is hard to encode. Our label structure only manages the synchronous case.
- Label structures are not by default associative and commutative. These properties are assumed when needed, which allows to point out the hypotheses needed to establish some properties. Other properties are also considered, such as diagonality, idempotence, ….

Contrary to [38], operations such as sum, product, renaming are not defined at the level of labels, but at the level of label structures. It means that we also define the resulting label

product operator. Thus, we get label structure transformations and we reason on the properties they preserve.

Furthermore, we have proved general properties about LTS composition derived from properties of label structures and used them in the next sections linked to timed automata. Moreover, even if the overall idea of the two proposals are close, our approach is quite different by the fact that we have defined several operators on label structures and see them as building blocks to construct complex label structures. Then, we consider LTS to be defined on label structures, not only on labels. Label structures act as a typing information which plays an important role in the next sections, helping in verifying the wellformedness of LTS-based expressions.

5 Timed systems through label structures

In this section, we show how timed systems and their semantics can be represented by instances of Labelled Transition Systems over suitable label structures. Then, we reuse the results we have established on label structures in order to prove the compositionality of timed automata semantics.

5.1 Timed transition systems (TTS)

A timed transition system is a transition system where transitions are labelled either by discrete events or by a duration belonging to a given time domain. Usually some constraints over timed transitions are imposed (additivity, continuity, zero-delay). We do not consider them here. Expressing such properties in a general setting as the one we adopt here would require the introduction of a new operator to our label structure. It would correspond to sequential composition and is left for future work.

Definition 16 (*TTS*) Given a label structure $LS = \langle L, \bowtie \rangle$, a Timed Transition System (TTS) over LS is a LTS over $LS \oplus TS$, where TS is the time label structure introduced in Sect. 2.3.

Remark 3 (*TTS parallel composition*) Thanks to the introduction of our label structure, the TTS composition is the composition of the underlying LTSs.

5.2 Timed automata (TA)

We first consider a definition of timed automata [4] in which no invariants are associated to its locations (this is close to timed graphs [3] since neither invariants nor committed states are modeled). The transitions are in the form of $guard/event/reset$ where the guards contain a conjunction of constraints represented as clock intervals and the reset actions consist in a set of clocks to be reset. This is represented as a product of three label structures. The first one manages the clock guards, the second one manages the synchronization events, and the third one manages the clock resets. In the rest of this paper, we consider a set C of clocks and a time domain Δ (e.g. $\mathbb{R}^+$).

5.2.1 Guard label structure

Based on the Alur Dill timed automata [4], a guard is a conjunction of interval constraints associated to clocks. Here, this is modeled as an abstract function in $C \to 2^{\Delta}$ as we are not concerned with decidability results. The composition of two guards g_1 and g_2 is defined as a

function which associates to each clock the intersection of the time domains given by each guard. The guard label structure is defined as:

$$G \triangleq \langle C \to 2^{\Delta},\ (g_1, g_2) \mapsto (c \mapsto g_1(c) \cap g_2(c)) \rangle$$

Proposition 10 *The guard label structure is **ACI**.*

Proof This property is a direct consequence of the ***ACI*** nature of set intersection. □

5.2.2 Action label structure

Based on the Alur Dill timed automata [4], an action associated to a discrete transition can reset some clocks while keeping the other clocks managed by the current timed automaton unchanged. In order to allow the composition of reset actions, the clocks not managed by a given timed automaton are left undetermined. Consequently an action is modeled by two disjoint sets: r denoting the clocks to be reset and u denoting the clocks to be left unchanged. Their composition is defined by respectively the union of the reset sets and the union of the unchanged sets provided that the reset and the unchanged sets are disjoint.

$$\begin{aligned} A \triangleq \langle\ & \{(r,u) \in 2^C \times 2^C \mid r \cap u = \emptyset\}, \\ & ((r_1,u_1),(r_2,u_2)) \mapsto (r_1 \cup r_2, u_1 \cup u_2) \textbf{ if } r_1 \cap u_2 = r_2 \cap u_1 = \emptyset \rangle \end{aligned}$$

Proposition 11 *The action label structure is **ACI**.*

Proof This property is easily proved by case analysis. □

5.2.3 Timed automata label structure

Given a set of clocks C, we define a timed automaton as a LTS such that its transitions are labelled by synchronization events (defined by some label structure LS), guards (the label structure G) and reset actions (the label structure A). For the time being, LS is left undefined and can either model the CCS-based synchronization or the CSP-based one. In the following, we will denote GA as the product $G \otimes A$.

Definition 17 (*Timed automata*) Given a label structure LS, a timed automaton (TA) over LS is a LTS over $LS \otimes GA$.

$$TA_{LS} \triangleq LTS_{LS \otimes GA} \text{ with } GA \triangleq G \otimes A$$

Remark 4 (***ACI*** *property*) Thanks to the ***ACI***-preservation by the product operator (proposition 2), if LS has the ***ACI*** property, $LS \otimes GA$ has also the ***ACI*** property and thus the timed automaton is over an ***ACI*** label structure.

Remark 5 (TA Composition) Thanks to our label structure, the TA composition is defined as the composition of the underlying LTS systems. In fact, several composition rules have been defined for TA. The one of [4] can be considered as based on CSP: it can be obtained by choosing $LS = Sync_{CSP}$. The one of UPPAAL[5] [10] can be considered as based on CCS: we take $LS = Sync_{CCS}$.

[5] We do not consider here the global variables and clocks can only be reset to 0.

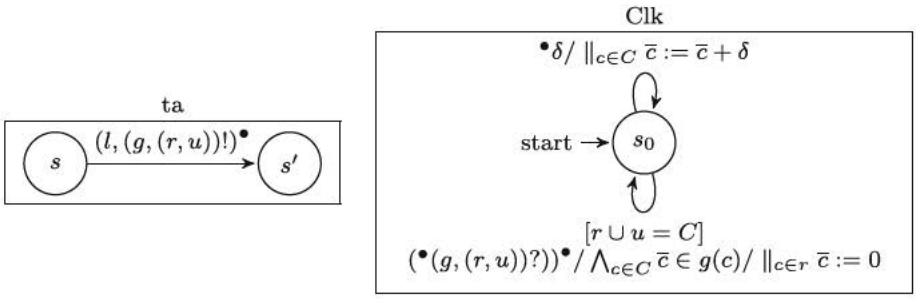

Fig. 2 Semantics of TA via LTS Composition

5.2.4 TA semantics

In this paragraph, we present an alternative semantics for timed automata which will be given in Definition 18. W.r.t. the standard semantics given in Sect. 5.2.5, we define the semantics of a timed automaton as a TTS over $LS \otimes GA_{!?}$. The proposed semantics is a LTS over $LS \otimes GA_{!?} \oplus TS$: these semantics makes explicit time transitions and adds communication with a server which manages clock valuations.

