A Data Flow Model with Frequency Arithmetic

Paul Dubrulle$^{(0000-0002-1158-6348)}$, Christophe Gaston$^{(0000-0001-6865-5108)}$, Nikolai Kosmatov$^{(0000-0003-1557-2813)}$, Arnault Lapitre$^{(0000-0002-2185-4051)}$, and Stéphane Louise$^{(0000-0003-4604-6453)}$

CEA, List, 91191 Gif-sur-Yvette France
`firstname.lastname@cea.fr`

Abstract. Data flow formalisms are commonly used to model systems in order to solve problems of buffer sizing and task scheduling. A prerequisite for static analysis of a modeled system is the existence of a periodic schedule in which the sizes of communication channels can be bounded for an unbounded execution (consistency), and that communication dependencies do not introduce a deadlock in such an execution (liveness). In the context of Cyber-Physical Systems, components are often interfaced with the physical world and have frequency constraints. The existing data flow formalisms lack expressiveness to fully cover the expected behavior of these components. We propose an extension to Synchronous Data Flow (SDF) formalism, called Polygraph, that includes frequency constraints and adjustable communication rates. We show that with these extensions, the conditions for a model to be consistent and live are no longer sufficient, and we extend the corresponding theorems with necessary and sufficient conditions to preserve these properties. We also introduce a framework to check the liveness of a Polygraph model, implemented in the tool DIVERSITY, along with preliminary experiments to validate this approach.

1 Introduction

Context. Cyber-Physical Systems (CPS) are increasingly present in everyday life. In these systems, the components require a certain amount of input data to produce a known amount of output data, and some of them must do so in synchrony with a reference time scale. For example, the next generation of autonomous vehicles will heavily rely on sensor fusion systems to operate the car. Sensors and actuators have specified frequencies. To produce its output, the fusion kernel requires a certain number of samples from several sources, with a temporal correlation between them.

Often, when implementing this kind of system, the prediction of its performance is important to the system designer. The performance prediction covers different characteristics of the system, including its throughput, memory footprint, and latency. In distributed implementations of such systems, an analysis of the communications between the components is necessary to configure a network capable to respect the application's real-time requirements.

Data flow formalisms [14, 3] can be used to perform this kind of performance analysis [4, 5, 10–12]. A prerequisite to analyze a model is the existence of a periodic schedule with two properties. The first property, *consistency*, requires that the sizes of the communication buffers remain bounded for an unbounded execution of the periodic schedule. In practice, if a model is not consistent, it is not possible to implement the communications without losing data samples. The second property, *liveness*, requires the absence of deadlocks in the schedule.

Motivation and Goals. The limitation of the existing data flow formalisms to model the considered systems is the lack of expressiveness regarding the synchronization on a common time scale for different components. Overcoming this limitation is the subject of recent research work [6]. Our goal is to extend an existing data flow formalism for which the consistency and liveness properties of a given model are decidable. In doing so, we want to ensure that the expressiveness extension does not impact the decidability of these properties. With this extension, all applicative constraints are taken into account when checking the prerequisites for a performance analysis. The verification can be performed in abstraction of a particular implementation's characteristics (like execution times or mapping), and the results are the same for different implementations. Moreover, the performance analysis can benefit from the additional information on the system provided by the extension.

Approach and Main Results. This paper introduces Polygraph, an extension to Synchronous Data Flow (SDF) [14] for specification of frequency constraints on the components. We use an arithmetic based on rational numbers to reason on data exchanges between components. We show that the theorems that provide a theoretical foundation for practical verification of consistency and liveness for an SDF model can be generalized to this new formalism. Finally, we propose a symbolic execution framework to decide the liveness of models expressed in Polygraph, in a way similar to [14, 11].

The contributions of this work include:

– a data flow formalism, called Polygraph, extending the well-known SDF [14] formalism, to support the synchronization of data production and consumption on a reference time scale;
– a demonstration that the decidability of two classical properties of dataflow models, namely consistency and liveness, is preserved for this new formalism;
– an adaptation to the new formalism of an existing symbolic execution technique for evaluation of liveness in the DIVERSITY tool and initial experiments to validate this approach.

Outline. The remainder of this paper is organized as follows. Section 2 gives an informal introduction to the proposed modeling approach, with a step-by-step explanation relying on an illustrative system. In Section 3, we formalize Polygraph and provide extended statements and a sketch of proof for the consistency and liveness theorems. Section 4 presents a framework to check the liveness property for Polygraph and a preliminary evaluation. In Section 5, we discuss related work, while Section 6 presents conclusion and perspectives.

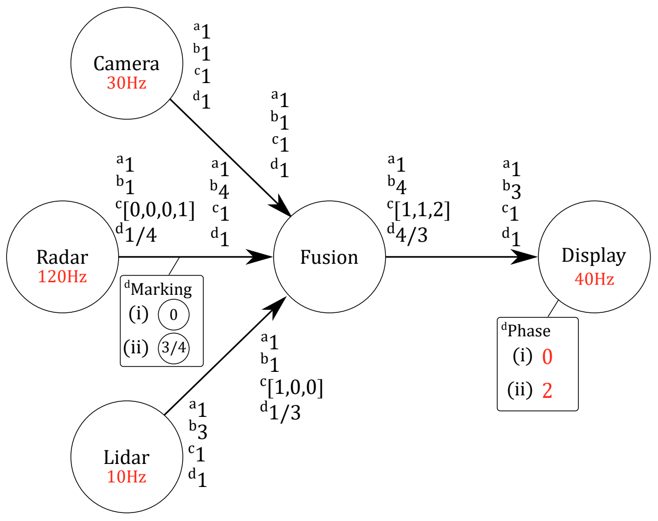


Fig. 1. Motivating example: a data fusion system modeled as a data flow graph. The upper indexes "a" to "d" denote an amount of data exchanged by the components in different variants of the model. The rates denoted by upper index "d" are those of Polygraph, and initial conditions for this configuration are denoted by (i) and (ii).

