Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 264 (2010) 177–197
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Coinduction in Concurrent Timed Systems

Jan Komenda[1,2]

Institute of Mathematics, Czech Academy of Sciences, Brno, Czech Republic

Abstract

An important class of timed transition systems can be modeled by deterministic weighted automata, which are essentially partial Mealy automata, and their extensions using synchronous compositions defined over extended alphabets. From a coalgebraic viewpoint, behaviours of deterministic partial Mealy automata are causal and length preserving partial functions between finite and infinite sequences of inputs and outputs, called stream functionals. After a study of fundamental properties of functional stream calculus an application to the definition by coinduction of the synchronous product of stream functionals is proposed.

Keywords: deterministic weighted automata, Mealy automata, final coalgebra, synchronous product, coinduction

1 Introduction

Universal coalgebra as a general theory of (dynamical) systems offers definitions and proofs by coinduction [16], which are complementary to classical approaches based on induction and turned out to be valuable in simplifying the definitions and proofs of many concepts and properties that are hard or even impossible to formulate within the algebraic framework. The techniques borrowed from coalgebra have proven their usefullness in many areas of theoretical computer science (e.g. functional and object orienting programming), but also in control theory, in particular of discrete state transition systems that are called in control community discrete-event systems. The reference model for discrete-event systems are partial automata, which are coalgebras of a functor on the category of sets. They have been studied in [15] as the model for control of discrete-event (dynamical) systems (DES) together with the partial automaton of (partial) languages as the final coalgebra. Purely logical DES in the form of partial automata have also been studied using coalgebraic techniques in [10].

[1] Email: komenda@math.cas.cz

[2] This work was supported by the Academy of Sciences of the Czech Republic, Inst. Research Plan No. AV0Z10190503 and by 7FP EU.ICT project DISC, N.224498.

doi:10.1016/j.entcs.2010.07.020

Deterministic weighted automata and more generally deterministic transducers are typical instances of state transition structures that can easily be recasted as coalgebras of set functors. Actually, general nondeterministic weighted automata may be viewed as coalgebras as well, but it is difficult to put in use the corresponding final coalgebra [1], because the corresponding set functor involves powerset (even though sometimes only finite powerset is considered).

Deterministic (sequential) K-weighted automata with input alphabet A and weights in a semiring K are essentially partial Mealy automata [7], where the output alphabet is just replaced by a semiring K of weigths. The initial and output functions of Weighted automata (WA) are neglected or viewed as functions having values $0, 1 \in K$ defining initial and final states.

Mealy automata have been studied from a coalgebraic perspective in [18]. It has been shown that Mealy automata have final coalgebras, namely causal functions between streams (infinite sequences) over the input alphabet and streams over the output alphabet. This can be extended to the case of partial Mealy automata, which are Mealy automata with transition function that is only partially defined. Such a case is motivated by applications in control theory, where state-transition functions are partial functions corresponding e.g. to automata models of manufacturing systems. These are often represented by (timed) Petri nets [19] and can be translated into deterministic (weighted) automata using reachability graphs of (labelled) Petri nets, which are naturally partial automata.

It will be shown that the final coagebra of partial Mealy automata is formed by causal partially defined and length preserving functions between finite or infinite sequences over input alphabet and finite or infinite sequences over output set (semiring K).

WA model state transition systems with a quantitative information encoded by output values of transitions that can be e.g. cost of a transition, a timing information (like duration of executing a transition) or probability of a transition between given states. The underlying semiring is then typically the $(\mathbb{R} \cup \{\infty\}, min, +)$, $(\mathbb{R} \cup \{-\infty\}, max, +)$ or the probability semiring $(\mathbb{R}^+, +, \times)$, respectively.

As for timed transition systems, there are two basic ways of representing complex timed systems with concurrency (i.e. simultaneous occurrence of events) by WA: use of nondeterminism and synchronous product constructs. The first one relies on nondeterminism. Indeed, it is well known that unlike logical automata, nondeterministic WA have significantly higher expressive power compared to deterministic ones. More specifically, it is known from [9] that nondeterministic (max,+)-automata have a strong expressive power in terms of timed Petri nets: every 1-safe timed Petri net can be represented by a special (max,+) automaton, called heap model. The advantage of nondeterministic WA is that these are typically much smaller than their deterministic counterparts with the same behavior (if these happen to exist at all as finite WA). However, this approach is not easy to apply, because of problems with determinization and decidability issues [14].

The second way of modeling complex timed transition systems is using explicit product constructs. Partial Mealy automata with a suitable (number or interval based) semiring as an output set naturally model a simple class of timed transition systems: deterministic one clock timed automata or equivalently timed state graphs (machines). Timed state graphs are timed Petri nets, where no synchronization is allowed: every transition has exactly one upstream and one downstream place. The corresponding Mealy automata are simply formed using reachability graphs, where duration of a transition in the timed Petri net is exactly the output value of the associated transition in the Mealy automaton. In fact, such a Mealy automaton may be viewed as a one clock timed automaton [2], where the single clock is implicite and is replaced by the corresponding (exact) duration or interval duration from the underlying semiring (typically the so called (max,+) semiring $\mathbb{R}_{max} = (\mathbb{R} \cup \{-\infty\}, max, +)$ or the associated interval semiring). Such systems are intrinsically sequential, because the duration of consecutive events are simply added (the classical addition is the multiplication of $(\mathbb{R}\cup\{-\infty\}, max, +)$ to compute the execution times of event sequences (words). Therefore, explicit synchronous product is needed to model more complex timed behaviors (corresponding to multi-clock timed automata). Another reason why synchronous product of deterministic weighted automata is needed comes from applications, e.g. in manufacturing systems, where the underlying timed Petri nets models are formed by elementary timed state graphs that are composed using shared synchronization transitions. The overall system can then be modeled by the synchronous product of elementary timed state graphs (components) as in [19].

We have proposed a (truly) synchronous composition of deterministic (max,+)-automata based on tensor linear algebra and extended (multi-event set) alphabet in [12]. It turned out that there is no algebraic formula in terms of local (algebraic) behaviors (formal power series), but only using linear (automata) representations. In this paper a coalgebraic definition is given using coinductive definitions on stream functionals.

The paper is organized as follows. In Section 2 Mealy automata as coalgebras are recalled and partial Mealy automata are proposed. Final coalgebras of partial Mealy automata are studied and two theorems of functional stream calculus are stated. Section 3 is an introduction to deterministic (max,+) automata and their algebraic and coalgebraic behaviors. The notion of a timed language is recalled and compared to formal power series and causal stream functions. In section 4 coinductive definition of synchronous product of causal stream functions is proposed. An example is presented that illustrates the coalgebraic approach to concurrent timed systems. Finally, section 5 proposes a discussion and hints for future investigations.

2 Partial Mealy automata and deterministic weighted automata

In this section we recall from [18] Mealy automata as coalgebras and extend them to the case of partially defined transition function. A fundamental theorem of functional stream calculus is proposed that is the counterpart of fundamental theorem of (ordinary) stream calculus presented in [17]. Other properties of stream functions called stream functionals by analogy with mathematical analysis are stated.

Let A, K be arbitrary sets (typically finite and referred to as the set of inputs or events). The empty string will be denoted by λ. Further notation is $1 = \{\emptyset\}$ to denote a special one element set that will encode partiality of the transition function (when no transition is defined).

Definition 2.1 A partial Mealy automaton with inputs in A and outputs in K is the structure (S, t), where S is the set of states and the transition function is $t : S \to (1 + (K \times S))^A$. This function maps any state $s \in S$ a function $t(s) : A \to (1 + (K \times S))$ that associates to any input event A either a pair $\langle k, s' \rangle$ consisting of the new state and the output $k \in K$ or the symbol $\emptyset \in 1$. The latter case means that there is no transition from s to s' labeled by a and this is donoted by $s \overset{a}{\not\to}$.

Thus, Mealy automata with partially defined transition functions are coalgebras on the category Set of the functor $F : Set \to Set$ given by $F(S) = (1 + (K \times S))^A$.

The following notation, borrowed from [18], will be used: $s \overset{a|k}{\to} s'$ iff $t(s)(a) = \langle k, s' \rangle$. The fact that there is no transition labeled by a from s to any state is denoted by $s \overset{a}{\not\to}$. Since we work with deterministic Mealy machines this notation is justified and is equivalent to the absence transition labeled by a from s to any $s' \in S$.