Lemma 1 *If LS is associative (commutative)*[6] *then* $LS \otimes GA_{!?} \oplus TS$ *is associative (commutative).*

Proof G, A, and TS are associative (commutative). By Proposition 2, $\oplus$ and $\otimes$ preserve the associativity (commutativity). By Proposition 3, the !? transformation also preserves these properties. □

The TA semantics is given via its composition with a *clock manager Clk* defined over $GA_{!?} \oplus TS$ (Fig. 2).

Clock Manager. The Clk automaton contains variables (denoted as $\overline{c}$) corresponding to the clocks c of C. It has two types of transitions:

- The first type corresponds to transitions of time evolution labelled by $^\bullet\delta$ which add the delay specified by the transition label to all the clocks.
- The second type corresponds to discrete transitions labelled by any $^\bullet(g,(r,u))$ such that $r \cup u = C$. Clocks are checked against their guard constraints ($\overline{c} \in g(c)$), and clocks belonging to r are reset.

In order to impose determinism of Clk, we suppose that $r \cup u = C$ in Clk, but we synchronize labels l of ta with labels l' of Clk when $l \leq l'$. We recall that this partial order relation $\leq$ has been defined from the join operator in Sect. 2.4.

Proposition 12 (Idempotence of the Clk automaton) *Clk is idempotent:*

$$Clk \ \langle\bowtie\rangle \ Clk \ \simeq \ Clk$$

Proof This property is a consequence of Proposition 7. It requires to establish that the Clk label structure $GA_{!?} \oplus TS$ restricted to the labels of Clk is diagonal, idempotent, and deterministic.

[6] This is verified for CCS and CSP.

– *diagonality*: labels used in Clk are reception labels or time labels. Their product is defined only if they are of the same kind (definition of the $\oplus$ label structure) and identical.
– *idempotence*: the restriction of the product to the labels used by Clk is idempotent: actually, for reception and time, the product is defined and idempotent.
– *determinism*: for each transition of Clk the destination state (location and clock values) is determined by the source state and the label.

□

Reconstructing the TA Semantics. We can now propose our definition of TA semantics by means of a composition between the syntactic ta and Clk where ta transmits the clock commands to Clk. Since the LTS composition is defined over LTS built on identical label structures, the label structures on which ta and Clk are defined have to be adapted so that they both become defined over $LS^\tau \otimes GA!? \oplus TS$: a label can be either a time label or a guard/action marked with send/receive and possibly associated with a label of LS. The following transformations are applied to ta which is an LTS over $LS \otimes GA$:

– the label part is embed in an option label structure ($\uparrow^\perp$)
– send markers are attached to actions ($\uparrow^!$)
– the labels of ta being pairs, the two previous transformations are composed to apply to pairs: ($\uparrow^\perp \otimes \uparrow^!$)
– these two updated fields are embed in a sum with the time part ($\uparrow^l$)

Concerning Clk, its labels, when untimed, must be extended to the product with LS^τ through $\uparrow^{\tau r}$. Timed labels are left unchanged ($\mathbb{1}$). Thus, the transformation ($\uparrow^{\tau r} \oplus \mathbb{1}$) is applied. Now, the product of the resulting LTSs, defined on the same label structure, is built. It is synchronized over labelled guarded actions, but asynchronous over time which is only present in the Clk automaton. For this purpose, we use the $\langle \bowtie \rangle$ operator. Lastly, as the LS label is always present in the resulting LTS, the optional label structure LS^τ is replaced by LS through the transformation ($\downarrow^\perp \otimes \mathbb{1} \oplus \mathbb{1}$) which does not modify the guard/action part and the time part of the label. Thus, we introduce the following definition:

Definition 18 (*Alternative TA semantics*) We define an alternative semantics for timed automata based on the product of the transformed timed automaton and the clock process.

$$[\![ta]\!] \triangleq [\downarrow^\perp \otimes \mathbb{1} \oplus \mathbb{1}]\,(\uparrow^l \circ(\uparrow^\perp \otimes \uparrow^!))(ta))\ \langle \underset{(LS\otimes GA!?)^\bullet}{\bowtie} \rangle\ C$$

with $C = (\uparrow^{\tau r} \oplus \mathbb{1})(Clk)$

Theorem 4 (*TA Semantics Compositionality*) *Given two timed automata ta_1 and ta_2, we have:* $[\![ta_1 \langle \bowtie \rangle ta_2]\!] \simeq [\![ta_1]\!] \langle \bowtie \rangle [\![ta_2]\!]$.

Proof This proof is based on the existence of a bisimulation between the semantics of the composition of ta_1 and ta_2 and the composition of their semantics. We start by unfolding the semantics of the TA composition $[\![ta_1]\!] \langle \bowtie \rangle [\![ta_2]\!]$ and by applying a sequence of bisimulations we reach $[\![ta_1 \langle \bowtie \rangle ta_2]\!]$. In the following proof, we denote:

$$\uparrow^{\tau\otimes!} = \uparrow^{LS^\tau\otimes GA!\oplus TS}_{LS\otimes GA} \qquad \downarrow^\tau = \downarrow^{LS^\tau}_{LS} \otimes \mathbb{1}_{GA!?} \oplus \mathbb{1}_{TS}$$

$$T_1 = \uparrow^{\tau\otimes!}(ta_1) \qquad T_2 = \uparrow^{\tau\otimes!}(ta_2) \qquad \langle \bowtie_\bullet \rangle = \langle \underset{(LS\otimes GA!?)^\bullet}{\bowtie} \rangle$$

We have:

$$[\![ta_1]\!]\langle\bowtie\rangle[\![ta_2]\!]$$
$$\simeq \{\text{Definition}\}$$
$$\downarrow^{\tau}(T_1\langle\bowtie_\bullet\rangle C)\langle\bowtie\rangle \downarrow^{\tau}(T_2\langle\bowtie_\bullet\rangle C)$$
$$\simeq \{\text{Factoring of } \downarrow^{\tau} \text{ since it is reverse composable (def : 9) : Corollary 5}\}$$
$$\downarrow^{\tau}((T_1\langle\bowtie_\bullet\rangle C)\langle\bowtie\rangle(T_2\langle\bowtie_\bullet\rangle C))$$
$$\simeq \{\text{Weakened associativity : law 2 of Corollary 3}\}$$
$$\downarrow^{\tau}(T_1\langle\bowtie_\bullet\rangle(C\langle\bowtie\rangle(T_2\langle\bowtie_\bullet\rangle C)))$$
$$\simeq \{\text{Commutativity : Proposition 6}\}$$
$$\downarrow^{\tau}(T_1\langle\bowtie_\bullet\rangle((T_2\langle\bowtie_\bullet\rangle C)\langle\bowtie\rangle C))$$
$$\simeq \{\text{Weakened associativity : law 2 of Corollary 3}\}$$
$$\downarrow^{\tau}(T_1\langle\bowtie_\bullet\rangle(T_2\langle\bowtie_\bullet\rangle(C\langle\bowtie\rangle C)))$$
$$\simeq \{\text{Idempotence: Proposition 7}\}$$
$$\downarrow^{\tau}(T_1\langle\bowtie_\bullet\rangle(T_2\langle\bowtie_\bullet\rangle C))$$
$$\simeq \{\text{Weakened associativity : law 1 of Corollary 3}\}$$
$$\downarrow^{\tau}((T_1\langle\bowtie_\bullet\rangle T_2)\langle\bowtie_\bullet\rangle C)$$
$$\simeq \{\text{Synchronous Composition: Proposition 5}\}$$
$$\downarrow^{\tau}((T_1\langle\bowtie\rangle T_2)\langle\bowtie_\bullet\rangle C)$$
$$\simeq \{\text{Substitution of } T_1 \text{ and } T_2 \text{ by their definitions}\}$$
$$\downarrow^{\tau}((\uparrow^{\tau\otimes!}(ta_1)\langle\bowtie\rangle \uparrow^{\tau\otimes!}(ta_2))\langle\bowtie_\bullet\rangle C)$$
$$\simeq \{\text{Factoring of } \uparrow^{\tau\otimes!} \text{ since it is reverse composable : Corollary 4}\}$$
$$\downarrow^{\tau}(\uparrow^{\tau\otimes!}(ta_1\langle\bowtie\rangle ta_2)\langle\bowtie_\bullet\rangle C)$$
$$\simeq \{\text{Definition}\}$$
$$[\![ta_1\langle\bowtie\rangle ta_2]\!]$$