2 Motivation and Running Example

Running Example. To introduce the modeling approach behind Polygraph, we use a toy example of a data fusion system that could be integrated into the cockpit display of a car, depicted in Figure 1. The system is composed of three sensors producing data samples to be used by a data fusion component, and a display component. The function of the sensor components is to read the data from their sensors, while the function of the data fusion component is to compute a result based on this data. The function of the display component is to render the fusion result on a screen. To do so, the sensor components send the data to the fusion component, and the fusion component sends the result to the display component. The first sensor component is a video camera producing frames. The other two sensor components analyze radar and lidar based samples to produce a descriptor of the closest detected obstacles. The fusion component uses this information to draw the obstacle descriptors on the corresponding frame.

The first step to model this system is to build a graph capturing data dependencies between the components. Each vertex of this graph models an *actor*, an abstract entity representing the function of a component. Each directed edge of the graph models a communication *channel*, the source actor being the producer of data consumed by the destination actor. The structure of the graph in Figure 1 illustrates the dependencies in our example. The communication policy on the channels is First-In First-Out (FIFO), the write operation is non-blocking, and the read operation is blocking. On each channel, the atomic amount of data exchanged by the connected actors is called a *token*, and all write and read operations are measured in tokens. An actor *produces* (resp. *consumes*) a certain number of tokens on a channel when it writes (resp. reads) the corresponding amount of data. With this policy, the graph can be assimilated to a Kahn Process Network (KPN) [13]. In a KPN, the communications are determinate, but in general it is not possible to decide if the sizes of the channels can be bounded for an unbounded execution of the system.

Synchronous and Asynchronous Constraints. In practice, sensors and actuators have a fixed sampling rate, and the production of each data sample occurs at that specified frequency. To model these constraints, we propose to label some actors with *frequencies*, corresponding to the real-life constraint. An actor with a frequency label must *fire* at that frequency. We further detail this notion of firing below, but for now it is sufficient to say that the firing of an actor is an atomic process, during which it performs the actions and communications expected from the modeled component. A global clock provides ticks to synchronize the firing of frequency labeled actors. For our example, we consider the frequency labeling illustrated by Figure 1.

Generally, in real-life systems, computation kernels compute when input data is available and do not have frequency constraints. In our frequency labeling, the actors modeling such components can be left without a frequency label. In our example, this is the case for the fusion actor.

The possibility to have unlabeled actors is an important part of our approach, as further discussed in Section 5. It allows to mix a synchronous firing policy for labeled actors, and an asynchronous firing policy for unlabeled actors. This means that the scheduling of firings has periodic constraints only where needed, which offers more options for optimization algorithms.

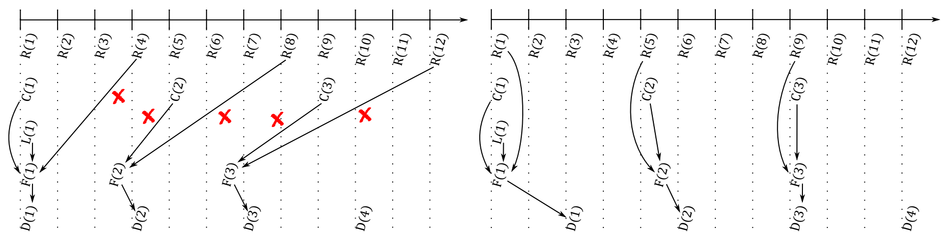
Static Rates. Another characteristic of real-life software components in our context is that they require a fixed number of input samples from each different source. Also, there must be a correlation between the production time of the samples consumed from different sources. In our example, the fusion component requires one token from each sensor, and these samples must have a close-enough production time. This constraint can be captured by KPN restrictions, such as Synchronous Data Flow (SDF) [14]. In SDF, both ends of each channel are assigned a communication rate, denoting the fixed number of tokens produced or consumed by the connected actors' firings. This characteristic allows to decide whether the sizes of the channels are bounded for an unbounded execution. Graphs respecting this property are said to be *consistent*.

Without taking frequencies into account, the communication rates denoted by an upper index "a" in Figure 1 match the description of the system. Indeed, the sensor actors produce one token each, the fusion actor consumes these tokens, and in turn produces one token to be consumed by the display actor. With these rates, considering a marking of the graph with any number of tokens stored in the channels, if firing all the actors once, the same number of tokens remains in the channels. Hence, the SDF graph is consistent. But when taking frequencies into account, the graph is no longer consistent. In this example, the camera produces 30 tokens per second, the radar produces 120 tokens per second, and the lidar produces 10 tokens per second. This means that per second, because of the production rate and frequency of the lidar, the fusion actor will be able to fire only 10 times. It will consume only 10 tokens from the camera and radar actors, leaving 20 and 110 unconsumed tokens per second on their respective channels. Hence, it is no longer possible to bound the size of these channels for an unbounded execution of the graph. This shows that to achieve consistency, for any frequency labeled actor, the number of asynchronous firings of its unlabeled predecessors and successors should be limited.

A possible adaptation of communication rates, denoted by upper index "b" in Figure 1, takes frequency inheritance into account and restores the consistency property. With the production and consumption rates both set to 1 on the channel connecting the camera and the fusion actors, the fusion actor basically inherits a frequency constraint of 30Hz. It inherits the same frequency constraint from the radar and lidar actors since it now consumes $4 \times 30 = 1 \times 120$ tokens per second from the radar, and $1 \times 30 = 3 \times 10$ tokens per second from the lidar. The rates on the channel connecting the fusion and display actors are also balanced. But with these rates, the number of tokens does not reflect accurately the expected behavior of the modeled components. For example, the fusion actor would consume 4 tokens per activation from the radar actor, while in reality the component only requires 1.

Cyclo-Static Rates. It is possible to use Cyclo-Static Data Flow (CSDF) [3] to get closer to the real communication requirements. In CSDF, the rates of the actors are fixed as in SDF, but the successive firings of an actor cyclically consume and produce a different number of tokens on every connected channel. The successive rates on each channel are expressed as a sequence of natural numbers. For example, an actor with a cyclo-static sequence of output rates $[1, 2]$ produces 1 token for its first firing, 2 tokens for the second, 1 for the third and so on. A zero rate may occur in the sequence, meaning that the actor does not push or pull tokens on the channel for the corresponding firing.