Remark 2.2 If K is a semiring, i.e. K is endowed with addition and multiplication satisfying the semiring axioms, one may view partial Mealy automata as deterministic weighted automata. Typically, weighted automata are nondeterministic and have also quantitative initial and final functions that may represent cost, probability or duration (time) and associate to any state the initial or final value in the corresponding semiring (i.e. $(R \cup \{\infty\}, min, +)$-semiring, probability semiring or $(R \cup \{-\infty\}, max, +)$-semiring), respectively. In other cases only logical initial and final functions are considered that determine simply initial or final states. The initial state and final states play no role in this study. It is implicitly assumed that any state is final and that there is exactly one initial state. Note that unlike weighted automata, where the fact that there is no transition from s to s' labeled by a is expressed by the zero value of the corresponding output from the semiring K, in our case the absence of transitions is encoded using special symbol $\emptyset$. Only later we will get rid of this symbol and represent partiality of transition functions by (only) partially defined stream functionals.

The basic cornerstones of coalgebras of partial Mealy automata are stated: homomorphisms and bisimulation relations. A *homomorphism* between two partial Mealy automata $S = (S, t)$ and $S' = (S', t')$ is a function $f : S \to S'$ such that for all $s \in S$ and $a \in A$: if $s \overset{a|b}{\to} s'$ then $f(s) \overset{a|b}{\to} f(s')$, which can be captured by the equality $F(f) \circ t = t' \circ f$ corresponding to the commutative diagram below:

$$\begin{tikzcd}
(1+(K\times S))^A \arrow[d, "F(f)"] & S \arrow[l, "t"'] \arrow[d, "f"'] \\
(1+(K\times S'))^A & S' \arrow[l, "t'"']
\end{tikzcd}$$

Definition 2.3 A *bisimulation* between two partial Mealy automata $S = (S, t)$ and $S' = (S', t')$ is a relation $R \subseteq S \times S'$ such that for all $s \in S$ and $s' \in S'$: if $\langle s, s'\rangle \in R$ then

(i) $\forall a \in A: \; s \overset{a|b}{\rightarrow} q \;\Rightarrow\; s' \overset{a|b'}{\rightarrow} q'$ such that $\langle q, q'\rangle \in R$, and $b = b'$, and

(ii) $\forall a \in A: \; s' \overset{a|b'}{\rightarrow} q' \;\Rightarrow\; s \overset{a|b}{\rightarrow} q$ such that $\langle q, q'\rangle \in R$, and $b = b'$.

(iii) $\forall a \in A$: $s \overset{a}{\not\rightarrow}$ iff $s' \overset{a}{\not\rightarrow}$

As usual, we write $s \sim s'$ whenever there exists a bisimulation R with $\langle s, s'\rangle \in R$. This relation is the union of all bisimulations, i.e. the greatest bisimulation also called bisimilarity.

2.1 Final partial Mealy machine

First we consider causal functions between infinite sequences over A and infinite sequences over $1 + K$. Only later these will be formulated in a different way, where partiality of the transition function on stream functions will be expressed without using special symbol $\emptyset \in 1$. Partial functions between finite or infinite sequences over A and finite or infinite sequences over K will be considered instead.

The set of all infinite sequences (streams) over a set A is denoted by A^ω. Similarly, the set of finite or infinite sequences (streams) over a set A is denoted by A^∞, i.e. $A^\infty = A^\omega \cup A^+$, where A^+ stands for $A^* \setminus \{\lambda\}$. The empty string λ is excluded from A^∞, because Mealy automata have no output on empty input (unlike Moore automata).

The elements of stream calculus as coinductive study of infinite sequences over a semiring K are first recalled from [17]. It relies on the fact that streams from K^ω together with the initial value (the initial output from K, also called head of the stream) and the stream derivative (also known as tail of the stream) form the final coalgebra $(K^\omega, \langle head, tail\rangle)$ of the set functor $F(S) = K \times S$. Formally, for $s = (s(0), s(1), s(2), s(3), \ldots) \in K^\omega$: $head(s) = s(0)$ and $tail(s) = s' = (s(1), s(2), s(3), \ldots)$.

The notion of stream derivative applies to both infinite and finite sequences. For a sequence $s = (s(0), s(1), s(2), s(3), \ldots, s(k)) \in A^+$ its stream derivative, denoted $s' \in A^*$, is defined by $s' = (s(1), s(2), s(3), \ldots, s(k))$. Otherwise stated, for $k = 0, 1, 2, \ldots$ $s'(k) = s(k+1)$. Obviously, s' does not preserve the length of finite sequences.

For $a \in A$ and $\sigma = (\sigma(0), \sigma(1), \ldots, \sigma(k), \ldots) \in A^\infty$ the following notation is adopted: $a : \sigma = (a, \sigma(0), \sigma(1), \ldots)$.

The notion of causality [18] applies to functions between both finite and infinite sequences. f is causal means that for any $\sigma \in A^\infty$ the n-th element of the stream $f(\sigma) \in K^\infty$ depends only on the first n elements of $\sigma \in A^\infty$. Formally, $f : A^\infty \to K^\infty$ is *causal* if $\forall n \in \mathbb{N}$, $\sigma, \tau \in A^\infty$: $\forall i : i \leq n$: $\sigma(i) = \tau(i)$ then $f(\sigma)(n) = f(\tau)(n)$.

Unlike [18], where Mealy machines with complete transition functions are studied another property (that we call consistency of f) is required. Finally, $f : A^\infty \to (1+K)^\infty$ is called *consistent* if for any stream $\sigma \in A^\omega$ the sequence $f(\sigma) \in (1+K)^\omega$ has the property that the symbols $\emptyset$ must only be placed on the rightmost part of the stream. Formally, f is consistent if $\sigma \in A^\omega$: $f(\sigma)(k) = \emptyset$ then $f(\sigma)(n) = \emptyset$ for any $n > k$. The concept of consistency is close to prefix closedness of languages.

The initial output and functional stream derivative of f are defined in the same way as in [18]. Hence, $f[a] = f(a : \sigma)(0)$, which is well defined (independent of σ) due to causality. The functional stream derivative $f_a : A^\infty \to (1+K)^\infty$ is defined by $f_a(\sigma) = f(a : \sigma)'$. There seems to be a problem with defining stream derivative of a sequence of length 1. Fortunately, we only need the stream derivatives of finite sequences in the context of functional stream derivatives of [18], i.e. $f(a : \sigma)'$. Since $\sigma \in A^\infty$ is of length at least one, $a : \sigma$ is of length at least two, hence $f(a : \sigma)' \in (1+K)^\infty$ is either of length at least one or is undefined.

It has been shown in [18] that causal functions from streams over A to streams over K, where functional stream derivative plays the role of the transition function form final coalgebra of Mealy machines with complete transition functions.

Now we are ready to propose the following (universal) partial Mealy automaton.

Definition 2.4 Let us define the partial Mealy automaton $\mathcal{F} = (\mathcal{F}, t_\mathcal{F})$ with the carrier set $\mathcal{F} = \{f : A^\omega \to (1+K)^\omega \mid f \text{ is causal and consistent}\}$.

The first output and functional stream derivative endow $\mathcal{F}$ with partial Mealy automaton structure $(\mathcal{F}, t_\mathcal{F})$, where $t_\mathcal{F} : \mathcal{F} \to (1 + (K \times \mathcal{F}))^A$ is defined by

$$t_\mathcal{F}(f)(a) = \begin{cases} \langle f[a], f_a \rangle & \text{if } f[a] \neq \emptyset \in 1, \\ \emptyset & \text{otherwise,} \end{cases}$$

The definition of derivative can be extended to strings using the classical chain rule: for $w \in A^*$ and $a \in A$: $(f_w)_a = f_{wa}$.

Below we point out that to any state of any partial Mealy automaton we can associate its behavior from $\mathcal{F}$ such that the mapping $f : S \to \mathcal{F}$ be a homomorphism of partial Mealy automata.

Proposition 2.5 *The partial Mealy automaton $(\mathcal{F}, t_\mathcal{F})$ is a final partial Mealy automaton: for every partial Mealy automaton (S, t) (with inputs in A and outputs in K), there exists a unique homomorphism $l : (S, t) \to (\mathcal{F}, t_\mathcal{F})$.*

Proof. For any Mealy automaton (S, t) we define a function $l : S \to \mathcal{F}$. It associates to a $s_0 \in S$ the function $l(s_0) : A^\omega \to (1+K)^\omega$ in the following way: for $\sigma \in A^\omega$ and $n \in \{0,1,2,\ldots\}$, the sequence of transitions corresponding to σ is considered (if it exists), i.e. $s_0 \overset{\sigma(0)|k_0}{\to} s_1 \overset{\sigma(1)|k_1}{\to} s_2 \ldots \overset{\sigma(n)|k_n}{\to} s_{n+1}$. We define in this case $l(s_0)(\sigma)(n) = k_n$. If there is no such a transition along the path labeled $\sigma(0)\ldots\sigma(n)$, then we put $l(s_0)(\sigma)(n) = \emptyset$. Otherwise stated, l maps any input sequence $\sigma \in A^\omega$ to the stream $(k_0, k_1, k_2, \ldots) \in (1+K)^\omega$ of outputs observed along this input starting in s_0. Then no transition possible starting from σ_m is expressed by putting special symbols encoding "empty" obervation: $k_l = \emptyset$ for $l \geq m$.