Since rewrites are conditional, their hypotheses must be verified. Namely, $(LS \otimes GA!?)^\bullet$ is stable over $LS \otimes GA!? \oplus TS$, $\alpha ta_1 \uparrow^{LS\otimes GA!?\oplus TS}_{LS\otimes GA} \subseteq (LS \otimes GA!?)^\bullet$, ***DID*** is preserved by the transformations, Clk verifies the ***DID*** properties, the product and the transformation are compatible w.r.t bisimulation and Lemma 1 (page 26). A transcript of this proof is given in Appendix E. □

5.2.5 Comparison with standard TA semantics

We now state the equivalence between our TA semantics and the standard one defined as follows:

Definition 19 (*Reference TA semantics*) We call reference semantics of timed automata the LTS defined by the function $[\![\_]\!]_{std} : LT\ S_{LS\otimes GA} \to LT\ S_{LS\otimes GA\oplus TS}$ such that

$$[\![\langle Q, Q^0, \rightarrow\rangle]\!]_{std} = \langle Q \times (C \to \mathbb{R}^+), Q^0 \times \{c : C \mapsto 0\}, \rightarrow_s\rangle \text{ where:}$$

$$\frac{q \xrightarrow{(l,(g,(r,u)))} q' \quad \forall c \cdot v(c) \in g(c) \quad \forall c \cdot v'(c) \begin{cases} = 0 & \textbf{if } c \in r \\ = v(c) & \textbf{if } c \in u \\ \in \{0, v(c)\} & \textbf{else} \end{cases}}{(q, v) \xrightarrow{(l,(g,(r,u)))}_s (q', v')}$$

$$\frac{}{(q, v) \xrightarrow{\delta}_s (q, v + \delta)}$$

Note that transitions are not necessarily determined by their labels: clocks not belonging to $r \cup u$ can be either reset or left unchanged. The motivation of this semantics is to make easy the composition of timed automata.

Theorem 5 (standard vs. revised semantics) *Given a timed automaton, its standard $[\![\_]\!]_{std}$ and proposed $[\![\_]\!]$ semantics where receive annotations have been deleted, are upto-bisimilar through the identity relation on states $(\mathbb{1})$ and the relation $(\mathbb{1} \otimes \leq \oplus \mathbb{1})$ on labels which compares through $\leq$ its guard-action part.*

$$[\![ta]\!]_{std} \simeq^{\mathbb{1}\otimes\leq\oplus\mathbb{1}}_{\mathbb{1}} (\mathbb{1} \otimes \downarrow^? \oplus \mathbb{1})[\![ta]\!]$$

Proof Let us sketch the proof. We have to show the double inclusion between the transition relations of the semantic LTS associated to a given timed automaton ta. Several cases should be considered depending on the structure of the label (timed or untimed). We only consider here an untimed label composed of a quadruple (l, g, r, u) where $l \in LS, g \in G, r \subseteq C$ and $u \subseteq C$.

– The following proof tree presents the main steps of one direction: from a transition for a transformation by $\mathbb{1} \otimes \downarrow^? \oplus \mathbb{1}$ of the proposed semantics $[\![ta]\!]$, we build a transition for the standard semantics. Reading the tree top-down, first we undo the transformation and restore the receive marker. By the definition of LTS product, we get a send transition in ta labelled by some $\overline{g, r, u}$ and a receive transition in Clk labelled by g, r, u such that $\overline{g, r, u} \leq g, r, u$. By the definition of Clk, $\{r, u\}$ is a partition of the set of clocks C. Thus $\overline{r} = r$ and $\overline{u} = u$. As a consequence, we get a transition labelled $l, \overline{g, r, u}$ for the standard semantics and the relation between labels is satisfied.

$$\frac{\dfrac{\dfrac{(s, v) \xrightarrow{l,g,r,u}_{(\mathbb{1}_{LS} \otimes \downarrow^{GA?}_{GA} \oplus \mathbb{1}_{TS})[\![ta]\!]} (s', v')}{(s, v) \xrightarrow{l,(g,r,u)?}_{[\![ta]\!]} (s', v')} \qquad \dfrac{s \xrightarrow{l,\overline{(g,r,u)}!}_{ta\uparrow} s' \quad v \xrightarrow{(g,r,u)?}_{Clk} v' \quad \overline{g, r, u} \leq (g, r, u)}{s \xrightarrow{l,\overline{g,r,u}}_{ta} s' \quad g(v), u = C \setminus r,\ \forall c \cdot v'(c) = 0 \textbf{ if } c \in r, v(c) \textbf{ else } g \subseteq \overline{g}, \overline{r} \subseteq r, \overline{u} \subseteq u}}{\overline{g}(v),\ \forall c \cdot v'(c) \begin{cases} = 0 & \textbf{if } c \in \overline{r} \\ = v(c) & \textbf{if } c \in \overline{u} \\ \in \{0, v(c)\} & \textbf{else} \end{cases}}}{(s, v) \xrightarrow{l,\overline{g,r,u}}_{[\![ta]\!]_{std}} (s', v')}$$

– Consider now a transition of the standard semantics $[\![ta]\!]_{std}$. In the same way, we build a transition for a transformation deleting receive markers of the proposed semantics $[\![ta]\!]$.