A cyclo-static sequence is necessary on a channel if the connected actors have frequency constraints conflicting with the expected communication behavior. In this case, we propose that one of the actors must be chosen as having the reference frequency for the communication, and the other actor must adapt its communication rate to a cyclo-static sequence accordingly. Back to our example (see variant "c" in Figure 1), the fusion actor requires one token from each sensor every firing. Since the component is synchronized on camera frames, we decide that the actor's reference frequency should be 30Hz. In this case, the frequency constraints do not conflict with the expected communication behavior, and we assign production and consumption rates of 1 on the channel connecting the fusion and camera actors. Now, considering the radar actor, the fusion actor only requires 30 tokens per second out of 120. Considering this ratio, we assign the sequence $[0, 0, 0, 1]$ as production rates for the radar actor, and the rate 1 for the fusion actor. The same logic applies for the lidar actor, the fusion actor



(a) Initial conditions (i). (b) Initial conditions (ii).

Fig. 2. Firings of actors of the motivating example: the firings are identified by the initial letter of the corresponding actor and the rank of the firing, arrows show data dependencies between firings, and a reference time scale constrains the firing of timed actors. The data dependencies marked by a cross in (a) introduce a causality issue.

requires 30 tokens per second, but only 10 tokens per second are produced. We then assign the cyclo-static sequence $[1, 0, 0]$ as consumption rates for the fusion actor, and the rate 1 for the lidar actor. A similar logic is applied for the display actor. The consequence on the stream of actual data values highly depends on the implemented function, and is therefore out of the scope of the data flow modeling. In the particular case of the radar actor in our example, the software implementation could perform a downsampling of the sensed data, or just send the latest sample.

The corresponding communication rates, denoted by upper index "c" in Figure 1, give a graph where only the required tokens are exchanged on the channels, and the consistency property is preserved. But in all generality, choosing the appropriate cyclic rate sequences for all the channels in a graph is time consuming and error prone.

Rational Rates. We propose instead to extend the SDF model with rational communication rates. A rational communication rate $r = p/q$ specifies that the actor produces or consumes p tokens every q firings, and the natural number of tokens produced or consumed by any firing is r rounded either up or down, denoted $\lceil r \rceil$ and $\lfloor r \rfloor$ respectively. With the semantic formalized in the next section, there is a unique default cyclo-static sequence that corresponds to a given rational rate. The default sequences for the rates denoted by an upper index "d" in Figure 1 are those denoted by upper index "c". As explained earlier when assigning cyclo-static sequences, in this extension, only one rate on a given channel can be a rational number with denominator greater than one. The methodology remains the same, for any channel, one actor's frequency is considered as a reference, and the other one adapts its rates according to that reference.

Initial Conditions. With the frequency labeling and rational communication rates, we obtain a model that describes as closely as possible the communication and timing requirements of our illustrative example. But there are causality issues in this model. Figure 2(a) illustrates the timing of actor firings in our example, and the data dependencies between them, according to the semantic defined in the next section. It is obvious that the data dependencies marked by a cross are not satisfied in time.

This kind of causality issue can also appear in SDF: in the case of cyclic graphs, the firings of the actors in a cycle all depend on each other. To prevent this, it is possible to *mark* the channels with an initial number of tokens, allowing sufficient initial firings to complete the firing of all actors in the cycle. The liveness property of an SDF graph is verified when all the cycles in the graph are marked with enough tokens to prevent a deadlock [14]. With the SDF extensions we propose, this condition is no longer sufficient. We need to be able to shift the production or consumption of tokens in order to make sure that when a firing requires input tokens, they are produced at an earlier tick of the global clock.

One way to achieve this is to rotate the default sequences defined by the rational rates. For this, we propose a rational initial marking of the graph. Each channel with natural rates at both ends can be marked with an initial number of tokens as in SDF. Each other channel with rational rate $r = p/q$ on either end can be initially marked with a rational number $n + k/q$ with $k < q$, which denotes that the channel initially holds n tokens (as in SDF), and the default sequence is rotated by k. If the rational rate is on the producer, the default sequence is rotated left, otherwise it is rotated right. In Figure 1, considering the default sequences denoted by "c", the corresponding rational rates denoted by upper index "d", and the initial marking (ii), the marking of 3/4 on the channel connecting the radar and fusion actors rotates the default sequence $[0, 0, 0, 1]$ by 3 elements to the right, yielding the sequence $[1, 0, 0, 0]$.

Another way to prevent unsatisfied data dependencies is to shift the first tick on which a frequency labeled actor must fire. We propose to add a *phase* to each of these actors, giving the offset from the first tick at which it must fire. With the semantic formalized in the next section, that phase is constrained in order to have a periodic global clock. Figure 2(b) takes into account the marking and phase denoted (ii) in Figure 1. With the rational marking, the dependencies between the radar and fusion firings are now satisfied, and with the phase on the display actor, the dependencies between the camera and display firings are also satisfied.

3 Formalization of the Polygraph Model

We denote by $\mathbb{B}$ the set $\{0, 1\}$, by $\mathbb{Z}$ the set of integers, by $\mathbb{N} = \{n \in \mathbb{Z} \mid n \geqslant 0\}$ the set of natural integers, and by $\mathbb{Q}$ the set of rational numbers. For any set S, the free semigroup on S is denoted S^+.

System graph. A *system graph* is a structure used to represent the topology of the communications. Formally, it is a connected finite directed graph $G = (V, E)$ with set of vertices V and set of edges $E \subseteq V \times V$ such that V is the set of *actors* and E is the set of *channels*. We use an index notation to identify elements with respect to a given actor or channel, considering that E and V are sets indexed respectively in $\{1, \cdots, |E|\}$ and $\{1, \cdots, |V|\}$. We denote v_i (resp. e_j) the actor (resp. channel) of index i (resp. j). For an actor $v \in V$, let $\mathrm{in}(v) = \{\langle v', v\rangle \in E \mid v' \in V\}$ denote the set of *input channels* of v and $\mathrm{out}(v) = \{\langle v, v'\rangle \in E \mid v' \in V\}$ the set of *output channels* of v.

Topology matrix and channel states. As for SDF and its derivations [14, 3], the communication rates are defined by a topology matrix with one row per channel and one column per actor. The only difference in this definition is that we rely on rational numbers. The absolute value of a rate in the matrix defines how many tokens are produced or consumed per firing of the corresponding actor on the corresponding channel, and the sign of that rate indicates if the tokens are produced (positive rate) or consumed (negative rate). For a given actor and channel, the rate must be 0 if the actor is not connected to the channel, or if the actor is connected to both ends of the channel.