It is immediately seen that $l(s_0)$ is consistent, because clearly $l(s_0)(\sigma)(n) = \emptyset$ for any $n > m$ whenever $l(s_0)(\sigma)(m) = \emptyset$. It is not difficult to verify that $l(s_0)$ is causal and that l is a homomorphism, which is moreover a unique one (up to isomorphism). □

The stream function $l(s_0)$ above is called the (input-output) behavior of s_0. Final coalgebra $\mathcal{F}$ has the property that bisimulation on $\mathcal{F}$ implies (and henceforth is equivalent to) equality. This opens the possibility of proving equality of two causal and consistent stream functions $f, g \in \mathcal{F}$ by coinduction, which amounts to showing that $f \sim g$: there exists a bisimulation relation $R \subseteq \mathcal{F} \times \mathcal{F}$ such that $\langle f, g \rangle \in R$. Since the transition function of $\mathcal{F}$ is defined using the tuple of the first output and functional stream derivative, the proof of R being a bisimulation (used further) consists of two steps. Firstly, it is shown that first outputs for any input coincide on all related pairs of stream functions, and secondly it is to be shown that the functional stream derivatives with respect to all input events are again related by R.

Now the final coalgebra $\mathcal{F}$ is formulated in an equivalent way, where $\emptyset$ is not needed, but both infinite and finite sequences of inputs and outputs are considered and the function between them must preserve their length. We will consider causal partial (partially defined) functions from finite or infinite sequences over A to finite or infinite sequences over K that are length preserving, i.e. finite sequences over A are mapped to finite sequences over K of the same length and infinite sequences over A are mapped to infinite sequences over K. It is easily seen that $\mathcal{F}$ is isomorphic to the following structure:

$$\mathcal{F}_\infty = \{f : A^\infty \to K^\infty, f \text{ length preserving,causal with } dom(f) \text{ prefix-closed}\}.$$

Note that the output set $1+K$ is replaced by K and f is a partial function between finite or infinite sequences over input and output set. The consistency of $f : A^\omega \to (1+K)^\omega$ enables to recast f as an element of $\mathcal{F}_\infty$.

It follows from the construction below. First, it is shown how to obtain from $f \in \mathcal{F}$ an element of $\mathcal{F}_\infty$. Since f is consistent, there are two possibilities: either for $\sigma \in A^\omega$ and any $n \in \mathbb{N}$ we have $f(\sigma)(n) \in K$, i.e. $f(\sigma)(k) \neq \emptyset$, or there is a $k \in \mathbb{N}$ such that $f(\sigma)(k) = \emptyset$. In the former case f is automatically an element of $\mathcal{F}_\infty$. In the latter case the consistency of f means that $f(\sigma)(n) = \emptyset$ for $n > k$. Then it is useless to evaluate f in the whole infinite sequence σ. It is then sufficient to consider only finite words like $\sigma(0)\sigma(1)\ldots\sigma(k) \in A^*$ and the corresponding finite words in the outputs $f(\sigma)(0).f(\sigma)(1)\ldots f(\sigma)(k) \in K^*$. It is clear that such a mapping f is only partial (as we forget the value of f on suffixes of $\sigma(0)\sigma(1)\ldots\sigma(k)$), and that f is length preserving. Let us not here that intuitively, $\sigma(0)\sigma(1)\ldots\sigma(k) \in A^*$ is in the (prefix-closed) language of the underlying partial Boolean automaton that forgets the outputs, while $\sigma(0)\sigma(1)\ldots\sigma(k+1)$ is already out of this prefix-closed language. It is immediately seen that $dom(f)$ is prefix-closed.

Conversely, to any length preserving and causal $f : A^\infty \to K^\infty$ with prefix-closed domain we can construct a causal and consistent mapping $f : A^\omega \to (1+K)^\omega$. Indeed, it is simply sufficient to extend the maximal (wrt prefix-order) finite words for which f is defined to infinite words and complete the image sequences by symbols $\emptyset$ that are placed on the rightmost part of the streams $f(\sigma)$. Naturally, f is again causal and consistent.

An important observation is that for $f \in \mathcal{F}_\infty$ we have in fact $f[a] = f(a)(0)$ whenever f is defined for $a \in A$. Otherwise, $f[a]$ is undefined and there is no a-transition in $\mathcal{F}_\infty$ from f.

For each length preserving and causal function $f \in \mathcal{F}$ we define the initial (first) value (of output) corresponding to $a \in A$ simply by $f[a] = f(a)(0)$. Also, the functional stream derivative of f (with respect to input a) is defined as the function $f_a : A^\infty \to (1+K)^\infty$ given by $f_a(s) = f(a : s)'$ if it is defined. Clearly, f_a is again causal and preserves the length of s, because prefixig a finite sequence by a increases the length by 1 and f keeps the length, but the derivatives reduces it back to the original length. The transition function $t_{\mathcal{F}} : \mathcal{F} \to (1 + (K \times \mathcal{F}))^A$ is defined using the first output function and the functional stream derivative, similarly $t_{\mathcal{F}_\infty} : \mathcal{F}_\infty \to (1 + (K \times \mathcal{F}_\infty))^A$ is defined by

$$t_{\mathcal{F}_\infty}(f)(a) = \begin{cases} \langle f[a], f_a \rangle & \text{if } f[a] \text{ is defined} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

2.2 Fundamental properties of stream functions

It is natural to call functions between streams by stream functionals to stress the analogy with functional analysis. First we recall other elementary concepts from stream calculus for streams over a semiring $K = (K, \oplus, \otimes, 0, 1)$. The constant stream corresponding to $r \in K$ is given by $[r] = (r, 0, \ldots)$. The notation $X = (0, 1, 0, \ldots)$ is important to describe any stream using constant streams, addition and multiplication. The addition (sum) of streams is defined using addition of K: For $\sigma, \tau \in K^\omega$:

$$(\sigma \oplus \tau)(n) = \sigma(n) \oplus \tau(n)$$

and Cauchy (convolution) multiplication of streams given by

$$(\sigma \otimes \tau)(n) = \bigoplus_{k=0}^{n} \sigma(k) \otimes \tau(n-k).$$

The symbol $\otimes$ for multiplication of streams is often left out as in the classical calculus, cf. $Xf_{\sigma(0)}(\sigma')(0)$ below meaning $X \otimes f_{\sigma(0)}(\sigma')(0)$ etc. The n-th Cauchy power of X, i.e. $X^n = (0,\dots,0,1,0,\dots)$ with 1 being placed on the $n+1$-st place. Finally the notation $\sigma^{(k)}$ is used to denote the k-th stream derivative of σ.

The following theorem is provided that is the counterpart of fundamental theorem of stream calculus in functional stream calculus. Fundamental theorem of stream functionals is stated in Theorem 2.6. Let us note that similarly as in the stream calculus the sum $\oplus$ is formal, although it is well defined, because there is only one element per component in $\oplus$ below.

Theorem 2.6 *For any $f \in \mathcal{F}$ and $\sigma = (\sigma(0), \sigma(1), \dots, \sigma(k), \dots) \in A^\omega$ we have:*

$$f(\sigma) = f(\sigma)(0) \oplus Xf_{\sigma(0)}(\sigma')(0) \oplus \dots X^k f_{\sigma(0)\dots,\sigma(k-1)}(\omega^{(k)})(0) \oplus \dots$$

or equivalently,

$$f(\sigma) = f[\sigma(0)] \oplus Xf_{\sigma(0)}[\sigma(1)] \oplus \dots X^k f_{\sigma(0)\dots,\sigma(k-1)}[\sigma(k)] \oplus \dots$$

Proof. It follows from the fundamental theorem of stream calculus, which states that for any $\sigma \in A^\omega$ it holds that $\sigma = \sigma(0) \oplus X.\sigma'$, which can be extended to $\sigma = \sigma(0) \oplus X\sigma'(0) \oplus X^2\sigma'(0) \oplus \dots$ Similarly, it sufficies to show that $f(\sigma) = f(\sigma)(0) \oplus Xf_{\sigma(0)}(\sigma')$, which is a direct consequence of the fundamental identity on K^ω for $f(\sigma)$: $f(\sigma) = f(\sigma)(0) \oplus Xf(\sigma)'$. Indeed, $\sigma = \sigma(0) : \sigma'$, hence by definition of functional stream derivative we get: $f_{\sigma(0)}(\sigma') = f(\sigma(0) : \sigma')' = f(\sigma)'$. Hence, $f(\sigma) = f(\sigma)(0) \oplus Xf_{\sigma(0)}(\sigma')$.