The main hint consists in adding to r clocks of which values are undetermined by the r, u action but of value 0 in v', thus obtaining $\overline{r}$. We get $\overline{g, r, u}$ satisfying $(g, r, u) \leq \overline{g, r, u}$ so that the product can be built to provide the ta transition. As a last step, the receive marker is deleted.

$$\dfrac{\dfrac{\dfrac{(s,v) \xrightarrow{l,(g,r,u)} [\![ta]\!]_{std}\ (s',v')}{g(v),\ \forall c \cdot v'(c) \begin{cases} = 0 & \textbf{if } c \in r \\ = v(c) & \textbf{if } c \in u \\ \in \{0, v(c)\} & \textbf{else} \end{cases}} \quad \dfrac{g(v),\ v \xrightarrow{\overline{(g,r,u)}?}_{Clk} v',\ \overline{g} = g,\ \overline{r} = r \cup v'^{-1}(0) \setminus u,\ \overline{u} = C \setminus \overline{r}}{\overline{g}(v),\ v \xrightarrow{(\overline{g,r,u})?}_{Clk},\ (g,r,u) \leq \overline{g,r,u}}}{s \xrightarrow{l,(g,r,u)!}_{ta\uparrow} s'}}{(s,v) \xrightarrow{l,\overline{g,r,u}?} [\![ta]\!]\ (s',v'),\ (g,r,u) \leq \overline{g,r,u}}$$

$$\dfrac{(s,v) \xrightarrow{l,\overline{g,r,u}?} [\![ta]\!]\ (s',v'),\ (g,r,u) \leq \overline{g,r,u}}{(s,v) \xrightarrow[(\mathbb{1}_{LS} \otimes \downarrow_{GA}^{GA?} \oplus \mathbb{1}_{TS})[\![ta]\!]}{l,\overline{g,r,u}} (s',v'),\ (g,r,u) \leq \overline{g,r,u}}$$

We can notice that labels ensuring the simulation property are not identical: not only the receive marker can be added or deleted, but the contents of the guarded action differs. This is because the target state is not fully specified by the source state and the action. This is done in order to allow synchronous composition with other timed automata. □

5.3 Timed automata with invariants

We now add state invariants to timed automata as defined in [23] where an invariant is an upper bound constraint that may be associated to each clock.

5.3.1 Invariant label structure

An invariant is defined as a partial function from clocks to the time domain Δ. The composition of two invariants associates to each clock, when it exists, the minimum of the two bounds. The invariant label structure is defined as:

$$I \triangleq \left\langle C \nrightarrow \Delta,\ (i_1, i_2) \mapsto \left(c \mapsto \begin{cases} \min(i_1(c), i_2(c)) \textbf{ if } c \in \textbf{dom}(i_1) \cap \textbf{dom}(i_2) \\ i_1(c) \textbf{ if } c \in \textbf{dom}(i_1) \setminus \textbf{dom}(i_2) \\ i_2(c) \textbf{ if } c \in \textbf{dom}(i_2) \setminus \textbf{dom}(i_1) \end{cases} \right) \right\rangle$$

The invariant label structure is ***ACI***.

5.3.2 Timed automata with invariants label structure

Given a set of clocks C, we define a timed safety automaton (TSA) [23] as a LTS such that its transitions are labelled by communication channels (defined by some label structure LS) and guarded actions (the label structure GA). Furthermore, invariants are attached to locations. These invariants are here stored on special looping transitions in order to synchronize with the Invariant Clock controller (IClk in paragraph 5.3.3). Given a label structure LS, a timed safety automaton over LS is a LTS over $LS \otimes GA \oplus I$.

$$TSA_{LS} \triangleq LTS_{LS \otimes GA \oplus I}$$

Remark 6 (TSA Composition) Thanks to our label structure, the TSA composition is defined as the composition of the underlying LTS systems.

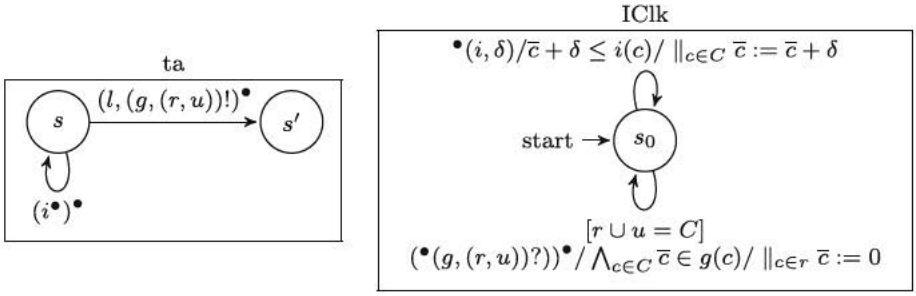

Fig. 3 Semantics of TSA via a composition of two LTSs

5.3.3 Revised TSA semantics

In this paragraph, we define the semantics of a timed safety automaton as a LTS over $LS \otimes GA!? \oplus I \otimes TS$. As before, this semantics makes explicit time transitions and adds communication with a server which manages clock valuations. In Sect. 5.3.4, it will be compared with the standard semantics.

Lemma 2 *If LS is associative (commutative) then* $LS \otimes GA!? \oplus I \otimes TS$ *is associative (commutative).*

Proof GA, I and TS are associative (commutative). Lemma 2 follows since by Proposition 2, $\oplus$ and $\otimes$ preserve associativity (commutativity). □

The TSA semantics is given via a composition with an invariant clock manager $IClk$ defined over $GA!? \oplus I \otimes TS$ (Fig. 3) where $\overline{c}$ denotes the variable associated with a clock $c \in C$.

Invariant Clock Manager. The $IClk$ automaton extends the Clk automaton by constraining the elapse of time. It synchronizes on the invariant i specified by the user-provided timed automaton to advance each clock valuation by some δ if the invariant is preserved, and on any GA label $(g,(r,u))$ with $r \cup u = C$ when the guard $g(c)$ is satisfied by each clock c, resetting clocks of r. The synchronization between the timed automaton and the controller is possible when the required $(r,u)!$ action is included in the accepted $(r',u')?$ action, which means at least reset and unchange orders are performed. Thus clocks not belonging to $r \cup u$ may be non deterministically reset or left unchanged.

Since $IClk$ is diagonal, idempotent, and deterministic (the arguments are the same as for Clk), it follows that:

$$IClk \ \langle\bowtie\rangle \ IClk \ \simeq \ IClk \ (Proposition\ 7)$$

Reconstructing the TSA Semantics. As before, the *revised TSA semantics* is defined by means of a composition between the syntactic tsa and $IClk$:

$$[\![tsa]\!] \triangleq \begin{array}{l}[\downarrow^{\perp} \otimes \mathbb{1} \oplus \mathbb{1} \otimes \downarrow^{\perp}] \\ \big((\uparrow^{\perp} \otimes \uparrow^{!} \oplus (\uparrow^{l\tau} \circ \uparrow^{!}))(tsa) \ \langle\bowtie\rangle \ (\uparrow^{\tau r} \oplus \mathbb{1} \otimes \uparrow^{\perp})(IClk)\big)\end{array}$$

Several transformations are applied to convert the underlying label structure of the two LTS in $LS^{\tau} \otimes GA!? \oplus I!? \otimes TS^{\tau}$. For tsa, the transformation adds a send annotation to GA labels and to I labels which are also embedded into $I!? \otimes TS$. These basic transformations

are themselves composed to be applicable on tsa. For $ICIk$, guarded actions are embedded into labels of the product $LS^{\tau} \otimes GA!?$.