Definition 1 (Topology matrix). *A matrix $\mathbf{\Gamma} = (\gamma_{ij}) \in \mathbb{Q}^{|E|\times|V|}$ is a* topology matrix *of a system graph G if for every channel $e_i = \langle v_j, v_k\rangle \in E$ we have:*

- *$\gamma_{il} = 0$ for all $l \neq j, k$;*
- *if $j \neq k$, then $\gamma_{ij} > 0$ and $\gamma_{ik} < 0$ are irreducible fractions, and at most one of them has a denominator greater than 1;*
- *if $j = k$, then $\gamma_{ij} = 0$.*

We also use a rational number per channel to track the communication state of the system during an execution. A channel state is a vector with one row per channel. Each coordinate in the vector tracks the respective number of firings of the connected actors, by addition of their rates when they fire, and that coordinate rounded down is the number of tokens in the channel.

Definition 2 (Channel state). *A vector $\mathbf{c} \in \mathbb{Q}^{|E|\times 1}$ is a* channel state *of a system graph G with topology matrix $\mathbf{\Gamma}$ if for every channel $e_i = \langle v_j, v_k\rangle \in E$, the denominator of c_i is the maximum between the denominators of γ_{ij} and γ_{ik}, and $\lfloor c_i \rfloor$ is the number of tokens in the channel. We denote $C \subseteq \mathbb{Q}^{|E|\times 1}$ the set of all these possible states.*

Timed actors and global clock. A subset $V_F \subseteq V$ of *timed actors* are constrained by a *frequency*, expressed as a strictly positive natural number. We use a frequency mapping $\omega : V_F \longrightarrow \mathbb{N}^{>0}$ in order to map the timed actors to their frequency. There is an implicit system time unit, and each timed actor $v_i \in V_F$ is supposed to be fired exactly $\omega_i := \omega(v_i)$ times per system time unit. In order to have a minimal system time unit, we consider that the greatest common divisor of all the frequencies is $\gcd(\omega[V_F]) = 1$. This is not limiting, since any set of frequencies and system time unit can be adjusted to fit this constraint.

In addition, the timed actors must fire synchronously with respect to a global clock. The *resolution* of that global clock is a sufficient number of *ticks* per system time unit to associate to each tick the set of timed actors that must fire at the corresponding date. For this, we consider the ticks $0, 1, \ldots, \pi - 1$ per system time unit, where π is the least common multiple of all the actor frequencies

$\pi = \mathrm{lcm}(\{\omega_i | v_i \in V_F\})$. Note that if V_F is empty, $\pi = 1$, and the global clock does not constrain the firing of any actor.

Given a timed actor $v_i \in V_F$, there should be ω_i out of π ticks associated with that actor's firings. To reflect the periodic nature of the firing of timed actors, for a timed actor v_i of period $p_i = \pi/\omega_i$, it fires every p_i-th tick.

As mentioned in Section 2, all the timed actors have a *phase*. We use a phase mapping $\varphi : V_F \longrightarrow \mathbb{N}$ to map the timed actors to their phase. The first firing of each timed actor $v_i \in V_F$ occurs at the tick $\varphi_i := \varphi(v_i)$. The only constraint to respect the expected frequency of the firings is that $\forall v_i \in V_F$ we have $0 \leqslant \varphi_i < \pi/\omega_i$.

Definition 3 (Global clock, firing ticks). *For a system graph G with frequency mapping ω, resolution π, and phase mapping φ, the* global clock *is a set* $\mathrm{T} = \{0, 1, \ldots, \pi - 1\}$ *and for each timed actor $v_i \in V_F$ there is a subset of* firing ticks $\mathrm{T}_i = \{\tau \in \mathrm{T} \,|\, \tau \equiv \varphi_i \,(\mathrm{mod}\ \pi/\omega_i)\}$.

Polygraphs. We now define the notion of *polygraph* which introduces a basic communication topology, a topology matrix, a frequency and phase mapping for all timed actors, and an initial marking of the graph.

Definition 4 (Polygraph, initial marking). *A* polygraph *is a tuple $\mathcal{P} = \langle G, \mathbf{\Gamma}, \omega, \varphi, \mathbf{m} \rangle$ where G is a system graph, $\mathbf{\Gamma}$ is a topology matrix, ω is a frequency mapping, φ is a phase mapping and $\mathbf{m} \in C$ is an* initial marking *such that $\forall e_i \in E$ we have $m_i \geqslant 0$.*

In the following, we consider that a polygraph $\mathcal{P} = \langle G, \mathbf{\Gamma}, \omega, \varphi, \mathbf{m} \rangle$ is given, with its global clock T and sets of firing ticks T_i for all the timed actors $v_i \in V_F$.

States and transitions. The state of a polygraph is composed of a channel state, the current tick of the global clock, and a vector with one row per actor used to track the number of firings of the timed actors since the last change in the current tick. This *tracking vector* is used to check that the timed actors respect their synchronous firing constraints.

Definition 5 (State). *A* state *of a polygraph $\mathcal{P}$ is a tuple $s = \langle \mathbf{c}, \tau, \mathbf{a} \rangle$ where $\mathbf{c} \in C$ is a channel state, $\tau \in \mathrm{T}$ is a tick, and $\mathbf{a} \in \mathbb{N}^{|V| \times 1}$ is a tracking vector. We denote $S \subseteq C \times \mathrm{T} \times \mathbb{N}^{|V| \times 1}$ the set of all possible states for $\mathcal{P}$.*

The effect of the firing of an actor on the channel state is to add its rates to the respective coordinate of all the channels. For an actor v_i, the i-th column of $\mathbf{\Gamma}$ gives all the rates per channel. Therefore, to extract that column from the matrix for each actor $v_i \in V$, we use a *unitary firing vector* $\mathbf{u} \in \mathbb{B}^{|V| \times 1}$, such that $u_i = 1$, and for all $j \neq i$ we have $u_j = 0$. We denote $U \subset \mathbb{B}^{|V| \times 1}$ the set of these vectors, and for convenience we denote the unitary activation vector of actor v_i by $\mathbf{u}_i$. With the unitary firing vector of any actor v_i, the product $\mathbf{\Gamma u}_i$ gives a vector holding for each channel e_j the rate of v_i on e_j. For any channel state $\mathbf{c}$, the channel state after the atomic firing of v_i is then $\mathbf{c} + \mathbf{\Gamma u^i}$. Also, the firing of a timed actor is tracked by adding its unitary firing vector to the tracking vector. The firing of an actor has no effect on the current tick.