$\square$

In the proof of Theorem 2.6 we did not make use of coinduction, but implicitly : it relies on fundamental theorem of stream calculus, which can be proven by coinduction. Let us also mention that similar fundamental theorem holds for functionals from $\mathcal{F}_\infty$. Now the set $\mathcal{F}_\infty$ is considered. In the next results partiality of the functionals from $\mathcal{F}_\infty$ is important. Functionals from $\mathcal{F}_\infty$ have interesting properties and some of them are proven below by coinduction on stream functions. Some of them are listed below. The fact that we can evaluate these functionals on finite words that are prefixes of (potentially) infinite words (streams) has interesting consequences. For instance, the lemma below.

Lemma 2.7 *For any $f \in \mathcal{F}_\infty$, $\omega \in A^\infty$, and $a \in A$: $f(a) : f_a(\omega) = f(a\omega)$. More generally, for any $u \in A^+$ and $\omega \in A^\infty$: $f(u) : f_u(\omega) = f(u\omega)$.*

Proof. First we stress that the $f(a) : f_a(\omega)$ is an element of K^∞. Hence, the equality can be shown by coinduction on streams [17] extended to finite and infinite sequences, which is the final coalgebra of partial stream automata given by the Set functor $F : S \to 1 + (K \times S)$. Put

$$R = \{\langle f(a) : f_a(\omega), f(a\omega)\rangle \cup \langle \sigma, \sigma\rangle \mid f \in \mathcal{F}_\infty,\ \sigma \in K^\infty,\ \text{and } a \in A.\}$$

This amounts to show that the heads (initial values) are the same and that the stream derivatives are also related by R. Firstly, $[f(a) : f_a(\omega)](0) = f(a)$ and $f(a\omega)(0) = f(a)(0) = f(a)$, because f is causal. Secondly, $\{f(a) : f_a(\omega)\}' = f_a(\omega)$ and $f(a\omega)' = f_a(\omega)$ from the very definition of functional stream derivative. It is also easy to see that $f(a) : f_a(\omega)$ can not make further transition iff $f(a\omega)$ can not (namely iff $f(a)$ is undefined), which shows the partial stream counterpart of (iii) of Definition 2.3. □

An interesting and useful observation is that the first output function at first input $f[a] = f(a)(0)$ can be seen as a simple stream functional. Indeed, the initial output function can be seen as a particular partial stream functional defined by

$$f^\infty[a](\sigma) = \begin{cases} f[a] & \text{if } \sigma = a, \\ \text{undefined} & \text{otherwise: } \sigma \neq a, \end{cases}$$

The following concatenation like multiplication (denoted by $\odot$) of a stream functional g with this special stream functional $f^\infty[a]$ on the left, helps formulating another fundamental identity of functional stream calculus.

Definition 2.8 For any $f, g \in \mathcal{F}_\infty$, $\sigma = (\sigma(0) : \sigma') \in A^\infty$, and $a \in A$ we define

$$(f^\infty[a] \odot g)(\sigma(0) : \sigma') = \begin{cases} f(\sigma(0)) : g(\sigma') & \text{if } a = \sigma(0) \in dom(f), \\ \text{undefined} & \text{otherwise,} \end{cases}$$

Note that the multiplication on the right is just stream concatenation on K^∞ following the notation of [17]. We make the convention that $f(a) : g(\sigma') = f(a)$ if $g(\sigma')$ is undefined. Addition on streams from K^∞ induces addition on $\mathcal{F}_\infty$. Simply, one defines for $f_i \in \mathcal{F}_\infty$, $i \in I$: $(\bigoplus_{i\in I} f_i)(\sigma) = \bigoplus_{i\in I}(f_i(\sigma))$. Then we can write:

Theorem 2.9 *For any $f \in \mathcal{F}_\infty$ we have: $f = \bigoplus_{a\in A} f^\infty[a] \odot f_a$.*

Proof. It can be shown by coinduction on $\mathcal{F}_\infty$. We put

$$R = \{\langle \bigoplus_{a\in A} f^\infty[a] \odot f_a, f\rangle \cup \langle f, f\rangle \mid f \in \mathcal{F}_\infty \text{ and } a \in A\}.$$

Then R is a bisimulation on $\mathcal{F}_\infty$. Indeed, for any $b \in A$ we get $(f^\infty[a] \odot f_a)[b] = (f^\infty[a] \odot f_a)(b)(0) = (f^\infty[a](b)$. Let us observe that according to definition of $f^\infty[a]$ we have either $f^\infty[a](b) = f[b]$ or it is undefined, depending on $b = a$ or not. Hence, $\bigoplus_{a\in A}(f^\infty[a] \odot f_a)[b] = f[b]$ and the first outputs are equal for $\bigoplus_{a\in A}(f^\infty[a] \odot f_a)$ and f.

Now, let $(\bigoplus_{a\in A} f^\infty[a] \odot f_a \xrightarrow{b|k} f'$. Then $f'(\sigma) = (\bigoplus_{a\in A} f^\infty[a] \odot f_a)_b(\sigma) = \bigoplus_{a\in A}\{(f^\infty[a] \odot f_a)(b\sigma)\}' = \{f[b] : f_b(\sigma)\}' = f_b(\sigma)$. Again, $(f^\infty[a](b) = f[b]$ in case $b = a$ has been used. Finally, it is obvious that $\bigoplus_{a\in A} f^\infty[a] \odot f_a \overset{a}{\not\to}$ iff $f \overset{a}{\not\to}$ (namely iff $f[a]$ is undefined), which shows (iii) of Definition 2.3.

Hence, $\langle(\bigoplus_{a\in A} f^\infty[a] : f_a)_b, f_b\rangle \in R$, which was to be shown. □

This equality may be viewed as a functional stream counterpart of the fundamental theorem of (multivariable) formal power series (behaviors of Moore automata): $s = s(\lambda) + \sum_{a\in A} a.s_a$ for $s : A^* \to K$. Therefore, multiplying a stream functional f by the elementary functional given by $f[a]$, i.e. $f^\infty[a]$, in the sense of definition 2.8 can be seen as a functional counterpart of formal power series integration, which is given by $a\sigma$, i.e. $X\sigma$ for monovariable streams over $X = (0, 1, 0, \ldots)$. This is expressed in the Proposition below.

Proposition 2.10 *For any $f \in \mathcal{F}_\infty$ and $a \in A$: $(f^\infty[a] \odot f)_a = f$*

Proof. The equality is shown by coinduction on $\mathcal{F}_\infty$. Let

$$R = \{\langle (f^\infty[a] \odot f)_a, f\rangle \cup \langle f, f\rangle \mid f \in \mathcal{F} \text{ and } a \in A\}.$$

Then R is a bisimulation on $\mathcal{F}_\infty$. Indeed, for any $b \in A$ we get $(f^\infty[a] \odot f)_a[b] = \{(f^\infty[a] \odot f)(ab)\}'(0) = (f^\infty[a] \odot f)(ab)(1)=\{f(a) : f(b)\}(1) = f(b)$.

Now, let $(f^\infty[a] \odot f)_a \overset{b|k}{\to} f'$. Then $f'(\sigma) = (f^\infty[a] \odot f)_{ab}(\sigma) = \{(f^\infty[a] \odot f)(ab\sigma)\}'' = \{f(a) : f(b\sigma)\}'' = f(b\sigma)' = f_b(\sigma)$. Hence, $(f^\infty[a] \odot f)_{ab} = f_b$ and therefore $\langle (f^\infty[a]\odot f)_{ab}, f_b\rangle \in R$. Also, it is easy to see that $(f^\infty[a]\odot f)_a \overset{a}{\not\to}$ iff $f \overset{a}{\not\to}$ (namely iff $f[a]$ is undefined), which shows (iii) of Definition 2.3.

□

Remark 2.11 In section 4 we need a semiring structure on K in order to introduce the synchronous product operation on causal and length preserving functions that are behaviours of deterministic time-weighted automata.