Theorem 6 (TSA Semantics Compositionality) *Given two timed safety automata tsa_1 and tsa_2, $[\![tsa_1 \langle \bowtie \rangle tsa_2]\!] \simeq [\![tsa_1]\!] \langle \bowtie \rangle [\![tsa_2]\!]$.*

Proof The proof of this theorem is similar to the one of Theorem 4. □

5.3.4 Comparison with standard TSA semantics

We now state the equivalence between our revised TSA semantics and the standard one in which a specific encoding of the state invariant is taken into account. We start by defining the *standard TSA semantics* by the function $[\![\_]\!]_{std} : LTS_{LS \otimes GA \oplus I} \rightarrow LTS_{LS \otimes GA \oplus TS}$ such that $[\![\langle Q, Q^0, \rightarrow \rangle]\!]_{std} = \langle Q \times (C \rightarrow \mathbb{R}^+), Q^0 \times \{c : C \mapsto 0\}, \rightarrow_s \rangle$ where:

$$\frac{q \xrightarrow{(l,(g,(r,u)))} q' \ , \ \bigwedge_{c \in C} v(c) \in g(c) \ , \forall c \cdot v'(c) = 0 \textbf{ if } c \in r, = v(c) \textbf{ if } c \in u, \in \{0, v(c)\} \textbf{ else}}{(q, v) \xrightarrow{(l,(g,(r,u)))}_s (q', v')}$$

$$\frac{q \xrightarrow{i} q \quad \forall c \in C \cdot v(c) + \delta \leq i(c)}{(q, v) \xrightarrow{\delta}_s (q, v + \delta)}$$

The transitions of these semantic rules are thus labelled either by tuples containing the synchronization label and memory access orders (guards to be checked, clocks to be reset or left unchanged), or by a delay δ.

Theorem 7 (Standard and revised semantics) *Given a timed safety automaton, its standard and revised semantics are bisimilar upto the relation* $(\mathbb{1} \otimes \leq \oplus \leq \otimes \mathbb{1})$.

$$[\![tsa]\!]_{std} \ \simeq_{\mathbb{1}}^{\mathbb{1} \otimes \leq \oplus \leq \otimes \mathbb{1}} \ [\mathbb{1} \otimes \downarrow^? \oplus \downarrow^? \otimes \mathbb{1}][\![tsa]\!]$$

Proof The proof of this theorem is similar to the one of Theorem 5. □

6 Towards behavioral label structures

In this section, we show how the previous semantic constructions could be generalized by attaching behaviors to label structures. Up to now, a label structure defines how labels are composed and consequently how labelled transition systems are composed. The attached LTS, acting as a controller, is used to build the semantics of a LTS. We apply the same methodology as for timed automata: starting from a *syntactic* LTS built on a product label structure $LS_u \otimes LS_m$ (user and medium label structures), its semantics will be defined over a label structure $LS_u \otimes LS_m!? \oplus LS_c$ through a composition with a contoller over $LS_u \oplus LS_c$. As before, we give two semantics, one called *standard*, the other called *revised* to keep the same vocabulary as for timed automata.

Definition 20 (*Behavioral Label Structure*) A behavioral label structure is a tuple $\langle LS_u, LS_m, LS_c, \mathcal{C} \rangle$ where LS_i are label structures, LS_m is supposed ***ACI*** and $\mathcal{C} = \langle Q_{\mathcal{C}}, Q^0_{\mathcal{C}}, \rightarrow_{\mathcal{C}} \rangle$ is an LTS over $LS_m!? \oplus LS_c$.

An LTS over such a behavioral label structure is an LTS over $LS_u \otimes LS_m$. We define its *standard semantics* by the function $[\![\_]\!]^{std}_{\mathcal{C}} : LTS_{LS_u \otimes LS_m} \rightarrow LTS_{LS_u \otimes LS_m \oplus LS_c}$ such that $[\![\mathcal{L} = \langle Q, Q^0, \rightarrow \rangle]\!]^{std}_{\mathcal{C}} = \langle Q \times Q_{\mathcal{C}}, Q^0 \times Q^0_{\mathcal{C}}, \rightarrow_s \rangle$ where:

$$\frac{q \xrightarrow{(l_u,l_m)} q' \ ,\ q_{\mathcal{C}} \xrightarrow{l'_m?^{\bullet}}_{\mathcal{C}} q'_{\mathcal{C}} \ ,\ l_m \leq l'_m}{(q,q_{\mathcal{C}}) \xrightarrow{(l_u,l_m)^{\bullet}}_{s} (q',q'_{\mathcal{C}})} \qquad \frac{q_{\mathcal{C}} \xrightarrow{\bullet l}_{\mathcal{C}} q'_{\mathcal{C}}}{(q,q_{\mathcal{C}}) \xrightarrow{\bullet l}_{s} (q,q'_{\mathcal{C}})}$$

The semantics is an LTS $\mathcal{L}$ over the product of the state space of the syntactic LTS $\mathcal{L}$ and the controller $\mathcal{C}$. Its transitions are built by joining transitions on $LS_u \otimes LS_m$ provided by $\mathcal{L}$ and $\mathcal{C}$ and adding transitions on LS_c provided by $\mathcal{C}$ only. In order to allow the composition with user-given LTSs, we add non-determinism through the introduction of the label l'_m such that $l_m \leq l'_m$. More precisely, two families of transitions are defined over the product space:

- transitions labelled by (l_u, l_m) in $LS_u \otimes LS_m$ are fired if they are present in $\mathcal{L}$ and accepted by $\mathcal{C}$, through a label $l'_m \geq l_m$, marked with a *receive* tag
- transitions labelled by elements of LS_c, asynchronously accepted by the controller $\mathcal{C}$.

This *standard* semantics has been chosen to match the one previously proposed for timed automata after abstracting from label structures and controller. This semantics is shown to be equivalent to the *revised semantics* $[\![\_]\!]_{\mathcal{C}}$ defined by reusing the label structure operators and labelled transition systems transformations:

$$[\![\mathcal{L}]\!]_{\mathcal{C}} \triangleq [\downarrow^{\perp} \otimes \mathbb{1} \oplus \mathbb{1}]\ (\uparrow^{l}((\uparrow^{\perp} \otimes \uparrow^{!})(\mathcal{L}))\ \langle \underset{LS_u?\otimes LS_m}{\bowtie} \rangle\ (\uparrow^{\tau r} \oplus \mathbb{1})(\mathcal{C}))$$

The LTS $\mathcal{L}$ is transformed by embedding LS_u labels in the optional label structure LS_u^{τ}, adding send markers to LS_m labels for synchronizations with the controller, and embedding the resulting label in a sum with LS_c. The controller labels are transformed so that when in LS_m, they are coupled with a τ from LS_u^{τ}. The LTS resulting from the product is then converted into a $LS_u \otimes LS_m!? \oplus LS_c$ LTS.