Definition 6 (Fire). *For a polygraph $\mathcal{P}$, the mapping* $\text{fire} : U \times S \longrightarrow S$ *maps a unitary activation vector* $\mathbf{u}_i$ *and a state* $s = \langle \mathbf{c}, \tau, \mathbf{a} \rangle$ *to the state* $s' = \langle \mathbf{c}', \tau', \mathbf{a}' \rangle$ *such that we have* $\mathbf{c}' = \mathbf{c} + \mathbf{\Gamma}\mathbf{u}_i$, $\tau' = \tau$, *and if* $v_i \in V_F$ *then* $\mathbf{a}' = \mathbf{a} + \mathbf{u}_i$, *otherwise* $\mathbf{a}' = \mathbf{a}$.

Remark 1. For two consecutive firings of any actors v_i and v_j from a state $s = \langle \mathbf{c}, \tau, \mathbf{a} \rangle$, the resulting state $s'' = \langle \mathbf{c}'', \tau'', \mathbf{a}'' \rangle$ does not depend on the order of the firings, and $\mathbf{c}'' = \mathbf{c} + \mathbf{\Gamma}(\mathbf{u}_i + \mathbf{u}_j)$. This property can be generalized to any finite number of consecutive firings.

The other possible transition between two states occurs when the global clock ticks. When the global clock ticks, the channel state is not changed, the current tick is adjusted, and the tracking vector is reset.

Definition 7 (Tick). *For a polygraph $\mathcal{P}$, the mapping* $\text{tick} : S \longrightarrow S$ *maps a state* $s = \langle \mathbf{c}, \tau, \mathbf{a} \rangle$ *to the state* $s' = \langle \mathbf{c}', \tau', \mathbf{a}' \rangle$ *such that we have* $\mathbf{c}' = \mathbf{c}$, $\tau' = (\tau + 1) \bmod \pi$, *and* $\mathbf{a}' = \mathbf{0}$.

Executions. The state of $\mathcal{P}$ can evolve by successive application of either *fire* or *tick*. An *execution* of $\mathcal{P}$ is a sequence of such applications starting from a state $s_1 \in S$ and leading to states $e = s_1 \cdots s_n \in S^+$. However, with the frequency constraints, there are some conditions for the applications.

Consider the firing $\text{fire}(\mathbf{u}_i, s)$ of a timed actor v_i in a state $s = \langle \mathbf{c}, \tau, \mathbf{a} \rangle$. In this case, v_i may fire only if the current tick τ is one of its firing ticks, *i.e.* $\tau \in \mathrm{T}_i$. Since it must fire exactly once on such a tick, an additional constraint to fire a timed actor v_i is that it has not fired yet, *i.e.* its coordinate in the tracking vector $\mathbf{a}$ is $a_i = 0$. To capture this constraint, we define a *tick firing vector* $\mathbf{t}^\tau \in \mathbb{B}^{|V| \times 1}$ for each tick $\tau \in \mathrm{T}$, in which a coordinate is set to one if the corresponding actor is expected to fire at tick τ. More formally, for any $v_i \in V \backslash V_F$ we have $t_i^\tau = 0$, and for any $v_j \in V_F$ we have $t_j^\tau = 1$ if $\tau \in \mathrm{T}_j$, and $t_j^\tau = 0$ otherwise. The constraint to fire $v_i \in V_F$ in a state with current tick τ and tracking vector $\mathbf{a}$ is then $a_i < t_i^\tau$.

The clock update $\text{tick}(s)$ in a state $s = \langle \mathbf{c}, \tau, \mathbf{a} \rangle$ is also subject to a constraint: the timed actors that were supposed to fire synchronously with the current tick have done so exactly once, *i.e.* $\mathbf{a} = \mathbf{t}^\tau$.

Definition 8 (Synchronous execution). *An execution* $e = s_1 \cdots s_n \in S^+$ *of a polygraph $\mathcal{P}$ is* synchronous *if* $\forall 1 \leqslant k < n$, *we have* $s_k = \langle \mathbf{c}, \tau, \mathbf{a} \rangle$ *such that:*

- *either* $s_{k+1} = \text{fire}(\mathbf{u}_i, s_k)$ *for some* $v_i \in V$, *and in addition, if* $v_i \in V_F$, *then* $a_i < t_i^\tau$,
- *or* $s_{k+1} = \text{tick}(s_k)$, *and in addition,* $\mathbf{a} = \mathbf{t}^\tau$.

Until now, we considered executions of a polygraph where the order of the firings is constrained only by the frequencies. However, for an actor to fire, there must be enough tokens on its input channels, or its rational communication rate must allow firings consuming 0 tokens. In order to fire an actor v_i in a state

$s = \langle \mathbf{c}, \tau, \mathbf{a} \rangle$, we require that for each input channel e_j of v_i, since the rate γ_{ji} is negative, the channel state c_j must be large enough to avoid reaching a negative state, *i.e.* $c_j + \gamma_{ji} \geqslant 0$, or equivalently $c_j \geqslant |\gamma_{ji}|$. This constraint requires an ordering of the actor firings such that a producer is fired a sufficient number of times for a consumer to be able to fire in turn.

Definition 9 (Non-blocking execution). *An execution $e = s_1 \cdots s_n \in S^+$ of a polygraph $\mathcal{P}$ is* non-blocking *if $\forall 1 \leqslant k < n$, we have $s_k = \langle \mathbf{c}, \tau, \mathbf{a} \rangle$ such that:*

– *either $s_{k+1} = \text{fire}(\mathbf{u}_i, s_k)$ for some $v_i \in V$, and in addition, $\forall e_j \in \text{in}(v_i)$, $c_j \geqslant |\gamma_{ji}|$,*
– *or $s_{k+1} = \text{tick}(s_k)$.*

Consistency property. If verified, the *consistency* property of $\mathcal{P}$ guarantees that it is possible to build a synchronous execution $e = s_1 \cdots s_n \in S^+$ such that $s_1 = \langle \mathbf{m}, 0, \mathbf{0} \rangle$ and $s_1 = s_n$. Such an execution is called a *consistent* execution of $\mathcal{P}$, and can obviously be repeated an indefinite number of times to build a consistent execution of arbitrary length. [14, Theorem 1] states that a necessary and sufficient condition for a given SDF graph to be consistent is that there is a non-trivial solution $\mathbf{x}$ to $\mathbf{\Gamma x} = \mathbf{0}$.