Finally, let us mention that the behavior of Mealy automata are typically described in the literature as formal power series $f : A^+ \to K$. Still we prefer to work with $\mathcal{F}$, because as is shown in the next section, in the case of deterministic time-weighted automata coalgebraic behaviors are similar to timed languages from timed automata theory.

3 Deterministic weighted automata and timed languages

In this section deterministic (max,+) and interval automata without initial and final delays are considered and three representations of their behaviors are discussed: formal power series, timed languages and functions between finite or infinite sequences of inputs and outputs.

Let us start with the definition of deterministic K-weighted automata. Let $K = (K, \oplus, \otimes)$ be a semiring.

Definition 3.1 A deterministic K-weighted automaton over the input alphabet A and with weights in K is the Mealy automaton (S, t), where S is the set of states and the transition function is $t : S \to (1 + (K \times S))^A$.

Two important cases of semiring K are considered in this paper. A deterministic *(max,+)-automaton* is a deterministic K-weighted automaton with $K = \mathbb{R}_{max} =$

$(R \cup \{-\infty\}, max, +)$. The zero element, i.e. $-\infty$ of $\mathbb{R}_{max}$ is denoted by ε in accordance with idempotent semiring notation [8] and the unit element is denoted by $e = 0$. A deterministic *interval automaton* is a deterministic K-weighted automaton with $K = \mathcal{I}_{max}^{max} = (\mathbb{R} \times \mathbb{R} \ \cup (-\infty, -\infty), \oplus, \otimes)$, where $\oplus$ is the componentwise maximum and $\otimes$ is the componentwise (conventional) addition.

At the first sight our deterministic K-weighted automata might seem very different from (non)deterministic K-weighted automata in algebra, which are defined by $G = (Q, A, \alpha, \mu, \beta)$, where Q is a finite set of states, A is the set of events, and the linear triple consists of $\alpha : Q \to K$, $t : Q \times A \times Q \to K$, and $\beta : Q \to K$, called input, transition, and output delays, respectively. Let us recall that t can be algebraically viewed as a collection of matrices

$$\mu : A \to K^{Q \times Q},\ \mu(a)_{q\,q'} \triangleq t(q, a, q').$$

Since the definition μ can be extended from $a \in A$ to $w \in A^*$ using the morphism property, i.e.

$$\mu(a_1 \dots a_n) = \mu(a_1) \otimes \dots \otimes \mu(a_n),$$

μ is often called a morphism matrix. Such a triple (α, μ, β) is called a linear representation of G. In this algebraic representation there is no need of using special symbols to express partiality of the transition function, because $t(q, a, q') = 0 \in K$ means there is no transition from q to q' labeled by a. The transition function associates to a state $q \in Q$, a discrete input $a \in A$ and a new state $q' \in Q$, an output value $t(q, a, q') \in K$ corresponding to the $a-$transition from q to q'.

As our K-weighted automata (viewed as coalgebras) are assumed to be deterministic, there is exactly one initial state and t is deterministic, i.e. $t : Q \times A \to (1 + K \times Q)$. Note that in this deterministic transition function t the set $\emptyset \in 1$ is needed to encode the partiality of the transition function, unlike the nondeterministic transition function, where the zero element of K, i.e. $0 \in K$ encodes the fact that there is no transition between two given states with a given label. It is immediately seen that this deterministic transition function can be recasted in the coalgebraic form in terms of the set functor $F(S) = (1 + (K \times S))^A$, cf. Definition 3.1.

Essentially, our attention is restricted to transition function, while initial and final weights (called delays in timed systems) are discarded, or at least α and β take their values in the Boolean subsemiring of K: e.g. $\forall q \in A : \alpha(q) \in \{0, 1\}$ meaning $\alpha(q) = 1$ iff q is the initial state. Initial and final state play no role in our study, because from a coalgebraic perspective any state can play a role of an initial state in the sense that the behavior homomorphism $l : S \to \mathcal{F}$ evaluated in $s \in S$ gives the behavior of S, where s is the initial state.

Below it is assumed that $K = \mathbb{R}_{max}$. From an algebraic viewpoint, behaviors of timed systems are timed languages or formal power series. Below the notion of a timed language is recalled from the theory of timed automata [2].

Definition 3.2 A timed word s_t is a (finite or infinite) sequence over the alphabet $A \times \mathbb{R}$, i.e. $s_t \in (A \times \mathbb{R})^\infty$, where $(\sigma_1, t_1) \dots (\sigma_n, \sigma_n) \dots$ means that the execution time of an event a_i is achieved at time t_i, $i = 1, 2, \ldots$. A timed language is a subset of timed words, i.e. $L_t \subseteq (A \times \mathbb{R})^*$.

It is easy to see the relationship between elements of the final coalgebra $\mathcal{F}$ or its equivalent presentation $\mathcal{F}_\infty$ from the previous section and timed languages. In fact, timed languages give the cumulated execution time of a sequence, which is the sum of durations of individual events in the sequence. This subsumes that the multiplication of the underlying output alphabet, which is the semiring $\mathbb{R}_{max}$ is the conventional addition. In particular, it means that the sequence of execution times $t_1 \ldots t_n \ldots$ from Definition 3.2 is nondecreasing.

On the contrary, for any partial, length preserving, and causal function $f : \mathcal{F}_\infty$ and $\sigma \in A^\infty$ the value $f(\sigma)$ gives the sequence of duration times of individual events from σ, which is naturally not nondecreasing in general. For a given $f \in \mathcal{F}_\infty$ it is easy to obtain the corresponding timed language by simply making the sum of the duration of consecutive events from the initial event up to a given one. For instance, the function $f : A^+ \to \mathbb{R}^+_{max}$ that maps the sequence a, b, c, b to the sequence $1, 2, 4, 3$ and is only defined on nonempty prefixes of a, b, c, d (e.g. a, b is mapped to the time sequence $1, 2$) corresponds to the finite timed language given by a single timed word $(a, 1)(b, 3)(c, 7)(b, 10)$. The timed words and such simple stream functionals are easy to obtain from one another. However, timed languages (subsets of timed words) are strictly more expressive than our stream functionals. This is natural, because timed words can express concurrent timed behaviors of general timed automata, while stream functionals are tailored to sequential (single clock) timed systems. Still it will be shown in section 4 that there is a class of concurrent timed behaviors that can be expressed by stream functionals using synchronous product based on extended alphabets.

Let us note that timed words are even closer to formal power series than stream functionals, but in general formal power series may hide some timing information. In the example above, the corresponding formal power series is $1a \oplus 3ab \oplus 7abc \oplus 10abcb$. However, for another formal power series $1a \oplus 7abc \oplus 10abcb$ there is no information about the execution time of ab, but only the one about abc is specified and we only know that ab is executed in time interval $(1, 7)$.

But timed languages have no such a nice and rich structure as partial, length preserving, and causal functions from $\mathcal{F}_\infty$. This feature is important for our main goal: coinductive definition of synchronous product of deterministic weighted automata. Therefore, we only work with behaviors from $\mathcal{F}_\infty$ in the rest of this paper.

4 Behaviors of concurrent deterministic (max,+) automata

In this section the main result of this paper is presented: coinductive definition of synchronous product of behaviors of deterministic time-weighted automata, i.e. functions from $\mathcal{F}_\infty$, is proposed and discussed in detail.

Let us first recall the automaton based definition of synchronous product proposed in [12]. It is assumed that a distributed timed system is given by two deterministic (max,+)-automata. Let $G_1 = (S_1, t_1)$ and $G_2 = (S_2, t_2)$ be two (max,+)-automata defined over local alphabets A_1 and A_2. Then associated natural projections are $P_1 : (A_1 \cup A_2)^* \to A_1^*$ et $P_2 : (A_1 \cup A_2)^* \to A_2^*$. We also need the Boolean matrices associated to morphism matrices :

$$[B\mu(a)]_{ij} = \begin{cases} e = 0, & \text{if } [\mu(a)]_{ij} \neq \varepsilon \\ \varepsilon = -\infty, & \text{else} \end{cases}$$

In order to avoid heavy notation, $B\mu(a)$ is in the sequel denoted by $B(a)$. This notation can be extended to a (Boolean) morphism on words from $B : A^* \to \{0, -\infty\}^{n \times n}$ using morphism property $B(a_1 \dots a_n) = B(a_1) \dots B(a_n)$.