Theorem 8 (Standard and revised semantics) *Given two LTSs $\mathcal{L}$ over $LS_u \otimes LS_m$ and $\mathcal{C}$ over $LS_m!? \oplus LS_c$ with LS_m **ACI**, $[\![\mathcal{L}]\!]^{std}_{\mathcal{C}}$ and $[\![\mathcal{L}]\!]_{\mathcal{C}}$ are bisimilar upto the relation $(\mathbb{1} \otimes \leq \oplus \leq \otimes \mathbb{1})$ and the removal of receive markers if LS_m labels of the transitions of $\mathcal{C}$ are maximal for $\leq$.*

$$[\![\mathcal{L}]\!]_{std} \simeq^{\mathbb{1}\otimes \leq \oplus \mathbb{1}}_{\mathbb{1}} (\mathbb{1} \otimes \downarrow^{?} \oplus \mathbb{1})[\![\mathcal{L}]\!]_{\mathcal{C}}$$

The proof of this theorem is similar to the one for timed automata. As for timed automata, the maximality hypothesis is used to conclude on the identity between matching labels of $\mathcal{L}$ and $\mathcal{C}$, as was the partition condition in Clk.

The compositionality result concerning the parallel operator of timed automata can also be generalized as follows:

Theorem 9 (Generalized Compositionality) *Given two LTSs $\mathcal{L}_1$ and $\mathcal{L}_2$ over $LS_u \otimes LS_m$ and a controller $\mathcal{C}$ over $LS_m!? \oplus LS_c$, we have:*

$$[\![\mathcal{L}_1 \langle \bowtie \rangle \mathcal{L}_2]\!]_{\mathcal{C}} \simeq [\![\mathcal{L}_1]\!]_{\mathcal{C}} \langle \bowtie \rangle [\![\mathcal{L}_2]\!]_{\mathcal{C}}$$

if the following conditions hold:

- *LS_u and LS_c are associative and commutative.*
- *LS_m is **ACI**.*
- *$\mathcal{C}$ is **DID**.*

This theorem has a similar proof as the timed automata one. Furthermore, all the hypotheses are satisfied in the timed automata context.

7 Conclusion

We have presented a formal semantic framework for studying, defining, and manipulating the composition of extended transition systems based on the composition of their labels. The framework is based on the idea of defining a label structure containing a composition operator. Depending on the language in question, a different label structure is defined and thus different composition laws are integrated. The label structure is then used as a parameter of labelled transition systems which describe the common semantic domain of the considered languages. We believe that the suggested parameterization of the behavioral framework is a promising work and may represent, especially with the perspectives we have, the first step towards giving a unified formal semantic framework for different process algebras and specification languages.

In this paper, we have studied parallel composition operators of process algebras regardless of other behavioral operators. In this context, we have pushed forward existing works of similar structures [8,24,26,34] by offering a richer set of operations and properties such as the composition of label structures and transformations between label structures.

Following our technique, the composition of different LTS extensions, whether it is a syntactic model or a semantic model, is captured by a unique composition operation defined on LTS. This is a direct result of the separation between the label structure and the behavioral framework. This result is different from those found in the literature since with each system, a different composition operation is provided. This can be seen classically in the composition operations of LTS and TTS. Even though a TTS is exactly a LTS having additionally time transitions, usually its composition operation does not reuse the LTS one.

Furthermore, generic results concerning label structures and LTS transformations are applied to establish well known properties of high-level structures such as the definition of timed automata semantics. We have shown that these semantics match with the standard timed automata semantics and that timed automata bisimilarity is compositional w.r.t. the parallel operator.

Finally, all the definitions and theorems related to the presented framework have been formalized and validated using the proof assistant Coq. The Coq theory may be found in [14]. The Coq formalisation has induced representation choices linked to the difference between type theory and set theory. For example, we cannot silently embed a set into a disjoint union as it is usually done in set theory. Furthermore, partial functions (such as our $\bowtie$ operator) and total functions are hard to integrate in a single notation.

We have also experimented automatic proof tools such as the SMT-based solver Z3 (invoked by the Why3 platform). Using Z3, we have efficiently solved one of several combinatory results, but failed on other proofs such as the preservation of associativity by the label structure product. However, even if Z3 can produce machine-checkable proofs, it is not interfaced with Coq for now.

We are now working on a dual view of this work which consists in coupling our label structures with states. This will help us to naturally take into consideration state-based mechanisms such as the committed states of UPPAAL [9]. We are also working on defining the formal semantics of real time languages (BIP [7], FIACRE [19] and ACSR [16]). Namely, we are interested in extending our label structure with priorities which are present in all of the three cited languages. Another extension would consist in adding other operators to label structures:

– sequential composition: this feature is required for example to express usual constraints on time transition systems, such as additivity or continuity. It could be done at the level of label structures.
– negation: it could be correlated to the absence of transition with a given label, as proposed in [13].
– refinement relations and more generally lattice relations as used in contract theories [31].

Lastly, it could be interesting to investigate this work within a categorical context, as proposed by [38] which sets the foundations for this perspective. Moreover, with respect to that work, it could also be interesting to study the preservation of dynamic properties (reachability, acyclicity, …) by LTS operators. We also consider reusing the existing works on rule formats dealing with properties like commutativity, associativity [18] and idempotence [2] in order to better structure the mechanization of the proposed framework.

Acknowledgements We would like to thank the anonymous reviewers for their careful reading of our manuscript and their helpful suggestions and comments.

A Coq definition of a label structure, associativity and commutativity

```
Record LblStr: Type := mkLS {
  Label: Type;
  Lprd: Label -> Label -> Label;
  Lcnd: Label -> Label -> Prop
}.
Implicit Arguments Lerr.
Implicit Arguments Lprd.
Implicit Arguments Lcnd.

Record isAssoc LS: Prop := {
  isAssoc1: ∀ l1 l2 l3, Lcnd LS l1 l2 -> Lcnd LS (Lprd
LS l1 l2) l3 -> Lcnd LS l2 l3;
  isAssoc2: ∀ l1 l2 l3, Lcnd LS l1 l2 -> Lcnd LS (Lprd LS l1
 l2) l3 -> Lcnd LS l1 (Lprd LS l2 l3);
  isAssoc3: ∀ l1 l2 l3, Lcnd LS l2 l3 -> Lcnd LS l1 (Lprd
 LS l2 l3) -> Lcnd LS l1 l2;
  isAssoc4: ∀ l1 l2 l3, Lcnd LS l2 l3 -> Lcnd LS l1 (Lprd LS
l2 l3) -> Lcnd LS (Lprd LS l1 l2) l3;
  isAssoc5: ∀ l1 l2 l3, Lcnd LS l1 l2 -> Lcnd LS (Lprd LS l1
l2) l3 ->
    Lprd LS l1 (Lprd LS l2 l3) = Lprd LS (Lprd LS l1 l2) l3
}.

Record isComm LS: Prop := {
  isComm1: ∀ l1 l2, Lcnd LS l1 l2 -> Lcnd LS l2 l1;
  isComm2: ∀ l1 l2, Lcnd LS l1 l2 -> Lprd LS l1 l2 =
 Lprd LS l2 l1
}.
```