To extend this result to polygraphs, as explained in the previous section, we need to take into account the frequencies of the timed actors. In other words, we need to make sure that it is possible to have a synchronous execution with x_i firings per actor v_i. The additional constraint due to the frequencies is that the number of firings x_i of all the timed actors v_i corresponds to a number $r \in \mathbb{N}$ of repetitions of the global clock period.

To state the conditions for a polygraph to be consistent, we thus want to separate the number of firings of the timed actors from the others. We define the vector $\mathbf{t} = \sum_{\forall \tau \in \mathrm{T}} \mathbf{t}^\tau$ giving for each timed actor v_i the number t_i of expected firings per period of the global clock. We then define the set $Y \subset \mathbb{N}^{|V| \times 1}$ of vectors $\mathbf{y}$ such that we have a number of firings $y_i \neq 0$ only for $v_i \in V \backslash V_F$.

Theorem 1. *A polygraph $\mathcal{P}$ has a consistent execution if and only if there exists a non-trivial solution $\mathbf{x} \in \mathbb{N}^{|V| \times 1}$ to $\mathbf{\Gamma x} = \mathbf{0}$ such that $\mathbf{x} = \mathbf{y} + r\mathbf{t}$ for some $\mathbf{y} \in Y$ and $r \in \mathbb{N}$. Any such solution is called a* repetition vector *of $\mathcal{P}$. Moreover, there exists a* minimal repetition vector $\mathbf{x}$ *such that for any other repetition vector $\mathbf{x}'$ we have $\mathbf{x}' = k\mathbf{x}$ for some $k \in \mathbb{N}$.*

Sketch of proof. First, we prove that the condition is sufficient, and suppose that there exists such a solution $\mathbf{x}$. Then we can decompose:

$$\mathbf{x} = \mathbf{y} + \underbrace{\underbrace{(\mathbf{t^0} + \ldots + \mathbf{t}^{\pi-\mathbf{1}})}_{=\mathbf{t}} + \ldots + \underbrace{(\mathbf{t^0} + \ldots + \mathbf{t}^{\pi-\mathbf{1}})}_{=\mathbf{t}}}_{=r\mathbf{t}}$$

The required consistent execution can be obtained by constructing sub-executions corresponding to this decomposition, relying on Def. 8 and Remark 1.

Claim (1). There exists a synchronous execution $e_1 \in S^+$ with starting state $s = \langle \mathbf{m}, 0, \mathbf{0} \rangle$ and ending state $s' = \langle \mathbf{m} + \mathbf{\Gamma y}, 0, \mathbf{0} \rangle$.

The execution e_1 is constructed by applying y_i firings of each actor $v_i \in V \backslash V_F$ (in any order). Since the fired actors are not timed actors, any such sequence is synchronous. The resulting channel state is $\mathbf{m} + \mathbf{\Gamma y}$ as per Remark 1.

Claim (2). For any starting state $s = \langle \mathbf{c}, \tau, \mathbf{0} \rangle$, there exists a synchronous execution $e_2 \in S^+$ starting from s with ending state $s' = \langle \mathbf{c} + \mathbf{\Gamma t}^\tau, (\tau + 1) \bmod \pi, \mathbf{0} \rangle$.

The execution e_2 for τ is constructed by firing exactly once each timed actor supposed to do so at tick τ, and then applying the tick mapping.

Claim (3). For any starting state $s = \langle \mathbf{c}, 0, \mathbf{0} \rangle$, there exists a synchronous execution $e_3 \in S^+$ starting from s with ending state $s' = \langle \mathbf{c} + \mathbf{\Gamma t}, 0, \mathbf{0} \rangle$.

The execution e_3 is obtained by successively executing e_2 for $\tau = 0, \ldots, \pi - 1$.

Claim (4). There exists a synchronous execution $e_4 \in S^+$ with starting state $s = \langle \mathbf{m}, 0, \mathbf{0} \rangle$ and ending state $s' = \langle \mathbf{m} + \mathbf{\Gamma}(\mathbf{y} + r\mathbf{t}), 0, \mathbf{0} \rangle$.

The sequence e_4 is constructed by executing e_1, followed by e_3 repeated r times. Hence, given that $\mathbf{\Gamma x} = \mathbf{0}$ and $\mathbf{x} = \mathbf{y} + r\mathbf{t}$, it can be easily checked that the ending state of e_4 is the same as its starting state, and e_4 is consistent. The fact that the condition is also necessary follows from the definitions. Since the current tick must return to 0 after a consistent execution, such an execution must perform a number r of periods of the global clock for some $r \in \mathbb{N}$, in other words it must contain $r\pi$ applications of the tick mapping and rt_i firings of each timed actor v_i. The existence of a minimal solution immediately follows from the fact that in this case $\operatorname{rank}(\mathbf{\Gamma}) = |V| - 1$ according to [14, Corollary of Lemma 2].

Due to lack of space, a detailed proof is left to the reader. □

Liveness property. If verified, the *liveness* property of $\mathcal{P}$ guarantees that it is possible to build a consistent execution $e = s_1 \cdots s_n \in S^+$ such that e is also a non-blocking execution. Such an execution e is called a *live execution.*

In a way similar to [14, Theorem 3], we define the notion of a scheduler building only synchronous and non-blocking executions. Our goal is to show that $\mathcal{P}$ has a live execution if and only if any such scheduler can build a consistent execution.

From now on, we consider that $\mathcal{P}$ is consistent with minimal repetition vector $\mathbf{x}$. We define the mapping *count* : $V \times S^+ \longrightarrow \mathbb{N}$ that given an actor v_i and an execution $e = s_1 \cdots s_n \in S^+$ returns the number of firings of v_i in e, *i.e.* the number of k such that $1 \leqslant k < n$ and $s_{k+1} = \operatorname{fire}(\mathbf{u}_i, s_k)$. Notice that since a live execution e of $\mathcal{P}$ is also consistent, by definition we have $\forall v_i \in V, \operatorname{count}(v_i, e) = x_i$. Also, we say that an actor $v_i \in V$ is *runnable* after an execution $e \in S^+$ with ending state s if $\operatorname{count}(v_i, e) < x_i$ and the one-step execution $ss' \in S^+$ with $s' = \operatorname{fire}(\mathbf{u}_i, s)$ is synchronous and non-blocking.