The synchronous composition of two (max,+)- automata is based on an extended alphabet composed of two types of events. Firstly, it includes all shared events. Secondly, it includes pairs of local sequences in between the synchronization event. Formally, $\mathcal{A} = (A_1 \cap A_2) \cup (A_1 \setminus A_2)^* \times (A_2 \setminus A_1)^*$. The problem is that the alphabet actually depends on the particular distributed timed system under consideration in the sense that not all pairs of local sequences (which would make the extended alphabet infinite) need to be included in $\mathcal{A}$. It suffices to include those pairs from $(A_1 \setminus A_2)^* \times (A_2 \setminus A_1)^*$ that are actually executed by the automata. In fact, we need to include in $\mathcal{A}$ exactly the pairs of maximal local strings (maximality is with respect to the prefix order) in between two consecutive synchronization events. $\mathcal{A}$ can in general be infinite, but only in the case, where at least one local automaton G_1 or G_2 has loops consisting of private events only from $A_1 \setminus A_2$ or $A_2 \setminus A_1$, respectively, and local systems are unable to get synchronized. Otherwise stated, if any loop in local subsystems contains at least one shared (synchronization) event then the extended alphabet $\mathcal{A}$ for a given distributed timed system is finite.

In order to define the synchronous product on behaviors from $\mathcal{F}_\infty$ it is necessary to use extended alphabet $\mathcal{A}$.

The Kronecker (tensor) product of matrices denoted by $\otimes^t$ is involved. If $A = (a_{ij})$ is a $m \times n$ matrix and B is a $p \times q$ matrix over K, then their *Kronecker (tensor) product* $A \otimes^t B$ is the $mp \times nq$ block matrix

$$(A \otimes^t B)_{ik,jl} = a_{ij} \otimes b_{kl}.$$

The parallel products of weighted automata à la A. Arnold [3] or P. Bucholtz [4] is not suitable for timed systems, because if these are applied to(max,+)-automata, their product as product of weighted automata remains a sequential model.

In the logical setting trace theory has been developed, where events that may occur simultaneously are related by independence relation. However, it is not clear how to extend the trace theory into the timed transition systems setting. Let us recall that classical composition of weighted automata [3] is simply given by tensor product of their linear representations. This corresponds to classical synchronous product of underlying Boolean automata, but the duration of a transition in the synchronous product is the product (i.e. conventional sum in $\mathbb{R}_{max}$)of the durations of participating transitions. This is not suitable for timed systems modeled by (max,+)-automata, unless there is no concurrency between events: e.g. the alphabets of subsystems are equal. Indeed, one would need to distinguish simultaneously executed events or strings and in the extreme case the duration of a global string is the sum of durations of corresponding (projected or local) strings. Therefore, definition below has been proposed, which corresponds to the intuition that in between two synchronizing transitions the local strings are executed in parallel, i.e. the duration of a string v of non shared events equals the maximum of durations of its projections $P_1(v)$ and $P_2(v)$ in local automata. A much simple coalgebraic counterpart of this definition will be given later in this section.

Definition 4.1 *Synchronous Composition* of (max,+)-automata $G_1 = (Q_1, A_1, \alpha_1, \mu_1, \beta_1)$ and $G_2 = (Q_2, A_2, \alpha_2, \mu_2, \beta_2)$, is the following interval automaton defined over the alphabet

$$\mathcal{A} = (A_1 \cap A_2) \cup [(A_1 \setminus A_2)^* \times (A_2 \setminus A_1))^*]$$

$$G_1 \| G_2 = \mathcal{G} = (Q_1 \times Q_2, \mathcal{A}, \alpha, \mu, \beta)$$

with $Q_1 \times Q_2$ set of states, $\mathcal{A}$ set of events, $\alpha = \alpha_1 \otimes^t \alpha_2$ the initial delay, $\mu : \mathcal{A}^* \to \mathbb{R}_{max}^{|Q| \times |Q|}$ the morphism matrix and $\beta = \beta_1 \otimes^t \beta_2$ final delay. The morphism matrix is defined by :

$$\mu(v) = \begin{cases} \mu_1(v) \otimes^t B_2(v) \oplus B_1(v) \otimes^t \mu_2(v), & \text{if } v = a \in A_1 \cap A_2 \\ \mu_1(P_1(v)) \otimes^t B_2(P_2(v)) \oplus B_1(P_1(v)) \otimes^t \mu_2(P_2(v)), & \text{if } v = (P_1(v), P_2(v)) \in \\ & (A_1 \setminus A_2)^* \times (A_2 \setminus A_1)^* \end{cases}$$

Let us now explain the intuition behind this seemingly complicated definition. The interval automata G_1 and G_2 are synchronized over the shared events set: $A_1 \cap A_2$, but in between two consecutive synchronizations the automata G_1 and G_2 are free to execute their respective private events belonging to $A \setminus (A_1 \cap A_2)$. These private events are represented by pairs of corresponding local strings $P_1(v) \in (A_1 \setminus A_2)^*$ and $P_2(v) \in (A_2 \setminus A_1)^*$. Not all events of $\mathcal{A}$ are needed in concrete synchronous products. In fact, only those pairs of local events are included that actually occur in local subsystems between two synchronizing events. The extended alphabet $\mathcal{A}$ is still potentially infinite, which is the main drawback of our approach. Fortunately, this may only happen if there are loops of private events in one of the subsystems and the subsystems are unable to synchronize. If this case is excluded a finite extended alphabet $\mathcal{A}$ can be found.

In [12] we could not find any algebraic definition compatible with this automata definition that would be based on formal power series $l_1 = l(G_1)$ and $l_1 = l(G_1)$, i.e. independent of the linear representation G_1 of l_1 and G_2 of l_2. A definition for behaviors is only possible if automata representations are fixed, but there is no algebraic definition independent of automata representation. Yet, the formula for behavior of the synchronous product from [12] is very complex and not practical to use, because for a given word over $A_1 \cup A_2$ it is the sum of a number of terms that is exponential in the number of synchronization events in this word. Still, even in the case of infinite alphabet $\mathcal{A}$, we have been able to compute the (algebraic) behavior of $G_1\|G_2$ on finite words from $A = A_1 \cup A_2$. Indeed, any $w \in A^*$ can be decomposed as $w = v_0 a_1 v_1 \dots a_n v_n$, where $a_i \in A_1 \cap A_2$, $i = 1, \dots, n$ are shared (synchronization) events and $v_i \in (A \setminus (A_1 \cap A_2))^*$, $i = 0, \dots, n$ are sequences of private local events. For such a v_i the corresponding local strings in G_1 and G_2 are given by natural projections $P_1(v_i) \in A_1^*$ and $P_2(v_i) \in A_2^*$, respectively. This way, any word over distributed (global) event set A^* can be seen as the word $w = P_1(v_0) \times P_2(v_0) a_1 P_1(v_1) \times P_2(v_1) \dots a_n P_1(v_n 0) \times P_2(v_n)$ over the extended alphabet $\mathcal{A}$. The duration of private strings $v_i \in (A \setminus (A_1 \cap A_2))^*$ is simply given by the maximum of the durations of local strings $P_1(v)$ and $P_2(v)$.

In this paper such a definition for behaviors is given using finality of $\mathcal{F}_\infty$ formed by causal and length preserving functions between $\mathcal{A}^\infty$ and K^∞ with prefix closed domains. They are endowed by partial Mealy automaton structure as in section 2, where A is just replaced by the extended alphabet $\mathcal{A}$. The first output function of $(l_1\|l_2)$ will be defined using first output functions of l_1 and l_2 extended to strings. The following concept is now needed. For $l_i \in \mathcal{F}_\infty$ over A_i and $v_i = a_1 \dots a_k \in A_i^+$ we define for $i = 1, 2$:

$$(l_i)[v_i] = (l_i)[a_1] \otimes (l_i)_{a_1}[a_2] \otimes \dots \otimes (l_i)_{a_1 \dots a_{k-1}}[a_k].$$

This is needed, because the whole local v_i string playing the role of $P_i(v)$ from definition below is executed in G_i in a sequential way: the duration of its execution is simply the usual sum, i.e. $\otimes$, of execution times of individual events $a_1, \dots a_k$ from v_i.