B Coq definition for the product of label structures

The ⋈ operator and its domain of definition are specified in Coq as follows:

```
Definition LSprod (LS1 LS2: LblStr): LblStr := {|
  Label := Label LS1 * Label LS2;
 Lprd l r := match l,r with
    (l1,l2), (r1,r2) ⇒ (Lprd _ l1 r1, Lprd _ l2 r2)
  end;
  Lcnd l r := match l,r with
    (l1,l2), (r1,r2) ⇒ Lcnd _ l1 r1 ∧ Lcnd _ l2 r2
  end
|}.

Notation "l1 *l* l2" := (LSprod l1 l2)
  (at level 110, left associativity).
```

C Coq definition for the sum of label structures

```
Inductive SumLabs L1 L2 :=
  LLab: L1 -> SumLabs L1 L2
| RLab: L2 -> SumLabs L1 L2.

Definition LSsum (LS1 LS2: LblStr): LblStr := {|
  Label := SumLabs (Label LS1) (Label LS2);
  Lprd l1 l2 := match l1,l2 with
    LLab x1, LLab x2 => LLab _ _  (Lprd _ x1 x2)
  | RLab x1, RLab x2 => RLab _ _ (Lprd _ x1 x2)
  | LLab x1, _ => LLab _ _ x1 (* unused *)
  | RLab x1, _ => RLab _ _ x1 (* unused *)
  end;
  Lcnd l1 l2 := match l1,l2 with
    LLab x1, LLab x2 => Lcnd _ x1 x2
  | RLab x1, RLab x2 => Lcnd _ x1 x2
  | _,_ => False
  end
|}.

Notation "l1 +l+ l2" := (LSsum l1 l2)
  (at level 112, left associativity).
```

D Coq statement of the general associativity lemma for LTS product

```
Record assoc_hyp LS
    (A1 A2 A3 A4 S1 S2 A1' A2' A3' A4' S1' S2': Label LS ->
 Prop): Prop := {
  ah1: S1 & !(S2 & Prd) <: S2';
```

```
  ah1b: S1 & !A4 <: S2';
  ah1c: S2' & (S1' & Prd)! <: S1;
  ah1d: S2' & A1'! <: S1;
  ah2: (S1 & ! (S2 & Prd)).1 <: S1';
  ah3: S1 & !A3 <: A3';
  ah4: S1 & !A3 <: S1';
  ah5: (S1 & !A4).1 <: A1';
  ah8: S2 & Prd & A2 <: S2';
  ah9: (S2 & A2).1 <: A2';
  ah13: (S2' & (S1' & Prd)!).2 <: S2;
  ah14: (S2' & A1'!).2 <: A4;
  ah15: S2' & A2'! <: A2;
  ah16: S2' & A2'! <: S2;
  ah17: S1' & Prd & A3' <: S1;
  ah18: (S1' & A3').2 <: A3;

  eh1: A4' <:> A2 & A4;
  eh2: A1 <:> A3' & A1';
  eh3: A2 & A3 <:> A2' & A3'
}.

Lemma general_par_assoc:
  ∀ LS (A1 A2 A3 A4 S1 S2 A1' A2' A3' A4' S1' S2': Label LS
 -> Prop)
    (T1 T2 T3: LTS LS),
  isAssoc LS -> assoc_hyp A1 A2 A3 A4 S1 S2 A1' A2' A3'
 A4' S1' S2' ->
  T1 |[A1,S1,A2]| (T2 |[A3,S2,A4]| T3) ==
  (T1 |[A1',S1',A2']| T2) |[A3',S2',A4']| T3.
```

E Coq script for compositionality

This Coq proof script illustrates how the various results established in the paper can be reused to prove the compositionality of the product of two automata.

```
Theorem sem_prod:
   forall LS, isAssoc LS -> isComm LS ->
     forall (T1 T2: TA LS),
        (sem (T1 |t| T2)) == ((sem T1) |t| (sem T2)).
Proof.
  intros.
  unfold sem.
  assert (isAssoc (LS *l* ACTION +l+ TIME)) as A.
  apply LSsumAssoc; auto.
  apply LSprodAssoc; auto.
  apply ACTION_assoc.
  apply TIME_assoc.
  assert (isComm (LS *l* ACTION +l+ TIME)) as C.
```

```
  apply LSsumComm; auto.
  apply LSprodComm; auto.
  apply ACTION_comm.
  apply TIME_comm.

  match goal with
    |- _ == ((?x | ?s | ?y) |t| ?z) =>
       rewrite <-(asyncl_par_assoc
                   (s: Label (LS *l* ACTION +l+ TIME)->Prop)
                   x y z A (isLLab_stable _ _)
                   (notIsLLab_stable _ _)); simpl; intros;
 auto
  end.

  match goal with
    |- _ == (?x | ?s1 | (?y |t| ?z)) =>
       rewrite (par_com C y z); simpl; intros; auto
  end.

  match goal with
    |- _ == (?x | ?s1 | ((?z | ?s2 | ?t) |t| ?y)) =>
       rewrite <- (asyncl_par_assoc
                    (s2: Label (LS *l* ACTION +l+ TIME)->
Prop)
                    z t y A (isLLab_stable _ _ )
                    (notIsLLab_stable _ _)); auto
  end.
  rewrite par_idem_det; simpl; intros; auto.

  match goal with
    |- _ == (?x | ?s1 | (?y | ?s2 | ?z)) =>
       rewrite (asynclr_par_assoc
                    (s2: Label (LS *l* ACTION +l+ TIME)->
Prop)
                    x y z A (isLLab_stable _ _)
                    (notIsLLab_stable _ _)); simpl; intros;
 auto
  end.
  rewrite <- (sync_async
       (isLLab: Label (LS *l* ACTION +l+ TIME)->Prop)
       ([[tr_inl TIME]] T1) ([[tr_inl TIME]] T2)
       (isLLab_stable _ _ )) ; auto.
  rewrite (fun h => apply_spar (tr_inl TIME) h (isInj_inl)
 T1 T2).
  reflexivity.
  apply stable_inl.
  apply alpha_inl.
  apply alpha_inl.
  apply isDet_tr.
```

```
  apply isDet_Clock.
  apply isInj_tfrl.
  apply isInj_extr.
  apply isTDiag_tr.
  apply isTDiag_Clock.
  apply isInj_tfrl.
  apply isInj_extr.
  apply isTIdem_tr.
  apply isTIdem_Clock.
  intros.
  apply (alpha_inl T2 l H1).
  apply (alpha_inl (LS2 := TIME) T1 l H1).
Qed.
```