Definition 10 (Scheduler). *A* scheduler *of $\mathcal{P}$ is a mapping $\sigma : S^+ \longrightarrow S^+$ that maps an execution $e = s_1 \cdots s_n \in S^+$ to an execution $e' \in S^+$ such that if we denote $s_n = \langle \mathbf{c}, \tau, \mathbf{a} \rangle$ we have:*

– *either* $e' = s_1 \cdots s_n s' \in S^+$ *with* $s' = \text{fire}(\mathbf{u}_i, s_n)$ *for some actor* v_i *runnable after* e*;*
– *or* $e' = s_1 \cdots s_n s' \in S^+$ *with* $s' = \text{tick}(s_n)$ *and* $\mathbf{a} = \mathbf{t}^\tau$*;*
– *or* $e' = e$ *if there is no runnable actor after* e *and* $\mathbf{a} \neq \mathbf{t}^\tau$*.*

An execution defined by a scheduler σ is the fixed point constructed by recursive application[1] of σ starting from an initial execution $e = (\langle \mathbf{m}, 0, \mathbf{0} \rangle)$.

Theorem 2. *Let* $\mathcal{P}$ *be a consistent polygraph with minimal repetition vector* $\mathbf{x}$*,* σ *a scheduler of* $\mathcal{P}$*, and* e *the execution defined by* σ*. Then* $\mathcal{P}$ *has a live execution if and only if* $\forall v_i \in V, \text{count}(v_i, e) = x_i$*.*

Sketch of proof. The condition is obviously sufficient. The proof that it is also necessary can be easily made by induction. If e is a live execution and e' is a synchronous and non-blocking execution constructed by σ so far, with $|e'| < |e|$, we can show that e' can be extended by one more step (*e.g.* by taking the first step present in e but not in e', since its preconditions are necessarily satisfied). $\square$

4 Tool Support for Liveness Checking

DIVERSITY is a customizable model analysis tool based on symbolic execution, available in the *Eclipse Formal Modeling Project* [17]. DIVERSITY provides a pivot language called *xLIA* (eXecutable Language for Interaction and Architecture) introducing a set of communication and execution primitives allowing one to encode a wide class of dynamic model semantics [9, 2], Communicating STS [1], and abstractions of hybrid systems [15]. In this work, we use it to analyze Polygraph models, to check their liveness in a similar way to that defined by a scheduler as per Def. 10.

The root entity in an xLIA model is a so-called *system*. A system is an executable entity that can be atomic (state-machine) or compositional or hierarchical. A Polygraph model translated to xLIA is a system where the actors are state-machines with input/output ports associated with the ends of the channels. They communicate asynchronously over FIFO queues, bounded or not, using xLIA connectors. Variables are used to store received tokens on input instructions in transitions, with guards conditioning their firing, and output statements to model their token productions.

Figure 3 represents such a state machine for any actor of the polygraph in Fig. 1. Each transition is labeled with xLIA macros representing the actions performed. The *init* macro moves the initial marking from the input queues to the counter of available input tokens, *canFire()* tests if enough tokens are present for a non-blocking firing, *consumption* decrements the counter of available input tokens, *production* sends the production rate on the successor's queue, and *reception* reads that rate and adds it to the number of available tokens. Regarding state machine semantics, all the states are pseudo-states, except *idle* which is stable. This means that any fired transition must be completed until returning

[1] Hence, a scheduler can be also defined as a *partial* mapping on $\sigma^*(\langle \mathbf{m}, 0, \mathbf{0} \rangle)$.

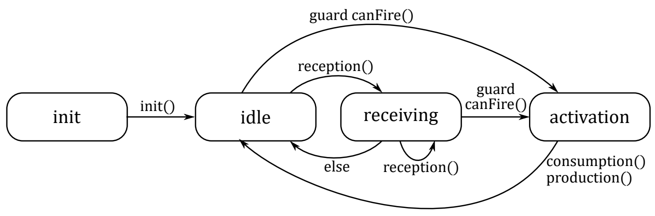


Fig. 3. xLIA statemachine pattern for an actor of a polygraph

to the idle state. The *else* transition will be evaluated if there is no possible *reception*.

The xLIA language allows a fine-grained definition of an execution model for the actors of a polygraph. Some instructions associate a sequence of actors to fire with each tick of a clock. When attempting to fire a timed actor, only one firing is triggered if possible, and when attempting the same for other actors, as many firings as possible are triggered. Hence, the timed actors are only fired at the expected tick, and cause a deadlock result if it's not possible. For the other actors, a counter limits their number of firings to their coordinate in the minimal repetition vector, as required by Theorem 2. With this setup, for a polygraph $\mathcal{P}$ with minimal repetition vector $\mathbf{x} = \mathbf{y} + r\mathbf{t}$, the length of a live execution path is $r\pi$, plus one for the initialization step handling the initial marking. Any path with less steps leads to a deadlock.

We tested this technique using DIVERSITY on an *Intel core i7*. For the polygraph of Figure 1 with initial marking (ii), the tool finds that the liveness property is verified. We also tested the initial marking (i), and the tool correctly identified a deadlock in less than 200ms. This example is extracted from a more complex polygraph modeling an Advanced Driver-Assistance System (ADAS), that we also used to evaluate the liveness checking tool. The considered polygraph has 18 actors (5 of which are timed actors), 32 channels (6 of which have an initial marking), where 10 actors have rational communication rates. For a correctly marked model, we find a live execution sequence in 4s.

5 Discussion and Related Work

In [16], an extension to SDF is proposed to add a single throughput constraint on a channel of a consistent graph. From this constraint, a firing frequency is derived for the actors by transitivity. This approach, while preserving the consistency property by construction, does not allow the expression of a frequency constraint per actor, based on a real-life constraint on the modeled component, nor the explicit synchronization of the firings on a reference time scale.