Definition 4.2 Define the following binary operation on $\mathcal{F}_\infty$ over $\mathcal{A}$: for $l_1, l_2 \in \mathcal{F}_\infty$ and $\forall v \in \mathcal{A}$:

$$(l_1\|l_2)_v = (l_1)_{P_1(v)} \| (l_2)_{P_2(v)} \text{ and}$$
$$(l_1\|l_2)[v] = l_1[P_1(v)] \otimes Bl_2[P_2(v)] \oplus Bl_1[P_1(v)] \otimes l_2[P_2(v)].$$

Note that equivalently one can write:

$$(1) \qquad (l_1\|l_2)[v] = \begin{cases} max(l_1[P_1(v)], l_2[P_2(v)]) & \text{if } l_i[P_i(v)] \neq \varepsilon \text{ for } i = 1, 2 \\ \varepsilon & \text{else, i.e. } \exists i = 1, 2 : l_i[P_i(v)] = \varepsilon \end{cases}$$

This definition is similar to the coinductive definition of synchronous product [15] of partial languages (behaviors of partial automata). Indeed, our functional input derivative for a shared event has the same form as the input derivative for languages. As for private events, the extended alphabet contains strings of these events (typically a finite number of them), but clearly our definition of functional input derivative is formally of the same form as the input derivative from the language definition [15] extended to strings. Namely, for partial languages $L_1 = (L_1^1, L_1^2)$, $L_2 = (L_2^1, L_2^2)$, and $w \in A^*$ we have in fact $(L_1\|L_2)_w = (L_1)_{P_1(w)} \| (L_2)_{P_2(w)}$. Of course, the first output function is specific to the timed setting, but it is easy to understand from the equivalent form (1). This formal simplicity of Definition 4.2 is another advantage of the coalgebraic approach compared to the algebraic one, where it is only possible to give automata definitions. It seems that there is no algebraic formula in terms of local (algebraic) behaviors (formal power series), but only using linear (automata) representations. Coalgebraic framework makes it simpler due to the fact that final coalgebras are endowed with the same automaton structure as another automaton of a given functor. In fact, algebraic behaviors (formal power series) formal power series can also be endowed with a coalgebra structure [17], but only for the Moore automata functor. Maybe for this reason the algebraic definition of synchronous product is not so elegant and it seems to works on automata representations only: we could not find any definition for formal power series.

Let us mention another two points. Firstly, in the definition of functional stream derivatives it is not necessary to distinguish the type of extended event. Secondly, the first output function can equivalently be written as follows, where we recall $\varepsilon = -\infty$ is the zero element of $\mathbb{R}_{max}$. The value ε in equation (1) can be viewed as undefined. In the special case with full synchronization, i.e. $A_1 = A_2$ there is no need for using extended alphabet, in fact in this case $\mathcal{A} = A_1 = A_2$. Note that for any $v = a \in \mathcal{A}$ we have in fact $P_1(v) = P_2(v) = a$.

Thus, we obtain for $l_1, l_2 \in \mathcal{F}_\infty$ and $\forall a \in A$:

$$(l_1 \| l_2)_a = (l_1)_a \| (l_2)_a$$

and $(l_1\|l_2)[a] = l_1[a] \otimes Bl_2[a] \oplus Bl_1[a] \otimes l_2[a]$. Interestingly, synchronous product differs from sum or Hadamard product of two functionals (with obvious definitions) in the initial condition, which is different from both sum and the Hadamard product.

4.1 Example

In this section the coinductive definition of the last section is applied to a concrete example. We consider a simple distributed timed system consisting of two subsystems: (max,+)-automata G_1 and G_2 over the alphabets $A_1 = \{a, b, d\}$ and $A_2 = \{a, c\}$, respectively, drawn in figure 1. Their synchronous product is by Definition 4.1 the following (max,+)-automaton :

$$\mathbf{G_1} \| \mathbf{G_2} = \mathcal{G} = (\mathbf{Q_1} \times \mathbf{Q_2}, \mathcal{A}, \alpha, \mu, \beta),$$

where $Q_1 \times Q_2$ is the set of states,

$$\mathcal{A} = \{a, (be, c), (d, c)\} \subseteq (A_1 \cap A_2) \cup (A \setminus (A_1 \cap A_2))^*,$$

$\alpha = \alpha_1 \otimes^t \alpha_2$, $\beta = \beta_1 \otimes^t \beta_2$, and

$$\nu(v) = \begin{cases} \mu_1(a) \otimes^t B_2(a) \oplus B_1(a) \otimes^t \mu_2(a), & \text{if } v = a \in A_1 \cap A_2 \\ \mu_1(be) \otimes^t B_2(c) \oplus B_1(be) \otimes^t \mu_2(c), & \text{if } v = (be, c) \\ \mu_1(d) \otimes^t B_2(c) \oplus B_1(d) \otimes^t \mu_2(c), & \text{if } v = (d, c) \end{cases}$$

can easily be computed. The synchronous product $G_1 \| G_2$ is drawn in figure 1 on the right.

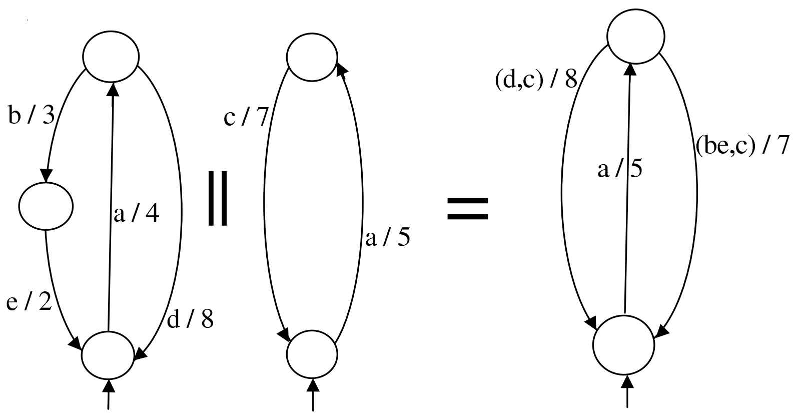

Fig. 1. G_1, G_2 and $G_1\|G_2$

The behavior of the composed automaton is the synchronous product of behaviors of local components over $A_1 = \{a, b, d, e\}$ and $A_2 = \{a, c\}$, which are $l_1 : (A_1)^\infty \to (\mathbb{R}_{max})^\infty$ and $l_2 : (A_2)^\infty \to (\mathbb{R}_{max})^\infty$.

An important feature is that the extended alphabet is based on the underlying untimed partial automata and it includes only those sequences of private events that can occur in between two synchronization events (a). In our case these are sequences bec, bce, cbe (not to be distinguished one from another) represented by their projections to the local alphabets, i.e. (be, c). and sequences dc, cd represented by (d, c). Hence, the extended alphabet is $\mathcal{A} = \{a, (be, c), (d, c)\}$. Note that $P_1(be, c) = be \in A_1^*$, $P_2(be, c) = c \in A_2^*$, and similarly for (d, c). Hence, the differential equation for $(l_1\|l_2)_v$, $v \in \mathcal{A}$ makes sense. The first output and derivative for $v = a$ are as follows:

$$(l_1\|l_2)_a = (l_1)_a\|(l_2)_a \text{ and}$$
$$(l_1\|l_2)[a] = l_1[a] \otimes Bl_2[a] \oplus Bl_1[a] \otimes l_2[a].$$

Let us mention that $(l_1\|l_2)(a) = (l_1\|l_2)(a)(0) = 5 = (l_1\|l_2)[a]$, because $(l_1\|l_2)$ is length preserving and $a \in A^\infty$ is of length 1. Now, according to the fundamental theorem of functional stream calculus we get

$$(l_1\|l_2)(a(d, c)) = (l_1\|l_2)(a(d, c))(0) \oplus (l_1\|l_2)_a(d, c)(0).$$

Direct application of the formulas for derivative and first output function yields

$$(l_1\|l_2)_a(dc) = ((l_1)_a\|(l_2)_a)(dc) = ((l_1)_a\|(l_2)_a)(dc)(0) = ((l_1)_a\|(l_2)_a)[dc]$$

$$= (l_1)_a[d] \otimes B(l_2)_a[c] \oplus B(l_1)_a[d] \otimes (l_2)_a[c] = (l_1)(ad)(1) \otimes B(l_2)(ac)(1) \oplus$$
$$B(l_1)(ad)(1) \otimes (l_2)(ac)(1) = 8 \otimes 0 \oplus 0 \times 7 = 8.$$

The second last equality follows from $f_a[d] = f_a(d)(0) = f(a : d)'(0) = f(ad)(1)$ for any $f \in \mathcal{F}_\infty$.

Similarly, we get $(l_1\|l_2)(a(be, c)) = (l_1\|l_2)(a(be, c))(0) \oplus (l_1\|l_2)_a((be, c))(0)$, where $(l_1\|l_2)_a(be, c) = \ldots = (l_1)(a(be))(1) \otimes B(l_2)(ac)(1) \oplus B(l_1)(a(be))(1) \otimes (l_2)(ac)(1) =$

$5 \otimes 0 \oplus 0 \times 7 = 7$. These lines shows that the synchronous product is easy to compute and moreover if the composed automaton is drawn, its behavior is very intuitive and can directly be written down, cf. Figure 1.