References

1. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering, 1st edn. Cambridge University Press, New York (2010)
2. Aceto, L., Birgisson, A., Ingólfsdóttir, A., Mousavi, M.R., Reniers, M.A.: Rule formats for determinism and idempotence. Sci. Comput. Program. **77**(7–8), 889–907 (2012)
3. Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Inf. Comput. **104**(1), 2–34 (1993)
4. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
5. Arbab, F.: Abstract behavior types: a foundation model for components and their composition. Sci. Comput. Program. **55**(1–3), 3–52 (2005)
6. Arnold, A., Point, G., Griffault, A., Rauzy, A.: The AltaRica formalism for describing concurrent systems. Fundam. Inf. **40**(2–3), 109–124 (1999)
7. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: SEFM, pp. 3–12. IEEE Computer Society (2006)
8. Bauer, S.S., Juhl, L., Larsen, K.G., Legay, A., Srba, J.: Extending modal transition systems with structured labels. Math. Struct. Comput. Sci. **22**(4), 581–617 (2012)
9. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004. Revised Lectures, volume 3185 of Lecture Notes in Computer Science, pp. 200–237. Springer (2004)
10. Bengtsson, J., Yi, W.: Timed automata: semantics, algorithms and tools. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This Tutorial Volume Originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In Addition to Lectures Given at ACPN 2003, Additional Chapters Have Been Commissioned], Volume 3098 of Lecture Notes in Computer Science, pp. 87–124. Springer (2003)
11. Berendsen, J., Vaandrager, F.W.: Compositional abstraction in real-time model checking. In: Cassez, F., Jard, C. (eds.) Formal Modeling and Analysis of Timed Systems, 6th International Conference, FORMATS 2008, Saint Malo, France, September 15–17, 2008. Proceedings, volume 5215 of Lecture Notes in Computer Science, pp. 233–249. Springer (2008)
12. Berry, G., Gonthier, G.: The Esterel synchronous programming language: design, semantics, implementation. Sci. Comput. Program. **19**(2), 87–152 (1992)
13. Bliudze, S., Sifakis, J.: A notion of glue expressiveness for component-based systems. In: van Breugel and Chechik [36], pp. 508–522
14. Bodeveix, J.-P.: http://www.irit.fr/~Jean-Paul.Bodeveix/COQ/LblStr
15. Brauer, W., Reisig, W., Rozenberg, G. (eds). Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, 8–19. September 1986, Volume 255 of Lecture Notes in Computer Science. Springer (1987)
16. Brémond-Grégoire, P., Lee, I., Gerber, R.: ACSR: an algebra of communicating shared resources with dense time and priorities. In: Best, E. (ed.) CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23–26, 1993, Proceedings, Volume 715 of Lecture Notes in Computer Science, pp. 417–431. Springer (1993)

17. Chatterjee, K., Doyen, L., Henzinger, T.A.: Probabilistic weighted automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009—Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1–4, 2009. Proceedings, Volume 5710 of Lecture Notes in Computer Science, pp. 244–258. Springer (2009)
18. Cranen, S., Mousavi, M.R., Reniers, M.A.: A rule format for associativity. In: van Breugel and Chechik [36], pp. 447–461
19. Farail, P., Gaufillet, P., Peres, F., Bodeveix, J.-P., Filali, M., Berthomieu, B., Rodrigo, S., Vernadat, F., Garavel, H., Lang, F.: FIACRE: an intermediate language for model verification in the TOPCASED environment. In: European Congress on Embedded Real-Time Software, ERTS'08 (2008)
20. Fares, E., Bodeveix, J.-P., Filali, M.: Event algebra for transition systems composition—application to timed automata. In: Proceedings of the 2013 20th International Symposium on Temporal Representation and Reasoning, TIME '13, pp. 125–132. IEEE Computer Society, Washington (2013)
21. Groote, J.F., Ponse, A.: The syntax and semantics of μcrl. In: Ponse, A., Verhoef, C., van Vlijmen, S.F.M. (eds.) Algebra of Communicating Processes: Proceedings of ACP94, the First Workshop on the Algebra of Communicating Processes, Utrecht, The Netherlands, 16–17 May 1994, pp. 26–62. Springer, London (1995)
22. Henzinger, T., Manna, Z., Pnueli, A.: Timed transition systems. In: de Bakker, J., Huizing, C., de Roever, W., Rozenberg, G. (eds): Real-Time: Theory in Practice, Volume 600 of Lecture Notes in Computer Science, pp. 226–251. Springer. doi:10.1007/BFb0031995 (1992)
23. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Inf. Comput. **111**(2), 193–244 (1994)
24. Hoare, T., O'Hearn, P.: Separation logic semantics for communicating processes. Electron. Notes Theor. Comput. Sci. **212**, 3–25 (2008)
25. Hüttel, H., Larsen, K.: The use of static constructs in a modal process logic. In: Meyer, A., Taitslin, M. (eds.) Logic at Botik '89, Lecture Notes in Computer Science, vol. 363, pp. 163–180. Springer, Berlin (1989)
26. I. O. for Standardization. Information processing systems-open systems interconnection-LOTOS—a formal description technique based on the temporal ordering of observational behaviour. International standard. ISO (1989)
27. Larsen, K., Pettersson, P., Yi, W.: Model-checking for real-time systems. In: Reichel, H. (ed) Fundamentals of Computation Theory, Volume 965 of Lecture Notes in Computer Science, pp. 62–88. Springer. doi:10.1007/3-540-60249-6_41 (1995)
28. Milner, R.: Calculi for synchrony and asynchrony. Theor. Comput. Sci. **25**(3), 267–310 (1983)
29. Milner, R.: Communication and Concurrency. Prentice Hall International, Upper Saddle River (1995)
30. Mousavi, M.R., Reniers, M.A., Basten, T., Chaudron, M.R.V.: PARS: a process algebra with resources and schedulers. In: Larsen, K.G., Niebert, P. (eds) Formal Modeling and Analysis of Timed Systems: First International Workshop, FORMATS 2003, Marseille, France, September 6–7, 2003. Revised Papers, Volume 2791 of Lecture Notes in Computer Science, pp. 134–150. Springer (2003)
31. Raclet, J.-B., Badouel, E., Benveniste, A., Caillaud, B., Legay, A., Passerone, R.: A modal interface theory for component-based design. Fundam. Inform. **108**(1–2), 119–149 (2011)
32. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall, Upper Saddle River (1997)
33. Roscoe, A.W.: On the expressiveness of CSP. https://www.cs.ox.ac.uk/files/1383/expressive.pdf (2011)
34. Sekerinski, E., Sere, K.: A theory of prioritizing composition. Comput. J. **39**(8), 701–712 (1996)
35. The Coq development team. The Coq proof assistant reference manual. LogiCal Project. Version 8.4 (2013)
36. van Breugel, F., Chechik, M. (eds): CONCUR 2008—Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19–22, 2008. Proceedings, Volume 5201 of Lecture Notes in Computer Science. Springer (2008)
37. Verhoef, C.: A congruence theorem for structured operational semantics with predicates and negative premises. Nord. J. Comput. **2**(2), 274–302 (1995)
38. Winskel, G., Nielsen, M.: Handbook of Logic in Computer Science. Chapter Models for Concurrency, vol. 4, pp. 1–148. Oxford University Press, Oxford (1995)