The programming model PTIDES [18] combines a real-time semantic for sensors and actuators, and a discrete event semantic for other components like computation kernels. These other components have an awareness of the real time through a logical time abstraction. The resulting execution semantic has similarities with Polygraph, since some components are constrained by real-time and others only react to their stimuli. The semantic of PTIDES is much more flexible than Polygraph, since it does not require fixed production or consumption rates. On the other hand, and as opposed to Polygraph, there is no way to derive a consistent and live periodic schedule in PTIDES, which makes static performance prediction more difficult. Nevertheless, since the semantics are similar, we believe that the notion of logical time as defined in PTIDES is applicable to practical distributed implementations of polygraphs.

Synchronous programming languages [7, 8] can be used to express a data flow between synchronous periodic nodes, in order to generate correct-by-construction programs. In these approaches, all the nodes are synchronous, while in Polygraph, some actors fire asynchronously when enabled. Also, the goal of our approach is to be able to reason formally on the modeled systems, and automate as many tasks as possible in its design, implementation and validation. Such a task could be the association of the asynchronous firings to ticks of the global clock, and the generation of a synchronous program for automatic code generation.

Recently published research [6] follows a similar approach to ours. By mixing elements from two existing formalisms, one allowing the specification of time-triggered tasks and the other the specification of data flow actors, the expressiveness of the resulting modeling framework is comparable to that of Polygraph. The main difference is that Polygraph is a single formalism with decidable properties and algorithms to check them in practice. In [6], the impact of the combination of constraints from two different formalisms on their respective properties is not discussed, as the proposed approach is more focused on the performance evaluation. The experimental results the authors obtained are in favor of the modeling approach we have in common.

6 Conclusion

We have introduced Polygraph, a data flow formalism extending SDF with synchronous firing semantics for the actors. We have shown that with this extension, the existing conditions to decide of a given SDF graph's consistency and liveness were no longer sufficient. We have extended the corresponding theorems and shown that the expressiveness extensions we proposed do not impact the decidability of these properties. Finally, as a first step towards tool assisted modeling of polygraphs, we have introduced a framework relying on DIVERSITY to verify their liveness.

Our next step is to further extend Polygraph to add flexibility in the execution semantic, with the same objective to preserve the capability to perform accurate static analysis of a system's performance. Still, with this first extension, there are already interesting research perspectives regarding the applicability of existing static performance analysis techniques, and their potential extensions to take into account the specifics of a polygraph's scheduling.

Acknowledgement. Part of this work has been realized in the FACE project, involving CEA List and Renault. The Polygraph formalism has been used as a theoretical foundation for the software methodology in the project.

References

1. Arnaud, M., Bannour, B., Lapitre, A.: An illustrative use case of the DIVERSITY platform based on UML interaction scenarios. Electr. Notes Theor. Comput. Sci. (2016)
2. Bannour, B., Escobedo, J., Gaston, C., Le Gall, P.: Off-line test case generation for timed symbolic model-based conformance testing. In: Testing Software and Systems (ICTSS). Springer (2012)
3. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.A.: Cyclo-static data flow. In: Proc. of the 1995 International Conference on Acoustics, Speech, and Signal Processing. vol. 5, pp. 3255–3258 (1995)
4. Bodin, B., Munier-Kordon, A., de Dinechin, B.D.: K-periodic schedules for evaluating the maximum throughput of a synchronous dataflow graph. In: Proc. of the 2012 International Conference on Embedded Computer Systems (SAMOS). pp. 152–159 (2012)
5. Bouakaz, A., Fradet, P., Girault, A.: Symbolic Buffer Sizing for Throughput-Optimal Scheduling of Dataflow Graphs. In: Proc. of the 22nd IEEE Real-Time Embedded Technology & Applications Symposium (RTAS 2016) (2016)
6. Breaban, G., Stuijk, S., Goossens, K.: Efficient synchronization methods for LET-based applications on a multi-processor system on chip. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2017. pp. 1721–1726 (2017)
7. Cohen, A., Duranton, M., Eisenbeis, C., Pagetti, C., Plateau, F., Pouzet, M.: N-synchronous Kahn networks: A relaxed model of synchrony for real-time systems. SIGPLAN Not. 41(1), 180–193 (2006)
8. Forget, J., Boniol, F., Lesens, D., Pagetti, C.: A multi-periodic synchronous data-flow language. In: 2008 11th IEEE High Assurance Systems Engineering Symposium. pp. 251–260 (2008)
9. Gaston, C., Le Gall, P., Rapin, N., Touil, A.: Symbolic execution techniques for test purpose definition. In: Testing of Communicating Systems (TestCom). Springer (2006)
10. Geilen, M., Basten, T., Stuijk, S.: Minimising buffer requirements of synchronous dataflow graphs with model checking. In: Proc. of the 42nd Design Automation Conference. pp. 819–824. IEEE (2005)
11. Ghamarian, A.H., Geilen, M.C.W., Stuijk, S., Basten, T., Theelen, B.D., Mousavi, M.R., Moonen, A.J.M., Bekooij, M.J.G.: Throughput analysis of synchronous data flow graphs. In: Proc. of the Sixth International Conference on Application of Concurrency to System Design (ACSD 2006). pp. 25–36 (2006)
12. Ghamarian, A.H., Stuijk, S., Basten, T., Geilen, M.C.W., Theelen, B.D.: Latency minimization for synchronous data flow graphs. In: Proc. of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007). pp. 189–196 (2007)
13. Kahn, G., MacQueen, D., Laboria, I.: Coroutines and Networks of Parallel Processes. IRIA Research Report, IRIA laboria (1976)
14. Lee, E.A., Messerschmitt, D.G.: Static scheduling of synchronous data flow programs for digital signal processing. IEEE Transactions on Computers C-36(1), 24–35 (1987)
15. Medimegh, S., Pierron, J.Y., Gallois, J., Boulanger, F.: A new approach of qualitative simulation for the validation of hybrid systems. In: Proc. of the workshop on Model Driven Engineering Languages and Systems (MODELS). ACM (2016)

16. Selva, M.: Performance monitoring of throughput constrained dataflow programs executed on shared-memory multi-core architectures. Theses, INSA de Lyon (2015)
17. The List Institute, CEA Tech: The DIVERSITY tool, `http://projects.eclipse.org/proposals/eclipse-formal-modeling-project/`
18. Zhao, Y., Liu, J., Lee, E.A.: A programming model for time-synchronized distributed real-time systems. In: Proc. of the 13th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS 2007). pp. 259–268. IEEE (2007)