The behavior functional $l_1 \| l_2 \in \mathcal{F}_\infty$ is only partially defined: essentially it is not defined for words that are outside the (prefix-closed) language of the underlying partial automaton and all infinite suffixes of such words. For instance, for $w = aa$ or $w = a^\omega$. On the other hand we have $(l_1 \| l_2)(a(d,c))^\omega = (5\ 8\ 5\ 8 \ldots) = (5\ 8)^\omega$.

The conclusion for this example is that both algebraic and coalgebraic framework can be used to compute the behaviors of synchronous product of (max,+)-automata. However, the coalgebraic approach does not use large matrices (but scalar behaviors), while the algebraic framework needs automata representations.

Remark 4.3 Two points are stressed. Firstly, our approach can be extended to more than two local components as described in [12], but it becomes quite complex. Secondly, synchronous product of interval automata, i.e. deterministic weighted automata with weights in $\mathcal{I}_{max}^{max}$, can be introduced in a similar way as the synchronous product of (max,+)-automata. Interval automata has been studied as B*ü*chi automata over interval based alphabets in [6] and their synchronous product are known as Product Interval Automata (PIA). The same construction based on extended event alphabet can be applied to interval automata and synchronous product of their behaviors can be defined by coinduction.

PIA correspond to an important class of timed automata, where the clocks are read (i.e. compared to constants in transition guards) and reset in a particular fashion: there are n clocks (one per component) and during a transition in a PIA only clocks that correspond to the components that are active in a transition are read and reset. This way the reading and reseting of clocks is compatible with the distributed event set structure. Thus, the usage of clocks can be completely avoided and PIA can be described by symbolic purely algebraic methods.

PIA are capable of modeling many interesting applications like asynchronous circuits. Hence, they represent a nice trade off between tractability (all fundamental problems are known to be decidable for this class of timed automata) and modeling power.

5 Concluding discussion

In this paper deterministic (sequential) weighted automata has been studied coalgebraically as partial Mealy automata. We have recasted their behaviors (causal and consistent stream functions) as partial, length preserving functions (also called stream functionals) and extended basic results of stream calculus into functional stream calculus.

The main advantage of the coalgebraic approach is the possibility to use coinductive definitions and proofs that are known to be pertinent in many applications. Moreover, final coalgebra itself is endowed with the same structure as another coalgebra of a given functor. This helps defining operations on behaviors of state transition systems, e.g. streams, (partial) languages or (partial) stream functions, because these behaviors are seen as corresponding types of automata, e.g. stream automata, (partial) automata or Mealy automata. The corresponding definitions on automata are then simplified into coinductive definitions on behaviors.

In this work another application demonstrating power of coinductive definitions compared to definitions by induction is given: synchronous product of behaviors of deterministic weighted automata are defined by coinduction. The main advantage of our approach is that the composed automaton remains deterministic. On the other hand, the composed system has many states and decentralized approaches must be used in order to avoid the state explosion problem. Since our approach is compositional by construction, it is tailored to decentralized (component-wise) techniques.

Recently we have developed supervisory control theory for (max,+) automata that is applicable to nondeterministic (max,+) automata as well [11] (because determinism plays no essential role.) However, a major problem is that the resulting controller series computed within a behavioral (i.e. formal power series framework) need not be (max,+)-rational. Recall that a series is (max,+)- rational (respectively (min,+)- rational) if it is in the rational closure of series with finite supports, *i.e.* if it can be formed from polynomial series (*i.e.* those with finite support) by rational operation $\oplus$ (corresponding to max, respectively to min), $\otimes$, and the Kleene star. A notion close to that of a deterministic series is a *unambiguous series*, which is a series recognized by unambiguous automata, i.e. automata in which there is at most one successful path labeled by w for every word w. It is known from Lombardy and Sacharovitch [14] that the class of formal power series that are at the same time (max,+) and (min,+) rational coincides with unambiguous series. Moreover, for these families of series, the equality (and inequality) of series is proven to be decidable. Let us recall that sequentialization of weighted automata (i.e. their determinizing) and its decidability status is not known for formal power series over idempotent semirings (unlike the ring case, which is not so interesting for applications in distributed timed systems). The results of [14] show that essentially, beyond the class of deterministic series (i.e. those for which deterministic representations exist) and hence deterministic representations of the timed systems and their (control) specifications there is a little chance of obtaining a rational (i.e. finite state) controller automaton. Also, equality of nondeterministic (max,+)-series is is known to be undecidable [13]. This is a major motivation for our study that consists in coding concurrent (non sequential) timed systems using synchronous product construct instead of using nondeterministic representations à la heap automata.

Among plans for further research, we plan to apply the coinductive definitions presented in this paper in the study of decentralized control (as in [10]) of distributed timed systems that are formed as synchronous compositions of sequential (i.e. one clock) systems. This would lead to an exponential saving of complexity of control synthesis compared to global control synthesis of distributed timed systems.

References

[1] J. Adámek, S. Milius and J. Velebil. *On coalgebra Based on Classes.* Theoretical Computer Science 316, pp.3-23, 2004.

[2] R. Alur and D. Dill. *The Theory of Timed Automata.* Theoretical Computer Science, 126:183-235, 1994.

[3] A. Arnold. *A. Arnold Finite Transition Systems. Semantics of Communicating Sytems*, Prentice-Hall, Englewood Cliffs, NJ, 1994.

[4] P. Buchholz, P. Kemper. *Weak Bisimulation for (max/+)-Automata and Related Models.* Journal of Automata, Languages and Combinatorics (2003) 8 (2), 187-218.

[5] S.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, 1999.

[6] D. D'Souza and P.S.Thiagarajan. *Product Interval Automata*, In Sadhana, Academy Proceedings in Engineering Sciences, Vol. 27, No. 2, Indian Academy of Sciences, pp. 181–208, 2002.

[7] S. Eilenberg. *Automata, Languages, and Machines*, Vol. A. Academic Press, New York, 1974.

[8] S. Gaubert. *Performance evaluation of (max,+) automata*, IEEE Trans. on Automatic Control, vol. 40(12), pp. 2014-2025, 1995.

[9] S. Gaubert and J. Mairesse. *Modeling and analysis of timed Petri nets using heaps of pieces.* IEEE Trans. on Automatic Control, vol. 44(4): 683-698, 1999.

[10] J. Komenda and J. H. van Schuppen: *Modular Control of Discrete-Event Systems with Coalgebra.* IEEE Transactions on Automatic Control, 53, N2, pp. 447-460, 2008.

[11] J. Komenda, S. Lahaye, and J.-L. Boimond. *Supervisory Control of (max,+) automata: a behavioral approach.* Discrete Event Dynamic Systems, 19, N4 , pp. 525-549, Springer, 2009.

[12] J. Komenda, S. Lahaye, and J.-L. Boimond. *Le produit synchrone des automates (max,+).* Modélisation des Systèmes Réactifs (MSR09), Nantes, France, 2009. In JESA (Journal Européen des Systèmes Automatisés), vol. 43, pp.1033–1047, 2009.

[13] D. Krob. *The equality problem for rational series with multiplicities in the tropical semiring is undecidable.* Internat. J. Algebra Comput., 4, pp. 405 - 425, 1994.

[14] S. Lombardy and J. Mairesse. *Series which are both max-plus and min-plus rational are unambiguous*, RAIRO - Theoretical Informatics and Applications 40, pp. 1-14, 2006.

[15] J.J.M.M. Rutten. *Coalgebra, Concurrency, and Control. Research Report CWI*, SEN-R9921, Amsterdam, November 1999. Available also at `http://www.cwi.nl/~janr`.

[16] J.J.M.M. Rutten. *Universal Coalgebra: A Theory of Systems. Theoretical Computer Science* 249(1):3-80, 2000.

[17] J.J.M.M. Rutten. *Behavioural differential equations: a coinductive calculus of streams, automata, and power series.* Theoretical Computer Science Volume 308(1–3), pp. 1–53, 2003.

[18] J.J.M.M. Rutten. *Algebraic Specification and Coalgebraic Synthesis of Mealy Automata.* Proceedings FACS 2005, ENTCS Vol. 160, Elsevier, 2006, pp. 305-319.

[19] J. Sifakis and S. Yovine. *Compositional Specification of Timed Systems.* Invited paper in Proceedings of the 13th Annual Symp. on Theoretical Aspects of Computer Science, STACS'96, pp. 347-359, Springer LNCS 1046, February 1996.